UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Automatic Service Quality Assessment in IoT Platforms and Ecosystems

Diploma Thesis

Elefterios Chatziefraimidis

Supervisor: Aspassia Daskalopulu

Volos 2021

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Automatic Service Quality Assessment in IoT Platforms and Ecosystems

Diploma Thesis

Elefterios Chatziefraimidis

Supervisor: Aspassia Daskalopulu

Volos 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αυτόματη αξιολόγηση της ποιότητας των υπηρεσιών σε πλατφόρμες και οικοσυστήματα IoT

Διπλωματική Εργασία

Ελευθέριος Χατζηευφραιμίδης

Επιβλέπουσα: Ασπασία Δασκαλοπούλου

Βόλος 2021

Approved by the Examination Committee:

Supervisor **Aspassia Daskalopulu**
Assistant Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Michael Vassilakopoulos**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Georgios Stamouli**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Date of approval: 23-2-2020

Acknowledgements

I would like to express my sincere gratitude to Prof. Aspasia Daskalopoulou, and also to Prof. Georgios Stamoulis and Prof. Michael Vassilakopoulos for the continuous guidance and support. Besides the supervising committee, i would like to thank Dr. Charilaos Akasiadis and Dr. Constantine D. Spyropoulos for providing me with the opportunity to participate in the SYNAISTHISIS project, and assisting me at every step until the completion of my thesis. Finally, i would like to thank all my professors and every person that encourage and help me the past years at the University.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Elefterios Chatziefraimidis
23-2-2021

Abstract

Internet of Things (IoT) is a rapidly increasing technology. It is a concept that encompasses various objects and methods of communication to exchange information and allow cyberphysical systems to realize their operation. This can be achieved by networks of physical objects or things embedded with communication, sensing processing, and actuating capabilities.

Nowadays billions of devices are interconnected and integrated as web services which can be used by developers to build complex applications. The definition of the Internet of things has evolved due to the convergence of multiple technologies, real-time analysis, machine learning, commodity sensors, and embedded systems. But in an Internet of Things (IoT) enviroment, the existence of a huge number of heterogeneous devices, which are potentially resource-constrained has led to quality of service (QoS) concerns.

In this thesis, we examine and evaluate the current state of the art in the domain of automatic QoS/QoE and different approaches and implementations that are proposed by several researchers. Some of these studies provide a theoretical approach from a platform perspective, while others focus on establishing a mechanism that achieves the appropriate QoS/QoE level. Therefore, using the above implementations and theoretical proposals, we design and implement our mechanism that belongs to the network layer techniques and provides an effective QoS level using outlier detection techniques. Our system captures the traffic of each service that is running on the SYNAISTHISIS IoT platform, and after that features extraction, we train a model using various outlier detection algorithms. Next, using this model, we identify the outliers that will reduce the QoS level, and also decrease the performance of the IoT platform.

Περίληψη

Το διαδίκτυο των πραγμάτων (Internet of Things - IoT) αποτελεί μια αναπτυσσόμενη τεχνολογία. Η γενικότερη ιδέα περιλαμβάνει αντικείμενα και μεθοδολογίες επικοινωνίας για την ανταλλαγή πληροφορίας και την υποστήριξη της λειτουργίας κυβερνοφυσικών συστημάτων. Το παραπάνω μπορεί να επιτευχθεί με την χρήση δικτύων από φυσικά αντικείμενα με ενσωματωμένα συστήματα για επικοινωνία, λήψη και επεξεργασία σημάτων.

Σήμερα δισεκατομμύρια συσκευές είναι διασυνδεδεμένες και ενσωματωμένες ως διαδικτυακές υπηρεσίες και μπορούν να χρησιμοποιηθούν από μηχανικούς (developers) για την δημιουργία πιο σύνθετων εφαρμογών. Ο ορισμός του διαδικτύου των πραγμάτων (Internet of Things - IoT) έχει εξελιχθεί εξαιτίας της σύγκλισης πολλαπλών τεχνολογιών, της αναλύσης σε πραγματικό χρόνο, της μηχανικής μάθησης και των ενσωματωμένων συστημάτων και αισθητήρων. Βέβαια σε ένα περιβάλλον που βασίζεται σε διαδίκτυο πραγμάτων, η ύπαρξη πολλαπλών ετερογενών συσκευών με περιορισμένες δυνατότητες οδηγεί σε προβληματισμό σχετικά με την ποιότητα των υπηρεσιών (QoS).

Σε αυτή την διπλωματική, θα εξετάσουμε την τωρινή κατάσταση του τομέα της αυτόματης παροχής ποιότητας υπηρεσιών και εμπειρίας (QoS/QoE), ποικίλες προσεγγίσεις και υλοποιήσεις που προτάθηκαν από πολλούς ερευνητές. Κάποιες από αυτές τις έρευνες παρέχουν μια θεωρητική προσέγγιση όσον αφορά την οπτική της πλατφόρμας, ενώ άλλες εστιάζουν στην δημιουργία ενός μηχανισμού που επιτυγχάνει το απαραίτητο επίπεδο ποιότητας υπηρεσιών και εμπειρίας (QoS/QoE). Στην συνέχεια, βασισμένοι στα παραπάνω θα σχεδιάσουμε ένα μηχανισμό που ανήκει στην κατηγορία των δικτυακών τεχνικών που παρέχει ένα αποδοτικό επίπεδο ποιότητας υπηρεσιών (QoS) χρησιμοποιώντας τεχνικές για εντοπισμό ασυνήθιστων συμπεριφορών. Το σύστημά μας συλλέγει την ‘κίνηση‘ των υπηρεσιών που τρέχουν στην πλατφόρμα ΣΥΝΑΙΣΘΗΣΗ και αφού εξάγει κάποια χαρακτηριστικά, εκπαιδεύει ένα μοντέλο με διαφορετικούς αλγορίθμους. Τέλος, χρησιμοποιώντας το παραπάνω μοντέλο αναγνωρίζουμε τις ανομοιομορφίες που μειώνουν το επίπεδο ποιότητας υπηρεσιών και την

γενικότερη απόδοση της πλατφόρμας.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 The IoT ecosystem

1.1.1 Historical Flashback and Definition

The Internet of Things is an archetype based on a number of interconnected heterogeneous devices, which are able to autonomously exchange data and deliver advanced cyber-physical systems and respective services. The term “Internet of Things“ was first introduced by Kevin Ashton in 1999 as a global newtwork of objects connected to Radio Frequency Identification [9]. Since then, IoT has spread to other application areas. The primary purpose of the IoT ecosystem remained the same to reinforce computers with capabilities related to information gathering without human intervention. The basic goal of the IoT is to expand the every day routine with computing power that provide various services through sensing, computing, communication and controlling the enviroment. These capabilities are desired in many use cases due to their impact from making our lives easier and safer. IoT brought a new viewpoint on the type and category of data that should be collected, and also the gathering frequency and the sources from which we would find the nessessary data [10]. That unlocked new possibilities because we could get information that was not available before.

The IoT archetype is described by its diversity and complexity and combined with the rapid progress from research has led to the lack of a standardised definition of IoT. Therefore, various definitions have been proposed by the academia and standardisation organisms including:

- **Definition by ITU:[11]** The Internet of Things is a global infastructure for information

society, enabling advanced services by interconnecting things based on existing and evolving interoperable information and communication technologies.

- **Definition by IERC:[12]** A dynamic global network infastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual things have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly intergrated into the information network.

- **Definition by ISOC: [13]** The term Internet of Things generally refers to scenarios where network connectivity and computing capability extends to objects, sensors and everyday items not normally considered computers, allowing these devices to generate, exchange and consume data with minimal human intervention. There is, however no single, universal definition.

Even if the above definitions have various common points, one commonly accepted definition hasn't yet been determined. The IEEE [14] captured and analyzed the different aspects of the definition with the goal to clarify the concept and construct one accepted commonly definition. Although, they realised that the definition of IoT is too extensive and they decided to provide two separate definitions, one for small and one for large enviroments:

- **Definition for small enviroments:** An IoT is a network that connects uniquely identifiable "Things" to the Internet. The "Things" have sensing/actuation and potential programmability capabilities. Through the exploitation of unique identification and sensing, information about the 'Thing' can be collected and the state of the 'Thing' can be changed from anywhere, anytime, by anything.

- **Definition for large enviroments:** Internet of Things envisions a self-configuring, adaptive, complex network that interconnects "Things" to the Internet through the use of standard communication protocols. The interconnected things have physical or virtual representation in the digital world sensing/action capability, a programmability feature and are uniquely identifiable. The things offer services, with or without human intervention, through the exploitation of unique identification, data capture and communication, and actuation capability. The service is exploited through the use of intelligent interfaces and is made available anywhere, anytime and for anything taking security into consideration as well.

The essential difference between the above two definitions lies to complexity, considering the fact that small specific systems have fewer services than large global systems that support many services and capabilities. Thus, taking into account that even IEEE couldn't synthesize properly one accepted IoT definition we can observe that the IoT has a large range of use cases that we should describe and analyze. The IoT ecosystem is very diverse and complex and in order to create efficient applications we need to embrace and deeply understand each part of that system. As the IoT defintion describes, connecting things to the Internet and giving them the ability to sense/actuate is a feature demaned everywhere. Thus, trying to categorize the use cases into more general groups can reduce the fuzziness and offer a greater comprehension.

1.1.2 Use Cases

The IoT ecosystem is becoming more and more complex, and it is essential to understand its potentials to improve the current infrastructure. As the above definition says, connecting things to the Internet and giving them an ability to sense/actuate is a feature demanded everywhere. However, various cases exist, so it is crucial to categorize the application use cases in more general groups for better comprehension. Since IoT provides individual use cases, different categorisations have been proposed based on the experience and viewpoint:

- Perera et al. [15] defined three general categories based on application domains: industry, enviroment and society.
- libelium [16] listed 54 application use cases under 12 categories: smart cities, smart enviroment, smart security etc.
- Atzori et al. [17] grouped applications into 4 categories: transport and logistics, healthcare, smart enviroment and personal and social.
- IERC [18] defined 10 categories: smart food/water monitoring, smart health, smart living etc.

We can observe that some researchers propose a more general classification, while others offer a more detailed one based on their scope and the requirements they want to cover. Most of those proposals consider the target audience to be more significant than the categories. Thus, taking into acount that perspective, we can separate the users into the following groups:

- **individuals:** individuals persons looking to improve their level of lifestyle.
- **society:** a community of people looking to find solutions for common issues.
- **industry:** any economic sector looking to satisfy customer's need with their product.

Every user category is an individual group with its own needs but in general these groups share some common goals: (i) maximize health and safety, (ii) minimise the amount of work while maximising the convenience of its execution, (iii) minimise the costs. As soon as we will understand every expectation that IoT has to fulfill we will identify and categorize the proper use cases that will lead to the achievement of the above goals. The defined user groups have a various range of interest and we would see that the number of categories will diverge based on the level of abstraction. Thus, depending on our level of abstraction we can group the use cases in a way that will satisfy the need of the user groups. For example, if we aim to limit the number of domains so that it would be clear what applications every domain covers, we can alter the IERC [18] and define only eight categories: smart buildings, smart healthcare, smart enviroment, smart city, smart energy, smart transport and mobility, smart manufacturing and retail, and smart agriculture. These eight domains cover the expectations and requirements that the user groups have so we can use keep that aspect to categorize the use cases for the IoT ecosystem. Usually we can categorize the use cases in a way that some application domains are influenced by a user group while others have to fulfill the needs of two or even all three user groups. Therefore, the overall categorization of the application use cases is influenced more by other technical aspects. Thus, we will describe the challenges that the IoT is facing to gain a better perspective of how this aspects are influencing the IoT ecosystem.

1.1.3 Challenges

Simultaneously with the rise of the IoT systems, different challenges are arising in various sections, including security issues [19], communication [20], and networking[20], among others. Based on the study of [21], these challenges can be divided into four categories: hardware, software, connectivity, and security. Each of these groups defines its specific issues that influence the overall quality of IoT solutions. Thus, we focus on each category separately.

Hardware: The number of IoT devices is rising exponentially, and that comes with some uprising issues. Firstly, IoT devices must be low cost to support massive infrastructures, otherwise, the value won't justify the cost. Also, the overall size and battery life will vary based on specific needs.

Software: Software is the baseline of the IoT system. In many cases, the existing software doesn't always meet the IoT requirements and that leads to new developments that have specific goals. For example, the IoT not only connects things to the Internet, but it also allows a machine-to-machine (M2M) communication. This approach requires particular comprehension and an AI system to control the process without human interaction.

Networking: The original Internet protocols are not designed for all the new emerging new data traffic characteristics. New network technologies are expected to support the Iot to address the upcoming challenges. These new network designs should be able to scale efficiently to support the systems with billions of interconnected IoT devices and also to extend the coverage and be more reliable than the old ones.

Security: Security is emerging to be the IoT's greatest challenge. The IoT systems, being a cyber-physical system, has been escorted with individuals vulnerabilities that sometimes may cause life-threating consequences. Thus, the proposed solutions should offer attack resistance and confidentiality.

Now, we can assume that to face efficiently all these issues and to satisfy the spectrum of the requirements, it is essential to implement some QoE/QoS mechanisms. But, first, we should examine those mechanisms to understand their usage and limitations from the IoT perspective.

1.2 Quality of Service/Experience from an IoT Perspective

1.2.1 Quality of Service through the layers

The Internet of Things is related to the use of intelligently heterogeneous connected devices and systems to gather data by embedded sensors and actuators in machines and other physical objects. Over the following years, the IoT will spread rapidly and will reveal new services that will improve the lives of consumers, unlocking new opportunities. The latest forethoughts have predicted that there will be between 26 to 50 billion devices by 2020[22].

