UNIVERSITÀ DEGLI STUDI DI PARMA

DOTTORATO DI RICERCA IN
"TECNOLOGIE DELL'INFORMAZIONE"

CICLO XXXIV

Security in IoT Systems and Applications

Coordinatore:
Chiar.mo Prof. Marco LOCATELLI

Tutore:
Chiar.mo Prof. Luca VELTRI

Dottorando: Amin TAYEBI

Anni Accademici 2018/2019-2021/2022

To my wife and my mother.

.

Contents

List of Figures

List of Tables

Introduction

Nowadays we are seeing an amazing grown up of IoT usage and its tremendous applications in the human life. IoT smart devices are spreading in all aspects of our society, including industry, smart health, smart city, smart home, smart transport, smart sport and many others.

Communication mechanism, long range wireless technologies seem to be particularly adapt for IoT applications. Examples are SigFox, NB-IoT, LTE-M, and LoRaWAN. In particular LoRaWAN is currently having a lot of attentions and deployments.

LoRaWAN is a low-power radio access and system architecture for LoRa devices and it is an open network standard developed by a non-profit association called LoRa Alliance. It is based on LoRa, a proprietary physical layer developed by Semtech Corporation. It is a universal standard for LoRa communication technology and provides solid connectivity between devices and networks without the need for complex installation. As designed for IoT applications, LoRa complies with important technical requirements such as two-way communication, end-to-end security, mobility and localization services. LoRaWAN specifications directly face different relevant aspects like device battery life, network capacity, quality of service, security and the variety of applications provided by the network.

One of the practical advantages of LoRaWAN is that it provides a complete and light IoT architecture for connecting low-power devices, using the long range wireless technology. LoRaWAN devices can be seen as LoRa converters which convert sensor digital data (e.g. soil moisture) to LoRa messages and send them to an application-server through one or more LoRaWAN gateways and proper backend system. The success of LoRaWAN in the last years is demonstrated by the appearance of large number of LoRaWAN devices and systems.

In this scenario security becomes a crucial aspect and it is essential to protect both the devices used by these systems and the data they exchange.

Indeed, given the broad utilization of IoT devices, malicious manipulations could cause deep implications on the security and the strength of the entire Internet. The Cyber attack launched by the Mirai malware [1] represents a clear example of the severity caused by instrumenting zombified IoT devices (bots) to launch a larger DDoS attack, and testifies the necessity of secure authentication mechanisms [2], together with proper traffic classification techniques.

0.1 Problems and Solutions

Unfortunately the constrains existing in several IoT devices, such as low computation power, low memory and constrained power sources, make it troublesome to apply current security solutions utilized in the standard Internet and solid cryptography mechanisms. For these reasons novel security measures ought to be outlined and be optimized taking into consideration the being special properties of the IoT environment.

Since the communication often performs over the Internet, IoT environment is vulnerable to several security threats. Thus implement secure infrastructure and secure communication between IoT components in this massive network has become a significant issue.

As the most founded attacks in IoT systems has been done on the network layer, so analyzing network layer is more in notice to physical layer and application layer. However, this engagement raises serious issues since a lot of network traffic as well as many different traffic classes exist, e.g., those generated from industrial machineries, driverless cars, health sensors, smart homes and other critical devices. As such, the requirements of various IoT applications demand for more security and protection that, in turn, involve accurate classification of network traffic to early detect attacks and perform proper countermeasures.

One important aspect in the protection of an IoT system is the possibility of early detection of possible attack attempts. Therefore, The need for early detection of IoT malicious traffic is a current hot topic, but the methods currently available, even those based on artificial neural networks are not satisfying for different reasons: i) do not reach yet the maximum possible accuracy, ii) most of the time have been tuned based on local ad hoc traffic datasets from one single network scenario, and iii) sometimes do not consider actual IoT traffic at all. In this research a novel applicable and effective model based on neural network have been worked up. Main contribution on this aspect includes:

- the analysis of recent IoT datasets publicly available;
- the creation of a large dataset of IoT traffic, encompassing different types of IoT attacks, and the computation of a large set of features, to foster new analyses and researches;
- the design and implementation of a deep neural network classifier, capable to achieve very high accuracy, more than all similar contributions in the recent literature, over a dataset made of traffic from different networks.

From the application point of view, in order to study and use IoT specific protocols and systems, a smart agriculture and precision irrigation scenario has been considered. This scenario has been also the focus of the Emilia-Romagna funded project POSITIVE (Protocolli operativi scalabili per l'agricoltura di precisione).

In this application scenario, security aspects related to LoRaWAN systems have been studied, and an extensive experimentation activity have been carried out. Different hardware and software LoRaWAN systems have been tested, and an integrated testbed formed by different sensor devices, gateways and LoRaWAN open networks has been deployed.

When trying to integrate different IoT platforms in a larger IoT system, possibly formed by both LoRaWAN and non-LoRa devices, some complications may arise without any off-the-shelf solution for the user. For this reason we studied and proposed possible interworking solutions, covering the cases when:

- non-LoRa devices need to be connected to to a LoRaWAN system;
- LoRaWAN devices need to be connected to an existing non-LoRaWAN IoT platform;
- different LoRaWAN and non-LoRa networks need to be connected together creating a larger and heterogeneous IoT system.

0.2 Structure of the Thesis

The rest of this thesis is organized as follows:

- In Chapter 1 novel application protocols (CoAP, MQTT) and access technologies (LoRa and LoRaWAN) are described.
- In Chapter 2 the security threats and vulnerabilities in IoT are described. Then security limitations which LoRaWAN encountered with them and their countermeasures are displayed.
- In Chapter 3 a possible solution aiming to combat threats in IoT systems via anomaly detection approach using Deep learning is proposed. First background of Internet traffic classification and deep neural networks is reported, then deep learning for IoT attack classification is discussed. Next both the considered feature model and neural network model are described in detail. Then the integrated dataset and the experimental settings presented, further it shows the obtained results as well as a comparison with traditional machine learning approaches.
- In continue Chapter 4 it is explained POSITIVE project scope, technologies and activities. Then LoRaWAN node prototyping and real world experimentation are presented. Moreover possible solutions for different interworking scenarios between LoRaWAN systems and other non-LoRa IoT systems are proposed. Finally testbed implementing of the proposed solutions is described.

- Ultimately, Chapter 5 sealed up the thesis with some conclusions and future works.

Chapter 1

IoT Technologies

IoT technologies and protocols recently has been developed to be more suitable with targeted IoT applications. They have been developed mostly in wireless communications due to the protocol difficulty or impossibility to connect devices and network components via a physical communications equipment.

Wireless systems are presently a portion of the lifestyle of numerous individuals and are utilized for numerous applications. As of late, modern technologies that empower low-power and long-range communications have developed. Long-range innovations are utilized to create Low-Power Wide-Area Systems (LPWAN). Numerous LPWAN technologies are available, and they offer diverse exhibitions, business models etc., replying diverse applications needs. This makes it difficult to select the proper device for a particular application scenario. Among SigFox, NB-IoT and LTE-M novel communication technologies LoRaWAN network has been developed. LoRaWAN with lots of advantages more satisfies IoT devices mostly the low power ones. It provides Long-Range IoT communications in unlicensed bands. In [3] the most conspicuous LPWAN advances, SIGFOX, Ingenu, and LoRa, have been presented and compared to the current short-range communication benchmarks. Their experimentation done using LoRa innovation have depicted that the LPWAN worldview has the potential to complement current IoT standards as an enabler of smart city applications.

IoT networks may interconnects a possible large number of heterogeneous devices, denoted as "smart objects", in an Internet-like structure, which extends the current Internet, enabling new forms of interactions between objects Machine-to-Machine (M2M). The IoT smart objects are embedded with electronics and software. They may also have one or more sensors and/or actuators. The devices collect various data, interconnect with each other and exchange the data. Smart objects are often constraint, mainly in terms of memory capacity, processing powers and limited energy sources. Nowadays, IoT systems are integrated in almost every field of life. The IoT is related to various applications addressing diverse needs of the society such as smart city and smart buildings, smart health and smart living, smart transport and smart energy, etc.

1.1 Application Protocols in IoT Systems

IoT communication protocols are the way the devices communicate and exchange data, which sometimes may include also some security services. IoT protocols can be classified into two separate categories:

- network and transport protocols which are used for connecting devices over the network (e.g. LoRaWAN);
- application protocols which are used for communication at application level between low power IoT devices (e.g. CoAP, MQTT, Web-socket).

The most preferred application protocols in the IoT applications are CoAP and MQTT.

1.1.1 Constrained Application Protocol (CoAP)

The Constrained Application Protocol (CoAP) is a specialized web transfer like protocol for use with constrained nodes and constrained networks in the IoT. The protocol is designed for M2M applications such as smart energy and building automation. For simplified integration with the web, CoAP has been designed to be easily translated to/from HTTP, while also meeting some specific requirements of the IoT: multicast support, very low overhead and overall simplicity. The following features of CoAP have been proposed by the Constrained RESTful Environments (CoRE) IETF group that has designed the protocol with these main requirements:

- RESTful protocol design which minimizes the complexity of mapping to HTTP
- URI and content-type support
- low header overhead
- low parsing complexity
- discovery of resources provided by known CoAP services
- simple subscription for a resource and resulting push notifications
- simple caching capability.

Similarly to HTTP protocol, CoAP is based on the REST (Representational State Transfer) model, according to which servers make resources available under a URL, and clients access the resources using appropriate methods such as GET, PUT, POST, and DELETE. CoAP has been designed to use minimal resources in terms of both the devices and the network. Instead of a complex transport stack, it runs over UDP on IP. A 4-byte fixed header and a compact encoding of options enables small messages which lead to no or little fragmentation on the link layer. The interaction model of CoAP is similar to the HTTP client/server model. However, machine-to-machine interactions usually result in a CoAP implementation acting both as a client and a server. A CoAP request is similar to that of HTTP and is sent by a client to a server in order to request an action (using a Method Code to specify the action) on a resource-identified by a URI. The server then replies with a response containing a Response Code; this response may also include the resource representation. However, unlike HTTP protocol, CoAP deals with those interchanges over a datagram-oriented transport such as UDP in asynchronous way. Both for requests and responses CoAP utilizes a short fixed-length (4 bytes) binary header which may be followed by a variable-length (0..8 bytes) Token value, followed by a sequence of zero or more

CoAP Options in Type-Length-Value (TLV) format, optionally followed by a payload that possesses the rest of the datagram.

The header is followed by the Token value which the length is varies from 0 to 8 bytes according to the Token Length field. The Token value is used to correlate requests and responses. Header and Token value are followed by zero or more Options. Each Option can be followed (i) by another Option, (ii) by the Payload Marker and the payload or (iii) by the end of the message. Following the header, token and options comes the optional payload. If it is present and of non-zero length, it is prefixed by a one-byte Payload Marker (0xFF). The payload data extends from after the payload marker to the end of the UDP datagram, i.e., the Payload Length varies based on the size of the datagram and the token and options if any. A CoAP message, appropriately encapsulated into a single UDP payload, which should fit within an IP packet (in order to avoid IP fragmentation).

1.1.2 Message Queuing Telemetry (MQTT)

MQTT is a messaging protocol based on publish/subscribe pattern that works in IPv4 and IPv6 networks on top of TCP. A standard MQTT system consists of clients communicating with a server, usually called a "broker". A client may be either a publisher of information or a subscriber, and in some cases both of them. Subscribers are clients that want to receive data on a specific topic. In order to do this, they send a subscription request (SUBSCRIBE) to the broker indicating the desired topic name/names. Publishers are clients that publish information on particular topics. When a publisher has some data to distribute, it sends a message (PUBLISH request) including those data and the topic name to the broker. The broker then forwards the information to all clients that have subscribed to the topic. In MQTT data are sent in plain text format, even such sensitive information as username and password. For this reason, the standard proposes to use the Transport Layer Security (TLS) protocol as a mean of protecting message exchanges providing authenticity and confidentiality of data exchanged within the system.

1.2 LoRaWAN

LoRaWAN is a low-power radio access and system architecture for LoRa devices and it is an open network standard developed by a non-profit association called LoRa Alliance. It is based on LoRa, a proprietary physical layer developed by Semtech Corporation. It is a universal standard for LoRa communication technology and provides solid interoperability between devices and networks without the need for complex installation. As designed for IoT applications, LoRa complies with important technical requirements such as two-way communication, end-to-end security, mobility and localization services. LoRa typically operates on free-license sub-gigahertz radio bands, allowing anyone to create a LoRaWAN network. LoRa technology provides long power transmission using very low power consumption. LoRaWAN specifications directly face different relevant aspects like device battery life, network capacity, quality of service, security and the variety of applications provided by the network.

LoRaWAN has been conceived for connecting low power IoT devices in a wireless wide coverage area. The physical layer uses the LoRa (Long Range) technology, a Chirp Spread Spectrum modulation technique, on license-free sub-gigahertz radio frequency bands, for the communication between the end devices and one or multiple LoRaWAN gateways. On top on the physical link, LoRaWAN defines a MAC layer and the entire network architecture for letting end devices communicate, via the gateways, with a backend network system and an application server [4].

The general LoRaWAN network architecture is summarized in Fig. 1.1. The architecture includes the following elements: the end devices, one or more gateways, a network server, a join server, and the application servers.

Possibly very constrained wireless devices communicate with a base station (LoRaWAN gateway) using LoRa technology The gateway relays the communication to a backend system formed by a network server, join server, and application server.

The end devices use LoRa technology to send and receive LoRaWAN MAC packets. A device may send data once in a while (class A device), receive data regularly (class B device) or listen continuously for data (class C device).

One or more LoRaWAN gateways act as relay systems receiving and relaying

MAC packets toward a network server. Similarly, the gateway receives MAC packets from the network server and relays them toward the target devices using the LoRa radio interface. In the backend system the network server acts as dispatcher element, by properly relaying MAC packets between the gateways and the proper destination entity, either a join server or application server.

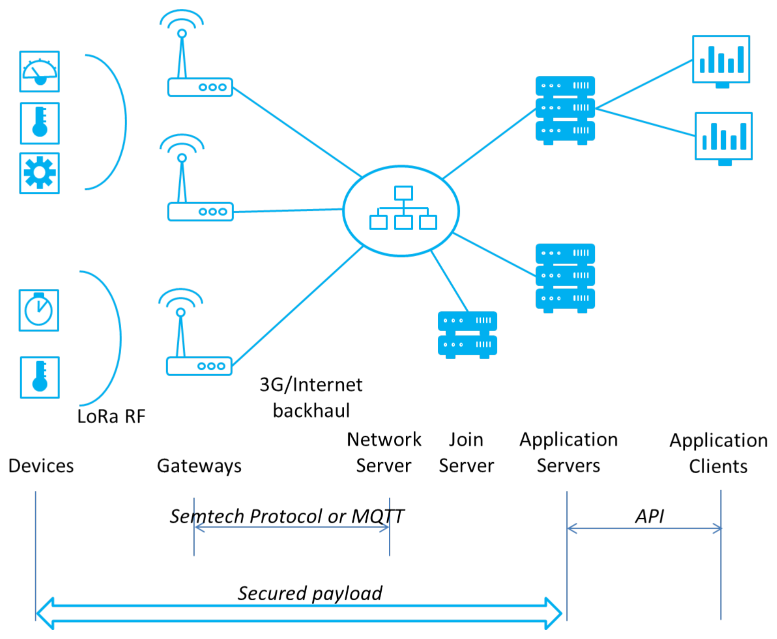

Figure 1.1: LoRaWAN network architecture.

The network server is the heart of a LoRaWAN network, it provides communication between sensor devices and user applications. It certifies message integrity and authentication through a cryptography message authentication code and sequence number tracing. AES-128 encryption is operated through the use of pre-shared keys. LoRa devices cannot be directly reached and communicate with the Internet because do not utilize the Internet Protocol (IP). LoRa devices can only talk to LoRaWAN applications to which they have been registered and the management is done on the LoRaWAN network server.

A LoRaWAN backend system implements the network server operations and provides frame control and security. The network server manages the entire network, dynamically controls the network parameters to adapt the system to ever-changing conditions, and establishes secure 128-bit AES connections for the transport of both the end to end data (from LoRaWAN end device to the user application) as well as for the control of traffic that flows from the LoRaWAN end device to the network server. While the network server ensures the authenticity and integrity of every message, it cannot access the application data that is end-to-end encrypted. Network server is responsible for:

- Managing all MAC layer functionality, by relaying these messages to/from Join server and Roaming, End-node address control
- Frame counter and authentication controls
- Acknowledgment mechanism
- Adaptive Data Rate (ADR)
- Data delivery to application servers
- Data delivery to end-devices from application servers

Although the semtech protocol itself doesn't provide an algorithm for ADR however, it gives necessary MAC commands to control it. Thus controlling the ADR with an API is essential issue which need to be considered.

The join server is responsible for securely handling joining procedure by processing Join requests and responding with Join Accept messages.

Finally the application server is in charge of securely handling the exchanged data by decrypting/encrypting the application payloads [5].

All cryptography functions are based on the AES cipher and on pre-shared symmetric secret keys associated to each device. If the Over-The-Air Activation (OTAA) is used by the end devices, during the joining procedure new session keys (a network session key and an application session key) are dynamically generated based on the pre-shared keys.

The communication between the end devices and the various network elements is performed using LoRaWAN messages encapsulated within LoRaWAN MAC packets. These MAC packets are transmitted using the LoRa radio physical layer between the end device and the gateway. The same packets are then relayed on non-LoRa links between the gateway and the network server and between the network server and the join or application server, encapsulated in other transport protocols. While the actual protocols used on non-LoRa links are left to the implementation, most gateways uses a de-facto standard and simple protocol originally proposed by Semtech and based on UDP. Another protocol supported by several gateways for communicating with the network server is MQTT [6]. In both cases the communication is based on IP and may use either a private IP network or Internet. The resulting protocol architecture is depicted in Fig. 1.2.

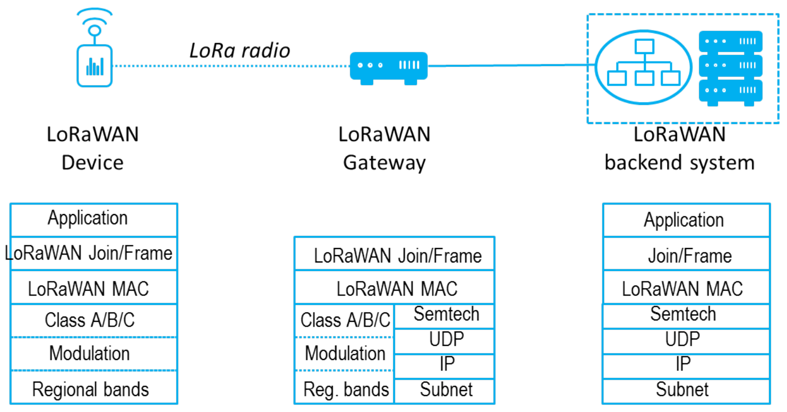

Figure 1.2: LoRaWAN protocol architecture.

One of the practical advantages of LoRaWAN is that it provides a complete and light IoT architecture for connecting low-power devices, using the LoRa wireless technology. Main strengths of LoRaWAN are as follows:

- offers cheap end devices and installation cost
- openness and possibilities of deploying private networks
- remove the need for service subscription (such as in SigFox or NB-IoT)

- decrease the operational costs for massy IoT deployments
- has large coverage with single gateway (over 1000 node per gateway)
- provide low power operation for end nodes
- high community support

1.2.1 LoRaWAN Radio Interface

The modulation used for LoRaWAN is based on Chirp spread spectrum (CSS), i.e. the modulation uses the whole bandwidth of the communication channel by encoding a message in sequences of chirps. The LoRa layer uses chirps to transmit symbols, which is an interval in which the transmission frequency is periodically increased. When it reaches the upper limit of the frequency channel, it overflows to the minimum limit and continues to increase from there. The chirp is completed when the start frequency has been reached. Because of the spreading with spreading factor (SF), every symbol will be a series of 2SF chirps. Semtech sets the chirp speed equivalent to the used bandwidth, so a bandwidth (BW) of 100 kHz results in a chirp rate (CR) of 100 kcps (Kilo chirp per second). The time on air for a single chirp is possible to be calculated using "online airtime calculator" [1]. Finally in LoRaWAN backend side all the send and receive (RX/TX) meta-data will be shown together with the raw LoRaWAN physical (PHY) payload in a readable format.

All LoRaWAN end devices for instance in the European region are required to support at least three default channels in the 868 MHz frequency band for uplink communication with 868.1, 868.3 and 868.5 MHz frequencies. An end device changes channel in a pseud-random fashion for every transmission which makes the system more robust to interferences. LoRaWAN devices are required to be able to store 16 channels, with support for in total 8 different frequencies (including the default frequencies). Furthermore in Europe the regional specifications specify the default maximum output transmit power as 25 mW (miliwat) for end devices which corresponds

[1] https://www.thethingsnetwork.org/airtime-calculator

to the maximum output power of frequency 868 MHz. Although the regional specifications also allow for an output power of 100 mW (milliwatt) the user is reminded that it is his responsibility to not exceed the regional regulatory limits.

The LoRaWAN specification provides for a maximum data payload (Frame payload/FRMPayload) size that is defined for each worldwide region in the LoRaWAN Regional Parameters document. This maximum payload size varies by DataRate [2] because of the maximum on-air transmission time allowed for each regional specification.

1.2.2 LoRaWAN Protocols

LoRaWAN uplink messages are sent to gateway encapsulated into radio packets.After a specified number of packets are exchanged between the device and the network, the forwarder initiates communication by sending a PUSH DATA message to the server.

As gateways can be connected to network servers using different technologies it involves network address translation which keeps a server from initially sending a message to the gateway. However, since this behavior is needed, the protocol regularly sends PULL DATA messages to the server to open a route to the gateway and at the same time informing the server about the open route which is in fact a keep alive message to hold the connection between gateway and the network server [7].

After acknowledging the PULL DATA message the server is now able to send PULL RESP packets downlink. Theses packets include the physical payload that the gateway transmits to the end device via transceiver. If a transmission to the gateway is successful the gateway responds with a TX ACK message.

Nodes broadcast LoRaWAN messages over the LoRa radio protocol. These messages are received by a number of gateways. The network server is the entity used by the backend platform for communicating with the gateways.

Their responsibility is to map a device to an application, to forward uplink messages to the correct application and to forward downlink messages to the correct

[2]https://lora-developers.semtech.com/documentation/tech-papers-and-guides/implementing-adaptive-data-rate-adr/implementing-adaptive-data-rate

Router (which forwards them to a Gateway). The Network Server is responsible for functionality that is specific for LoRaWAN.

Firstly, there are gateway-related functions such as scheduling and managing the utilization of the gateways. Scheduling is needed because a gateway can only do one transmission at the same time. The information is used to evenly distribute load over different gateways and to be compliant with the related region duty cycles [3] e.g. Europe.The greatest duty-cycle, characterized as the most extreme rate of time amid which an end-device can possess a channel, could be a key limitation for systems working in unlicensed bands.

