Filipe José Santa Bárbara Sequeira

# Development and Integration of Specialized Probes for IACS Applications

Dissertation/Internship Report
Master in Informatics Engineering
advised by Tiago José dos Santos Martins da Cruz
and presented to the Department Informatics Engineering
of the Faculty of Sciences and Technology of the University of Coimbra

July 2018

· U C ·

Universidade de Coimbra

UNIVERSITY OF COIMBRA

MASTER THESIS

# Development and Integration of Specialized Probes for IACS Applications

*Author:*
Filipe José Santa Bárbara Sequeira

*Advisor:*
PhD. Tiago CRUZ
*Co-advisor:*
PhD. Paulo SIMÕES

*A thesis submitted in fulfilment of the requirements for the degree of Masters in Informatics Engineering*

July 1, 2018

• U C •

FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

# *Abstract*

With the emergence of the Industry 4.0 paradigm, Industrial Automation and Control Systems are becoming highly distributed, making them increasingly prone to cyber-attacks, particularly when compared with previous Industrial Control System generations. To cope with the increased complexity and dimension of such infrastructures, it is necessary to use a diversified set of security probes/components.

However, the integration of new and already existing security probes into existing Intrusion and Anomaly Detection System platforms is no easy task. This is mostly due to the lack of common/standard communication protocols and APIs, as well as suitable data models (for configuration and eventing). This is crucial to simplify and streamline the configuration and monitoring processes but also the provisioning and registration of those mechanisms, used for such specialised agents.

With this in mind, this thesis will focus on developing a coupling layer, providing the means to integrate agents/probes in an Intrusion and Anomaly Detection System platform. This layer will provide the necessary generic adaptors to implement homogeneous management and event capabilities for agent/probe integration, as well as the required server-side components.

**Keywords** Critical Infrastructures, Industrial Automation and Control Systems, Management Agents

# Contents

**Abstract** **iii**

**1 Introduction** **1**
1.1 Problem Statement . . . 1
1.2 ATENA project overview . . . 2
1.3 Motivation . . . 3
1.4 Objectives . . . 3
1.5 Expected outcomes . . . 4
1.6 Thesis structure . . . 4

**2 State of the Art** **7**
2.1 Network management and monitoring background . . . 7
2.1.1 Network Management Functional Areas . . . 8
2.1.2 Network Management System . . . 9
2.1.3 Network Management System Components . . . 9
2.2 Monitoring and Management Protocols . . . 10
2.2.1 SNMP . . . 10
2.2.2 CoAP and LwM2M . . . 11
2.2.3 MQTT . . . 14
2.2.4 TR-069 . . . 18
2.2.5 NETCONF . . . 20
2.2.6 Management Protocols Wrap-up . . . 21
2.3 Protection of SCADA Systems . . . 22
2.3.1 Evolution . . . 22
2.3.2 Vulnerabilities . . . 23
2.3.3 Types of vulnerabilities . . . 25
2.3.4 Efforts to prevent attacks to SCADA systems . . . 27
2.4 Security Probes . . . 28
2.4.1 Examples of Security Probes . . . 29
2.5 State of Art - a brief review . . . 31

**3 System Overview** **33**
3.1 Reference Architecture . . . 33
3.2 Requirements . . . 36
3.3 Adopted Technologies and data formats . . . 38
3.3.1 Management solution . . . 38
3.3.2 Events solution . . . 39
3.4 System overview - a brief review . . . 39

**4 Development / Implementation** **41**
4.1 Application and Probe integration . . . 41
4.2 Management and monitoring solution . . . 43
4.2.1 Provisioning . . . 44

4.2.2 Connection . . . 45
4.2.3 Handling the management components . . . 46
4.2.4 Request types . . . 47
4.2.5 Failover mechanisms . . . 49
4.2.6 Security aspects . . . 50
4.2.7 Web interface . . . 51
4.3 Events . . . 51
4.3.1 Data collector . . . 51
4.4 Development overview . . . 52

**5 Tests and validation** **53**
5.1 Scenarios . . . 53
5.1.1 Agent Management . . . 53
5.1.2 Event Collector . . . 55
5.2 Test list . . . 55
5.2.1 Functional . . . 55
5.2.2 Performance (non-functional) evaluation . . . 68
5.3 Results and observed limitations . . . 68

**6 Work Plan** **87**

**7 Conclusions** **91**
7.1 Contributions . . . 91
7.2 Future Work . . . 92

**A Requirements** **93**

**B Management Protocol** **109**

**C Results** **117**

**Bibliography** **123**

# List of Figures

2.1 SNMP Usage . . . 11
2.2 LwM2M architecture . . . 13
2.3 Publisher-Subscriber Model . . . 15
2.4 LwM2M over MQTT Overview . . . 18
2.5 Example CWMP/TR-069 Session . . . 19
2.6 SCADA architecture . . . 22
2.7 How Stuxnet happened . . . 23
2.8 What happened in Stuxnet . . . 24
2.9 TRITON architecture . . . 25
2.10 Honeypot use case . . . 29
2.11 NIDS architecture . . . 30
2.12 HIDS architecture . . . 31

3.1 Incident and vulnerability management with ATENA modules . . . 33
3.2 IADS Referenced Architecture . . . 34
3.3 Management architecture and probe integration . . . 35
3.4 Generic Probe Architecture . . . 36

4.1 IADS Management architecture and probe integration . . . 42
4.2 Registration process on MQTT management solution . . . 44
4.3 Client disconnection and re-connection on MQTT management solution 46
4.4 Request process on MQTT management solution . . . 47
4.5 Client file change notification process on MQTT management solution 49
4.6 Broker failover on MQTT management solution . . . 49
4.7 Controller failover on MQTT management solution . . . 50

5.1 First Scenario - Management . . . 54
5.2 Second Scenario - Management . . . 54
5.3 Third Scenario - Events . . . 55
5.4 Test 1 - bar chart with results average . . . 69
5.5 Test 1 - box plot 5 runs . . . 70
5.6 Test 2 - bar chart with results average . . . 71
5.7 Test 2 - box plot 5 runs . . . 71
5.8 Test 3 - bar chart with results average . . . 72
5.9 Test 3 - box plot 5 runs . . . 72
5.10 Test 4 - bar chart with results average . . . 73
5.11 Test 4 - box plot 5 runs . . . 74
5.12 Test 5 - bar chart with results average . . . 75
5.13 Test 5 - box plot 5 runs . . . 75
5.14 Test 6 - bar chart with results average . . . 76
5.15 Test 6 - box plot 5 runs . . . 77
5.16 Test 7 - bar chart with results average . . . 78
5.17 Test 7 - box plot 5 runs . . . 78

5.18 Test 8 - bar chart with results average . . . . . 79
5.19 Test 8 - box plot 5 runs . . . . . 79
5.20 Test 9 - bar chart with results average . . . . . 80
5.21 Test 9 - box plot 5 runs . . . . . 81
5.22 Test 10 - bar chart with results average . . . . . 82
5.23 Test 10 - box plot 5 runs . . . . . 82
5.24 Test 11 - bar chart with results average . . . . . 83
5.25 Test 11 - box plot 5 runs . . . . . 83
5.26 Test 12 - bar chart with results average . . . . . 84
5.27 Test 12 - box plot 5 runs . . . . . 84
5.28 Test 13 - bar chart with results average . . . . . 85
5.29 Test 13 - box plot 5 runs . . . . . 85

6.1 Gantt Chart . . . . . 88
6.2 Gantt Chart updated . . . . . 89

B.1 Registration process on MQTT management solution . . . . . 110
B.2 Request process on MQTT management solution . . . . . 111
B.3 Client file change notification process on MQTT management solution 112
B.4 Broker failover on MQTT management solution . . . . . 113
B.5 Controller failover on MQTT management solution . . . . . 114
B.6 Client disconnection and re-connection on MQTT management solution 115

# List of Tables

2.1 SNMP Versions [10] . . . 11
2.2 LwM2M Object Resources (ID Range) . . . 13
2.3 Summary of Key Comparison Criteria (adapted from [51]) . . . 21
2.4 Number of Vulnerabilities in Industrial Control Systems Components by [57] . . . 26
2.5 Classification of Vulnerabilities in Industrial Control Systems According to Type by [57] . . . 26

3.1 Management Probes functional requirements summary . . . 37

5.1 Test 1 (Scenario 1 - Management) . . . 56
5.2 Test 2 (Scenario 1 - Management) . . . 57
5.3 Test 3 (Scenario 1 - Management) . . . 58
5.4 Test 4 (Scenario 1 - Management) . . . 59
5.5 Test 5 (Scenario 1 - Management) . . . 60
5.6 Test 6 (Scenario 2 - Management) . . . 61
5.7 Test 7 (Scenario 2 - Management) . . . 62
5.8 Test 8 (Scenario 2 - Management) . . . 62
5.9 Test 9 (Scenario 2 - Management) . . . 63
5.10 Test 10 (Scenario 2 - Management) . . . 64
5.11 Test 11 (Scenario 2 - Management) . . . 65
5.12 Test 12 (Scenario 2 - Management) . . . 66
5.13 Test 13 (Scenario 2 - Management) . . . 66
5.14 Test 14 (Scenario 3 - Events) . . . 68
5.15 Test results summary . . . 69