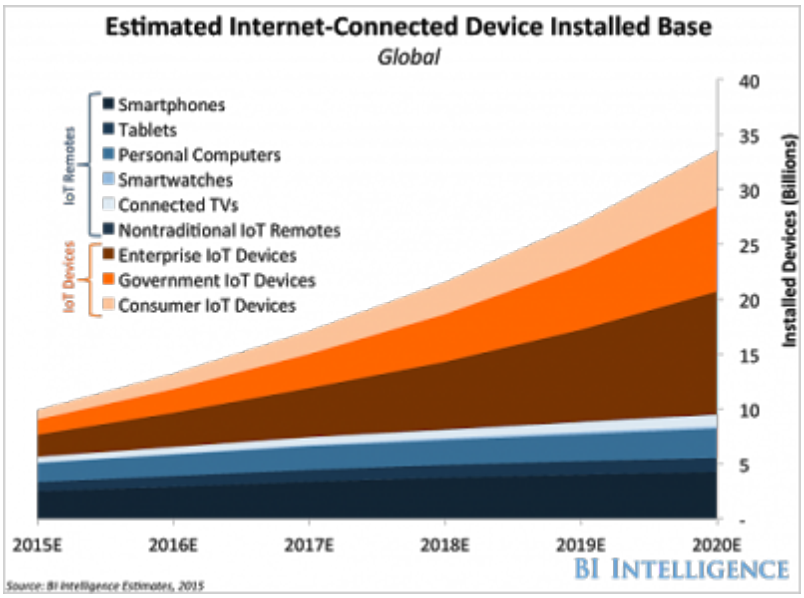

Figure 1.1: Estimated Internet-Connected Devices[1]

The huge amount of devices will introduce new services from a wide variety of sources such as home appliances, surveillance cameras, monitoring sensors, actuators, displays, vehicles, machines and so on. They will allow the development of applications in various domains, such as home automation, industrial automation, medical aids, traffic management, and many others[23]. As we have already mentioned these devices are gathering enormous amounts of data the so–called Big Data and often stored in Cloud data centers. While data processing speeds have increased rapidly, bandwidth to carry data to and from data centers has not increased equally fast. Thus, supporting the transfer of data from/to billions of IoT devices is becoming hard to accomplish due to the volume and the geodistribution of those devices. A potential way for solving such issues would be the implementation of QoS techniques that will supply an efficient and effective path to communications [24]. A few of the IoT-based applications that will be developed even if they will have a range of QoS requirements will not be provided by proper QoS. The implementation of a proper mechanism at each layer of the IoT system is required to afford a standard QoS service. It's also essential to define the quality of factors that will be used for the evaluation of the QoS and also identify the significant.

Quality of Service (QoS) is the key for the IoT ecosystem since assessing the actual quality of what service the device provides and what service users are paying has become a mission-critical practise requirement. Even if QoS entails a number of individual issues, since the ever increasing complexity of individual components, middleware, and inteconnec-

tion infastructures, reveal that QoS is and will be a vital facility in the IoT-Cloud computing scenario. To ensure an essential level of QoS for crucial applications there must be QoS approaches at every layer [22]. For example, when we have a delay at any layer of a physical sensor that will affect different parts from home automation to healthcare applications. Thus, we need to assure that these delays can be prevented at any level anytime using various approaches that will allow regulation and feedback between the different layers. In literature, there are different approaches and implementations for designing IoT architectures such as: (a) defining the architecture using applicable protocols and access networks[4], and (b) the baseline to implement IoT systems relies on individual middlewares [2]. A mechanism that provides QoS should be implemented in each layer of the IoT system using different software and hardware components.A service in the IoT ecosystem can be related to data handling for specific applications using various combinations of functionalities and interactions to satisfy the necessary requirements.

The Quality of Service is the capability to provide satisfactory service by different providers and systems. The above can be achieved by implementing QoS methods and techniques that will reform the quality of service parameters. These QoS parameters are defined from various perspectives based on the paper [25]: (a) User and Application, (b) Operator and Network, (c) Communication resources, (d) Technological resources. To maintain decent service quality for the users, it is essential to adopt specific quality parameters related to the combination of the above perspectives. An efficient IoT environment that satisfies admissibly the users involves additional services and infastructures and also it should rely on agreed SLAs(Service Layer Agreements). However, in many cases, achieving the SLAs to satisfy the user by reaching the maximum QoS may not be related to QoS parameters. Thus, in somer scenarios even after satisfying the SLAs the system may not even reach the minimum QoS. An implementation of QoS in an IoT system with specific agreements may involve various tradeoffs and regulations based on the requirements to achieve the quality of service that the user is expecting.

Based on the review and study of various architectures and schemes, these QoS parameters should be applied at each layer of the IoT architecture to satisfy the demanded requirements. Bhaddurgatte et al. [25] and many other researchers [26],[27] proposed a three layer architecture in which the QoS attributes are embedded as non-functional components in different layers. We will adopt for this thesis the above architecture schema, whose functional-

ities are described in more detail below:

Application Layer: The application layer is related to specific applications that provide various services to the user. This layer includes several applications that are deployed by the IoT ecosystem, for example, smart homes, smart cities, and health. The layer, also consists of modules that gather real-time data for analysis and computations. Bhaddurgatteet et al. [25] separates the parts of this layer into two categories:

- End Users: Users or machines that retrieve the data that are captured by the sensors.
- IoT modules: This part includes the functionalities like optimization of data transfer from/to devices and dynamic decisions based on different events.

Network Layer: The network layer consists of modules that determine the route of the data through the different layer of the IoT system. This layer will, also include various types of access networks and protocols that will be used to fulfill the specific purposes and requirements. Moreover,it will handle the communication devices that will be used in order to carry out the connectivity between the devices in the IoT ecosystem using individual routing functions and modules. In general, this layer communicates with the other individual layers and it is responsible for every action related to transferring and transmitting.

Perception-Sensing Layer: The perception-sensing layer handles the data gathering for the real world that is related to objects, machines, and people. Also, this layer includes modules and functions that regulate the sensors based on the values that they retrieve, and that sometimes influences the other layers.

Having defined all aspects of the QoS architecture at each layer, we can observe some of the QoS parameters that we mentioned above. These parameters are associated with a particular layer and affect the performance of the IoT ecosystem.

1.2.2 Quality of Experience

Since the IoT ecosystem is rising with an exponential rate, billions of devices will emerge that will use various technologies and have unequal capabilities in terms of processing, communication, and energy consumption. However, they will provide services in critical user-centric applications, and it is essential to adopt various QoE techniques. Being user-centric, the QoE provides a more conceptual understanding of the system's factors,while concerning the QoS attributes. Also, being closer and related to the user perspective, the QoE indicates

Application Layer	Network Layer	Perception Layer
Service time, Availability	Bandwidth	Time syncronization
Service delay	Delay	Location/mobility
Information Accuracy	Packet loss rate	Sensing coverage
Fault tolerance	Network resources utilization	Actuation coverage
Service perform cost	Real-time throughput	Reliability

Table 1.1: QoS attributes of three IoT layers

the real influence on human life and the crucial improvements. But, it is essential to consider also other quality indicators such as Quality of Data or Quality of Information that can be combined with the QoE to provide a more desirable outcome. In many papers, researchers have proposed various definitions related to the QoE approach such as:

- **Definition by ITU-T SG 12 in 2007 [28]:** "Degree of delight or annoyance of the user of an application or service as perceived subjectively includes the complete end-to-end system effects that are influenced by user state, content and context"
- **Definition by Dagstuhl seminar 2009 [29]:** "Describes the degree of delight of the user of a service, influenced by content, network, device, application, user expectations, and goals, and context of use."
- **Definition by [30]:** "QoE is the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state."

From the above definitions, only the last one is used nowadays and is considered a working definition. However, none of the above can provide a practical and exact definition of how to measure the QoE or how it impacts on the user's expectations. Since the QoE is a new perception in the IoT ecosystem, researchers are facing new challenges while analyzing the QoE Influencing Factors(IFs) that are used by the evaluation phase as metrics. Qualinet et al. [30] defined the IFs as follow: "Any characteristic of a user, system, service, application, or context whose actual state or setting may have influence on the Quality of Experience for

the user.“. These IFs can be classified into four individual categories based on the approach of [31]:

- **User-related IF:** any property and characteristic of a human user. The characteristic can describe the demographic and socio-economic background related to the physical and emotional state of the user.

- **System-related IFs:** properties and general characteristics that define the generated quality of service.They are related to media capture, transmission, storage, rendering and display,also the communication of information from content production to user.

- **Context-related IFs:** are factors that include any situation property related to user’s enviroment, in terms of physical, temporal, social, economic and technical characteristics. These properties are associated, for example, with the user’s location or the purpose of using a specific service.

- **Content-related IFs:** the information regarding the offered content by the service or application under study. In the case of video for example, they are associated with with video format, encoding rate, resolution, duration,type and content of the video.

Several works provided other extra external factors, e.g. the performance of the user’s hardware and mobility [32].Also, five standards of video quality metrics (join time, buffer ratio, rate of buffer events, average bit-rate, and rendering quality) were presented in [33]. Even if several papers provided some extra external factors, that influence the QoE in specific applications, they don’t satisfy the properties that most IoT applications have. Thus, only the described categories can be characterized as Influencing Factors. To have a better perspective of these factors, we can observe the interaction of the QoE with the IFs in Figure 2.

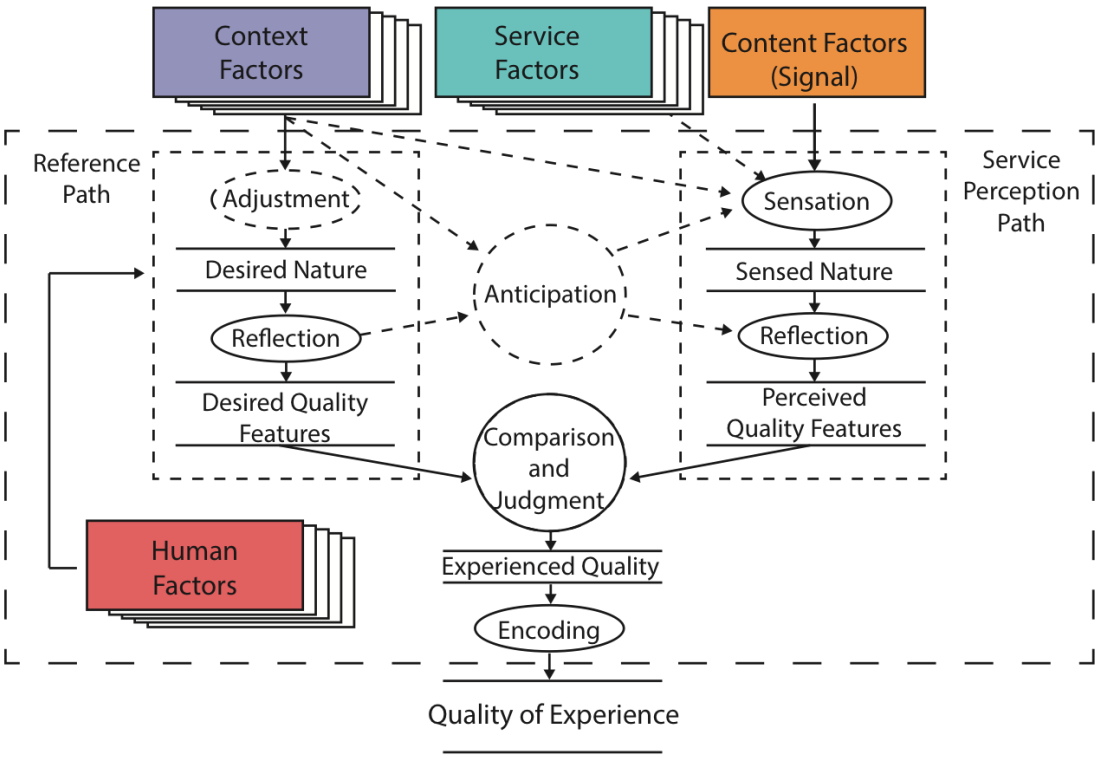

Figure 1.2: Influence Factors of QoE

Based on the above IFs, we can monitor these factors and optimize the QoE accordingly. Takanori Hayashi [34] proposed a process that consists of four steps and can achieve the optimized QoE performance that every IoT requires. These steps are: (a) QoE quantification, (2) QoE measurement and collection, (3) QoE analysis and visualization, and (4) QoE control. At the first step in QoE quantification, the purpose is to determine the relationship between QoE and some quality measures and to define the QoE traffic that will be observed during the process. Then, in QoE measurement and collection, you need to collect and measure correctly the information related to the QoE. Next, the current step analyzes and visualizes the collected information using the state of service provision. Lastly, using predefined conditions, the QoE control uses the analyzed and visualized data to optimize the performance of the QoE. However, to understand how each step works and what it achieves, we should describe in more detail the process that is followed by each action.

QoE quantification: The QoE quantification step defines service quality and management guidelines based on the human perceptive and features that respect the quality of communication services. At this step is also implemented the infrastructure for measuring the QoE using various quality metrics related to network, communication and user-centric data.

QoE measurement and collection: The QoE information that is used by different services is collected and measured to send it at the next level for QoE analysis and visualization. Since QoE is affected by some influence factors that we described above, information on user's service usage is needed to improve and optimize the performance of the QoE after the extraction and visualization of specific characteristics.

QoE analysis and visualization: Useful collected information for improving user satisfaction can be visualized at the current step by various visualization techniques, and also big data analysis methodologies can be applied for the extraction of the information related to these visualizations. This information involves statistics related to network state, signs of service fault, the scope of quality degradation on services, etc. By visualizing the QoE information, we can improve the quality of the network communications and reduce the cost of maintenance while increasing the user's satisfaction.

QoE control: The QoE control is triggered when the QoE rate is decreasing based on the visualizations of the previous step to improve the QoE. However, in an IoT ecosystem with real-time situations, QoE optimization addresses some difficulties and requires a powerful mechanism that is responsible for traffic control to support the service provision. Similarly, in our approach, the above system will be an effective feature of our mechanism for our future work.

1.3 Thesis Objectives

1.3.1 Research Background

In the twentieth century, society went through two significant periods of economic transformation: (a) industrialization and (b) computerization. These two processes led to massive productivity gains, economic growth and major improvements in living standards. Many countries rest their hopes on digital technology in terms of automation. The Internet of Things(IoT) has the means to push digitalization to a new level. Nowadays, the IoT ecosystem supports both clients and enterprises. For clients, the IoT delivers solutions that improve energy sufficiency, security, health, education, and many other aspects of daily life. For enterprises, it improves productivity, agriculture, and other regions. In general, IoT is related to the use of heterogeneous connected devices and systems that are capable of gathering and analyzing data for their purposes. Also, it offers a new point of what kind of data should be gathered, how

often and from which place, so it would be possible to get information that was not available before. In many cases, however, this benefit means the communication of low power devices over lossy networks, which creates high demands on the used technologies. Thus, to establish a more reliable user experience and to improve the characteristics of IoT applications, a certain service level is needed to motivate the users to use the IoT applications.