Another important issue is monitoring the status of each gateway. Secondly, a device-related functions that manage the state of devices in the network needed. As device address are non-unique, the network has to keep track of which addresses are used by which devices in order to map a message to the correct device and application. Moreover the network must keep track of are the security keys and frame counters. Then it will also start keeping track of the network utilization of each node. Thirdly there is some functionality related to applications. For example, the network server need to know to which server traffic for a specific application needs to be forwarded.

LoRa concentrator/gateways are intermediaries that allow sensing devices to transmit data to the lora backend. LoRaWAN uses some identifiers for devices, applications and gateways such as: DevEUI - 64 bit end-device identifier, EUI-64 (unique) DevAddr - 32 bit device address (non-unique). finding the actual device that belongs to that address is done by matching the cryptographic signature (MIC) of the message to a device in the database. AppEUI - 64 bit application identifier, EUI-64 (unique). When you create an application, the server allocates an AppEUI from the address block. GatewayEUI - 64 bit gateway identifier, EUI-64 (unique). The network server assigns device addresses to devices (based on configuration). Devices which use ABP activation mode have to request an address from the network server. For devices use OTAA, the network server will assign an address when the device joins.[4]

[3]https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/

[4]https://lora-developers.semtech.com/documentation/tech-papers-and-guides/semtech-network-

Applications running on gateways. They take care of sending demodulated radio packets arriving at the gateway to a network server and transmitting packets coming from the same device via a built in LoRa transceiver. Semtech packet forwarding is based on the connectionless UDP Protocol. Messages are sent to the application listening on the server with IP-address and port number specified in the destination fields of the datagram. Also source fields with IP-address and port number of the sending application exist. There are no mechanisms in the form of acknowledgement or retransmission to avoid packet loss. As this protocol was implemented in the first packet forwarder publicly available most gateways include a basic packet forwarder running this protocol.[5]

server-user-guide/find-your-end-device-activation-keys/

[5]https://github.com/Lora-net/packet_forwarder/blob/master/PROTOCOL.TXT/

Chapter 2

IoT Security

Recently, a wide adoption and deployment of IoT infrastructures and systems for various crucial applications such as logistics, smart cities, transportation, manufacturing and healthcare drawn, so evaluation of threats of IoT network and possible countermeasures has become emerging. Due to a large number of IoT attacks, it is important to categorize them to have a better understanding to appropriately provide solutions to detect and neutralize such threats. First we reviewed the state of the art surveys and solution in IoT attacks, further we classified them based on OSI-layers. Totally some of the most effective approaches addressed which trace an attack by analyzing the IoT network traffic and behavior of each device but here concentration is on the vulnerabilities on the communication layer.

Privacy is also a serious concern among users. For instance, Nest smart cameras, Google's solution for IoT indoor and outdoor surveillance, were recently targeted by hackers. After compromising a Nest baby monitor, an attacker did broadcasts of voice messages threatening parents, who experienced a dreadful situation, feeling invaded and uncomfortable [8]. Future security solutions for IoT will need to take into account that IoT devices with vulnerabilities may often be present in the user's network and co-exist with other devices during their whole device lifetime. IoT devices usually connected to a gateway router offering wireless and wired interfaces for connecting IP enabled devices to the network. The adversary's goal is to exploit IoT devices to either i) ex-filtrate data, security credentials or encryption keys, ii) compromise other IoT devices in the network with the help of a compromised device, or, iii) inject false or tampered information into the user's network [9].

Securing IoT is especially complicated as smart devices target:

- Low power in computation
- Low memory capabilities
- Low communication resources
- Low battery-powered
- Have limited user interface
- Closed devices
- Heterogeneous
- Distributed architectures
- Low maturity

2.1 Security Threats, Vulnerabilities and Countermeasures in IoT Systems

Important types of attacks are those which performed through Botnet. According to [10] in IoT systems malicious bots, to avoid detection, encrypt their information inside the code moreover encrypt their communication channels. Sink-holing is a process where the attack traffic is filtered using packet filtering upstream from the victim. Based on Deogirikar et al. [11] research regarding Sinkhole attack, if the node authentication is supplied, an attacker cannot compromise the node. According to their survey most of IoT systems uses IPv6 because IPv6 allows billions of addresses, which is sufficient to provide IP address to each object instead of each device. They also did their work in the domain of IPv6. They found that it is difficult to implement a solution for each attack because of computing and battery power constraint. Each attack has done on a significant layer (Application, Network, Physical). Authors mentioned the prominent section in side-channel attack is cryptanalysis of a Simple Modular Exponentiation where Diffie-Hellman and RSA operations involve calculation of R = y mod n, where R is RSA key, n is public key and y is a key which can be obtained by a listener. The adversary wants to search the secret key. As countermeasure they used a monitoring verification scheme (MOVE) which is able to check the monitoring node(s)' result and correctly identify any malicious behavior.

Some countermeasures and tactics such as restricting forged (spoofed) traffic, preventing hosts from being used as reflectors, managing routing advertisements are all useful in preventing some types of DDoS attacks. Moreover gating to weakly configured Internet of Things (IoT) devices.

Miettinen et al. with their IoT SENTINEL [9] aimed to directed all requests from clients to an Intrusion Detection System(IDS). Legitimate requests from clients pass through the IDS onto the server. Anirudh et al. [12] show that if the IDS detects any anomalies in the requests (Example: Spam requests to initiate a DoS attack), the requests are passed onto the honeypot and the information related to the attacker (IP Address, MAC Address, etc.) are stored as logs in a database. Hackers usually find and shut down the honeypots in the IoT network consequently the solution is a random set of servers is used as honeypots. They also mentioned that generally these kinds of attacks are concentrated towards the main server rather than the individual devices connected in the system. The main reason is that it is easier to access the main server rather than the individual devices because the protocol for data transmission in each device might be configured differently and also another main reason is that, by crashing the main server the whole system is supposedly shut down since another potential user can't access the main server to link to any other device.

According to the acknowledgment, the versifier node will decide whether the node is malicious or not. However implementation of all these security measures and techniques together consumes computation as well as battery power of devices which is not acceptable for IoT technology and its devices. They provided a comparison table contains possible attacks, vulnerabilities and their countermeasures. On the Data

Link layer, attacks are classified as collision, resource exhaustion, and unfairness. The key part in Zhang et al. [13] is the application of Hidden Markov Model (HMMs) which was found relating to Sybil attack detection, where semi-supervised learning with HMMs was successfully implemented in their model. The attack graph can be constructed from network configuration, network and firewall logs. An Analyzer in Mohsin et al. research [14] uses state machines to determine attack likelihood and costs for an IoT network given attacker capabilities, vulnerability scores and configurations. The different individual vulnerabilities are connected together to find attack paths, using network configuration, firewall and network logs from more than twenty tools. All the previous works did not capture generic attack typologies simultaneously with threat provenance in large scale IoT network setting.

In advance, Mosenia and Jha [15] utilized the Cisco seven-level reference scheme [16] to display different comparing attack scenarios. They investigated various IoT and focused on attacks and pinpointed their conceivable moderation approaches. The authors highlighted the significance of having a proactive approach for securing the IoT environment. They introduced communication level in edge-side layer. The communication level consists of all the components that enable transmission of information or commands: (i) communication between devices in the first level, (ii) communication between the components in the second level, and (iii) transmission of information between the first and third levels (edge computing level). They focused on this prominent fact that when packets also carry access control information, such as node configuration, shared network password, and node identifiers, eavesdropping can take place for critical information. In communication level, with injecting fraudulent packets threat an attacker can access the communication links using three different attack methods: (i) insertion, (ii) manipulation, and (iii) replication (replay).

In previous studies countermeasures against networked scenario threats has been done using IDS e.g. IoT SENTINEL [9], MOVE [11], SVELTE [17].

In [18] authors mostly referred to the side channel attacks and physical layer threats. IoT devices vendors generally do not send updates and patches to their devices unless user-initiated firmware updates so this is also a threat for IoT end nodes. Due the IoT device behaviors could be changed with other devices or environmental conditions, it is difficult to define a certain set of fine-grained permission rules for them. The protocols designed by IT companies may have many potential security problems. Moreover, different protocols have different semantic definitions, the attackers also could use this point to find security vulnerabilities like BadTunnel [1] when they uncorrected work together. Yin et al. [19] design honeypot and sandbox system to collect attack samples from IoT devices, and found the most remote network attack.

Threats can be derived by: (i) physical access (ii) networked scenario, (iii) Software.

Physical access: It might happens usually if IoT devices operate in an unattended fashion with no or limited tamper resistance policies and methodologies. The majority of IoT devices operate autonomously in unattended environments with little effort, an adversary might obtain unauthorized physical access to such devices and thus take control over them.

Threats related to networked scenario:

- Eavesdropping attack: if the communication channel (often a wireless medium) is not adequately protected keying materials or security parameters/protocols not correctly set-up.
- Man-in-the-middle attacks: particularly during key establishment and/or device authentication procedure device authentication/authorization may be not automated and need support of a human decision process, since things usually do not have a prior knowledge about each other
- Routing attack: routing information in IoT can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/ shorten source routes, etc. relevant routing attacks may include:
  - Sinkhole attack (or Black-hole attack): an intruder declares himself to have a high-quality route/path to the base station, thus allowing him to do

[1] https://www.blackhat.com/docs/us-16/materials/us-16-Yu-BadTunnel-How-Do-I-Get-Big-Brother-Power-wp.pdf

anything to all packets passing through it

  - Selective forwarding: an attacker may selectively forward packets or simply drop it
  - Sybil attack: an attacker presents multiple identities to other things

- Privacy threats: an attacker can gather and track thing's information, concluding behavioral patterns

- Denial-of-Service attack: typically, things have tight memory and limited computation, they are thus vulnerable to resource exhaustion attack attackers can continuously send requests to be processed by specific things so as to deplete their resources this is especially dangerous in the IoTs since an attacker might be located in the backend and target resource-constrained devices in an LLN (Low power and Lossy network). This attack can be launched by physically jamming the communication channel thus breaking down the communication channel network availability can also be disrupted by flooding the network with a large number of packets (Flooding attack).

- battery-draining attack is in result of DOS attack: Power consumption is an issue of some scenarios that attack can easily succeed in consuming the device resources.

Security Countermeasures in IoT Systems:

General security policies secure communication protocols and cryptography algorithms to enforce the following security services:

- Availability: It is necessary that the services, devices and interfaces are always available for the end client.

- Peer authentication/authorization: Data protection (authentication/integrity, confidentiality): Using proper cryptography tools symmetric block ciphers, hash functions, asymmetric cryptography (Elliptic Curve Cryptography is a well-known encryption algorithm suitable for IoT systems). In addition a complete key management infrastructure is fundamental to handling the required cryptography keys complicated in IoT scenarios than in standard Internet secure routing protocol.

- Anonymity: Is a security service allowing entities to communicate with each other in such a way that no third party knows that they are the participants of a certain message exchange. Hiding the fact that communication is taking place between certain source and destination entities is the biggest challenge of preserving privacy.

Countermeasures against physical threats:

- security policies
- protect smart objects in safe places, when possible
- safe supplying and installation measures
- avoiding untrusted manufacturers and installers

Countermeasures against network and software threats:

- secure communication protocols and cryptographic algorithms
- robust authentication and key management
- vulnerability Assessment
- Honeypots
- Intrusion Detection systems (IDS)
- Machine Learning-based Network IDS

2.2 Security in LoRaWAN

In [20] authors mentioned LoRaWAN important practical threats, such as end-device physical capture, rogue gateway and self-replay, which need special attention by developers and organizations while deploying LoRa network.

Threats related to the networked scenario and physical access specified in LoRaWAN technology:

- plain-text recovery
- malicious message modification (bit-flipping)
- falsification of delivery reports
- hello flood
- ack-spoofing can be executed through selective jamming attack
- Jamming
- battery exhaustion attack

  Threats related to the physical access to smart objects: Extraction of security parameters from physically unprotected devices if a group key is used and compromised this way, the whole network may be compromised as well
- Firmware Replacement attack: when a thing is in operation or maintenance phase its firmware or software may be updated to allow for new functionality or new features Cloning of smart things by untrusted manufacturers and/or Malicious Substitution of things during the installation

Countermeasures for both network scenarios and physical access in LoRaWAN technology:

- Link-layer encryption and authentication through AEAD coverage (Authentication encryption for associated data)
- multi-path routing

- identity verification
- authenticated broadcast
- bidirectional link verification
- firmware update

The LoRaWAN protocol stack is implemented on top of LoRa modulation (PHY). Security Challenges of LoRaWAN include:

- Network Entities based such as Gateway, Servers, End node,
- Key Distribution based such as ABP based, OTAA based,
- Implementation based includes Exit procedure, DevEUI, Frame counters, JoinEUI, Secure Storage.

However, all communications are done with a dynamic 32 bit device address (DevAddr) of which 7 bits are fixed, leaving 25 bits that can be assigned to individual devices, a procedure called Activation. According to the LoRaWAN join procedure there are two different modes that define or compute keys for MAC frame payload encryption which are activation procedures:

- OTAA (Over-the-Air Activation): During this procedure a dynamic DevAddr is assigned and security keys are negotiated with the device.
- ABP (Activation by Personalization): In some cases you might need to hard-code the DevAddr as well as the security keys in the device. This means activating a device by personalization (ABP). This strategy might seem simpler, because you skip the join procedure, but it has some downsides related to security.

Regardless of the mode used with LoRaWAN communication, messages are protected by two session keys, AppSKey and NwSKey, which are used to encrypt messages in Counter with CBC-MAC (CCM) mode, a variation of Counter (CTR) mode.

CTR mode is used to encrypt the payload using AppSKey and authenticates the message with a Cipherbased Message Authentication Code (CMAC) based on the NwkSkey.

According to [21] LoRaWAN is secure by design using encryption, integrity and authentication on both the network and application level. Join Servers maybe used to safely store LoRaWAN keys and issue session keys to the Network Server and Application Server. This decouples secure storage from packet routing, allowing users to host Join Servers on-premises and using hardware secure modules (HSMs) to stay in full control over security keys. Once an end node has joined a LoRaWAN network, either through OTAA or ABP, all future messages will be encrypted and signed using a combination of NwkSKey and AppSKey. As the NwkSKey key is only known by the network server and specific end node, as well as the AppSKey key is only known by the application server and the end nod. They mentioned how a node join to the LoRa network, and how keys transfers in detail. Keys are distributed in one of two ways depending on how an end node joins the network.

In [22] authors mentioned LoRaWAN v1.0.2 uses a pair of two distinct keys, the network key NwkSKey, and the application key AppSKey As the communication channel is wireless and thus available to anyone for injection and modification, also the authenticity of communication in other words do the packets indeed originate from the alleged source and the protection against originally legitimate but maliciously re-injected traffic become a concern. Raise the issue of device and key enrollment(key establishment) is necessary. As IoT devices are produced and stored in bulk and shipped to an enduser who would like to connect a device to his or her account, downloading a device/user/application-specific key to the unit becomes necessary. Here authors described that the approach LoRaWAN takes on each of these aspects. Establishment protocol (enrollment protocol) to activate the end device in the network via OTAA or ABP. Threats eavesdrop and decrypt the content of a frame under certain circumstances. Second, authors show that the content of a packet may be modified outside of the integrity check provided by the protocol. Third, authors highlight that messages could either be replayed, or a node tricked into believing that a message has been received by the gateway when it actually has not, and out-line a battery exhaustion attack. This compromises the availability of the network. They also mentioned threats in ABP method. They designed and described threats (5 attacks):

- replay attack leads to the selective denial-of-service on individual IoT devices,
- plaintext recovery
- malicious message modification
- falsification of delivery reports
- battery exhaustion attack

Miller group [2] describes the location of the key material in a LoRaWAN setup, and alerts that flaws in key management could compromise a backend. Miller did not analyze the protocol nor evaluates the security of message exchanges. Tracing and finding DDOS attacks in LORA networks is not detectable because the attack leaves no abnormal adversarial traffic such as flooding which is detectable by the network. They opens widely and in detail completely how the attack accomplished in practical by them selves practically these attacks: replay attack, eavesdropping, bit flipping, ACK spoofing. Then redo all the above scenarios for LoRaWAN class B.

2.2.1 LoRaWAN Threats

- False Join Packets: In implementation related vulnerabilities, renewal of JoinEUI and DevEUI values were reported to be problematic. However, even they are old in value, they can still be used for MIC: Join-Request and Join-Accept messages are protected with MIC (by using JoinEUI and DevEUI values) and also with another unique nonce (JoinNonce) value, it is quite unlikely this attack to happen.
- MITM Attacks: Bit-flipping or Message Forgery Attack: Especially, a specific version of MITM attack called "bit-flipping attack", in which an adversary (or a

[2]https://labs.f-secure.com/archive/lo/

rogue network server) changes the content of the messages in between NS (network server) and AS (application server). Bit-flipping attack still constitutes a threat for LoRaWAN v1.1 as also declared in the specification document.

- Frame Payload Attack: Handover-roaming enables more possibilities for a MITM attack, as the unprotected FRMPayload's are first transported from the serving-NS to the homing network servers, and from there to the AS. Henceforth, network servers are considered as trusted entities by default. However, deploying engineers are recommended to use extra precautions (end-to-end security solutions) if they are wishing to have end-to-end confidentiality and integrity protection against MITM attacks.

- Network Flooding Attack: End-devices can be captured and used to perform attacks against the rest of the network. For instance, it is possible for an end-device to degrade the network by flooding it with packets. Nevertheless, end-devices should comply with regulatory airtime restrictions and some networks might protect themselves from flooding attacks by imposing added airtime restrictions.

- Network Traffic Analysis: This is a passive attack and also called eavesdropping attack if happens in the physical layer (radio signals). In this kind of attack, an attacker can setup a Rogue-GW by using the vulnerabilities mentioned to receive packets and deduce some knowledge about the data being transmitted or the keying material being used. Observe that, without access to key material, it is infeasible for the attacker to be able to decode the contents of the packets received and the usefulness of this traffic analysis will be highly application specific. For instance, a LoRa network deployed in a smart-city application to deduce noise levels in the city might leak information related to the level of activity in certain locations through simple observation of transmissions.

- Physical Attacks: Network entity and implementation related vulnerabilities of LoRaWAN can be exploited by the adversaries to perform the following attacks:

1. Destroy, Remove, or Steal End-device: In both versions of LoRaWAN, root keys (from which session keys are generated) are uniquely generated for each device during fabrication or before deployment. Therefore, revealing of a single root key will not compromise any information in the network other than the data stored at that specific device.
2. Security Parameter Extraction: The LoRaWAN specification requires protection of the relevant key material against reuse, but nodes need to be adequately protected against firmware change that might indirectly lead to key material reuse.
3. Device Cloning or Firmware Replacement: An attacker with physical access to the device and can replace firmware or steal/reuse key material.

- Plaintext Key Capture: If the secure storage elements are not used and keys are stored in regular files (such as text files) in the ED's, then this attack can be a major threat for the confidentiality, integrity and availability of LoRaWAN network.

- RF Jamming Attack: In the past, it has been shown that by using low cost and commodity hardware jamming of the RF signals is possible. RF jamming attacks in wireless networks lead do Denial-of-Service (DoS), which are generally easy to detect. However, selective RF jamming attacks are harder to detect an very harmful for wireless communications as it is not trivial to avoid. Therefore, in LoRaWAN, it is possible to jam reception of the signals at a gateway or a node by using RF jamming. This could open-up some advanced attacks as it is described in the Replay attack, which partially relies on the ability to perform selective RF jamming.

- Rogue End-Device Attack: To be part of the network, an end-device needs access to key materials that should not be accessible to malicious users. One way to do this is by capturing a device and replacing its firmware such that it reuses key materials in the captured device (as discussed in physical-capture-attack).

End-devices are data sources and are not authorized to reach information at servers. Therefore, any rogue end-device with legitimate authentication information will not able to access or leak data from the network. The most harm can be done by injecting false data to the application server. End-devices can also be exploited by the attackers to be used as jammers in the network. This might cause the LoRaWAN network to be not available for legitimate end-devices (DoS). However, this will be a restricted area in the vicinity of the Rogue end-device, and the rest of the network continues regular operation. Rogue end-devices can be also used to perform replay attacks. Packets being transmitted by the neighbors can be captured and replayed later on, which generally will be detectable due to the use of different nonce values (e.g., DevNonce, Join-Nonce). However, this might cause waste of available resources in the network and decrease the availability of the gateway for the legitimate end-devices.

- Rogue Gateway Attack: From the beginning (including LoRAWAN v1.0), the gateways are always considered as legit and obeying relays. However, this assumption raises the question: "What happens if adversaries hack/capture/replicate one or multiple gateways (GWs) are vulnerable and therefore can be a target for persistent attackers as described:

  i) Beacon Synchronization DoS Attack: Class B sessions might be vulnerable to Rogue-Gateway attack, which is in line with Session Hijacking Attacks. In LoRaWAN, Class B beacons are not secured by any means, indicating that an attacker can set up a Rogue-GW to send fake beacons. This could result in class B end-devices to receive messages in windows out-of-sync with the Rogue-GW and also increased collisions on packets being transmitted. Addressing this threat would require issuing a key that GWs could use to authenticate beacon transmissions.

  ii) Impersonation Attack: GWs can also be impersonated to set-up attacks against end-devices (EDs). EDs can be listened to and their network addresses can be determined. More importantly, a triangulation method (minimum 3 GWs are needed in this case to perform the intended capturing attack towards the ED) can be used to determine the physical location of the EDs.

- Routing Attacks: Mainly two types of attacks can be considered under this category: i) Selective Forwarding Attack: In this attack type, an attacker can selectively forward packets and can cause one or several nodes in the network to be totally blocked, or totally overwhelmed. ii) Sinkhole or Blackhole Attack: In these attacks, an attacker attracts the traffic through itself by falsely advertising modified routing information. As a result, whole network traffic might collapse. In LoRaWAN, in order routing attacks to be successful, an attacker needs to capture a GW or a server, which is a non-trivial and complicated task. These attacks cannot be perpetrated from the EDs, as they do not participate in routing. Therefore, the probability of this attack to happen is very Low

- Self-Replay Attack: Although OTAA is devised to enhance the security level of the LoRaWAN, itself can be the target of attacks, as details described below: Attacks that are exploiting the join procedure of LoRaWAN v.1.1 is possible by using selective RF jamming attack. An advanced attacker can selectively jam the signals that are being used for OTAA session. In this type of attack, the transmission of a Join-request with DevNonce from a legitimate ED is successfully observed. The corresponding Join-accept message from the NS to ED is jammed by using selective-jamming techniques. After waiting for a timeout to receive the Join-accept message from the NS, the ED retries to join the network and sends again the same Join-request message with the same DevNonce value as required by the specification. This causes the NS to respond the Join-request, because still the join procedure is not fulfilled and everything is legitimate and in order according to the specification. This attack will continue till the daily message quota of the ED depletes.

