Universidade do Minho
Escola de Engenharia

André Filipe Pereira Alves

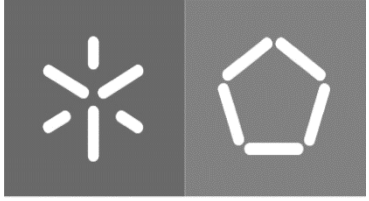
Integrating an Intrusion Detection System with Heterogeneous IoT Endpoint Devices

julho de 2020

Integrating an Intrusion Detection System with Heterogeneous IoT Endpoint Devices

André Filipe Pereira Alves

UMinho | 2020



Universidade do Minho
Escola de Engenharia

André Filipe Pereira Alves

Integrating an Intrusion Detection System with heterogeneous IoT endpoint devices

Dissertação de Mestrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Tiago Gomes

Guimarães, Julho 2020

Acknowledgments

Aos meus pais e à minha irmã, deixo o meu profundo agradecimento pelo apoio que me deram durante esta dissertação, mas também durante todo o meu percurso académico. Sem dúvida, nunca me faltou absolutamente nada para que eu pudesse tirar o maior proveito possível desta minha passagem pela Universidade do Minho.

Ao meu orientador, o Professor Doutor Tiago Gomes, quero agradecer por ter assumido para comigo um papel altamente desafiante e exigente, mas sempre presente e amigo. O seu conhecimento, experiência e liderança marcaram bem a imagem do que deve ser um orientador exímio. Deixo também um forte agradecimento ao futuro PhD Miguel Silva por ter também mostrado sempre um papel ativo e enriquecedor na minha dissertação.

A todos os meus amigos, um enorme abraço e agradecimento por todo o companheirismo e amizade demonstrados ao longo do meu percurso académico. Uma especial menção, não poderia faltar, ao grupo "ESRG Top Students".

A todos os meus professores cabe um sincero agradecimento por tudo o que me ensinaram dentro e fora da sala de aula.

À minha restante família do NEEEICUM e da AAUMinho, um honesto obrigado por terem também marcado o meu crescimento e aprendizagem durante estes 5 anos.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

Com o permanente desenvolvimento tecnológico, o mundo tem vindo a assistir a um crescimento exponencial no número de dispositivos eletrónicos presentes no quotidiano das pessoas, desde ferramentas de trabalho a dispositivos de uso pessoal. Devido a este contínuo desenvolvimento e utilização de aparelhos eletrónicos, a Internet das Coisas está cada vez mais presente nas casas, empresas, e ruas das cidades, com dispositivos munidos de desde sensores a atuadores, para demais propósitos.

O crescimento desta indústria levou a que os fabricantes priorizassem a produção de dispositivos com menores dimensões, energeticamente mais eficientes, com maior processamento, a custo reduzido. Contudo, este crescimento e a necessidade de ligarmos tudo em rede, expõe estes dispositivos a ameaças vindas da Internet. Existe assim uma óbvia urgência no desenvolvimento de soluções adequadas para proteger empresas, consumidores e infraestruturas críticas, bem como garantir maior confiabilidade nesses dispositivos e na sua utilização.

O objetivo desta dissertação consiste na exploração de soluções de segurança ao nível da rede, para dispositivos low-end ligados à Internet. Deste trabalho resultou o desenvolvimento de um sistema de detecção de intrusões, o IDIoT, capaz de detectar, corrigir, e eliminar, ataques provenientes de dispositivos maliciosos, tais como ataques de encanhamento e ataques de *Denial of Service (DoS)*. Dada a conhecida escassez de recursos nestes dispositivos, o mecanismo proposto deve apresentar requisitos de memória o menor possíveis, bem como aumento no consumo energético que não comprometa o desempenho do sistema.

Nesta dissertação é apresentado um estudo teórico para o estado da arte referente a arquitecturas de sistemas de deteção de intrusões existentes referentes a estes dispositivos. É apresentado o desenvolvimento e implementação do Intrusion Detection on the Internet of Things (IDIoT), assim como a sua integração numa plataforma com as capacidades de simulação desses dispositivos. Seguidamente, são apresentados resultados experimentais com o objetivo de comprovar o funcionamento e eficácia da solução proposta. Por último, são apresentadas as conclusões e perspectivas futuras de trabalho.
Palavras-Chave: Internet of Things (IoT), Intrusion Detection System (IDS), Ataques DoS, Ataques de Mapeamento, Operating System (OS), Segurança, Conectividade.

Abstract

With the technological development, the world is witnessing a phenomenon that can be described as a flooding of gadgets and electronic devices in everyone's daily life, both for work and personal usage. Due to this continuous increasing of development and deployment of gadgets, the IoT is continuously increasing its presence in city houses, companies and streets, with various devices that can work as sensors and/or actuators for many purposes.

The quick growth of this industry is leading to the manufacturers the prioritizing production of devices with smaller dimensions, higher efficiency concerning energy consumption, greater processing capabilities and, ultimately, at a reduced cost. Meanwhile, it has been observed that these devices, and the networks in which they are integrated, still remain very vulnerable and require stronger protection mechanisms. Therefore, there is an obvious urgency in the development of appropriate solutions to protect businesses, consumers, and critical infrastructure as well as ensuring greater reliability on these devices.

Hereupon, this dissertation consists in the development of the IDIoT, an IDS for the IoT, designed for heterogeneous endpoint devices. This system improves endpoint devices security with network layer control by being able to detect and correct to the most popular attacks targeting low-end devices, such as DoS and routing attacks. Since it targets low-end devices, this mechanism must require the smallest memory footprint possible, without sacrificing the energy consumption and the overall system performance.

Throughout this dissertation, several IDS systems and possible attacks are studied in order to endow IDIoT with the best possible features. In order to test the solution over a network that can represent real world scenarios, the IDIoT was tested over the Cooja simuator, which is a network simulator that is able to emulate several IoT motes under a chosen network topology. Furthermore, some experimental results are presented in order to prove the efficiency of the proposed solution, through the simulation of attacks against an emulated network where devices are running the IDIoT. At last, the conclusion and future work perspectives are presented.

Keywords: IoT, IDS, DoS Attacks, Routing Attacks, OSes, Security, Connectivity.

Table of Contents

List of Figures

List of Tables

Listings and Algorithms

Acronyms List

6BR 6LoWPAN Border Router.

6LoWPAN IPv6 over Low Power Wireless Personal Area Networks.

6TISCH IPv6 over the Time Slotted Channel Hopping (TSCH).

CoAP Constrained Application Protocol.

CPU Central Processing Unit.

CSMA Carrier Sense Multiple Access.

CUTE mote CUstomizable and Trustable End-device for the internet of things.

DAG Directed Acyclic Graph.

DAO DODAG Advertisement Object.

DAO-ACK DODAG Advertisement Object Acknowledgment.

DDoS Distributed Denial of Service.

DIO DODAG Information Object.

DIS DODAG Information Solicitation.

DODAG Destination-Oriented Directed Acyclic Graph.

DoS Denial of Service.

FAM Frequency Agility Manager.

FPGA Field Programmable Gate Array.

HIDS Host-based Intrusion Detection System.

HTTP Hypertext Transfer Protocol.

ICMP Internet Control Message Protocol.

IDIoT Intrusion Detection on the Internet of Things.

IDS Intrusion Detection System.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IoT Internet of Things.

IP Internet Protocol.

IPv4 Internet Protocol version 4.

IPv6 Internet Protocol version 6.

LLN Low Power and Lossy Network.

LoRa Low Range.

LoWPAN Low Power Wireless Personal Area Networks.

LPM Low Power Mode.

MAC Medium Access Control.

MCU Microcontroller Unit.

MQTT Message Queue Telemetry Transport.

MTU Maximum Transmission Unit.

NFC Near Field Communication.

NIDS Network-based Intrusion Detection System.

OS Operating System.

OSI Open System Interconnection.

PAN Personal Area Network.

RAM Random Access Memory.

RCU Reconfigurable Computing Unit.

RF Radio Frequency.

RFID Radio Frequency Identification.

ROM Read Only Memory.

RPL Routing Protocol for low power and lossy networks.

RTOS Real Time Operating System.

SIEM Security Incident and Event Management System.

SoC System on Chip.

TCP Transmission Control Protocol.

TI Texas Instruments.

TSCH Time Slotted Channel Hopping.

UDP User Data Protocol.

WSN Wireless Sensor Networks.

Chapter 1

Introduction

This chapter starts by addressing the problem statement of this thesis as well as the expected goals intended to achieve. Finally, a brief description of the structure of this document is presented.

1.1 Problem Statement

Connecting myriads of end-devices in the IoT can bring several challenges to the way such systems are designed and deployed in a wide-range of applications [1][2][3]. Such challenges not only comprise the connectivity and interoperability of heterogeneous wireless nodes, where a large volume of data is exchanged with the Internet, but also arise security- and privacy-related issues, even at the network edge. Such issues demand for a robust solution to tackle the ever-growing amounts of data transferred over the network, and also the security and performance requirements. Nowadays, hybrid hardware platforms, which combine a Microcontroller Unit (MCU) and a Field Programmable Gate Array (FPGA) on the same System on Chip (SoC), are becoming more cost-effective solutions that can be used even at the IoT network edge. Such platforms add extra processing capabilities to already existing systems by allowing the deployment of dedicated hardware accelerators [4][5][6] on the FPGA fabric. For instance, CUstomizable and Trustable End-device for the internet of things (CUTE mote), a mote specially designed for the edge network, is an in-house project that uses such platforms [7][8][9].

Nowadays the IoT industry is increasingly reaching more ecosystems, from infrastructures, such as transportation systems and power plants, services that reach out to entire cities and countries, and even household appliances such as electronic gadgets, smart home devices from light bulbs and wall plugs, to personal use machines. This vast presence of the IoT concept in the daily activities of our society is definitively a real concern in cybersecurity matters. Bringing connectivity to low-end devices significantly increases breaches for attacks through the network in which these devices are connected. With such

breaches and security issues, devices and networks are significantly more compromised ,which makes them less reliable. Therefore, efforts need to be made in regards to seeking for more cost-effective solutions and more processing capable devices , and in the security- and privacy-related issues [3][10][11]. Such issues demand greater and stronger mechanisms that enables more and more security at the IoT device and network paradigm. Mainly due to the quick growth of this industry, the rush for cheaper and faster devices has been the prior reason for the security and privacy concerns that have become a smaller priority for the manufacturers. However, adding additional capabilities to such resource-constrained devices is a compromise between integrating security mechanisms and reducing the device's processing capabilities, increasing energy consumption and other concerns [12]. This pros and cons balancing situation is definitively a challenge researchers have in hands because the impact of such solutions. One solution for tackling this problem is fetching an IDS mechanism for the IoT networks and protocols, allowing for a greater possibility of detection and prevention of many kinds of malicious intrusions in IoT devices and networks. For instance, an implementation of an IDS towards the IoT constrained devices, can become a very useful tool for the manufacturing industry.

For this reason, this thesis focuses on the study of IDS development capable of improving devices security againts two popular attacks: routing and DoS, with a possibility of integration in aforementioned hybrid hardware platforms such as the CUTE mote [7]. Due to the ferocity of such attacks and the liability of these resource constraint devices, the IDS must be able to standalone contribute with major safety capabilities.

1.2 Goals

The main goal of this thesis is to integrate an IDS with heterogeneous IoT endpoint devices. This security mechanism must be able to:

- Detect and prevent against routing attacks;
- Detect and prevent against DoS attacks;
- Improve safety in the data transferred through the network;
- Induce little overhead regarding memory and Central Processing Unit (CPU) usage;
- Be OS agnostic;

The final goal is to integrate this solution with a heterogeneous architecture for an IoT endpoint device, such as CUTE mote [7].

1.3 Thesis Structure

This thesis will proceed to the Chapter 2, State of the Art, where structural topics used in this dissertation are covered, performing a theoretical context necessary for the discussion and conclusions taken down the line. Furthermore, the related work is presented, in which some state-of-the-art IDS solutions are studied. Information about the platform and tools used during and for the project are also shown. In Chapter 3, System Model and Design is covered, by showing the developed architecture of the solution, the decisions and trade-offs taken considering the best design. Moving on, is the Chapter 4 which covers the Implementation of the IDIoT solution. Some core-algorithms are presented aiming to deeply explain how the previous system model is implemented and integrated with the entire system. Evaluation takes place in Chapter 5 and here are presented the results for simulations and benchmarks performed on the system. Then, an analysis occurs, concerning the proposed solution and the obtained results. Limitations of the solution are also covered. At last, the conclusions for this project are presented in Chapter 6, referring to the solution beneficial features as well as limitations. A final section is reserved for future work discussion.

Chapter 2

State of the Art

Throughout the development of this work, several topics and domains must be covered. These include security and connectivity, applied to IoT end-devices and IDSs. These principles are the backbone of this study. The first section of this chapter will briefly describe what these topics are and where they stand in the current state-of-the-art. Firstly, IoT history, importance and architecture will be approached, in Sections 2.1.1 and 2.1.2. Moreover, OSes for the IoT are discussed in Section 2.1.5 as these are fundamental for our work, which will present a solution that runs as one or more OS applications. Furthermore, security threats addressed to the Wireless Sensor Networks (WSN) and the IoT are discussed in Section 2.1.6, an important topic for a security concerned project such as this one. At last, IDS as a defense mechanism is approached in 2.1.7, regarding possible architectures and common actuation methods. Next, the second section of this chapter addresses current state-of-the-art that is somehow closed to this work. Finally, last section covers the platform and tools required to the successful realization of this work.

2.1 Background

2.1.1 The Internet of Things

The Internet of Things (IoT) as an idea has been around for a while now, the first example of an application more related to the IoT concept appears in the early 1980s where local programmers from the Carnegie Melon University used the internet connection to access a Coca Cola machine in the university facilities. They would use the connection to the refrigerated appliance to check if there was a drink available, and if it was cold, before making the trip to the cafeteria room. Despite some similar applications appearing as such, the IoT was not even officially named until 1999 [13]. This imminent technology is growing at a fast rate and it is already adding up large importance on various environments, on a daily basis. Whether we

are improving the production of an industry, monitoring our health, or providing services for the community of a city such as live parking information, it is the common IoT platform that allows the communication between all its intervenients. The IoT, for this reasons, is resulting in regular day-to-day tasks to become more easy and automated [14] [15]. IoT can be described as a collection of billions of devices sharing and collecting data through the internet [16]. These devices are now able to communicate with each other, and with the user, in a network made of things, due to the emergence and integration of the IoT [17]. IoT devices can be used in a multitude of applications, in an endless amount of configurations and network topologies [1] [18] [2][3]. Each IoT device usually performs a specific set of tasks anf procedures such as data collection, like sensors, or performing services, like actuators. Communication with IoT devices through the Internet is only possible by integrating gateways to the network [19]. Some benefits of using IoT technology are: access to high-quality data; better tracking and management; more efficient resource utilization; automation and control; comfort and convenience; time and money savings [20]. Some fields of application are the human body, our homes, the environment, cities, and industries [16] [18] [21]. Mimo [22] is a new kind of infant monitor that provides parents with real-time information about their baby's breathing, skin temperature, body position, and activity level on their smartphones. To help to prevent problems like sudden death syndrome, these devices improve health and life quality. SmartTrash [23] is a community service directed appliance. Products like cellular communication enabled Smart Trash to use real-time data collection and alerts to let municipal services know when a bin needs to be emptied. This information can drastically reduce the number of pick-ups required and translates into fuel and financial savings for community service departments. OnFarm [24] is a solution that combines real-time sensor data from soil moisture levels, weather forecasts, and pesticide usage from farming sites into a consolidated web dashboard. Farmers can use this data with advanced imaging and mapping information to spot crop issues and remotely monitor all of the farm's assets and resource usage levels. Many more appliances can be developed and integrated using the IoT, for instance, [16] gathers some big amount of examples regarding this, concerning five major scenarios. These are: Transportation and Logistics, with examples of environment monitoring application; Healthcare, with examples of tracking application; Smart Environments, with examples of comfortable homes and offices; Personal and social, with examples of thefts and losses; Futuristic scenarios, at last, with examples of robot taxi's.

Statista, a business data platform, estimates that by the year of 2020, around 31 billion smartphones, wearables, smart watches, cars, and other devices will be connected to the IoT [25]. As previously discussed, there is a vast versatility of IoT appliances all around the tecnologic world. In Figure 2.1, adapted

from [26], is categorized a structure for the IoT architecture.

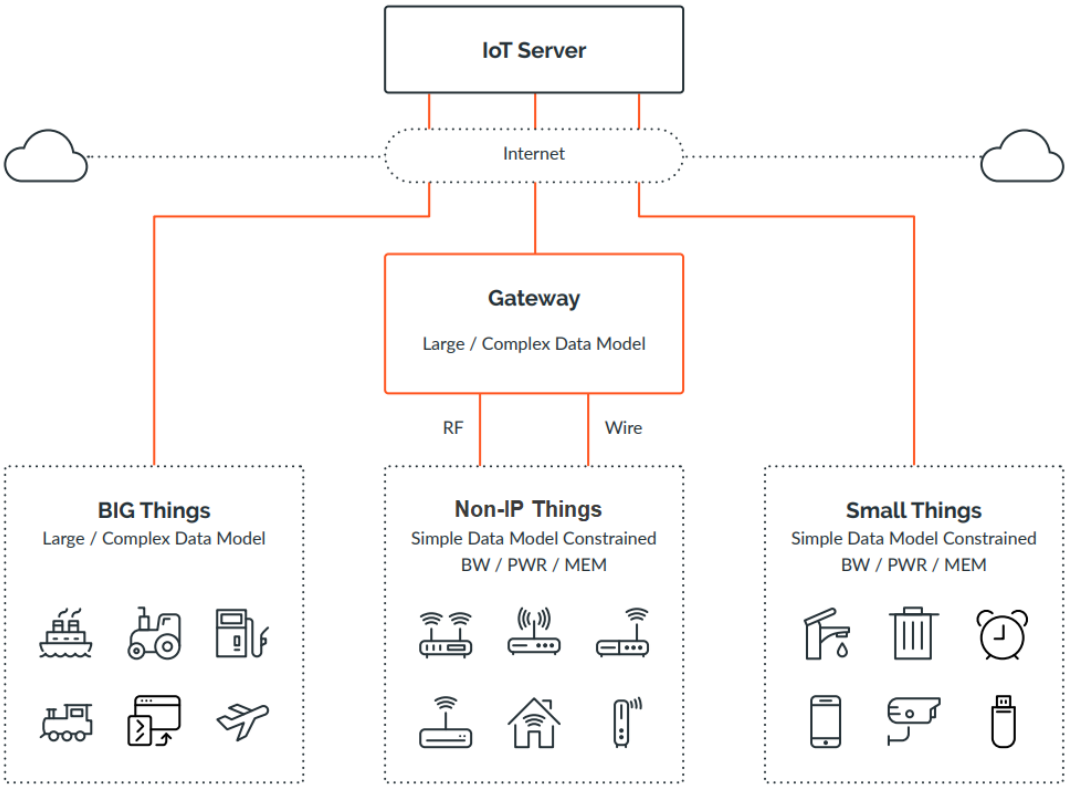


Figure 2.1: Variants of things and networks in the Internet of Things.

Similarly, to the Open System Interconnection (OSI) model, the most accepted architecture layout for the IoT in the current literature [27][18] is structured, from top to bottom, with application, network and perception layers. In Figure 2.1 it is possible to understand this categorization and organization. On the top of the architecture stands the application Layer that manages the data exchanged with the layers below, and uses this data to provide required services or operations. It is also known as the business layer and here various applications can exist, each having different requirements. Moving downwards, in the network Layer the processed information from the perception layer is received here. Then, the routes to transmit the data and information to the IoT hub are determined. This is the most important and dense layer of the architecture because the numerous amounts of devices and communication technologies that are integrated. The last one is the perception layer, or in other words the physical or the sensor layer and its the bottom layer in this example of IoT architecture. This layer has the capabilities of measuring

and collecting data which is then processed to the upper layers. Sometimes these network nodes can exchange data with each other and even perform operations without intervention from above.

Concerning the perception layer, there can be many kinds of devices/things in IoT network. For instance, the IoT compliant devices can be described in three types of things, which are presented in Figure 2.1. All of these communicate with the IoT server via an Internet connection. *Big Things* are considered large devices, not specifically concerning the size of the device, but also as the complexity of the function it can perform and the vast expected resource availability. These devices are normally computers, big industrial machines, cars, etc... *Small Things* are devices with simpler function and complexity. Normally executing censoring or actuating operations, these devices are expected to be constrained in resources such as memory, power, and bandwidth. *Non IP-Things* are very much similar to the small things, described above. They establish a connection with a Gateway, using communication protocols such as Low Range (LoRa), bluetooth and Radio Frequency (RF). The Gateway performs the analysis of data which is later transmitted to the IoT Server via an Internet connection.

2.1.2 The IoT Network Stack

The network layer of the IoT architecture is a layer concerning connection and connectivity. Protocols are specifically designed for the IoT and must tackle all of the communication challenges involved in this kind of network, which can be supporting and connecting to the Internet a large amount of heterogeneous smart devices [28] [18]. Such challenges are: addressing and identification, low power communication, low power routing protocols, high speed and nonlossy communication and, also, mobility of smart things [29]. In this section, we will briefly approach the IoT network stack and it's most important protocols which will later be used during the development of the IDIoT solution. To develop a security mechanism for the IoT low-end devices, it's important to understand what protocols are used within an IoT network, not only for communication between its devices but also for communication between the gateway of the network and the Internet. Additionally, it is also important to refer to which routing protocols are designed to IoT networks, most importantly the protocol we will be using, since routing is very important for the subject of preventing attacks, taking in consideration that these can take place by taking unauthorized actions against routing information.

Generally, IoT devices connect to the Internet through the Internet Protocol (IP) stack, which most typically nowadays is used the IPv6, as an upgrade to the previous, but still in use, Internet Protocol version 4 (IPv4) [30]. However, some IoT devices are too resource-constrained to secure such complex and dense

network protocols. For this reason, these devices tend to connect locally through non-IP networks and, using a border router acting as a gateway, connecting the network to the wide Internet [31]. For instance, taking a glance at Figure 2.2 it is possible to further understand how all the above mentioned layers and protocols reside within a protocol stack such as the 6LoWPAN protocol stack.

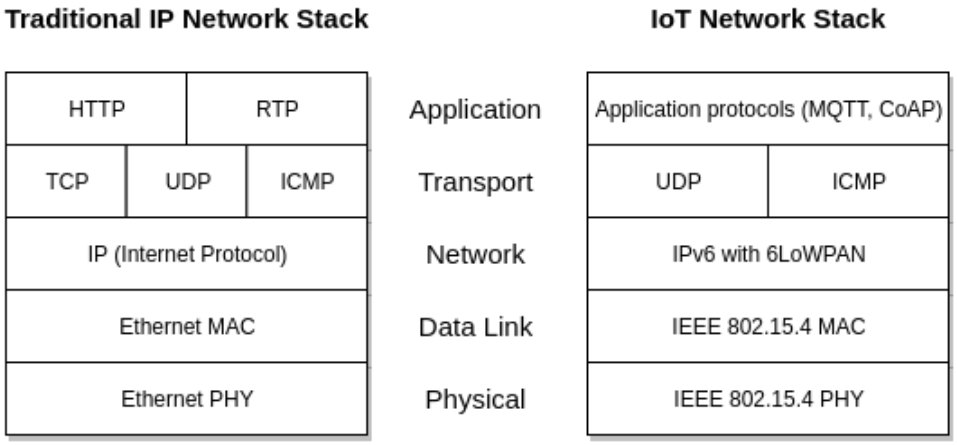


Figure 2.2: IoT Network Stack.

There are many existing protocols for these non-IP networks, such as Bluetooth, Radio Frequency Identification (RFID), Near Field Communication (NFC), Institute of Electrical and Electronics Engineers (IEEE) 802.15.4, low power wifi, Sigfox, LoRa, Zigbee and some more [32][33] [34]. These are very popular however they are limited in range, making most of the above networks limited to small-medium Personal Area Network (PAN) [17]. In order to adapt an IP stack to better suit IoT networks, 6LoWPAN was developed by the Internet Engineering Task Force (IETF) and constituted one of the most important protocols of an IoT network stack. Moving upwards to the transport layer, IoT networks can use protocols such as UDP and Transmission Control Protocol (TCP), however, due to being a connection-oriented protocol and having a considerable bigger overhead, TCP is not a good option for communication in low power environments and UDP is the preferred option. Some networks also employ Internet Control Message Protocol (ICMP) at this layer, instead of UDP. Finally, at the top of a device's communication architecture, the application layer also has many protocols specifically developed for the IoT networks and two of the most used are Message Queue Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP). These will not be taken into details in this thesis.

2.1.3 6LoWPAN

6LoWPAN consists of the Low Power Wireless Personal Area Networks (LoWPAN) adaptation layer which defined encapsulation and header compression mechanisms enabling IPv6 over IEEE 802.15.4-based networks. IEEE 802.15.4 networks. It is an open standard regulated by the IETF, which defines other standards used in the internet, e.g., Hypertext Transfer Protocol (HTTP), UDP and TCP. The IETF regulated 6LoWPAN through RFC 4919 and RFC 4944 between 2007-2009 [33][32]. This open standard is expected to be used by resource constrained embedded devices in low power wireless networks.

Internet protocol packets can be carried efficiently within small link-layer frames because 6LoWPAN has defined encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over IEEE 802.15.4-based networks [18]. This way, 6LoWPAN performs a crucial function at adapting the packet sizes of the two networks, since IPv6 requires the Maximum Transmission Unit (MTU) to be at least 1280 bytes while IEEE 802.15.4 standard's packet size available for media access control layer is just 102 to 81 bytes [35].

On a 6LoWPAN network, there is a dedicated edge router, just like a sink node in a WSN. The 6BR acts as gateway communication with the internet. On a 6LoWPAN network each node can have an IPv6 address assigned to each interface, which enables them to be directly reached from anywhere on the Internet. This way, applications running on the nodes can exchange IP packets to a server on the Internet [36].

In Figure 2.2 a comparison is made between the IoT network stack, using the 6LoWPAN, and the traditional Internet network IP stack [36].

2.1.4 RPL

Routing Protocol for low power and lossy networks (RPL) is a routing protocol developed specifically for Low Power and Lossy Network (LLN)s which is a class of networks in which both the routers and their nodes are expected to be constrained concerning the fundamental resources such memory, power sources, and processing capabilities. This is a pro-active protocol based on a distance vector algorithm and provides support for multipoint-to-point traffic as well as point-to-multipoint traffic, from inside of the LLN network to the outside, and vice-versa. Support for point-to-point traffic is also available in RPL [37].