To ensure the appropriate level of service in several papers is proposed the usage of QoS and also QoE mechanisms that will manage the system requirements using the proper resources. This thesis aims to analyze the work related to QoS/QoE techniques at each layer of the IoT architecture and also describe how these mechanisms are implemented. However, we will consider the QoS part of a service level and we specify a QoS mechanism enabling the guarantee of a service level that is required. After describing our QoS mechanism, we implement and evaluate our proposed approach using the SYNAISTHISI IoT platform as a testbed.

1.3.2 Sections

The structure that is followed in this thesis is described below: Section I describes the general meaning of the IoT ecosystem and highlights the purpose of the QoS/QoE. Section II describes the related work and points out the main mechanisms that provide QoS/QoE. Section III introduces the SYNAISTHISI Platform that we will use it as a testbed for our mechanism. Furthermore, this section describes the conceptual approach of our QoS mechanism and also the outlier detection algorithms that we will use. Section IV includes the setup of our implementation at the IoT platform and the evaluation through various experiments. Finally, we summarize our work in Section V and discuss about the future work and the improvements to our system.

Chapter 2

Related Work

2.1 Quality of Service

2.1.1 Introduction

In the previous section, we introduced the approaches of QoS and provided motivation on how it affects the IoT ecosystem since every service in an IoT-based environment has a range of QoS requirements. To fulfill these requirements and to provide guaranteed services to critical applications it is necessary to build useful mechanisms at each layer of the IoT architecture. The layers of the above schema are assigned as follows [25]: (1) Application layer, (2) Network layer, (3) Device layer. A delay in any layer from the physical sensor to the user can cause problems in several crucial applications in different domains from automated driving vehicles to healthcare applications. This mapping reveals areas that the literature concentrates on, and points out areas that need more attention. There is a respective number of established approaches through the layers of the IoT that allows negotiation and feedback between the different layers. To this end, we now present the most important approaches from the literature, which are tailored for each of the layers that constitute the IoT stack.

2.1.2 Application Layer Techniques

Due to the advantages of performance and scalability, the number of distributed systems that use pub/sub middlewares has increased rapidly [35]. This implementation is used in different large-scale application domains, varying from shipboard computing to grid computing. The middleware supports policies that affect the QoS of the IoT system. The basic rules across

different approaches include persistence and durability. However, even if the policies provide control of the system, QoS evaluations must address a multitude of dimensions, especially in large-scale dynamic environments [36]. For example, a simple protocol may provide latency QoS only when a publisher sends a small number of subscriptions. Also, some middleware mechanisms used to provide QoS properties may not be applicable for different environment configurations. Challenges mostly arise when multiple QoS policies interact with each other. The solution to the above is the framework that is called Adaptive Middleware And Network Transports(ADAMANT). This mechanism was introduced by Joe Hoffert et al. [2] to address the challenges that pub/sub middleware had by integrating the following technologies: (1) QoS-enabled pub/sub middleware, (2) adaptive transport protocols, and (3) machine learning to manage specified QoS withing dynamic environments.

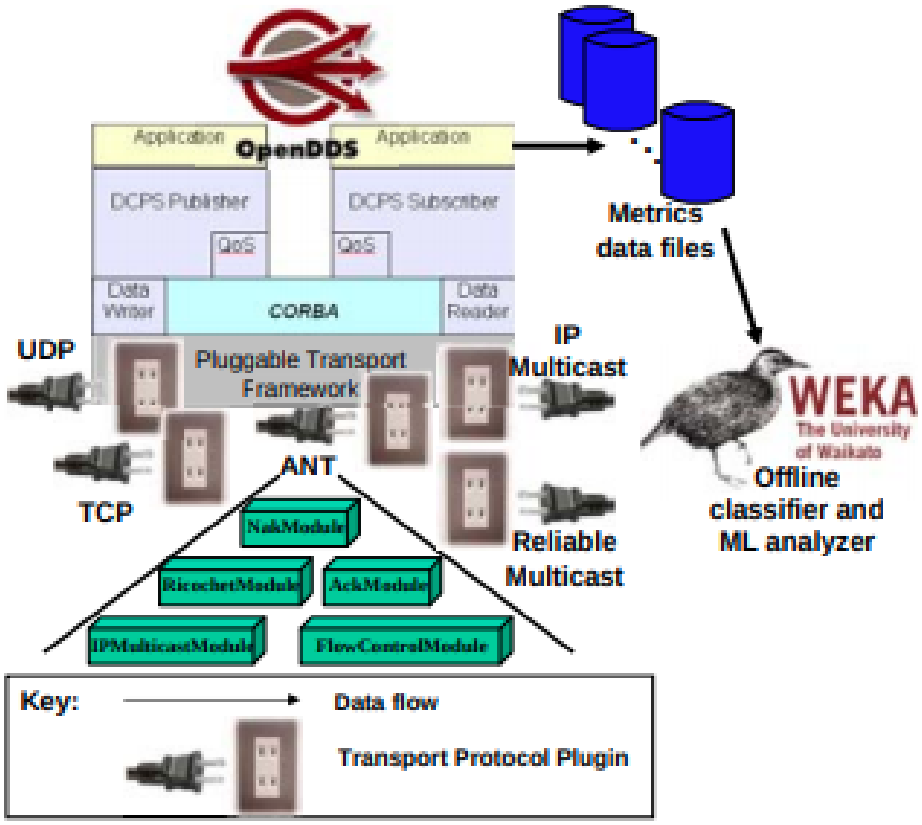

Figure 2.1: ADAMANT prototype [2]

The Adaptive Middleware And Network Transports is a pub/sub middleware that regulates the transport protocols and associated parameters to maintain specified QoS. It addresses

the emerging challenges using the following features:

- Standard QoS-enalbed pub/sub middleware addresses the scalability issues by separating data senders from data receivers.
- Adaptive network transport protocols work to address the challenges by providing the baseline system to maintain QoS even within dynamic circumstances.
- Machine learning models determine appropriate transport protocols and parameters given specified QoS and environments configurations.

ADAMANT has integrated the OpenDDS implementation with the Adaptive Network Transports framework, which supports various transport protocol types. As we can observe in the above figure ANT is also included and helps in building the properties provided by the scalable transport protocols. Finally, the framework provides the capability to input collected metrics and configuration information into weka and after analyzing the data determine techniques that provides the best results. A high level description of the ADAMANT framework is shown in Figure 3. Likewise, for our approach, we adopt from the ADAMANT framework the general pipeline, the packet capturing phase, and also the feature extraction step.

Nowadays, Quality of Service is the key for every platform's success, since every service has its own QoS requirements. However, QoS faces several issues since the IoT ecosystem consists of complex components, middlewares, and interconnected infrastructures. QoS monitoring is also an essential facility in cloud computing environments. Due to the dynamic conditions of the cloud infrastructure , continuous monitoring is necessary. One theoretical architecture that is proposed in the literature and provides the solution to the above cloud computing scenario is called QoS-MONaaS: Quality of Service MONitoring as a Service [3]. The application is portable so that it can be transported with minimal effort and configurations. J.Hoffert et al. [3] claim that QoS monitoring should be available to all cloud users in a seamless way and the "as a Service" model is the ideal solution. The implementation of QoS monitoring has some requirements and the platform quarantees that:

- All messages must be routed to the QoS service.
- Message authenticity remains, this means that the payload is unmodified and reliable.
- The identity of the user groups involved in the exchanges are kept anonymous. This is essential to avoid security issues between participants.

QoS-MONaaS implements a individual QoS facility which is made available to all applications running on a cloud platform. A detailed approach of the model involved in the monitoring process is presented below. For the framework to achieve the monitoring of the actual QoS delivered to the user, the Service Provider must already have settled an agreement with the cloud platform that describes the SLAs that will be applied. The agreement also contains information related to the Key Performance Indicators of interest with a description of the business aspect. These indicators evaluate the success of a service related to some levels of operational goals. Accordingly, choosing the right KPIs relies upon a good understanding of what is necessary to the platform in general.

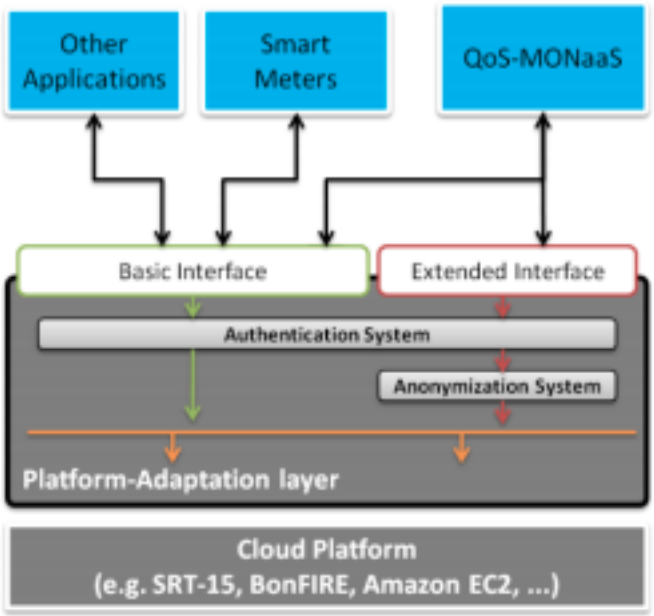

Figure 2.2: QoS-MONaaS interfaces [3]

As we can observe in the Figure 4, the framework introduces an anonymization system that prevents from revealing the real identity of monitored participants. It is essential for ensuring the “trust by design“ principle and the monitor to ignore the real identification to avoid cheating. QoS-MONaaS is already an application that runs on top of the SRT-15 cloud [3] platform in the form of a web service. The features provided by SRT-15 have made the implementation of QoS-MONaaS possible. From the above approach, we obtained for our implementation, the concept that our mechanism should be separate from the IoT platform, and also each service should have an individual instance of our model.

2.1.3 Network Layer Techniques

With the exponentially fast growth of the multimedia traffic in the IoT ecosystem, IoT traffic now requires a different approach regarding the Quality of Service. Under limited training data conditions though, existing classification methods are limited to the performance and are not efficient in classifying all the types of traffic[37]. To improve QoS-aware traffic classification, researchers have proposed various techniques in the literature. Changhe Yu et al. [4] divided these techniques into three categories. The first category is based on deep packet inspection. The DPI can identify the network packet of the data and classify that packet to use it later at the QoS classification step. However, a series of restrictions and the encryption of the packets restrict the DPI and to identify all the packet flows and the underlying protocols you need to reverse engineer all the previous pieces. The second category is related to machine learning. Machine Learning(ML) classification differentiates from the DPI in the part that the first extract the characteristics of the flow than checking the payload. That solves the previous problem with the encrypted content. The third category combines multiple classification techniques to achieve traffic classification. For this, Changhe Yu et al. [4] propose a flow classification framework using DPI and semi-supervised learning with multiple classifiers. Firstly, they use DPI to analyze the data flows and tag them with specific applications to form a partially labeled dataset. After that step, they train the classifier with this dataset and sort different applications into QoS categories so that the framework provides differentiated services for the individual types of applications. The classification scheme that they incorporated for their purpose lies below in Figure 5.

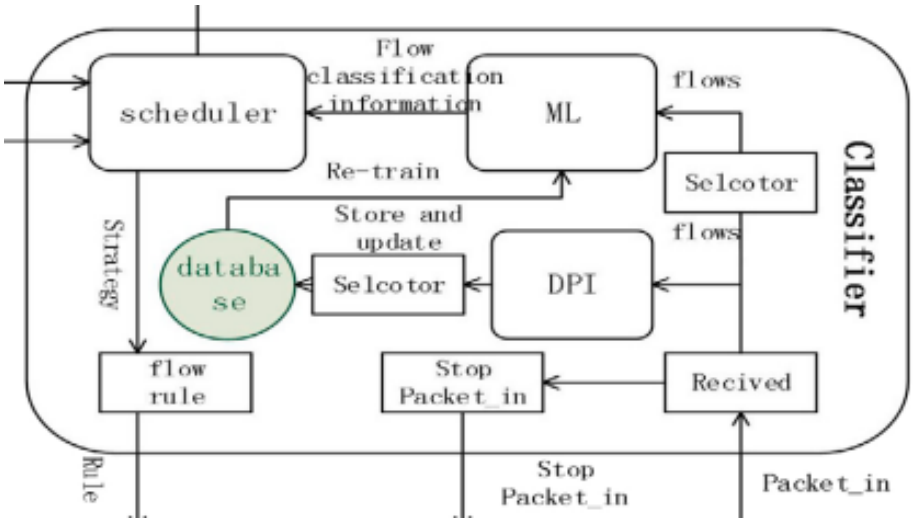

Figure 2.3: Traffic classification scheme [4]

Although there are a lot of unknown flows in the network, they observe that the applications with the same QoS classification have similar features. So they used a semi-supervised approach that X_L represents the labeled dataset and $X_L = \{(x_1, y_1), (x_2, y_2), ..., (x_l, y_l)\}$. The common semi-supervised mechanism is called tri-training. Tri-training is a semi-supervised mechanism that uses labeled datasets and three identical classifiers to enable iterative training. After that, the steps that they follow are described below:

- Generate three subsets L_a, L_b, L_c
- Use classifiers to make classification to unlabeled dataset. If a sample meets the below condition : $L_i = \{x|x \in U|C_a(x) = C_i(x)\}$. Then set L_i as the initial database to generate new training instances. If the error is less than last cycle we can continue to expand.
- Use the multi-classifiers for the classification of the unlabeled dataset.

To sum up, the proposed architecture combines DPI mechanisms and Multi-classifier semi-supervised learning mechanisms. DPI can analyze the packet flows and create the database, while the ML classifier can provide quick classification. Based on this approach, we designed the internal components of our model that have a similar pipeline, and some of the components are also used in our technique.

Another approach related to the network layer is the research of Zehui Liu et al. [5] , which is based on the fact that 80 % of the flows are smaller than 10KB and last under a few milliseconds while the 10 % of the traffic is related to the so-called elephant flows, which are extremely large (in total bytes) continuous flows measured over a network link. Such detection can be classified into 4 categories: (1) pull-based statistics, (2) host-based trigger, (3) sampling, (4) host-based detection. Motivated by the above, they proposed an adaptive approach for elephant flow detection using a dynamical traffic learning algorithm to configure the threshold value in real-time circumstances. The steps of the process that they follow:

- Obtain the traffic statistical counters of switches.
- Analyze the relationship between traffic and elephant flow, the system learns the changing traffic and decides the threshold with DTL algorithm.
- Given the estimated threshold, the elephant flow is classified in real-time.