# Abbreviations and Acronyms

| | |
|---|---|
| **AMQP** | **A**dvanced **M**essage **Q**ueuing **P**rotocol |
| **API** | **A**pplication **P**rogramming **I**nterface |
| **ATENA** | **A**dvance **T**ools to ass**E**ss and mitigate criticality of ICT compo**N**ents and their dependecies over Critical Infr**A**structures |
| **ACS** | **A**uto **C**onfiguration **S**ervers |
| **BIOS** | **B**asic **I**nput/**O**utput **S**ystem |
| **CISUC** | **C**enter of **I**nformatics and **S**ystems of the **U**niversity of **C**oimbra |
| **CMOS** | **c**omplementary **m**etal-**o**xide-**s**emiconductor |
| **CoAP** | **C**onstrained **A**pplication **P**rotocol |
| **CPE** | **C**ustomer **P**remises **E**quipment |
| **CWMP** | **C**PE **W**an **M**anagement **P**rotocol |
| **DCS** | **D**istributed **C**ontrol **S**ystem) |
| **DDS** | **D**ata **D**istribution **S**ervice |
| **DIDS** | **D**istributed **I**ntrusion **D**etection **S**ystem |
| **DPWS** | **D**evices **P**rofile for **W**eb **S**ervices |
| **EMQ** | **E**rlang **M**QTT **B**roker |
| **ESR** | **E**mbedded **S**pecification **R**equest |
| **HIDS** | **H**ost-based **I**ntrusion **D**etection **S**ystem |
| **HMI** | **H**uman **M**achine **I**nterface |
| **HTTP** | **H**yper**t**ext **T**ransfer **P**rotocol |
| **IACS** | **I**ndustrial **A**utomation and **C**ontrol **S**ystems |
| **IADS** | **I**ntrusion and **A**nomaly **D**etection **S**ystem |
| **ICT** | **I**ndustrial **C**ontrol **T**echnology |
| **IDS** | **I**ntrusion **D**etection **S**ystem |
| **IEC** | **I**nternational **E**lectrotechnical **C**ommission |
| **IETF** | **I**nternet **E**ngineering **T**ask **F**orce |

| | |
|---|---|
| **IoT** | **I**nternet of **T**hings |
| **I/O** | **I**nput/**O**utput |
| **IPS** | **I**ntrusion **P**revention **S**ystem |
| **IP** | **I**nternet **P**rotocol |
| **ISO** | **I**nternational **O**rganization for **S**tandardization |
| **JSON** | **J**ava**S**cript **O**bject **N**otation |
| **LCT** | **L**aboratory of **C**ommunications and **T**elematics |
| **LwM2M** | **L**ightweight **M**achine to **M**achine |
| **LWT** | **L**ast **W**ill and **T**estament |
| **MIB** | **M**anagement **I**nformation **B**ase |
| **MQTT-SN** | **M**essage **Q**ueuing **T**elemetry **T**ransport for **S**ensor **N**etworks |
| **MQTT** | **M**essage **Q**ueuing **T**elemetry **T**ransport |
| **NAT** | **N**etwork **A**ddress **T**ranslation |
| **NIDS** | **N**etwork **I**ntrusion **D**etection **S**ystem |
| **NMA** | **N**etwork **M**anagement **A**pplication |
| **NMS** | **N**etwork **M**anagement **S**ystem |
| **OMA** | **O**pen **M**obile **A**lliance |
| **OSI** | **O**pen **S**ystems **I**nterconnection |
| **PLC** | **P**ower **L**ine **C**ommunication |
| **PLC** | **P**rogrammable **L**ogic **C**ontroller |
| **QoS** | **Q**uality **o**f **S**ervice |
| **RAM** | **R**andom **A**ccess **M**emory |
| **REST** | **RE**presentational **S**tate **T**ransfer |
| **RPC** | **R**emote **P**rocedure **C**all |
| **RSMB** | **R**eally **S**mall **M**essage **B**roker |
| **RTU** | **R**emote **T**erminal **U**nit |
| **SCADA** | **S**upervisory **C**ontrol and **D**ata **A**cquisition |
| **SDN** | **S**oftware **D**efined **N**etwork |
| **SGMP** | **S**imple **G**ateway **M**anagement **P**rotocol |
| **SIEM** | **S**ecurity **I**nformation and **E**vent **M**anagement |
| **SIS** | **S**afety **I**nstrumented **S**ystem |

| | |
|---|---|
| **SNMP** | **S**imple **N**etwork **M**anagement **P**rotocol |
| **SOAP** | **S**imple **O**bject **A**ccess **P**rotocol |
| **SSH** | **S**ecure **Sh**ell |
| **SSL** | **S**ecure **S**ockets **L**ayer |
| **STOMP** | **S**imple/Streaming **T**ext **O**riented **M**essaging **P**rotocol |
| **TCP** | **T**ransmisson **C**ontrol **P**rotocol |
| **TLS** | **T**ransport **L**ayer **S**ecurity |
| **TR-069** | **T**echnical **R**eport **069** |
| **UDP** | **U**ser **D**atagram **P**rotocol |
| **URI** | **U**niform **R**esource **I**dentifier |
| **vNIDS** | virtual **N**etwork **I**ntrusion **D**etection **S**ystem |
| **XML** | E**X**tensible **M**arkup **L**anguage |
| **XMPP** | E**x**tensible **M**essaging and **P**resence **P**rotocol |

# Chapter 1

# Introduction

The main goal of this Chapter is to introduce the work performed in our thesis. We start this Chapter by introducing the problem statement and providing an overview of the **A**dvance **T**ools to ass**E**ss and mitigate criticality of ICT compo**N**ents and their dependecies over Critical Infr**A**structures (ATENA) project. Next, we describe the reasons that lead us to develop the solutions followed by the thesis objectives. Finally, we provide a brief description of the thesis structure.

## 1.1 Problem Statement

The exponential growth of the industry and consequently the number of devices used to control the processes within the industry led to the need of having a centralised and uncomplicated way of controlling and monitoring these systems.

In a **I**ndustrial **A**utomation and **C**ontrol **S**ystems (IACS) there are multiple components that need to be managed. These components are developed by multiple entities with no standard communication payload, communication protocol, configuration or even management. Therefore, it is necessary to find a unified solution for its management. Furthermore, these components do not have a standard output so there is the need to find a uniform wrapper to communicate with the **I**ntrusion and **A**nomaly **D**etection **S**ystem (IADS).

The multiple components IADS encompasses need to be monitored, managed and sometimes provisioned. To solve this, there are two main steps to go through: finding an adequate protocol and set of tools as well as develop a solution based on those tools and protocol. In order to find the adequate protocol, it is mandatory to have in mind that:

- the developed agents should have the ability to register into the IADS platform;
- the developed agents have to be easily configurable from the platform;
- the developed agents need to communicate in a uniform encoded data model;
- the platform needs to activate and deactivate registered agents or components within the agents, or even change some parameters like the verbose level of the agent;
- the platform has to monitor some of the developed agents;
- the developed server component will be responsible for managing and registering the agents.

Before explaining the objectives and tasks meant for this thesis, we need to establish that the developed solutions in this project were created to fulfil the requirements of ATENA's project.

## 1.2 ATENA project overview

This thesis is framed on the ATENA project, to fulfil several requirements of the Work Package 4 - IADS - from which **L**aboratory of **C**ommunications and **T**elematics (LCT)/**C**enter of **I**nformatics and **S**ystems of the **U**niversity of **C**oimbra (CISUC) is responsible. In this Work Package it is being developed a solution capable of collecting, analysing and displaying information collected from IACS.

ATENA Project is part of a European Commission Framework Program for Research and Technological Development called Horizon 2020. This project focus on innovating Cyber Security and **I**ndustrial **C**ontrol **T**echnology (ICT). This project was created with the intention to develop an innovative framework of tools and processes to increase security and resilience of Critical Infrastructures, which is made by using new anomaly detection algorithms and risk assessment methodologies within a distributed cyber-physical environment, therefore, it provides a suite of ICT networked components and advanced tools embedding innovative algorithms for both correct static Critical Infrastructures configuration and fast dynamic Critical Infrastructure reaction in presence of adverse events [1].

Having this in mind, the solution of this project is a modular framework that consists of the aggregation of multiple components made by the several partners of the project. Besides, the development of the solution is also required for designing the architecture, use cases and respective documentation. Although the whole infrastructure is supposed to be a uniform solution, there is the need of independence in each of the components to avoid risks associated to partner failures and to allow reuse of components.

In LCT is being developed a solution that will focus on the internal infrastructure of the industry, essentially protecting the **S**upervisory **C**ontrol and **D**ata **A**cquisition (SCADA) environment. This solution will consist of acquiring data from the network, filtering it and analysing it to detect anomalies that are considered threats. Therefore, in the Work Package 4 will be designed and developed cybersecurity detection agents and a distributed awareness layer.

The proposed IADS developed in the Work Package 4 consists of a solution divided into 4 major components: management and monitoring, **S**ecurity **I**nformation and **E**vent **M**anagement (SIEM), domain processors, and probes. The management part is parallel to the other components, being responsible for monitoring and controlling them.

Probes are the components responsible for collecting data into the platform, which can encompass two different types:

- **Normal** - all the data collected is sent as it is, for example, a network packet collector.
- **Processed data** - for instance, a **N**etwork **I**ntrusion **D**etection **S**ystem (NIDS), or information coming from a honeypot, which will only send information considered important such as attempts of attack or similar threats.

Afterwards, the data collected is sent to the Domain Processor, where the distributed streaming platform is responsible for processing and transporting the data. This way, the data will be properly stored and secured. Next, the data will be sent to

an open-source cluster-computing framework (Spark [2]) where it will be processed using some machine learning algorithms to distinguish between potential threats and normal traffic. Those algorithms can detect anomalous patterns that were not detected in previous components of the infrastructure.

Finally, all the information is visualised in a dashboard with graphics and tables to consolidate the information in a human readable manner. Furthermore, the platform has the necessary tools to control the infrastructure, meaning there is a management part exposed in the dashboard.

## 1.3 Motivation

The creation of both management and data collection allows us to solve some of the problems identified in the ATENA project:

- The creation of a management solution allows to expose multiple software configurations in a converged way without exposing the real configurations or configuration file paths to the users. Furthermore, it allows monitoring of virtually any component the administrator wants to.
- The creation of a data collecting solution solves the problem of having software using different communication payloads and different communication protocols by converging the payloads to a unique data model which eases data mining tasks.
- The creation of uncoupled solutions allows the solutions to be reused in other projects.
- The creation of a management solution that monitors parameters of those components in a lightweight and secure manner is what makes this project so appealing.

## 1.4 Objectives

In order to address the aforementioned issues, we need to create a solution allowing the data collection and management of security probes (as well as other components) in IACS. In order to complete this objective, it is required to find or create a suitable managing and monitoring protocol allowing to reduce some of the security flaws SCADA has since its beginning.