This attack is especially valid for LoRaWAN OTAA, since the communication of the messages is limited and under quota. For instance, each ED is allowed to transmit at most 14 packets per day (maximum packet payload of 12 Bytes), including the acknowledgments for confirmed up-links. In order this attack to be fully successful, it requires that the attacker can receive packets from the NS before the ED does and also, at the same time, jam their reception by the ED. This can be achieved, for example, by having a device receiving (or sensing) the packet from the NS outside the interference range of the jamming device and jam the signal before it reaches the destined ED, which might not always be possible depending on the transmission range and distance to the nearest GW. This attack is made substantially more difficult for the attacker with the existence of several receive paths for the NS packets to the ED. Thus, if the NS's transmissions to ED can be provided by multiple GWs (let us say if the ED is under coverage of multiple GWs), this attack will be extremely difficult to perform. However, GWs are typically deployed fewer in numbers in a LoRa network and therefore this attack has a possibility to succeed.

Moreover Network attack surfaces is well protected in LoRaWAN (in theory)

- AES security performs both authentication/integrity protection and data confidentiality, from end-device side to application server side
- security is based on AES secret keys (long term keys) that have to be shared between the device and the application server possible issues depend on how these keys are managed problem when the devices have to be reused with a different application server
- security of the overall application depends also on the application behind the application server in LoRaWAN backend

2.2.2 Threats in OTAA Mode (Over-The-Air Activation)

If the end-device supports the Join function and can store dynamically generated keys, a join procedure can be performed to compute keys for encrypting and protecting packet integrity. The process of computing new keys:

- The end-device sends a Join Request
- The network will generate keys
- If the device can Join the network, a Join Accept message is sent by the network encrypted with AppKey, providing an App Nonce

- With the given parameters sent via the Join Accept message, the end-device can compute the new encryption and integrity keys

According to the LoRaWAN 1.0.3 specifications, the Join Request is sent in clear-text to the gateway with the following parameters:

- DevEUI: unique end-device identifier in IEEE EUI64 address space, Deveui is assigned to the device by the embedded chip manufacturer. (Usually represented in the form HH:HH:HH:HH:HH:HH:HH:HH where HH are pairs of hex digits)

- AppEUI: the application identifier in IEEE EUI64 address space which is a random DevNonce of 2 bytes.

From the LoRaWAN 1.0.3 specifications, DevNonce values are tracked to avoid replay attacks. The message integrity code (MIC) for this message, which also enables the network to check if the AppKey is correct, which computed as follows:

$$cmac = aes128\_cmac(AppKey, MHDR|AppEUI|DevEUI|DevNonce) \tag{2.1}$$

$$MIC = cmac[0..3] \tag{2.2}$$

If the device is allowed to join the network and the MIC is correct, the network sends an encrypted Join Accept message with the following fields:

AppNonce in LoRaWAN 1.0: random value (3 bytes)

NetID, called Home_NetID in 1.1 network ID (3 bytes)

DevAddr Device ID (3 bytes)

DL Settings: downlink parameters

RxDelay: delay between TX and RX (1 byte)

CFList: optional list of channel frequencies (16 bytes)

The Join-accept payload is encrypted as follows:

aes128_decrypt(AppKey, AppNonce | NetID | DevAddr | DLSettings | RxDelay | CFList | MIC)

The sessions keys are computed in the network and end-device sides as follows:

All the communications will then be encrypted using AppSKey, and the integrity protected with NwkSKey. Nevertheless, only one AppKey key is used to compute the MIC as well as AppSKey and NwkSKey, which leaves a bigger opening for an attacker to crack either the MIC of the Join procedure's message or the Join-accept message to retrieve this key. Retrieving the key could allow an attacker to eavesdrop on the messages between an end-device and a gateway.

2.2.3 Threats in ABP Mode (Activation by Personalization)

The ABP method is simpler than OTAA as there is no Join Procedure. Nevertheless, it has downsides in terms of security, as session keys are hardcoded on versions 1.0 and 1.1. Indeed, session keys stay the same until the user manually changes them or when a firmware update/upgrade is applied. Because of this, ABP is more vulnerable to a cryptanalysis attack compared to OTAA.

MAC Frame Payload (FRMPayload) Encryption: With given session keys, we can protect MAC frame payloads on Data Up and Down messages. The key "K" is chosen depending on the FPort (Port field) of the data message:

For the encryption, the message is split into a sequence of blocks:

$$A_i \, for \, i = 1..k \, with \, k = ceil(len(pld)/16)(with \, pld = FRMPayload) \tag{2.3}$$

So, the block A_i is encrypted as follows:

$$S_i = aes128\_encrypt(K, A_i) \, for \, i = 1..k \tag{2.4}$$

$$S = S_1|S_2|..|S_k \tag{2.5}$$

Encryption and decryption of the blocks are then performed as follows:

$$(pld|pad16) \, xor \, S \tag{2.6}$$

The MIC is computed as follows:

$$msg = MHDR|FHDR|FPort|FRMPayload \tag{2.7}$$

$$MIC = cmac[0..3] \tag{2.8}$$

2.2.4 Threats Against LoRaWAN

Most of the attacks documented below have been performed on LoRaWAN class A devices. However, a class B attack is also possible; these are usually done to drain the end-device's battery.

DoS in ABP mod:

Given that the counter in FRMPayload is only 16 bits long in LoRa 1.0, a denial-of-service (DoS) attack is actually possible. In [22] authors mentioned that a malicious actor could replay a captured packet, wait until the counter overflows, and replay this packet to realize a DoS attack.

Session keys stay the same until manually changed or with a firmware update/upgrade. Because of this, ABP could be more vulnerable to a cryptanalysis attack than OTAA.

Eavesdropping:

Aforementioned authors also highlighted that the key could be retrieved if the counter is reset, as it is not used securely in LoRaWAN 1.0. Indeed, when the counter is reset, the keystream will be reused and could allow an eavesdropping attack. Another factor that leaves it vulnerable is the use of CTR to encrypt information. The same researchers also highlighted a bit-flipping issue in v1.0 that is backward compatible with v1.1.

Bit-Flipping:

Regardless of the version, the integrity of LoRaWAN messages is protected thanks to the MIC and encryption. Nevertheless, the messages decrypted by the network server and sent to the application are no longer protected.

To protect messages from the network server to the application server, it would require designing the network to use an SSL tunnel, preferably with a client SSL certificate in order to protect the communication or by using another MIC field inside the MAC Layer Payload computed by the AppSKey.

Ack Spoofing:

An acknowledgment mechanism was introduced in LoRaWAN to maximize battery life by reducing the time the radio needs to be powered up.

To illustrate this issue, researchers proposed to selectively jam the downlink when an end-device sends a confirmed message to the gateway that will confirm the reception. The confirmation message will never get to the end-device, and the confirmed message will be retransmitted seven times. Then, the message will be considered lost or refused.But during the jamming session, the attacker can capture the downlink message for the confirmation and can play the confirmation for the first message when the end-device sends a second confirmation message.

Beacon (Class B) Synchronization Attack:

Most setups use the class A mode, which specifies that downlink traffic must/can follow an uplink one. Class B tends to reduce the amount of spent energy by telling end-devices to wake up periodically to wait for any incoming messages during a receiving window. The durations of the receiving windows are specified by beacon broadcast messages that are comprised of a PHY layer header followed by a beacon payload.

An attacker can easily compute the different publicly known fields with malicious parameters to: i) Find the location of a LoRa gateway.

ii) Drain the battery by sending crafted beacons framed with an extreme wakeup time value. It would be better to use a MIC instead of a CRC to check the time's field integrity.

Threats in Root Key Management:

Indeed, the backend is exposed to the internet, which leaves it open to attacks (LFI, SQL injection, deserialization vulnerability, etc.). A malicious actor would be able to get the secret key, read the data, craft downlink packets, and more. Here are some security points to check in a LoRaWAN setup:

- Use randomly generated keys
- Avoid the exposition of key management servers and services (exposed key management service accessible on the internet)
- Preferably use HSM (Hardware Security Module) to keep the keys

- Preferably use OTAA mode and LoRa version 1.1

2.2.5 The State of Security

A few security tools exist towards the LoRaWAN technology, released starting this year. Some researchers released a framework called LAF (LoRaWAN Auditing Framework). They also provided a tool that can parse, send, craft, analyze, audit a setup, and crack some LoRaWAN 1.0.3 packets using weak/default keys. Nevertheless, this framework still has some limitations: i) It only works with a Gateway ii) It can only listen to uplink packets iii) It can only listen to eight out of 64 channels iiii) Generation and fuzzing depends on LoRaWAN (Go) using an inflexible format such as JSON

Over the same period, another type of tool called "LoRa Craft" was released to intercept packets using Software Defined-Radio and craft packets using dedicated LoRaWAN v1.0 and v1.1 Scapy layers developed for this tool. But this tool is mainly a do-it-yourself (DIY) tool and needs much more assistance than those already released, like the crypto-helpers for Join-Accept payloads and MIC to help crack weak keys.

Later another framework called ChirpOTLE was released by the SEEMOO Lab. This demonstrated two attacks affecting the availability of LoRaWAN networks, like time drifting in LoRa class B and a novel ADR spoofing attack to manipulate frame metadata. But still, the researcher's setup was limited since they only chose a few default channels to demonstrate their attack on each node.

2.2.6 Security Countermeasures to Improvement LoRaWAN

In this section, we provide our suggestions for the further improvement and distribution of LoRaWAN as follows:

Usage of Public Certificates for Generation of Root Keys at the End-Devices: Secure provisioning, storage and usage of root keys are very important. A solution to this might be the use of PKI server (Public key infrastructure), can be installed to either join server (JS), AS or NS, so it can be used as a certificate server. Certificates from the PKI server can be pre-installed during or after the fabrication of the end-devices along with the serial numbers, nonces, and other counters. Root keys can be used by using this certificate and along with an authenticated key-exchange algorithm such as Authenticated Diffie-Hellman key Exchange (ADHE) algorithm In this scheme, end-device revocation and/or key-update/key-redistribution is an easy task compared to the manual update of the keys (root keys). This topic is very important as it will enable the renewal of the root keys when needed, hence will provide flexibility to the network operator while adding/revoking keys.

Nonce Related Improvement: Each end-device may have nonce which might be associated with the serial number of itself. The NS holds the list of nonces (Nonce-List) and revokes each nonce whenever its used first time. Nonces can be updated by hand-shaking, after successful key initiation phase. Both NS and the end-device might agree on a new nonce and that nonce might be written to a non-volatile memory of the end-device. The old nonces would be moved to the "used nonces list" and the new nonce should be moved to "valid nonces list". By this way, any attempt to use an old nonce would not cause any security breach in the network. Besides, these attempts can easily be detected by the NS by comparing the nonce with the nonce lists. The join request can be granted if only the submitted nonce is in the valid nonces list otherwise, if the nonce is in the used nonces list, then this is a possible situation of cloning, node-capture or replay attack.

Introduction of End-to-End Encryption between Servers: This topic has prime importance, as it will completely prevent any kind of man-in-the-middle attacks towards servers. Although it is mentioned in the LoRaWAN v1.1 specificationthat the communication in between the servers is considered as secure.

Inclusion of Padding: To divert packet-sniffing attacks, it is always a good practice and counter-measure to set packet length to a certain constant value and apply padding when needed. Eventually, this will come with a price of increased airtime of the added extra padding data.

Packet Count Limitation: This opens a venue for attacks for LoRaWAN, since the communication per ED is limited and under quota. For instance, each ED is allowed to transmit at most 14 packets per day (maximum packet payload of 12 Bytes), including the acknowledgments for confirmed up-links.

Time Synchronization: Hence time synchronization of the EDs is provided within v1.1 this can be efficiently used as a time-stamp to improve the freshness and integrity of the messages.

Gateway Related Improvement: An authentication mechanism for the GWs is necessary in order to prevent the network from Rogue-Gateway attacks. For instance, mutual authentication can be performed between the couples of ED-GW and GW-NS.

Server Trust Related Suggestion: In LoRaWAN, servers need to be trusted entities otherwise, they can create single point of failure for the network. By default, all the servers in LoRaWAN are assumed to be trusted entities as mentioned in the specifications.

End-Device Related Improvement: Physical capture threat against end-devices (and eventually extraction of security parameters) can be neutralized by usage of tamper-resistant hardware for the storage of the keying materials in the end-devices. Although this comes with an extra cost in safety-critical installations where Cyber threats need to be mitigated, the use of tamper-resistant hardware should be mandatory

2.2.7 Key Management Mechanism

Regarding the key management of root keys (AppKey and NwkKey), a practical issue is that these secret keys must be shared with the backend platform. In LoRaWAN 1.1 two keys have been introduced (in place of just one root key) in order to reduce the number of systems that know the key used for encrypting the payload (AppSKey). In fact in LoRaWAN 1.1 the AppSKey is derived from the AppKey that can be shared just with the application server (and of course the device). The other keys used for integrity protection and for joining the network are instead derived from the NtkKey. However, in different practical cases the application server is not owned by the same entity that owns the device (for example when the device is attached to third party backend system like TTN). In this case the AppKey is disclosed to an entity (the application server) that eventually may not be trusted any more (for example if the user want to change the backend system provider). In this case the original root keys cannot be considered secret any more. In order to overcome this issue, the following possible solutions can be used:

- Change the root keys: However this requires the change of these key stored in the device. This can be complicated or impossible due to some limitations in device configuration (no configuration interface is available), or in device access (the device is not physically accessible and secure Over-the-air (OTA) configuration is not available)
- Introducing an automatic root key renewal mechanisms: We experimented this approach by letting the device to periodically update the root keys (in our case the single AppKey). The key is updated using a deterministic algorithm based on a secret master key. At the same time a proper client application is introduced by the the device owner that periodically updates the root keys on the current backend platform (in our case we used TTN and the corresponding TTN REST API for the updating the AppKey)
- Adding an extra encryption layer for end-to-end security: In this case the payload passed to the LoRaWAN upper layer is encrypted using a symmetric key share only between the actual application endpoints, i.e. the data provider (e.g. a sensor) and the data consumer, both administrated by the same entity or two reciprocally trusted entities. In order to minimize the cost of this extra encryption operation, the same cipher loRaWAN payload encryption algorithm (AES EC) can be used

We just implemented the last aforementioned case. A current demonstrator consists of:

- a virtual LoRaWAN device configured with an additional DataKey. Data passed to the LoRaWAN client is first encrypted using this key
- a virtual LoRaWAN gateway that is used by the device to connect to TTN
- an application that uses the TTN REST API to receive device data. The received payload is further decrypted using the same DataKey and finally showed in clear-text to the user.

Chapter 3

Anomaly Detection in IoT Systems

According to the studies in Sec. 2 authors attempted to provide a comprehensive list of vulnerabilities in IoT systems. In their researches it is understandable that the vulnerabilities existed in the communication layer are more than physical layer and application layer, so most attacks have hit this layer, thus we focused on this category to analyze the network layer. In addition based on the surveys in [23, 24, 25] we found that the most effective and light strategy to combat threats in IoT heterogeneous network is data analysis model based on machine learning by network traffic analysis in IoT systems. Finally we proposed a novel security countermeasure based on state of the art machine learning models.

As most founded attacks have been done on the network layer. IDS systems used to discover vulnerabilities in the network layer. Machine-learning-based-IDS systems are more beneficial in comparison with the traditional IDS for Anomaly Detection and Deep-learning-based-IDS is one of most proper effective approach in terms of Anomaly Detection on Traffic classification. Neural network architecture one of most successful between Deep learning models for traffic classification in IoT network. Moreover it would be able to detect also zero attacks [1] too.

By study recently done researches on traditional IDS systems following achievements obtained:

[1]Attacks which are not recognized by security specialists and systems yet in the world

- Machine-Learning-based-IDS systems are more beneficial in comparison of the traditional IDS systems
- Deep learning is a novel technique in IDS systems which is able to provide our needs in IoT
- Limitations of methods in the literature, even those based on artificial neural networks

3.1 Traffic Classification

For traffic classification and anomaly detection, usually bi-directional network flows are considered. These are composed of a series of ordered packets, exchanged between two terminal points, and are uniquely identified through the following quintuple: *source IP address*, *destination IP address*, *source port*, *destination port*, *transport protocol*. Source and destination ports and addresses may be pairwise interchangeable and identify the two single main unidirectional sub-flows (from source to destination and vice versa) a flow is made of.

Internet traffic can be captured by using standard network sniffers like tcpdump[2] and Wireshark[3], or network emulators [26]. They permit one to get traffic traces, composed of various packets belonging to different sessions or flows, flowing inside private or public networks.

Historically, the main traffic classification methods can be roughly divided into three categories [27]: i) Session-based ii) Content-based and iii) Statistical approaches. The usage of well-known ports belongs to the first category, while the exhaustive packet payload analysis is a proponent of the second category. In this part, we focused on the third category, which exploits concepts of statistics, information theory as well as artificial intelligence, and usually it does not require any application-level protocol information.

[2]http://www.tcpdump.org/

[3]http://www.wireshark.org/

As concerns the inherent nature of statistical traffic classification approaches, they usually perform their tasks at two different levels:

- at a fine-grained level, to detect the particular *application protocol* that generated a certain flow [28];
- at a coarse-grained level, to identify a larger group of protocols (e.g., bulk transfer, mailing, web browsing, etc.), and not a specific protocol.

Whatever the considered granularity, all statistical classification algorithms usually consider transfer-based, time-based and protocol-based features of the packets to characterize a flow [29]. However, the description of flows in terms of numerical features may be carried out at different levels of abstraction [30]. As a matter of fact, there exist i) methods that consider packets as well as their inherent and simplest properties (e.g., size, inter-arrival time, the relative position in the flow, etc.); and ii) methods relying on aggregated statistical features of a flow or sub-flow (e.g., maximum, minimum, mean and standard deviation of the total volume in bytes, the overall duration, etc.).

Here we focused on the second approach, i.e., the one that regards aggregated statistical features of the flows (and their two main sub-flows). As concerns the granularity of the classification, we will address both a binary classification (distinguishing normal traffic from abnormal one) and a more fine-grained multiclassification (identifying normal traffic and different types of malicious flows).

3.2 State of The Art on Anomaly Detection based on Deep Learning

Deep Learning (DL) regards a particular branch of machine learning, encompassing techniques that allow the simulation of information processing typical of biological nervous systems [31]. A DL architecture is made of a set of related layers, wherein each layer obtains different inputs from another layer and reorganize the information in a hierarchical fashion, useful to perform feature learning and pattern classification.

DL algorithms are usually considered more suitable than other machine learning techniques in contexts featuring a high level of complexity (i.e., several attributes and a great number of data).

The training of a neural network has the two main phases:

- *the feed-forward phase*, wherein the activation of the nodes of the network is performed from the input layer, usually containing a number of nodes equal to the number of the considered features, to the output one, usually containing a number of nodes equal to the number of classes, in classification problems. Except for the nodes in the input layer, all subsequent nodes, in the intermediate layers, represent neurons which activate their output according to a proper and ad-hoc activation function (e.g., ReLu) [32].

- *the back-propagation phase*, which allows one to improve the overall network performance by assigning to the connection between the nodes proper and updated weights, as well as bias values if necessary, with the aim of increasing the overall performance of the whole neural network.

In [33, 34] authors surveyed anomaly detection mechanisms in IoT scenarios with artificial intelligence techniques.

Lopez-Martin et al. [35] proposed an unsupervised anomaly Network Intrusion Detection System (NIDS) for IoT environments, based on a Conditional Variational AutoEncoder (CVAE). Their method is unique due to its ability to carry out feature reconstruction, i.e., it can retrieve missing features from incomplete training datasets. The used dataset was a refined version of the NSL-KDD[4] one with 116 features and 5 possible labels. They proved experimentally that their work is less complex compared to other unsupervised NIDS, with better classification metrics (accuracy, precision, recall and F-measure) than well-known algorithms like random forest, linear SVM, multinomial logistic regression and multi-layer perception, both for binary and multiclassification problems.

Thing [36] analyzed wireless network threats and proposed an anomaly NIDS to detect and classify attacks in IEEE 802.11 networks, based on Stacked Auto-Encoder

[4]https://www.unb.ca/cic/datasets/nsl.html

(SAE), a neural network built by stacking multiple layers of sparse auto-encoders with both two and three hidden layers. The author experienced different activation functions for the hidden neurons. To test his strategy, he used the AWID_CLS_R dataset generated from a lab-emulated Small Office - Home Office (SOHO) infrastructure. He achieved an overall general accuracy of 98.6688% in a 4-class classification (legitimate traffic, flooding attacks, injection attacks and impersonation attacks), but with only a 2-layer neural network.

Diro and Chilamkurti [37] applied Fog Computing principles in IoT environments to detect intrusions. In particular, they equipped edge layer devices with intelligent detection capabilities to improve efficiency and reduce the data transported to the Cloud. The authors proposed a deep learning approach to detect known and unseen intrusion attacks, but without specifying clearly the number of used layers. The distributed parallel deep learning approach gets better results in accuracy than centralized deep learning NIDS and also than shallow machine learning algorithms, but the employed machine learning algorithms are not specified. Diro et al. used the aforementioned NSL-KDD dataset, with some modifications, thus considering 123 features. They performed 4-class detection and achieved 98.27% of overall accuracy, 96.5% detection rate, as well as 2.57% of false alarms rate using the deep learning model, while the shallow machine learning classifiers achieved an accuracy of 96.75%, 93.66% of detection rate and 4.97% of false alarms rate. They also noted an increase in the overall detection accuracy while increasing the number of fog nodes from around 96% to over 99%.

Moustafa et al. [38] proposed an Adaboost ensemble method for intrusion detection, based on decision tree, Naive Bayes and Artificial Neural Network, to mitigate particularly botnet attacks against DNS, HTTP and MQTT protocols utilized in IoT networks. A set of 36 features, some typical of a single protocol, are extracted from two datasets, namely UNSW-NB15 and NIMS. Then, a feature selection step is performed to extract the most important ones. This step enables the reduction of the computational cost of the overall system. The ensemble achieved an overall accuracy, for the binary classification, between 98.54% and 98.97% for UNSW-NB15 dataset and between 98.29% and 98.36% for the NIMS dataset, while the performance of the artificial neural network by itself, whose number of layers is not specified, does not pass 96.27%.

In the contribution in [39] the authors used an intelligent system to maximize the recognition rate of network attacks by embedding the temporal behavior of the attacks into a Tap Delay Neural Network (TDNN) structure, a particular type of recurrent neural network. It has only 2 layers and a Tap Line or Tap Delay Line (TDL), which consists of a group of taps that orders the temporal inputs. The system has been compared with SNORT and with the MIT DARPA Intrusion Detection Evaluation system over the old DARPA 1998 dataset with a claimed 100% recognition rate for both port scan and host sweep attacks. No clues are given about the considered features.