The architecture that they proposed is illustrated in Figure 6. The top layer collects and analyzes network traffic. The collector captures the traffic and sends it to the analyzer. After that, the analyzer gathers the proper information and feeds the DTL algorithm. The second layer includes the training phase of a model , the smooth mechanism and also the weighted optimization related to the value of the threshold. The purpose of weighted optimization is to improve the training data accuracy. A smooth mechanism, which is composed of an algorithm described in the paper [5], decides whether to replace the previous threshold value or not.

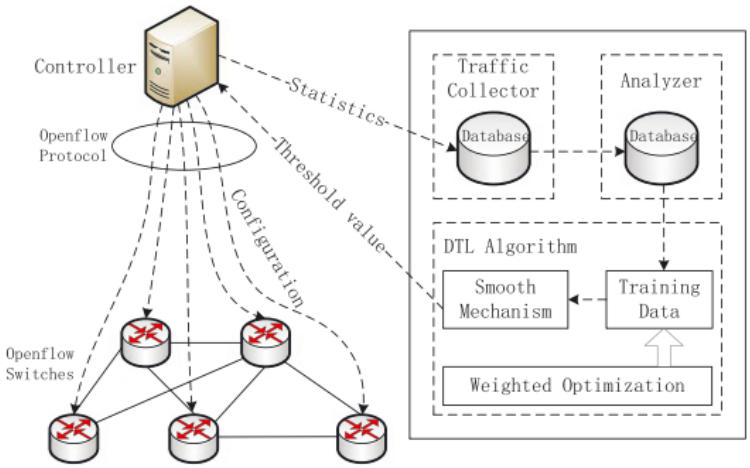

Figure 2.4: Elephant flow detection architecture [5]

Machine learning techniques automatically learn specific patterns and characteristics from the dataset that we provide. This approach is often more efficient from a manual configuration of an algorithm. Thus, they used a statistical methodology that is based on ML to build the mathematical model. The workflow that is followed, is described below:

- When the switch captures a new flow, it decides whether it belongs to elephant flow or mice flow according to the predefined threshold.
- The collector collects the incoming traffic characteristics.
- Analyzer receives the characteristics from the traffic collector and forwards them to the second layer for the training phase.
- Training data build the model and predicts the threshold using the weighted optimization technique.

- Smooth mechanism decides if the threshold should be changed.

To sum up, they proposed a adaptive framework to implement and evaluate a elephant flow detection,an approach that dynamically changes threshold value to meet the demands of the traffic characteristics in the network. Similarly, in our approach, we decided to create a traffic collector and analyzer and also to follow a related structure of the components. However, we didn't use the same algorithm for our training and evaluation of our model.

2.1.4 Network Layer Techniques - Outlier Detection

A different approach that is also network-based focuses on detecting various abnormal behaviors at the traffic and more specific at the packets that arrives at an IoT ecosystem. This technique is known as outlier or anomaly detection, and can also be adopted as a QoS mechanism. Some of the algorithms that can be categorized as outlier detection algorithms and are widely used are the following: (a) Isolation Forest(iForest), (b) Local Outlier Factor(LOF), (c) Minimum Covariant Determinant(MCD). Each of the mentioned methods has its advantages and disadvantages. In detail, iForest is only sensitive to global outliers and is weak in dealing with local outliers. Although LOF performs well in local outlier detection, it has high time complexity. Lastly, the Minimum Covariant Determinant method is a robust and agile estimator of multivariate location and scatter. Since estimating the covariance matrix is the cornerstone of several statistical methods, it can also serve as an outlier detection approach.

However, the above algorithms should be described in more details before we analyze the implementations and systems that are build based on these methods. Firstly, iForest is an unsupervised anomaly detection method and works on the principle of isolating anomalies instead of adopting the point's characteristics. Thus, the most common techniques construct a profile of what is "normal" so that anomalies are reported as those instances that don't match the previous profile. Although, the iForest uses a different approach and avoids building a model of normal circumstances and explicitly isolates anomalous points in the dataset. Furthermore, the LOF shares some concepts with DBSCAN and OPTICS and also belongs to the anomaly detection algorithms. It separates the normal from the outliers by measuring the local density of a given data point concerning its neighbours. The local density is estimated by the distance at which a point can be "reached" from its neighbors. By comparing the local density of an object to the local densities of its neighbors, it identifies regions with similar density that have lower density than their neighbors and are considered outliers. Using a more

statistical approach, moreover, the MCD, being resistant to outlying observations makes, the current method to be considered a well-known outlier detection approach. The basic ideas is to build a minimum covariance determinant model and then compute the Mahalanobis distance as the outlier degree of the data. The method can be applied on Gaussian-distributed data and could be still relevant on data from a unimodal,symmetric distribution.

In literature, there are different approaches and implementations, that are adopting the above methods to build efficient outlier detection systems that can accurately predict the outliers in complex datasets with low time complexity. Zhangyu Cheng et al. [6] proposed a two-layer ensemble method that combined the iForest and Local Outlier Factor that overcome their weaknesses. The algorithm applies the iForest to swiftly scan the dataset, drop the normal data, and generate a dataset with only the outliers. After that, LOF is implemented to separate the outlier candidate set and get more accurate outliers. The proposed model takes advantage of two algorithms and concentrates valuable computing resources. We can see the workflow of the proposed method at the Figure 7.

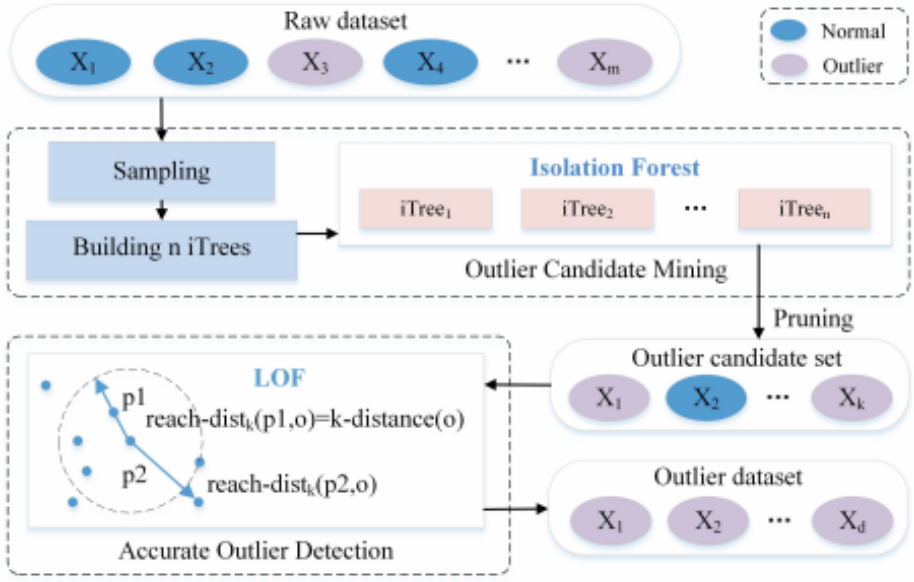

Figure 2.5: Workflow of the proposed ensemble model [6]

Thus, using the below model they conducted comparative experiments on six synthetic datasets and six real-world datasets, evaluated the outlier detection algorithm, and confirmed the accuracy effectiveness of the proposed model. A different implementation, using the DetMCD, was also proposed by Bart De Ketelaere et al. [7] that focuses on the Minimum Covariance Determinatnt approach Rousseeuw (1984) which provides highly robust estimators

for multivariate location and covariance matrices but with the difference that the DetMCD is deterministic,does not use random subsets, and is significantly faster for the huge sample sizes. The only disadvantage that the proposed method has is the loss of affine equivariance. The algorithm that the researchers followed is described below:

- Each variable of the dataset X is standardized.
- Six initial estimates $S_k(Z)$,$k = 1, ..., 6$ of the scatter of Z are contructed.
- As the eigenvalues might be inacurate, they denoted the covariance of matrix by $\sum_k(Z)$ and its location by $\mu_k(Z)$.
- Each $(\mu_k(Z), \sum_k(Z))$ is used to start C-steps until the convergence.
- The raw DetMCD covariance estimate $\sum_{raw}$ is chosen as the $C_k(Z)$ with the lowest determinant.
- A reweighting step is applied to improve the accuracy.
- The robust distances $RD_i = d(z_i, \mu_{rew}, \sum_{rew})$ classifies the observations into Inliers and Outliers.

The steps of the DetMCD are summarized at the following flowchart.

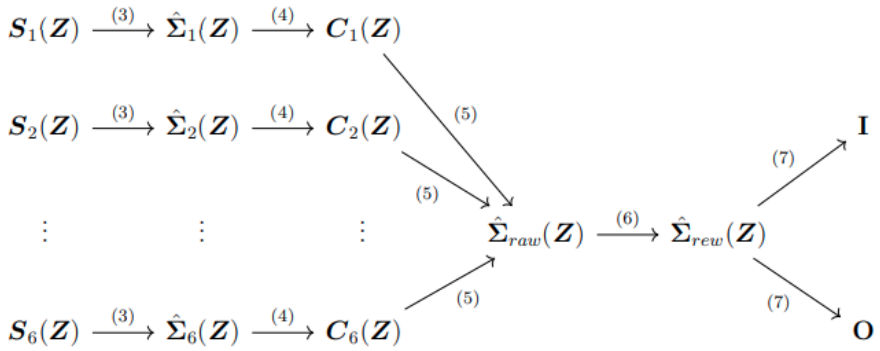

Figure 2.6: Flowchart of the DetMCD [7]

To sum up, we observe that many outlier detection algorithms have been proposed by researchers in the literature that can separate the normal for abnormal behaviors. Thus, we can use these techniques also to categorize the traffic from an IoT system and predict the possible abnormal circumstances in order to provide an efficient QoS mechanism. We can observe the mentioned techniques and also the implementation of them in Section III.

2.2 Quality of Experience

2.2.1 QoE Controller

A future feature of the upcoming applications is to provide the available resources to satisfy the user's requirements. Such conditions can be described by properly defined Quality of Experience. Based on literature the QoE is defined as *the overall acceptability of an application or service, as perceived by the end-user*. A large amount of research is focusing on the field of QoE evaluation and also to the relation between QoE and QoS parameters. Francesco Delli Priscoli et al. [8] adopted a approach related to QoE control. Once the QoE evaluator has estimated the personalized perceived QoE level, a QoE controller should take proper control decisions to reduce the difference between personalized Target and Perceived QoE levels. Moreover, at discrete time k with a time period T the QoE controller makes decisions. Each application is handled by an Agent i and the QoE error is defined as:

$$e_i(t_k) = PQoE_i(t_k) - TQoE_i$$

The purpose of the QoE Controller is to ensure that a nonnegative QoE Error for all Agents i exists to avoid the presence of underperforming applications. For a better perception of the model, we can observe the sketch of the QoE architecture in Figure 9.

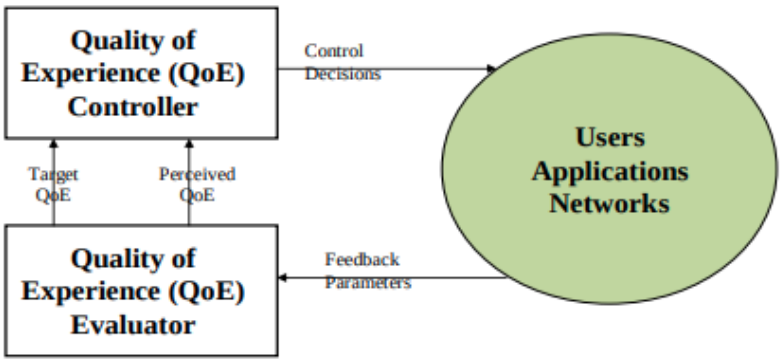

Figure 2.7: QoE Architecture [8]

In more detail, when a new application is running, the Agent i begins to evaluate the Target QoE and feed the controller with the computed values. As a result, the controller, based on the received values up to time t_k has to choose the most appropriate action $a_i(t_k)$ which the Agent i should enforce at time t_k. The same behavior is followed by the rest of the agents.

Another methodology using as baseline the above model is to associate the Perceived QoE with the application type of each application instance [38]. Let M denote the total number of application types, let $m \in \{1, ..., M\}$ denote a generic application type and also let $i(m)$ denote an Agent. Then the Perceived QoE for Agent $i(m)$ is computed as:

$$PQoE_{i(m)}(t_k) = g_m(\phi_m(t_k)),$$

where $\phi_m(t_k)$ represents a set of Feedback Parameters for the m-th application type and g_m is a function related to the application type and the Feedback Parameters.

Using the initial model approach, the QoE controller may produce outputs that contain Reference Values and also Security Reference Values [39]. Based on these values, the controller has to select the most appropriate that drive the Perceived QoE close to the Target QoE. However, this procedure is time-consuming so in order to solve it, they proposed the use of the Classes of Services(CoS), as described in [40]. In their paper, they assumed that each CoS is associated with a predefined set of QoS Reference Values. The above approach can be applied even in the case when each CoS is not related to QoS.

2.2.2 MARL-Q Algorithm

As we mentioned above, [8] solves the problem that we described previously using model-based control techniques. The QoE controller should evaluate the interaction between its decisions and the Perceived QoE. Thus, a mindset based on real-time learning by trial and error is followed. The strategy we stated uses the model-free MARL algorithm at each time step. The MARL algorithm uses the observations of a joint reward $r(t_{k+1}, a_1(t_k), a_2(t_k), ..., a_N(t_k)) = r(t_{k+1}, a_1, a_2, ..., a_N)$ which is received by each Agent at time t_{k+1} as a result of the policy joint $\pi(a_1, a_2, ..., a_N)$ at time t_k. The MARL algorithm is focusing to maximize the $R(\pi)$:

$$R(\pi) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k \cdot r(t_{k+1}, a_1, a_2, ..., a_N)\}$$

where $\gamma \in [0, 1)$ is a discount rate. Also, it is necessary to define the state space, the action spaces, and the reward function to make the algorithm work properly. Francesco Delli Priscoli et al. [8] proposed a Multi-Agent Q-Learning algorithm that provided a suitable solution and achieved the goal. The MARL-Q algorithm depends on action-value function $Q(s, a_1, a_2, .., a_N)$, as the return of the system when it starts from state s, takes the joint action $(a_1, a_2, ..., a_N)$ and follows the policy π. Then, the policy update step contains a random

action with probability equal to ϵ and with probability equal to 1-ϵ a joint action :

$$(a_1^{'}, a_2^{'}, ..., a_N^{'}) = arg\max Q(t_k, a_1, a_2, ..., a_N)$$

The parameter $\epsilon \in (0, 1)$ is exploration rate.