This protocol configuration works by creating a network topology map, which is called a Directed Acyclic Graph (DAG), and every node present has an assigned rank. The Destination-Oriented Directed Acyclic Graph (DODAG) root node has the smallest rank of the network, 0, and node's rank increases as they get

further from the root since the rank of the node describes their logic distance to the root. A parent-child hierarchy is created where the nodes closer to the root are parents, and below them are the children, with higher ranks.

In the genesis of a DODAG, a node of the network is chosen as root. Most times the border router is set by the network administrator as the root. This root node will then send to all network neighbors a DODAG Information Object (DIO), spreading the message through all neighbors, by link-local multi-casting, that he is, in fact, the root. The root will then inherit rank value 0 and, consequently, nodes that receive the DIO message will discover the new DODAG and replace the old one. This way, nodes in the network can join the DODAG by making use of the DIO messages received. Upon reception of such a message, the node selects the best parent according to the rank of its neighbors in the DODAG and then broadcasts its own DIO message to other nodes.

The DODAG demands for initialization and maintenance. For this, and to exchange information about the graph over the network, four different control messages are used:

- DODAG Information Object (DIO), which was covered before, is a control message which contains information about the routing graph such as IPv6 address of the root and current ranks of all the nodes;
- DODAG Information Solicitation (DIS) is used to solicit DIO messages from router nodes in RPL. It may be used to probe neighbor nodes in adjacent DODAGs;
- DODAG Advertisement Object (DAO) allows for nodes to advertise the path to the root in order to fully build the DODAG;
- DODAG Advertisement Object Acknowledgment (DAO-ACK) is a simple acknowledgment message for a node to send when a DAO is received.

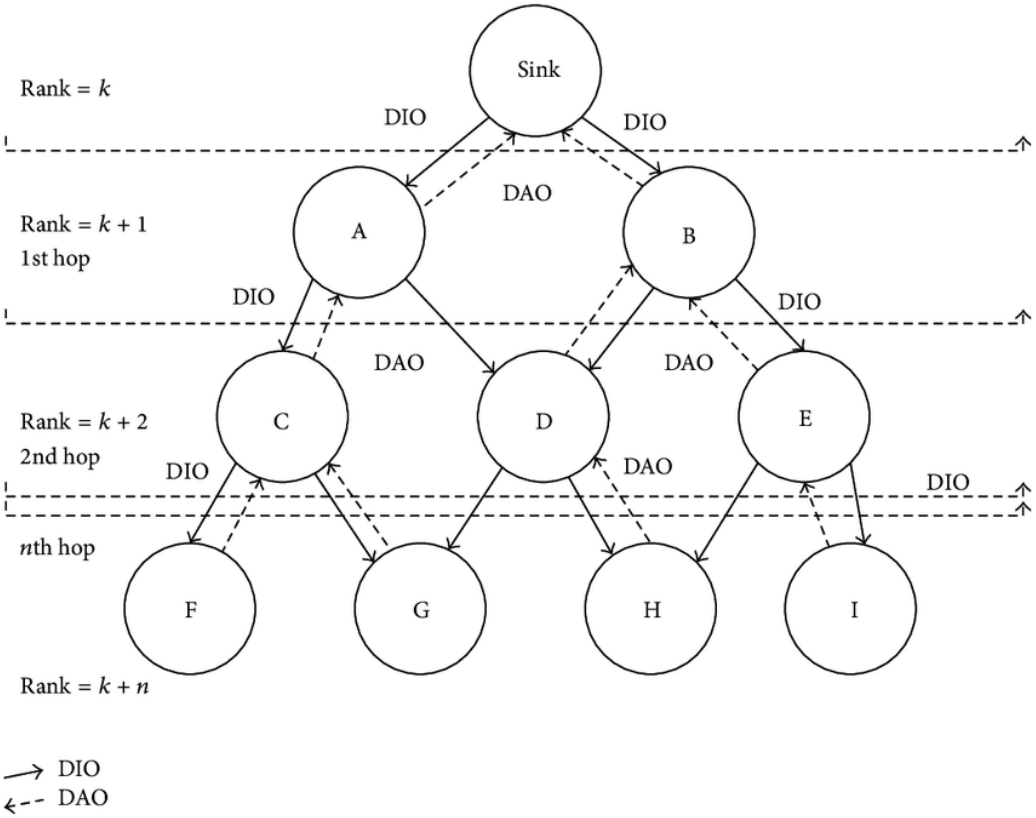


Figure 2.3: Flow of DIO and DAO messages in RPL network.

In Figure 2.3 DIS and DAO control message's regular behaviour is shown. As explained previously, the Sink has the smallest value of rank and this value increases and nodes go further from the sink. DIS messages are spread from the border router to all the nodes in the network and each node responds with DAO messages containing their and their neighbors information [38].

RPL protocol is widespread on IoT low-end devices and networks, with current implementation under several embedded IoT OSes, such as: Contiki-NG, TinyOS, LiteOS, T-Kernel, EyeOS, RIOT.

Furthermore the Contiki-NG supports two versions: RPL-Classic, the first version and presented in the early versions of Contiki, and the new implementation, RPL-Lite, introduced with the Contiki-NG. RPL-Lite is a lightweight version of the classic Contiki-RPL implementation and it removes all the support for storing mode, working always with the non-storing mode. It removes the possibility of having multiple instances and DODAGs. This way, it allows for more performance, less ROM footprint and more stability. However, this implementation looses interoperability with other current implementations.

2.1.5 Operating Systems for the IoT

An Operating System (OS) usually is a big complex system of software that manages machine hardware and software resources in order to provide the user of such a machine an environment of abstraction where generic software applications can be executed. OSes manage the hardware resources, which include: input devices such as keyboards, output devices such as displays, network devices such as routers and network connections, storage devices such as internal or external drivers. This way, applications can be developed for specific OS without having to take into consideration the specific hardware details or even other tasks, e.g., memory allocation [29].

OSes, as we know them today, appeared as early as 1960 and have since been developed side by side with the technological improvements of computers and devices in general, by the hands of companies such as General Motors, IBM, Apple, Linux, Microsoft and many more. Worldwide, OSes are nowadays the most vital tool for any device. Around March 2019 Microsoft announced that Windows 10 was now running over 800 million devices. Concerning the personal computing platform area of smartphones and watches, Google's Android is dominating with over 2.5 billion users.

As we can see, there are many successfully OSes for many kinds of machines and devices due to different personal preferences of users and enterprises worldwide, but also due to the different requirements of each system. For instance, OSes for cellphones and smart-phones were developed because the already existing OSes for bigger machines, e.g., computers and laptops, were not suitable for these environments. The same process leads to the development of OSes for even smaller and resource-constraint devices, such as the low-end devices in the IoT. These IoT low-end devices, like Arduino, TelosB motes, Zolertia Z1, Tmote Sky, etc, are too resource-constrained in terms of energy, processing capabilities, and memory capacity, to be able to run traditional OSes.

The constrained low-end devices bring novel challenges for the development of OSes for such devices [39]. The most important requirements that a generic OS for low-end IoT devices should focus on satisfying are the following:

- Small Memory Footprint - one of the most crucial requirement in a IoT low-end device making this a challenge for the OS to fit within such constraints. The IETF uses three classes of memory capacity classification for these devices, where class 0 is for devices with the smallest resources («10kB of RAM and «100kB flash) and class 1 for medium-level resources (10kB of RAM and 100kB Flash);

- Support for Heterogeneous Hardware - IoT low-end devices are now running in a huge diversity of hardware and communication technologies making thus a requirement for and OS to be able to support this heterogeneity. IoT low-end devices are based on various microcontroller architectures which can vary from 8, 16 and 32 bit. Additionally, some may have different ROM/RAM limitations;

- Network Connectivity - IoT devices are all about connectivity. Thus, these devices usually provide support to at least one communication interface, which can vary from low-power radio technologies, e.g., IEEE 802.15.4, 802.11 (WiFi), or Ethernet. This way, an OS for the IoT must have a network stack based on IP protocols relevant for the IoT;

- Energy Efficiency - An OS for the IoT must be able to provide energy saving options to the upper application layers, as well as using those energy-saving options itself as much as possible, since battery powered devices must provide a long-term duration;

- Real-Time Capabilities - are also very important for these devices since most of the applications they will be running demand an accurate timing execution, e.g., medical purposes or precise-timed sensor readings for many purposes. This way, an Real Time Operating System (RTOS) can have an important role in specific applications of the IoT and thus any OS should be able to supply with features in accordance with the application requirements;

- Security - Even tough most IoT devices will have security and safety measures provided by the network control or even the industrial systems in which they make part of, an OS must be able to provide Root-of-Trust, provided by hardware, for these Internet-connected devices.

OSes for such low-end devices started appearing as early as 2002. Back then, such OSes were developed only for the WSN and only later started being integrated with the IoT low-end devices requirements and specifications. Table 2.1 summarizes the most prominent OSes for IoT, according to their category and their most important features [29].

Table 2.1: Key features of representatives of several categories of OSes.

Name	Category	MCU w/o MMU	< 32 kB RAM	6LoWPAN	RTOS scheduler	HAL	Energy-efficient MAC layers
Contiki	Event-driven	✓	✓	✓	✗	✓	✓
RIOT	Multithreading	✓	✓	✓	✓	✓	✗
FreeRTOS	RTOS	✓	✓	✗	✓	✗	✗
uClinux	Multithreading	✓	✗	✓	✗	✓	✗
Android	Multithreading	✗	✗	✗	✗	✓	✗
Arduino	Other	✓	✓	✗	✗	✗	✓

Contiki, first released around 2002, is an event-driven OS with cooperative scheduling approach and with support for lightweight pseudo-threading [40]. It runs on 8, 16 and ARM 32-bit MCUs and works with its native network stack: µIP and Rime Stack. RIOT [41] runs a micro-kernel-based RTOS with multi-threading support, using a kernel inherited from FireKernel. It started being developed in 2012 and runs in 8, 16 and ARM 32-bit MCUs. RIOT can run its native stack, gnrc, and also has support for OpenWSN and ccn-lite. FreeRTOS [42] is equipped with a preemptive micro-kernel with support for multi-threading and started being developed in 2002 for general embedded systems architectures. It supports 16, 32 and 64-bit MCUs and has no native stack, however, it presents support for others. TinyOS [43] is a really lightweight OS which first appeared in 2000. It runs on extreme constraint nodes running 8 and 16 bit MCUs and has an event-driven with cooperative scheduling approach. BLIP his its native network stack. OpenWSN [44] has event-driven with cooperative scheduling approach kernel and started being developed in 2010. It runs on its own popular network stack, the OpenWSN. Amazon Free RTOS is a more recent OS for the IoT which is rapidly gaining popularity. It runs on a multi-threading programming model.

2.1.6 Security Threats Addressed to the IoT

Connectivity is nowadays a basic requirement of any devide. However, and since they can be accessed from anywhere in the world, connectivity drastically increases the attack surface [36]. Moreover, as the IoT uses network architecture similar to traditional networks, IoT networks also inherit problems and liabilities from these traditional networks [45]. Provided that this work is concerned with the security of IoT networks and devices, it is fundamental to briefly approach this problematic. The Figure 2.4, adapted from [45],

organizes the security threats for the IoT in the perspective of four different entry points, pointing out the most dangerous attack of each category.

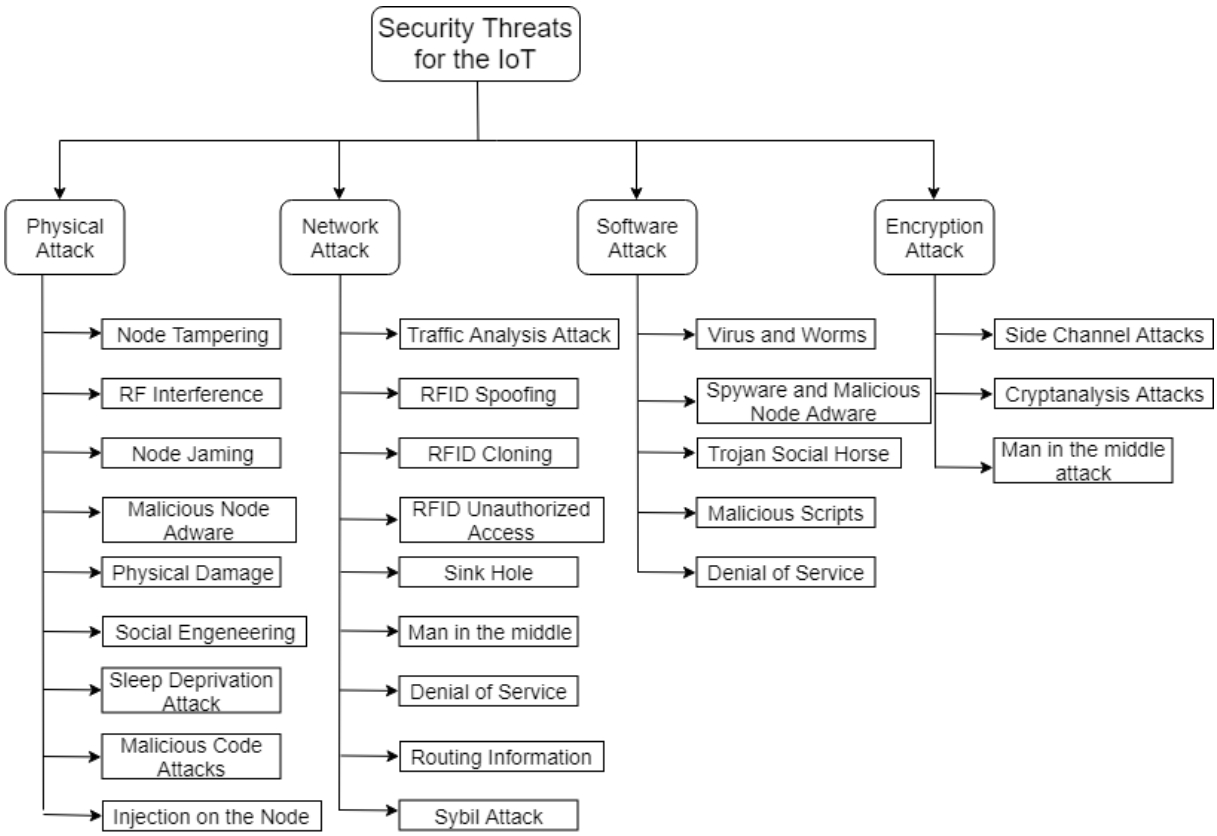


Figure 2.4: Taxonomy of Security Threats in the IoT.

Physical attacks are performed within the hardware devices in the network and these mainly target the physical layer of the OSI model. Malicious code injection, which not only stops the services but also modifies the data, is considered as the most dangerous of these. Network attacks refer to those focused on the network layer of the IoT system which presents many similarities to the network layer of the OSI model. This means that these attacks mainly affect the communication protocols. Sinkhole and DoS are considered the most dangerous attacks [36][46] [47]. The next category is concerned with software and these are performed using a worm, spyware, adware, etc., in order to steal data and deny the services. The authors [45] claim virus and worms to be the most harmful attack. This type of attack consists of a virus that searches for files and services within a device to attack. The fourth group are the encryption attacks. These depend on destroying the encryption techniques and obtain the private keys. In this category, side-channel attacks are considered the most difficult to prevent. Here, the attacker uses side-channel

information emitted by encrypting devices in order to detect the encryption key. It is performed by using information about the encryption operation, power and time required, faults frequency, etc.

Many surveys have been published towards security threats in the IoT [48][46][27]. Such surveys mostly study threats which target the devices through the network and application layer. In accordance with the previously presented taxonomy in 2.4, these are: Routing, Man-in-the-middle, and Denial of Service attacks [46]. Routing attacks affect the routing information by spoofing, modifying or replacing this information. These attacks aim to create routing loops, to attract or repel network traffic, to increase or decrease source routes, etc... Some specific routing attacks are Sinkhole, Selective-forwarding, Wormhole and Sybil attacks. Man-in-the-middle attacks work by interfering with the communication between two entities. The attacker node modifies or obstructs the communication between entity A or B, without these noticing, or even just captures the traffic for unauthorized data analysis. Denial of Service are attacks launched into a specific network or a specific device within a network, intending to disable the normal operation of this device, or network, by an exhausting their resources. These attacks usually take place by flooding of communications or even jamming of the communication channels. There is a variety of DoS and Distributed Denial of Service (DDoS) attacks, such as IPv6 UDP Flooding Attack, Syn Flood, Land attack, ICMP flood, Smurf attack.

Routing attacks and DoS attacks are considered the most frequent and more destructive mainly due to the current IoT device's security breaches and constraint of resources [46][49][36]. Routing-wise, Sinkhole attack is the most important and takes place when an attacker introduces a fake node inside a network. This node will advertise, to all neighbor nodes, a minimum cost routing path in order to make all the adjacent nodes forwarding their packets through this malicious node. On the other hand, concerning the DoS attacks, one of the most destructive IPv6 UDP flood, a type of DoS attack in which the attackers overwhelms random ports on the target with IP packets containing UDP datagrams. As this happens, the receiving host will try to find the corresponding application associated with the received datagrams, but none will be found. Then, the host will try to send back a "Destination Unreachable" packet but having sent many and many UDP packets, the attacker will be able to overwhelm the system and make it unresponsive to other clients.

2.1.7 Intrusion Detection System

An IDS is a type of security software designed to detect malicious actions performed by intruders with the aim of obtaining unauthorized access to a computer or a network. In order to protect systems

and networks, an IDS performs analysis on the network traffic, using an analysis engine, so it can detect ongoing intrusions and report a system alert to the network administrator. The network administrator, or other tools assisted by the IDS, can then take action against this attack by blocking such harmfull user or device, stopping the flow of that attacked network, and even saving the signature of the attack for later recognition and earlier prevention [27]. Intrusions can be external or internal to the network. An external intrusion happens when a user outside the target network tries to access one system without authorization. An internal intrusion happens when an inside device attempts to raise its access privileges in order to perform non-authorized actions [50].

The first research about this subject first appeared by 1980 by [51] where a preliminary concept of an IDS was delineated at the National Security Agency and consisted of a set of tools intended to help administrators review audit trails, normally composed by access logs, file access logs and system access logs [52]. Various kinds of networks co-exist in the Internet. For instance, some networks in the Internet enable peer-to-peer communication, where devices may communicate directly with each other and, for this reason, security mechanism based in the networks switch or router may not be preventive in case of a device becomes malicous. Therefore, in order to tackle these adverse difficulties, two kinds of IDS configuration also exist. Network-based Intrusion Detection System (NIDS) connects to one or more network segments and monitors network traffic for malicious activities. Host-based Intrusion Detection System (HIDS) is attached to a computer device and monitors malicious activities occurring within the system. The HIDS can also perform analysis to system calls, running processes, file-system changes, interprocess communication and application logs [48]. Figure 2.5 depicts the topology of a NIDS and a HIDS.

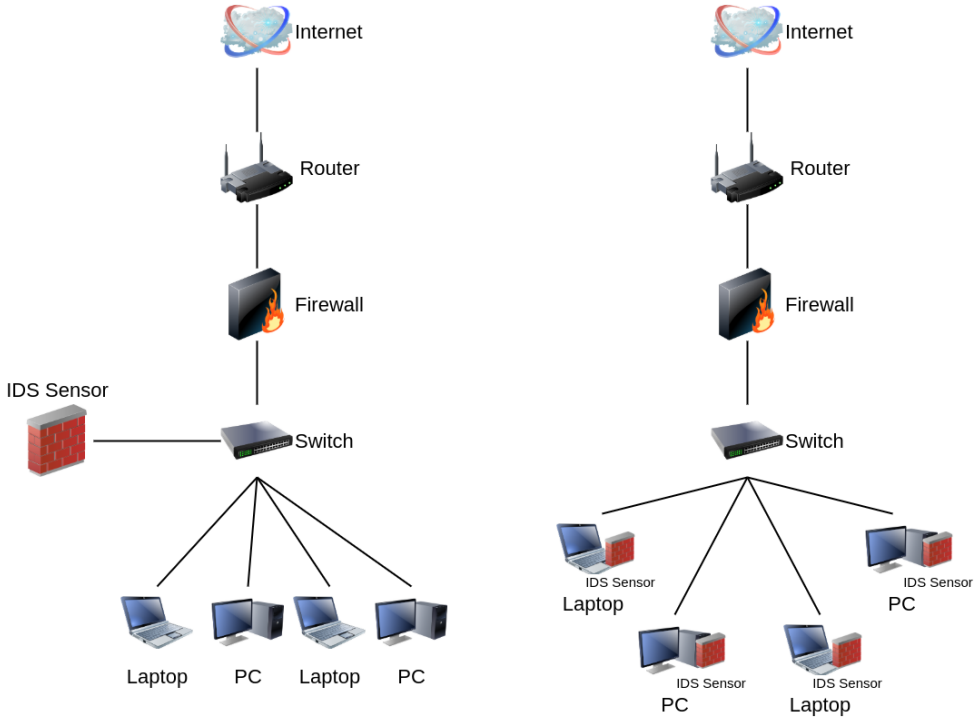


Figure 2.5: Network-based IDS (left) vs Host-based IDS (right).

Even though IDSs are considered a matured technology for traditional networks, IDSs solutions are not so adequate for the IoT systems. Due to the lack of security concerns and resource constraints in the devices currently used in IoT networks, there are many development barriers for traditional IDS to operate within the IoT networks. These IoT devices are generally resource constrained due to reduced power, memory, processing capabilities and more. This way, the IDS as a concept of security mechanisms is also being imported for the WSNs as these devices become more connected with the Internet due to the IoT. Many solutions are already spread within the academic community and will later be studied in depth, in Section 2.2.2. For this purpose, some authors have published surveys [48] [27] to the current literature of IDS for the IoT and have categorized this solutions in a taxonomy that juggles with four approaches, being these: placement strategy, detection method, security threats which the IDS was designed to protect, and validation strategy.

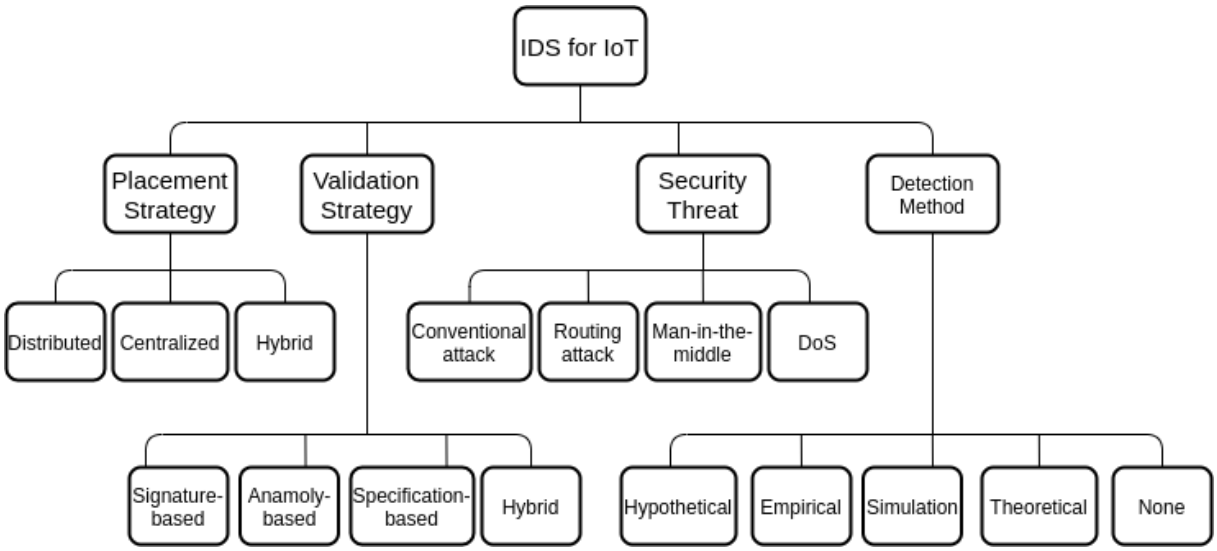


Figure 2.6: IDS for the IoT Taxonomy.

The IDS placement strategy concerns the strategy and location of IDS within the systems architecture. This strategy can be classified into three classes as seen in Figure 2.6. In a distributed strategy, IDS modules are placed in every physical object of the network. In this case, the drawbacks are that each physical object of the IoT network will be resource-constrained so the IDS module must be greatly optimized. Centralized strategy happens when the IDS is strategically placed in a centralized component of the network, e.g., a border router or a dedicated host, so all the traffic exchanged between the network and the Internet can be analyzed by the IDS. In a Hybrid placement, both strategies are combined in order to take each others strong points and to avoid each others drawbacks.

The IDS detection method describes the functional mechanism used for the detection engine. In signature-based approach the behavior of usual and known attacks are stored in the IDS internal database so it can match certain network or node behavior to a stored attack signature. Signature-based are accurate and very effective when it comes to known threats, however, they are ineffective to detect new and unknown attacks because signature matching will never occur. Anomaly-based engine is composed by a set of rules defining how the network should behave. These rules define a normal behavior threshold and therefore if a deviation from these values occur, an alert is triggered. This approach is very efficient to detect attacks related to abuse of resources, however, they can consider an intrusion to anything that does not match a normal behavior and this can become a drawback. Specification-based is very alike with the anomaly-based where a set of rules and thresholds are defined to represent the expected behavior but in this approach, a human expert should manually define the rules of each specification. This allows for rules to

be set for network specifications such as nodes, protocols and routing tables. Hybrid approach uses all the above-described strategies in order to maximize their advantages and minimize their drawbacks.

An IDS is also categorized by the security threats it tries to prevent. There are many and the most relevant are, sinkhole and DoS [45] [36]. At last, the validation method is also an important metric for a classification on an existing IDS solution. Some validation examples are hypothetical, by simulation, theoretical, or even no validation at all. These last are self-explanatory.

2.2 Related Work

Securing devices and networks with security measures like an IDS is an important topic that has been addressed for a long time. This method was so reliable and there was already established a mechanism for all networks, anti-virus, etc., that with the technological development and appearance of new technologies such as the IoT, the adoption of an IDS of IDS was kept within the goals of a secured network and/or device. However, developments in these technologies resulted in new security challenges to whom traditional IDSs were not suitable. This way, new solutions are demanded either from adapting the existing solutions to the new paradigm of the IoT, either from developing new solutions from the scratch. In this section we will present and discuss the most relevant related work concerning IDS based solutions. High- and Middle-end IDSs are discussed in 2.2.1. Low-end IDSs for the IoT are discussed in 2.2.2. At last, a brief analysis of the platform and tools is presented in Section 2.3.

2.2.1 High- and Middle-end IDS

IDS are usually one of the best tools for securing networks and devices. These security mechanisms were usually appointed as one of the best tools for securing traditional networks and devices. High- and middle-end IDS solutions are currently at its peak of development and well established in the Internet networks. For this class of devices, Snort and Suricata, depicted in 2.7, are two well-know solutions that have been widely used [53] [54].