Vinayakumar et al. [40] proposed a highly scalable and hybrid Dense Neural Network framework called scale-hybrid-IDS-AlertNet (SHIA IDS), which can be used in real-time to effectively monitor network traffic and host-level events to proactively alert possible cyber-attacks. The authors considered both a multiclassification, trying to detect each different attack, and a binary classification by combining all attacks together and labeling them as “abnormal” traffic. They tested their model in different public datasets such as CICIDS-2017, NSL-KDD, KYOTO, WSN-DS, KDDCup-99, UNSW-NB15. They evaluated deep learning architectures of up to 5 hidden layers performing also experiments with a reduced set of features and some comparisons with traditional machine learning algorithms. They achieved a best overall accuracy over all classes, in the mutliclassification experiments, of 87.3% on the CICIDS-2017 dataset and of 93.57% on the UNSW-NB15 dataset.

Finally, the recent work in [41] considered two datasets, namely ISCX-IDS-2012 and CIC-IDS-2017, to evaluate the performance of a proposed Hybrid Neural Network method. Like the contribution in [40], they tested their model in two scenarios, namely multiclassification and binary classification, and they considered different types of features of a traffic flow (sequential, statistical and environmental). The achieved overall accuracy in the binary classification is 99.57% on the CIC-IDS-2017 dataset and 99.58% on the ISCX-IDS-2012 dataset, while the values are 99.35% and 99.61%, respectively, in the multiclassification scenario.

According to our paper [42] in this section first the proposed feature model and then the used deep learning neural network is described.

3.3 Feature Extraction Model

Similarly to the anomaly detection on traditional Internet traffic, also in case of IoT traffic we consider bi-directional network flows composed of a series of ordered packets, exchanged between two terminal points. The set of features considers all the aspects that characterize a flow as well as its two main sub-flows, one for each direction of the communication (forward, Fwd, or backward, Bwd). As a consequence a flow F_i, identified through the 5-tuple described in Sec. 3.2, can be considered as a series of features (f_j, $j = 1...n$) mirroring an instance as follows:

$$F_i = \{f_1, f_2, f_3, ...f_n\} \;\; i = 1...M \tag{3.1}$$

where n is the number of features per flow, and M the number of considered flows.

Specifically, we extracted from the considered traffic flows a set of 70 statistical features, as summarized in Table 3.2: these features have been already successfully adopted within the traffic classification schemes discussed in [30, 43]. Table 3.2 lists, for each feature, a brief description, as well as the unit of measurement (UoM) of the feature itself (where μs and B stand for micro-seconds and bytes, respectively). The features encompass mainly size- and time-related characteristics, as well as their maximum, minimum, average and standard deviation values, but they also comprise information about the flags, the number of packets with certain characteristics or further sub-flows inside the two main sub-flows.

In order to make a flow a full instance, we had to label each flow according to the type of traffic, adding the so called *class* as the $(n+1)^{th}$ feature. In this paper we considered both a binary classification, where the flows were label only as BENIGN or ATTACK, and a multinomial classification, where we considered the particular type of attack.

Based on the the previous surveys we mentioned in the Sec. 3.2 and state of the art IoT datasets the most popular and famous attacks on IoT systems are in particular the following attacks:

- DOS: a Denial of Service attack, performed also in a distributed fashion.
- MIRAI: an attack launched by a Mirai bot.
- MITM: a Man-in-the-Middle attack.
- SCANNING: comprising both OS scanning and service scanning attacks.

3.4 Deep Learning Model

The considered deep learning architecture was inspired by the work in [44], its main components are presented in Fig. 3.1, and they are described in the following:

- one *Input layer*: the entry point of the network, encompassing a number of nodes equal to the number of considered features (70 as described in Sec. 3.3);
- a *Batch Normalization layer*: useful to improve the training of the neural network, since it increases the speed of training and permits one to adopt higher learning rates as well as to saturate possible non-linearities. This usually results into a higher accuracy on both validation and test, thanks to a stable gradient propagation within the network itself [45].
- a variable number of *Hidden layers*: they are made of artificial perceptrons and output a weighted sum of their inputs, passed through a *ReLu* activation function. We made experiments with a different number of hidden layers to evaluate the differences in the overall performance.
- a *Dropout layer*: we considered this layer tightly coupled with the aforementioned one and immediately following it. As a matter of fact, in the above mentioned experiments, we replicated different times the pair hidden layer-dropout layer. The dropout layer helps to prevent over-fitting by means of a regularization technique that turns off randomly several neurons in a layer according to a probability p drawn from a Bernoulli distribution. We considered the same

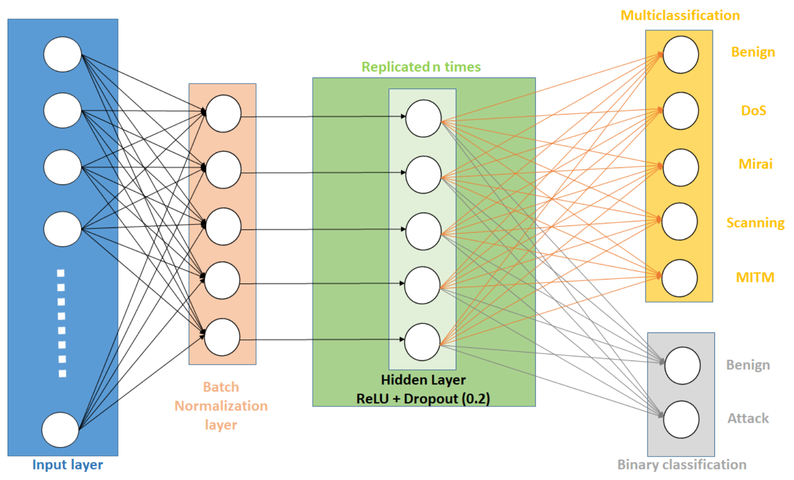

Figure 3.1: The used neural network model for both the binary (gray) and multi- (orange) classification.

probability for each node of the coupled hidden layer and we set its value to 0.2.

- An *Output layer*: this layer produces the final classification outcome and is composed of a number of nodes equal to the number of classes. In Fig. 3.1, we show 2 different output layers, since they refer to the binary classification and multi-classification problem, respectively. However, when we ran our experiments we considered for each problem only one of the two depicted output layers at a time, which are a dense layer using a $softmax$ as activation function.

3.5 Experimentation Description

In this section, we present the application of the deep neural network architecture, described in Sec. 3.3 to a large IoT dataset. We constructed it by merging together traffic traces coming from different sources.
First of all, we describe the four datasets we used as well as the statistical characteris-

tics of the whole merged large dataset, secondly, we detail the parameters we used as well as the considered metrics, and finally, we report the evaluation results together with a brief discussion.

3.5.1 Dataset Scenario

The literature review, reported in Sec. 3.2, shows that the studies about IoT attacks and traffic classification mainly employ *ad hoc* built datasets to evaluate particular and specific attacks or traffic flows. Some of the limits of these datasets are that they are i) small, so not apt for being used with deep learning techniques, ii) with a few attacks or with difficulties in separate clearly attacks from benign traffic, iii) not directly comparable and scarcely usable to evaluate and compare different approaches, and iv) usually coming from the same network scenario, where flows exhibit the same easy to learn pattern across the considered features.

Stemming from these considerations, we decided to build a large and integrated dataset by merging together four different IoT datasets, with different dimensions and different types of attacks and of traffic, in order to validate the proposed deep neural network model. The integration procedure is composed of the following steps: i) datasets selection, ii) datasets transformation; iii) datasets cleaning, and iv) datasets merging.

The selection of the datasets regarded finding out in the Web useful and recent datasets, built in the last few years and containing a sufficient number of instances regarding both benign and malicious IoT traffic.

The transformation step concerned the creation of proper CSV files, with more features than the 70 ones described in Sec. 3.3, from raw .pcap files, as well as the consistent labeling of the flow instances by exploiting the information each single dataset was endowed with. These tasks have been performed by using the CICFLOWMETER tool[5] [46], which is a Java network traffic flow generator allowing for flexibility in terms of choosing the features to compute and a control of the duration of the flow timeout.

[5]https://github.com/ahlashkari/CICFlowMeter

The cleaning step regarded the removal of instances containing inconsistent values such as NaN, Infinity, and the like, and it has been performed using a proper Python script. This step involved also the reduction of the initial available features, in order to remove non-computable features or constant features throughout all the instances. This step led to have the surviving 70 features described in Sec. 3.3.

Finally, the merging step, carried out through a Python script as well, provided the integration of the considered datasets into a unique large dataset, whose statistics, together with the ones of the 4 datasets it is composed of, are summarized in Table 3.1.

Dataset D1[6] [47] has been released on September 2019 and built considering two typical smart home devices, i.e., SKT NUGU (NU 100) and EZVIZ Wi-Fi Camera (C2C Mini O Plus 1080P), as well as some laptops and some smartphones, connected to the same wireless network. All attacks except Mirai Botnet category contain packets captured while simulating attacks using tools such as Nmap. In the case of Mirai Botnet attack, the packets were generated on a laptop and then manipulated to make it appear as if they originated from the IoT device. For dataset D2 [48] [49] [50] [51] [52] [53], created by designing a realistic network environment in the Cyber Range Lab of UNSW of Canberra, we considered only 5% of the entire dataset and only scanning attacks in order to make the whole merged dataset unbalanced and test our DL architecture in these conditions. For dataset D3 [54], whose data were collected for IEEE TMC 2018 [55], we only considered benign traffic, in order to increase the number of instances of this type of traffic up to the order of magnitude of scanning attacks and, at the same time, to vary the network scenario from which traffic traces are captured. Indeed, these traces contain traffic from a great variety of IoT devices, such as Amazon Echo, Netatmo Welcome, TP-Link Day Night Cloud camera, Samsung SmartCam, etc.

Finally, dataset D4 [56] contains mainly IoT malware traffic of type “Mirai” captured at the Stratosphere IPS laboratory at the Czech Technical University in 2018 and 2019, and we exploited it partly to achieve an order of magnitude of Mirai instances, in the whole dataset, equal to the one of benign traffic and scanning attacks.

[6] https://ieee-dataport.org/open-access/iot-network-intrusion-dataset

Table 3.1: Statistics of the considered datasets.

Dataset ID	No. instances	Benign	DoS	MITM	Mirai	Scanning
D1	26246	496	3273	378	18623	3476
D2	794767	-	-	-	-	794767
D3	437322	437322	-	-	-	-
D4	546720	-	-	-	546720	-
Whole	1805053	437817	3273	378	565343	798243

The integrated whole dataset comprises a total of 1805053 flow instances, 437817 of normal traffic and 1367236 of abnormal traffic (attacks). The figures of the four considered attacks (DoS, Mirai, MITM and Scanning) are detailed in Table 3.1. As one can see, the instances for DoS and MITM are quite small compared to the other types of traffic, and this was done purposely to test the robustness of the proposed DL model against unbalanced data, as it will be described in Sec. 3.6.

3.5.2 Evaluation Settings

The assessment has been performed on the aforementioned integrated dataset in order to identify i) benign and malicious traffic, and ii) normal traffic as well as 4 different attacks. The evaluation is performed using both the deep learning architecture and by using some traditional machine learning algorithm, namely Hoeffding Tree (HT), useful for the classification of streaming data from IoT devices, and Naive Bayes (NB), with the aim to demonstrate that, for the considered IoT attack scenario, a machine learning approach is overcome by deep neural networks with several hidden layers. The considered deep neural network architecture has been implemented using the Python programming language, in particular Tensorflow 2.1.0[7], an open source software library for high-performance numerical computation, and Keras 2.3.1[8], a Python-based high-level neural networks API, cable to run on top of TensorFlow and to simplify the creation of an artificial neural network. The deep neural network model, described in Sec. 3.4, was trained by using categorical cross-entropy [57] as

[7]https://www.tensorlow.org/

[8]https://keras.io/

Table 3.2: The list of the 70 considered features used to characterize a flow.

Feature	Description	UoM
Flow duration	Duration of the flow in microseconds	μ s
Total Fwd Packet	Total packets in the forward direction	pck
Total Bwd packets	Total packets in the backward direction	pck
Total Length of Fwd Packet	Total size of packets in forward direction	B
Total Length of Bwd Packet	Total size of packets in backward direction	B
Fwd Packet Length Min	Minimum size of packets in forward direction	B
Fwd Packet Length Max	Maximum size of packets in forward direction	B
Fwd Packet Length Mean	Mean size of packets in forward direction	B
Fwd Packet Length Std	Standard deviation size of packets in forward direction	B
Bwd Packet Length Min	Minimum size of packets in backward direction	B
Bwd Packet Length Max	Maximum size of packets in backward direction	B
Bwd Packet Length Mean	Mean size of packets in backward direction	B
Bwd Packet Length Std	Standard deviation size of packets in backward direction	B
Flow Byte Rate	Number of flow bytes per second	B/s
Flow Packets Rate	Number of flow packets per second	pck/s
Flow IAT Mean	Mean time between two packets sent in the flow	μ s
Flow IAT Std	Standard deviation time between two packets sent in the flow	μ s
Flow IAT Max	Maximum time between two packets sent in the flow	μ s
Flow IAT Min	Minimum time between two packets sent in the flow	μ s
Fwd IAT Min	Minimum time between two packets sent in the forward direction	μ s
Fwd IAT Max	Maximum time between two packets sent in the forward direction	μ s
Fwd IAT Mean	Mean time between two packets sent in the forward direction	μ s
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction	μ s
Fwd IAT Total	Sum of all IATs in the forward direction	μ s
Bwd IAT Min	Minimum time between two packets sent in the backward direction	μ s
Bwd IAT Max	Maximum time between two packets sent in the backward direction	μ s
Bwd IAT Mean	Mean time between two packets sent in the backward direction	μ s
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction	μ s
Bwd IAT Total	Sum of all IATs in the backward direction	μ s
Fwd PSH flag	Number of times the PSH flag was set in packets traveling in the forward direction (0 for UDP)	-
Bwd PSH Flag	Number of times the PSH flag was set in packets traveling in the backward direction (0 for UDP)	-
Fwd URG Flag	Number of times the URG flag was set in packets traveling in the forward direction (0 for UDP)	-
Bwd URG Flag	Number of times the URG flag was set in packets traveling in the backward direction (0 for UDP)	-
Fwd Header Length	Total bytes used for headers in the forward direction	B
Bwd Header Length	Total bytes used for headers in the backward direction	B
Fwd Packet Rate	Number of forward packets per second	pck/s
Bwd Packets Rate	Number of backward packets per second	pck/s
Min Packet Length	Minimum length of a packet in the whole flow	B
Max Packet Length	Maximum length of a packet in the whole flow	B
Packet Length Mean	Mean length of a packet in the whole flow	B
Packet Length Std	Standard deviation length of a packet in the whole flow	B
Packet Length Variance	Variance length of a packet in the whole flow	B
FIN Flag Count	Number of packets with FIN	pck
SYN Flag Count	Number of packets with SYN	pck
RST Flag Count	Number of packets with RST	pck
PSH Flag Count	Number of packets with PUSH	pck
ACK Flag Count	Number of packets with ACK	pck
URG Flag Count	Number of packets with URG	pck
CWR Flag Count	Number of packets with CWE	pck
ECE Flag Count	Number of packets with ECE	pck
Down/Up Ratio	Download and upload ratio	-
Average Packet Size	Average size of packets in the whole flow	B
Avg Fwd Segment Size	Average size of a segment observed in the forward direction	B
Avg Bwd Segment Size	Average size of a segment observed in the backward direction	B
Subflow Fwd Packets	Average number of packets in a sub flow in the forward direction	pck
Subflow Fwd Bytes	Average number of bytes in a sub flow in the forward direction	B
Subflow Bwd Packets	Average number of packets in a sub flow in the backward direction	pck
Subflow Bwd Bytes	Average number of bytes in a sub flow in the backward direction	B
Init Win bytes forward	Total number of bytes sent in initial window in the forward direction	B
Init Win bytes backward	Total number of bytes sent in initial window in the backward direction	B
Act data pkt forward	Count of packets with at least 1 byte of TCP data payload in the forward direction	pck
min seg size forward	Minimum segment size observed in the forward direction	B
Active Min	Minimum time a flow was active before becoming idle	μ s
Active Mean	Mean time a flow was active before becoming idle	μ s
Active Max	Maximum time a flow was active before becoming idle	μ s
Active Std	Standard deviation time a flow was active before becoming idle	μ s
Idle Min	Minimum time a flow was idle before becoming active	μ s
Idle Mean	Mean time a flow was idle before becoming active	μ s
Idle Max	Maximum time a flow was idle before becoming active	μ s
Idle Std	Standard deviation time a flow was idle before becoming active	μ s

a loss function, and stochastic gradient descent (SGD), with learning rate equal to 0.1, momentum equal to 0.09, decay of $1e^{-6}$ for optimizing the loss function. Moreover, SGD has been integrated with Nesterov accelerated gradient (NAG) correction to avoid excessive changes in the parameter space [58]. The metrics that we used to evaluate the classification results have been the following: Precision, Recall, Accuracy and F-measure. As a matter of fact, the considered datasets is somehow unbalanced with respect to certain attack classes as it has been described in Subsec. 3.5.1, so only the overall accuracy would have not been a significant parameter. Precision has been evaluated as the proportion of samples that truly belong to a given attack (or normal flow) among all those which were assigned to it. It is computed as the ratio of the number of relevant detected samples (true positive) to the sum of irrelevant detected samples (false positives) and relevant detected samples (true positives):

$$Precision = \frac{true\,positives}{true\,positives + false\,positives}. \tag{3.2}$$

On the other hand, the recall has been evaluated as the proportion of samples assigned to a given attack (or normal flow), among all the samples that truly belong to the attack (or normal traffic) itself. It is computed as the ratio of the number of relevant detected samples (true positive) to the total number of relevant samples (the sum of true positives and false negatives):

$$Recall = \frac{true\,positives}{true\,positives + false\,negatives}. \tag{3.3}$$

The F-measure, or F-score, is the weighted harmonic mean of precision and recall, and is computed according to the following formula:

$$F - score = 2\frac{PR}{P+R}, \tag{3.4}$$

where P and R are precision and recall, respectively. While precision and recall can be computed both per class and on average, the accuracy is an overall metric and has been computed as the ratio of the sum of true positives and true negatives to the total number of samples:

$$Accuracy = \frac{tp+tn}{tp+fn+tn+fp}, \tag{3.5}$$

where tp means true positives, tn means true negatives, fn means false negatives, and fp means false positives.

The experiments have been run on an Intel Core i7 7^{th} gen machine, equipped with 1 GPU and 16GB of RAM.

3.6 Results

In this section we present some of the obtained results, as regards both the training phase of the considered Deep learning (DL) model as well as the test phase and the comparison with traditional machine learning algorithms. All the results have been obtained considering a 5-fold cross validation process.

Deep learning networks are usually trained for several epochs, a hyper-parameter defining the number of times the learning algorithm will present the entire training dataset to the network under training. To make the network reaching its best performance during the test phase, it is important to set the number of epochs to a value corresponding to the point when the network accuracy vs the numbers of epochs is not increasing anymore. The loss function is usually inversely proportional to the accuracy and when the accuracy increases, the network is learning and the loss trend is decreasing. In a symmetric way to what said about the accuracy, the network does not learn anymore when the loss function reaches a constant trend.

Figures 3.2 and 3.3 show the trends of accuracy and loss versus an increasing number of epochs for the considered DL model with a hidden number of layers equal to 7 (DL7), the configuration reaching the best performance in terms of accuracy. As one can see, after about 50 epochs both the accuracy and the loss trends stabilize. Similarly, Figures 3.4 , 3.5 presents how accuracy and loss vary during the epochs when considering the multinomial classification and the training of the proposed DL model with 6 hidden layers (DL6), the configuration achieving the top accuracy in this type of classification. As one can see, the trend is less smooth than in the case of the binary classification, and this is possibly due to the major difficulty in separating more classes, but the number of epochs when the trend stabilizes is more or less the same.

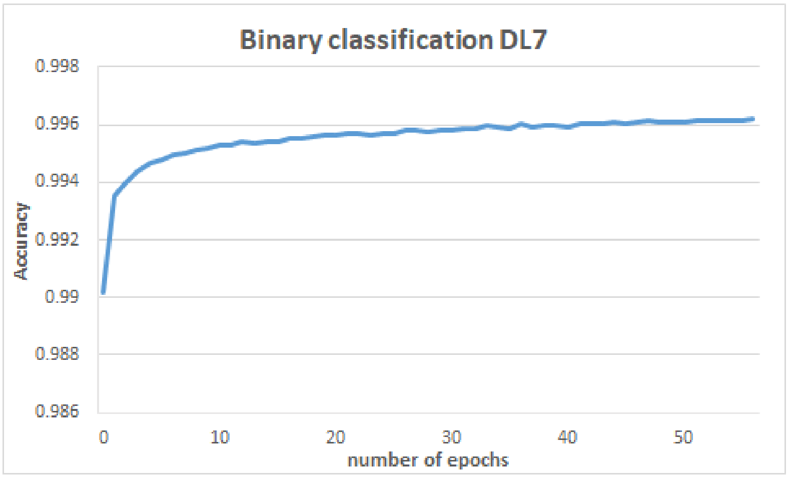

Figure 3.2: Accuracy trends for binary classification with 7-layer DL.

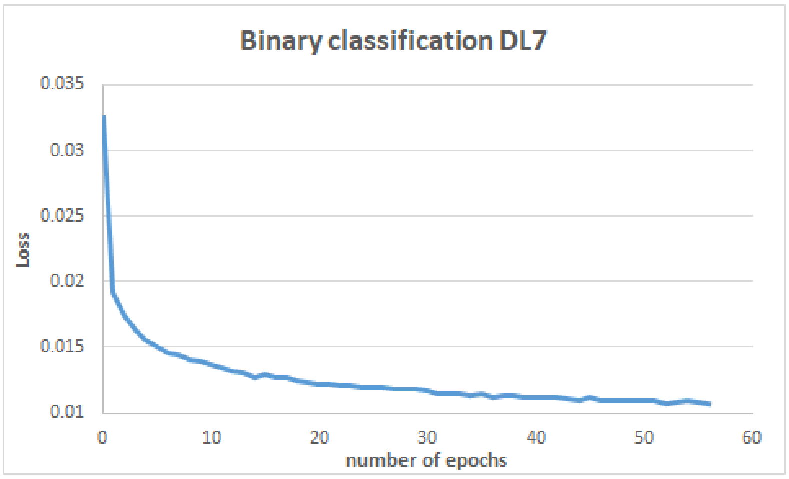

Figure 3.3: Loss trends for binary classification with 7-layer DL.