To summmarize, at each time step t_k the QoE controller follows the below process until it converges under an initial policy:

- Updates the function Q.
- Choose the joint action with the above probabilistic behavior.
- Update all the Agents so that each Agent i should enforce the a_i.
- Calculate the joint reward according to the selected reward function.

The algorithm, converges to the satisfaction of the initial policy. The convergence speed depends on the learning rate, the exploration rate, and the discount rate. However, even if the current algorithm is a useful approach that satisfies some purposes, it is followed by some flaws:

- The first challenge that the MARL algorithm has to face is the curse of dimensionality. Moreover, the complexity of the estimation for each possible state is exponential. Thus, it is impractical the implementation of the dynamic CoS using this specific version of the algorithm.
- Each Agent faces the moving-target learning problem since any Agent's action depends on the actions taken by the other Agents, and that constraints the progress.

To handle the above difficulties, the researchers proposed another form of the above algorithm. The H-MARL-Q is an alternative algorithm that reduces the join action space, and that leads to the acceleration of the dynamic CoS assignment without redundant information.

2.2.3 H-MARL-Q Algorithm

The H-MARL-Q algorithm selects only a subset of the joint action space providing the ideal solution to the dynamic CoS assignment. At each timestep, the joint action space includes several actions that have few possibilities of being the best ones. So the basic steps that the H-MARL-Q algorithm follows to reduce that joint actions are described below:

- Identification of the Reduced Joint Action Space is performed when a new Agent is born.

- Identification of the Suboptimal Joint Action is performed at each time step t_k to identify the joint action that will be selected.

Whenever a new Agent is created at time t_k, it updates the QoE controller by providing its QoE requirements in terms of Target QoE. Then the QoE controller emulates the behavior of the system in $N-1$ two-player test games between: (i) the new Agent and (ii) each already active Agents. In each two-player test game, the optimal policy $\pi^*(a_i, a_j)$ is calculated by using the MARL-Q algorithm described previously. This optimal policy calculates the pair of actions (a_i^*, a_j^*) where a_i^* and a_j^* that the two Agents should enforce. Every time step at which N Agents are active the controller stores $\frac{N(N-1)}{2}$ actions pairs. The Reduced Joint Action Space contains these pairs and also a subset of the entire joint action space A. Moreover, the Reduced Joint Action Space contains N Subspaces, where the ith Subspace is related to the two-player game of the ith Agent. Thus, the Reduced Join Action Space includes only SN joint actions related to the S^N that appear at the entire join action space A [8].

The second part of the H-MARL-Q algorithm includes the use of the MARL-Q algorithm with the Reduced Joint Action Space instead of the entire joint space A. In the current step, the QoE controller is using the process that we described in the previous algorithm to perform the required tasks but with a different reduced action space. Thus, this improvement reduces the required computational cost since the range of the pairs is narrowed down by the first step of the H-MARL-Q algorithm and at the same time it fulfills the purpose of the paper to create a Multi-agent QoE system that we can use it.

2.2.4 Combination of QoE with Resource Estimation

Several services related to multimedia are becoming more desired and have a variety of quality of service requirements. Thus, it is essential to focus on QoE-based systems that play a crucial role related to the fulfillment of those QoS requirements. Mohammad Aazam et al. [41] proposed a QoE-based dynamic resource estimation system that it depends on the QoE Ratio (QoER). They mentioned in the paper that there is a different approach to acquire the QoE through the net promoter score (NPS). Clients provide NPS-based QoE feedback on a scale of 0-10. When they respond supplying a score between 0-6 are known as detractors, 7-8

are passives, and 9-10 are promoters. The final NPS is estimated by substracting the detractors from the promoters. In the case that there is no feedback provided, the score is the mean of passive, which is 7.5. To define the NPS ratio, the researchers are using two parameters:

- The overall NPS(NPS_0) which is related to the feedback from all the users that already are using the service.
- The NPS(NPS_c) of a customer C that requests the service S.

The described implementation can play an essential part in the improvement of the QoS by observing the changing state of the service. Also, in their paper [41], they introduce a QoER model for resource estimation. The steps that the model follows are the below:

- Clients requests a service S.
- The service provider ensures that all QoS requirements are satisfied.
- The client responds with his/her feedback. The client's feedback provides the baseline of the user's expectations, so we should consider that when we estimate our resources.
- An NPS ratio is generated using the overall NPS and the NPS provided by the current client.

Thus, with that approach resources are allocated more efficiently according to the expectations and requirements of the clients. When the QoS requirements are not met, then the resources are increased to provide more effective services. The NPS ratio that we mentioned above is calculated as follow:

$$NPS_r = \begin{cases} \sum_{i=0}^{n} \frac{NPS_d}{\bar{x}NPS_{oi}}, \text{ if n=0} \\ \sum_{i=0}^{n} \frac{\bar{x}NPS_{oi}}{NPS_d}, \text{ if } \bar{x}NPS_{oi} > NPS_d \\ \sum_{i=0}^{n} \sum_{k=0}^{n} \frac{\bar{x}NPS_{oi}}{\bar{x}NPS_{ck}}, \text{ if } \bar{x}NPS_{oi} > \bar{x}NPS_{ck} \\ \sum_{i=0}^{n} \sum_{k=0}^{n} \frac{\bar{x}NPS_{ck}}{\bar{x}NPS_{oi}}, \text{ if } \bar{x}NPS_{ck} > \bar{x}NPS_{oi} \end{cases}$$

NPS_r is the NPS ratio defined by the overall NPS(NPS_o) of the service i that is requested by a client with a historical NPS(NPS_c). As we can observe, at the above definition, there are four cases represented. The first case represents the situation when a client is new, and the default NPS is applied. Although the service some times may have a higher overall

NPS(NPS_{oi}) than the default, so we got the second case of the model. When a historical NPS(NPS_c) already exists, then case 3 or 4 is applied depending on the number of the previous instances.

The ratio inceases as the difference between NPS_{sk} and NPS_{oi} increases to fulfill the required demands. The NPS ratio is calculated as follow:

$$NPS = NPS_{pr} - NPS_{dt}$$

where NPS_{pr} represents the promoters and NPS_{dt} represents the detractors. In order to evaluate the QoER model they proposed that we should use some metrics related to the NPS ratio:

- **Overall NPS:** Average NPS of all the clients that used a service
- **Default NPS:** The default NPS, which is the average of the passive(7.5)
- **Customer's Historical NPS:** The average NPS of a client that requested a service.

With the above metrics, they tried to evaluate the performance of their model and run some tests on an environment that they created. To sum up, they defined a mathematical model based on a QoE ratio that can be applied to any IoT ecosystem.

Chapter 3

Network System For Anomalies Detection - QoS

3.1 Introduction

Inspired by the related work, we will focus our research and implementations on building a network infrastructure that will capture, analyze and identify the abnormal network packets using some of the well-known outlier detection techniques. In detail, we will build a network outlier detection system that will capture some packets and will extract some features. After that, we will apply three of the most well-known outlier detection algorithms and categorize these packets as normal or abnormal. The algorithms that we decided to adopt are the following that was also described in the related work section: (a) Isolation Forest(iForest), (b) Local Outlier Factor(LOF), (c) Minimum Covariance Determinant. These methods have their advantages and their weakness that we also described at the related work where we quickly analyze them. Thus, our purpose is to use the SYNAISTHISI IoT Platform as a testbed and implement a system that will provide a network-based outlier detection mechanism that will be regarded as an efficient QoS layer for the IoT ecosystem. With the usage of the outlier detection techniques, each suspicious action will be caught and blocked by the system without interfering with the client's services. The system will provide stability and give a more reliable view of the incoming and outcoming traffic and guarantee a protected and fair use of the platform resources to the clients. However, to better understand the implementation

that we will try to build, we should first examine and describe further each part of the IoT platform and also how it works and what it provided to the clients.

3.2 The SYNAISTHISI Platform

The purpose of the SYNAISTHISI platform is to contribute efficient, secure applications and services to clients, which aim to minimize the cost, delays, and usage of resources. The platform consists of five layers each contributing functionalities at different aspects. These layers manage the physical objects, the communication with the platform through interconnected channels, and also control of the current resources. The current IoT platform provides interfaces that support five of the most commonly used Application Layer Protocols(ALPs): MQTT, AMQP, WebSockets, CoAP, and REST HTTP and it is based on open-source components that are interconnected. To use the SYNAISTHISI platform, we need to set up an instance of the platform as a dockerized container that contains some open-source components that implement those five ALPs. The modules and the internal components, which form the IoT platform, are displayed in Figure 10.

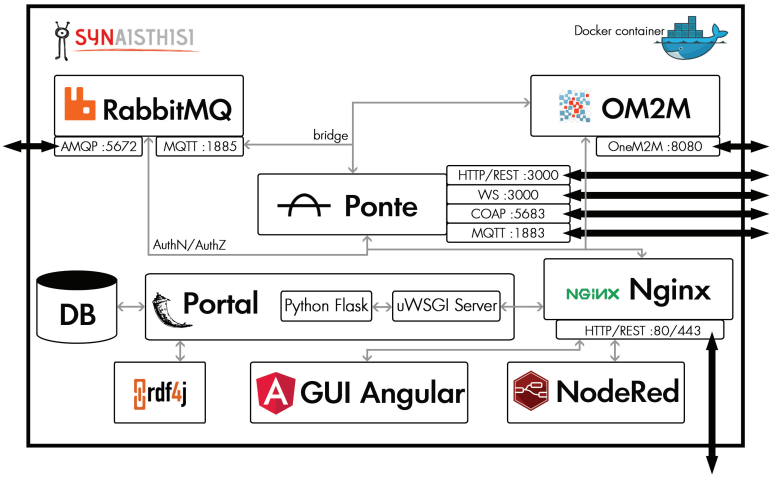

Figure 3.1: Platform components

We can observe that the platform is using docker to run sandboxed instances of processes, and also all the services run with the same approach to be executed on top of the existing operating system and to be isolated from each other. Furthermore, one of the core components is Ponte,

which is an open-source framework that bridges various ALPs such as MQTT, MQTT over WebSockets, REST/HTTP, and CoAP. Lastly, other components coexist inside the instance of the platform, and are used for parsing, querying, data transfer, and storing as we discern in the above figure. To better understand the infrastructure of the platform we should define the two main entities:

- **Topics** are the communication channels established by the message brokers that are responsible for message exchanging.
- **Services** are related to the algorithms/methods operating on the data that are transferred by the topics.

After the setup, we need to create an account as a client to access the functionalities and capabilities of the SYNAISTHISI. Then, after registering, we need to create a new service and describe your application flow using input and output topics based on given ontologies. A topic as we described previously from the IoT aspect is a path where an IoT device subscribes and publishing messages to provide input or to retrieve the output to/from specific services. At the figure below, we can see a service that generates random integers. It has two input topics, the frequency and the range that specifies how often should an integer be generated, and the range limit of the output.

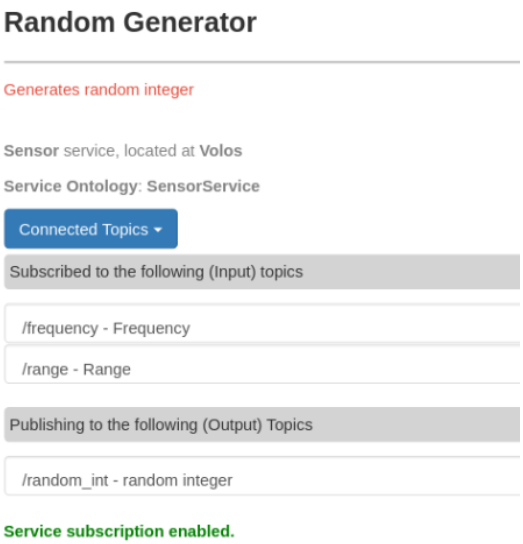

Figure 3.2: Random Integer Generator

Moreover, it has an output topic that is called random_int where we can retrieve the generated integer. Thus, when an IoT device wants to receive a random integer, it can connect to the

output topic and wait until an integer is generated. Additionally, the client can tweak the frequency and the range of the generated integers based on the requirements that he/she needs to fulfill by just accessing each topic. However, to use a service, we need first to generate one and then deploy it to the IoT platform. Thus, it is essential to write our algorithm using python, and after that, we need to dockerize your program because the IoT platform accepts only dockerized applications. Next, we need to upload our Dockerfile and all the other essential resources that are required. We can examine the upload/build step in Figure 12.

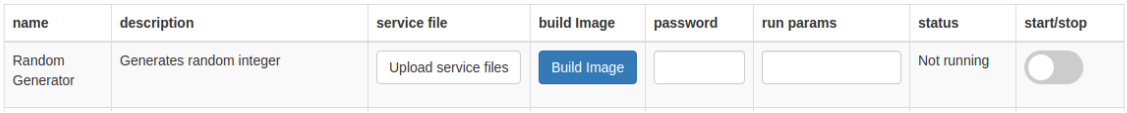

Figure 3.3: Upload Dockerfile for the build

This step includes the choice of the previously mentioned files and the image building after the insertion of your password. After the successful upload and image construction, you can begin the service to use your application through the IoT platform. When the service is up and running, then you can connect your IoT device and via the input topics run your algorithm and retrieve the estimated output using only the platform's resources.

Hence, now that we described the entire process of service construction and image building, we should examine the topic-based system we will implement. Briefly, we will build an infrastructure that will capture the network packets that transfer the messages from the topics. After that, we will extract and analyze the features that are provided by these packets using various machine learning techniques. In that way, we will achieve some QoS requirements to make the IoT platform perform in a more reliable and fair perspective.

3.3 Proposed System

Here, we describe how we build a system that uses network traffic to identify anomalies and abnormal behavior at the IoT platform known as SYNAISTHISI. Let's assume that we have a scenario where the IoT platform has deployed N services and each of these has unique id i , $i = 1, 2, ..., N$. Each of those running services has k_i input and m_i output topics. We can observe the described scenario at the Figure 13. As we can see, each service is running at its own container because all the services are using docker images and each one is independent of the other without sharing any information or messages. Thus, we should create a QoS

implementation that runs separately and creates an instance of itself for every running service to provide more reliable and unique protection for each service.