2.2.1.1 Snort

Snort is a Network-based IDS developed in 1998 and spread for the web community like an open-source tool in 1999. It is a High-End IDS which only counted with 1200 lines of code back in 1999. The

user's community has had a major role in developing and keeping this mechanism updated over the years through submitting bug alerts, bug fixes and more, as discussed in [55].

The Snort core engine is single-threaded, designed for the popular computers back in the day. It is a light-weight cross-platform network sniffing tool that works based on a bunch of defined rules and analyzing the packets exchanged over the network. This way, and having in mind the classification stated before in Figure 2.1.7, it is a signature-based IDS. A typical snort installation can process traffic at a rate of 100-200 megabits per second [56]. Snort works not only as intrusion detection but also as an intrusion protection system, by having the possibility of dropping and rejecting packets. In order to enable full sets of rules, Snort must be running in a 64-bit machine [53].

An architecture example [56] for Snort can be depicted in Figure 2.7.

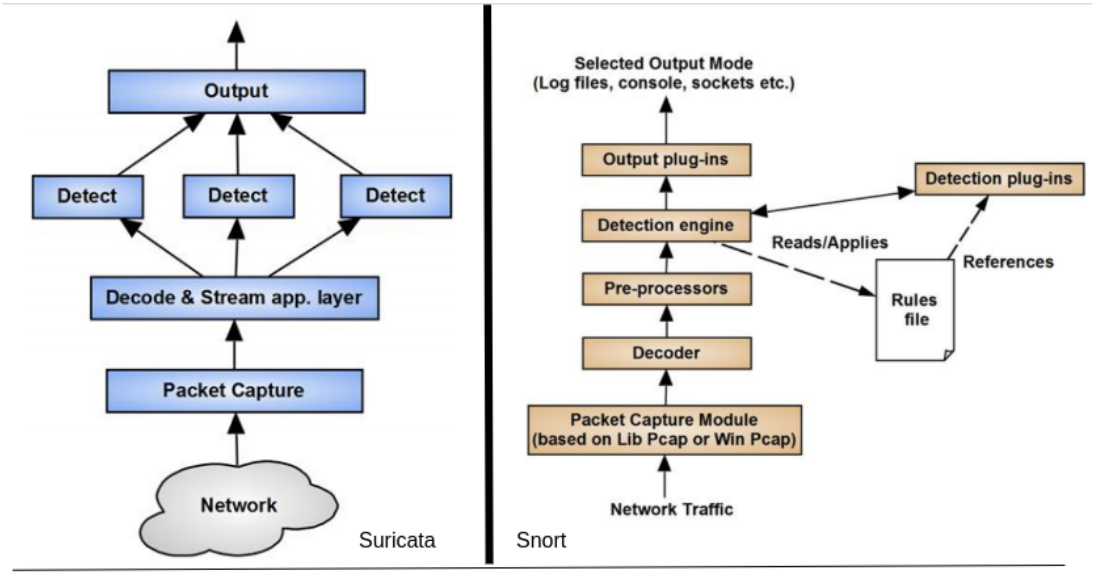


Figure 2.7: Architecture examples for Snort and Suricata, side by side.

Snort sniffs and analyses all of the network traffic. Hence, Snort has a dedicated packet capture module and sent to the Decoder where their structure is analyzed for suspicious behaviors. These can be presented as wrong packet sizes, protocol settings, etc. If something suspicious takes place here, the Decoder is able to trigger an alert, otherwise the packets follow to the pre-processor which prepares this data for proper rules application by the detection engine. At last, output plug-ins define how to deal with the triggered alerts.

2.2.1.2 Suricata

In 2009 a set of private companies, with the support and resources from the US Department of Homeland Security, founded the OISF. The biggest goal of this foundation was to develop an alternative to Snort tools called Suricata. Later in 2010, and inspired by Snort, Suricata is brought to the world as a Network-based IDS with a very similar to Snort's structure [54].

Unlike Snort, which was 10 years old and was developed for the computers at that time, Suricata has a core engine working with a multi-thread architecture. This way, Suricata is expected to much better in multi-core systems, as evidenced in [55]. Just like Snort, Suricata also performs both as intrusion detection as intrusion protection and also needs to be running in a 64-bit machine in order to load the full set of rules. This is one major example of why we consider these mechanisms as middle- and high-end.

In Figure 2.7, an example for the Suricata architecture is presented [56]. Network traffic, which can either come directly from the network interface or from pre-recorded traffic, is sent to the decode and the stream application layer where they suffer two operations. Firstly, packets are decoded just like in Snort, but then, instead of being sent in a queue to the following stage, here the packets are assembled into stream-queue which will later be fetched, for processing, by a certain thread. All threads compose the thread engine and this is how the multi-threading operations of Suricata takes place.

2.2.2 Low-end IDS

Even though the IoT is still a recent and incipient paradigm in modern technology, it was already granted with some security and privacy challenges in keeping the information and collected data safe, trustable, and consistent. However, these systems face some challenges in authentication, confidentiality, access control, mobile security and many more. For these issues, protecting networks with heterogeneous IoT devices using IDSs as a security measure is a must. Nevertheless, an IDS designed for IoT devices will have different requirements than IDSs designed for traditional networks, thus, for this reason, adapting traditional IDS approaches to the IoT concept is still a topic of great interest.

Several IDS systems for low-end devices can be found in the literature [49] [57] [58] [59], [60], [61] [62] [63]. Table 2.2, adapted from [48], summarizes the current state of the art of IDS regardign their placement strategy, detection method, security threats they can detect, and the validation strategy.

Table 2.2: Eighteen IDS solutions for the IoT.

Key Reference	Placement Strategy	Detection Method	Security Threat	Validation Strategy
Cho et al. (2009)	Centralized	Anomaly-based	Man-in-the-middle	Simulation
Liu et al. (2011)	-	Signature-based	-	None
Le et al. (2011)	Hybrid	Specification-based	Routing attack	None
Misra et al. (2011)	-	Specification-based	DoS	Simulation
DEMO (2013a) [57]	Centralized	Signature-based	DoS	Empirical
Wallgren et al. (2013)	Centralized	-	Routing attack	Simulation
Svelte (2013) [49]	Hybrid	Hybrid	Routing attack	Simulation
Gupta et al. (2013)	-	Anomaly-based	-	None
Kasinathan et al. (2013b)	Centralized	Signature-based	-	Hypothetical example
Amaral et al. (2014)	Hybrid	Specification-based	-	Empirical
Oh et al. (2014)	Distributed	Signature-based	Multiple conventional attacks	Empirical
Lee et al. (2014)	Distributed	Anomaly-based	DoS	Simulation
Krimmling and Peter (2014)	-	Hybrid	Routing attack and Man-in-the-middle	Simulation
INTI (2015) [59]	Distributed	Hybrid	Routing attack	Simulation
Summerville et al. (2015)	-	Anomaly-based	Conventional	Empirical
Thanigaivelan et al. (2016)	Hybrid	Anomaly-based	-	None
Le et al. (2016)	Hybrid	Specification-based	Routing attack	Simulation
Pongle and Chavan (2015)	Hybrid	Anomaly-based	Routing attack	Simulation

However, when handling a list of eighteen solutions, proper sampling must be used and, in this case, there are some vital concerns about the presented solutions which rule out many works. First and foremost, IoT network is continuously building and shaping itself and, for this reason, when studying the oldest works from these surveys, we found some of the older ones somewhat outdated. Secondly, many of these works are overlapping each other either in the problems they secure, either on the approaches they use in order to secure the communication or routing problems of such devices. Therefore, the ones who show overlapped approaches or targets, and show the weakest or even none results in evaluation or benchmarking concerns, must be dismissed. Thereafter, the result from a selective tapering of the previous table of solutions results in less than a handful of solutions which have the ability to represent all the relevant placement strategies, detection methods and different kind of security threats. These works are gathered in the Table 2.3 and represent the works we will be studying in the following sections.

Table 2.3: Key features of representative IDS solutions for each category.

Key Reference	Placement Strategy	Detection Method	Security Threat	Validation Strategy	Detected Attacks
DEMO (2013a) [57]	Centralized	Signature-based	DoS	Empirical	IPv6 UDP flooding attack
Svelte (2013) [49]	Hybrid	Hybrid	Routing attack	Simulation	Sinkhole and selective-forwarding attacks
INTI (2015) [59]	Distributed	Hybrid	Routing attack	Simulation	Sinkhole attacks
A Signature-based IDS of IoT (2018) [64]	Hybrid	Signature-based	DoS	Empirical	"Hello Flooding" and Version Number Modification

2.2.2.1 SVELTE: Real-time IDS in the IoT

In 2013, Raza et al. [49] have proposed Svelte, an IDS developed to primarily target routing attacks such as spoofed or manipulated data, as well as sinkhole and selective-forwarding. Svelte is designed for 6LoWPAN networks and makes use of RPL.

Following the taxonomy proposed and discussed in [48], this IDS has a hybrid placement strategy, because it has both centralized and distributed modules within the 6LoWPAN network. Centralized modules are found in the 6BR while the distributed modules can be found in the constrained nodes.

The architecture proposed by [49] can be seen in Figure 2.8. It is possible to understand that this execution flow consists of many network nodes capable of sending information and traffic reports to the border router. This last is responsible for performing intrusion detection analysis to detect if any malicious activity is taking place. This is possible because of two main software modules located in the 6BR, which are the *6Mapper* and the *Mini-Firewall* and their corresponding light-weight modules in the regular network nodes.

The most important module of this architecture is the *6Mapper*, which gathers information about RPL network and reconstructs the RPL DODAG in the 6BR. Having a secure and trustworthy mapping of the 6LoWPAN network is very efficient for the detection of many attacks. For example, sinkhole attacks, which take place when an attacker advertises a better routing path, can be prevented with the RPL DODAG information. To reconstruct the DODAG, *6Mapper* sends mapping requests to the nodes in the 6LoWPAN network. These mapping requests are five bytes long and contain some information such as RPL instance ID, DODAG ID and DODAG version number. Nodes in the network will respond to the mapping request with packets with a size of 13 bytes, with four extra bytes per each neighbor of such node. The response packet contains the following information: node rank, parent ID and all neighbors IDs and ranks. As a

security measure, the authors state that packets used to map the network must be indistinguishable from other packets, otherwise attackers can perform a selective-forwarding attack exploiting this breach.

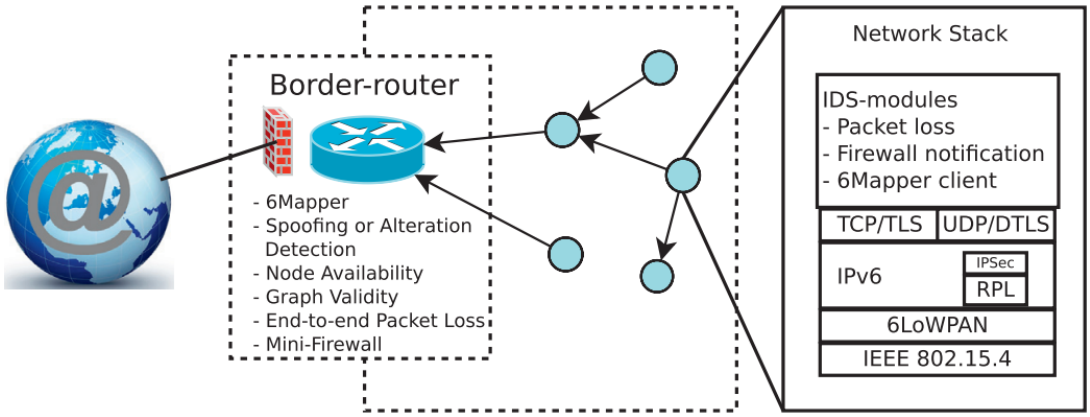
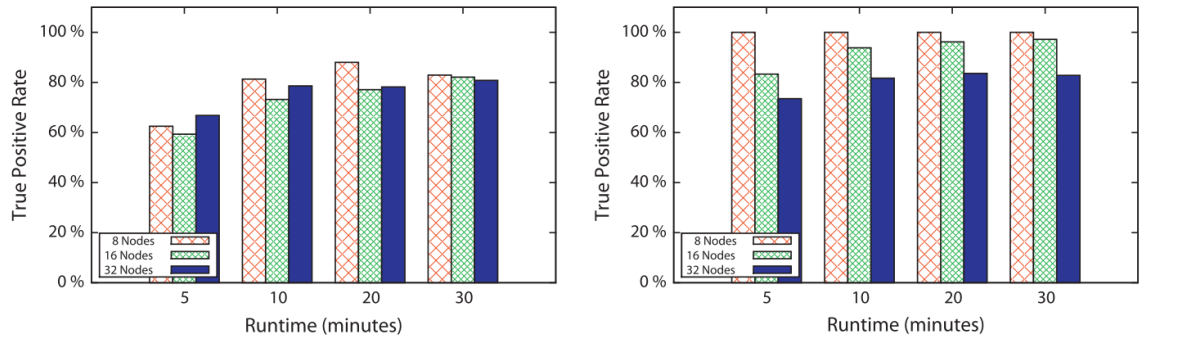


Figure 2.8: An IoT setup where IDS modules are placed in 6BR and also in individual nodes.

The other big contribution of [49] work is the *Mini-Firewall*, which has a module in the 6BR and also in the regular network nodes. Besides providing typical blocking functionalities against already known external attackers, specified by the network administrator, the firewall can block the external malicious hosts specified in real-time by the nodes inside the 6LoWPAN network. Nodes inside the 6LoWPAN can only choose to filter the traffic destined to itself. For an external to be blocked to all nodes, a minimum set of nodes need to have complained about such specific host.

At last, Svelte has a few more Intrusion Detection Algorithms for network analysis, which support the *6Mapper* and the Firewall. Such algorithms are Network graph inconsistency detection; Checking node availability; Routing graph validity; End-to-end packet loss adaptation. It is also convenient to refer that *6Mapper* is capable of passively protect the network against Sybil and Clone ID attacks.

All evaluations show high success rates for all the tested attacks. During the evaluation described by [49], with an example of such in Figure 2.9, it is possible to conclude that Svelte behaves the best when in a lossless network when compared to a lossy network. This is due to the time necessary for the *6Mapper* to build a stable and secure network map.



(a) *Lossy* network suffering from selective forwarding attack (b) *Lossless* network suffering from selective forwarding attack

Figure 2.9: Evaluation of Svelte performance in lossless and lossy networks for a selective forwarding attack.

Also, the evaluation shows that the rates drop with the amount of nodes in the network, once again, because bigger networks will demand more time to be mapped. In lossy networks, with few run-time, e.g., 10 minutes, and with many nodes, e.g., 32, the true positive rates drop up to 60%. These rates will, however, behave much better in lossless networks with considerable run-time and fewer nodes, where positive rates surpass the 90% rates.

Svelte modules show a small memory footprint, as they only require an additional 0.365kB of RAM in a 10kB Tmote Sky. The authors also claim that 1.76k required ROM of the *6Mapper* is perfectly suitable for constraint nodes, such as Tmote Sky with 48k, even though it is designed for 6BR which are typically less constraint in-memory resources, like a desktop or laptop computers.

Concerning energy overhead, Svelte induces negligible consumption in network-wide without duty cycling, where the radio is always turned on to receive and transmit packets. However, in duty-cycled networks, where the radio is off for approximately 98% of the time, the overhead can reach up to 30%, in bigger networks. The current drawback of Svelte is that it was released in Contiki 2.6 version, and never updated, thus only provides support for the classic version of RPL, which is being replaced by RPL-Lite in the latest Contiki-NG. Furthermore, Svelte has no reference whatsoever to DoS attacks, which are known to be another ferocious attack towards these devices.

2.2.2.2 DEMO: An IDS Framework for IoT Empowered by 6LoWPAN

Also in 2013, Kasinathan et al. [57] proposed a centralized solution called DEMO, where the main goal for this IDS was to detect DoS attacks in IoT 6LoWPAN-based networks. DEMO is a signature-based IDS with empirical validation, developed against IPv6 UDP flooding attacks.

Contributions made with [57] work are reported only has enhancements to an already existing structure proposed in [58], however, from the same author. These point out that security breaches may happen if the information packets in a security mechanism like IDS are transmitted within the wireless medium. In their solution, wired communication is established between the IDS engine, and the IDS probes sniffing the network.

Figure 2.10 ilustrtes the DEMO framework architecture. This architecture has an advanced event monitoring system and also the IDS engine. The monitoring system consists of Frequency Agility Manager (FAM) and Security Incident and Event Management System (SIEM). Frequency Agility Manager is a mechanism that allows a network to become aware of the interference level by analyzing channel occupancy states in real-time, thus, making it possible to choose the best available channel at a given time. FAM switches the operating channel when the interference level exceeds a certain threshold. Prelude SIEM is responsible for monitoring the attacks events or alerts. It receives information from the IDS engine and is also capable to receive interference information from the FAM. Prelude integrates a wide range of security tools under one monitoring system, making it possible to minimize the false positive alerts by correlating information available from other network monitoring tools such as FAM. Attack confirmations can be triggered by Prelude, which can extend notifications by sending email or SMS alerts to the network administrators.

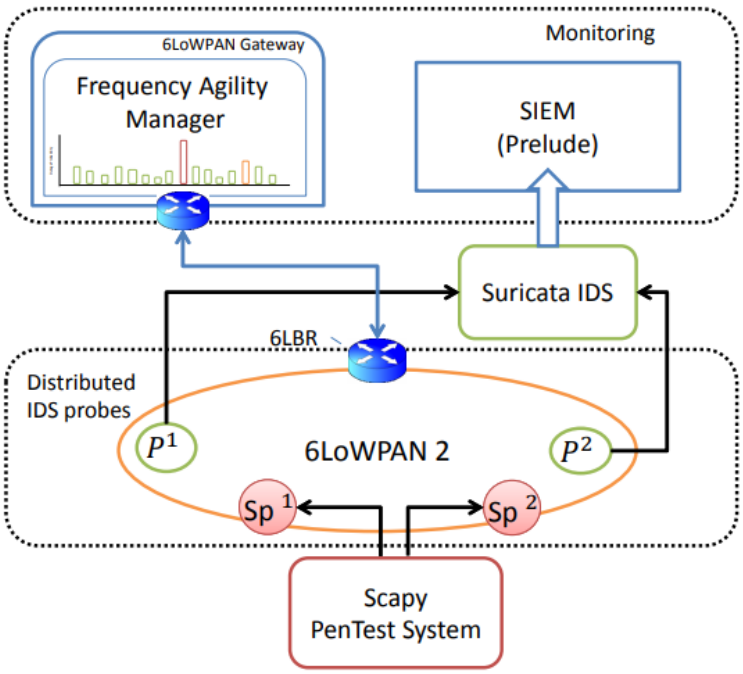


Figure 2.10: DoS detection architecture for the 6LoWPAN.

This solution's IDS engine is made possible through Suricata. Since Suricata was developed to detect attacks on traditional networks, protocols from WSN and 6LoWPAN are not understood by Suricata so there was a need to develop decoders for these protocols, to Suricata, so the engine could analyze the network packets.

No evaluation is presented on the paper, however, the authors explain that testing is made using the Scapy tool, a powerful packet manipulation program that can generate attacks, for example, flooding attacks, but also more complex attacks such as RPL-rank attacks. According to the conclusions made, results proved that the proposed solution was scalable and stable, also appearing as a promising solution for the future 6LoWPAN security. DEMO's capabilities can be developed to detect more complex attacks by developing specific modules for Suricata, e.g., by extending Suricata's engine to support anomaly detection.

2.2.2.3 INTI: Detection of Sinkhole Attacks for Supporting Secure Routing on 6LoWPAN for the IoT

Proposed in [59] and described as Detection of Sinkhole Attacks for Supporting Secure Routing on 6LoWPAN for the IoT, INTI combines concepts of trust and reputation with watchdogs for detecting and mitigating attacks. This system implements a hierarchical structure of nodes where each node as a role in the system. These can be leader, associated, member or a free node. The main task of each node, within the IDS play-role, is to monitor a superior node estimating its traffic patterns like inbound and/or outbound traffic. When a node detects a sinkhole attack, it broadcasts a message to alert other nodes.

In comparison with Svelte [49], INTI discusses and presents the concept of mobility within the 6LoWPAN network nodes, stating that such mobility is expected in real life network behavior, resulting in a non-fixed network mapping possibility. Thus, INTI operates with a virtual clustering of nodes, and also a hierarchy within the clusters, which is constantly being updated and altered with changes related to mobility but also changes resulting from attacks. This way, the network is equipped with self-organizing and self-repairing properties. Since RPL was only developed to work with static devices and environments, INTI also developed a new routing protocol inspired on the RPL, taking into account the device's mobility and cluster information.

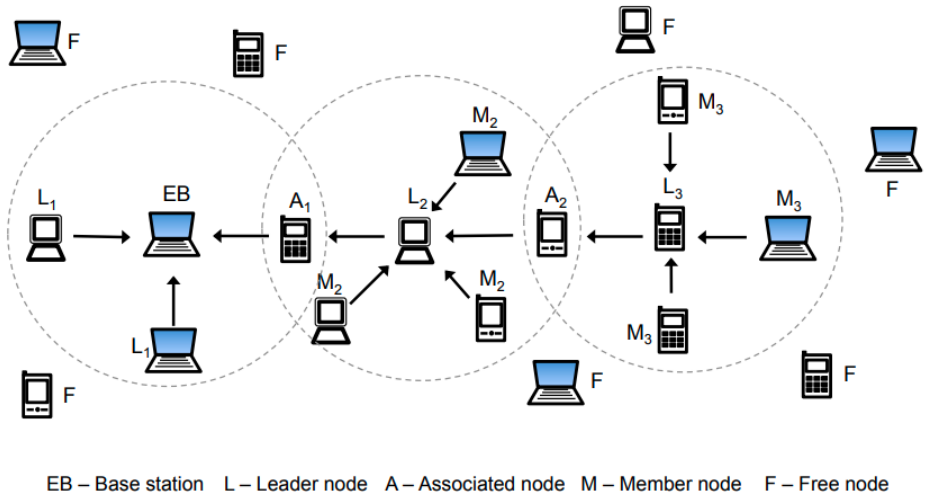


Figure 2.11: INTI IDS system entities.

In Figure 2.11 it is presented an example of a INTI network entities arrangment, with all of the existing entities in the INTI system. The dashed line is representing the created virtual-cluster. INTI system architecture has four modules: cluster configuration, routing monitoring, attack detection, and attack isolation. Initially, all nodes are free nodes but the cluster is built with runtime. INTI makes use of the Beta Probability Density Function in order to estimate the state of a node concerning its past behavior. This function is essential for trust and reputation algorithms. Attack detection in INTI takes place by evaluating all of the node's trust and reputation, in real-time. Whenever a node detects his neighbor's confidence value below a defined threshold, this node will then broadcast an alarm message in order to alert all the neighbor nodes.

Even though INTI has a distributed placement strategy, having all of its software running on the constraint nodes, the authors still imply that this solution results in less resource consumption, in comparison with Svelte [49] and DEMO [57], resulting in a smaller hit on network and system performance. However, no evaluation is presented concerning energy consumption in [59].

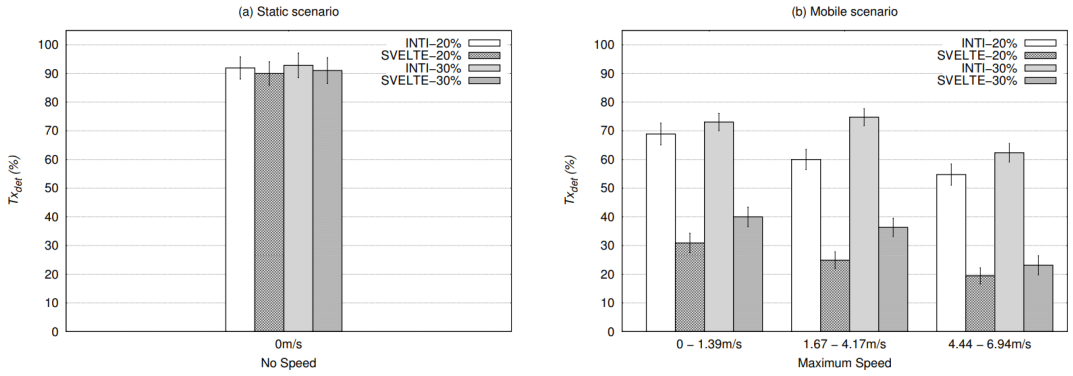


Figure 2.12: INTI IDS system evaluation in comparison with Svelte IDS for false positives and false negatives.

INTI is evaluated in the Cooja simulator, just like Svelte, given out the purpose of comparing both systems. Evaluations are performed with large networks, with 50 nodes, and comparisons are made concerning four metrics: detection rate, false negative, false positive and delivery rate. All evaluations are realized with sinkhole attacks, no other routing attacks are covered in this paper neither are Sybil ID and cloned ID, which Svelte claims protection against.

It is possible to see in Figure 2.12 an example of the charts presented, in which is presented the detection rates of sinkhole attacks for both systems, taking account no mobility on the left, whereas, on the right, mobility is present. Detection rates are very similar in a fixed scenario whereas in a mobile scenario INTI shows a much more positive rate. As false negatives concern, INTI behaves really good in the fixed scenario but with mobile nodes, Svelte provides the best results, curiously. Rates for false positives are again very similar. At last, Svelte presents a delivery rate that reaches 99% in fixed scenarios whereas INTI takes the best score around 75% for mobile scenarios. No energy consumption's neither memory usage evaluations are presented in INTI.

2.2.2.4 A Signature-based Intrusion Detection System

The last work reviewed in this thesis concerning existing low-end IDS solutions for the IoT is a Signature-based Intrusion Detection System proposed in [64], which involves both centralized and distributed IDS modules, resulting, this way, in a hybrid solution for detection of DoS and Routing attacks.

This IDS solution, just like Svelte, which is also hybrid concerning the placement strategy [49], also places the centralized IDS module in the main router, which can be the border router of the 6LoWPAN network. However, this solution does not place the distributed IDS modules running inside the constraint

nodes of the network, lightweight modules are deployed in the network in close proximity to the nodes for the purposes of traffic monitoring and reporting. Just like DEMO [58], all the IDS modules are connected via wired communication channels in order to avoid jamming or other types of wireless attacks which can happen if these packets were being transmitted within a wireless medium.