As concerns the test phase, we report in Table 3.3 the values of overall accuracy, precision, recall and F-measure for both some traditional machine learning techniques and the proposed Deep leaning model. As stated in Sec. 3.5.2, we considered

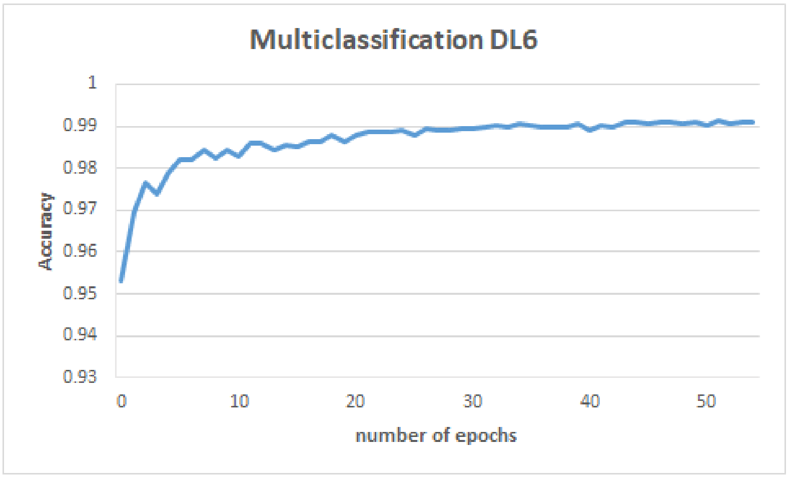

Figure 3.4: Accuracy and loss trends for multiclassification with 6-layer DL.

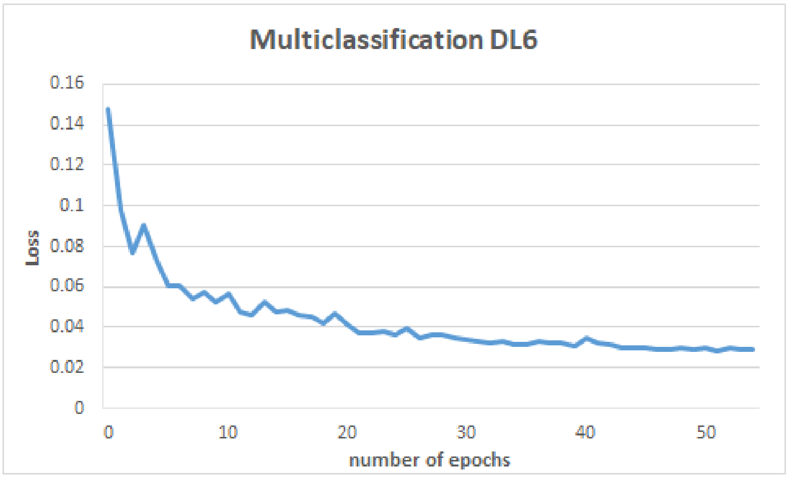

Figure 3.5: Loss trends for multiclassification with 6-layer DL.

as machine learning techniques HT and NB and their Weka[9] implementation, while for the proposed DL model we report the results for different hidden layers, from 4 to 7. As one can see, the best accuracy is reached with a DL architecture made of

[9]https://www.cs.waikato.ac.nz/ml/weka/

Table 3.3: Classification results on the test set.

	Binary classification				**Multiclassification**			
Alg.	**Acc.**	**P**	**R**	**F**	**Acc.**	**P**	**R**	**F**
HT	0.9930	0.993	0.993	0.993	0.9736	0.974	0.974	0.974
NB	0.8723	0.877	0.872	0.859	0.6702	0.879	0.670	0.748
DL4	0.9968	0.9935	0.9935	0.935	0.9942	0.9875	0.9854	0.9864
DL5	0.9969	0.9936	0.9936	0.9936	0.9943	0.9884	0.9864	0.9874
DL6	0.9970	0.9938	0.9938	0.9938	**0.9973**	0.9886	0.9867	0.9877
DL7	**0.9975**	0.9937	0.9937	0.9937	0.9937	0.9874	0.9854	0.9864

7 layers for the binary classification, while it is achieved by a DL architecture made of 6 layers for the multiclassification. In general, the DL approach outperforms the two considered traditional machine learning techniques and can reach the best results compared to the related works surveyed in Sec. 3.2. As concerns the introduced unbalance in the dataset, we can see that for the binary classification the effect is negligible, given that the averaged precision, recall and F-measure are always greater than 0.99, while it becomes somewhat relevant in the multiclassification, even if the considered metrics are always better than those of traditional ML techniques.

3.7 Threats to Validity

As concerns the construct validity threats, some inaccuracies and omissions can be due to the reliability of the captured traffic traces and of the tools used to extract or compute the features. In order to limit this threat, we have considered four different datasets, from four different network scenarios, as well as a very renown and used tool like CICFLOWMETER. Moreover, regarding the internal validity, if the adopted datasets are not correctly labeled or are obtained with a non-rigorous process, we could have classification errors. This risk is strongly mitigated because the used datasets are well documented and referenced in papers already published in reputable venues. Finally, threats to external validity may involve the generalization of the discussed findings. We have evaluated our approach on a great number of flows from four existing datasets having different sizes and characteristics. However, in the future, it is possible to integrate more datasets with many more IoT flows.

Chapter 4

Application Scenario

4.1 POSITIVE Project, Scope and Technologies

The POSITIVE project has been funded under the 2018 call POR FESR Emilia-Romagna (Regional Operational Program - European Regional Development Fund), Strategic industrial research projects aimed at the priority areas of the Intelligent Specialization Strategy. The total amount of the project is 1.1 million Euros, of which 70 percent covered by the regional contribution.

POSITIVE aims to make agronomic interest indices available on a regional scale from the satellite images of the Copernicus project and to prepare an IT infrastructure that makes precision irrigation and fertigation usable throughout the entire region.

The activity plan was divided into 5 phases, not to be understood as strictly sequential but functional, with evident levels of interdependence. As regards precision agriculture, the project not only deals with the specificity of the individual phases, but also connects with the seasonality of the irrigated / fertigation campaigns. Phase 1 designs and implements the POSITIVE Sat Service. To this end, it defines and implements the procedures for downloading data from the Copernicus hub, the processing necessary to obtain the indices and agronomic parameters that can be derived from satellite data, the organization of the archive and the data access service by users via the Web API. Phase 2 concerns the execution of experimental measurements on the ground for the validation (ground truthing) of both the indices produced by the processing of satellite images and the data collected by ground sensors (proximity or in vivo) in the companies served by the participating companies. These operations will be repeated in both the useful crop campaigns of the project. Phase 3 intervenes on the IRRINET irrigation service, improving it at two levels: in the forecast model of water needs, with the introduction of temporally updated data derived from satellite and, where available, from ground sensors; in operations and functionality, with the adaptation of interfaces to users and to data acquisition services / systems. Phase 4 is divided into two sub-phases. The first deals, in agreement with the project companies and the partners associated with the project, of the design and implementation of Scalable Operational Protocols (POS). In other words, components and interfaces will be developed to manage the flow of information from satellite or ground data to irrigation advice services and from these to precision and variable rate irrigation systems. The realized architecture will be available in open format, documenting the data generated by the components and their interfaces. The second includes the implementation of the POSITIVE Server information system, to coordinate and organize the data flow between the aforementioned services and systems, and its integration into the information system oriented to the SAMS business management referred to in phase 5. Phase 5 provides for the validation of the Scalable Operating Protocols by means of a demonstrator information system consisting of the integration of the POSITIVE Server with the information system called Smart Agronomic Management System (SAMS) prepared by CRPA Lab. The demonstration of the operation of the irrigation management system of precision and variable rate is carried out by applying it to the management of the irrigation campaign in pilot farms equipped with precision irrigation equipment, on different types of crops.

POSITIVE Objectives:

- set up a stable service, with coverage on a regional scale and managed by institutional actors, which makes updated maps of the most significant agronomic indices at 10 x 10 m resolution available on a regular basis and interfaces it to the IRRINET system;
- develop components and interfaces to manage the flow of information from

the production of index maps (including those generated by proximity or in vivo sensors) to irrigation advice systems and from these to precision irrigation systems;

- implement a demonstration system for the management of precision irrigation with a high degree of automation that relies on these services and on protocols shared with companies producing irrigation machines;

- improve the FERTIRRINET functionality of the IRRINET service in the quality of irrigation and fertigation advice thanks to the systematic integration of satellite data and, where available, of data from ground sensors;

- producing variable rate irrigation plans for crops and plots that can benefit from them. The standards and protocols developed in the POSITIVE project will be available and documented to be freely usable by all companies interested in the topic of precision fertigation.

Expected Results:

Service for the supply of high resolution satellite maps for the entire RER territory. A service, called Positive Sat Service, will be created for accessing Sentinel2 satellite data. These data will consist of derived maps of biophysical parameters relating to the main crop classes of the region, georeferenced and updated every 5 days. The maps, with a resolution of 10 or 20 m, can be used, alongside other data, to feed the forecast models of irrigation and fertigation needs, from the scale of the district to that of the plot and its internal parts. Users (farms, service providers, reclamation consortia, producer associations) will be able to access the data in a flexible way (eg for delimited areas) and will have a high quality database which is also useful for statistical and forecast studies. Service of supply of irrigation and fertigation "recipes" with high spatial resolution and variable rate. At the end of the project, the irrigation and fertigation recipe today provided by the IRRINET / FERTIRRINET service of the CER will be improved in terms of accuracy and reliability thanks to the integration of satellite information on crops and, where present, from the company sensor networks on the ground. The high spatial resolution of satellite or ground data will make it possible to generate different recipes within the plot. The service thus enhanced, here called IRRINET +, will continue to be provided by the ERC as an institutional service, thus bringing the potential and resources for precision and variable rate irrigation to full capacity for the entire region. Protocols for interfacing public and private service servers and precision irrigation machines. The adhesion to the project of important manufacturers of different irrigation machines and equipment will allow to jointly develop and document the standards and interfaces to enable the interconnection of a significant repertoire of such machines to public and private irrigation council services. Some machines enabled by the project will be operational at different farms in the region on diversified crops. A geographic information system (POSITIVE-SAMS) will document the validity and effectiveness of the protocols developed by displaying the data collected over time (eg sensory measurements, irrigation, yields). The data will be available at different levels of detail, distinguishing between private data or data pertaining to the individual company and aggregate data of interest on a larger scale (eg. Protocols for interfacing field sensors and public and private service servers. By virtue of the interfacing protocols developed in the project, the data provided by field sensors, interconnected in the network, will be more easily acquired by expert decision support systems and irrigation or fertigation advice, helping to validate the indications. With the contribution of companies participating in the project, having significant experience in the installation and configuration of such sensors, configurable interfaces will be defined capable of encompassing a wide range of measurement situations on a proximal and in vivo scale, also enabling the interconnection of emerging types of sensors according to the Internet of Things paradigm. Demonstrator of scalable operational protocols for precision irrigation. POSITIVE Server integrated with POSITIVE-SAMS will constitute a demonstrator of the project, i.e. a distributed IT infrastructure that communicates with the data producers (satellite or land), with the IRRINET + service, with users and irrigation equipment capable of receiving commands and returning data. The demonstrator will be based on data models in standard formats, accessible via documented APIs, and will include software components to periodically query information providers, collect and process information, send data, commands and recipes to registered users or keep them available for a supply on request. The system will demonstrate the functionality of the operational protocols and the interoperability of services.

Within the project, the aspects regarding data capturing, by means of proper on field sensors (e.g. soil temperature and humidity sensors), and transmission have been faced.

As LPWAN technology the LoRaWAN protocol and the corresponding network architecture have been selected.

The carried out activities are hereafter described, focusing on: (i) LoRaWAN sensor devices, (ii) LoRaWAN gateways and (iii) LoRaWAN backend platforms.

4.2 Practical Experimentation

This part of the research mainly focused on the evaluation of LoRaWAN technologies and corresponding experimentation for the transmission of sensor data in precision agriculture scenarios. We started by deeply analyzing the state of the art of LoRa and LoRaWAN standard, versions 1.0.2 and 1.1, facing supported features, procedures and communication mechanisms between the different LoRaWAN components, that are the end nodes, gateways, join-server, network-server, and application-server, possibly involved in both join procedure and uplink and downlink data transmissions.

4.2.1 LoRaWAN Devices

We used two categories of LoRaWAN devices (nodes) in our Experimentation.

- Off-the-shelf sensor devices: Temperature-humidity sensors (e.g. LHT65, LSE01 devices)
- Ad-hoc devices
  - STM32 WL55JC (Included lora module)
  - STM32 L072Z (Included lora module)
  - STM32 L476RG (Without lora module)

For LoRaWAN device prototyping there are some development boards like Reyax which includes MCU, antenna and lora module. Lora-module is communicating with MCU via SPI bus. STM32 causes to increase lora-module performance. It can be found in many commercial products. By studying papers and real world scenarios important parameters in prototyping lora node can be:

- Power consumption
- Easy programming (UPLOADING CODE TO THE MCU)
- Memory (Ram) depends on the codes we need to upload in the device
- MCU speed
- Price

Note: Each Lora device can be easily play the Lora-gateway role by adding an embedded Linux board like Raspberry Pi.

Lora Node Prototyping and Experimentation:

We studied all possible hardware components necessary to implement our experimentation in real world experimentation and state of the art Lora Node Prototyping scenarios (more than 20 prototyping scenarios) then by comparing them and the features of each hardware component we gathered necessary list of material (LOM) to deploy our prototyping base on a less power consumption, less CPU and memory usage. Main hardware components to form a LoRaWAN node are:

- Board/MCU
- Lora module
- Battery
- Possibility of connecting other interfaces (e.g. RS232-485-SDI12) to the LoRaWAN node

Boards and their features: Most common proper boards for LoRaWAN device prototyping are such as Arduino Atmega base, stm32 arm based, Esp32 ,Pycom board) and comparison done according to our studied the best MCU was STM32 because of:

- Includes all components (Board/MCU, Lora module, Antenna) to gather Low power consumption
- High performance due to the MCU speed
- Easy programming (method of uploading code to the MCU)
- Memory size
- Economic aspect

In order to quantifying the link performance two main parameters are considered:

(i) Link budget (is the sum of all of the gains and losses from the transmitter, through the medium, to the receiver in a telecommunication system)

(ii) Receiver sensitivity (is the lowest power level at which receiver can receive or demodulate the signal).[1]

Lora modules and features:

Common lora module are such as SM32WLX, MICROCHIP, Adafruite, HopeRF, Rylr890, Dragino. Parameters can be considered are such as:

- Simple connectivity to the board
- Exist practical method to use it in real world
- Radio power consumption in standby/Active mode depending on Duty-cycle [2]
- Sensitivity
- Link budget

[1] https://lora.readthedocs.io/en/latest/

[2] https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/

- High speed (Lora-module is communicating with MCU via SPI bus)

LoRa transceiver modules:

(i)HopeRF RFM96 LoRa transceiver module: which uses RF95W part number. On its chip is marked RF96 which means the chip is using the SX1276 chip.

(ii)LoRa-E5 LoRa module: It has a lot of capabilities e.g. Ultra-low Power Consumption, Extremely Compacted Size. High Performance and long Distance Use.

Conclusion: HopeRF with sensitivity down to -148 dBm, maximum link budget 168 dB and sleep current 0.2uA is number one between all others. Battery and it's features for LoRaWAN device prototyping:

power supply and battery plays an integral rule on life time of the node as node power consumption should be as less as possible. Most common battery types Li-Soci2, Li-ion, Li-po. Most important features to select battery are:

- Battery life
- Temperature conditions the battery can tolerate (-30 to 70)
- Economic aspects

Important notes:

- With less power the MCU has less performance e.g. in ATmega MCU speed decreases from 16mhz to 8mhz e.g transmit each 20 minutes. So in order to have as less as possible power consumption, Arduino nano (Arduino uno) can power on from 1.8 to 5.5 volt
- Using a capacitor with the battery could increase battery life many years more.
- Rechargeable (Recharging battery using SOLAR PANEL, not useful for LoRaWAN projects as it decrease the life time of battery until 2 years) moreover the accessibility of node could be hard to change the battery.
- Thus Li-Soci2 type is proper as it's life duration is around 10 years minimum in case device sends packet every 20 minutes.

We did also reverse engineering in LSE01 sensor hardware components as it was a suitable LoRaWAN node to start.

Possibility of connection interfaces (e.g. RS232-485-SDI12) within the board (Stm32 or any type of board):

- RS-232 link: It cannot be directly wired to an Arduino serial link pins (voltage levels are not compatible). It needs a voltage/logic inversion converter such as a MAX232 chip.
- RS485 link: there is a RS485-module which add to board to be able
- SDI12 link: easily can connect to Arduino and no need any converter
- Use serial to LoRaWan bridge (converters) which directly convert sensor data from rs232 or rs485 to gateway also it is possible to use AT Commands to change parameters

It is interesting we found out that LSE01 sensor consists of a LoRaWAN RS485 converter and a soil moisture sensor, so it is possible also to use it as a RS485 converter.

SDI-12 protocol (Serial Digital Interface) Soil moisture sensors (e.g. Sentek prob): The SDI-12 specification is maintained by the SDI-12 Support group. SDI-12 is an asynchronous serial communications protocol often used for environmental monitoring due to it's low power operation. It is intended for use in systems that are battery powered, low cost, and require a multiple sensors attached to one cable. SDI-12 is a standards-based protocol for interfacing sensors to data loggers and data acquisition equipment. Multiple sensors with unique addresses can share a common 3-wire bus (power, ground, and data). Two-way communication between the sensor and logger are possible by sharing the data line for transmit and receive as defined by the standard. Sensor measurements are triggered by protocol command. An intelligent sensor typically takes a measurement, makes computations based on the raw sensor reading, and outputs the measured data in engineering units. For example, an SDI-12 pressure sensor may take a series of pressure measurements, average them, and then output pressure in psi, inches of mercury, bars, millibars, or torrs. The sensor's micro-processor makes the computations, converts sensor readings into the appropriate units, and uses the SDI-12 protocol to transfer data to the recorder.SDI-12 has several advantages in application projects:

- battery powered operation with minimal current drain
- use of one data recorder with multiple sensors on a single cable
- use with microprocessor-based sensors that perform complex calibration algorithms or make internal computations

Using micro-processor based sensors: A micro-processor in the sensor may calibrate the sensor, control sensor measurements, and convert raw sensor readings into engineering units. The micro-processor also controls the SDI-12 interface. It accepts and decodes instructions received from the data recorder, starts the measurements, controls all timing, and uses the SDI-12 protocol to communicate with the data recorder.

Connect more than one sensor to a single data recorder: SDI-12 is a multi-drop interface that can communicate with multi-parameter sensors. Multi-drop means that more than one SDI-12 sensor can be connected to a data recorder. The SDI-12 bus is capable of having ten sensors connected to it. Having more than ten sensors, however, is possible. Some SDI-12 users connect more than ten sensors to a single data recorder.

Take more than one measurement from a sensor: Multi-parameter means that a single sensor may return more than one measurement. For example, some water quality sensors return temperature, conductivity, dissolved oxygen, pH, turbidity, and depth.

Advantages of SDI-12 possess for environmental data collection:

- A serial-digital interface is a logical choice for interfacing microprocessor-based sensors with a data recorder. This has advantages for sensors and data recorders.

- Unique and complex self calibration algorithms can be done in microprocessor-based sensors.
- Sensors can be interchanged without reprogramming the data recorder with calibration or other information.
- Power is supplied to sensors through the interface.
- Hybrid circuit and surface mount technologies make it practical to include the power supply regulator, a microprocessor, and other needed circuitry in small sensor packages.
- Sensors can use low cost EEPROMs (electrically erasable programmable read only memory) for calibration coefficients and other information instead of internal trimming operations.
- The use of a standard serial interface eliminates significant complexity in the design of data recorders.
- Data recorders can be designed and produced independently of future sensor development.
- SDI-12 data recorders interface with a variety of sensors.
- SDI-12 sensors interface with a variety of data recorders.
- Personnel trained in SDI-12 will have skills to work with a variety of SDI-12 data recorders and SDI-12 sensors.

Conclusion: According above studied the best micro-controller (MCU) could be STM32 because of low power consumption and high performance so SM32WLX board types together with their including lora module was the best choice due to the aforementioned essential factors.

Although lora device prototyping is the first step in lora deployment it seems researchers have not been prepared a scientific sufficient paper regarding best hardware for prototyping yet perhaps because this section is an interdepartmental study field between electronic and computer area.

4.2.2 LoRaWAN Gateways

We used two types of gateways in our experimentation:

1. LoRa Gateway (Dragino LG02)

   LG02 is an open source dual channels LoRa Gateway. It lets to bridge LoRa wireless network to an IP network via WiFi, Ethernet, 3G or 4G cellular. The LoRa wireless allows users to send data and reach extremely long ranges at low data-rates. It provides ultra-long range spread spectrum communication and high interference immunity. LG02 have rich internet connection methods such as WiFi interface, Ethernet port and Cellular (Optional). These Interfaces provide flexible methods for users to connect their sensor networks to Internet. LG02 can support customized LoRa transition protocol. The design of LG02 is use the Linux to directly control two sx1276/sx1278 LoRa modules which lets the LoRa can works in full duplex LoRa mode and increase the communication efficiency. LG02 can be used to provide a low cost IoT wireless solution to support 50 300 sensor nodes. LG02 can support multiply working mode such as: LoRa repeater mode, MQTT mode, TCP/IP Client mode, TCP/IP Server mode to fit different requirement for IoT connection.

   There is limitation because the LG02 only listen one Frequency and DR, while the LoRaWAN end node transmit the data on multiply frequency and different DR. For example, in EU868 LoRaWAN,

   The base requirement to fully compatible with LoRaWAN protocol requires the gateway support 8 channels. The LG02 only support two channels and can only support limited LoRaWAN protocol. Below are limitations:

   - It works only on one frequency at a time. It can support multiply end nodes, but all end nodes must transmit data at the same frequency so the LG02 can receive it. For example: if the End node transmits at 868.1Mhz, The LG02's RX setting must be 868.1Mhz so to receive this packet.
   - It works only for one DR at a time. DR specifies the Spreading Factor and Bandwidth. In LG02, even the rx frequency match , if DR doesn't match,

it still can’t get the sensor data.

- LoRaWAN compatible issue In LoRaWAN protocol, the LoRaWAN end nodes send data in a hopping frequency. Since LG02 only supports one single frequency, it will only be able to receive the packets sent from the same radio parameters (frequency and DR) in LG02.
- Downlink and OTAA issue: According to the LoRaWAN class A spec, the end node will open two receive windows to get the message from LoRaWAN server for OTAA or downlink function. These two receive windows are quite short (milliseconds), if LoRa packet from the gateway can’t reach End Node in the receive window time, the end node won’t get the rx message and downlink / OTAA won’t work.

2. LoRaWAN Gateway (Dragino model DLOS8)

   The DLOS8 is an open source outdoor LoRaWAN Gateway. Like LG02 gateway it lets bridge LoRa wireless network to an IP network via WiFi, Ethernet, 3G or 4G cellular (3G/4G is supported by optional module). The LoRa wireless allows users to send data and reach extremely long ranges at low data-rates. The DLOS8 uses Semtech packet forwarder and fully compatible with LoRaWAN protocol. It includes a SX1301 LoRaWAN concentrator, which provide 10 programmable parallel demodulation paths. DLOS8 has pre-configured standard LoRaWAN frequency bands to use for different countries. User can also customize the frequency bands to use in their own LoRaWAN network. DLOS8 can communicate with ABP LoRaWAN end node without LoRaWAN server. System integrator can use it to integrate with their existing IoT Service without set up own LoRaWAN server or use 3rd party LoRaWAN service.