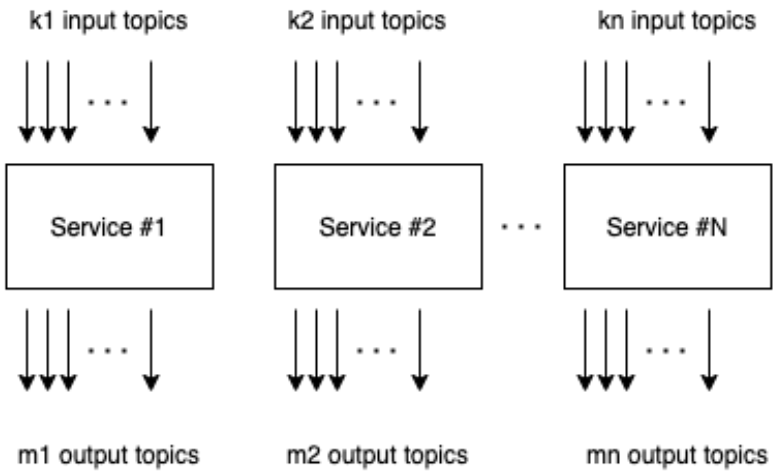

Figure 3.4: Services

Every instance of our implementation will behave as a parallel monitor subscribing to the respective topics at every service and won't interfere with the service's algorithm. The purpose of our mechanism is to capture the traffic of each service and identify the outlier using various machine learning techniques after the creation of a trained instance that fullfils our QoS requirements. Thus, to reach our purpose first, we need to accomplish some initial steps that will provide the trained model that is essential. The steps that our technique requires are the following:

- Packet capturing.
- Feature extraction.
- Feature filtering.
- Model Creation.
- Evaluation using real time circumstances.

Every step that we described has its individual role to play and provides to the implementation with the important requirements that will be used to keep our system running and fulfill its goal. Therefore, we should not avoid any of the above steps because if that happens, then we won't have the result that we expected to have. We now describe each step of our procedure

separately to gain a better understanding of our perspective. However, before that, we can examine the complete implementation visually of our infrastructure below in Figure 14.

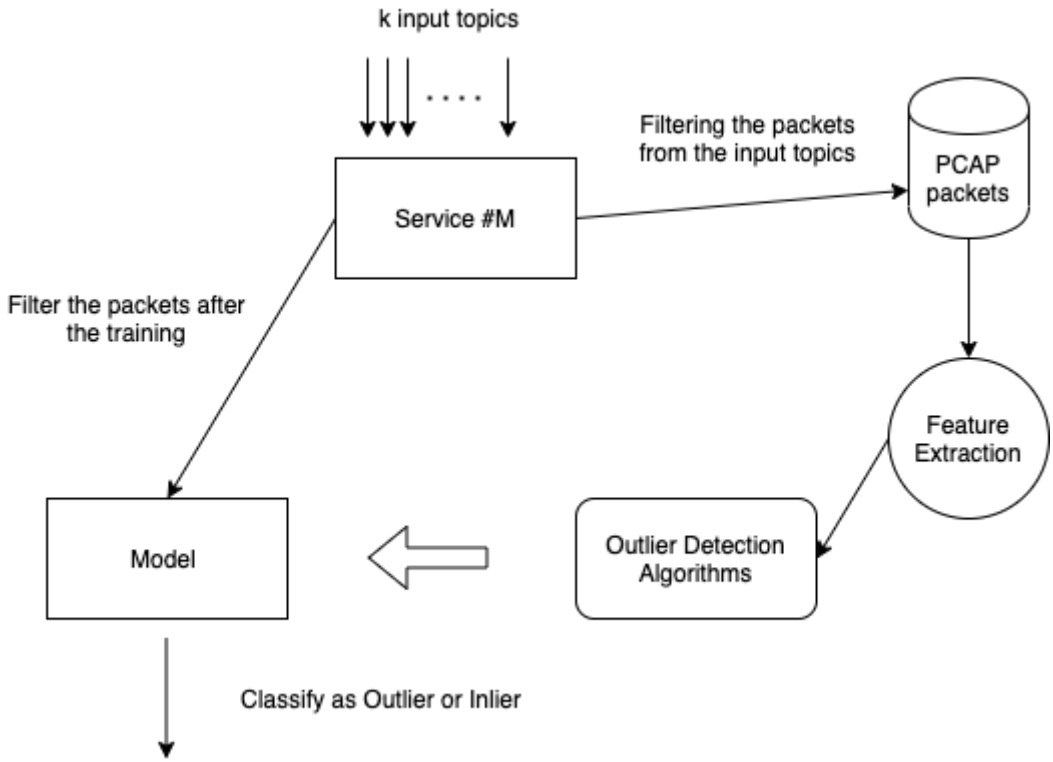

Figure 3.5: Outlier Detection System

The first step of our technique is related to the capturing of the network packets. We analyzed the structure of the IoT platform previously, and we declared that each mechanism is in a container. Thus, each instance of our sytem needs to monitor the traffic from the container that is related to message transporting through the topics, and that container is the one with the Ponte running inside. For the packet capturing we can use the tcpdump tool using the following command:

```
$ tcpdump -i <CONTAINER ID> src <CONTAINER IP>
    -w <FILE>.pcap
```

After obtaining the PCAP files from the Ponte container, we should hold the ones that are related to the topics that the current instance of our system is responsible for and drop all the others. Thus, each instance of our system that is responsible for each service will hold the PCAP file (or populate a corresponding database) that contains information about their service's topics. After the capture phase has ended, then each instance will start the preprocessing step. That step includes the extraction of specific features that are related to the PCAP

files and will help us build our machine learning model using the algorithms that we mentioned. The features that we choose are also examined by other researchers. Goodman et al. [42] describe a Word2Vec approach, used for text processing, and apply it to packet data for automatic feature extraction using some of the features that we also adopt. Furthermore, Sikos [43] focus on packet analysis for network forensics and also uses some of the features that we also choose. Thus, we should examine each of the features separately as shown below at the Table 2.

Features	Description
version	Internet Protocol Version (IPv4 or IPv6)
ihl	Internet Header Length
tos	Type of service
len	Packet Length
ip_flags	Various flags related to the packet
frag	Fragmentation flag
ttl	Time to Live
proto	Protocol
ip_options	Options related to the packet
time	The time that the packet arrived
sport	Source port
dport	Destination port
dataofs	Offset indicating the start of the header
reserved	For future use and should be set to zero.
tcp_udp_flags	Another option related to different protocols
window	The highest possible numeric value for a receive window(in bytes).
urgptr	Number of urgent packets.
tcp_udp_options	Options related to the packet
land	1 if connection is from/to the same host/port. 0 otherwise.
time_diff	Time difference between two packets.
payload	The encapsulated data

std_dev_payload	Standard deviation of the payload
Avg_syn_flag	The average of packets with syn flag active in a window of packets.
Avg_urg_flag	The average of packets with urg flag active in a window of packets.
Avg_fin_flag	The average of packets with fin flag active in a window of packets.
Avg_ack_flag	The average of packets with ack flag active in a window of packets.
Avg_psh_flag	The average of packets with psh flag active in a window of packets.
Avg_rst_flag	The average of packets with rst flag active in a window of packets.
Avg_DNS_pkt	The average of DNS packets in a window of packets.
Avg_TCP_pkt	The average of TCP packets in a window of packets.
Avg_UDP_pkt	The average of UDP packets in a window of packets.
Avg_ICMP_pkt	The average of ICMP packets in a window of packets.
Duration_window_flow	The time from the first packet to last packet in a window of packets.
Avg_delta_time	The average of delta times in a window of packets. Delta time is the time from a packet to the next packet.
Min_delta_time	The minimum delta time in a window of packets.
Max_delta_time	The maximum delta time in a window of packets.
StDev_delta_time	The Standard Deviation of the delta time in a window of packets.
Avg_pkts_length	The average of packet length in a window of packets
Min_pkts_length	The minimum of packet length in a window of packets
Max_pkts_length	The maximum of the packet length in a window of packets.
StDev_pkts_length	The standard deviation of the packet length in a window of packets.

Avg_small_payload_pkt	The average of packet with a small payload. A payload is considered small if his size is lower than 32 Byte.
Avg_payload	The average of payload size in a window of packets.
Min_payload	The minumum of payload size in a window of packets.
Max_payload	The maximum of payload size in a window of packets.
StDev_payload	The standard deviation of payload size in a window of packets.
Avg_DNS_over_TCP	The average of ration DNS/TCP in a window of packets.

Table 3.1: Features extracted from PCAP files

However, even if these features provide several indications related to different aspects of the traffic, we shouldn't include them all because our model will become complex and with additional inefficient redundant data. Thus, we reduce the amount of features using multiple techniques that will drop the columns with low standard deviation and high covariance ratio providing a more reliable and useful model that avoids overfitting and also holds only the required knowledge. To extract the features with low standard deviation, we calculate the standard deviation of the collected data and remove the instances with standard deviation equals to zero. For the high covariance, we compute the correlation matrix of the features from our dataset and then remove the features with a ratio that is over 85%. After using the above methods, our dataset now carries valuable features, that includes all the essential details for building our mechanism and are presented in Table 3.

Features	Description
time	The time that the packet arrived
time_diff	Time difference between two packets.
payload	The encapsulated data
std_dev_payload	Standard deviation of the payload
Avg_psh_flag	The average of packets with psh flag active in a window of packets.
Avg_TCP_pkt	The average of TCP packets in a window of packets.

Duration_window_flow	The time from the first packet to last packet in a window of packets.
Avg_delta_time	The average of delta times in a window of packets. Delta time is the time from a packet to the next packet.
Min_delta_time	The minimum delta time in a window of packets.
Max_delta_time	The maximum delta time in a window of packets.
StDev_delta_time	The Standard Deviation of the delta time in a window of packets.
Avg_pkts_length	The average of packet length in a window of packets
Avg_small_payload_pkt	The average of packet with a small payload. A payload is considered small if his size is lower than 32 Byte.
Avg_payload	The average of payload size in a window of packets.

Table 3.2: Final features that will be used

The next step that we follow is associated with building our model. Now that we have pre-processed our data, we can feed our model with the features that we choose. We depend on different python libraries that provide us with the implementation of the outlier detection algorithms that we mentioned previously(iForest,LOF,MCD). Thus, using these implementations, we train and build a model that will identify the outliers and abnormal behaviors at each service, and that means that this step is identical for every instance of our system. Lastly, using the model that we created in the previous step, we can evaluate the behavior of our implementation with the new incoming packets and categorize them as normal or anomalies, and after various experiments decide if our system perform well or not. These experiments depend on changing the frequency of the traffic, the payload of the packets, and various modifications to verify the capabilities and limits of our system. These trials are analyzed in detail in the next chapter.

Now that we have an overview of our system, we focus on the algorithms that we use for detecting the outliers and analyze how they achieve that. Hence, in the next section we examine separately each of the following algorithms: (1) Isolation Forest, (2) Local Outlier Factor, (3) Minimum Covariance Determinant, and also specify how each of them handles the features and decides the category of each network packet.

3.4 Outlier Detection Algorithms - Model

In the previous subsection, we described each part of our implementation, including the evaluation step. But, we didn't focus on explaining the algorithms that our model uses to identify the outlier from the usual traffic. Thus, now we analyze and examine each of the three mechanisms that our model adopts to have a better perspective of the way that they work. This will help us later to understand why each technique chooses the specific outliers based on the nature of each implementation.

In this study, we include an isolation algorithm that focuses on separating the outliers from the rest of the data points. However, this method differs from the other techniques as it isolates the anomalies instead of identifying the normal instances. The concept of Isolation forest was brought by Liu et al. [44] and uses random forests to compute an isolation score for each data point. The model is built by performing recursive random splits on attribute values, hence generating trees able to isolate any data point from the rest of the data. The score of a point is then the average path length from the root of the tree to the node containing the single point, a short path denoting a point easy to isolate due to attribute values significantly different from nominal values. The author states that his algorithm provides linear time complexity. We can see how the algorithm works by observing Figure 15 where the iForest is trying to isolate the x_i and x_0.

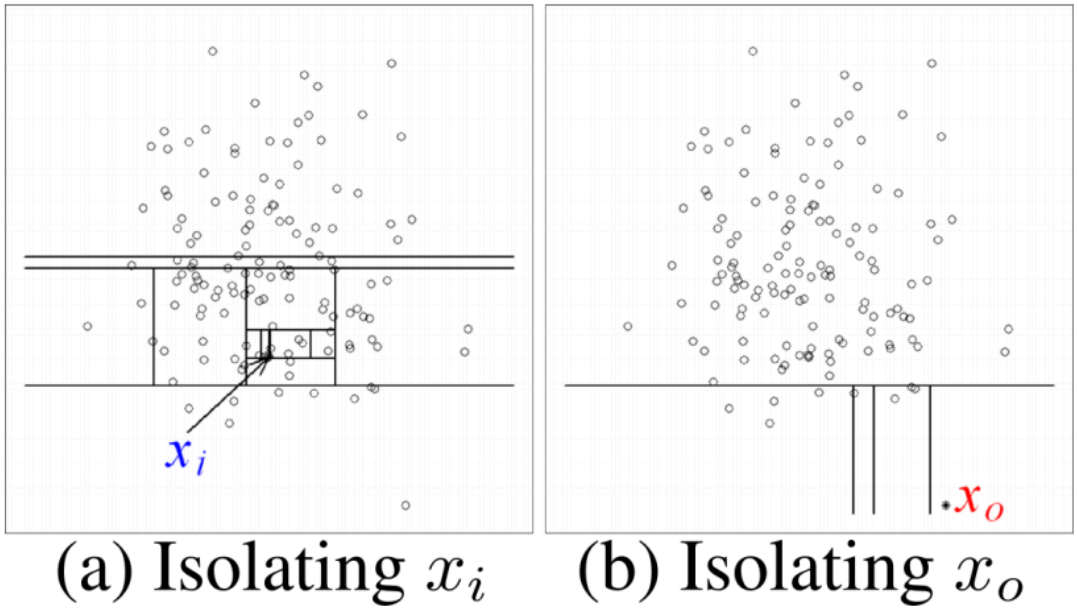

Figure 3.6: Outlier detection using iForest

However, we can recognize that x_0 is isolated using fewer iterations than the x_i and has a short path at the tree. The shorter path means that the point is different from the rest of the data points, and is classified by the algorithm as an outlier. Hence, using a similar approach to the one that we defined above is applied by our model. At the preprocessing step, an instance is created with various features, and that instance is isolated and then categorized as a normal or an abnormal data point. The whole procedure is based on the iForest, which decides the category for each data point. Thus, with that in mind, we have a better perception of our system and also the algorithm that helps us to accomplish our purpose to create an outlier detection mechanism that provides QoS to the IoT platform.