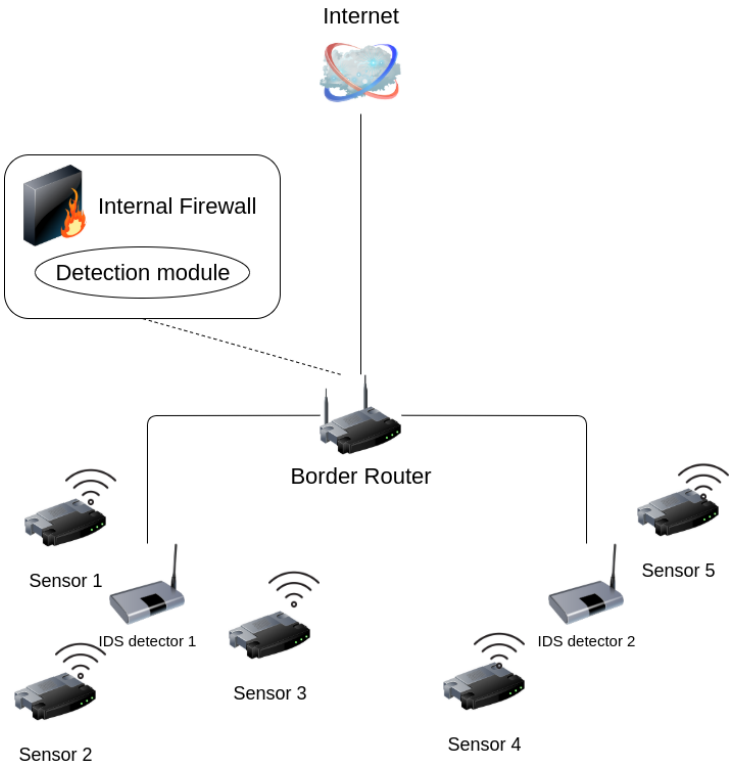


Figure 2.13: DoS detection architecture for the signature-based IDS.

In Figure 2.13 we can see the architecture for this solution. There are two new devices added to a standard network, the IDS router which runs not only the detection module but also a firewall, and the IDS detectors that monitor the network and send suspicious traffic to the router. Besides from sending the data to the IDS router, lightweight modules also perform algorithms to only forward the necessary traffic. Malicious patterns are stored in the border router. For malicious activity detection, four main metrics are constantly measured, being these the Received Signal Strength, packet data drop rate, packet sending rate and the number of nodes ID in the network.

Cooja was also used for testing and experimentation purposes. The authors tested only two scenarios in Cooja, using the same network specifications in both. In the first scenario, there is no threat, however, in the second scenario, one of the nodes was performing both hello flooding and version number alteration. This way, the authors showed the negative effects these attacks can have on a network using RPL. Neither

evaluations nor results are presented for this IDS solution, concerning memory usage, power consumption or detection rates.

2.3 Platform and Tools

In this section, we will briefly approach the platform and tools used throughout the implementation of this project. There is one very-known application from the Contiki OS that constitutes a vital tool for the development of this project, the Cooja Network Simulator. This tool will allow for all the network simulations of this project, and we will better approach this tool in the following Section 2.3.1.

2.3.1 Cooja Network Simulator

Developing an IDS for the IoT is only feasible if reliable testing of the system is performed. Setting up a physical network of devices in a secured and controlled environment would be too much of a burden, if even financially possible, and for this reason, simulation is the wisest choice. Simulating an IoT network where an IDS would be integrated requires for a network simulator not only capable of simulating large networks of devices but also capable of simulating different devices running different firmware. Furthermore, the network simulator should be able to supply with a reliable computation of performance metrics of the IoT based smart devices.

Contiki OS, previously analyzed in Section 2.1.5, offers Cooja Simulator, a network simulator specifically designed for WSN which allows for emulation of real hardware platforms. For this convenient reason, and also for the acknowledged performance of this tool [65][66], Cooja was used as the preferred, and only, network simulator throughout the development of this project.

Cooja supports many sets of hardware motes and radio transceivers, such as TR 1100, Texas Instruments (TI) CC2420, and allows real simulation with message decoding of standard protocols such as RPL, IEEE 802.15.4, uIPv6 stack and uIPv4 stack. Cooja also provides a flexible and easy-to-use interface built up in different windows. There are five main windows, as seen in Figure 2.14, such as: the network window, which displays the physical arrangement of the runtime window, mote output, etc. Furthermore, this window can also display a 10-meter grid, different colors to different mote functionality in the network (sink, server, client, etc...) and radio environment of each node; The simulation window controls the operation of the simulation, through the speed parameter and the pause, start and reload button; The mote output window allows for a time-based visualization of all data printed and/or exchanged between each mote.

This window can allow filters to be added, in the bottom field, enabling a mote-specific view. The timeline window enables visualization of both power consumption and network traffic in the 6LoWPAN network. At last, the mote interface viewer allows for multiple types of configurations that can track multiple metrics of the node in runtime. For instance, in Figure 2.14 there are three different mote interface viewers displaying the status of three node-metrics. The far-left one allows us to press the node's button during the simulation, the next is showing the status of the node's LED's, and the last is showing the node's serial port. More metrics can be analyzed in this mote interface viewer.

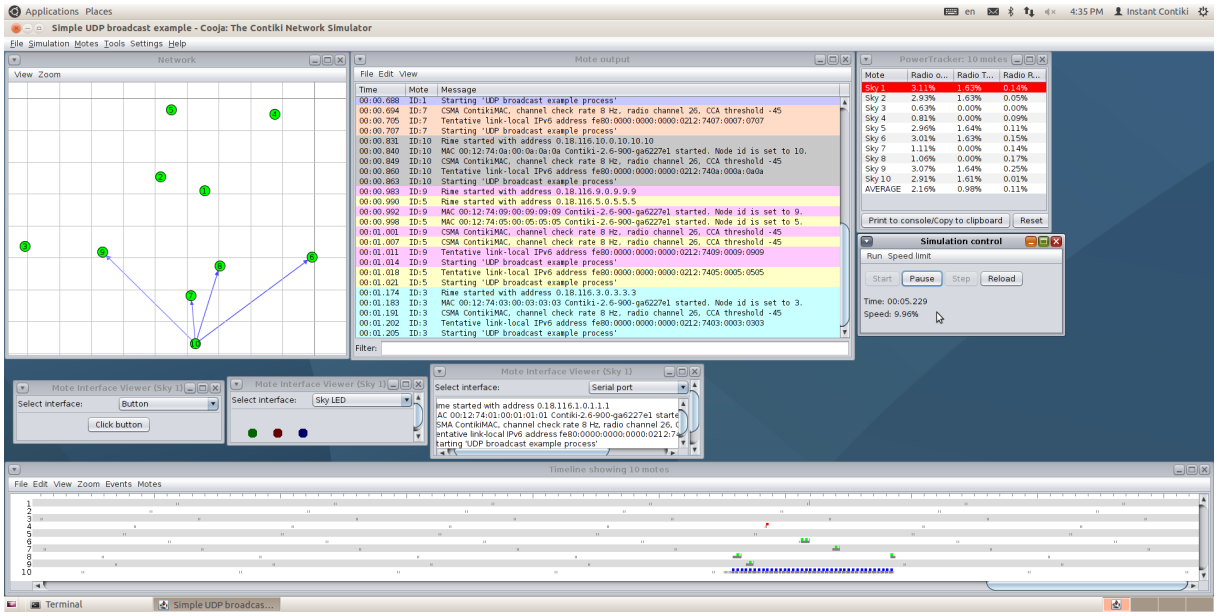


Figure 2.14: Contiki Cooja Network Simulator Environment.

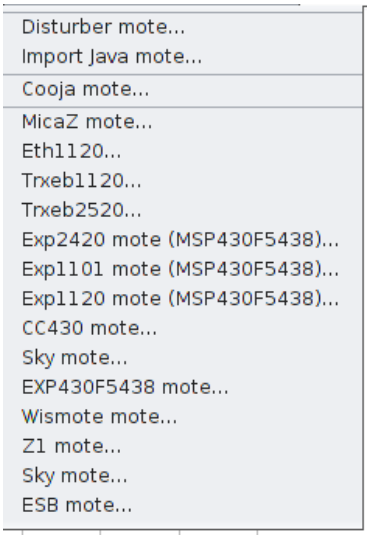


Figure 2.15: Contiki Cooja Network Simulator available motes for emulation.

Cooja network simulator has support for many devices, such as the MicaZ mote, MSP-EXP430F5438 mote from TI, Wismote, Z1 mote from Zolertia, Tmote Sky mote and some more. Hardware motes available to emulation by Cooja can be seen in Figure 2.15. Cooja can successfully emulate all these hardware platforms and build reliable simulations. For instance, Tmote sky is an ultra low power wireless module for use in sensor networks, monitoring applications, and rapid application prototyping. It consists of an 8 MHz TI MSP430 MCU with 10kB RAM and 49kB flash as well as a 250kbps 2.4GHz IEEE 802.15.4 chipcon wireless transceiver. This is the preferred device to simulate in a project, since its one of the most constraint devices supported by Cooja, allowing for the developers to always remember the resource constrained environment they are developing, and to test network applications before the real deployment.

Chapter 3

System Model and Design

This chapter will proceed on presenting the system model and design for the IDIoT IDS. Firstly, a GAP analysis is performed in Section 3.1 in order to understand which contributions are possible to add with this solution to the current literature. Then, the proposed solution is detailed in 3.1.1 followed by all system assumptions in Section 3.1.2. Then, for the rest of the chapter, the system model and design is presented in Section 3.2, where the entire system will be covered. For a more in-depth view, we will be analyzing the modules running in both border router and constraint nodes of the network, in Section 3.2.

3.1 GAP Analysis

In previous Section 2.2.2 it was possible to understand that many solutions addressing the implementation of IDS for the IoT paradigm have been published. For instance, [27] published a survey studying twenty published solutions, with dates ranging from 2009 to 2017. Furthermore, Zarpelão et al. [48] is the author of the survey discussed before, in Section 2.2.2, which also analyses eighteen published solutions, which release dates range from 2009 to 2016. Both works have impressive content regarding the research of present literature for IDS for IoT and both present some concerns and considerations about the lack of conclusions and closures regarding many metrics of studied solutions.

Concerns are drawn as the research in IDSs for the IoT are still in its infancy and incipient [27] [48]. Works reviewed do not cover or address a lot of IoT technologies and cannot detect a large variety of attacks [27][48], the most usual being routing attacks, DoS and man-in-the-middle. Furthermore, the proposed solutions do not investigate the strong and weak points of each possible detection method and placement strategy, as well as not reaching a consensus of which is the most proper one. Additionally, in most of the cases, security over the communication between IoT nodes is only an assumption. This is a real concern since many of the services provided by IoT networks and devices contain private data. This problem with

security in communication between network nodes also applies to the communication and traffic related to the IDS mechanism, as most of the solutions did not refer to any kind of security over the control data from the IDS itself. Last but not least, for all the studied, there is a clear lack of extendibility and configurability, meaning that a lack of clear instructions and details for adding more attacks to the detection engines is found, thus increasing a barrier in the use of already designed tools [48]. Concerning the configurability, most of these solutions also fail to present some kind of adjustment to the network needs during runtime of the devices and their tasks.

In the previous Sections 2.2.2.1 to 2.2.2.4 the 4 most relevant low-end IDS solutions for the IoT were presented. These solutions suffer from some or even all of the problems previously pointed out and, at this moment, none of these seem really capable of effectively and stand-alone securing an IoT network with constraint nodes exposed to the malicious users which may arise from the connection to the Internet.

Regarding Svelte, the main concern is the lack of clear strategies to prevent or protect against DoS attacks, which are as disruptive to a network as the sinkhole attack which is the main focus of Svelte. Also, Svelte makes use of the RPL protocol but fails in delivering protection against exploits of the RPL using control messages against itself. At last, Svelte assumes the communication between nodes to be secure and this can also be an issue with networks where nodes handle data with higher privacy concerns.

In Section 2.2.2.2 DEMO was described, along with its particularity of using wired communications. DEMO deploys physical IDS probes to the network, more than one, depending on the size of the network, in order to send data to the centralized IDS engine. Although the authors introduce this particularity as a great solution in order to secure the communication of control data from the IDS, there are major concerns regarding the drawbacks of this approach, namely the increased cost of having to deploy physical devices to the network, running their specific firmware for the IDS. Additionally, the major difficulty for the network administrator to connect these wired devices, greatly limiting the possibility of having a network with mobile characteristics. Furthermore, DEMO presents no protection against routing attacks like Sinkhole and Selective-forwarding, protected in Svelte, and does not implement security measures against all RPL exploits. DEMO does not deliver any kind of evaluation of its metrics against the DoS attacks and cannot be considered an open-source solution since there is no access to the project or related algorithms and flow charts.

INTI solution is an improvement to Svelte concerning the mobility features. In spite of having a good evaluation concerning detection rates and false positives, no evaluation is presented regarding energy or memory overhead which is a major concern for this solution since it empowers the constraint nodes

with many more functionalities and responsibilities for the construction of the network map, for example, these nodes are constantly calculating neighbors trust and reputation levels using Beta Probability Density Function. Much like Svelte, INTI only covers routing attacks and fails to deliver protection against DoS attacks. At last, INTI also assumes communication between nodes to be secure and cannot be considered as an open-source solution since there is no trace of the project or any kind of algorithms and flow charts.

Finally, the forth and last work covered, in 2.2.2.4, proposes a signature-based solution which, just like DEMO, is focused in protecting DoS attacks but has no protection strategy againts routing attacks. Furthermore, and having the same major concerns as DEMO does, this solution is based on the deployment of physical IDS probes to sniff the network which are wired-connected to the central IDS module. As discussed before, these strategy leaves major concerns with increased cost and reduced mobility of the networks. This signature-based solution makes use of RPL protocol nonetheless it does not prevent any kind of exploits of the RPL control messages. No evaluation or benchmarking is performed and this solution cannot be considered as an open-source solution since there is no trace of the project or any kind of algorithms and flow charts.

3.1.1 Proposed Solution

In accordance with the survey of existing IDS solutions for the IoT, [50], we draw the conclusion that a hybrid IDS integrated into the 6LoWPAN protocol, would be the most promising IDS for the future of the IoT. Additionally, combining detection methods in order to make the best of anomaly-based and specification-based strategies would be most successful concerning detection rate and energy overhead. Furthermore, for solutions making use of the RPL protocol, it is essential for the solution to secure this protocol and the devices againts RPL exploits, namely the DIS and DAO attacks [67] [68] [69]. Regarding the attack protection, the IDS solution is expected to be capable of securing the network against both two most popular attacks, Routing and DoS attacks [70].

The GAP Analysis performed in previous Section 3.1 was the base for the development of the IDIoT solution and these metrics compose the IDIoT description which are presented in Table 3.1. On top of these introduced metrics for the IDIoT, it is also intended for the proposed solution to be able to accomplish a role of independence towards the OS of the platform it is deployed. Furthermore, it is envisioned a configurable and extensible point of view of the solution in order to both being able to deploy more protection rules at the moment of the integration with the network, and also of configuring the thresholds of such rules. At last, we pretend to deliver this project to the community making this an open-source solution.

Table 3.1: IDIoT metrics side by side with studied IDS for the IoT solutions.

Key Reference Name	**Placement Strategy**	**Detection Method**	**Security Threat**	**Validation Strategy**	**Detected Attacks**
DEMO (2013) [58]	Centralized	Signature-based	DoS	Empirical	IPv6 UDP Flooding attack
SVELTE (2013) [49]	Hybrid	Hybrid	Routing attack	Simulation	Sinkhole and Selective-Forwarding Attacks
INTI (2015) [59]	Distributed	Hybrid	Routing attack	Simulation	Sinkhole attacks
A Signature-based IDS of IoT (2018) [64]	Hybrid	Signature-based	DoS	Empirical	"Hello Flooding" and Version Number Modification
IDIoT	**Hybrid**	**Hybrid**	**DoS + Routing Attack**	**Simulation**	**Sinkhole and Denial of Service**

3.1.2 System Assumptions

First, concerning the positioning of our proposal, it is assumed that the communication between the network nodes is secured and authenticated. Secure communications can be provided by upper layers, and for the sake of simplicity, the evaluation was performed with packets being transmitted in plain text, which did not affect the system deployment and its proper operation. We also assume that for the routing attacks, the malicious node is already inside the network. For the DoS attacks, they can be performed from anywhere inside the network or from the Internet.

Regarding the IoT devices, this solution is intended to run in low-end IoT devices with tiny resources. They are expected to have below 10kB of RAM and less than 100kB of flash.

Moreover, it is expected that networks where these low-end IoT devices are running, can perform some kind of critical service and therefore data availability is a priority. Regarding the network, it is assumed that a network will always be composed of more than 1 constraint nodes, and we assume that a network must always have a 6BR with higher resource availability, over 100kB of RAM or even the size of a Raspberry Pi or small laptop.

Regarding mobility of the devices, we assume that these can change location once in a while, during their run time, for example, a smart light-bulb changing rooms within a house or sensors for a farm being changed from place to place in accordance with the needs of the farmer to track this or that plantation. This way, we assume the mobility will not be constant, for example as a smartphone being carried in a postman's pocket.

At last, motes will not run in networks under arsh and remote environments. These motes are expected to work within environments ranging from indoor houses with stable air flows and network connections, up to some outdoor applications where a little wind and dense air can be expected at some times.

3.2 System Overview

The proposed solution shares some strategies with an already existing and previously studied solution of an IDS for the IoT. For instance, with a careful look at table 3.1, both IDIoT and Svelte are hybrid, concerning both placement strategy and detection method. Furthermore, both solutions intend to protect the devices and networks against routing attacks such as the sinkhole and the selective-forwarding. Svelte implementation makes use of 6LoWPAN and the RPL as is intended in our solution and Svelte's successfully proved its truthfulness with simulations concerning memory overhead, energy overhead, and many more, whithin the constraint nodes, which we believe is crucial for a solution of this kind. We intend in doing the same extensive evaluation.

For these reasons, and after an thorough study of the Svelte architecture and implementation, the conclusion was that not developing an IDS engine and architecture from scratch, but instead improving the Svelte IDS for all the intended requirements of our solution, would be the foremost choice in order to achieve the best results for our contribution. Many improvements and contributions to the existing architecture of Svelte will be performed. The refered solution was published in 2013 [49] and was designed over Contiki-2.6 at that time. Architecture was mostly constituted of modules in the application layer and some minor changes to the network layer.

Contiki developers continued developing this OS during the years and in May-2018 a huge change was released, the Contiki-NG. At that moment, the Contiki-3.x version was the most recent and stable version of the Contiki OS. It was kept available for the community to work with, but developers shifted all work for the new and improved Contiki-NG. In the current Contiki-NG, the old RPL implementation was set aside for a brand new implementation called RPL-Lite. The old implementation even was considered to be deleted by some developers, but the community concluded that users and the community would benefit the most if it remained available as its last stable version. It is now called the RPL-Classic and its possible to use it in Contiki projects. Now ahead is the RPL-Lite which has many new features and seems much promising to the Contiki future and therefore promising for any application running in Contiki aswell.

For this reason an upgrade and refurbish of Svelte for the Contiki-NG is mandatory. For instance, Svelte used the Contiki-RPL implementation from Contiki-2.6 to Contiki-3.x. The architecture will be accordingly upgraded in order to make use of Contiki-NG's RPL-Lite implementation. Futhermore, enhancements to the *Firewall* will also be alongside with the efforts on the DoS attack protection, as each the rule-based model and the mini *Firewall* will be co-dependent. At last, enhancements to the *6Mapper* module and the detection and correction modules, inherited from Svelte, will also be developed in order to provide for a more stable and reliable flow of data collection and attack detection.

Figure 3.1 compiles an overview of the IDIoT architecture, showing an example of a network layout with three agents on sight: a border router which is the network root, some network nodes scattered randomly, and an internet connection, performed through the border router. On both the border router and the network nodes, a zoom-in table is displayed showing the network stack of the device in which is possible to locate the placement of the IDIoT.

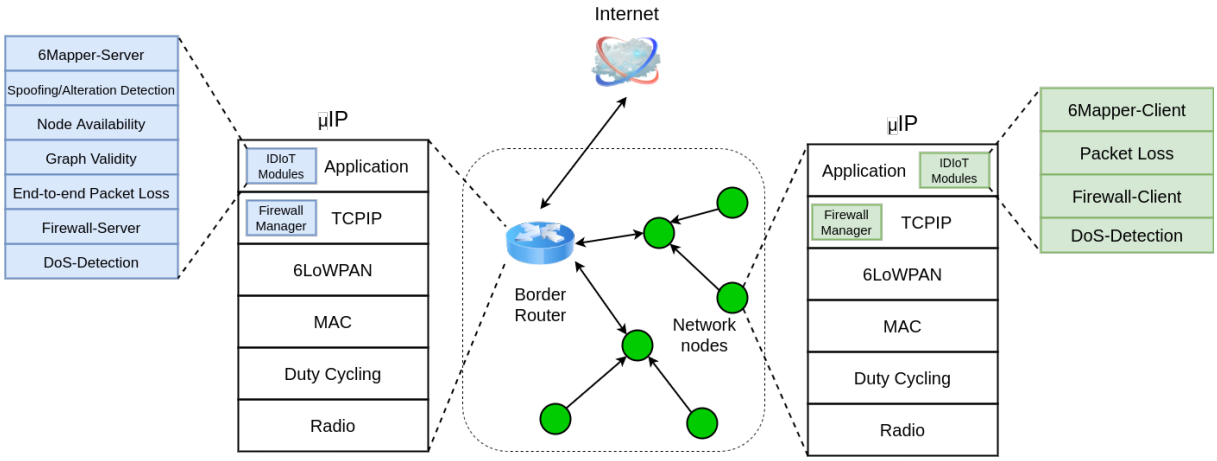


Figure 3.1: Overview of the IDIoT architecture.

In Figure its possible to understand the hybrid nature of the IDIoT regarding the placement strategy, as there are both centralized and distributed modules present in the border router and the network nodes, respectively. On the the network nodes, the modules inherited from Svelte are the *6Mapper* client and the Packet Loss, whereas the modules added in our project are the *Firewall-Client*, the *firewall-Manager* and the DoS detection. In the 6BR, the modules inherited by Svelte are the *6Mapper*, Spoofing/Alteration Detection, Node Availability, Graph Validity, End-to-end Packet Loss and the *Firewall-Server*, whereas the modules now developed in our project are the DoS detection and the improved *Firewall-Server* and *manager*.

The IDIoT architecture is present in the application layer and network layer, just like Svelte, which worked in the same grounds. However, Svelte was mainly present in the application layer and only few

features run in the network layer. At this moment, IDIoT introduces more modules to the application layer, in both border router and network nodes, but also in the network layer, where features are more significant in both devices due to many new capabilities from the *Firewall*. Figure 3.2 depicts the border router, which is the central device with all centralized modules, detailing its dependencies between other software modules.

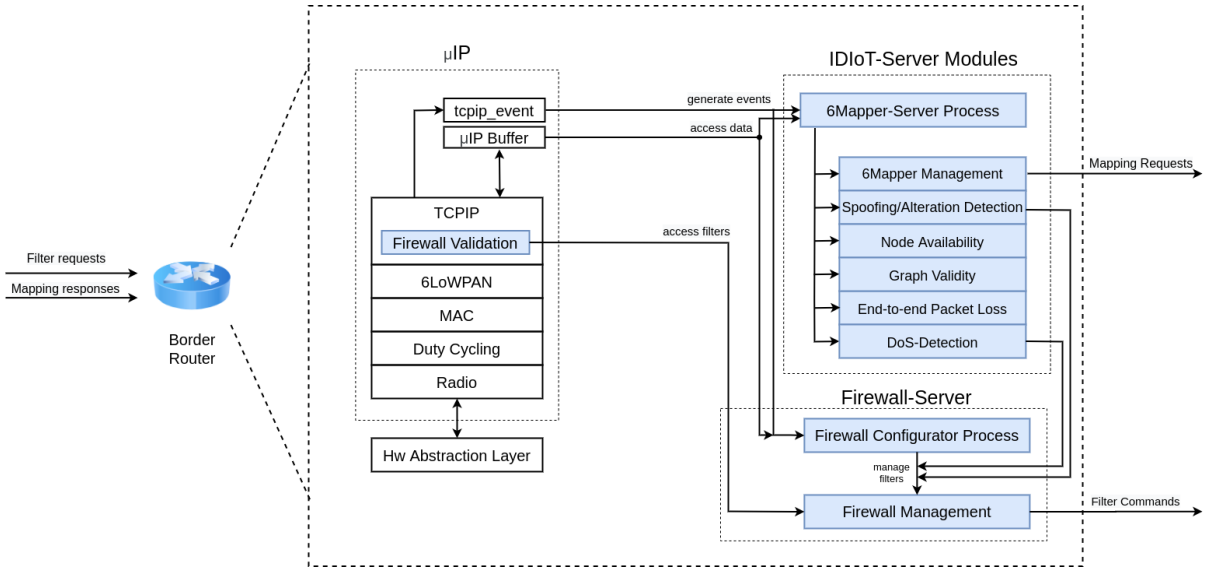


Figure 3.2: Overview of the IDIoT Border-Router architecture.

These communications concern the IDIoT flow between modules, so basically they represent control packets and these can be both periodically and occasional. On the left side of the Figure 3.2, it is possible to see that this device receives two kinds of control packets: filter requests and mapping responses. These packets are sent from the network nodes. On the right side of Figure 3.2 there are the control packets generated by this device, and sent to network nodes: the mapping requests and the filter commands. Furthermore, it is possible to see the communication between the TCP/IP and the *Firewall-Server* management for filter inquiry, as well as intra-modular communications, for example, between the DoS detection module and the *Firewall* management. In Figure 3.3, the network nodes architecture overview is shown.

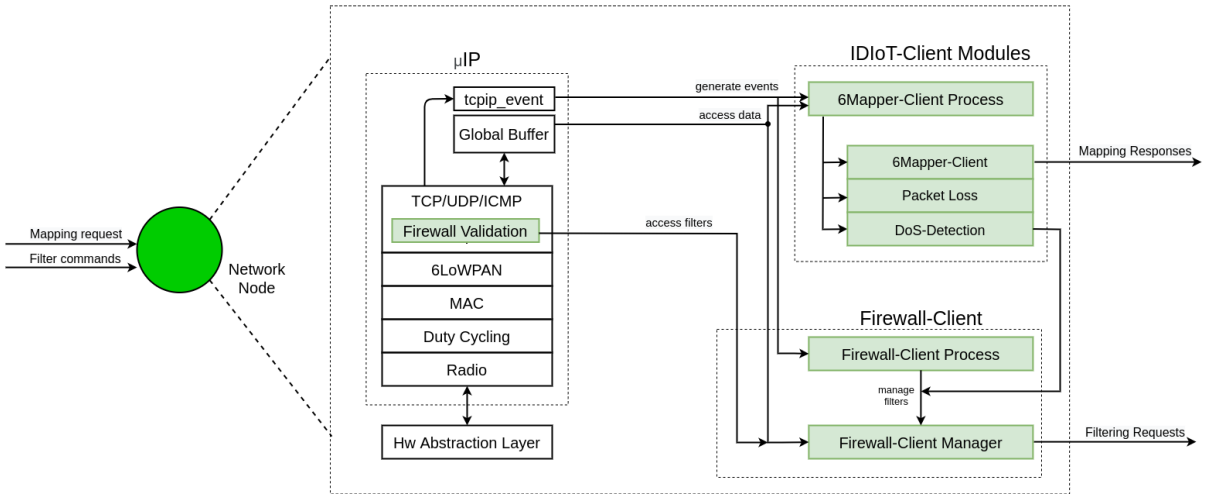


Figure 3.3: Overview of the IDIoT regular nodes architecture.