Besides the management solution, we also need to develop a generic solution for data collection in the probes. These probes have the objective of improving the infrastructure security by, for instance, collecting data as a NIDS. However, the objective is not developing parts of a solution but a solution with components that work as a whole system, this meaning the probes (and their data collection mechanisms) have to be controlled by the management system that will orchestrate them as necessary.

In order to fulfil these main objectives the following tasks were identified:

- Acquire knowledge of Industrial Automation and Control Systems based on SCADA;
- Identify the available solutions, tools, and frameworks to develop the solutions;

- Propose an architecture for the management solution to be implemented;
- Analyse components and tools to implement the agents;
- Implement both agents and management solution;
- Test and validate the solutions presented;

## 1.5 Expected outcomes

The objectives of this dissertation implicitly encompass an expected set of outcomes, namely:

- The development of a server to manage devices (agents);
- The development of management and communication adapters (wrappers);
- The integration and deployment of agents in the system;
- And, finally, the development of a security probe (for Proof of Concept purposes).

To achieve these outcomes, a requirement elicitation step was undertaken as part of the early planning and design stages, in order to properly establish the necessary groundwork for their proper development. Such effort will be further documented in the next chapters.

## 1.6 Thesis structure

The next chapter - **State of the art** (Chapter 2) - presents a brief background in network management explaining the essentials of its functionalities, architecture, and objectives. Afterwards, the evolution of network monitoring and management protocols, and afterwards an analysis of those protocols. This allowed us to understand what were the mistakes made and what are the features to preserve for future protocols. Following, a comparison of protocols with identical capabilities in network monitoring and management, which leads to the decision of the technologies used in this work. Next, an explanation of what SCADA is and the evolution made alongside the generations. We also focus on its vulnerabilities and attacks, that we intend to monitor and detect on the IADS. Finally, a description of security probes, as well as the functionalities they are designed for. We also provide an overview of relevant probes.

The third chapter - **System overview** (Chapter 3) - presents the Requirements of the management solution and security probes developed. Afterwards, we propose a reference architecture for the solutions, explain the technologies used during the development and the data formats used to communicate between components. Finally, a brief explanation of the reasons why we made the choices based on the previously acquired knowledge from the State of the art (Chapter 2).

The fourth chapter - **Development / Implementation** (Chapter 4) - contains information about all the developed components and the management protocol, i.e., how we implemented the solutions, the decisions made and deviations from the original plan. We also explained its interaction with the rest of the IADS platform.

The fifth chapter - **Tests and validation** (Chapter 5) - explains the used testbed/scenarios used to validate our solutions. Both hardware and test description are as detailed as possible.

The sixth chapter - **Work Plan** (Chapter 6) - explains what was the proposed schedule to develop the solutions and the deviations this schedule had. This information is complemented by Gantt diagrams to validate the proposed scheduling.

Finally, in the seventh chapter - **Conclusions** (Chapter 7) - a brief review is made, the contributions made across the development of this project are described and proposed some features that may be developed as a future work.

# Chapter 2

# State of the Art

In this chapter, we described the main topics covered in this document. We started by introducing the concept, and some fundamentals of network management and monitoring, since we are developing a server to manage and monitor components and agents (probes). Although network management systems are not directly related to our tasks, there are multiple paradigms and methodologies, we can learn/use from these kind of systems since we are managing devices spread across networks. From those similar paradigms and methodologies, we can collect information about the necessary requirements and characteristics for our management solution and API, and we can avoid mistakes when planning our solution by learning from errors and solutions (types of payloads and security mechanisms) committed by other in this kind of management systems. Before starting to develop our solutions we analysed some management and related protocols and tools. Then we described the characteristics present in each of these protocols, what distinguishes them and what are the main advantages and disadvantages of them. After, we make a brief contextualisation into the SCADA paradigm and why it is necessary to have a correct device management system for this paradigm. We also introduced the concept of security probes and give some examples of probes since our developed solutions focus on these agents.

## 2.1 Network management and monitoring background

To properly understand how a management agent works and what are the features and concerns that need to be taken care of, it is necessary to understand the chronological evolution of these management agents, what were the problems that led to their substitution and what improvements were brought that should be maintained for future technologies.

Even before explaining the evolution of a Network Management Agent had in the history it is necessary to understand four particular aspects of Network Management. These aspects are:

- Network Management Requirements;
- Network Management System;
- Network Management components;

### 2.1.1 Network Management Functional Areas

**I**nternational **O**rganization for **S**tandardization (ISO) defined the key functional areas of a network management for **O**pen **S**ystems **I**nterconnection (OSI) environment. However, these aspects ended being adopted not only by vendors of proprietary network management systems but also by vendors of standardised network management systems. These functional areas are the following:

#### Fault Management

The objective of this procedure is to guarantee that the component failure will not compromise the whole network infrastructure or even a factory if this is the case. Therefore, whenever a fault occurs the following steps happen:

- Determine where the fault happens with precision;
- Isolate the network from the failing component to ensure the network is not affected by the faulty component and continues to work as expected;
- Modify or (re)configure the network without the affected components minimising the damage to the component as efficiently as possible;
- Replace (or if possible repair) the affected components to restore the normal network behaviour.

#### Accounting Management

When some of these unfortunate events occur and it is necessary to replace or repair a component, there is an entity responsible for identifying the end user using that network resources and also tracking costs. In this way, it is possible to track an end user or a group abusing from privileges and prejudicing other users and the network. It is also possible to understand if users are making inefficient use of the network, in this way it is possible to assist changing procedures to improve the performance. Finally, if the network manager has knowledge about the end-user activity it is easier for him to plan network growth.

#### Configuration and Name Management

The main objective of configuration and name management is to monitor network and system configuration information, enabling tracking the changes of both software and hardware and managing them accordingly to our needs. For instance, we can have a device that is generally used as a router, but it is necessary for another part as a temporary node of the infrastructure, and we can configure it to act as a switch. This kind of changes is defined to happen transparently to the end users.

#### Performance Management

When talking about performance management facilities we are expecting them to evaluate and analyse the effectiveness of managed objects and communication activities. Questions like if there is enough bandwidth for all services or what is the actual capacity utilisation and how much is still left, if there are bottlenecks, or even if the response time is adequate for the infrastructure. Clearly, to perform this kind of tests it is needed to have a default evaluation with the standard resources to measure the performance in other time instances.

**Security Management**

Finally, the facilities are responsible for ensuring that the network devices are not compromised and to protect them in the case they are. Some of the aspects this entity has to ensure are access to facilities and protecting managing information. Among other tasks, it is necessary to control, create and distribute passwords and access keys, control access to networks, monitor the collected data from nodes and control the logs not only from the network but also the devices monitored in it.

### 2.1.2 Network Management System

A **N**etwork **M**anagement **S**ystem (NMS) is a collection of tools or framework, aggregated with the necessary hardware, used to monitor and control a network. The main purpose of this system is to see the network infrastructure as a unique component having a centralised way to monitor and control it as a whole. Ideally, this solution has the minimum addition of both software and hardware, of course for this to happen it is necessary the hardware producers use a widely accepted protocol. Having a single interface with user-friendly instructions and a set of simple instructions to perform the network managing tasks is desirable. For each network this set of tools responsible for network tasks is designated as a network management entity, this entity is responsible for tasks like:

- Collect statistics and logs of network-related activities
- Store the previously collected data locally
- Respond to orders from a control centre, such as:
  - Transmit the data to the control centre
  - Change configurations/ parameters
  - Providing devices' health-related and other status information
- Alert the control centre whenever significant changes happen

In addition to the network management entity, we also have a **N**etwork **M**anagement **A**pplication (NMA) that is designated as the manager or network control host, where we have an operator interface that allows an authorised user to control the network. The NMA is used for the allowed users to obtain information accordingly to the inputted commands or even sending instructions to the network management entities.

### 2.1.3 Network Management System Components

Even though there are multiple components in a Network Management System, some of them are always necessary otherwise it is not possible to have a monitoring and control solution.

- **Network device discovery** - This component is responsible for identifying the devices present in the network topology.
- **Network device monitoring** - Here we have the component responsible to determine the status and health of each individual component.

- **Network performance analysis** - Packet loss, available bandwidth, latency, uptime and availability of routers, servers switches or any device using a network management protocol, all these parameters are tracked in this component.
- **Intelligent notifications** - Whenever there is a fault in the system or a component behaves in an abnormal way its necessary to send a configurable alerts to the entity responsible for the network management, this alert can be from one of the following types email, text message or even a phone call if there is a gateway for that purpose.

## 2.2 Monitoring and Management Protocols

In this subsection we analyse the most recent managing protocols and corresponding tools being used by multiple enterprises including network providers to manage their devices and network devices.

The reason that led us to analyse this protocols is the necessity of developing a management protocol for IACS applications.

Remote monitoring and management of industrial devices have evolved significantly since its primordial times. For instance, in the seventies and part of eighties, this task was made using 1200 baud modems which at the time were reliable since most of the telemetry protocols, like Modbus, were deterministic [3] (there are no uncertain delays in network transmission).

However, nowadays with the huge amount of data and devices IACS has these solutions are not viable. Therefore, we need to analyse what are the requirements for the industry, what is not acceptable and what is necessary to resolve. Hence this information could be summed up with the necessity of creating a generic, lightweight and secure protocol, otherwise, it is impossible to expect widespread acceptance.

### 2.2.1 SNMP

It was the year of 1988 when a team of 6 people developed a protocol that had widespread acceptance as a management agent of network devices. This protocol was designated as **S**imple **N**etwork **M**anagement **P**rotocol (SNMP), and became a widely used protocol in the industry. What led to its adoption was the increasing number of devices using **I**nternet **P**rotocol (IP)-based hardware, but these devices had a problem they used proprietary protocols and SNMP emerged as an open and generic protocol capable of uniforming the monitoring of devices [4].