Another method that we are applying for detection the outliers also well-known and called Local Outlier Factor (LOF). This technique indicates the degree of outlier-ness for each object in the dataset that we build, and also it is a concept of an outlier that quantifies how outlying an object is. Furthermore, the local outlier factor algorithm is an unsupervised anomaly detection method that computes the local deviation of a given data point with respect to its neighbors.

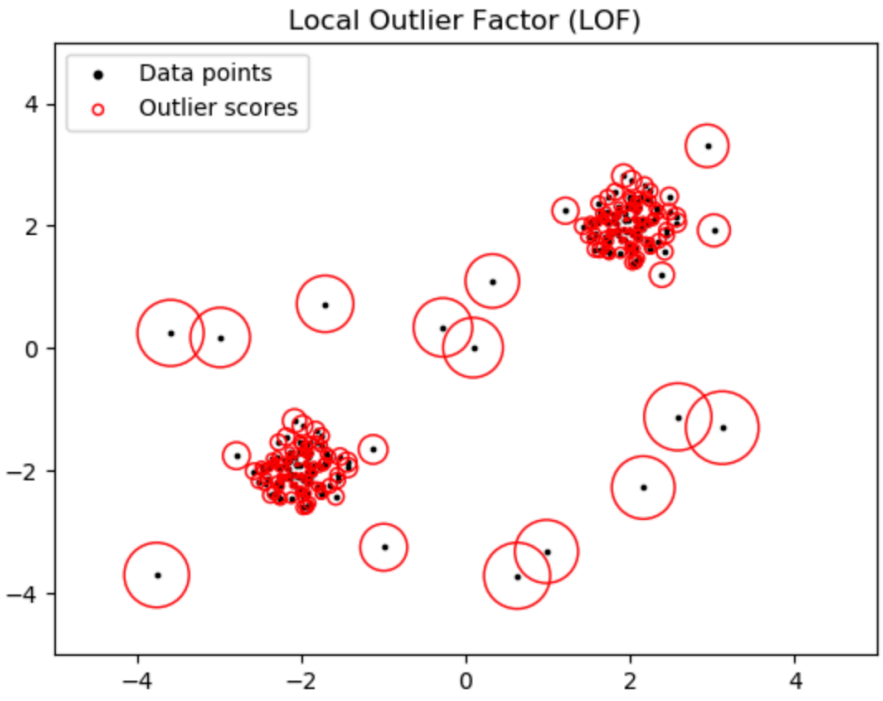

Figure 3.7: Outlier detection using LOF

The definition of “local“ is related to the fact that the algorithm takes into account only a restricted neighborhood of each object. We can examine the above observations in Figure 16, where we can see the performance of the LOF algorithm at a random dataset. This technique is loosely related to density-based clustering. However, it doesn’t require any explicit or implicit

notion of clusters. Thus, after the calculation of the local deviation, it considers as outliers the samples that have a substantially lower density than their neighbors. Moreover, another detail related to the performance of the LOF algorithm, is that both normal and anomalous instances are needed during the train phase. To fully understand the implementation of the LOF algorithm, we present a formal explanation below. Consider that n-distance(A) be the distance of the instance A to the n-th nearest neighbor. We define the set of n nearest neighbors as $N_n(A)$. This distance is determined as reachability distance in terms of the LOF algorithm, and defines the true distance of two objects:

$$reachability - distance_n(A, B) = max\{n - distance(B), d(A, B)\}$$

Now that we have the above metric, we can define the local reachability density of an object A with the below equation:

$$lrd_n(A) = \frac{1}{\frac{\sum_{B \in N_n(A)} reachability - distance_n(A,B)}{|N_n(A)|}}$$

With the above definition, we can calculate every local density of each point in our dataset and compare it with the neighbors. If the density of a point is lower than the neighbor's density, then the point is considered an outlier. In summary, we also apply the current algorithm to our model to identify the abnormal traffic using this density-based approach.

The last algorithm that we are using for our system is also a robust method related to outlier detection. The method was introduced in 1985 by Rousseeuw [**?**] and it's one of the first equivariant estimators of multivariate location and scatter. The MCD is resistant to outlying observations, and that why it is used to develop many multivariate techniques, which include principal component analysis(PCA) and multiple regression. However, for our purposes, we will adapt the MCD that uses the covariance matrix of the data points to provide an efficient outlier detection mechanism. Geometrically, the covariance matrix specifies an ellipsoid that circumscribes the primary dimensions of the data in N space. Outlier data stretches the ellipsoid along the axis of the outlier relative to the mean. We can examine the above observation in Figure 17.

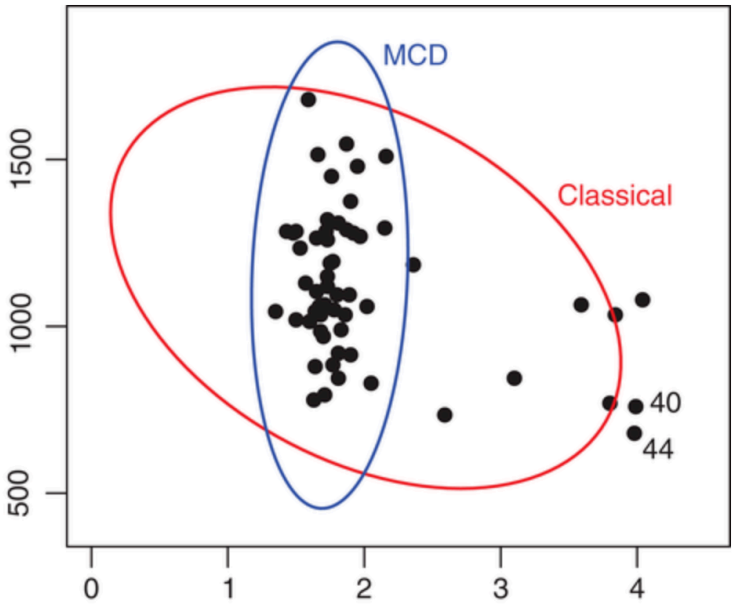

Figure 3.8: Minimum Covariance Determinant vs Classical

Now that we defined the geometrical approach of the MCD algorithm, let's analyze the steps that will separate the outlier from the rest of the data points. We assume that we have a random sample of size h. We can confirm the similarity between data points in the two datasets using the Mahalanobis distance [45]. Let M be the mean of the random subset and S be the standard covariance of the random subset.

$$D = [(x - M)S^{-1}(x - M)]^{\frac{1}{2}}$$

The Minimum Covariance Determinant algorithm is performed as follows:

- Choose a random subset of A of X, with size h.
- Determine covariance S and mean M of the subset A.
- Determine distances d(X_i) for all X_i relative to A with the Mahalanobis distance.
- Choose the h smallest distances and create a new subset A_1.
- Repeat with A = A_1 , until A and A_1 are equal or 0.
- Evaluate for K times and determine the selection that had the smallest volume.

Thus, our model has implemented internally the MCD algorithm using some python libraries [1][2][3] that follow the same steps that we described. After the training, our model is ready to identify the outliers again using the same methodology, and that provides us a robust outlier detection mechanism that offers to the IoT platform a reliable QoS tool. At this point, we have finally described every part of our system and how the entire infrastructure works. Thus, now we should proceed to the testing phase, where we will evaluate our implementation through various experiments.

[1] Scikit-learn
[2] Scapy
[3] PyOD

Chapter 4

Setup and Evaluation

In the current section, we will focus on describing the setup of our mechanism related to the platform and how the entire infrastructure works together. In detail, we will explain how our implementation will be combined with the platform's workflow without affecting the performance of the running services. Moreover, our model would be a separate component of the platform, but it would capture the traffic of each service that is running at the IoT platform. Without this approach, our implementation wouldn't be able to obtain the essential data for the training and evaluation phase. Next, after explaining each required step for setup at the platform, we will describe various experiments that we will attempt to understand the limits and the capabilities of our algorithm. Also, with those experiments, we will evaluate the performance of our mechanism and identify the weak spots to improve those vulnerabilities in the future. Finally, through this general evaluation, we will get a glimpse of the effectiveness of our method and also discover different techniques that will improve our implementation and provide more reliable and efficient results.

4.1 Setup of the QoS mechanism

Our mechanism, as we mentioned already in Chapter 3, relies on different Python libraries that implement the algorithms that we will need for our purpose. We adopt these libraries to create the infrastructure that we previously described that provides an efficient anomaly detection system that will ensure the fulfillment of the basic QoS requirements. Thus, we

need first to install Python on our machine to run our mechanism since every part of the implementation uses it. After that step, we should run the initial script that setups a tool that generates processes after the creation of a new service so that these processes will run the anomaly detection algorithm for each service. Therefore, for each service that is running, we have a "twin" of our mechanism handling each service and is independent of the other processes. In that way, every instance will not be affected by the other ones and will create an individual model that reflects the behavior of the service that is responsible for. We can understand better the initial script with the below algorithmic approach:

Algorithm 1 Create processes for each service

```
services = [ ]
while true do
    check the database
    if new service created and service not included in services then
        add service to services
        create a duplicate of our mechanism for that service
    end if
end while
```

Now we are done with the setup, and after that, each process starts filtering the traffic from the container, extracts the features, and generally follows the steps that we described in Chapter 3 (Page 49-50). The setup of our mechanism is not very complicated and has minimum requirements for the configuration. The only obligation that someones has is to run a few scripts, and everything is up and running. Our method is kept simple with specific algorithms that achieve our goal without burdening the entire IoT platform. After the process creation that we described, another script is triggered by our mechanism that will setup the feature extraction, training, and evaluation phase. When our model is trained using the three outlier detection algorithms, we can evaluate the performance of our method with new traffic instances that are not included in the training dataset. All the above steps are included in a script that runs the outlier detection system for each service and can be examined below:

For our model we used open source python libraries and also a simple approach to create a system for outlier detection that would be transparent to everyone and easy to transfer and setup to another IoT platform. Now that we explained the whole implementation and also the setup of our mechanism, we should proceed to the experiments to evaluate the performance

Algorithm 2 Model's Algorithm

```
iterations = 0
while true do
    capture the traffic and save it to pcap files.
    extract the features from the pcap files.
    if iterations >0  then
        retrain the model with the new instances.
    else
        train the model.
    end if
    increase the iterations by one
end while
```

and the effectiveness of our method.

4.2 Evaluation through Experiments

4.2.1 Service Setup

Before starting the experiments and see the effects of altering different variables, we should create a service to use it as a testbed for our purpose. Based on that service we will analyze the traffic that produces and setup our model at that instance to test the performance of our mechanism. The service that we use is a random generator that returns integers within a range every k seconds that we specify, and that sample is provided by the IoT platform. We chose a basic approach as a testbed to quickly install that service and also to understand better the effects after applying various reconfigurations. Those reconfigurations are related to the modification of the packet frequency, and also the payload that each packet transfers. Therefore, to install that service, we upload the docker file to the IoT platform, and we press the "build" button to build the docker for our service as we can observe in Figure 18.

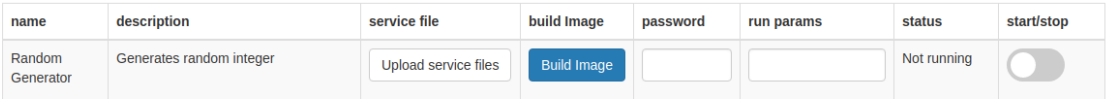

Figure 4.1: Upload dockerfile and build service

Then we are ready to run the service and start the experiments. When the service starts running, we can recognize the traffic and the requests that the service is handling at our terminal, as we can also examine in Figure 19. Additionally, we can retrieve the logs of the running service by entering the container, and inside /var/log see all the events/traffic, and also the results that the random integer generator produced. Another essential factor related to the service is that the random integer generator has two inputs (frequency, range), that define the range of the generated values and how often these values are produced.

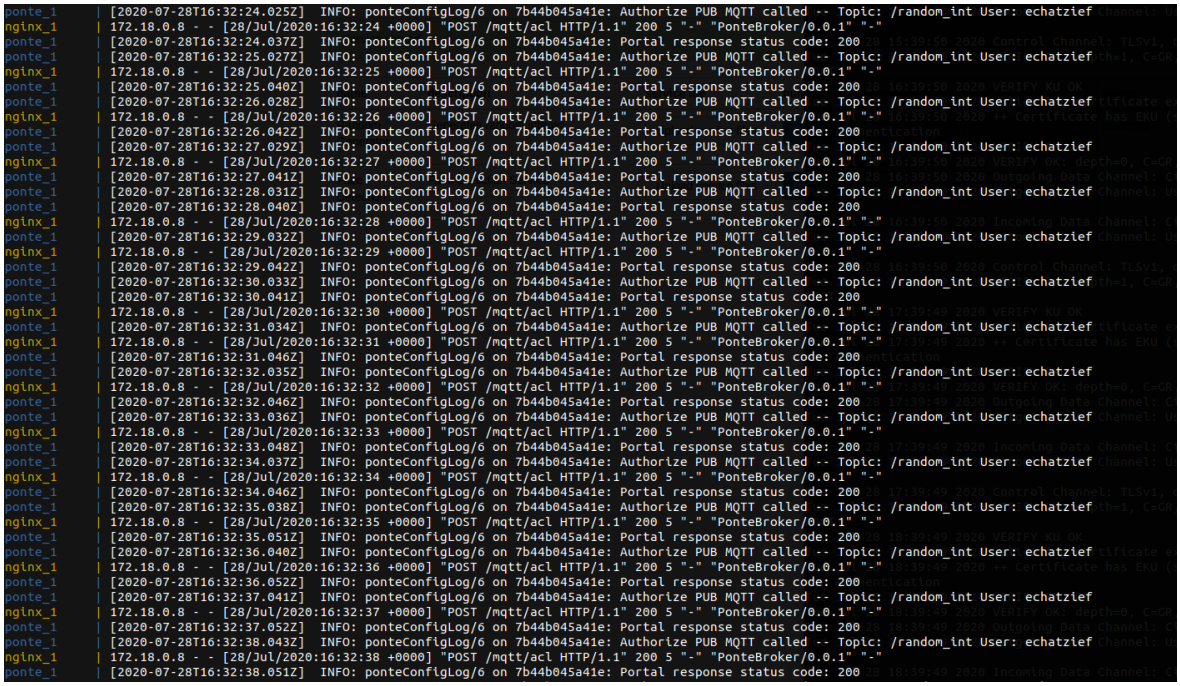

Figure 4.2: Service's traffic

Thus, the experiments that we would perform are related to these two fields that we tweak to examine the performance of each of the three proposed algorithms. To achieve the different variations to the two mentioned values, we will use a tool that is called mosquitto_pub and the command for altering an input is the following:

```
$ mosquitto_pub -h localhost -t <input> -i
  <container-id> -p 1883 -u <username>
  -P <password> -m <value>
```

Now that we explained all the components of the initial setup phase, we need to proceed to the following part and start the various modifications at the service's inputs.