Packets received are the mapping requests and the filter commands, sent from the border-router. Packets sent are the filter requests and the mapping responses, in accordance with what was seen before in Figure 3.2. As it would be expected, the amount of traffic generated by these control packets of the IDIoT is mostly experienced by the border router. Since a network node only receives two types of control packets from one border router, network nodes undergo much less traffic than the border router which receives two types of control packets from all the nodes functioning in the network.

Moving on, regarding the internal communications of the modules and the network stack, IDIoT modules of both devices are established on two Contiki processes which allow the flow of communications. There is one process for the *6Mapper* and another for the *firewall*. Both processes are first executed on the device reset and configure the work flow. In the course of the device runtime, these processes are triggered in accordance with the established events. Such events are either timers expiring or packets received trough the UDP connection. Both the timer and the UDP connection are previously established in the first execution of the process. Once again, in Figure 3.3, as well as in Figure 3.2, DoS detection module performs communications with the firewal managment which evidences the dependency between detection modules and *Firewall* procedures to filter attackers.

From this point on to the rest of this thesis, for consistency purposes, the aforementioned and presented IDIoT modules will be referred as: *IDIoT-6Mapper*, *IDIoT-Routing-Detection*, *IDIoT-Firewall*, *IDIoT-DoS-Detection*. Some of these may even be referred concerning the placement in the network nodes, such as the *IDIoT-6Mapper-Server* and *IDIoT-6Mapper-Client* , as well as the *IDIoT-Firewall-Server* and the *IDIoT-Firewall-Client*.

3.2.1 IDIoT-6Mapper

The largest module of the IDS is the *IDIoT-6Mapper* . This module, as seen in Figures 3.1 to 3.3, is present in both server and clients even though its main role takes place in the server. This module's underlying role is to collect data from all network nodes and store it in the central node.

In order to improve this module from its original version, many changes occurred and the most structural ones are concerning the RPL implementation. The improved mapper will be able of using one of both Contiki RPL implementations: RPL-Lite and RPL-Classic.

The *IDIoT-6Mapper* is responsible to send, periodically, a mapping request to all available network nodes and collecting all mapping responses sent from these nodes. The information in the mapping request is the RPL instance ID, DAG ID, version number and a timestamp. This information is mostly necessary for authentication purposes from the *IDIoT-6Mapper-Server* to the *IDIoT-6Mapper-Client*. On the opposite side, the *IDIoT-6Mapper-Client* node sends all the required information for the detection modules, as follows: node ID, instance ID, DAG ID, version number, timestamp, the current rank of node and its parent ID, number of neighbors and for each neighbor, it's ID and rank. These packets are transferred over an UDP connection established between server and client nodes.

Upon reception of these packets, the *IDIoT-6Mapper-Server* stores all data regarding each network node. The exchanged timestamps have the purpose of letting the mapper know when to send new mapping requests, asking for updated data, by comparing the current timestamp of the mapper server with the timestamp of the last data received from a specific node. The timestamp variation for the *IDIoT-6Mapper-Server* to consider such timestamp as outdated relies on a configurable value which is defined at the reset moment or deployment of the device on the network. This value should be in accordance with the network's expected mobility and is an important parameter to configure with care with the penalty of sending too many or too few mapping request packets. For instance, if the network mobility ends up being much greater than expected, the *IDIoT-6Mapper-Server* module may end up sending the mapping requests at a slower rate then the necessary to keep the information updated and stable. On the other hand, if the network ends up being much less mobile than what was expected, the *IDIoT-6Mapper-Server* module may end up sending mapping requests in a higher rate then necessary, resulting in resource consumption's which could be avoided.

3.2.2 IDIoT-Routing-Detection

The *IDIoT-Routing-Detection* concerns the detection of routing attacks that manipulate the rank of one or more nodes in order to diverge or converge the network traffic flow with a malicious intention. This modules requires for the *IDIoT-6Mapper* to have collected network information concerning all neighbors relationships and ranks. This module makes use of a strong requirement implemented on the RPL protocol to detect such data manipulaton: the parent and child relation. The parent must always have an lower rank then the children and if's there's an inconsistency, the system will consider the child to be conflicting. This comparison is possible to the *IDIoT-Routing-Detection* provided that this module is in the possession of ranks collected by the *IDIoT-6Mapper* as well as the minimum rank increase determined by the RPL protocol.

Furthermore, node availability is a small module within this detection module, responsible for checking if nodes information is reaching the server and, if not, tagging these as offline nodes. There are many reasons for a node to be offline, maybe an attacker implementing a wormhole or a sinkhole, or just a faulty node which needs to be replaced. For this reason, the *IDIoT-Routing-Detection* should be able to notify the network administrator about this issue, or even to remove out this node from the entire network, avoiding the energy dissipation of other nodes trying to use this node as a valid route.

Upon detection of an attacker, Svelte's original work would try to correct the rank inconsistencies by updating the rank information with information from other nodes who reported valid information for this node. A new feature was added in order to improve the network quality when an attacker was present. The classification of the attacker is increased in one possible state, now becoming susceptible to a light correction and/or a severe correction. The ground for this approach was that not all attacks were successfully avoided with the rank correction approach, since the attacker kept on advertising fake ranks to all network reachable nodes, despite the server correcting these ranks, time after time. This resulted in a constantly unstable network that would only become stable if the attacker was successfully filtered or removed.

The severe correction now takes place when rank correction is not possible because all neighbors of the attacker were successfully poisoned by the spoofing of the attacker. Two actions are taken in this situation. Firstly, the node is marked as a severe attacker and the correction method triggers a method that reaches out to the firewall server module to globally report this node. A global filter command will be sent from the *IDIoT-Firewall-Server* to the entire network in order for the attacker to be filtered from all network nodes. Subsequently, the *IDIoT-Routing-Detection* launches the RPL global repair, a method owned and implemented by the RPL protocol which restarts the RPL formation. At this moment, the filtered attacker

may even be able to receive the initial DIO and DIS messages sent from the network nodes, however, the responses, which will once again carry a fake rank, will never reach any network nodes. For this reason, the node will successfully be filtered out from the entire network which will then become stable.

3.2.3 IDIoT Firewall System

Svelte proposes an interesting architecture of a *Mini-Firewall*, which we also took as the basis of our *IDIoT-Firewall*. This mini firewall could become a bigger contribution to our goal of providing more centralized and distributed security to all network nodes, but mainly providing more independence for distributed nodes to have a minimal security layer against easily detected attackers which can still cause considerable amounts of damage as energy consumption and DoS attacks to smaller nodes if these have no firewall features.

For this reason, the most significant extensions this project introduces to the firewall is: its necessary integration with the Contiki-NG and as well with RPL-Lite; distributed capability of all nodes to filter attackers on the TCP/IP layer, opposite to before when this operation was only performed by the border router; the capability of all nodes not being only able to filter existing malicious nodes using their IP address, but also using their link address, if both attacker and network node are neighbors, having a considerable reduction in energy consumption because the packets are filtered sooner on the network stack, which saves processing overhead; minor changes to these procedures to prevent missing cases and exploits of this system, some of them detected while studying the original Svelte.

When a node starts, the configuration of both *IDIoT-Firewall-Server* and *IDIoT-Firewall-Client* will happen due to the two processes running in the border router and the network nodes. These processes set up the filter rules, empty at this moment, and with a determined size by a given value which takes into account the memory allocation necessary for each filter. The *IDIoT-Firewall-Server* has a larger configuration process as well as a larger activity since all network firewall tasks are centralized in the *IDIoT-Firewall-Server*. Regular network nodes have the capability, in case of a detection module, detecting an attacker and send a filter request to the *IDIoT-Firewall-Server*. Furthermore, regular nodes are able to automatically filter the attacker locally in the distributed firewall. They check whether the attacker is a neighbor and, in accordance, applying the filter rule using its link-layer address. Otherwise, the global IP address is used instead. The *IDIoT-Firewall-Server*, not only has the capability of filtering nodes for itself, as explained before with the regular nodes but is also responsible for receiving all filtering requests from all network nodes, making thus the *IDIoT-Firewall-Server* responsible for the management of the active filters of the entire network.

There are two types of filters. The first is the small filter which represent a specific filter set for an individual attacker for an individual network node. These result from the previously mentioned filter requests sent from *IDIoT-Firewall-Clients* to the *IDIoT-Firewall-Server*. Secondly there are the global filters, which are specific filters applied to individual attackers to the entire network. These are applied for attacker which have proven to be abusive to two or more network nodes. When a filter is promoted from small to global, the *IDIoT-Firewall-Server* not only filters all packets from this attacker, just like introduced in Svelte, but also executes a procedure that sends an advertisement to all network nodes making it possible for these devices to filter the attacker locally. This capability is able to filter attackers inside the network, i.e., before the border router. This way, the attacker is completely filtered from the entire network. Moreover, the *IDIoT-Firewall-Server* is even able of triggering a procedure from the RPL protocol which is the RPL-global-repair. This procedure restarts the entire RPL topology construction and thus is a costly procedure concerning resource consumption because the entire network routing is restarting, however, it can have great benefits since the new network would completely exclude the attacker. All network nodes would even receive the attacker's control packets, like the DIO and DIS messages, as seen in RPL Section 2.1.4 but will never respond to this node since its now being blacklisted, resulting in a network routing where this attacker is not included and therefore a much more stable network.

As mentioned before, both *IDIoT-Firewall-Server* and *IDIoT-Firewall-Client* will have a blacklist of filters which can be filled with an attacker or empty if not necessary. This mechanism requires a nature of memory allocation, even if with a small impact since it only needs to save addresses, however, needs to be safely accounted because an over-sized number of possible filters would result in bigger memory allocation, and also in greater latency induced to every packet reception, because for every packet received the *IDIoT-Firewall* compares the sender address with all addresses saved in the filters, searching for a match.

3.2.4 IDIoT DoS Detection Module

The *IDIoT-DoS-Detection* was developed from scratch and its main goal is to provide IDIoT with the ability of detecting DoS attacks. These attacks can greatly affect the main purpose of these IoT small devices which is to provide communication or service, such as data collection and transmission, as well as greater energy consumption.

This module was developed with an action plan similar to other modules of the architecture, as this module is responsible for the detection of an attacker and after such, uses the procedures implemented by

the *IDIoT-Firewall* for an action such as alert/filter. This module is mainly present in the application layer despite having a minor change to the network layer, the TCP/IP, just like the *IDIoT-Firewall*. Its a module developed to run both in the border router and regular network nodes, allowing for all network nodes to be more resilient against a DoS attacker, without need to wait for an intervention of the border router which would never happen, in case of a more discrete and targeted DoS attack, or when the attack starts from a node inside the network.

The action flow of this method can be characterized as an anomaly-based, or even signature-based, considering the assumption that such would detect a well-known attack like the UDP flood, which behavior is known. This way, *IDIoT-DoS-Detection* executes a form of a counter which will be responsible to calculate the packet receiving rate for all packets received on the device it is running under a certain amount of time. If the packet receiving rate is higher than the configured threshold, different rule can be applied, such as blacklist filters. This module has a direct connection with the TCP/IP, where for each packet received, a call for the *IDIoT-DoS-Detection* is generated.

One of the requirements for the *IDIoT-DoS-Detection* was the capability of detecting an attack but also generating the minimum possible overhead in the network. On the other hand, it is possible to understand that if such module had a counter like the one explained before, running each time we received a packet and comparing such packet sender with all known addresses from neighbors or even exterior hosts, a huge overhead would be induced on the device since it would require memory to store all these addresses of possible attackers as well as generate a great amount of latency in the comparisons perform in each packet reception. This way, this module was developed to flow within three work states: Off, low alert, and high alert.

The off-state is only a transitional state, occurring at the reset moment of the device where the *IDIoT-DoS-Detection* is running and will run the configurations of the module before switching to the low alert state. The off-state was designed to run for only a few minutes after reset and it also has been designed this way in order to avoid extra energy consumption or even detect a false attack, having in consideration that the first minutes of a network deployment there is a much greater amount of packet exchange resulting from the network formation itself, where DIO, DIS, and DAO messages flow at a higher frequency during a short amount of time for the device to integrate the network. After this, the *IDIoT-DoS-Detection* increases its state to the low alert which has the main goal of being the most lasting state during the life-span of the node, where small comparisons are performed allowing for a smaller impact in latency but still being able to be an awake agent to detect any suspicious behavior and taking due precautions, namely increasing

the state to high alert. This way, the low alert state is responsible for counting the total amount of packets received, on the device it is running, for a certain period of time. For this reason, a periodic timer is used. At each timer overflow, the packet counter is reset to 0. For each packet received, the counter is increased and its value is compared to the established threshold which sets the limits for the number of packets to be received for that time interval. If at any moment the number of received packets surpasses the referred threshold, the mechanism switch from low alert to high alert state.

The high alert state is the one where the number of actions performed is much higher and thus where more latency is induced to the system, however, it is a state which was designed also not to be active for a great amount of time. Its main goal is to effectively detect which agent is causing this increased packet rate and to classify him as the cause, take actions in accordance, and return to the low alert state. This action is most likely to be a filtering order of the attacker and after such, the network traffic, at least the network traffic directed at this node should suffer a significant reduction and the *IDIoT-DoS-Detection* shall return naturally to the low alert state. This state workflow is very similar to the low alert state but now there are many increased actions. A finite number of detection agents are created, which are structures that store the IP address of a possible suspect as well as three packet counters assigned to this suspect. Each counter will be assigned to a protocol, such as the UDP, TCP and ICMP. In similarity to the low alert state, thresholds are configured which represent the maximum expected amount of packets to be received, in a certain amount of time, for a specific sender and protocol. If the received packets surpass any of these thresholds, the *IDIoT-DoS-Detection* is now detecting the attacker and the next step shall be to take actions in accordance. The *IDIoT-DoS-Detection* triggers a firewall procedure in order for this attacker to be locally filtered and reported to the border router running the *IDIoT-Firewall-Server*. As previously described, if any other network node's *IDIoT-DoS-Detection* sends a report to the *IDIoT-Firewall-Server* concerning this same attacker, then it will be promoted to a global filter and will be filtered out of the entire network, after the *IDIoT-Firewall-Server* command.

These referred thresholds have a great responsibility on the *IDIoT-DoS-Detection* and efforts are taken into making these the most representative of real network behavior, under penalty of having a detection module detecting false attacks and filtering innocent nodes, in a network where the traffic is just higher than previously expected, or under penalty of the *IDIoT-DoS-Detection* not being able to detect a possible attack being taken against the node where he is running, if the network actually is running with an inferior flow of traffic then previously expected.

Chapter 4

Implementation

This Chapter presents the implementation of the system discussed in Chapter 3. It starts with an overview given by Section 4.1, followed by a detailed implementation in Section 4.2, where all developed modules are explained and discussed.

4.1 Implementation Overview

The first approach to implement of the IDIoT solution consisted in studying the Contiki-2.6, which was used by the first release of the Svelte, and comparing it with the latest Contiki-NG. Svelte was firstly ported from Contiki-2.6 to Contiki-3.x, where the main difference lies in the RPL implementation. At last, and the most difficult porting exercise, was upgrading Svelte from Contiki-3.x to Contiki-NG, where several changes were performed: changes in the *Makefiles*, the declaration of the DAG root and its initialization, the use of the UDP connections, the *project-conf.h* files, and some other minor changes. At last, and probably the hardest task was to port the Svelte integration with RPL to support both RPL-Classic and RPL-Lite implementations, present in Contiki-NG.

Listing 4.1 shows an example of a *Makefile* suited for a UDP-server with the DAG-root, running the IDIoT modules. Some things remain from the old Contiki, for example, the need for inclusion of the top *Makefile*, the *Makefile.include*, and the definition of the root directory. However, this *Makefile* shows that for the inclusion of applications, formerly named as *APPS* in the previous Contiki versions, are now called *MODULES*. Also, in the previous Contiki versions, no routing protocol was specified because only the native version of RPL was available. By default, Contiki-NG uses the RPL-Lite, however, if the user intends in using the RPL-Classic, the *MAKE_ROUTING* define should be defined with *MAKE_ROUTING_RPL_CLASSIC*, as shown in line 12 in Listing , 4.1. Furthermore, the user can also define the desired Medium Access Control (MAC) protocol by defining the *MAKE_MAC*, e.g., *MAKE_MAC_CSMA*, as depicted in line 13 of Listing 4.2.

Listing 4.1: Contiki-NG Makefile for RPL-border-router running IDIoT modules.

```
CONTIKI = ../../..
CONTIKI_PROJECT = udp-server

include $(CONTIKI)/Makefile.dir-variables
# Include IDIoT modules
MODULES += $(CONTIKI_NG_SERVICES_DIR)/ids-server
MODULES += $(CONTIKI_NG_SERVICES_DIR)/firewall-server
MODULES += $(CONTIKI_NG_SERVICES_DIR)/dos-detector
# Include in the project all flags defined in project-conf
CFLAGS += -DPROJECT_CONF_PATH=\"project-conf.h\"

#MAKE_ROUTING = MAKE_ROUTING_RPL_CLASSIC
MAKE_MAC = MAKE_MAC_CSMA
all: $(CONTIKI_PROJECT)
include $(CONTIKI)/Makefile.include
```

Listing 4.2: Contiki-NG Makefile for UDP-client running IDIoT modules.

```
CONTIKI = ../../..
CONTIKI_PROJECT = udp-client

# Include all needed modules for this firmware.
include $(CONTIKI)/Makefile.dir-variables
MODULES += $(CONTIKI_NG_SERVICES_DIR)/ids-client
MODULES += $(CONTIKI_NG_SERVICES_DIR)/firewall-client
MODULES += $(CONTIKI_NG_SERVICES_DIR)/dos-detector
# Include in the project all flags defined in project-conf
CFLAGS += -DPROJECT_CONF_PATH=\"project-conf.h\"

#MAKE_ROUTING = MAKE_ROUTING_RPL_CLASSIC
MAKE_MAC = MAKE_MAC_CSMA
all: $(CONTIKI_PROJECT)
include $(CONTIKI)/Makefile.include
```

It is important to notice that all necessary modules, such as the *IDIoT-6Mapper-Server*, the *IDIoT-Firewall-Server*, and the *IDIoT-DoS-Detection*, must be included also in the *Makefile*. For instance, if considering the *Makefile* for a regular network node, such as an UDP-client, running the IDIoT as a client, a *Makefile* for the Contiki-NG using the RPL-Lite would look like the one presented in Listing 4.2. As seen in both *Makefiles*, a reference is given, in line 10, so that flags from the file *project-conf* header file would be included in the compiling flags for the project. This is a regular practice when building applications in Contiki as there are many possible configurations for the user regarding modules, variables size, and more. Listing 4.3 will show an example *project-conf* header file for a RPL-border-router running IDIoT modules.

As presented before in Section 2.3, Tmote Sky was the preferred mote for the development of the project, mainly because Svelte's implementation and evaluation was performed in this mote, for consistency and results comparison purposes. As this device is highly limited in resources, great efforts were taken during the implementation of saving ROM. For instance, as seen in Listing 4.3, in line 3, TCP stack was disabled in this project and therefore the support for such was turned of, allowing for a considerable amount of memory to be saved. Furthermore, in line 6, the amount of possible UDP connections is also limited to three due to an increasing amount of memory needed for each connection. Three connections were chosen as the maximum since this project only uses one for the UDP-server and UDP-clients packet exchanges, another for the *IDIoT-6Mapper-Server* and *IDIoT-6Mapper-Clients* exchanging the mapping requests and responses, and a last one for the *IDIoT-Firewall-Server* and *IDIoT-Firewall-Client* exchanges.

Moving on to the *project-conf.h*, shown in Listing 4.3, there is a flag, assigned to zero at this moment, which allows for the entire project code to know whether it is supposed to use the RPL-Lite or the RPL-Classic implementation. If the user intends to switch from one RPL implementation to another, one must set both variables: this last referred one, and the variable seen in the *Makefile*, in Listing 4.1. By the end of the project configuration header file, three flags are used for module enabling or disabling management. In Listing 4.3 it is better detailed how these flags turn of or of the use of modules.

Many other compiling flags can be set, enabled, or disabled in accordance with the user's intended application. For instance, it is possible to check the configuration header files of a specific platform building system, like the T mote Sky or others, to check all the possibilities of configuration, such as the: uIP buffer size, RPL maximum instances, RPL maximum DAGs per instance, RPL DAG lifetime and more.

Listing 4.3 shows an example of process definition for a specific application running the IDIoT modules. The two defined variables *CONF_FIREWALL* and *CONF_DOS_DETECTION*, in lines 21 and 22, are used to enable or disable the modules. Listing 5.1 shows how that is implemented.

Listing 4.3: Contiki-NG application project configuration header file example.

```
// Save some ROM by turning of the TCP stack.
#undef UIP_CONF_TCP
#define UIP_CONF_TCP              0

/* Defines the maximum amount of UDP connections available*/
#define UIP_CONF_UDP_CONNS 3

/* Define the max number of neighbors and routes allowed in each node's table */
#undef UIP_CONF_MAX_ROUTES
#define NBR_TABLE_CONF_MAX_NEIGHBORS 13
#define UIP_CONF_MAX_ROUTES 13
#define CSMA_CONF_MAX_NEIGHBOR_QUEUES 13
#define NETSTACK_MAX_ROUTE_ENTRIES 13

/* Turn of to use RPL-Classic. Turn off to 0 to use the default: RPL-Lite*/
#define RPL_CONF_CLASSIC    0

// Auxiliar flags for improved code organization
#define CONF_DAG_ROOT             1
// Auxiliar flag for easy on/off states of addittional modules
#define CONF_FIREWALL             1
#define CONF_DOS_DETECTION        1
```

Listing 4.4: Contiki-NG application processes start example.

```
#include "mapper-client.h"

#if CONF_FIREWALL
#include "firewall-client.h"
#endif /* CONF_FIREWALL */

#if CONF_DOS_DETECTION
#include "dos-detector.h"
#endif /* CONF_DOS_DETECTION */
/*---------------------------------------------------------*/
PROCESS(mapper_client_process, "UDP client process");
AUTOSTART_PROCESSES(&mapper_client_process, &mapper_client
#if CONF_FIREWALL
,&firewall_client
#endif /* CONF_FIREWALL */
#if CONF_DOS_DETECTION
,&dos_detector
#endif /* CONF_DOS_DETECTION */
);
```

4.2 IDIoT Modules

For the implementation of the IDIoT, we have developed (based on the original Svelte), the following modules: (1) *6Mapper*; (2) routing attack detection; and (3) Firewall. To these modules there were added several missing features that were not yet developed. Moreover, and in order to enable DoS attack detection, it was also created the DoS detection module, which is able to detect and stop UDP flood attacks.

4.2.1 IDIoT-6Mapper

Regarding the implementation of the *IDIoT-6Mapper*, the work of this project consisted mostly in upgrades and not so much in improvements. The upgrades refer to the changes performed to the Svelte original *6Mapper* for this to be compliant with the Contiki-NG and its two RPL implementations. This was by far the most effort demanding task concerning the *IDIoT-6Mapper*.

The *IDIoT-6Mapper* makes use of a simple UDP connection between server and client, using ports 4714 and 4713, respectively, for mapping requests and mapping response communications. These control packets have already been referred in Sections 3.2.1 and 2.2.2.1. Figure 4.1 presents the structure of both mapping request and mapping response exchanged variables and their size in bytes. Algorithm 5 shows the procedure which periodically sends these control packets to the network nodes.

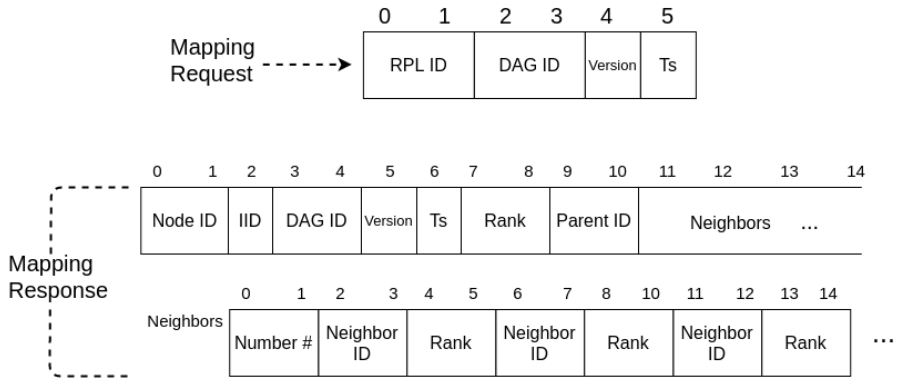


Figure 4.1: Svelte improved mapper.

In this implementation, the *IDIoT-6Mapper-Server* makes use of two timers for the management of mapping request send interval. Both timers are from the *etimer* Contiki library and the first, *map_timer*, overflows at a larger amount of time than the other, the *host_timer*. Firstly, the *map_timer* is responsible for generating the event which will start the entire process of network mapping. This process is periodic and the mapping interval must be set accordingly with the network mobility, as said in the previous system

Algorithm 5: *IDIoT-6Mapper-Server* Algorithm for network construction.

```
Input: N - List of the Network Nodes
Input: rpl_info - The RPL info of this node
Input: Timestamp - Timestamp for this node at given moment
Input: host_timer
Input: map_timer
Output: mapping_request
if map_timer.expired then
    if host_timer.expired then
        for network_node in N do
            if network_node.timestamp == outdated then
                mapping_request.instance_ID = rpl_info.instance_ID ;
                mapping_request.dag_ID = rpl_info.dag_ID ;
                mapping_request.version_number = rpl_info.version_number ;
                mapping_request.timestamp = timestamp ;
                simple_udp_send( network_node.ipaddress , mapping_request );
            end
            host_timer.restart ;
            return ;
        end
        map_timer.restart ;
        return ;
    end
end
```

design Section 3.2.1. When this timer expires, the *IDIoT-6Mapper-Server* proceeds on sending mapping requests to all network nodes and then restarts the *map_timer* and repeats the process. Sending mapping requests from one node to another is also a periodical procedure executed at the expiring event of the *host_timer*. A simple configuration is set: the amount of time of which *host_timer* runs before overflow is equal to the *map_timer* divided by the number of network nodes. For instance, if the network administrator sets the mapping interval, the *map_timer* value, of 120, and there are 6 *UDP-clients* in the network, the *RPL-border-router* will send mapping request to each node, one at a time, every 20 seconds.