This application-layer protocol derived from **S**imple **G**ateway **M**anagement **P**rotocol (SGMP), and it already has reached the third version. Although, there is a slight problem, the industry adopted the **SNMPv1** (described in [5]) despite its simplicity to configure and its effectiveness when comes to monitoring (Figure 2.1) it represents a security hole in the entire monitoring infrastructure, since all its communication uses plain text [5]. In other words anyone that is in the network will have access to the community string (this is the credential necessary to obtain information) in plain text, which may lead to masquerading, eavesdropping, modification of information, message stream modification, traffic pattern analysis and its even susceptible to denial of services, therefore, providing no security at all. The widespread acceptance of this protocol is obtained in major due to the **M**anagement **I**nformation **B**ase (MIB) trees, the database used for SNMP to manage the devices or entities in a network [6].

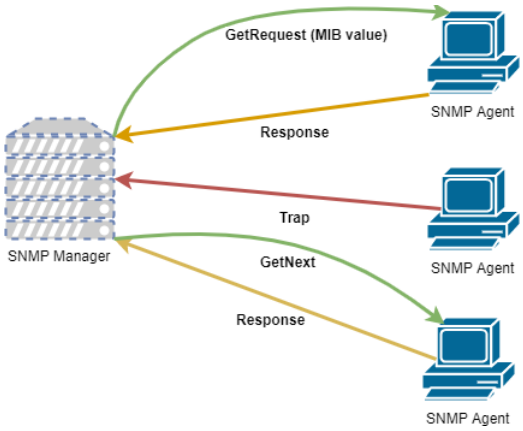


FIGURE 2.1: SNMP Usage

As an attempt to rectify the mistakes done in the first version, there were 3 other versions of SNMPv2. Two of them weren't initially accepted since they lost the retro-compatibility with the older protocol. The one accepted was **SNMPv2c** (described in RFC1902), and the two protocols with a user-based security model (SNMPv2u and SNMPv2*) replaced in prole of the retro-compatibility, which led to another version, also insecure, relying just on community strings, that once again compromised the security of the protocol [7].

Finally, with the third version of this protocol, **SNMPv3**, it was implemented both authentication and encryption, that can be used at the same time, separately or even without any of them. These characteristics made this protocol a full **I**nternet **E**ngineering **T**ask **F**orce (IETF) standard, but they achieved another mark, with the overload added to the protocol and higher configuration complexity, this new IETF standard was not accepted for both industry and general networks [8] [9].

| Protocol/ Characteristic | Level | Authentication | Encryption | What happens to access the agent |
|---|---|---|---|---|
| **SNMPv1** | No Authentication, No Privacy | Community String | No | Plain text community string |
| **SNMPv2c** | No Authentication, No Privacy | Community String | No | Plain text community string |
| **SNMPv3** | No Authentication, No Privacy/ Authentication, No Privacy / Authentication, Privacy | Username or MD5 / SHA or MD5 / SHA | No/ No/ DES | Username match / Hash-based Message Authentication Code MD5 or SHA / Hash-based Message Authentication Code MD5 or SHA. DES 56-bit encryption |

TABLE 2.1: SNMP Versions [10]

Although SNMP (Table 2.1) proved to be an adequate management protocol, it is not suitable to be used in the agents to integrate with the IADS, since it lacks some of the main requirements discussed in Chapter 3, and consequently would need the addition of other protocols to solve this problem, creating overhead. Among the lacking requirements, the absence of a uniform management solution for multiple devices makes SNMP inappropriate.

### 2.2.2 CoAP and LwM2M

In this section, we introduce **C**onstrained **A**pplication **P**rotocol (CoAP), an application protocol for constrained devices and applications. The reason that led us to introduce this protocol is the management protocol **L**ightweight **M**achine to **M**achine (LwM2M), which is based on CoAP.

### CoAP

CoAP is a RESTful specialised Internet application protocol (described in RFC7228 [11] and is an IETF standard) that runs over **U**ser **D**atagram **P**rotocol (UDP). This means it follows **RE**presentational **S**tate **T**ransfer (REST) Architecture although it is lighter than the one provided in **H**yper**t**ext **T**ransfer **P**rotocol (HTTP). To protect its transmissions CoAP uses Datagram **T**ransport **L**ayer **S**ecurity (TLS), analogous to TLS in HTTP (HTTPs) but when it comes to performance its proven to be less demanding than HTTPs and also more responsive [12]. The REST endpoints represent resources (like sensors) as RESTful objects and have some methods (GET, POST, PUT and DELETE) available to perform actions [11].

These resources are identified as **U**niform **R**esource **I**dentifier (URI), and using the right method it is possible to monitor/observe certain parameters/resources/objects that will transmit information from time to time, allowing the detection of failing components and consequently preventing a bigger issue.

To summarise CoAP is fast and efficient due to UDP (low-overhead), follows a RESTful model, has datagram level security which guarantees the integrity and confidentiality of the messages, allows resource observation and block wise resource transport (in this way enables the possibility to exchange transceiver data without updating the whole data to reduce overhead), allows interaction with HTTP through a proxy, has multicast support and finally has the ability to discover resources as a result of URIs. For instance the server uses a well-known URI [13]. There are some concerns about the UDP reliability when it comes to packet loss rate and also some problems when working in networks with **N**etwork **A**ddress **T**ranslation (NAT)-enabled routers [14].

### LwM2M

The LwM2M has a client-server model and it uses REST-based **A**pplication **P**rogramming **I**nterface (API) that implements a Resource and Object Data Model. It is also important to refer that this protocol is based on a secure and efficient communication mechanism called CoAP.

As illustrated in Figure 2.2, LwM2M has 4 main interfaces: Bootstrap, the Client Registration, Information Reporting and Device Management.

The Bootstrap initialises the objects and the connection. This means that this protocol offers mechanisms for instantiation and configuration of the agents after we reach the phase where a client can make is registration to a server, sending it the information about all of its objects that were founded and initialised. Another peculiar characteristic is the fact that a client can register to more than one server, allowing some redundancy for security and quality of service reasons in the Probe management system.

A Device Management interface has the interactions to trigger tasks like object handling, instances or even resources. Lastly, the Information Reporting handles the resources observation.

There are some standard objects in the LwM2M specification. These are available for consulting on the **O**pen **M**obile **A**lliance (OMA)'s Website. To try to achieve a way to compress the relevant objects information, Table 2.2 was created below.

As shown in the Table 2.2 there are 3 different ranges or pools, of values corresponding to the object IDS. The first range was created by OMA, and they are the only set of individuals who can edit them. The second range was created for any company that needs to create an object that will be vastly used, and so they are able

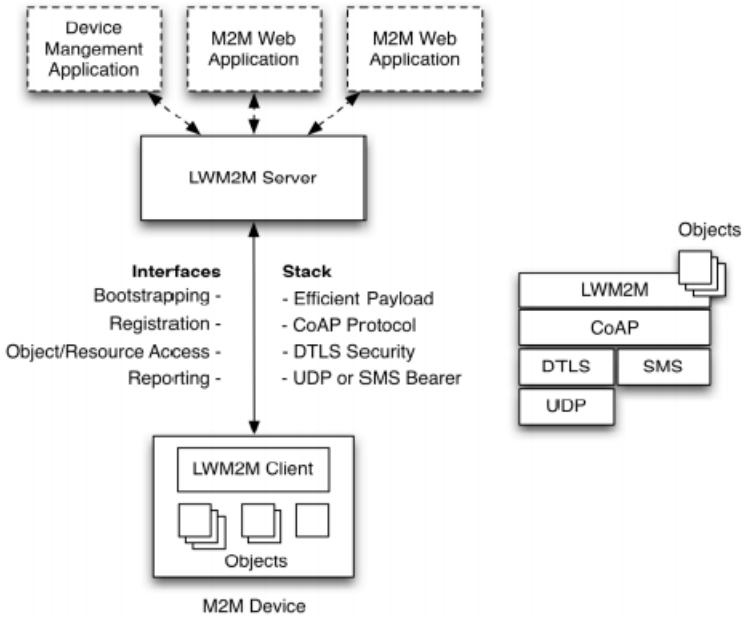


FIGURE 2.2: LwM2M architecture (adapted from [15])

| Category | Resource ID Range |
|---|---|
| Common Resources | 0 - 2047 |
| Reusable Resources | 2048 - 26240 |
| Private Resources | 26241 - 32768 |

TABLE 2.2: LwM2M Object Resources (ID Range)

to define it this way. And finally, the last range where anyone can use the object for private usage, although it is not possible to register them. Some of the predefined objects are made to monitor the voltage or current, to switch a button state or even to deactivate some components. For Instance one of the already defined objects corresponding to the voltage has the ID 3316, and the network endpoint is urn:oma:lwm2m:ext:3316. This indicates that the object should only be used with a voltmeter sensor to report measured voltage between two points. It may also provide information about the minimum and maximum values obtained or even the range of values the sensor can measure.

In our platform, we will need to introduce some new objects, but unlike this voltage object, our object will belong to the private resources category, this means that the ID will need to be between [26241; 32768] and this object may, for instance, define or monitor the IP or port of a server.

Considering all the objects already referred, it is deductible that there is a vastly wide range of objects and agents, ones with more humble resources than the others, so the necessity of having multiple LwM2M implementations/ solutions is a must have.

The LwM2M solved some of the problems with machine-to-machine communications with fragmentation of devices and systems. The capability of merging the configurations throughout systems in one "centralised", alongside other features already referred makes this protocol the best candidate (according to the specification) to handle the management API.

Although this protocol was probably the most suitable for our requirements there are some issues with it. Solving a problem using a solution that was overqualified

to the requirements, means that there are a set of functionalities that would never be used and in that case could in the future become a set of threats/vulnerabilities in the system. The lack of documentation or at least quality of it is also quite deterrent to develop an adequate management solution.

#### LwM2M implementations

We previously mentioned that there are multiple implementations of this protocol, therefore we decided to show some of the most updated repositories with LwM2M solutions (from [1]):

- **ARM mbed** - An **I**nternet of **T**hings (IoT) software platform developed in C++ that includes an operating system for constrained devices and uses LwM2M in its communication stack.
- **AVSystem Anjay** - A C library used for development of fully featured LwM2M clients.
- **Awa LightweightM2M** - An API for machine-to-machine communication developed in C that implements the LwM2M standard.
- **Eclipse Leshan** - A project maintained by the Eclipse Foundation. It features a Java implementation of the LwM2M standard that provides libraries for the development of M2M applications. It provides examples of fully-fledged LwM2M client and server implementations.
- **Eclipse Wakaama** - Another project maintained by Eclipse that features an implementation of the LwM2M standard and provides libraries written in C for the creation of LwM2M client and server M2M applications.
- **IoTerop IOWA** - A commercial LwM2M stack. Provides C libraries for the creation of LwM2M server and client-based applications.