   DLOS8 is fully compatible with LoRaWAN protocol. It uses the legacy Semtech Packet forwarder to forward the LoRaWAN packets to server. The structure is as below.

Then we did a comparison between lorawan gateways and their radio features to be adaptable with the Lorawan node (main radio types in recent lorawan Gateways such as sx1308, sx1301, sx1276, sx1272, sx1257)

4.2.3 LoRaWAN Backend Platforms

We evaluated most of common third-party (cloud-based) and self-deployed (local-based) LoRaWAN platforms (more than 20 platforms like Loriot, Resiot, Kerlink, Digitalnordix, Acklio, Multitech, Orange, Floranet, Xisiot, etc), performing a comparison of different platform features. Many LoRaWAN backend platforms have been developed and are currently available, under either free or commercial licenses.

User can deploy her/his own system (located on the user infrastructure) or use the online (cloud-based) system provided by a third party operator. As examples, two well-known open-source backend LoRaWAN platforms are The Things Network (TTN) and ChirpStack.

- Local (ChirpStack, Semtech Network Server, Xisiot LoRaWAN System, Gotthardp LoRaWAN Server)

- Online (e.g. RetePAIoT, The Things Network (TTN), Digitalnordix, Multi-Tech, Kerlink, Acklio, Loriot, Actility, OrbiWAN, etc) LoRaWAN Platforms comparison

LoRaWAN system is composed by three different parts: (i) the LoRa devices, (ii) one or more gateways, and (iii) a backed system that includes different functional LoRaWAN entities (network servers, join servers, and application servers).

In general these three parts are formed by hardware and software components produced by different manufacturers and possibly managed by different entities. The interoperability between these parts are guaranteed by the compliance with the LoRaWAN specifications and, between the gateway and the backend platform, by the use of standard protocols (usually the Semtech protocol or MQTT).

In this section we focused on the backend system. We present and compare different currently available platforms. We consider both (i) platforms that a user can deploy as independent instances, and/or (ii) online (cloud-based) platforms. One of the most well-known LoRaWAN backend platform is The Things Network (TTN)[3]. It is commonly known to developers as a free online cloud-based LoRaWAN platform,

[3]https://www.thethingsnetwork.org

however it provides also enterprise-level LoRaWAN commercial solutions. The TTN platform is based on their open-source LoRaWAN stack implementation[4]. TTN is most a complete and fully documented platform for both open-source/licensed and Local-based and cloud-based platform and thus could be so helpful for developers to discover DNA of lora-backend deeply. TTN is probably the most complete and fully documented platform used in several projects. It has benchmarked for 100000 devices with 12 confirmed uplinks per day. Like other platforms, TTN includes a powerful web interface for registering and configuring gateways and devices and for monitoring all activities.

TTN backend is internally formed by four different components: (i) Router (ii) Broker (iii) Handler (iiii) Discovery server.

The Router is a microservice that acts as network server. It receives messages from the gateways and finds a Broker to forward the message to. This component is tasked with handling gateway data and status, and to transfer data to the rest of the network. The Broker is a microservice that identifies the device, deduplicates traffic and forwards the packet to the Handler where the application is registered. The Handler is microservice that encrypts and decrypts the payload and publishes it to message brokers to be used by applications. The Discovery Server keeps track of various components that are registered in the network and a Network Server which contains the device registry alongside keeping track of device states.

On the interface between the network server and the gateway the TTN supports three protocols: the Semtech UDP protocol, their Gateway Connector protocol, and MQTT. Additional gateways not only will extend the capacity of the network but decrease the distance between end devices and gateways. For shorter distances, faster data rates can be used. It causes downgrade airtime and thus channel utilization. Gateway connector protocol: With this protocol Gateways are identified by an ID and by a key. Sending a message to a router requires to know the combination. Messages are sent encoded in protocol buffers. This serialisation technology allows transfer of data from a program to another, in native types, regardless of the language. With the gateway connector protocol, messages can be exchanged through two network

[4]https://github.com/TheThingsNetwork/lorawan-stack

protocols: If hardware and software supports it, through gRPC, which supports protocol buffers natively. gRPC supports TLS encryption natively. Otherwise, protocol buffer-encoded messages can be sent through MQTT. MQTTS a variant of MQTT enforces TLS encryption. Gateways can support between 8 to 64 channels, which allows millions of messages per day processed by a network. The quality of the radio network (coverage, robustness, performance, up-time, reliability) directly depends on the quality of the gateways used in the network.

The goal of TTN is to be very flexible in terms of deployment options. The preferred option is to connect to the public community network hosted by TTN Foundation or its partners. In this case the Application connects to a Public Community Network Handler usually using the MQTT API. It is also possible to deploy private networks, by running all these components in a private environment. In this way, all data will remain within the private environment, but still make use of TTN's hosted Account Server for authentication and authorization. The most simple option for this, is for someone to run his own Handler, allowing them to handle the encryption and decryption of messages. A more complicated option is a private network that exchanges data with the public network. For this to work, private Routers will have to connect to public Brokers and vice versa. In this case the private network can offload public traffic to the community network and use the public community network as backup. The latter is not yet possible with the current implementation of the TTN backend. One of the main features of TTN is Packet Broker allowing the exchange of traffic between TTN backend and private LoRaWAN networks which increases LoRaWAN network coverage, performance and capacity, and prolongs the end device battery life.

ChirpStack[5] is another popular open-source LoRaWAN backend platform. It was previously known as "Loraserver". A web interface is available for configuring the applications and devices.

They both support all existing LoRaWAN versions, operation modes A, B and C, and all regional parameters as released by the LoRa Alliance. In particular TTN provides a cloud based system that users can freely use for connecting their gateways

[5]https://www.chirpstack.io/

and devices.

Other interesting systems are: Semtech Network Server[6], Gotthardp LoRaWAN Server[7], Xisiot LoRaWAN System[8], FloraNet[9], Digitalnordix (DNX)[10], Acklio[11], MultiTech[12], Kerlink[13], Loriot[14], ResIOT[15], OrbiWAN[16], Actility[17], and Orange[18]. Some of them are based on a third pary backend platform like TTN or ChirpStack. These LoRaWAN backend platforms are compared in Table 4.1. For each platform the table reports the type of the license, whether the platform can be used as cloud service and/or can be deployed locally, the list of main extra features, whether it support the Firmware Update Over the Air (FOTA)[19] (This can be achieved using the LoRaWAN protocols using the multicast capability in conjunction with an efficient fragment scheme, which significantly reduces the number of required transmissions), and whether it is available as a released Docker image.

Moreover all these platforms conclude following features:

- Support both ABP and OTAA activation methods;
- Support A, B, and C device classes;
- Support Adaptive Data Rate (ADR used to control spreading factors, power emissions and retransmits to optimize network capacity);
- Web graphical interface (GUI) for the backend application server;

[6]https://lora-developers.semtech.com/resources/network-server
[7]https://github.com/gotthardp/lorawan-server
[8]https://github.com/xisiot/lora-network-server
[9]https://github.com/Fluent-networks/floranet
[10]https://digitalnordix.se/en/
[11]https://www.ackl.io/software/private-lora-network
[12]https://www.multitech.net/developer/software/lora/lora-network-server
[13]https://www.kerlink.com/lora
[14]https://www.Loriot.io
[15]https://www.resiot.io
[16]https://orbiwise.com/orbiwan
[17]https://www.actility.com/lorawan-network-server
[18]https://liveobjects.orange-business.com
[19]Push binary firmware updates over-the-air to a single or group of devices

Table 4.1: Comparison of LoRaWAN Platforms

Platform name	**Free/Commercial**	**Local / Cloud installation (L/C)**	**LoRa Alliance member**	**Filtering inappropriate packets**	**Major extra characteristics**	**Firmware update over the air (FOTA)**	**Docker implementation**
The Things Network (TTN)	Free for non-commercial	L,C	YES	YES	- Supports LoRaWAN 1.1 - Import bulk device (in 3rd version) - Command line interface - Packet Broker connection (exchange traffic between LoRaWAN networks) - Multi tenancy (support multiple organization which users can be assigned) - Includes OAuth 2.0 Identity Server	YES	YES
ChirpStack	Free	L	NO	NO	- Supports LoRaWAN 1.1 - Imports bulk device via gRPC/RESTful API - TLS - Multi tenancy - Channel configuration	YES	YES
Semtech Network Server	free for 3 gateways and 10 sensor nodes	L,C	YES	NO	- Developers platform based on ChirpStack - Supports only MQTT protocol (the Semthech/UDP protocol can be used installing a Gateway Bridge)	YES	YES
Gotthardp LoRaWAN Server	Free	L	NO	NO	- Integrates both network-server and application-server - Command line interface - Supports multicast to user-defined groups - Can send health alerts via e-mail or Slack - Implemented in Erlang, designed for fault-tolerant systems	YES	YES
Xisiot LoRaWAN System	Free	L	NO	NO	- Implemented in JavaScript on Node.js - A Docker image is available	NO	NO
FloraNet	Free	L	NO	NO	- Supports Class A and Class C end-devices (no class B) - No support for EU433 bands - Implemented in Python - Command line interface	NO	NO
ptNetSuite (Patavina tech)	Commercial (free for 6 months)	C	YES	YES	- Supports LoRaWAN 1.1 - Audit for suspect devices	YES	YES
Digitalnordix (DNX)	Commercial	C	YES	YES	- Quality development for network operation with Machine Learning - Command line interface	YES	YES
Acklio	Commercial	C	YES	NO	- Support external Join-Server (e.g. Gemalto) - Provided specific codec - Transfer resources between accounts - Query language - Avoid technology lock-in and silos of proprietary syntax - Device alerts - Import Bulk device/gateways - Device history retrieval	YES	NO
MultiTech	Commercial	C	YES	YES	- Command line interface	YES	YES
Kerlink	Free trial 1 month	L,C	NO	NO	- Based on Kubernete - Wanesy Management Center	YES	YES
Loriot	Free for 1 gateway and 10 sensor nodes	C	YES	YES	- Strong Logging/debugging management - SSH tunnel - TLS	YES	YES
ResIOT	Free for 1 gateway and 15 sensor nodes or 1 gateway and 5 sensor	C/L	NO	NO	- Telegram BOT - Real time dashboard alert - Email alerts with private SMTP	YES	YES
OrbiWAN	Commercial	L/C	YES	NO	- Option to recovery lost gateways e.g. via SMS - Local packet storage on gateway during back-haul failure - Encrypted and authenticated back-haul to gateways via dedicated gateway agent - Support third-party Join Servers - Storage of device data for customizable period of time - gateways alarms - Device management in groups - Import bulk device via CSV file	YES	NO

- Monitor, analyze, manage, and process received messages;
- Show gateway location on map;
- REST API, MQTT API, webhook API;
- Integration with other platforms such as AWS IoT, Azure IoT, IBM;
- Automatic decoding of application payloads (e.g. binary sensor data to JSON);
- Support custom payloads;
- Choose best gateway optimized on SNR and duty-cycle;
- Logging of gateway events and MAC messages;
- Possibility to export logging data;
- Set airtime limits for uplink and downlink per users and per devices.

Preliminary Experimentation and Tests

Evaluation has been done on LoRaWAN technologies, and corresponding experimentation done for the transmission of sensor data in precision agriculture scenarios.

We did some experiments with physical IoT devices connected to either cloud-based or local-based LoRaWAN backend systems. In particular we connected LoRaWAN end nodes to a LoRa gateway; we connected LoRa devices to some cloud-based LoRaWAN platforms; we connected above components to a local-based LoRaWAN platform; we designed and connected virtual LoRaWAN nodes and virtual LoRaWAN gateways to the local-based and cloud-based LoRaWAN platforms, and finally we did the prototyping of the LoRaWAN end node capable of connecting various industrial sensors.

We started experimentation with running a private LoRa network and transmitting packets between LoRaWAN devices and a gateway sited in the lab. We used two LoRaWAN devices (Dragino LSE01 and Dragino LHT65) and a LoRa gateway

(Dragino LG02). Standard LoRaWAN devices transmit packets in 8 different channels while the gateway we used for the experimentation was a LoRa gateway (not LoRaWAN gateway) which works only with a single channel. This means that it could just transmit packets in 1 channel while LoRaWAN devices were transmitting packets to it in 8 different channels so we tried to modify the device configuration to solve this issue. We finally succeeded in letting the device joining a backend system via the single channel gateway. In order to communicate with the device via PC we used a USB-TTL-converter interface and a middle-ware to be able to run the AT-commands in the LoRa device.

In second activity we did experimentation with a cloud-based platform, The Things Network (TTN), using the two LoRa devices to transmit their data to this platform. By setting up necessary applications and parameters in TTN console we transmitted data to TTN and vice versa (as downlink) from TTN platform to LoRaWAN end node through the LoRa gateway.

Then in third activity we experienced transmitting sensor data to RetePAIoT which is another cloud-based platform provided by Lepida. It includes several LoRaWAN gateways and LoRaWAN backend network-server and application-server. First we tested the signal strength between a LoRa node, sited in the Campus area of the University of Parma (more precisely on the parking area behind building 2 of the Department of Engineering and Architecture) and a RetePAIoT gateway, sited in the center of Parma (Piazza Garibaldi). It resulted in a very high distance for joining the LoRaWAN node to the LoRaWAN backend without any repeater or gateway in the middle (around 6 Km distance including buildings and trees).

In fourth activity we locally deployed a LoRa backend system which could act as a complete LoRaWAN backend. It includes the following main components: LoRaWAN network-server, join- server, application-server and gateway-bridge. For this purpose we used Chirpstack which is a popular LoRaWAN platform (Chirptack is also used for local LoRaWAN platform on Google Cloud). In this scenario we set up a PostgreSQL database, prerequisites and related configurations to store data of the gateway-bridge, application-server and network-server; then we set up backend components and their configuration files. We also implemented a LoRa-virtual-gateway in Java (we are going to release it publicly) that is capable to run several gateway instances and several virtual device instances at the same time. It also emulates different sensors including the Dragino LHT65 and LSE01. Furthermore we implemented a RetePAIoT virtual device which sends the data obtained from RetePAIoT application-server (via RetePAIoT token) using REST API. We used these virtual gateways to test the local backend. We set up and started web interface of the the backend and we obtained the frame payload of virtual LoRa device transmitted to the application-server and web interface. We created simple java-script functions to decode the uplink payloads and encode the downlink payloads. By accessing the REST API of the Chirpstack we were able to transfer the payload in another web application-server or any web server e.g. create a XAMP server and access the frame payload via Curl commands; payloads can be saved also in the local host (in a text file or in a DBMS). In particular in the heart of Chirpstack backend there are two MQTT brokers: (i) gateway-broker which interacts with gateway-bridge and network-server; (ii) integration-broker which interacts with gateway-bridge and application-server.

Reverse engineering on Chirpstack: According to our last discuss we subscribed in below two ways to our chirp-stack broker and received the payloads but as shown in screenshot they are encoded (i) Mqtt-box (an extension for chrome) (ii) Commands Terminal (Subscription to the Topic $gateway/+/event/+$)

Based on the attached logs (from gateway-bridge, net-server, app-server) there is only one broker in chirp stack backend which works in two ways: (i) gateway/mqtt (interact with gateway-bridge,net-server)

(ii) integration/mqtt (interact with gateway-bridge, app-server) Not only we attached all components log-file but also we separated Pub Sub mechanism from all of them as shown in the attached file. By subscribing to the broker we could receive payloads due to their related TOPIC.

After subscription on broker, is it possible to decode the payloads. By publishing a json string on the broker (on this topic: gateway/35506e3b00b176bd/event) is it possible to receive this payload in chirp-application-server.

In fifth activity we started studying hardware components to prototype a LoRaWAN end node. Different assemble scenarios with various micro-controllers (e.g.

Atmega, STM32, ESP32), and different boards (e.g. PI, Arduino, Dragino, Adafruit, Reyax, Pycom, Seedstudio, etc) were considered and pros and cons of each scenario have been evaluated. According to the above study a possible solution could be using the STM32WLX board that integrates a MCU and a LoRa radio module in one chip, performing low power consumption and high performance. An experimentation activity with this board has been started.

We did also some traffic analysis by filtering and decoding the inner payloads of the gateway packets. For this purpose we developed proper tools in Python and Java.

In order to connect lora nodes via USB-Ttl to TTN following steps done:

- On LSE01 board set FLASH-mode for (change parameters) programming, also the way shows RXD/TXD has been attached to the converter are vise versa.
- While the device is sending join-req it is possible to write commands, or wait until 5 min to finish its sending.
- In the user manual shows it can connect without need to attach sensation cable (black -yellow - green) but without attaching those cables to the board it doesn't work.
- Steps should be in this order first turn off device, connect all wires then turn on device.

In order to work the lora device with single channel gateway (LG02) we set following parameters via AT-command by attaching the device via USB-Ttl converter. To return back to OTAA activation and standard Lorawan setting we used AT-commands too.

1. Build LoRaWAN end node using L072ZLRWAN STM32 board and transmit data to TTN

   A lorawan end node have created by setting the keys of the device in TTN server platform. Device parameters Configured, codes uploaded (Flashed) to the device via STM32-cube-IDE tool and finally activate it in the TTN to transfer the packets. Voltage and the temperature of the MCU in the board were visible in the TTN online platform.

Description: First, choose and install the best tool (tool-chain) to configure and upload the codes(FLASH) the device. Cube-IDE includes CubeMX and Cube programmer tools (for Linux OS) and could be the best between IAR, EWARM, KAIL, Arm and mbed online tools. Secondly, download the LoRaWAN end node example and import it to the Cube IDE. Thirdly, upload the LoRaWAN application code to the device. Fourthly, choose and install a terminal tool for tracking the packets. Finally, creating an account and other entities in TTN (e.g. add gateway, application and a device).

Hardware prerequisites:

- BL072Z LRWAN STM32 BOARD
- Type C USB cable (connect board to the PC)

Software equipment:

- Cube-IDE software
- Cutecome terminal application (or any terminal simulator application)

There are some additional resources regarding these simple experimentation:

- Frequency setting according to the zone to set in the lora device and TTN (Frequency plan) [20]
- Stm32 BL072Z LRWAN pins out [21]

2. Build LoRaWAN end node using WL55JC STM32 NUCLEO board and transmit data to TTN

   Description: First we created a project in cube-IDE Setting the keys and parameters of Lorawan end node then Save, we compiled and Debug the project. Then we checked Terminal tool (Cutecome) the data transmission. We checked the data is transmitted to TTN (Create an application, end-device) thing network

[20]https://www.thethingsindustries.com/docs/reference/frequency-plans/
[21]https://os.mbed.com/platforms/ST-Discovery-LRWAN1/

(TTN) version 1.0.3 is TTI(The Thing Industrial) or The Things Stack Community or The Things Stack. One of the most energy efficient boards between Dragino, Pycom, Arduino, Raspberry pi, Adafruit, Esp32 and Semtech in 2021 is stm32 nucleo wl55Jc. It is one of the best choices in case of performance too. It includes all necessary pins and equipment to prototype a LoRaWAN and other prototyping.

Hardware prerequisites:

- WL55JC STM32 NUCLEOBOARD
- Type C USB cable (connect board to the PC)

Software equipment:

- Cube-IDE software
- Cutecome terminal application (or any terminal simulator application)

There are some additional resources regarding these simple experimentation:

- Video of deploying the scenario [22]
- Frequency setting according to the zone to set in the lora device and TTN (Frequency plan)[23]
- In addition an introduction Getting Started with STM32 and LoRaWAN (STM32 LoRaWAN gateway for The Things Stack) [24]

3. Retrieve DHT11 sensor data to L476RG STM32 board:

Goal is to transmitting Temperature and Humidity of DHT11 sensor via L476RG STM32 board to output (In a Terminal simulator application).

Description: Based on the usermanual of DHT11 a library wrote to implement 4 steps mentioned in the usermanual to be able to read DHT11 sensor data (Temperature and Humidity) and interact with DHT11 to read it's data. Main functions defined

[22]https://drive.google.com/drive/folders/193a0y-XQt243AbnOHGVEFdcdlx3MfqSE?usp=sharing
[23]https://www.thethingsindustries.com/docs/reference/frequency-plans/
[24]https://www.youtube.com/watch?v=uJU3YM1MWN4

are (DHT11 start, Check response, DHT11 read) but besides there would be needed other functions ((Delay, Set Pin input, Set Pin output). Pull up and pull down procedures for the MCU pin(MCU Input Output or GPIO input/output) implemented and set them to high or low based on the DHT11 usermanual. Defining timer and related interrupt parameters studied. Then a timer defined to be used in our DELAY function in our code to be able to interact with DHT11 sensor and read its data.

Hardware equipment:

- DHT11 sensor
- L476RG STM32 board
- Related USB cable (connect board to the PC)
- 3 Jumper wires

Software equipment:

- Cube IDE software
- Cutecome terminal application (or any terminal simulator application)

There are some additional resources regarding these simple experimentation:

- Video of deploying the scenario [25]
- Links for downloading STM cube IDE [26]
- L476RG STM32 BOARD pins out [27]
- DHT11 usermanual [28]

[25]https://drive.google.com/drive/folders/193a0y-XQt243AbnOHGVEFdcdlx3MfqSE?usp=sharing

[26]https://www.st.com/en/development-tools/stm32cubeide.html

[27]https://os.mbed.com/platforms/ST-Nucleo-L476RG/

[28]https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

4.2.4 Interworking between LoRaWAN and Non-LoRa IoT Systems

IoT may be a later communication worldview that envisions a near future, in which the objects of way of life will be prepared with microcontrollers and transceivers for advanced communication, and reasonable protocol stacks that will make them able to communicate with one another and with the users, getting to be an necessarily portion of the internet.

Such a heterogeneous field of application makes the distinguishing proof of arrangements able of fulfilling the prerequisites of all conceivable application scenarios a impressive challenge. This trouble has driven to the expansion of diverse and, some of the time, contradictory proposition for the viable realization of IoT frameworks.

LoRa is an IoT communication technology that provides very long power transmission using very low power consumption. LoRaWAN specifications directly face device battery life, network capacity, quality of service, security, and the variety of applications provided by the network. LoRaWAN devices can be seen as LoRa converters which convert sensor digital data (e.g. soil moisture) to LoRa messages and send them to an application-server through one or more LoRaWAN gateways and proper backend system.

LoRaWAN is an efficient and easy-to-deploy complete solution for LPWAN IoT scenarios. However, since it specifies all protocol layers from LoRa radio access to the application, the interoperability with non-loRa devices and/or non-LoRaWAN backend is not straightforward and it is left to the user.