Algorithm 6: *IDIoT-6Mapper-Client* Algorithm for handling mapping requests and responses.

```
Input: rpl_info - RPL info of this node
Input: Nbr - List of neighbors to this node
Input: mapping_request
Output: mapping_response
if mapping_request.instance_ID == rpl_info.instance_ID then
    if mapping_request.dag_ID == rpl_info.dag_ID then
        if mapping_request.version_number == rpl_info.version_number then
            mapping_response.node_ID = rpl_info.my_ID ;
            mapping_response.instance_ID = rpl_info.instance_ID ;
            mapping_response.dag_ID = rpl_info.dag_ID ;
            mapping_response.version_number = rpl_info.version_number ;
            mapping_response.timestamp = current_timestamp ;
            mapping_response.node_rank = rpl_info.my_rank ;
            mapping_response.parent_ID = rpl_info.myparent_ID ;
            for neighbor in Nbr do
                mapping_response.neighbors_IDs « neighbor.ID ;
                mapping_response.neighbors_ranks « neighbor.rank;
            end
            simple_udp_send(mappin_response.sender_ip, mapping_response);
        end
    end
end
```

Each timer from the *etimer* library is only present in the *IDIoT-6Mapper-Server* as the *IDIoT-6Mapper-Client* only sends mapping responses after the reception of a mapping request, thus, timers are not required. The client's behavior when receiving a mapping request is described in Algorithm 6.

At last, the algoritm 7 shows *IDIoT-6Mapper-Server* procedure in receiving a mapping response. The *IDIoT-6Mapper-Server* increasingly builds a topology of the network packet after packet, adding new nodes

to the internally constructed network as the nodes refer to such node as a neighbor. With this information is then possible to perform the detection algorithms present in the following section.

Notice that in Algorithms 6 and 5, the instance ID, DAG ID, and version number are always checked for safety concerns. This procedure happens for both RPL-Lite and RPL-Classic implementation even tough the RPL-Lite only has room for one instance and one dag. This procedure is mandatory since a node from another network, belonging to another DAG may intercept a mapping request and try to answer to such, which would create information mismatches in the *IDIoT-6Mapper-Server*. As expected, the RPL-Lite implementation induces little overhead when compared to RPL-Classic, mainly due to the existence of only one RPL instance and only one dag for such instance. This way, the RPL-Lite is the preferred RPL implementation when considering a solution that will be building a network topology of its own.

Algorithm 7: *IDIoT-6Mapper-Server* Algorithm for managing the network mapping.

```
Input: N - List of the Network Nodes
Input: rpl_info - The RPL info of this node
Input: mapping_response
if mapping_response.instance_ID == rpl_info.instance_ID then
    if mapping_response.dag_ID == rpl_info.dag_ID then
        if mapping_response.version_number == rpl_info.version_number then
            N « mapping_response.node_ID ;
            N « mapping_response.parent_ID ;
            for network_node in N do
                if network_node.node_ID == mapping_response.node_ID then
                    network_node.node.parent = mapping_response.parent_ID ;
                end
            end
            for neighbor in mapping_response.neighbors do
                N « neighbor_ID ;
                N « neighbor_rank ;
            end
        end
    end
end
```

4.2.2 IDIoT-Routing-Detection

In the similarity of the previous module, the *IDIoT-Routing-Detection* has also suffered several upgrades and improvements. However, contrary to the previous module, upgrading the detection module was the smallest task. This is because almost all of the procedures performed by the *IDIoT-Routing-Detection* are performed over the collected data from the *IDIoT-6Mapper* and therefore do not handle directly with structured RPL data. For this reason, the effort performed in *IDIoT-6Mapper* to upgrade the module to work with both RPL versions resulted in only minor changes for upgrading this module. However, improving the module was the most effort demanding task.

The Algorithms 9 and 10 will show some of the improvements performed in this solution, over the existing work presented by Svelte. For instance, Algorithm 8 is very much like the one presented by Svelte since the child and parent relation has not suffered changes. It was a relation specific to the RPL protocol as published in the standard [37].

Algorithm 8: *IDIoT-Routing-Detection* checking child-parent relation for rank inconsistencies.

```
Input: N - List of the Network Nodes
for Node in N do
    if Node.timestamp == outdated then
        continue ;
    else
        for Node in N do
            if Node.rank + 0.2*Node.rank < Node.parent.rank + MinHopRankIncrease then
                Node.fault = Node.fault + 1 ;
                Node.parent.fault = Node.parent.fault + 1 ;
            end
        end
        for Node in N do
            if Node.fault > FaultThreshold then
                Raise alarm ;
            end
        end
    end
end
```

The Algorithm 9 presents the detection and correction algorithm of possible attacks or information mismatches. Algorithm 9 shows one major improvement im comparison to the original Svelte algorithm

for the same procedure. In the Svelte's version, it was not considered the existence of a severe attack such as a sinkhole that can be so harmfully to the network that cannot be simply counterattacked with the Svelte's original correction. For instance, when a sinkhole is so harmful that can imply in all of its neighbor a spoofed rank, there is no chance for the Svelte's rank correction procedure to work. Because Svelte would be replacing the spoofed rank of the attacker with information of one of its neighbors. However, all that neighbor's information is compromised since it was in direct contact with the sinkhole attacker. Svelte would be stuck in a loop replacing bad information of an attacker with information expected to be secure but was, in fact, as bad as the original. This would result in the attacker running the attack freely.

For this reason, IDIoT biggest improvement to the detection and correction module is acknowledging that there may be times were an attacker shall be marked as a severe attacker if, after such attacker was identified, and accordingly be filtered from the entire network as a global filter, proceeded by an *RPL-global-repair* and *IDIoT-6Mapper*. An attacker will be identified as severe if, when trying to replace its faulty information with the information of other nodes which have him as a neighbor, *IDIoT-Routing-Detection* determines that all of the nodes which have the attacker as a neighbor claim that such attacker has a rank that does not fulfill the relation presented in Algorithm 9: Rank reported for the attacker by any neighbor < *Attacker.parent.rank* + *MinHopRankIncrease*. At last, Algorithm 10 shows the procedure for when a severe attack is raised.

Algorithm 9: Detecting and correcting RPL DODAG Inconsistencies.

```
Input: N - List of the Network Nodes
for Node in N do
    if Node.timestamp == outdated then
        continue ;
    else
        for Neighbor in Node.neighbor do
            Diff = |Node.neighbor.rank - Neighbor.rank| ;
            Avg = (Node.neighbor.rank + Neighbor.rank)/2 ;
            if Diff > Avg * 0.2 then
                Node.fault = Node.fault + 1 ;
                Neighbor.fault = Neighbor.fault + 1 ;
            end
        end
    end
end
for Node in N do
    if Node.timestamp == outdated then
        continue ;
    else
        if Node.fault > FaultThreshold then
            for Node in N do
                for Neighbor in Node.neighbors do
                    if Rank reported for Node by any neighbor > Node.parent.rank +
                     MinHopRankIncrease then
                        Remove severe alarm ;
                        Node.rank = Rank reported for Node by such neighbor ;
                        for Neighbor in Node.neighbors do
                            Node.neighbor.rank = Neighbor.rank ;
                        end
                    else
                        Raise severe alarm ;
                    end
                end
                if Severe alarm raised then
                    perform mapper global repair ;
                end
            end
        end
    end
end
```

Algorithm 10: Correcting severe attacks with mapper and RPL global-repair.

```
Input: N - List of the Network Nodes
Input: attacker - node reported by detection module
if Severe alarm raised then
    if attacker == dag root then
        do nothing ;
    else
        firewall.add_global_filter( attacker ) ;
        firewall.broadcast_global_filter ( attacker ) ;
        perform rpl_global_repair ;
        perform mapper global repair ;
    end
end
```

4.2.3 IDIoT-Firewall

The firewall used in the IDIoT shares its architecture with Svelte, as explained previously in system design, but is greatly improved and extended. The path for this implementation was to firstly extend the existing firewall to the regular network nodes, by deploying a very similar but more limited firewall from the one already existing in Svelte at the firewall server. Secondly, by improving the firewall in both devices adding new features. In contrast to the *IDIoT-6Mapper*, there were not a significant need of upgrading the firewall to the Contiki-NG or the new RPL implementations since the firewall itself is only composed of actions triggered by the detection modules of the IDIoT.

The native version of this firewall had already established communication between the firewall server and firewall client, however, the firewall client's actions were limited to sending report requests to the firewall server. At this moment, the *IDIoT-Firewall-Client* is now capable of filtering nodes itself, locally, and even manage those local filters. For this reason, the communications between *IDIoT-Firewall-Server* and *IDIoT-Firewall-Client* were accordingly extended as both control packets, filter requests, and filter commands structure and size, in bytes, can be seen in Figure 4.2.

As covered in Section 3.2.3 regarding the *IDIoT-Firewall*, the filter request is a control packet sent from the *IDIoT-Firewall-Client* of a regular node to the *IDIoT-Firewall-Server* of the DAG-root and includes an IP address corresponding to an external host that is intended to be filtered and excluded from the network. This is now greatly extended since the network regular nodes are running the *IDIoT-DoS-Detection*, and therefore can generate these report requests accordingly. The filtering command is an order to filter sent by the *IDIoT-Firewall-Server* to the *IDIoT-Firewall-Client*. These control packets are exchanged using an

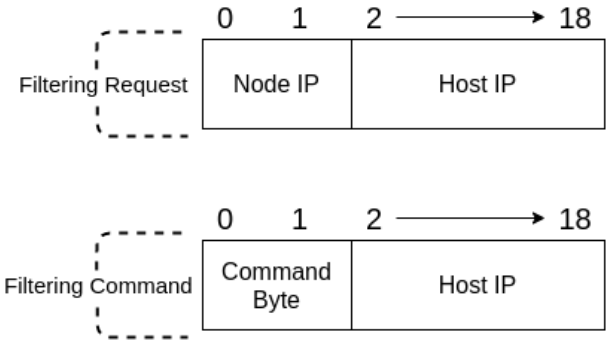


Figure 4.2: *IDIoT-Firewall* exchanged packets.

simple UDP connection between server and clients, at ports 4801 and 4800, respectively.

The Algorithm 11 shows how the *IDIoT-Firewall-Client* manages the local filters available in the *IDIoT-Firewall-Client* of such node whenever receives a filtering command from the *IDIoT-Firewall-Server*, or simply receives a filter request from the *IDIoT-DoS-Detection* running inside the node itself.

Furthermore, Algorithm 11 shows how the *IDIoT-Firewall-Client* manages the capability to filter the attacker in the MAC layer if one finds itself to be neighbor with the attacker. This is also a feature enabled by the *IDIoT-Firewall*. Whether or not the attacker is a neighbor, the attacker is also given a local filter where the IP address is stored. Only then the *IDIoT-Firewall-Client* checks if the attacker is a direct neighbor, applies a MAC address filter instead. This is the implemented methodology to prevent the attacker from being physically moved within the network range and becoming a non-neighbor to the node under attack. If this happened, and the node under attack had not set a local filter to the attacker through his IP address, the attacker would be able to engage with the complaint once again.

Shifting attention to the *IDIoT-Firewall-Server* running in the DAG-root, many features are also deployed in accordance with the ones deployed in the *IDIoT-Firewall-Client*. In Algorithm 13 it is clear that in the *IDIoT-Firewall-Server* there are small and global filters, and how the promotion from a small filter to a global filter happens. This as also been explained in the previous system design Section 3.2.3. Notice that when this happens we set the state to that existing small filter as unactive, since there will now be a global filter blocking this attacker, there is no need for this small filter to be active since every active filter is a latency overhead when receiving a packet. Moreover, as explained before, when a filter becomes a global filter, we extended the *IDIoT-Firewall-Server* to send a broadcast command to all network nodes. This takes place in Algorithm 13 as well, and the procedure itself is seen in Algorithm 12.

Algorithm 11: Firewall client local filter management.

```
Input: sender & attacker
Input: LF - A list of Local Filters
Input: localfilter_index
Input: LLF - A list of Local Link Filters
Input: locallinkfilter_index
Input: Nbrs - A list of neighbors to this node
/*Got a new message of sender asking to filter packets from attacker*/;
for localfilter in LF do
    if localfiler.state == active then
        if localfilter.ipaddress == attacker.ipaddress then
            return /*Already existing filter*/
        end
    end
end
LF[localfilter_index].ipaddress = attacker.ipaddress ;
LF[localfilter_index].state = active ;
localfilter_index = localfilter_index + 1 ;
for neighbor in Nbrs do
    if attacker.ipaddress == neighbor.ipaddress then
        for locallinkfilter in LLF do
            if locallinkfilter.state == active then
                if locallinkfilter.linkaddres == attacker.linkaddress then
                    return /*Already existing filter*/
                end
            end
        end
    end
end
LLF[locallinkfilter_index].linkaddress = attacker.linkaddress ;
LLF[locallinkfilter_index].state = active ;
locallinkfilter_index = locallinkfilter_index + 1 ;
```

In Algorithm 11, the procedure for MAC address filtering management of the *IDIoT-Firewall-Client* was shown. Due to algorithm size concerns, the MAC address filtering management for the *IDIoT-Firewall-Server* was not included in Algorithm 13, however, its behavior is the same. There are no MAC address filters addressed to the network, in contrary with the small and global filters in the *IDIoT-Firewall-Server*, they always address attacker to a specific node since they rely on the neighbor condition. For this reason, the *IDIoT-Firewall-Server* performs the same as the *IDIoT-Firewall-Client* regarding MAC address filters: after blocking locally an uIP address of a given attacker, the *IDIoT-Firewall* checks if such attacker is a neighbor and if so, proceeds to create a MAC address filters for such.

The last algorithm for the *IDIoT-Firewall* is Algorithm 12 showing the procedure for the filter command broadcast. This is performed by the firewall using information collected from the *IDIoT-6Mapper*, the list of network nodes. A simple packet, as seen in Figure 4.2 is generated and sent to all network nodes. The first two bytes of the packet is a command byte which is known only to the server and the clients in order to a client be able to ensure that such command came from the *IDIoT-Firewall-Server*. Also, when a *IDIoT-Firewall-Client* receives the filter command, a comparison between the filter command sender IP address, provided by the simple UDP connection, with the stored *DAG-root*, takes place.

Algorithm 12: Firewall server broadcast filter command.

```
Input: sender & attacker
Input: N - List of the Network Nodes
for Node in N do
    if Node != sender then
        if Node != attacker then
            simple_udp_send(Node.ipaddress , filter_command , attacker.ipaddress) ;
        end
    end
end
```

Algorithm 13: Firewall client network filters management.

```
Input: sender & attacker
Input: GF - A list of Global Filters & GF_index
Input: SF - A list of Small Filters & SF_index
/*Got a new message of sender asking to filter packets from attacker*/;
for Globalfilter in GF do
    if Globalfiler.state == active then
        if Globalfilter.ipaddress == attacker.ipaddress then
            return /*Already existing filter*/
        end
    end
end
for Smallfilter in SF do
    if Smallfilter.state == active then
        if Smallfilter.ipaddress == attacker.ipaddress then
            if Smallfilter.dest == sender then
                return /* Already existing filter for this destination */
            end
            /*New complaint of an existing filter. Promoting to global*/ ;
            GF[GF_index].ipaddress = attacker.ipaddress ;
            GF[GF_index].state = active ;
            GF_index = GF_index + 1 ;
            Smallfilter.state == unactive ;
            broadcast global filter ;
            return ;
        end
    end
end
for Smallfilter in SF do
    if Smallfilter.state == unactive then
        Smallfilter.ipaddress = attacker.ipaddress ;
        Smallfilter.state = active ;
        Smallfilter.dest = sender ;
        return ;
    end
end
/* All small filters are active, so replace with the oldest */ ;
SF[SF_index].ipaddress = attacker.ipaddress ;
SF[SF_index].dest = sender ;
SF_index = SF_index + 1 ;
```

4.2.4 DoS Detection Module

Concerning the implementation of the *IDIoT-DoS-Detection*, two algorithms will be shown for a better understanding of what was performed. The first, Algorithm 14 shows the procedure that takes place every time a node, whose *IDIoT-DoS-Detection* is enabled, receives a valid packet that reaches the networking layer of the uIP stack. At this point, the TCP/IP performs a link to this procedure so it may be executed. If a node has the *IDIoT-DoS-Detection* disabled, this call will not take place and the following procedures will not take place as well. Algorithm 15 shows the procedure for each timer overflow. This module makes use of a timer, similarly to the ones used to by the *IDIoT-6Mapper* as previously explained, and the moment this timer expires, an event is triggered allowing for this procedure to execute.

As explained in the previous system design of this module, Section 3.2.4, the *IDIoT-DoS-Detection* uses three states: *OFF*, *LOW ALERT*, *HIGH ALERT*. In *OFF* state no data structures are used, only an auxiliary variable cooldown used for asserting the state from off to low only after a configurable amount of time. In both the *LOW ALERT* and the *HIGH ALERT* state, a counter is used to keep track of the total amount of received packets for a specific amount of time and is represented in Algorithm 14 as for *packet_counter*. At last, the *HIGH ALERT* states makes use of a list of agents. These agents are called detecting agents and are implemented in a structure with the attacker uIP address, three counters, one for each protocol, respectively UDP, TCP and ICMP, and, at last, a state variable. There were two main concerns regarding the detecting agents: how detecting agents should be allocated, and how which selection process should be used when considering who will be assigned for the detecting agent to keep track. Firstly, the amount of allocated detecting agents will have an impact on memory allocation and an impact on latency as well since, at each packet received, if in the *HIGH ALERT* state, the comparison of the packet sender and the list of detecting agents takes place. The longer the list, the higher the latency for each received packet. Secondly, the selection process when considering which node will be assigned for such a detecting agent is also a concern, because the system relies on at least of the detecting agents being effectively allocated to an attacker. If all detecting agents are assigned to track regular network nodes, and the attacker is not being tracked, the system will never leave the *HIGH ALERT* state, causing even higher deniability of service for such node, the opposite of what is intended with this module.

Algorithm 14: DoS Detection module at each packet reception.

```
Input: packet.sender ip address
Input: State_Threshold & udp_threshold
Input: A - A list of Agents
switch State do
    case Off do
        Do nothing ;
    end
    case Low Alert do
        packet_counter = packet_counter + 1;
        if packet_counter > State_Threshold then
            state = High_Alert_State;
        end
    end
    case High Alert do
        packet_counter = packet_counter + 1;
        for agent in A do
            if packet.sender == agent.ipaddress then
                agent.udp_counter = agent.udp_counter + 1;
                if agent.udp_counter > udp_threshold then
                    report attacker;
                end
                return;
            end
        end
        /*If no agent is assigned for the sender of this packet, try to assign one*/;
        for agent in A do
            if agent.state == 0 then
                agent.ipaddress = packet.sender;
                agent.state = 1;
                agent.udp_counter = agent.udp_counter + 1;
                return;
            end
        end
    end
end
```

After some speculation and simulations, we found that the most reasonable amount of detecting agents for this system would be dynamic and with the value of a third of all network nodes. It is not expected that so many nodes can be compromised all at the same time and even if so, a third of the network nodes

is a reliable value for allowing the system to detect attackers, even if only one at a time. Moreover, and regarding the selection process, the conclusions taken from simulations with a regular network where one node was receiving a DoS attack were that such node would always receive a packet from such attacker at a higher packet rate than any of node, making thus possible for a selection process to work by, after the *IDIoT-DoS-Detection* reaching the *HIGH ALERT*, and given, e.g., five as the amount of detecting agents, the first five nodes that would send a packet to this node, would be the ones selected to be tracked with a detecting agent. Empirically, this was the number found to be ideal.

In Algorithm 14, and in case of *HIGH ALERT* active, it is shown only one counter and one logic comparison using the UDP counter for such detecting agent. This procedure is implemented this way since, at this moment, only support for UDP packet counting is enabled. Support for other protocols is left for future work. Regardless of the protocol implementation, an attacker will be detected when the number of packets counted exceeds a previously defined threshold. This threshold can be configured by the network administrator and in this implementation, we used a threshold in accordance with the maximum expected traffic flow for one node within a regular network of thirteen nodes, in which one is an UDP-server and the remaining are UDP-clients. The threshold values were, for this reason, obtained through simulation of this network delimiting what could be considered normal traffic and from what point it could no longer be considered normal.

At last, and concerning the timer referred in Algorithm 15, the implementation of the *IDIoT-DoS-Detection* makes use of a timer to generate expiring events in order to trigger the procedure shown in Algorithm 14. This timer value is configurable by the network administrator at the moment of deployment, such as the mapping interval used by the *IDIoT-6Mapper*. If deploying this solution in a greatly resource-limited node, one of the possibilities in the IDIoT is to set this timer value to be the same as the *IDIoT-6Mapper* mapping interval timer and the system will make use of the *IDIoT-6Mapper* timer to generate these events as well.

Algorithm 15: DoS Detection module at timer overflow.

```
Input: etimer expired event
Input: State_Threshold
Input: A - A list of Agents
switch State do
    case Off do
        if cooldown == 0 then
            state = Low_Alert_State;
        else
            etimer_restart;
        end
    end
    case Low Alert do
        packet_counter = 0;
        etimer_restart;
    end
    case High Alert do
        if packet.sender < State_Threshold then
            state = Low_Alert_State;
        end
        packet_counter = 0;
        memset(agents,0);
        etimer_restart;
    end
end
```

Chapter 5

Evaluation

This chapter will focus on evaluating our solution regarding its ability to accomplish what was proposed in previous chapters. It presents functional tests, performance evaluations and system characterizations, e.g., memory footprint and energy consumption.

5.1 Evaluation Overview

Adding new features to a stable version of firmware aiming to improve certain capabilities of the targeted device is a bold intention in the research community. For this reason, one cannot simply expect community interest or even approval for its work if detailed evaluations are not presented. Therefore, it's widely established that some evaluations and benchmarks shall be performed and presented for any kind of feature developed with the aim of deployment in stable devices. Our solution slightly changes the behavior or a device within a network, regarding its routing behavior and its communications activity. For this reason, metrics for this evaluation should be: the device's capability of detecting, reporting and filtering an attack, the device's overhead when exchanging communications, and the device's increased footprint in memory.

The device's capability of detecting, reporting, and filtering an attack is not a quantifiable metric in contrary with, for example, the devices increased memory footprint, measured in kB. For this reason, to evaluate this metric, then, a finite number of simulation samples will be used, where in each sample a different simulation test scenario is used.

Simulations performed will be available to the research community in an open-source repository and the structure is as follows: For the device's capability of detecting, reporting, and filtering an attack, we have three test scenarios of networks, and we will run each test scenario twice. First time without the IDS modules and secondly with the IDS modules. These three test scenarios will be (1) network without

attacks, (2) network under a sinkhole attack from inside the network, (3) under a DoS attack from inside the network. All test scenarios run the same base structure which is one UDP-server acting as DAG-root, running IDS-server modules when IDS is active, and many UDP-clients, also running IDS modules if such is active.

Regarding the device's overhead when exchanging communications, which is a quantifiable metric, many factors can be measured and for this reason, we will be using an excellent tool which has proven to be suited or these types of works. The Thread-Metrics Benchmark Suite allows researchers and developers to access the amount of overhead introduced by an RTOS to determine if its services are worth the additional performance cost [71]. It consists of creating eight OS processes that output a score value denoting the system's availability. A higher score represents more system availability to execute the application rather than OS services. On the other hand, a lower score denotes less availability as the OS is busy processing other services/events.

From the Thread Metrics Benchmark Suite, we will be benchmarking our system with the following tests: (1) Basic Processing Test, (2) Cooperative Scheduling Test, (3) Preemptive Scheduling Test, (6) Message Processing Test, (7) Synchronization Processing Test, and the (8) RTOS Memory Allocation test. All tests will output a score that will be presented in following sections. Both Interrupt Processing Test and Interrupt Preemption Processing Test require for some processor-specific instructions which are not available for the processor being used in Tmote Sky, the MSP430, and for this reason, these will not be performed. We will be running four additional test scenarios to measure the overhead induced by our solution. This test scenario is just like the aforementioned, with the difference that, in all four of these tests, the UDP-server will be running the Thread Metrics tests, each at a time. The score from these tests will be shown for comparison.

Cooja Network Simulator will be used for all simulations, as discussed in Section 2.3.

5.2 IDIoT Routing Detection

In order to evaluate the IDIoT Routing Detection functionality, a sinkhole attack is launched within the network. In Figure 5.1 it is possible to see the printed route links from RPL-Lite after one hour of simulation. The network topology used for this simulation is shown in Figure 5.2 as well as the routing information collected after 1 hour of runtime, represented in arrows, from a parent to child direction. This information shows which are the current routes for each node to the network root. This routing table is obtained from

this network while using the RPL-Lite protocol, however using the RPL-Classic the results are very similar, due to the fact that the attack is not active, so no interference is present to affect routes from the typical flow.

```
1:11:20.113  ID:1   Routes [13 max]
1:11:20.113  ID:1   Routing links:
1:11:20.115  ID:1   Link:  fd00::c30c:0:0:3   parent:  fd00::c30c:0:0:1) 480s
1:11:20.122  ID:1   Link:  fd00::c30c:0:0:d   parent:  fd00::c30c:0:0:3) 420s
1:11:20.128  ID:1   Link:  fd00::c30c:0:0:6   parent:  fd00::c30c:0:0:3) 420s
1:11:20.135  ID:1   Link:  fd00::c30c:0:0:2   parent:  fd00::c30c:0:0:1) 480s
1:11:20.141  ID:1   Link:  fd00::c30c:0:0:4   parent:  fd00::c30c:0:0:3) 480s
1:11:20.147  ID:1   Link:  fd00::c30c:0:0:7   parent:  fd00::c30c:0:0:3) 540s
1:11:20.154  ID:1   Link:  fd00::c30c:0:0:a   parent:  fd00::c30c:0:0:6) 480s
1:11:20.160  ID:1   Link:  fd00::c30c:0:0:5   parent:  fd00::c30c:0:0:2) 360s
1:11:20.167  ID:1   Link:  fd00::c30c:0:0:8   parent:  fd00::c30c:0:0:4) 420s
1:11:20.173  ID:1   Link:  fd00::c30c:0:0:b   parent:  fd00::c30c:0:0:7) 420s
1:11:20.180  ID:1   Link:  fd00::c30c:0:0:9   parent:  fd00::c30c:0:0:5) 480s
1:11:20.186  ID:1   Link:  fd00::c30c:0:0:c   parent:  fd00::c30c:0:0:8) 480s
```

Figure 5.1: Clean routes in network under no attacks.