Among these implementations there was one that excelled itself, the Eclipse Leshan. Eclipse Leshan is a Java implementation of LwM2M server and client, and it is the project with more features implemented from the LwM2M specification. Leshan provides libraries for people to develop their own LwM2M server and client, besides it also provides demonstration projects with examples of how to develop a LwM2M server, a bootstrap server or a client, as well as an example of the Leshan API providing a user interface for the server where is possible to perform operations in the clients. Obviously, this set of features are not reached by a standalone library but by a set of Java libraries including a HTTP server to expose the API and the Leshan library to allow the management operations between server and clients.

### 2.2.3 MQTT

In this section we introduced **M**essage **Q**ueuing **T**elemetry **T**ransport (MQTT), a lightweight messaging protocol focused on telemetry. The reason that led us to introduce this protocol is its set of characteristics like being optimized for high-latency or unreliable networks and used for monitoring devices which is one requirement for the managing solution in the IADS.

### MQTT

MQTT (described in ISO/IEC PRF 20922) is also a lightweight messaging protocol that allows constrained network devices with a simple way to communicate telemetry information. This protocol uses a publish/subscribe communication pattern and is used for machine-to-machine communication, playing an important role in the IoT.

MQTT is a good choice in networks with varying levels of latency due to unreliable connections and bandwidth constraints. If the subscribing client loses connection to the broker, the broker will buffer the messages and push them to the client when it becomes online again. If the publishing client loses connection to the broker, the broker will close the connection and send a message (**L**ast **W**ill and **T**estament (LWT)) to every subscribed client.

This protocol is called lightweight because of its fixed size header and its messages have a small code footprint. The fixed header with only 2 bytes allows this protocol to be easily used in constrained devices, although it also allows a message payload of 256 MB of information. This is a size that allows the protocol to be used in almost anything. MQTT has a Quality of Service(QoS) message system that allows setting levels of guaranteed message delivery. This means that there are levels 0, 1 and 2 that respectively means, the message is going to be delivered at most once, at least once or exactly once. In addition, we have the benefit that **T**ransmisson **C**ontrol **P**rotocol (TCP) allows TLS/**S**ecure **S**ockets **L**ayer (SSL) for security purposes, but it came disabled by default [16].

MQTT architecture consists of three components: publisher, subscriber and broker. Furthermore, the presence of the broker allows publishing nodes not knowing the address of other nodes, or even knowing if other nodes have all messages synchronised. This architecture works in the following manner (Figure 2.3):

1. An interested device registers as a subscriber for wanted topics. In this way it will be informed by the broker when the publishers create topics that are subscribed.

2. The publisher acts as a provisioner of data and next transmits the information to the interested subscribers through the broker.

3. Additionally, the broker checks authorisation of the publishers and the subscribers for enhanced security.

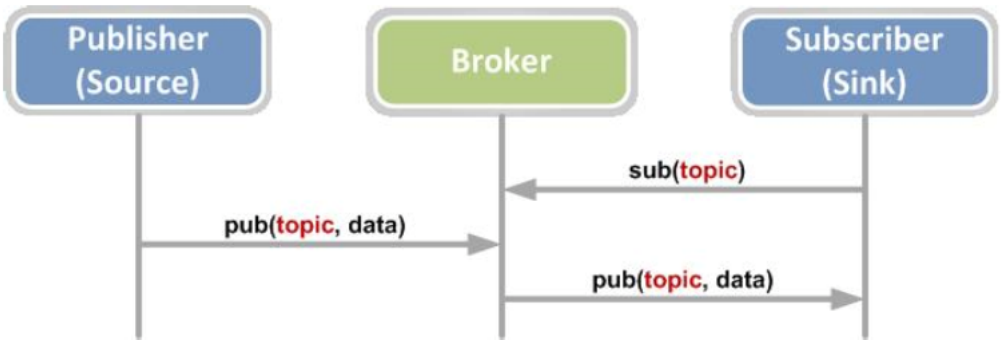


FIGURE 2.3: Publisher-Subscriber Model by [17]

However, there are a few issues associated to this protocol. When we have a small network of always-connected devices, the centralized broker and TCP have

their downsides, since the MQTT broker is overshot. A different case is when we have all the devices working properly but the broker fails. This causes the whole system to stop working since all the messages, subscriptions and publications need to go through the broker. Finally, all the complexity and mechanism for reliability in TCP like the handshakes and acknowledges drains power and is highly demanding for the network, not to mention the constraint devices it is usually used on [18].

However, these downsides are easily compensated with the reliability this protocol brings and, as a proof, we have multiple implementations of this protocol some with highly active communities.

#### MQTT implementations

Below are some of the available projects with active development:

1. **Adafruit IO** - An IoT software platform developed in Ruby on Rails and Node.js by Adafruit that includes MQTT and REST API's client implementation [19].
2. **M2Mqtt** - A C# library used for development of fully featured MQTT clients it is maintained by eclipse foundation [20].
3. **Machine Head** – A MQTT client implementation with the intuition to cover all MQTT features while maintaining a well-documented, well tested and properly design API unlike Paho Java's [21].
4. **Paho MQTT** – Probably the most complete MQTT client implementation containing implementations in multiple coding languages like C, Java, Javascript, Python, Lua, C++, Go, among others it is maintained by eclipse foundation [22].
5. **WolfMQTT** – Is written in C, and has built-in support for SSL/TLS it is targeted for IoT, embedded and RTOS environments [23].
6. **HiveMQ** – An enterprise MQTT broker with multiple plugins, connectors and great documentation [24]. It also has a set of articles with MQTT fundamentals a great MQTT initiation walkthrough [25].
7. **Mosquitto** – Eclipse Mosquitto is an open source MQTT message broker made with Java [26].
8. **RabbitMQ** - A polyglot broker capable of handling MQTT, **S**imple/Streaming **T**ext **O**riented **M**essaging **P**rotocol (STOMP) and **A**dvanced **M**essage **Q**ueuing **P**rotocol (AMQP) messages and subscriptions [27].
9. **GnatMQ** - A C# broker (lacks some functionalities like Sessions, retained and will messages) used for clients to publish messages and subscribe to topics [20].
10. **Moquette** – This implementation aims to be a MQTT compliant broker coded in Java [28].
11. **EMQ** - **E**rlang **M**QTT **B**roker (EMQ) is a distributed, massively scalable, highly extensible MQTT message broker written in Erlang/OTP. It is one of the few MQTT brokers allowing clustering [29].

#### Message Queuing Telemetry Transport for Sensor Networks (MQTT-SN)

MQTT-SN (described in [30]), unlike MQTT, is built over Sensor Networks. Since this protocol is not build to use IP, it is not suitable for our purpose. Although it is possible to make it communicate via UDP using an MQTT-SN serial to UDP bridge and **R**eally **S**mall **M**essage **B**roker (RSMB) as a broker (has native support for MQTT-SN with UDP), or even to connect to an MQTT broker [31].

#### MQTT in configuration management and monitoring

Although we already referred MQTT in this section, we only mentioned it as a messaging/telemetry protocol, but the protocol characteristics have much more potential and there are multiple MQTT usages as a management solution.

#### LwM2M over MQTT

There are other promising developments in configuration management using MQTT, but there is one protocol where we can see multiple entities working independently, which means there is potential and therefore interest from diverse companies.

This project is designated LwM2M over MQTT and there is a Profile Specification [32] developed by **E**mbedded **S**pecification **R**equest (ESR) Consortium. In this specification we can find the details on how to implement a server or embedded client that complies to the LwM2M-MQTT specification. In this protocol CoAP messages are transferred as payload in MQTT topics, making MQTT just a high-level transport layer for the CoAP messages.

There are a LwM2M client and a server connected through two MQTT topics, named transport topics, used for bi-directional (one topic for upstream and other for downstream) communications. Each pair of transport topics connect to only a single LwM2M client and a single LwM2M server. LwM2M specification defines that its clients have the possibility to connect to multiple servers. In this protocol, this is achieved by using the same MQTT broker connection but with multiple distinct transport topics pairs (Figure 2.4). To ensure these topics are properly distributed we need to follow some conventions using the client id and the direction (upload or download) of the data transfer as the topic name.

Since this protocol consists of the merge of two protocols both with **Q**uality **o**f **S**ervice (QoS) it is defined that MQTT should have its mechanisms disabled, letting this features to the LwM2M server and client. Among other features, the usage of MQTT instead of CoAP as a transport layer allows for instance to bypass the problem of having a firewall in the connection since both client and server subscribe to a broker.

#### LwM2M over MQTT implementations

As previously mentioned the LwM2M over MQTT protocol is being used by multiple entities. There is a company that uses this specification (**Microej**) and they have developed a solution that implements this protocol but it is closed source. Eclipse also has a connector of MQTT to LwM2M in its *eclipse kura* an IoT router, EMQ broker [29] has a similar function in its broker, essentially its a connector between this two protocols that allow the development of similar functionalities.

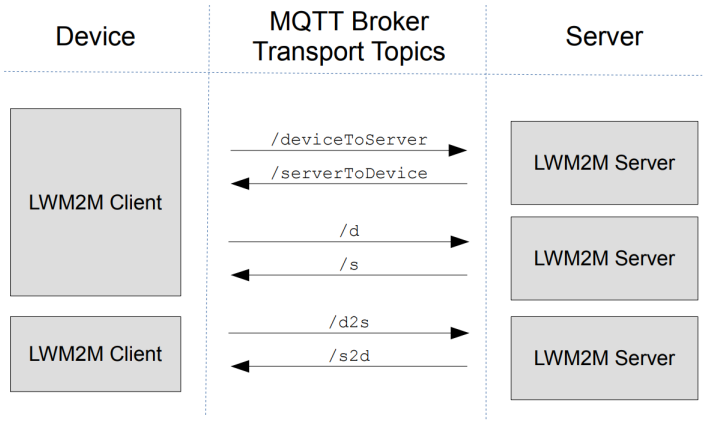


FIGURE 2.4: LwM2M over MQTT Overview (by [32])

Finally, there was an attempt project [33] developed by an IBM employee to implement a LwM2M server prototype using MQTT as the transport layer, but it lacked multiple LwM2M features including bootstrap. This project was presented in IoT/LwM2M MQTT Binding/Meeting (Apr 2, 2015 conference call) as a proof of concept in a demo using a Raspberry Pi device to the Leshan server via MQTT protocol and reporting CPU temperature.