4.2.2 Frequency Modifications

As we proposed, we will use the random integer generator for our evaluation purposes. This service has two input topics and one output topic. Those input topics are called frequency and range. The default values of frequency and range are 1 and 10. Thus, we will train our mechanism using the default case, and after that, we will tweak these two inputs to evaluate the performance of the three algorithms. First, we will be focusing on changing the frequency to receive integers with different time windows. In that way, we will observe how each implementation behaves when the timestamp of the packets differentiates. Therefore, using the mosquito_pub command, we increase the frequency to 5 seconds. After that, we will receive an integer from the service every 5 seconds, and that will increase the time difference between two incoming packets. Thus, using the iForest as the main component of our model instantly separates the outlier and recognizes that the packet flow has changed and tracks the outlier with a percent of 89%. Next, we try the same experiment using the Local Outlier Factor and Minimum Covariant Determinant, and identify the outlier with a percent of 96 % and 93% accuracy. Now, we should try to increase more the frequency to expand the time window of the traffic. Hence, we change the frequency topic to 10 seconds to make the contrast more obvious. After repeating the equivalent process and running each of the algorithms, we saw that all of them achieved 100 % accuracy at detecting the outliers. However, the above results are expected since most of the algorithms that we used are density-based. Thus, since the algorithms rely on distance metrics, it was apparent that the outliers will be identified by the algorithms because the time "distance" diverges among the experimental cases. The above experiments and the results are also displayed in Table 4.

Algorithm	5 second	10 second
iForest	89 %	100 %
Local Outlier Factor	96 %	100 %
Minimum Covariant Determinant	93 %	100 %

Table 4.1: Outlier detection accuracy after frequency changes.

4.2.3 Range Modifications

Now that we have compared the algorithms after the frequency altering, we can continue and tweak the range of the values that the random integer service produces. The range is the second input topic of the service that we use for the experiments, and it defines the range of the value that the generator will return. For example, if the range is 100, then the generator will provide numbers between 0 and 100. The default value of the range is 10, and we will try to change that value to 1000 and 10000 to observe how that will affect the three algorithms. To include the range into our dataset, we have defined a field called payload and also some other metrics related to the average of the payload, the minimum, and maximum value that we already introduced in Chapter 3. Thus, when we would modify the range of the generator, the payload and also the related metrics will diversify, and that will affect the behavior of the dataset. After that, we will evaluate the performance the three algorithms using the above dataset. Therefore, let's begin with the range modification to 1000 using the mosquitto_pub command. Using the iForest as the main component of our model, it achieves only 67 % accuracy to identify the outliers based on the range altering. However, the other two algorithms that are density-based achieve better performance with 97 % (LOF) and 96 % accuracy (MCD). Next, if we repeat the same experiment with a range of 10.000, we achieve similar results with the iForest being 72% and the other two 100% accurate. Since the range and also the frequency are distance related input topics, a density-based algorithm operates better than a more statistical one. Due to that fact the LOF and MCD that are density-based techniques perform better at the outlier detection than the iForest that has a more statistical approach. We can observe the above experiments and also the results at Table 5.

Algorithm	0 - 1000	0 - 10000
iForest	67 %	72 %
Local Outlier Factor	97 %	100 %

Minimum Covariant Determinant	96 %	100 %

Table 4.2: Outlier detection accuracy after range modifications.

Chapter 5

Conclusion

5.1 Summary

The rising importance of IoT-based services has resulted in the adoption of more sophisticated ways to provide improved services that meet rising QoS expectations. QoS is a major concern when it comes to multimedia services and tasks that require significant power and resources. Moreover, QoS is a design paradigm to map real-time constraints from the application and also offers a way to handle time requirements as delay, jitter, and throughput at any level of the architecture. Current methods either have low selection accuracy or are highly time-consuming (e.g., exponential time complexity), neither of which are desirable in large-scale IoT applications. In this paper, we presented various methodologies that are proposed in different researches. The researchers applied those techniques at different layers of the IoT stack, and each of those implemented a handling mechanism that fulfilled the QoS requirements. Next, we proposed a network-based QoS approach with a model that combine machine learning techniques to achieve outlier detection at the IoT platform called SYNAISTHISI. We used three well-known outlier detection techniques: (a) iForest, (b) Local Outlier Factor, (c) Minimum Covariant Determinant. The first one uses a statistical approach to define how an outlier behaves while the others are density-based algorithms that use distance metrics to identify the outlier from the rest of the dataset. Thus, through our experiments and evaluation, we realize that the problem that we are trying to eliminate can be solved using the density-based algorithms since our metrics perform better using numerical that statisti-

cal approaches. Finally, we came to the conclusion that our model can provide an efficient outlier detection system that will satisfy the QoS requirements of an IoT platform using the LOF, and MCD algorithms are the main components of our mechanism. All the source code for the entire project and some of the test cases that was used in these thesis are included in a public repository.

5.2 Future Work

As the Connected Life evolves, the number of interconnected devices is set to rise dramatically. These devices will bridge the physical and digital worlds, enabling a new category of services that will improve the quality of life and productivity. However, these devices have various QoS requirements to fulfill in order to provide those services and achieve their purposes. In recent years, several researchers have developed and construct different methods and using machine learning to satisfy the requirements that every IoT ecosystem needs. Thus, while machine learning techniques will evolve and rise, new mechanisms will be created every day. Therefore our future work will include an ensemble model of our technique that will combine two or all the algorithms that we proposed using a mechanism in the form of a pipeline that will provide a better outlier detection identification. Finally, another alternative would be to combine the current model using futuristic procedures that will be introduced in the next years.

Bibliography

[1] J. Liaperdos, A. Arapoyanni, and Y. Tsiatouhas. Bi intelligence estimates. Sept. 2015.

[2] Joe Hoffert, Daniel Mack, and Douglas Schmidt. Using machine learning to maintain qos for large-scale publish/subscribe systems in dynamic environments. In *8th Workshop on Adaptive and Reflective Middleware*, pages 1–5, Jan. 2009.

[3] O. Adinolfi, R. Cristaldi, L. Coppolino, and L. Romano. Qos-monaas: A portable architecture for qos monitoring in the cloud. In *8th International Conference on Signal Image Technology and Internet Based Systems*, pages 1–8, Nov. 2012.

[4] Changhe Yu, Julong Lan, JiChao Xie, and Yuxiang Hu. Qos-aware traffic classification architecture using machine learning and deep packet inspection in sdns. pages 1–8, Jan. 2018.

[5] Zehui Liu, Deyun Gao, Ying Liu, Hongke Zhang, and Chuan Heng Foh. An adaptive approach for elephant flow detection with the rapidly changing traffic in data center network. In *International Journal of Network Management*, pages 1–13, June 2017.

[6] Zhangyu Cheng, Chengming Zou, and Jianwei Dong. Outlier detection using isolation forest and local outlier factor. pages 1–7, September 2019.

[7] Bart De Ketelaere, Mia Hubert, Jakob Raymaekers, Peter J. Rousseeuw, and Iwein Vranckx. Real-time outlier detection for large datasets by rt-detmcd. In *Chemometrics and Intelligent Laboratory Systems*, pages 1–28, January 2020.

[8] Francesco Delli Priscoli, Alessandro Di Giorgio, Federico Lisi, Salvatore Monaco, Antonio Pietrabissa, Lorenzo Ricciardi Celsi, and Vincenzo Suraci. Multi-agent quality of experience control. pages 1–12, Decembers 2015.

[9] K.Ashton. That 'internet of things' thing. In *RFiD Journal*, pages 1–8, June 2009.

[10] Andreas Plageras, Kostas E. Psannis, Christos Stergiou, and Haoxiang Wang. Efficient iot-based sensor big data collection-processing and analysis in smart buildings. In *Future Generation Computer Systems*, pages 1–16, October 2017.

[11] ITU-T Y.4050/Y.2069. Terms and definitions for the internet of things. In *GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS AND NEXT-GENERATION NETWORKS*, pages 1–14, July 2012.

[12] H.Sundmaekerm, P.Guillemin, P.Friess, and S. Woelffle. Vision and challenges for realising the internet of things. In *Cluster of European Research Projects on the Internet of Things*, pages 1–15, April 2010.

[13] H.Sundmaekerm, P.Guillemin, P.Friess, and S. Woelffle. Vision and challenges for realising the internet of things. In *Cluster of European Research Projects on the Internet of Things*, pages 1–15, April 2010.

[14] R. Minerva, A. Biru, and D. Rotondi. Towards a definition of the internet of things (iot). In *IEEE Internet Initiative*, pages 1–35, August 2015.

[15] C.Perera, A.Zslavsky, P.Christen, and D.Georgakopoulos. Context aware computing for the internet of things: A survey. In *IEEE Communications Surveys and Tutorials*, pages 1–16, May 2013.

[16] libelium. 50 sensor applications for a smarter world. pages 1–30, May 2016.

[17] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. In *International Journal of Applied Mathematics*, pages 104–110, December 2010.

[18] O. Vermesan and P. Friess. Internet of things-from research and innovation to market deployment. pages 1–50, December 2014.

[19] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu. Security of the internet of things: perspectives and challenges. In *Wireless Networks*, pages 2481–2501, May 2013.

[20] H.Aksu, L.Babun, M.Conti, G.Tolomeit, and A.S.Uluagac. Advertising in the iot era: Vision and challenges. In *IEEE Communications Magazine*, pages 1–7, January 2018.

[21] Zozo Hassan, Hesham Arafat Ali, and Mahmoud M Badawy. Internet of things (iot): Definitions, challenges, and recent research directions. In *October 2015International Journal of Computer Applications*, pages 37–47, October 2015.

[22] Gary White, Vivek Nallur, and Siobhan Clarke. Quality of service approaches in iot: A systematic mapping. In *Journal of Systems and Software*, pages 1–19, May 2017.

[23] Keyur K. Patel and Sunil M. Patel2. Internet of things-iot: Definition, characteristics, architecture, enabling technologies, application and future challenges. pages 6122–6131, 2016.

[24] Anjum Sheikh, Asha Ambhaikar, and Sunil Kumar. Quality of services improvement for secure iot networks. In *International Journal of Engineering and Advanced Technology*, pages 1–9, December 2019.

[25] Ravi C Bhaddurgatte and Vijaya Kumar BP. A review: Qos architecture and implementations in iot environment. In *Research Reviews: Journal of Engineering and Technology*, Sept. 2015.

[26] Pallavi Sethi and Smruti R. Sarangi. Internet of things: Architectures, protocols, and applications. In *Journal of Electrical and Computer Engineering*, pages 1–24, Jan. 2017.

[27] Prachi Jain, Kancha Jha, and Sanjivani Patwa. Architecture of internet of things. In *IEEE Wireless Communications*, pages 1–14, June 2017.

[28] D. Hands, O. V. Barriac, and F. Telecom. Standardization activities in the itu for a qoe assessment of iptv. In *IEEE Communications Magazine*, pages 78–84, March 2008.

[29] M. Fiedler, K. Kilkki, , and P. Reichl. 09192 executive summary from quality of service to quality of experience. In *Dagstuhl Seminar Proceedings*, March 2009.

[30] K. Brunnstrom, S. A. Beker, K. De Moor, A. Dooms, S. Egger, M.-N. Garcia, T. Hossfeld, S. Jumisko-Pyykko, C. Keimel, and C. Larabi. Qualinet white paper on definitions of quality of experience. In *EU COST action 1003 QUALINET*, pages 1–12, March 2013.

[31] Parikshit Juluri, Venkatesh Tamarapalli, and Deep Medhi. Measurement of quality of experience of video-on-demand services: A survey. In *IEEE Communications Surveys and Tutorials*, pages 1–20, March 2016.

[32] A. Mellouk, H. A. Tran, and S. Hoceini. Quality-of-experience for multimedia. pages 1–40, March 2013.

[33] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang. Understanding the impact of video quality on user engagement. In *Communications of the ACM*, pages 1–12, October 2011.

[34] T. Hayashi. Qoe-centric operation for optimizing user quality of experience. pages 1–10, September 2015.

[35] M. Cilia, L. Fiege, C. Haul, A. Zeidler, and A. P. Buchmann. Looking into the past: Enhancing mobile publish/subscribe middleware. In *Databases and Distributed Systems Group*, pages 1–9, August 2003.

[36] Gary White, Andrei Palade, Christian Cabrera, and Siobhan Clarke. Quantitative evaluation of qos prediction in iot. In *Conference: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, pages 1–6, June 2017.

[37] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *ACM SIGCOMM Computer Communication Review*, pages 1–7, August 2015.

[38] S. Canale, F. Facchinei, R. Gambuti, L. Palagi, and V.Suraci. User profile based quality of experience. In *18th International Conference on Circuits, Systems, Communications and Computers (CSCC 2014)*, volume 2, pages 1–12, July 2014.

[39] L. Ricciardi Celsi, S. Battilotti, F. Cimorelli, C. Gori, Giorgi, S. Monaco, M. Panfili, V. Suraci, and F. Delli Priscoli. A q-learning based approach to quality of experience control in cognitive future internet networks. In *2015 23rd Mediterranean Conference on Control and Automation (MED)*, pages 1–12, June 2015.

[40] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *ACM SIGCOMM Computer Communication Review*, pages 1–12, July 2003.

[41] Mohammad Aazam and Khaled A. Harras. Mapping qoe with resource estimation in iot. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 1–15, April 2019.

[42] Eric L. Goodman, Chase Zimmerman, and Corey Hudson. Utilizing word2vec for feature extraction in packet data. In *MLDM 2019*, pages 1–15, April 2020.

[43] Leslie F.Siko. Packet analysis for network forensics: A comprehensive survey. In *MLDM 2019*, pages 1–15, March 2020.

[44] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *Data Mining, 2008*, pages 1–10, Jan. 2009.

[45] Mia Hubert, Michiel Debruyne, and Peter J. Rousseeuw. Minimum covariance determinant and extensions. In *Wiley Interdisciplinary Reviews: Computational Statistics*, pages 1–11, September 2017.