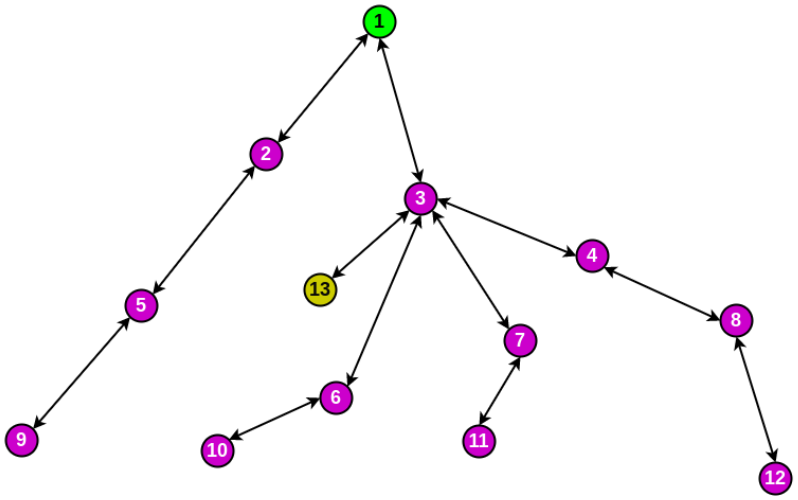


Figure 5.2: Topology used for the evaluation of the sinkhole attack detection.

Regarding the attack implementation, one of the deployed UDP-clients will be modified in order to run a sinkhole attack. This node, represented in simulations by number 13, behaves as a regular UDP-client, just like the remaining, however with a slightly modified behavior regarding RPL specifications. These modifications are the rank it is advertising in all of his DIO messages sent to neighbors, as well as the frequency at which RPL trickle timer is operating. In order to perform the sinkhole attack, we have modified his rank advertisement to be as high as the root rank and the frequency of the trickle timer to be as high

as possible, resulting in a higher frequency of DIO being sent to neighbors so these always have this node in high stakes in their routing tables.

The same topology from Figure 5.2 was repeated, however, this time, the attacker was enabled. Once again, Figure 5.3 shows the printed route links, stored in the DAG-root, node 1, after 1 hour of simulation. It is now possible to notice that are many links headed by node 13, whereas in previous Figure 5.1 there was none. In Figure 5.4 it is possible to see to what extent the sinkhole can effect the routing tables of the network. Routes are shown in Figure 5.4 were collected after 1 hour of simulation. Most neighbors of node 13 have redirected their routing tables through him since he his advertising a much more pleasant rank.

```
1:00:31.614  ID:1   Routes [13 max]
1:00:31.614  ID:1   Routing links:
1:00:31.620  ID:1   Link:  fd00::c30c:0:0:3   parent:  fd00::c30c:0:0:1) 600s
1:00:31.627  ID:1   Link:  fd00::c30c:0:0:d   parent:  fd00::c30c:0:0:3) 120s
1:00:31.633  ID:1   Link:  fd00::c30c:0:0:6   parent:  fd00::c30c:0:0:d) 120s
1:00:31.640  ID:1   Link:  fd00::c30c:0:0:2   parent:  fd00::c30c:0:0:1) 120s
1:00:31.646  ID:1   Link:  fd00::c30c:0:0:4   parent:  fd00::c30c:0:0:3) 600s
1:00:31.652  ID:1   Link:  fd00::c30c:0:0:7   parent:  fd00::c30c:0:0:d) 300s
1:00:31.659  ID:1   Link:  fd00::c30c:0:0:5   parent:  fd00::c30c:0:0:d) 120s
1:00:31.665  ID:1   Link:  fd00::c30c:0:0:a   parent:  fd00::c30c:0:0:d) 600s
1:00:31.672  ID:1   Link:  fd00::c30c:0:0:b   parent:  fd00::c30c:0:0:d) 120s
1:00:31.678  ID:1   Link:  fd00::c30c:0:0:8   parent:  fd00::c30c:0:0:7) 180s
1:00:31.685  ID:1   Link:  fd00::c30c:0:0:c   parent:  fd00::c30c:0:0:8) 180s
1:00:31.691  ID:1   Link:  fd00::c30c:0:0:9   parent:  fd00::c30c:0:0:5) 180s
```

Figure 5.3: Modified Routes in network under sinkhole attack.

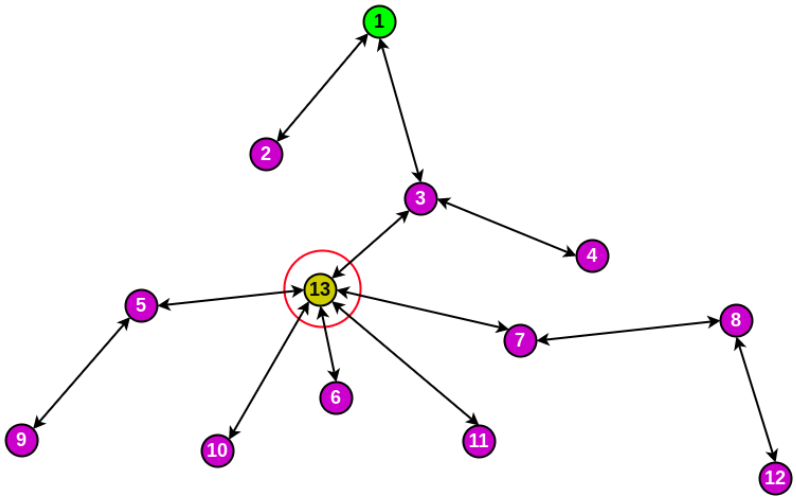


Figure 5.4: Modified topology caused by a sinkhole attack.

After the first data collection from the *IDIoT-6Mapper*, the detection and correction module starts running and analyzing the collected data. At this configuration, the minimum hop rank increase is 128, which is also the rank of the DAG-root. For this reason, node 13 should be reporting at least a rank of 256 in order to be in accordance with the parent-child relationship. Furthermore, in simulations where the maximum transmission distance is fewer than 70 meters, node 13 is not a neighbor of node 1, making it mandatory for him to be a child of either node 2 or 3. This should result in a rank of at least 3 times the minimum hop rank increase: 384. However, node 13 his being said, by all of its neighbors, to have a rank value of only 130. The attacker is detected and since the light rank correction is not possible because all neighbors are contaminated with false rank information, the attacker is marked as severe. For this reason, the detection and correction module triggers the firewall server to globally filter with the attacker by sending filter commands to all network nodes. Furthermore, the RPL-global-repair procedure is triggered, which re-starts the entire RPL formation, and, at last, *IDIoT-6Mapper* performs a global repair of his own collected data. Figure 5.5 shows part of the Cooja mote output window from *IDIoT-6Mapper-Server* information where it is possible to see the attacker marked as a global filter and aside from the remaining network. It also shows the re-constructed *IDIoT-6Mapper* network graph after the global repair process. It is possible to check, for all network nodes present, none is claiming to have node 13 as neighbor, as this is completely filtered out from the network. Moreover, and for a better visualization, Figure 5.6 shows a final instance where all routing information shows how the attacker was successfully globally filtered from the network. At last, it is possible to see in Figure 5.7 the same procedure of route links being printed, as seen before, after one hour of simulation and, ultimately, node 13 has vanished.

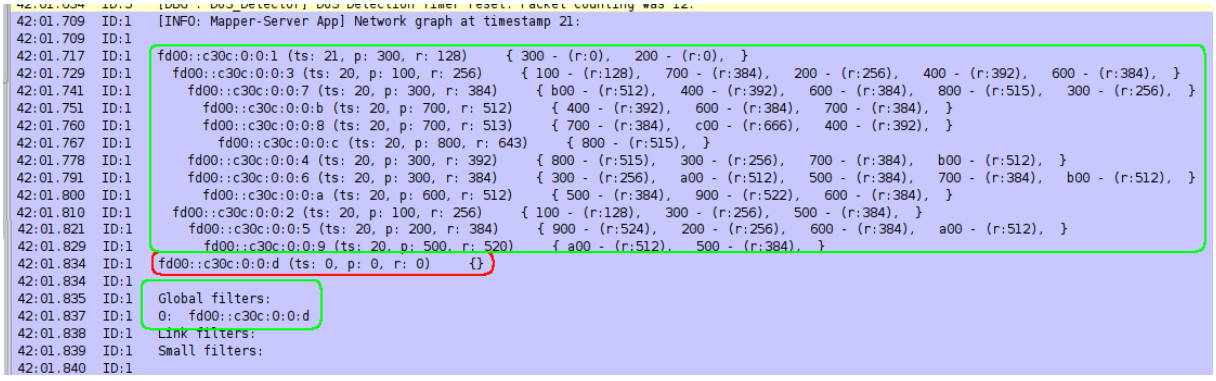


Figure 5.5: *IDIoT-6Mapper* network graph recovered after global repair.

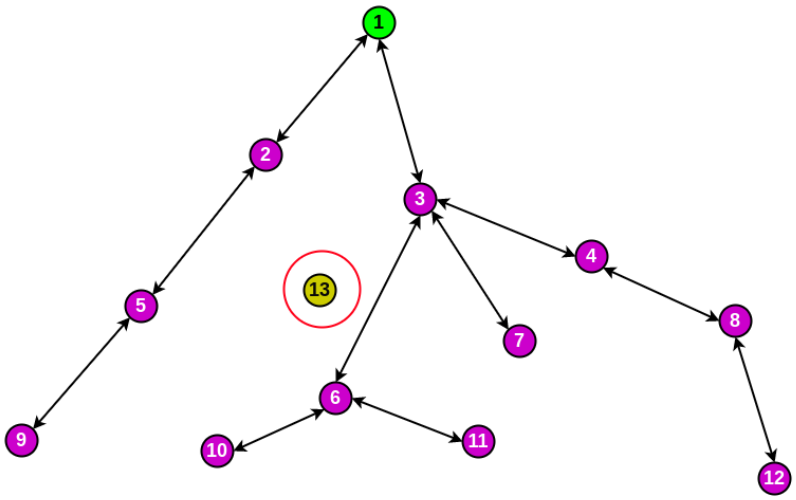


Figure 5.6: Network routes after recovery from sinkhole attack.

```
1:08:24.032  ID:1   Routes [13 max]
1:08:24.033  ID:1   Routing links:
1:08:24.039  ID:1   Link:  fd00::c30c:0:0:3   parent:  fd00::c30c:0:0:1) 120s
1:08:24.046  ID:1   Link:  fd00::c30c:0:0:6   parent:  fd00::c30c:0:0:3) 240s
1:08:24.052  ID:1   Link:  fd00::c30c:0:0:2   parent:  fd00::c30c:0:0:1) 180s
1:08:24.059  ID:1   Link:  fd00::c30c:0:0:4   parent:  fd00::c30c:0:0:3) 180s
1:08:24.065  ID:1   Link:  fd00::c30c:0:0:7   parent:  fd00::c30c:0:0:3) 180s
1:08:24.072  ID:1   Link:  fd00::c30c:0:0:a   parent:  fd00::c30c:0:0:6) 120s
1:08:24.078  ID:1   Link:  fd00::c30c:0:0:5   parent:  fd00::c30c:0:0:2) 180s
1:08:24.084  ID:1   Link:  fd00::c30c:0:0:b   parent:  fd00::c30c:0:0:6) 240s
1:08:24.091  ID:1   Link:  fd00::c30c:0:0:8   parent:  fd00::c30c:0:0:4) 180s
1:08:24.097  ID:1   Link:  fd00::c30c:0:0:9   parent:  fd00::c30c:0:0:5) 180s
1:08:24.104  ID:1   Link:  fd00::c30c:0:0:c   parent:  fd00::c30c:0:0:8) 180s
```

Figure 5.7: Routes in network recovered after the sinkhole was detected and corrected.

Results shown in Figures 5.1 to 5.7 have been collected from the topology presented Figure 5.2, with a simulation speed of 100%, meaning that a real-life second was equivalent of a simulation second. We used both a TX and RX ratio of 100% and a transmission range of 50 meters for nodes. The MAC layer protocol was the Carrier Sense Multiple Access (CSMA) and the RPL protocol was the RPL-Lite.

This evaluation allows for a direct proof of detection and correction modules effectiveness and, moreover, for an indirect proof of *IDIoT-6Mapper* and *IDIoT-Firewall* effectiveness over the course of the simulation, since the *IDIoT-Routing-Detection* relies on procedures from these two mentioned modules to collect network data and take actions against the attacker, respectively. It was possible for a network to fully recover to a stable function after suffering a sinkhole attack launched from within the network. These results

are in accordance with the expectations for the improvements and extensions provided to this Svelte original module. During the performed simulations, it was possible to come to the conclusion that detection of such a routing attack happens best with higher proximity of the attacker with the DAG-root, meaning that the detection rate is decreased for further distances, e.g., nodes 11 or 12.

5.3 DoS Detection Module

For the DoS detection evaluation, a similar topology as the one shown in Figure 5.2 is used. However, for this time, routing information is not a factor of interest since the attack will not affect the routing, but the communication capabilities. The sinkhole attacker is now disabled and a regular network is simulated until a stable point where all nodes have established a server/client relationship. Regular network nodes acting as UDP-clients are sending packets at a rate of one packet every 10 seconds, in order to recreate a possible example of network traffic rate from a specific IoT application.

Regarding the attack implementation, one of the deployed UDP-clients will be modified in order to run a DoS attack. This node, represented in simulations by number 13, performs a regular UDP-client function, just like the rest, however with an alteration concerning the packet transmission rate. For this simulation, we are using a rate of 200 packets per second as seen in Listing 5.1. The attack is set to start only after 2 minutes into the start of the simulation. Moreover, the DoS detection modules in the network nodes are only set to start after 6 minutes into the simulation. This configuration is for evaluation purposes, allowing for the attacker to have considerable time to deliver damage to the network service availability before the detection module starts running, after which the attacker will be detected and filtered. In similarity with all other regular UDP-clients in the network, the UDP *flooder* will be running IDIoT modules as well, with the exception of the DoS detection module.

Listing 5.1: UDP Flood implementation as an modified UDP-client

```
// Force connection with fixed IP for udp-server.
uip_ip6addr(&dest_ipaddr, 0xFD00, 0, 0, 0, 0x0212, 0x7401, 0x0001, 0x0101);

etimer_set(&periodic_timer, random_rand() % CLOCK_SECOND);

while(1) {
PROCESS_YIELD();
if (etimer_expired(&periodic_timer)) {
   LOG_INFO_("Sending request %u to ", count);
```

```
10    LOG_INFO_6ADDR(&dest_ipaddr);
11    LOG_INFO_("\n");
12    snprintf(str, sizeof(str), "hello %d", count);
13    simple_udp_sendto(&udp_client_conn, str, strlen(str), &dest_ipaddr);
14    count++;
15 etimer_set(&periodic_timer, SEND_INTERVAL/200 );
16 }
```

This UDP flood attack will have an enormous effect on the network traffic from the moment such attack is enabled. These overwhelming amount of traffic induced by the UDP flood will result in much higher demand of CPU availability of the DAG root which is not sustainable. Figure 5.8 shows a capture of the Mote Output window of Cooja Simulator in which is possible to see such overwhelming amount of traffic. At the time of capture, after 3 minutes and 24 seconds of the start of the simulation, it is possible to see how the UDP flooder has already sent to the DAG root over 3600 packets whereas each one of the other UDP-clients in the network has only sent 12 packets.

In Figure 5.9 a capture of the Cooja Network mote output is shown. It is possible to see node 3 from the topology detecting the UDP *flooder*, by means of its DoS detection module. This node filters the attacker locally and reports the attacker to the DAG-root. This last creates a small filter as seen in Figure 5.9. Few minutes will pass and the RPL mechanisms will allow the attacker to understand that node 3 is no longer reachable. At this time, the RPL mechanisms present in attacker node will create new routes to the root, this time through node 2, and once again transmit the overwhelming amount of packets. In accordance with node 3, node 2 will also detect this abusive behavior, thanks to its DoS detection module, and perform the same actions as mentioned before for node 3. This time, however, the root, upon reception of the filtering request from node 2, will not raise a small filter against the attacker, as it has been already reported by node 3, but instead a global filter and network repair. This global filter and network repair procedures were already presented previously in the Routing Detection, in section 5.2. As the UDP-flooder is also the mote 13 in this simulation, the graphical result of this detection is very similar to Figures 5.5 and 5.6 so these will not be repeated.

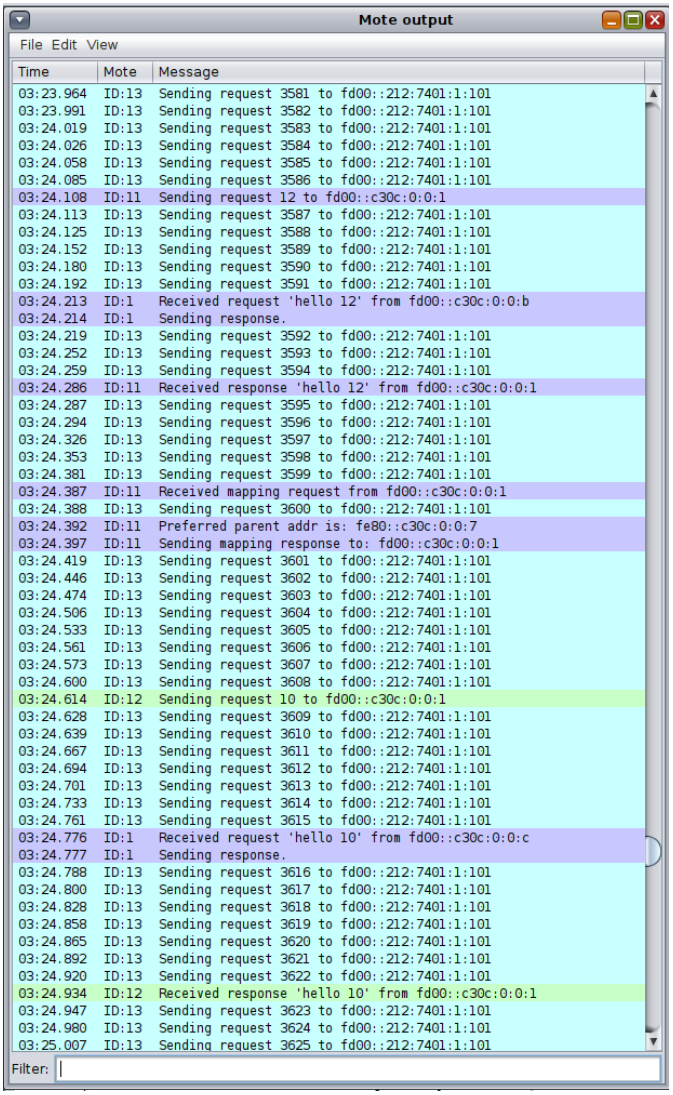


Figure 5.8: Network traffic resulting from UDP-flood attack.

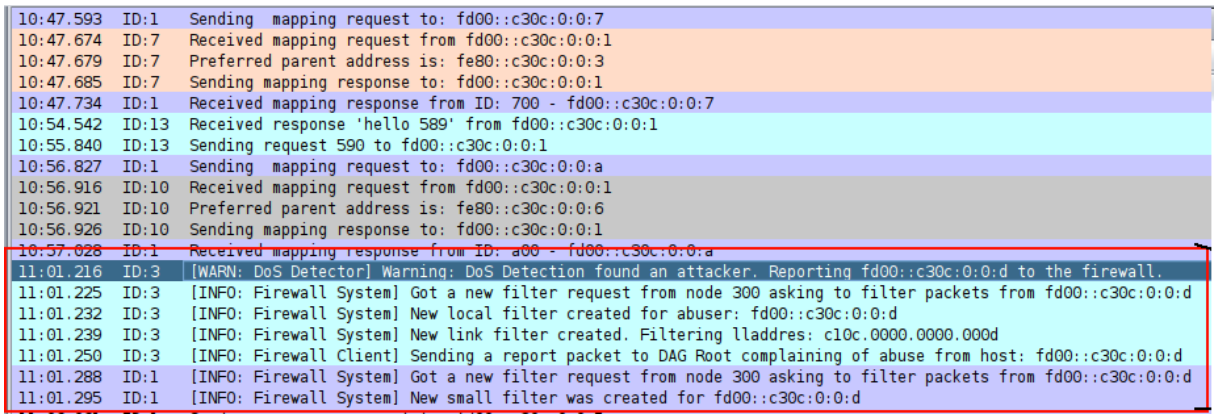


Figure 5.9: DoS detection module from node 3 detecting node 13(d) as abusive.

Thread-Metrics evaluation results have been collected for test scenarios running the DoS attack in a network with and without the IDIoT modules. We performed all the 6 tests and present the results in table 5.1. Furthermore, Figure 5.9 shows a bar chart representing the evaluation performed for the 6 Thread-Metrics tests aforementioned in Section 5.1. All simulations were performed with a topology similar to the one presented in Figure 5.2 and lasted for 15 minutes.

Evaluation performed for the DoS attack show that expectations are accomplished as the *IDIoT-DoS-Detection* deployed in both central and distributed devices found little difficulty to detect the launched UDP flood attack. Simulations were performed with different topology metrics and even with two instances of the UDP-*flooder* in the same network. The detection modules were capable to detect the attack in all simulations performed. Regarding the evaluation using the Thread-Metrics tests, the presented benchmarks in Figure 5.10 are very satisfactory and could have been gaudier if a better implementation of a DoS attack was achieved. It is possible to see, in the test scenario with IDIoT modules enabled, a network running with 97% of the performance of the network which had no DoS attack. Furthermore, the unprotected network reports performance drops up to 40% when compared with the network under DoS attack with IDIoT modules enabled.

Table 5.1: Thread Metrics results for all tests executed.

Test Nr.	No DoS & No IDS	No DoS with IDS	Under DoS without IDS	Under DoS with IDS
1	4434	4431	2757	4392
2	413606	412191	295477	403591
3	423227	422000	290034	415163
6	363882	362640	263215	360606
7	382010	380708	270625	361352
8	293343	292508	199045	290080

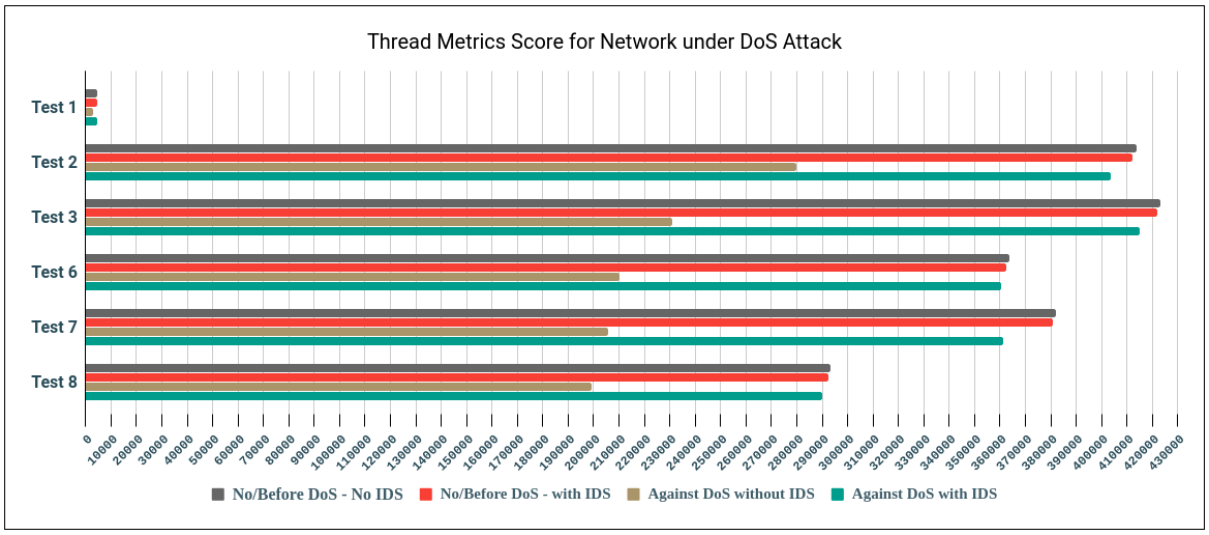


Figure 5.10: Thread-Metrics score for all tests, evaluating all topologies.

5.4 Memory footprint

Regarding the memory footprint, it is expected for the DAG-root running the IDIoT modules to have a significant higher ROM and RAM overhead in comparison with the distributed motes, accounting with the higher presence of IDIoT modules in this centralized device. Table 5.2 results for ROM and RAM overhead are shown, both for the centralized DAG-root and the distributed network node. Furthermore, in Figures 5.11, and 5.12, a graphical representation of such values is presented for a clear understanding. Results are obtained by inspecting the binary file where is possible to see the size for each code section. These code sections show the size of static allocated RAM, in sections *.data* and *.bss*, and ROM, in section *.text*. Contiki does not really use dynamic memory allocation, so this information is sufficient to determine the runtime usage. For a more thorough evaluation values are obtained while enabling each module at a time, as these can be enabled or disabled in accordance with the network administrator intentions.

Table 5.2: Memory Footprint for both UDP-server and -client with the IDIoT modules.

	DAG-root				**Regular node**			
Memory	IDIoT off	with *6Mapper*	*6Mapper*+FW	all modules	IDIoT off	with *6Mapper*	*6Mapper*+FW	all modules
RAM (kB)	6090	7454	7566	7664	6136	6176	6360	6692
ROM (kB)	40931	45051	46345	47009	41275	42337	43625	44605

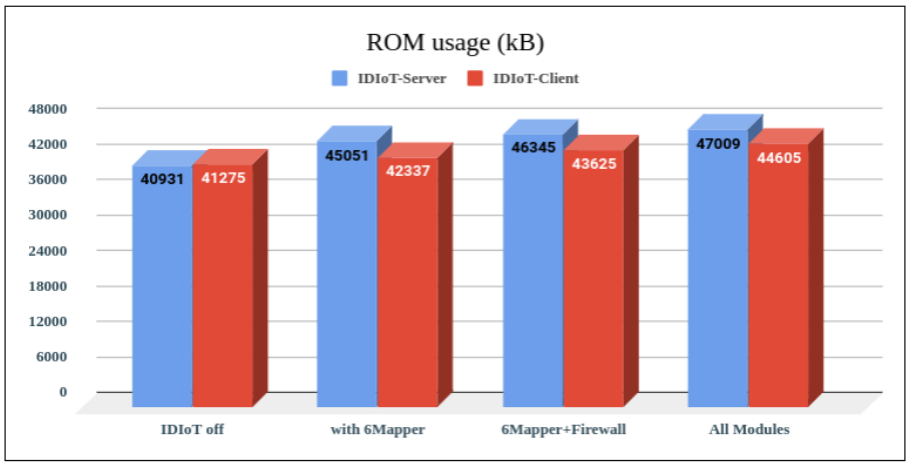


Figure 5.11: RAM usage overhead evaluation for devices running IDIoT modules.