From all these projects the only companies that still maintain their projects are the Microej (since it software requiring licensing), Eclipse Foundation (responsible for multiple IoT projects) and EMQ. In summary there are few open source solutions using this protocol, the documentation is not the best and consequently, the support from communities and companies is also somehow poor.

#### An implementation of MQTT as configuration management protocol

In [34], the authors used MQTT to create an auto-configuration and Management device, created a testbed using an Arduino device which acts as a MQTT client. Since MQTT does not have discoverability capabilities, they used **D**evices **P**rofile for **W**eb **S**ervices (DPWS) a minimal set of implementation constraints that enables discoverability and eventing among other features on resource-constrained devices which sends advertisement **S**imple **O**bject **A**ccess **P**rotocol (SOAP)-over-UDP based messages to enable the auto-discovery. However, since DPWS uses UDP they are just adding more overhead to the TCP overhead of MQTT.

### 2.2.4 TR-069

**T**echnical **R**eport **069** (TR-069) makes part of the Broadband Forum's **C**PE **W**an **M**anagement **P**rotocol (CWMP) suite which includes other technical documentation such as TR-106, TR-181 and TR-157, it describes respectively describe the generic data model template, the TR-069 device data model and specific component objects for devices. This protocol works as a bidirectional SOAP/HTTP-based protocol, providing the communication between a **C**ustomer **P**remises **E**quipment (CPE) and an **A**uto **C**onfiguration **S**ervers (ACS).

This protocol ensures a secure connection between a CPE and an ACS by encrypting its TCP connection with SSL. One of the main features of this protocol is the role switching over the same persistent connection granted by the use of a **R**emote **P**rocedure **C**all (RPC)-based paradigm, which allows client and server to

switch roles between each other, in this way the protocol can bypass connections using firewalls.

This protocol has four main steps in its session (Figure 2.5): Connection establishment, authentication, CWMP operations and session termination. After the connection established and authenticated the server can perform operations over its clients without any firewall restriction problem.

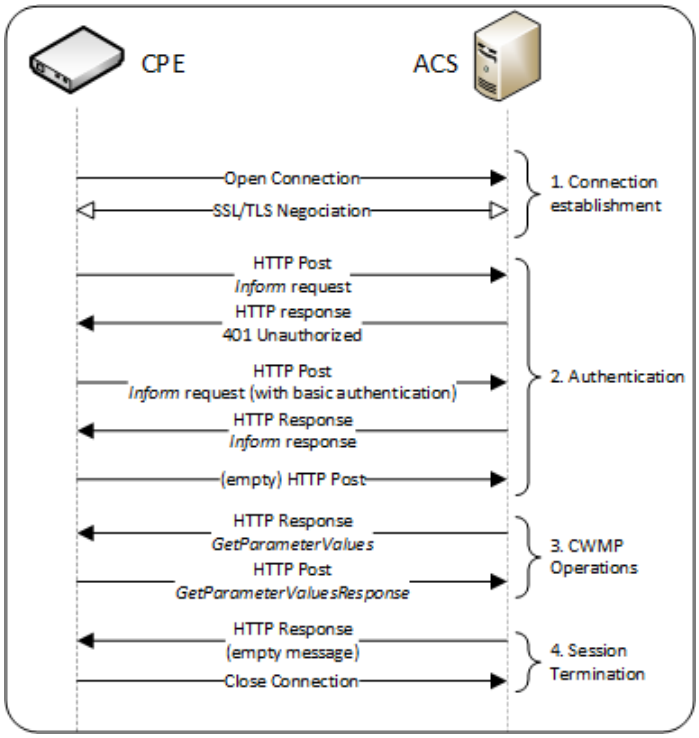


FIGURE 2.5: Example CWMP/TR-069 Session (by [35])

### TR-069 implementations

In the last years were developed multiple solutions for TR-069 most of them were discontinued, but there are some active open source solutions including every feature described in the TR-069 specification. These are some of the projects with TR-069 solutions:

- **Modus** - A Java generic client for TR-069 designed by France Telecom, available under the Apache License V2.0 and is discontinued since 2011 [36].
- **OpenACS** - An automatic configuration server that start as an open source effort in 2008, available under the GPL v2 license. It was one of the most extensive implementations of TR-069. The last update occurred on Aug 08, 2017 [36].
- **LibreACS** - A fork of OpenACS which emerged in 2014 when the project owner removed all releases and source code from the project page [37].
- **Perl CWMP** - A command-line automatic configuration server written in Perl, it contains some quirks or violations of the protocol specification. The Last update occurred on Oct 10, 2014 [38].

- **Grail ACS** - An automatic configuration server and TR-069 client emulator written in groovy by an employee of Telenor in Denmark. The only update on this project occurred on May 17, 2010 [39].
- **Open TR069** - A client coded in C and licensed under GNU Lesser GPL v2. This project was supported by the networking solutions company Aximcom and is discontinued since 2011 [40].
- **CWMP client** - A client developed at the National Technical University of Athens, written in C as part of a European project MUSE and is discontinued since 2011 [41].
- **TR-069 D-Link** - An open source automatic configuration server developed by D-Link and written in C. The last update occurred on Aug 07, 2012 [42].
- **GenieACS** - An automatic configuration server focused on performance by means of concurrency and modern technologies(Node.js and MongoDB). It supports aliasing parameters which allow achieving multiple vendor configurations who changes default parameters [43].
- **EasyCWMP** - An open source client developed by PIVA Software, a company specialised in embedded software development. It is licensed under GPLv2 and was tested with OpenWRT and Ubuntu [44].
- **FreeACS** - According to its author it is the most complete open source automatic configuration server, with web-interface, web-services interface, TR-111 support. It is written in Java and still in active development [45].

### 2.2.5 NETCONF

NETCONF [46] is an IETF standard that had its first specification release in 2006. This protocol was created as an alternative to both Command Line Interfaces, which allows automating the tasks of monitoring and remote management equipment using languages such as Perl or Unix tool expect, or even to SNMP which as stated before was becoming less used due to its security flaws. NETCONF enables mechanisms in network devices, providing the ability to install, manipulate and delete configurations. Also, this protocol can be enhanced with new capabilities/operations. This protocol uses an RPC-based operation paradigm and supports several mechanisms, such as **S**ecure **Sh**ell (SSH), TLS or SOAP and using E**X**tensible **M**arkup **L**anguage (XML) based data encoding for the configuration data and the protocol messages.

#### NETCONF implementations

These are some of the repositories/projects implementing NETCONF specification as a solution:

- **Go-netconf** - A library with a simple NETCONF client, written Juniper Networks in Go language, under BSD license. The last update occurred on Apr 26, 2018 [47].
- **Netconf4Android** - This library has an API with that allows building client applications, available under BSD license. The last update occurred on Sep 11, 2012 [48].

- **Libnetconf2** - A library written in C intended for building both clients and servers. It provides mechanisms to connect NETCONF client and server among them via SSH/TLS. The last update occurred on Jun 05, 2018 [49].
- **Netopeer2** - A repository that includes both server and client, based on libnetconf2. The client includes a command-line interface allowing simple requests to be manually generated. The last update occurred on Jun 08, 2018 [49].
- **Tail-f Systems** - Has a server, command line and web interfaces, SNMP agent, REST interface and integrated configuration store. It also has an open source implementation of a client library. The last update occurred on Apr 24, 2017 [50].

### 2.2.6 Management Protocols Wrap-up

From the analysed protocols we can willingly say that TR-069, NETCONF and SNMP are automatically excluded from our final choice since they do not fit in our requirements. The management solution needs low overhead and a protocol/tool that fit our requirements, avoiding overqualified solutions.

This left us with two possible solutions CoAP/LwM2M and MQTT. MQTT is the most similar protocol to CoAP, when searching for features. It is a messaging protocol based on a publish-subscribe pattern. Opposite to CoAP, MQTT it is built over TCP gaining some advantages over UDP when it comes to reliability, being suitable not only for unreliable constrained devices or low bandwidth connections but also for networks with NAT-enabled routers in which CoAP/LwM2M has trouble operating. In Table 2.3 we tried to show summarise the main characteristics of these protocols explained in this subsection. However, our final choice is explained in Section 4.2.

| | **MQTT** | **CoAP/LwM2M** |
|---|---|---|
| **Abstraction** | Pub/Sub | Request/Reply |
| **Architecture Style** | Brokered | P2P (Server-Client) |
| **QoS** | 3 levels | Confirmable or nonconfirmable messages |
| **Interoperability** | Partial | Yes |
| **Performance** | Typically 100s to 1000+ messages per second per broker | Typically 100s of requests per second |
| **Transport** | TCP | UDP |
| **Subscription Control** | Topics with hierarchical matching | Provides support for Multicast addressing |
| **Data Serialization** | Undefined | Configurable |
| **Standards** | Proposed OASIS MQTT standard M | Proposed IETF CoAP standard |
| **Dynamic Discovery** | No | Yes |
| **Mobile devices (Android, iOS)** | Yes | Via HTTP proxy |
| **Security** | Simple Username/ Password Authentication, SSL for data encryption | DTLS |

TABLE 2.3: Summary of Key Comparison Criteria (adapted from [51])

## 2.3 Protection of SCADA Systems

Although the main purpose of this thesis is not SCADA systems, the agents and management solution were designed to be used in SCADA systems, therefore, this section explains the evolution of SCADA systems, the vulnerabilities and recent security threats.

### 2.3.1 Evolution

SCADA systems started being developed even before networks creation (in the 1960s). They were developed in the United States of America and in the primordial times, it was just a "mainframe" isolated from other systems this was the first generation of SCADA [52].