An interesting system analysis of LoRaWAN is presented in [59]. The authors highlight the following main strengths of LoRaWAN: (i) offer cheap end devices and possibilities of deploying private networks, (ii) remove the need for service subscription such as the one in SigFox or NB-IoT, (iii) decrease the operational costs for dense IoT deployments, iv) have large coverage with single gateway and finally, v) provide low power operation for end nodes.

Instead in [60] a large-scale measurement study is presented. In particular it is shown that LoRa performance is severely affected by obstructions such as buildings and vegetation. Moreover, the promise of prolonged battery life requires extreme tuning of parameters.

In [61] authors depicted taxonomies of the design and implementation of an Interoperability in Internet of Things.

A good description of the design and implementation of an interworking IoT platform in cloud is provided in [62]

There are implementation of interworking between IoT devices in [63] and a great implementation of interworking between IoT devices and IoT systems in [64]

Model for Interworking between LoRaWAN and Non-LoRa IoT Systems:

According to our paper [65] we proposed solutions for different interworking scenarios where:

- one or more non-LoRaWAN devices have to be connected to a LoRaWAN backend system;
- a set LoRaWAN devices, connected to a LoRaWAN backend, must interact with a non-LoraWAN IoT system;
- different LoRaWAN and non-LoRaWAN networks, possibly implemented through different platforms, have to be interconnected forming a larger heterogeneous IoT system.

Connecting Non-LoRa Devices to LoRaWAN

The first case we consider is when a non-LoRa sensor or actuator has to be connected to a LoRaWAN backend system. This could be useful in application scenarios where both LoRaWAN and non-LoRa devices should coexist and a LoRaWAN backend system is used as application platform. The non-LoRa devices may support other connectivity technologies, either wireless or wired.

To face this problem two approaches can be used:

- connecting the non-LoRa devices directly to the backend systems using non-standard (non-LoRaWAN) interfaces;
- connecting the non-LoRa devices to a proper virtual gateway acting as interworking element between the non-LoRa devices and the LoRaWAN backend system.

Although the first approach can be used in some applications, the latter is more convenient since it is independent from the particular backend platform. This approach can be further extended by decoupling the interworking element (between the non-LoRa device and LoRaWAN protocols) from the actual gateway functionality. In this case, from the gateway point of view, the interworking element acts on behalf of the original non-LoRa device, representing a new LoRaWAN device.

A further step is to remove the possible unnecessary LoRa link between the interworking element and the gateway, simplifying the resulting architecture (in this case the interworking element may also be conveniently located on the side of the gateway or connected via an IP network). This last scenario is depicted in Fig. 4.1, where the interworking element is refereed to as virtual device, and is connected to the gateway via a non-LoRa interface. In order to distinguish this non-LoRa gateway from a standard LoRaWAN one, we refer to it as virtual getaway. This two entities can be characterized as follows:

- Virtual LoRaWAN device: it is a node that implements the LoRaWAN MAC layer and above and exchanges corresponding packets with a virtual LoRaWAN gateway and a remote LoRaWAN network;
- Virtual LoRaWAN gateway: it is a gateway node that can be connected to a standard LoRaWAN network (network server, join server and application server), possibly using standard mechanisms (e.g. MQTT or Semtech protocols), and performs standard proxing functionality by relaying LoRaWAN MAC packets between some virtual devices and the LoRaWAN backend system.

From the protocol architecture point of view, a virtual LoRaWAN device is a standard LoRaWAN client node where the LoRaWAN physical layer, normally used to physically communicate with the gateway and for transmitting LoRaWAN MAC packets, is replaced by a different transmission mechanism. In case of a packet network is available between the virtual device and the gateway (e.g. IEEE 802.11 WiFi, IEEE 802.15.4, Ethernet, IP), the corresponding communication protocols can be used. In particular the LoRaWAN MAC packets can be encapsulated directly as

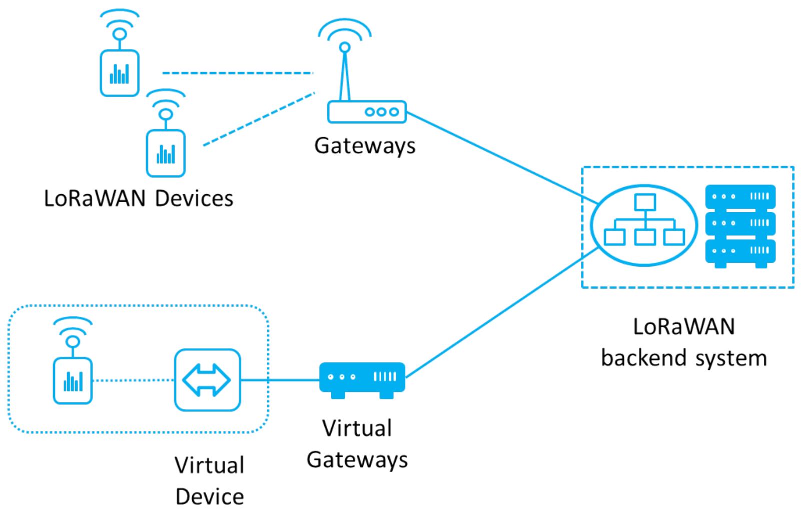

Figure 4.1: Virtual gateway scenario.

datagram payload within any datagram protocol (e.g. Ethernet, IP, or UDP), without requiring any additional adaptation layer. In our implementation we used UDP as encapsulating protocol. The UDP has been selected in order to guarantee easy portability on several types of devices with different access technologies. The same protocol must be implemented on both sides: the device and the virtual gateway. Then the gateway will extract LoRaWAN MAC packets coming from the devices and forward them to the backend system (actually a LoRaWAN network server). Similarly, LoRaWAN MAC packets in the opposite direction are encapsulated and sent to the virtual device. The resulting protocol architecture is depicted in Fig. 4.2.

Note that the new LoRaWAN virtual devices can be useful also in the following applications:

- Realization of a *digital twin*, that is a virtual instance or digital representation of a physical system that includes properties, conditions, and behavior of a corresponding real-life system and can be used for either testing or planning purposes in different application scenarios such as Industry 4.0, smart cities, healthcare, etc [66].

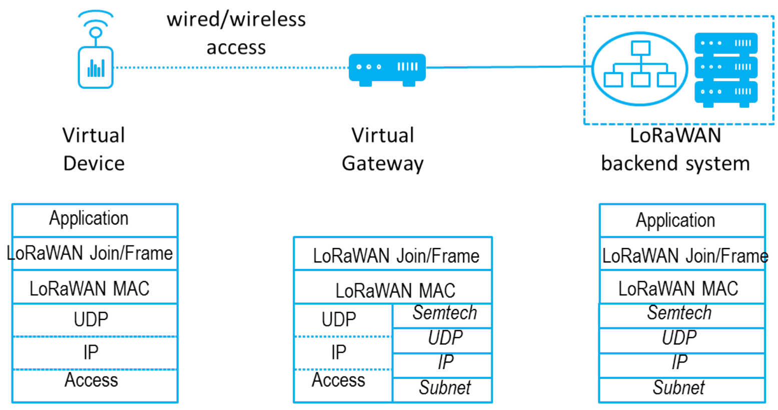

Figure 4.2: Virtual gateway protocol architecture.

- Implementation of a *sidecar object*, that is a separate piece of software dedicated to handling specific issues, which are not directly related to the smart object/device and should be managed outside the object's scope and/or realization [67]. It is deployed as additional component of a pre-existing object without interfering with the mechanisms and behaviors already implemented.

A special application for LoRaWAN virtual devices is when physical non-LoRa devices are requested to be connected to a LoRaWAN system. Two approaches can be considered depending on whether the non-LoRa devices can be modified or not:

1. non-LoRa devices are extended in order to support the protocol architecture described in Fig. 4.2 and to directly communicate with the virtual gateway;

2. a separate virtual device is implemented and used as interworking element between the physical device(s) and the gateway.

Of course the advantage of using an independent element is that existing (off-the-shelf) non-LoRa devices can be easily integrated without any modification.

Regarding the physical deployment of the virtual devices, we can consider two possibilities: (i) a single element (virtual device) for each physical non-loRa device, or (ii) a single element for all non-LoRa devices, providing the virtualization of these devices. In this case this element can coincide also with a IoT gateway, if preset in the non-LoRa network.

A special case is when the non-LoRa devices use a standard application protocol like CoAP [68] or MQTT [6] for communicating with other nodes. This is focused in the following Subsec. 4.2.4 and Subsec. 4.2.4.

CoAP-to-LoRaWAN Virtual Devices

In order to connect a CoAP-based smart object (either a sensor or actuator) to a LoRaWAN system, a pair of virtual LoRaWAN device and gateway can be used. The LoRaWAN virtual device will include both the upper layers of the LoRaWAN stack, for communicating with the gateway and with the LoRaWAN servers, and the CoAP protocol, for communicating with the smart object. The resulting protocol architecture is depicted in Fig. 4.3.

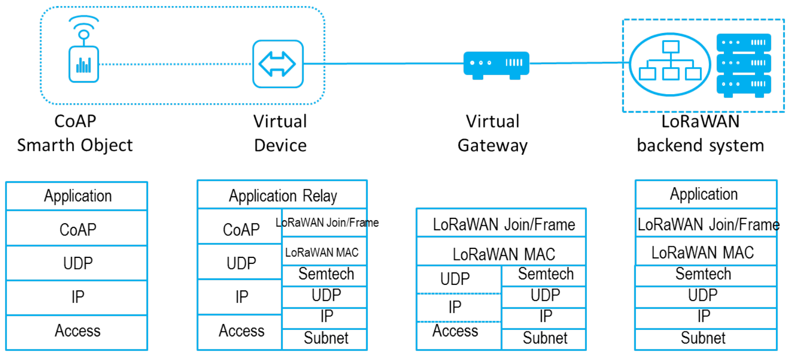

Figure 4.3: CoAP virtual device protocol architecture.

The actual CoAP role (client or server) and the type of interaction implemented by the virtual device, depend on the communication paradigm implemented by the CoAP smart object. The following cases can be distinguished:

- The smart object is a CoAP client and produces and/or consumes data. In this case the virtual device should implement a CoAP server supporting the corresponding methods (PUT and/or GET). In particular it is a PUT server in case the object is a producer, GET server in case the object is a consumer

- The smart object is a CoAP server and provides support for reading data from it and/or writing data to it. In this case the virtual device should implement a CoAP client with the corresponding methods (GET and/or PUT)

Note that both asynchronous data request/response or synchronous data publish/subscribe interactions can be supported, based on the application scenario and the communication paradigm supported by the CoAP object. In case of a publish/subscribe interaction, the CoAP *observe* extension [69] should be used.

MQTT-to-LoRaWAN Virtual Devices

Similarly, in case a MQTT-based smart object has to be connected to a LoRaWAN system, a pair of virtual LoRaWAN device and gateway can be used too. In this case both the smart object and the virtual device will implement MQTT client that subscribes and publishes data on a remote MQTT broker. Depending on the role of the smart object (publisher or subscriber or both), the virtual device will implement the corresponding opposite method (subscribe or publish).

For example if the smart object is a sensor that periodically publishes data on a MQTT broker, the virtual device will subscribe for the corresponding topic on the same broker. Each time that new data is published, a MQTT publish message is received by the virtual device that will forward it as a LoRaWAN data message to the virtual gateway that will relay it to the LoRaWAN application server. Similarly, if the smart object is an actuator that subscribes for a given topic on a MQTT broker, the virtual device will publish on the same broker with the same topic the payload data received from the LoRaWAN application server through the virtual gateway.

LoRaWAN-to-CoAP Proxy

In case some LoRaWAN devices connected to a LoRaWAN backend system have to be integrated into an IoT system, other mechanisms have to be properly introduced. In particular if the devices have to be controlled/accessed via REST API based on the standard CoAP protocol, a LoRaWAN-to-CoAP interworking element is required. This element could be either:

- A LoRaWAN-to-CoAP *proxy* that includes both a LoRaWAN application server and a CoAP client, letting the client forward LoRaWAN application payload to a remote CoAP server (sent through CoAP PUT requests), and/or send back to the LoRaWAN device data received from a CoAP server (as a result of GET request/response exchanges or as result of a publish/subscribe service according to the CoAP observe extension [69]);

- A LoRaWAN-to-CoAP *reverse proxy* that acts externally as CoAP server supporting PUT and/or GET methods, possibly providing also a publish/subscribe paradigm with the CoAP observe extension. The data is retrieved from or sent to the LoRaWAN device via an internal or external LoRAWAN application server.

Depending on whether the LoRAWAN application server is implemented within the proxy or it is deployed as independent element, the interaction with the application server is done via external (e.g. REST or MQTT) or internal APIs respectively, as depicted in Fig. 4.4.

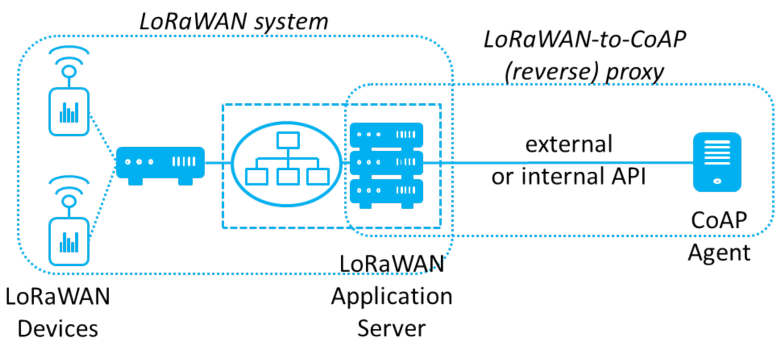

Figure 4.4: LoRaWAN-to-CoAP proxy architectures.

Larger Data-centric IoT Platform

In this section we try to further extend the interworking scenario by creating a larger data-centric (data-driven) IoT system with heterogeneous access technologies. The objective is to allow a user to receive data from and/or to send data to devices associated to different LoRaWAN networks and non-LoRa IoT networks, in a simple, transparent, and scalable way.

For this scope we consider an architecture where different LoRaWAN and non-LoRaWAN networks are interconnected through an intermediate system providing a standard application interface to the user clients. The LoRaWAN networks may also be heterogeneous in terms of inner implementations and used protocols (e.g. the protocol used between the gateway and network server, or the protocol used between the application server and application client).

The proposed architecture uses the publish/subscribe paradigm, through the MQTT protocol and a MQTT broker, for exchanging data between IoT subsystems and user clients. The resulting system is depicted in Fig. 4.5.

In order to connect an IoT subsystems (e.g. a LoRaWAN network) to the central MQTT broker, if MQTT is not natively supported by the subsystems as interface protocol, a proper adapter system has to be included. In case of LoRaWAN IoT subsystems, this adapter system includes both a client used to exchange data with the LoRaWAN application server, and a MQTT client used to communicate with the MQTT broker.

Network of Brokers

This architecture can be further extended in order to obtain better performances in presence of a large number of connected devices, LoRaWAN networks, and user clients.

In order to achieve more scalability and reliability the single broker can be replaced by a network of brokers. For example, a possible scenario is the one where there is one broker for each LoRaWAN network. In general, each LoRaWAN network may be connected to one or more MQTT brokers, and the same applies for the

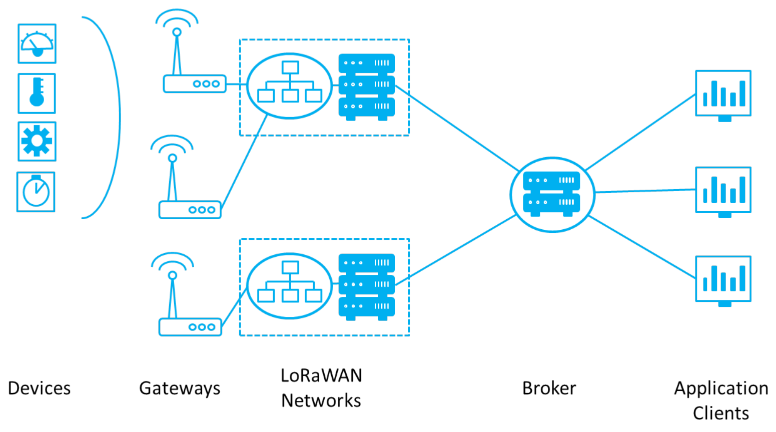

Figure 4.5: Proposed architecture with single broker.

user clients. However, the most common scenario is the one where each LoRaWAN network is connected to a single broker, and each client also connects and uses just one broker for sending/receiving data from different LoRaWAN networks. The new architecture is depicted in Fig. 4.6.

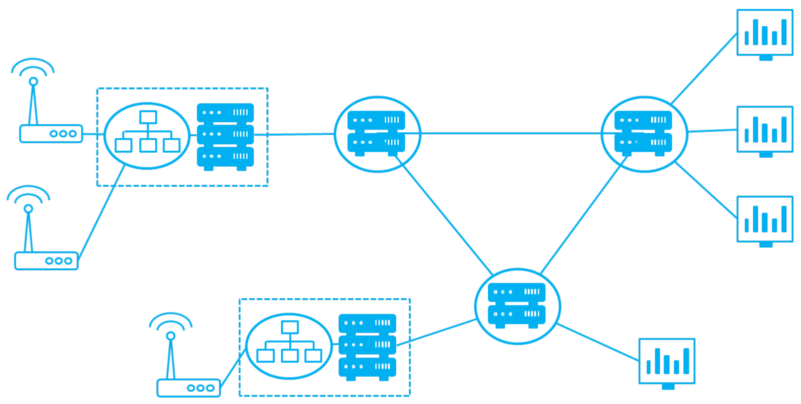

Figure 4.6: Proposed architecture with network of brokers.

4.2.5 Testbed

The proposed interworking systems described in Subsec. 4.2.4 and the extended network architecture have been implemented and tested.

A completely new light Java implementation of the LoRaWAN MAC, Join/Answer, and Frame layers has been realized and released[29] as open-source.

The current release includes also the implementation of:

- a *virtual gateway*, that can be used to connect to any standard LoRaWAN platform;
- a *virtual device*, that can be used to virtualize any non LoRa devices.

The release has been further extended by including:

- a *CoAP client* virtual device - that is a LoRaWAN virtual device that periodically retrieves data from an external CoAP server and sends it to the LoRaWAN application server;
- a *CoAP server* virtual device - that is a LoRaWAN virtual device that includes a CoAP server allowing one or more resources to be updated via the CoAP PUT method. The updated data is then automatically sent to the LoRaWAN application server.

The implementation of the CoAP components has been based on the mjCoAP library [70], a light CoAP Java implementation available open-source[30].

For testing the LoRaWAN functionalities two different LoRaWAN open networks have been used: the well-known TTN platform[31] and RetePAIoT[32]. TTN provides a cloud-based free LoRaWAN backend (network server, join server, application server) integrated with a powerful web interface with a user console, together with two different APIs (HTTP REST API and MQTT) for remotely connect user own applications.

[29]https://github.com/thingsstack/lorawan
[30]https://github.com/thingsstack/mjcoap
[31]https://www.thethingsnetwork.org
[32]https://www.retepaiot.it

Since TTN provides only the LoRaWAN backend, we connected two physical gateways to it: a Dragino OLG02[33], and a Dragino DLOS8[34].

As physical devices connected to the two gateways we used a Dragino LSE01[35], that is a soil moisture sensor, two custom temperature sensors based on the STM32 B-L072Z-LRWAN1[36] and STM32 NUCLEO-WL55JC1[37] boards, respectively, and two Arduino-based sensors with Dragino LoRa Shield[38].

In addition we connected to TTN our virtual gateway with different virtual devices including a CoAP-client virtual device and a CoAP-server virtual device.

The overall testbed is depicted in Fig. 4.7.

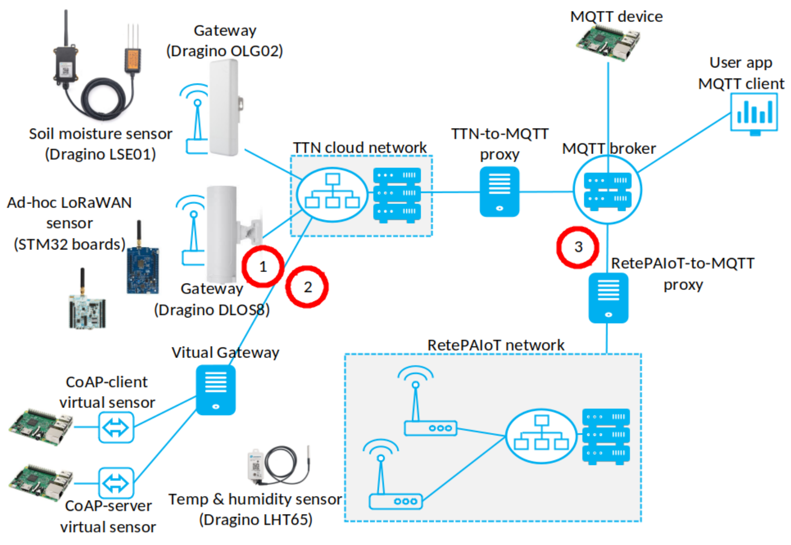

Figure 4.7: Implemented testbed.

In order to access to the sensor data, we used both the TTN HTTP REST API

[33]https://dragino.com/products/lora-lorawan-gateway/item/135-lg02.html
[34]https://dragino.com/products/lora-lorawan-gateway/item/160-dlos8.html
[35]https://dragino.com/products/lora-lorawan-end-node/item/159-lse01.html
[36]https://www.st.com/en/evaluation-tools/b-l072z-lrwan1.html
[37]https://www.st.com/en/evaluation-tools/nucleo-wl55jc.html
[38]https://dragino.com/products/lora/item/102-lora-shield.html

and the MQTT interface. While for accessing data via MQTT we used a stand alone MQTT client, for accessing data via REST API we developed a proper TTN client[39] and we integrated it in a new TTN-to-MQTT proxy as depicted in Fig. 4.7.

The second network is the RetePAIoT, a LoRaWAN open network operated by an Italian regional public administration, aiming to provide an open IoT network for both citizens and companies. In this case the network inlcudes both a LoRaWAN backend system and several physical LoRaWAN gateways deployed within the regional area. We connected two temperature and humidity sensors (Dragino LHT65[40]) and Arduino-based LoRaWAN devices, and we used the REST API provided by the RetePAIoT for interacting between the application server and a client included within our RetePaIoT-to-MQTT proxy depicted in Fig. 4.7.

In addition, we set up a LoRaWAN backend platform based on ChirpStack that we have used to connect a Raspberry Pi based LoRaWAN gateway and Arduino LoRaWAN devices. We also connected other virtual devices (digital twins of LHT65 and LSE01 sensors) to this backend through a virtual gateway.

ChirpStack LoRaWAN platform:

The aforementioned scenario also implemented with ChirpStack LoRaWAN platform too as a local platform instead of TTN platform which is on the cloud. Here just implementation of ChirpStack Local LoRaWAN Platform described.