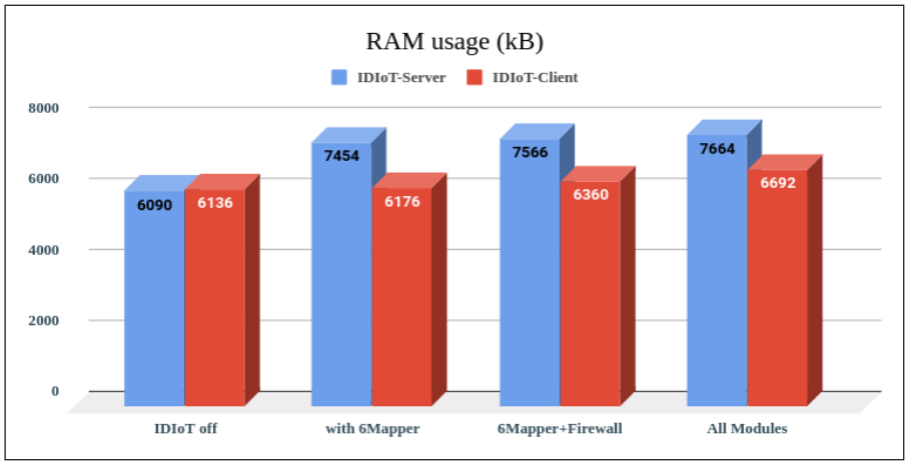


Figure 5.12: ROM usage overhead evaluation for devices running IDIoT modules.

Memory footprint evaluation fulfills the all expectations. It is possible to confirm what was expected concerning which device had a higher overhead with the inclusion of the IDIoT modules: the centralized node. An increased 6kB of ROM is necessary for all modules of the IDIoT-server and 3.3kB for the IDIoT-client. Regarding the RAM usage overhead, an increased usage of 1.5kB is seen for the IDIoT-server whereas the IDIoT-client modules induce an overhead of 0.5kB of RAM. These results are very positive concerning the possibility of deploying both IDIoT-server and -client versions on such a resource-constrained device as the Tmote Sky, with only 48kB of ROM and 10kB of RAM available.

5.5 Energy Consumption

The following Table 5.4 presents the evaluation performed on energy usage of the IDIoT modules. Usually, nodes in the IoT, such as the Tmote Sky, for instance, are battery powered and hence energy is a scarce resource. For this reason, Contiki offers a tool named PowerTrace [72] which allows for developers to access the total time different parts of the system were active. With this data, it is possible to estimate with high accuracy the power consumption of a node, when one knows the nominal values from the device in use. In Table 5.3 data is retrieved for the Tmote Sky datasheet [73], which is used for the aforementioned calculations.

Table 5.3: Tmote Sky nominal operation values.

Typical conditions	**Minimum**	**Nominal**	**Maximum**	**Unit**
Supply Voltage	2.1		3.6	V
Operating free air temperature	-40		85	C
MCU on, Radio receiving (RX)		21.8	23	mA
MCU on, Radio transmitting (TX)		19.5	21	mA
MCU on, Radio off		1800	2400	µA
MCU idle, Radio off		54.5	1200	µA
MCU standby		5.1	21.0	µA

Given the nominal operation conditions from Table 5.3 and the on and off time for each part of the system, provided from the PowerTrace application, it is possible to determine the energy, in *mJ*, for each device, with the following calculations. For *transmit* we understand the amount of time when the MCU was on and the radio was transmitting. For *listen* we understand the amount of time when the MCU was on and the radio was receiving. For CPU we understand the amount of time when the MCU was on and the radio was off, and for Low Power Mode (LPM) we understand as the amount of time when the MCU was in idle mode and the radio was off.

$$Energy(mJ) = ((transmit * 19.5mA) + (listen * 21.8mA)$$
$$+(CPU * 1.8mA) + (LPM * 0.0545mA)) * (3V/4096)$$

Evaluation of resource consumption was performed with the same topology as described in Section 5.2 and 5.3. Simulations were performed with and without the attack, as well as with and without the IDIoT modules. The obtained results can be seen in Table 5.4.

Table 5.4: Energy Consumption for devices running IDIoT modules for 15 minutes, in 4 topologies.

	IDIoT-Server				**IDIoT-Client**			
Resource	Clean	Clean+IDS	Under DoS	Under DoS + IDS	Clean	Clean+IDS	Under DoS	Under DoS + IDS
CPU (s)	2	2	2	2	0	0	18	2
LPM (s)	898	898	898	898	900	900	882	898
listen (s)	25	28	136	30	16	18	20	20
transmit (s)	875	872	765	870	884	882	880	880
Energy (mJ)	14.43	14.43	14.72	14.44	14.43	14.43	14.9	14.43

It is possible to notice, in Table 5.4, an increased energy consumption for a network under DoS attack and where the IDIoT is not enabled. However, this is only an increase of two percent in energy consumption. This is immensely negligible having in consideration that the network was set to run for 15 minutes. This negligible increase can be explained by the limitations presented by Cooja concerning energy consumption estimations of the emulated radio module. It is possible to see in Table 5.4 the radio was either on listen or transmit mode for the entire duration of the simulation, 15 minutes, 900 seconds. For this reason, it is not feasible to estimate the energy consumption of the emulated radio device on different network topologies. However, the results obtained for the MCU energy consumption, alone, are trustworthy and relevant for the metric in study. For this reason, Figure 5.13 delivers the obtained results for energy consumption concerning the MCU operation of the devices under the same circumstances as seen in Table 5.4. The topology used for this evaluation was the same seen in Figure 5.2, where the IDIoT-server is running in node 1. The DoS attack used for this simulation was the same as the one described in Section 5.3 which is launching an UDP flood towards the DAG-root.

Furthermore, and in accordance with the memory evaluation performed in 5.4, it is possible for an evaluation to be presented regarding the energy consumption of each module in the IDIoT solution, a relevant evaluation as different IDIoT modules can be enabled or disabled in accordance with the network administrator intentions.

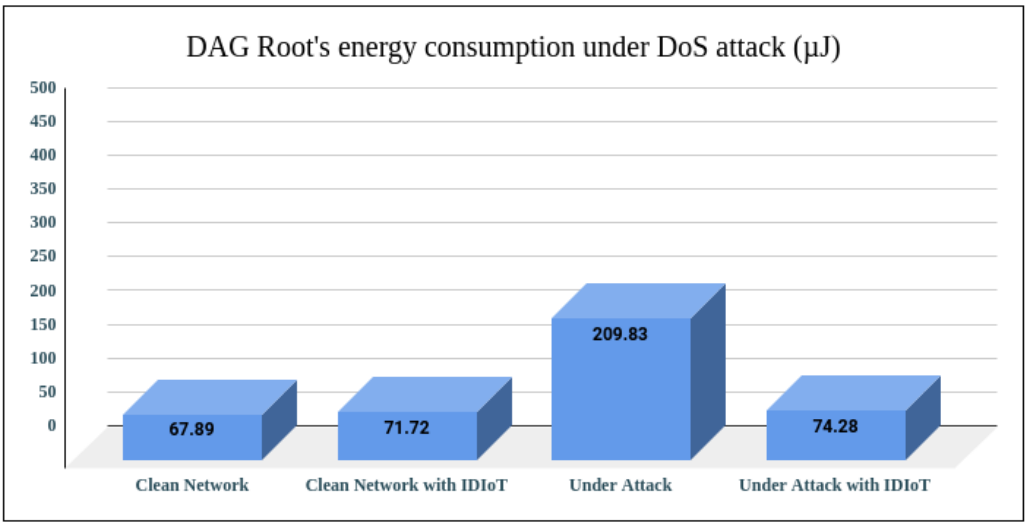


Figure 5.13: Energy consumption overhead for the DAG-root under DoS attack over 15 minutes.

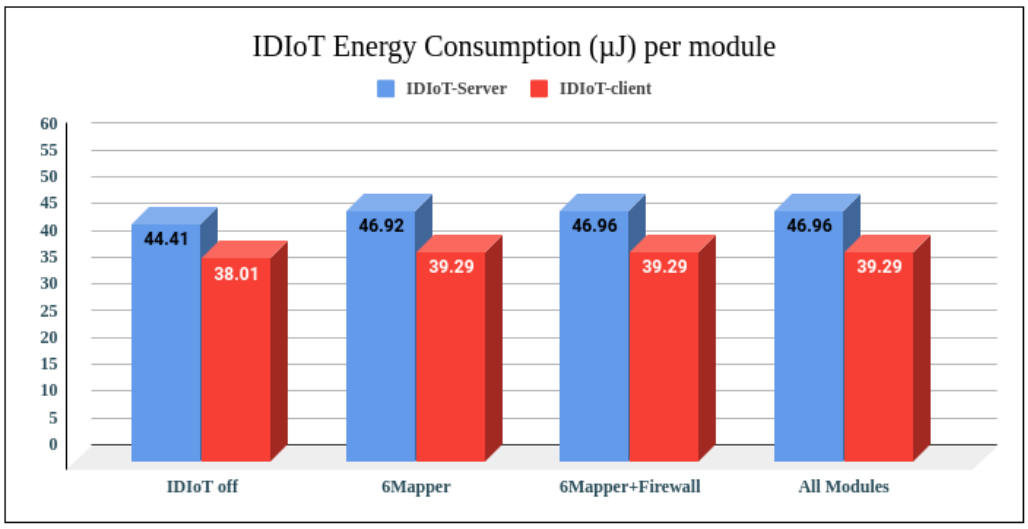


Figure 5.14: Energy consumption overhead related to each IDIoT module enabled, for a 10 minute simulation.

Energy consumption evaluations presented in Figure 5.13 shows how the target of a DoS attack can rapidly have is battery drained out. It is also possible to verify how a IDS solution like the IDIoT can be used to prevent such a simple implementation of a DoS attack to have unbearable results of a device's battery life. Moreover, in Figure 5.14, evaluation of the energy overhead in a topology with no attacks is possible to re-assure how little energy consumption these IDIoT modules induce of a device.

Chapter 6

Conclusion

The IoT has continuously advanced throughout the years since it first appeared. The deployment and utilization of IoT devices in a wide range of applications is increasing like never before. However, connecting all these devices to the Internet, enables for a vast surface of attacks to such devices and the networks these are integrated in. Security mechanisms for IoT low-end endpoint devices are demanded in order to address this issue and to establish reliability in these devices and networks. For this reason, solutions like the IDS are developed for the IoT paradigm and the current state-of-the-art presents a great amount of literature, published over the years, concerning integration of an IDS with the IoT network architecture. However, much work can still be done concerning low-end IDS solutions, since these are still in its infancy and incipient. Works do not cover or address many IoT technologies and can not detect a large variety of attacks. Moreover, one of the best solutions found, Svelte, is, nonetheless, an outdated solution running an older version of Contiki OS as well as its previous RPL implementation.

This thesis was based on the development of the IDIoT, a low-end IDS solution which is capable of detecting routing and DoS attacks, such as the sinkhole and the UDP flood. The development of the IDIoT was based on Svelte, a low-end IDS developed in Contiki-2.6. A refactoring was performed to some modules and after developing others from scratch, the resulted IDIoT solution is, to the best of our knowledge, the first of its kind to integrate both routing and DoS security mechanisms in order to prevent malicious actions towards IoT constrained devices. IDIoT is capable of detecting sinkhole attacks as well as UDP flood attacks, thanks to its detection modules running in both centralized and distributed devices of the network. Furthermore, the IDIoT also integrates into its design a Firewall which allows for centralized and distributed network nodes to perform basic filtering actions toward detected attacks, preventing this way issues such as the denial of services, increased energy consumptions, and more.

The IDIoT modules are deployed on the Contiki-NG, a promising OS for the IoT devices, as well as its most recent RPL implementation, the RPL-Lite, which enables new features concerning the routing in

low-end IoT networks. Moreover, the IDIoT is a highly configurable and extensible solution where one can choose which modules should be enabled in the network and has support for the previous RPL-Classic implementation.

The evaluation performed to the IDIoT modules show great results concerning the detection of sinkhole attacks launched from inside the network as well as UDP flood attacks which can be launched from inside the network as well as from the Internet. Thread-metrics benchmarks show, that for a UDP flood attack, in a network with IDIoT enabled, such attack will not succeed in affecting the network. However, if a network is not protected with IDIoT modules, the DoS attack will be able to degrade up to 40% of network's performance. Moreover, energy consumption evaluations also assure the detection and prevention enabled by the IDIoT modules during these attacks, which would result in much rapidly drained batteries for unprotected devices. At last, memory footprint evaluations show that the overhead of all IDIoT modules are suitable for low-end devices with resource-constrained hardware, e.g., the TMote Sky.

6.1 Future Work

Despite the IDIoT being a good solution as an IDS for the low-end IoT endpoint devices, there are still some improvements and features that can be added in future developments:

- **Integration the IDIoT with CUTE mote** - Hybrid hardware platforms, which combine a MCU and a FPGA on the same SoC are popular even at the network edge. These platforms add extra processing capabilities to already existing systems by allowing the deployment of dedicated hardware accelerators on the FPGA fabric [7] [74]. CUTE mote [7] is a in-house project specially designed for the edge network that proposes a heterogeneous architecture which combines a MCU and a Reconfigurable Computing Unit (RCU) with an IEEE 802.15.4 radio transceiver. CUTE mote supports a series of hardware accelerators concerning the IoT e related with the network stack. By having available FPGA fabric, it is possible to integrate the IDIoT IDS along with the existing network filters, allowing for the IDIoT blacklist to be directly offloaded in hardware, relieving MCU usage.

- **Integrating the IDIoT with IPv6 over the TSCH (6TISCH) based model** - The 6TISCH proposes a protocol stack rooted in the TSCH mode of the IEEE 802.15.4 standard, supports multihop topologies with the RPL and is IPv6-ready through the 6LoWPAN [75]. Having in mind the best feature of the IDIoT, the detection of an attacker, would be a great topic in research to integrate such

a solution with the 6TISCH protocol. This way, instead of limiting the IDIoT actions when detecting an attacker to filtering such attacker with the resource to the *IDIoT-Firewall*, action techniques using channel hopping provided by the TSCH could be investigated;

- **Developing an OS agnostic framework** - IoT motes are expected to have a certain degree of interchangeability concerning its functions and communications in regards to the OS being used. This is possible through compatibility provided by using the same standards and protocols. For this reason, and since the IDIoT modules are currently only running in the application and networking layer of these devices, a great improvement shall be to develop an OS agnostic framework of IDIoT so it can run in devices with, e.g., RIOT OS or TinyOS;

- **Extending the architecture to prevent more attacks** - Existing IDIoT modules are easily extensible for more attack prevention which would enrich the security mechanism. For instance, a wormhole attack is also a popular attack in wireless networks [76]. If the *IDIoT-6Mapper* is extended, in the mapping responses the clients send to the server, with the signal strength of each node's neighbor it is also possible to detect a wormhole attacks;

- **Extend the DoS detection module** - Many improvements can performed to the *IDIoT-DoS-Detection* over the basis we have created. For instance, the module can be improved to secure more flooding attacks, in similarity with the implemented UDP-flood, but this time, with other protocols such as TCP or ICMP; Moreover, the *IDIoT-DoS-Detection* can be extended for other DoS attacks which exploit well known and used protocols. For instance, concerning the TCP, it is possible to improve this module to prevent the *syn-flood*.

References

[1] T. Gomes, D. Fernandes, M. Ekpanyapong, and J. Cabral, "An IoT-based system for collision detection on guardrails," in *2016 IEEE International Conference on Industrial Technology (ICIT)*, March 2016, pp. 1926–1931.

[2] S. Pinto, J. Cabral, and T. Gomes, "We-care: An IoT-based health care system for elderly people," in *2017 IEEE International Conference on Industrial Technology (ICIT)*, March 2017, pp. 1378–1383.

[3] S. Pinto, T. Gomes, J. Pereira, J. Cabral, and A. Tavares, "IIoTEED: An Enhanced, Trusted Execution Environment for Industrial IoT Edge Devices," *IEEE Internet Computing*, vol. 21, no. 1, pp. 40–47, Jan 2017.

[4] T. Gomes, S. Pinto, F. Salgado, A. Tavares, and J. Cabral, "Building IEEE 802.15.4 Accelerators for Heterogeneous Wireless Sensor Nodes," *IEEE Sensors Letters*, vol. 1, no. 1, pp. 1–4, Feb 2017.

[5] T. Gomes, F. Salgado, S. Pinto, J. Cabral, and A. Tavares, "Towards an FPGA-based network layer filter for the Internet of Things edge devices," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2016, pp. 1–4.

[6] T. Gomes, F. Salgado, S. Pinto, J. Cabral, and A. Tavares, "A 6LoWPAN Accelerator for Internet of Things Endpoint Devices," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 371–377, Feb 2018.

[7] T. Gomes, F. Salgado, A. Tavares, and J. Cabral, "CUTE Mote, A Customizable and Trustable End-Device for the Internet of Things," *IEEE Sensors Journal*, vol. 17, no. 20, pp. 6816–6824, Oct 2017.

[8] T. Gomes, S. Pinto, T. Gomes, A. Tavares, and J. Cabral, "Towards an FPGA-based edge device for the Internet of Things," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sep. 2015, pp. 1–4.

[9] A. Ribeiro, C. Rodrigues, I. Marques, J. Monteiro, J. Cabral, and T. Gomes, "Deploying a Real-Time Operating System on a Reconfigurable Internet of Things End-device," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, Oct 2019, pp. 2946–2951.

[10] S. Pinto, D. Oliveira, J. Pereira, N. Cardoso, M. Ekpanyapong, J. Cabral, and A. Tavares, "Towards a lightweight embedded virtualization architecture exploiting ARM TrustZone," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Sep. 2014, pp. 1–4.

[11] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 51, no. 6, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3291047

[12] S. Raza and R. Magnusson, "TinyIKE: Lightweight IKEv2 for Internet of Things," *IEEE Internet of Things Journal*, vol. PP, pp. 1–1, 08 2018.

[13] K. Ashton. (2009) That 'Internet of Things' Thing. [Online]. Available: https://www.rfidjournal.com/that-internet-of-things-thing

[14] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[15] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[16] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," *Computer Networks*, pp. 2787–2805, 10 2010.

[17] L. Mainetti, L. Patrono, and A. Vilei, "Evolution of wireless sensor networks towards the Internet of Things: A survey," in *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, Sep. 2011, pp. 1–6.

[18] P. Sethi and S. R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications," *Journal of Electrical and Computer Engineering*, vol. 2017, pp. 1–25, 01 2017.

[19] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized Protocol Stack for the Internet of (Important) Things," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1389–1406, Third 2013.

[20] C. Perera, C. H. Liu, S. Jayawardena, and M. Chen, "A Survey on Internet of Things From Industrial Market Perspective," *IEEE ACCESS*, vol. 2, pp. 1660–1679, 01 2015.

[21] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[22] M. Baby. (2018) Sleep Trackers for little ones. [Online]. Available: https://www.mimobaby.com/

[23] S. Trash. (2019) An overview of Smart City waste management market. View wireless solution providers, case studies and data platform offerings. [Online]. Available: https://www.postscapes.com/smart-trash/

[24] S. Agriculture. (2019) Filter and discover IoT Agriculture Resources. View smart farm case studies, sensor applications and potential resource and labor saving dashboards, tools and apps. [Online]. Available: https://www.postscapes.com/smart-agriculture/

[25] S. R. Department. (2016) Internet of Things connected devices installed base worldwide from 2015 to 2025 (in billions). [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[26] O. Oman. What is IoT? Understanding IoT Device Management and Standards.

[27] L. Santos, C. Rabadao, and R. Gonçalves, "Intrusion detection systems in Internet of Things: A literature review," in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, June 2018, pp. 1–7.

[28] A. Dunkels, "Design and implementation of the lwIP TCP/IP stack," *Swedish Institute of Computer Science*, vol. 2, 03 2001.

[29] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, "Operating Systems for Low-End Devices in the Internet of Things: A Survey," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 720–734, Oct 2016.

[30] R. Navas, "State of the art of IETF security related protocols for IoT," 11 2016.

[31] J. Granjal, E. Monteiro, and J. Sá Silva, "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 07 2015.

[32] G. Montenegro, J. Hui, D. Culler, and N. Kushalnagar, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944, Sep. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4944.txt

[33] G. Montenegro, C. Schumacher, and N. Kushalnagar, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals," RFC 4919, Aug. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4919.txt

[34] S. Raza, S. Duquennoy, J. Höglund, U. Roedig, and T. Voigt, "Secure communication for the Internet of Things—a comparison of link-layer security and IPsec for 6LoWPAN," *Security and Communication Networks*, vol. 7, 12 2014.

[35] K. Devadiga, "IEEE 802 . 15 . 4 and the Internet of things," 2011.

[36] A. Arış, S. F. Oktuğ, and T. Voigt, *Security of Internet of Things for a Reliable Internet of Services*. Cham: Springer International Publishing, 2018, pp. 337–370. [Online]. Available: https://doi.org/10.1007/978-3-319-90415-3_13

[37] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6550.txt

[38] M. Khan, M. Lodhi, A. Rehman, A. Khan, and F. Hussain, "Sink-to-Sink Coordination Framework Using RPL: Routing Protocol for Low Power and Lossy Networks," *Journal of Sensors*, vol. 2016, pp. 1–11, 07 2016.

[39] M. Silva, D. Cerdeira, S. Pinto, and T. Gomes, "Operating Systems for Internet of Things Low-end Devices: Analysis and Benchmarking," *IEEE Internet of Things Journal*, vol. PP, pp. 1–1, 09 2019.

[40] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *29th Annual IEEE International Conference on Local Computer Networks*, Nov 2004, pp. 455–462.

[41] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2013, pp. 79–80.

[42] R. Barry. FreeRTOS, a free open source RTOS for small embedded real time systems.

[43] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, *TinyOS: An Operating System for Sensor Networks*.

Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 115–148. [Online]. Available: https://doi.org/10.1007/3-540-27139-2_7

[44] R. Barry. Berkeley's OpenWSN project.

[45] J. Deogirikar and A. Vidhate, "Security attacks in IoT: A survey," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Feb 2017, pp. 32–37.

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

[47] L. Liang, K. Zheng, Q. Sheng, and X. Huang, "A Denial of Service Attack Method for an IoT System," in *2016 8th International Conference on Information Technology in Medicine and Education (ITME)*, Dec 2016, pp. 360–364.

[48] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in internet of things," *Journal of Network and Computer Applications*, vol. 84, pp. 25 – 37, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517300802

[49] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870513001005

[50] A. A. Gendreau and M. Moorman, "Survey of Intrusion Detection Systems towards an End to End Secure Internet of Things," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, Aug 2016, pp. 84–90.

[51] J. Anderson, "Computer Security Threat Monitoring and Surveillance," 01 1980.

[52] A. Ashoor and S. Gore, "Intrusion Detection System (IDS): Case Study," *International Journal of Scientific and Engineering Research*, vol. 2011, 06 2019.

[53] S. Community. (2020) Snort, an open source IPS capable of real-time traffic analysis and packet logging. [Online]. Available: https://www.snort.org

[54] O. I. S. Foundation. (2020) Suricata, Open Source IDS / IPS / NSM engine. [Online]. Available: https://suricata-ids.org/

[55] J. S. White, T. Fitzsimmons, and J. N. Matthews, "Quantitative analysis of intrusion detection systems: Snort and Suricata," in *Cyber Sensing 2013*, I. V. Ternovskiy and P. Chin, Eds., vol. 8757, International Society for Optics and Photonics. SPIE, 2013, pp. 10 – 21. [Online]. Available: https://doi.org/10.1117/12.2015616

[56] R. Fekolkin, "Intrusion Detection and Prevention Systems: Overview of Snort and Suricata," 01 2015.

[57] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits, "Denial-of-Service detection in 6LoWPAN based Internet of Things," in *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2013, pp. 600–607.

[58] P. Kasinathan, G. Costamagna, H. Khaleel, C. Pastrone, and M. A. Spirito, "DEMO: An IDS Framework for Internet of Things Empowered by 6LoWPAN," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1337–1340. [Online]. Available: https://doi.org/10.1145/2508859.2512494

[59] C. Cervantes, D. Poplade, M. Nogueira, and A. Santos, "Detection of sinkhole attacks for supporting secure routing on 6LoWPAN for Internet of Things," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 606–611.

[60] A. Le, J. Loo, M. Chai, and M. Aiash, "A Specification-Based IDS for Detecting Attacks on RPL-Based Network Topology," *Information*, vol. 7, 05 2016.

[61] F. Medjek, D. Tandjaoui, I. Romdhani, and D. Nabil, "A Trust-based Intrusion Detection System for Mobile RPL Based Networks," 06 2017.

[62] Y. Fu, Z. Yan, J. Cao, O. Koné, and X. Cao, "An Automata Based Intrusion Detection Method for Internet of Things," *Mobile Information Systems*, vol. 2017, pp. 1–13, 01 2017.

[63] S. Suganth and D. Usha, "A Survey of Intrusion Detection System in IoT Devices." *International Journal of Advanced Research*, vol. 6, pp. 23–30, 06 2018.

[64] P. Ioulianou, V. Vassilakis, and I. Moscholios, "A Signature-based Intrusion Detection System for the Internet of Things," 07 2018.

[65] T. Mehmood, "COOJA Network Simulator: Exploring the Infinite Possible Ways to Compute the Performance Metrics of IOT Based Smart Devices to Understand the Working of IOT Based Compression and Routing Protocols," 2017.

[66] A. Velinov and A. Mileva, "Running and Testing Applications for Contiki OS Using Cooja Simulator," vol. 1, 06 2016, p. 279.

[67] B. Ghaleb, A. Al-Dubai, E. Ekonomou, M. Qasem, I. Romdhani, and L. Mackenzie, "Addressing the DAO Insider Attack in RPL's Internet of Things Networks," *IEEE Communications Letters*, vol. PP, 10 2018.

[68] P. Perazzo, C. Vallati, G. Anastasi, and G. Dini, "DIO Suppression Attack Against Routing in the Internet of Things," *IEEE Communications Letters*, vol. 21, no. 11, pp. 2524–2527, Nov 2017.

[69] A. Verma and V. Ranga, "Mitigation of DIS flooding attacks in RPL□based 6LoWPAN networks," *Transactions on Emerging Telecommunications Technologies*, pp. 1–25, 10 2019.

[70] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, no. 10, pp. 54–62, 2002.

[71] W. Lamie and J. Carbone, "Measure your RTOS's real-time performance," 01 2020.

[72] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level Power Profiling for Low-power Wireless Networks," p. 14, 04 2011.

[73] M. Corporation. (2005) Tmote Sky User Manual and Datasheet. [Online]. Available: https://fccid.io/TOQTMOTESKY/User-Manual/Users-Manual-Revised-613136

[74] A. Engel and A. Koch, "Heterogeneous Wireless Sensor Nodes that Target the Internet of Things," *IEEE Micro*, vol. 36, no. 6, pp. 8–15, Nov 2016.

[75] X. Vilajosana, T. Watteyne, T. Chang, M. Vučinić, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: A Tutorial," *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 595–615, 2020.

[76] P. Pongle, "Real Time Intrusion and Wormhole Attack Detection in Internet of Things," Ph.D. dissertation, 06 2015.