At that time communication with **R**emote **T**erminal **U**nit (RTU)s was performed in most of the cases by proprietary protocols (although there were already some Wide-area Networks). Management difficulty and task costs at that time was proportional to the size of the infrastructure to supervise, since it was generally just checking if the components were connected/running or not. This generation consisted of a Master station or client and a RTU. the RTU was used to convert the sensor state to a digital format and communicate it over the network. the Master station was used to control the RTUs or even **P**rogrammable **L**ogic **C**ontroller (PLC)s using an **H**uman **M**achine **I**nterface (HMI) to monitor the components states.

The first two versions (just like the previously mentioned SNMP protocol in Section 2.2.1 had no security, but on the third version (Figure 2.6) was introduced a feature that allowed to divide components across physical areas, uncompromising the whole network infrastructure. This version introduced some new components such as a Database and Field devices, these field devices are probably the most important in the perspective of monitoring since they interact directly with the process and can change some aspects in the process accordingly to the read values. Due to the proprietary protocols used SCADA environments was already filled with some vulnerabilities, but when the IP was introduced it brought, even more, vulnerabilities to the existent architecture [53].

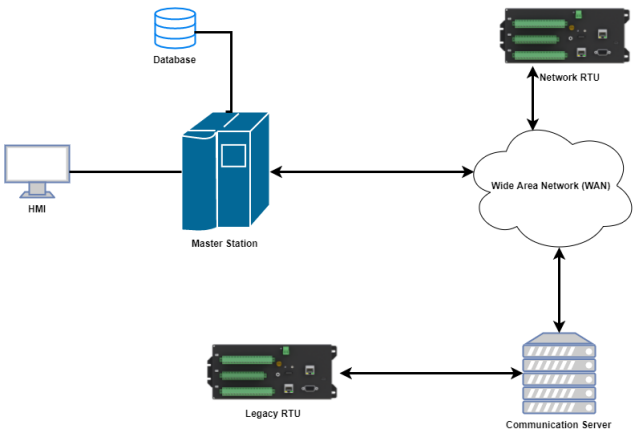


FIGURE 2.6: SCADA architecture: Third generation (adapted from [54])

One important aspect of the vulnerabilities in the project, this thesis focus on, is that they need to be detected but not prevented. In other words, it is a detection

system, not a prevention system, so the important part here is related to monitoring and controlling.

The fourth version is designated as IoT-cloud-based SCADA system [55] and brought some important aspects to the SCADA systems, easy maintenance and integration, increased data accessibility, flexibility, cost efficiency, optimisation and scalability. But brought also a big question, SCADA environments are yet to become secure, adding this kind of technology(IoT) (which in most of the cases doesn't have security). Will IoT-cloud-based SCADA system bring more security threats to the SCADA environment? Are we going to have more benefits than problems?

### 2.3.2 Vulnerabilities

Another interesting fact about vulnerabilities in SCADA environments is that before 2010 the systems seemed almost 100% secure, but the reality is that no one was paying enough attention or doing research in that area and then everyone knows what happened, Stuxnet (Figure 2.7) was detected and with that came both an increase of vulnerabilities detected but also the number of patches/ fixes and updates.

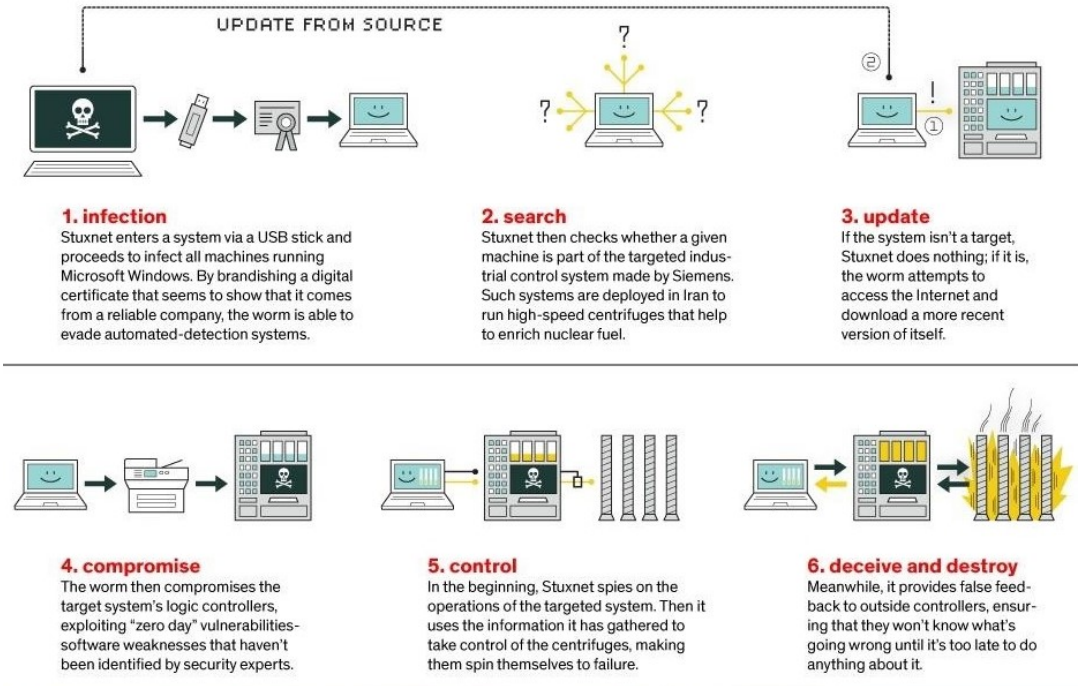


FIGURE 2.7: How Stuxnet happened (by [56])

Accordingly to Positive Technologies report [57] from 2010 to 2012, there was an increase with a factor of almost 10 times in the number of detected vulnerabilities (increased from 11 to 98). But this is not the only interesting fact, in the first ten months of 2012 were found more vulnerabilities than in the entire preceding five years, 65% of this security holes were considered critical and finally, 40% of the SCADA systems available via the internet were vulnerable.

Now, 5 years later what are the changes made?, Some companies like Siemens (the most affected by Stuxnet) created teams that focused on correcting and detecting vulnerabilities. The Positive Technologies states that most of the companies fixed almost all the vulnerabilities reported, and that makes us think the problem is solved, but Duqu, Flame [58] and Gauss [59], proved that even with all the fixes made at that

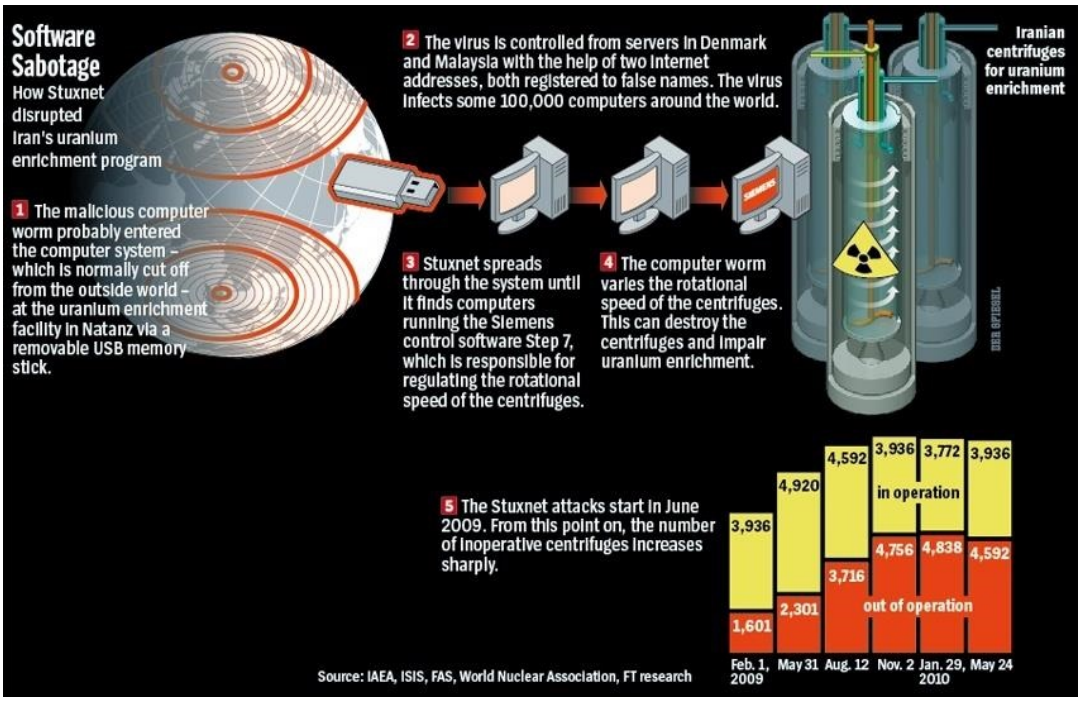


FIGURE 2.8: What happened in Stuxnet (by [56])

time the systems still vulnerable. Both Duqu and Stuxnet were made with the intention to destroy the systems they infected (Figure 2.8), but Flame and Gauss have a different purpose collect data, Crysys report defined Flame as the most advanced malware ever built capable of:

- activating microphones and webcam
- logging keystrokes
- taking screenshots
- extracting geolocation
- sending and receiving commands and data through Bluetooth

When it comes to Gauss(detected by Kaspersky Lab) it had a similar behaviour but instead of masquerading as a fake windows proxy for updates it would inject modules into different browsers, in this way it would be capable of:

- intercepting user sessions and steal passwords, cookies and browser history
- collecting information about the computer's network connections, processes and folders
- collecting information about **B**asic **I**nput/**O**utput **S**ystem (BIOS), **c**omplementary **m**etal-**o**xide-**s**emiconductor (CMOS) and **R**andom **A**ccess **M**emory (RAM)
- collecting information about local, network and removable drives
- infecting USB drives with a spy module to steal information from other computers

- interacting with a command and control server, sending the information collected and downloading additional modules if necessary

It is the year of 2017, last month of the year when suddenly FireEye reports that one new malware was detected, it is called *TRITON* (Figure 2.9) [60], and just like Duqu and Stuxnet, it seems that the attack was made with a destructive purpose being capable of doing three kinds of attacks described below.