After setting up prerequisites and their related configurations to store data of application server and network server we set up the mentioned backend components and their related configuration files. Steps are as follows:

- set up postgre SQL database and related configurations to store data of application server and network server; Setup ChirpStack software repository

- set up following components and their configuration files; chirpstack gateway bridge; chirp network Server; chirpstack Application Server

- check that the server is running; check if packet forwarder sends data

[39]https://github.com/thingsstack/ttnapi

[40]https://dragino.com/products/temperature-humidity-sensor/item/151-lht65.html

- generate the data from the virtual LoRa gateway (e.g. device-counter)
- set up and start a web interface for application server using its configuration, inside the application server: Define a network server; Define organization; Create service profile; Create a device profile; Define gateway, application and device
- receive the payload of the virtual device on the web interface (application-server) also in the logs of virtual gateway in the terminal;

A java-script function also added for decoding the received sensor data. Moreover by accessing the ChirpStack REST API we were able to receive the application payload from a different web application.

Description of the Graphs in Fig. 4.8

- Graphs 1 and 4 report outdoor temperature and humidity measured at the university Campus in Parma. The values are captured using a LoRaWAN sensor and transmitted to the TheThinksNetwork (TTN) platform (via a LoRaWAN Gateway (Dragino DLOS8) placed at the Department of Engineering and Architecture Using a TTN webhook, data is automatically push to a server of our laboratory that resends it to the Data Server (AllThingsTalk)
- Graph 2 shows temperature sent by a virtual LoRaWAN sensor to TTN via a virtual LoRaWAN gateway temperature values are obtained from a remote service (OpenWeather) Using TTN webhook, data is re-sent real-time to WEB api (AllThingsTalk)
- Graph 3 shows the outdoor temperature measured in Downtown Parma by a LoRaWAN sensor Connected to the RetePAIoT LoRaWAN (via the LoRaWAN gateway present on top of the Parma's City Hall) Values are automatically relayed by the application server (using the RetePAIoT REST PUSH API) to a server inside our laboratory. the graph plots values captured during the last year, stored in a DB and sent to the Data Server via a RetePAIoT-to-Data-Server proxy

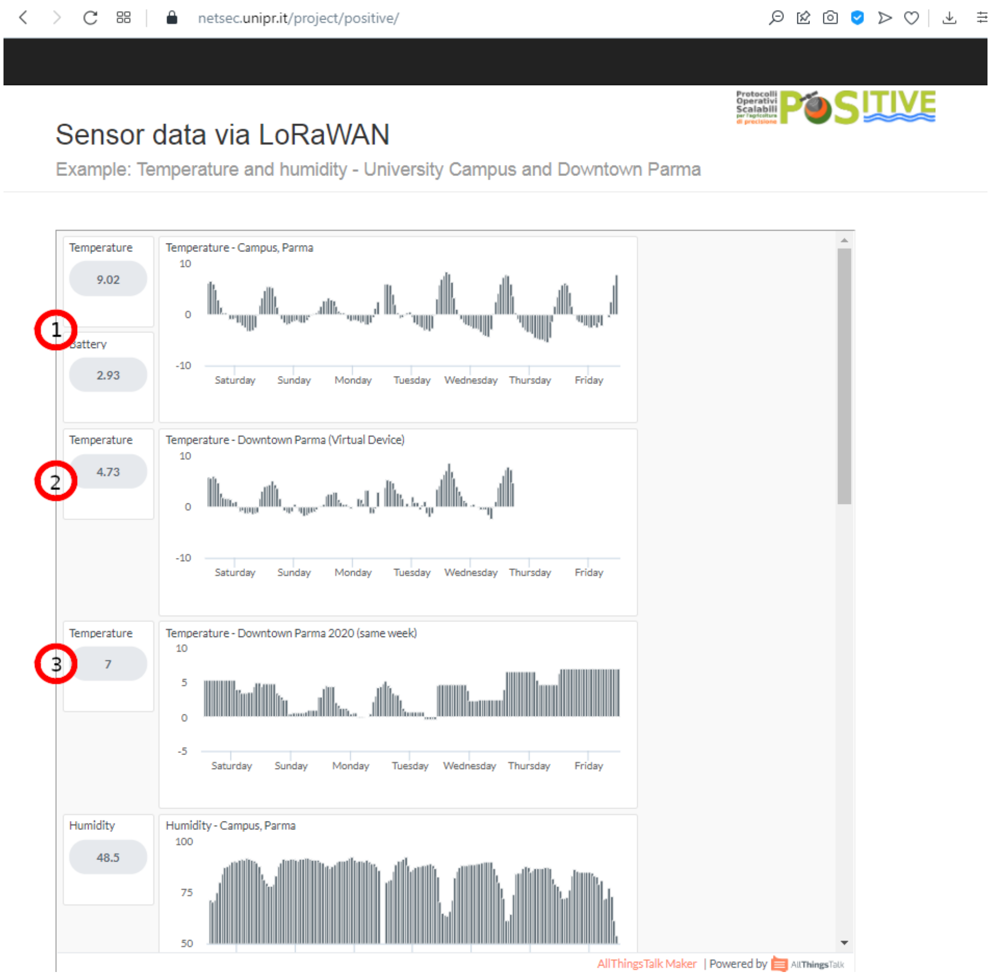
netsec.unipr.it/project/positive/
Protocolli Operativi Scalabili per l'agricoltura di precisione
POSITIVE
Sensor data via LoRaWAN
Example: Temperature and humidity - University Campus and Downtown Parma
Temperature
9.02
Battery
2.93
Temperature - Campus, Parma
Temperature
4.73
Temperature - Downtown Parma (Virtual Device)
Temperature
7
Temperature - Downtown Parma 2020 (same week)
Humidity
48.5
Humidity - Campus, Parma
Saturday
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
1
2
3
AllThingsTalk Maker | Powered by AllThingsTalk

Figure 4.8: Temperature and humidity - University Campus and Downtown Parma.

- As Data Server we used the AllThigsTalk platform

Chapter 5

Conclusions

LoRaWAN is an efficient and easy-to-deploy complete solution for LPWAN IoT scenarios. The main advantages of LoRaWAN are in terms of long range connectivity and indoor coverage, long battery life, high capacity in term of number of connected devices, low-cost for devices and infrastructure hardware, standard architecture from end devices to application servers.

In this thesis novel IoT technologies and security aspects including threats and vulnerabilities in IoT and LoRaWAN has been described besides the corresponding countermeasures counted. We found that deep learning using anomaly detection is most proper approach to combat IoT attacks so we designed a model within the following steps to discover attacks in existed IoT traffic datasets. A large dataset of IoT concludes benign and malicious flows analyzed, it has been built by integrating four different recent datasets. The analysis has been carried out using both traditional machine learning algorithms and a proper Deep Leaning architecture that resulted to achieve very good results and to outperform traditional machine learning techniques, in terms of overall accuracy, precision, recall and F-measure, for both a binary and multi-classification.

As application scenario, we considered the case when heterogeneous sensors have to be connected to a backend IoT system. In particular we focused on an access scenario based on LoRaWAN technology. However, in order to let different IoT access technologies to coexist we designed and implemented an interworking IoT platform which is able to connect heterogeneous IoT devices and networks to gather via virtual components. We tried to face the interoperability issue by proposing different interworking solutions. We considered both cases where one or more non-LoRaWAN devices have to be connected to a LoRaWAN backend system, and where a set LoRaWAN devices, connected to a LoRaWAN backend, have to interact with a non-LoraWAN IoT system. The case where different LoRaWAN and non-LoRaWAN networks have to be interconnected forming a larger heterogeneous IoT system has been also considered. The proposed solution have been implemented and tested. The code of the developed systems has been publicly and freely released. Finally we created an online graph from our testbed. It demonstrates the aggregated data from different categories of sensors (Adhoc devices, Off-the-shelf devices and Virtual devices) and explained each flow of the data.

5.1 Future Work

In future work, we will focus on integrating much more datasets and instances to apply the proposed Deep leaning model with much more layers, and on techniques of feature selection to verify whether all the considered features are useful or not for the classification tasks we considered. A further future improvement may also regard the consideration of different types of neural networks, such as LSTM, and different architectures of the neural network itself.

Intrusion is defined as an unauthorized task that is executed by an intruder (adversary) in a network. Based on the adversary's capabilities, an attack can be passive (i.e., eaves- dropping of information during communication) or active (i.e., malicious packet injection and packet dropping). An intrusion detection system (IDS) is treated as a device (software) application that can detect any suspicious event happening in the network. Once a malicious IoT device is detected by an adversary, so a feature needed to be added in future IDS to quickly detect the identity of that device so that further damage in the system cannot occur.

Moreover we expect that future IDS systems need to be designed and deployed in resource-constrained IoT devices. Due to resource limitations of IoT devices, we need to design lightweight IDS mechanisms for securing the IoT environment. Thus, managing machine-to-machine authorization mechanisms among IoT devices using the IDS system will be another interesting future work.

Experimentation showing the overhead in terms of performance added by the architecture could be an improvement so solving the problem of overhead besides increase accuracy of the neural network model is one of the future aims.

Increasing the end-to-end data security with LoRaWAN by dealing with the re-generation (and installation) of the secret key when the application server changes is a future direction.

Larger deployment of outdoor sensors and data collection is another plan for the future.

According to the all-in-one architecture of LoRaWAN lacks in interoperability with non-LoRa IoT systems, that is left to the users therefore, designing lightweight and secure privacy-preserving techniques in resource-constrained IoT devices is an interesting future research work.

As there are very few approach to detect MQTT exploit it could be possible way to use machine learning approaches to over come this issue.

In the future, IoT will play a significant role in the next-generation agriculture. IoT connects monitoring devices to cloud platforms thereby enabling a agriculture systems in real-time and in a non-invasive way. Since the information is confidential, the information collected by the IoT devices must not be disclosed to any unauthorized party. Hence, secure the monitoring system by leveraging novel lightweight and secure device authentication for device-to-device communications is desired.

Another improvement method to come over threats is anomaly detection which performed directly onto the possibly large dataset formed by the sensor data.

5.2 Publications

- Riccardo Pecori, Amin Tayebi, Armando Vannucci and Luca Veltri. "IoT Attack Detection with Deep Learning Analysis" IEEE WCCI-IJCNN 2020.

- Amin TAYEBI, Luca VELTRI, Francesco ZANICHELLI and Stefano CASELLI. "Interworking between LoRaWAN and non-LoRa IoT systems" IEEE PerCom-IoT-PROD 2022.

Bibliography

[1] G. Perrone, M. Vecchio, R. Pecori and R. Giaffreda. The Day After Mirai: A Survey on MQTT Security Solutions After the Largest Cyber-attack Carried Out through an Army of IoT Devices. In *Proc. 2nd Int. Conf. on Internet of Things, Big Data and Security - Vol. 1: IoTBDS,*, pages 246–253. INSTICC, SciTePress, 2017.

[2] M. Calabretta, R. Pecori, and L. Veltri. A Token-based Protocol for Securing MQTT Communications. In *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6, 2018.

[3] Marco Centenaro, Lorenzo Vangelista, Andrea Zanella, and Michele Zorzi. Long-range communications in unlicensed bands: the rising stars in the iot and smart city scenarios. *IEEE Wireless Communications*, 23(5):60–67, 2016.

[4] LoRaWAN Specification. Standard, LoRa Alliance.

[5] M.A. Erturk, M.A. AydÄśn, M.T. Buyukakkaslar, and H. Evirgen. A survey on lorawan architecture, protocol and technologies. *Future Internet*, 11:216.

[6] A. Banks et al. MQTT Version 5.0. Standard, OASIS, March 2019.

[7] Lukas Simon Laufenberg. Impersonating lorawan gateways using semtech packet forwarder. *CoRR*, abs/1904.10728, 2019.

[8] Arash Tayebi, Setevan Berber, and Akshya Swain. Wireless sensor network attacks: An overview and critical analysis. In *2013 Seventh International Conference on Sensing Technology (ICST)*, pages 97–102, 2013.

[9] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N. Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184, 2017.

[10] James A. Jerkins. Motivating a market or regulatory solution to iot insecurity with the mirai botnet code. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–5, 2017.

[11] Jyoti Deogirikar and Amarsinh Vidhate. Security attacks in iot: A survey. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 32–37, 2017.

[12] M Anirudh, S Arul Thileeban, and Daniel Jeswin Nallathambi. Use of honeypots for mitigating dos attacks targeted on iot networks. In *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, pages 1–4, 2017.

[13] Kuan Zhang, Xiaohui Liang, Rongxing Lu, Kan Yang, and Xuemin Sherman Shen. Exploiting mobile social behaviors for sybil detection. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 271–279, 2015.

[14] Mujahid Mohsin, Muhammad Usama Sardar, Osman Hasan, and Zahid Anwar. Iotriskanalyzer: A probabilistic model checking based framework for formal risk analytics of the internet of things. *IEEE Access*, 5:5494–5505, 2017.

[15] A. Mosenia and K. Jha. A comprehensive study of security of internet-of-things. *IEEE Access*, 2017.

[16] CISCO. The internet of things reference model. Technical report, CISCO, 2014.

[17] Shahid Raza, Linus Wallgren, and Thiemo Voigt. Svelte: Real-time intrusion detection in the internet of things. *Ad Hoc Networks*, 11(8):2661–2674, 2013.

[18] Wei Zhou, Yan Jia, Anni Peng, Yuqing Zhang, and Peng Liu. The effect of iot new features on security and privacy: New threats, existing solutions, and challenges yet to be solved. *IEEE Internet of Things Journal*, 6(2):1606–1616, 2019.

[19] Yuchen Yang, Longfei Wu, Guisheng Yin, Lijie Li, and Hongbin Zhao. A survey on security and privacy issues in internet-of-things. *IEEE Internet of Things Journal*, 4(5):1250–1258, 2017.

[20] N.; Gidlund M Butun, I.; Pereira. Security risk analysis of lorawan and future directions.

[21] Jun Lin, Zhiqi Shen, and Chunyan Miao. Using blockchain technology to build trust in sharing lorawan iot. New York, NY, USA, 2017. Association for Computing Machinery.

[22] Xueying Yang, Evgenios Karampatzakis, Christian Doerr, and Fernando Kuipers. Security vulnerabilities in lorawan. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 129–140, 2018.

[23] Mohamed Faisal Elrawy, Ali Ismail Awad, and Hesham Hamed. Intrusion detection systems for iot-based smart environments: a survey. *Journal of Cloud Computing*, 2018.

[24] Nataliia Neshenko, Elias Bou-Harb, Jorge Crichigno, Georges Kaddoum, and Nasir Ghani. Demystifying iot security: An exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. *IEEE Communications Surveys Tutorials*, 21(3):2702–2733, 2019.

[25] Nadia Chaabouni, Mohamed Mosbah, Akka Zemmari, Cyrille Sauvignac, and Parvez Faruki. Network intrusion detection for iot security based on learning techniques. *IEEE Communications Surveys Tutorials*, 21(3):2671–2701, 2019.

[26] L. Veltri et al. Nemo: A flexible and highly scalable network emulator. *SoftwareX*, 10:100248, 2019.

[27] R. Pecori and L. Veltri. A statistical blind technique for recognition of Internet traffic with dependence enforcement. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 328–333, Aug 2014.

[28] P. Ducange, G. Mannará, F. Marcelloni, R. Pecori, and M. Vecchio. A novel approach for Internet traffic classification based on multi-objective evolutionary fuzzy classifiers. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, July 2017.

[29] José Camacho et al. A generalizable dynamic flow pairing method for traffic classification. *Computer Networks*, 57(14):2718 – 2732, 2013.

[30] Yu Wang, Yang Xiang, Jun Zhang, Wanlei Zhou, and Bailin Xie. Internet traffic clustering with side information. *Journal of Computer and System Sciences*, 80(5):1021 – 1036, 2014.

[31] Li et al. Deng. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[32] M. W. Gardner and S.R. Dorling. Artificial neural networks (the multilayer perceptron) A review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14):2627–2636, 1998.

[33] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki. Network Intrusion Detection for IoT Security Based on Learning Techniques. *IEEE Communications Surveys Tutorials*, 21(3):2671–2701, 2019.

[34] Elrawy, M. F. and Awad, A. I. and Hamed, H. F.A. Intrusion Detection Systems for IoT-based Smart Environments: A Survey. *J. Cloud Comput.*, 7(1):123:1–123:20, December 2018.

[35] Lopez-Martin, et al. Conditional Variational Autoencoder for Prediction and Feature Recovery Applied to Intrusion Detection in IoT. *Sensors*, 17(9):1967, Aug 2017.

[36] V. L. L. Thing. IEEE 802.11 Network Anomaly Detection and Attack Classification: A Deep Learning Approach. In *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, March 2017.

[37] A. A. Diro and N. Chilamkurti. Distributed attack detection scheme using deep learning approach for internet of things. *Future Generation Computer Systems*, 82:761 – 768, 2018.

[38] N. Moustafa, B. Turnbull, and K. R. Choo. An Ensemble Intrusion Detection Technique Based on Proposed Statistical Flow Features for Protecting Network Traffic of Internet of Things. *IEEE Internet of Things Journal*, 6(3):4815–4830, June 2019.

[39] O. Al-Jarrah and A. Arafat. Network Intrusion Detection System using attack behavior classification. In *2014 5th Int. Conf. on Information and Communication Systems (ICICS)*, pages 1–6, April 2014.

[40] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman. Deep Learning Approach for Intelligent Intrusion Detection System. *IEEE Access*, 7:41525–41550, 2019.

[41] C. Ma, X. Du, and L. Cao. Analysis of Multi-Types of Flow Features Based on Hybrid Neural Network for Improving Network Anomaly Detection. *IEEE Access*, 7:148363–148380, 2019.

[42] Riccardo Pecori, Amin Tayebi, Armando Vannucci, and Luca Veltri. Iot attack detection with deep learning analysis. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[43] J. Zhang and X. Chen and Y. Xiang and W. Zhou and J. Wu. Robust Network Traffic Classification. *IEEE/ACM Transactions on Networking*, 23(4):1257–1270, Aug 2015.

[44] M. L. Bernardi et al. Keystroke analysis for user identification using deep neural networks. In *2019 Int. Joint Conf. on Neural Networks (IJCNN)*, pages 1–8, July 2019.

[45] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. 32nd Int. Conf.on Machine Learning - Vol. 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[46] A. H. Lashkari et al. Characterization of Tor Traffic using Time based Features. In *Proc. 3rd Int. Conf. on Information Systems Security and Privacy - Vol. 1: ICISSP,*, pages 253–262. INSTICC, SciTePress, 2017.

[47] Hyunjae Kang, Dong Hyun Ahn, Gyung Min Lee, Jeong Do Yoo, Kyung Ho Park, and Huy Kang Kim. Iot network intrusion dataset, 2019.

[48] Nour Moustafa Elena Sitnikova Koroniotis, Nickolaos and Benjamin Turnbull. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. 2019.

[49] Nour Moustafa Elena Sitnikova Koroniotis, Nickolaos and Jill Slay. Towards developing network forensic mechanism for botnet activities in the iot based on machine learning techniques. 2017.

[50] Nour Moustafa Koroniotis, Nickolaos and Elena Sitnikova. A new network forensic framework based on deep learning for internet of things networks: A particle deep framework. 2020.

[51] Nickolaos Koroniotis and Nour Moustafa. Enhancing network forensics with particle swarm and deep learning: The particle deep framework. 2020.

[52] Nour Moustafa Francesco Schiliro Praveen Gauravaram Koroniotis, Nickolaos and Helge Janicke. A holistic review of cybersecurity and reliability perspectives in smart airports. 2020.

[53] Nickolaos Koroniotis. Designing an effective network forensic framework for the investigation of botnets in the internet of things. 2020.

[54] F. Loi A. Radford C. Wijenayake A. Vishwanath A. Sivanathan, H. Habibi Gharakheili and V. Sivaraman. Classifying iot devices in smart environments using network traffic characteristics. 2018.

[55] A. Sivanathan et al. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759, Aug 2019.

[56] Agustin Parmisano Sebastian Garcia and Maria Jose Erquiaga. Iot-23: A labeled dataset with malicious and benign iot network traffic, 2020.

[57] Mannor, Shie and Peleg, Dori and Rubinstein, Reuven. The Cross Entropy Method for Classification. In *Proc. 22nd Int. Conf. on Machine Learning*, ICML '05, pages 561–568, New York, NY, USA, 2005. ACM.

[58] Sutskever, Ilya et al. On the Importance of Initialization and Momentum in Deep Learning. In *Proc. 30th Int. Conf. on Machine Learning - Vol. 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.

[59] Jetmir Haxhibeqiri, Eli De Poorter, Ingrid Moerman, and Jeroen Hoebeke. A survey of lorawan for iot: From technology to application. *Sensors*, 18(11), 2018.

[60] Jansen C. Liando, Amalinda Gamage, Agustinus W. Tengourtius, and Mo Li. Known and unknown facts of lora: Experiences from a large-scale measurement study. 15(2), February 2019.

[61] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. Interoperability in internet of things: Taxonomies and open challenges mobile networks and applications.

[62] Ahmad S Mehmood F and Kim D. Open accessarticle design and implementation of an interworking iot platform and marketplace in cloud of things.

[63] Diana Yacchirema, Andreu BelsaPellicer, Carlos Palau, and Manuel Esteve. Onem2m based-interworking architecture for heterogeneous devices interoperability in iot. In *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6, 2018.

[64] Diana Yacchirema and Carlos Palau. Interworking of onem2m-based iot systems and heterogeneous iot devices. In *2020 XLVI Latin American Computing Conference (CLEI)*, pages 262–267, 2020.

[65] Tayebi A. and Caselli S Veltri L, Zanichelli F. Interworking between lorawan and non-lora iot systems. 15, November 2022.

[66] Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. Digital twin: Enabling technologies, challenges and open research. *IEEE Access*, 8:108952–108971, 2020.

[67] Stefano Busanelli, Simone Cirani, Lorenzo Melegari, Marco Picone, Mirco Rosa, and Luca Veltri. A sidecar object for the optimized communication between edge and cloud in internet of things applications. *Future Internet*, 11(7), 2019.

[68] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[69] Klaus Hartke. Observing Resources in the Constrained Application Protocol (CoAP). RFC 7641, September 2015.

[70] Simone Cirani, Marco Picone, and Luca Veltri. mjcoap: An open-source lightweight java coap library for internet of things applications. *Interoperability and Open-Source Solutions for the Internet of Things. Lecture Notes in Computer Science - Springer*, 9001, 2015.

Ringraziamenti

I would first like to thank my supervisor, Professor Luca VELTRI, whose expertise was invaluable in formulating the research. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would like to acknowledge my colleagues in Department of Engineering and Architecture, laboratory and POSITIVE project additionally Riccardo PECORI for their wonderful collaboration. I would also like to thank Professor Francesco ZANICHELLI, for their valuable guidance. You provided me with the tools that I needed to choose the right direction.

I would particularly like to single out Professor Stefano CASELLI, Professor Monica MONACHELLI and Professor Marco LOCATELLI, I want to thank you for your patient support and for all of the opportunities I was given to further my research.

In addition, I would like to thank my wife for their wise counsel and sympathetic ear. You are always there for me.