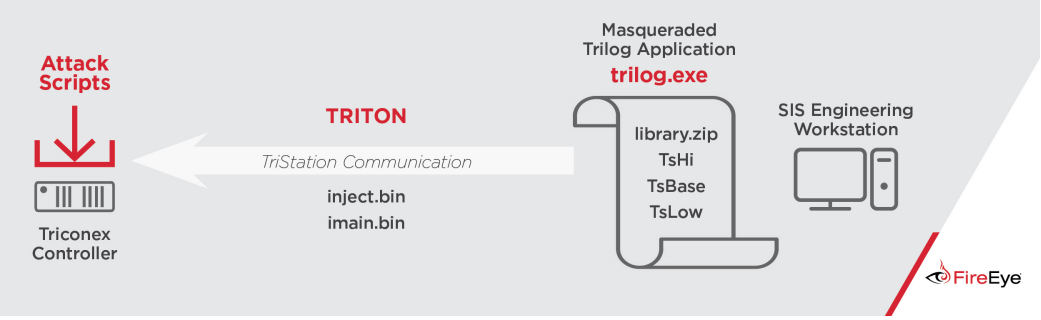


FIGURE 2.9: TRITON Architecture and Attack Scenario (adapted from [60])

Shutting down the process using the Triconex **S**afety **I**nstrumented **S**ystem (SIS), allow the attacker to reprogram the SIS logic, that makes it trip and shut down a process that is in a safe state, creating a false positive. This could create financial losses, caused by the start-up procedure after the Shutdown, and downtime. Controllers reprogramming the Triconex SIS to allow a unsafe state. By that, the attacker can reprogram the SIS logic to let the unsafe condition to persist. It increases the risk of a hazardous situation and will cause physical consequences due to loss of SIS functionality. Or using it once again in a **D**istributed **C**ontrol **S**ystem) (DCS) to create a unsafe state or hazard here the attacker while preventing the SIS from functioning correctly, the attacker can change the process to a unsafe state, that has an impact in human Safety, environmental consequences, or eventually damaging the equipment. All of that will depend on the physical constraints of the process, and the design of the plant.

### 2.3.3 Types of vulnerabilities

Positive Technologies report has divided the vulnerabilities into some subcategories, SCADA, HMI, PLC, Hardware, Software and Interface/Protocol. Within this categories were found 175 vulnerabilities (Figure 2.4), from which only 8 were related with Software and Interface/Protocol meaning, the other 95% were found in the first 4 categories, actually, almost 50% of the vulnerabilities were found in SCADA.

Kelly Jackson in *"SCADA Security In A Post-Stuxnet World"* [61] stated Siemens and other vendors made quite a big evolution, Siemens even created a department ProductCERT to change their reactive behaviour to a proactive doing research on exploitation and security-related aspects. Most of the improvements were made on server side security controls, but on the PLC side, almost nothing was done.

When it comes to the software products we have even more hideous results since over 35% of the vulnerabilities had exploits, from each 50% allowed remote code execution. From these vulnerabilities, more than 40% were considered critical.

For instance, Data Integrity attacks in SCADA environments can lead to manipulation of the sensor data or manipulate the control signals this could easily

| System Type | Number of Vulnerability |
|---|---|
| **SCADA** | 87 |
| **HMI** | 49 |
| **PLC** | 20 |
| **Hardware** | 11 |
| **Software** | 7 |
| **Interface/Protocol** | 1 |

TABLE 2.4: Number of Vulnerabilities in Industrial Control Systems Components by [57]

| Vulnerability Type | Vulnerability Percentage |
|---|---|
| **Buffer Overflow** | 36 |
| **Remote Code Execution Web** | 13 |
| **Web (client-side)** | 9 |
| **Web (server-side)** | 11 |
| **Local Privilege Escalation** | 2 |
| **Authentication / Key Management** | 23 |
| **DoS/Data Integrity** | 6 |

TABLE 2.5: Classification of Vulnerabilities in Industrial Control Systems According to Type by [57]

lead to the destruction or deterioration of an industrial process. In this experiment [62] the attacker modified the frequency and tie-line flow measurements on an AGC(Automatic Generation Control) Control System in order to dissimulate the operator in thinking there was a load increase or decrease, making the operator perform the wrong action overloading the systems.

What would happen in a specific case would be that the operator would assume an increase in demand, and to fix this demand would increase the generation with the respective lacking amount. However when performing this action, not forgetting that there was nothing wrong with the system and it was load balanced, the increase in the generation would put the system in an abnormal operating state disturbing the normal balance of the system.

Just like the previously mentioned attack, a Buffer Overflow [63] can also lead to a manipulation of an industrial process by stopping the components with a Denial of service, in this way the component won't be able to transmit information about that specific process, another way is to make the overflow so extreme that will lead to a reboot of that device which is really bad in a SCADA environment since most of the components are intended to be working without ever rebooting.

But there is an aspect quite interesting in this attacks, we can be monitoring the process and everything would seem to be working in a normal behaviour [62], but we go further and take the monitoring to what happens in the network we can see this attacks happening, and consequently we can prevent them from happening and control the defect components/devices and parts of the infrastructure.

The Future of SCADA vulnerabilities [55] is dependent on three aspects, first the type of vulnerabilities that are prominent to appear, secondly, the challenges to secure this new generation of SCADA systems based on IoT and the advances that are being made to stop these threats, for instance in monitoring and control of this network environments.

Some literature suggests that the threats to the fourth generation SCADA systems are the following:

- **Advanced Persistent Threats**: these attacks are attempts by people trying to steal data using zero-day attacks to gain access, these attacks aren't meant to damage the infrastructures [64]
- **Lack of Data Integrity**: can happen when there is physical tampering or interception
- **Replay Attacks**: as the name describes it consists of reproducing a valid message with data over and over again, this will lead to performance loss in the SCADA system [65]
- **Denial of service**: these attacks tend to overload the device resources in a way that makes it unresponsive or unavailable [66]
- **Man in the middle**: an example of these attacks is the sniffing and spoofing attacks. In the sniffing, the intruder monitors the request in the network without permission. In the spoofing, the person dissimulate to be another entity to gain privileges

### 2.3.4 Efforts to prevent attacks to SCADA systems

To prevent these threats there are efforts being made in both SCADA and SCADA systems in IoT-Cloud Environment when it comes to general SCADA security, in general, there are 3 main aspects: Policy management, Data Integrity and Weak Communication.

#### General security

Policy management is related to Internet originated attacks in general and the lack of roles or passwords in SCADA environments [55], and if this was not a big problem when SCADA was created, it became a huge one with the addition of internet support. The author states that the software becomes more secure with proper authorisation and authentication.

When it comes to Data Integrity [66] described some models to investigate attacks, see what were the effects of these attacks and creating mitigation plans. Another author suggested and evaluated that the algorithms used in these systems need to be attack-resilient to survive in case of attack [67].

Finally, Weak Communication is related to the use of protocols without integrity, confidentiality and authentication like Modbus. There were also solutions based on IP-Sec and TLS/SSL that are capable of securing the IP traffic and avoid Denial of service attacks, but there is a downside since IP-Sec is less scalable and too complex. The TLS/SSL solution has the drawbacks of using a solution with already known vulnerabilities, but still better than no security at all.

#### IoT-Cloud environment related security

There are some similarities in the areas that are being analysed in this generation of SCADA compared to the general SCADA efforts, one example of that is Data Integrity and Privacy, authentication and encryption. But most of the concerns in this field are not really related with SCADA but with IoT itself.

For instance, there is the concern about Data Logging because unlike general SCADA IoT-based needs a solution to aggregate the track of all cloud-based system logs which is not a simple task.

When using Cloud services we are always susceptible to their license and even when we agree with this contract we are always dependent on them and really own the data.

And there are even more concerns, but these are the ones that we probably need to focus when comes to our concerns with monitoring and managing the environments.

## 2.4 Security Probes

The reason we introduce security probes is to better understand what functionalities and how we should design the management system, since it will work with this components. Therefore, to provide a better contextualization of security probe, we define probes.

Defining the term probe is a complex task even when reduced only to the context of computer networking. We can define probe as an act of collecting information. This information can be about the state of a control process, network, server, device or even a service. An example of this information sampling is to use the system call ping as an ICMP packet allowing to discover part of the network topology and with luck the whole topology, another usage is to gather information that will permit exploiting the device [68].

Although there are more types of security probes, we will only mention some. One of the reason is that, eventually, we will need to create new ones. The following three probes will be used in the IADS:

- **The Shadow RTU** which is meant to be installed at a specific network juncture allowing it to monitor and collect data from a particular process like the communication between a client and the PLC or RTU.
- **Honeypot** that is an easily exploitable target to mislead attackers from attacking the actual infrastructure.
- **NIDS** which is meant to monitor and detect attacks on the network platform.

Probing methodologies provide statistics related to the system functionality. On the network functionality, we have, for instance, traffic analysis and protocols used. These agents can also notify the system administrator whenever there is a suspicious behaviour on some device so that we can check the last anomalous log and verify if it was a false positive or a real threat. This kind of notifications and alerts make viable to manage and monitor the network, ensuring the performance and safety of all the components without having the necessity to be constantly verifying every component and machine, which would be unsustainable.

With this in mind, the proposed agents need to have some particularities, starting with their functionalities and the kind of data they are supposed to evaluate and generate. Having the previously mentioned aspects defined allows the creation of efficient agents, capable of monitoring and detecting some kind of attacks like denial of service, spoofing and others, that SCADA environments might be susceptible to.

### 2.4.1 Examples of Security Probes

#### Honeypot

As mentioned before, honeypots(Figure 2.10) are designed and developed to be easy targets for attackers. But they have a particularity, they are just a decoy to avoid the attacker to compromise the real components of the industrial controlling system, making real SCADA devices protected. In this way, besides having an attack prevention, since the attacker is misled to attack the honeypot, we also have the privilege of monitoring all the traffic and attack attempts from this attack, which can be analysed preventing future attacks that the honeypot is not capable of intercept.

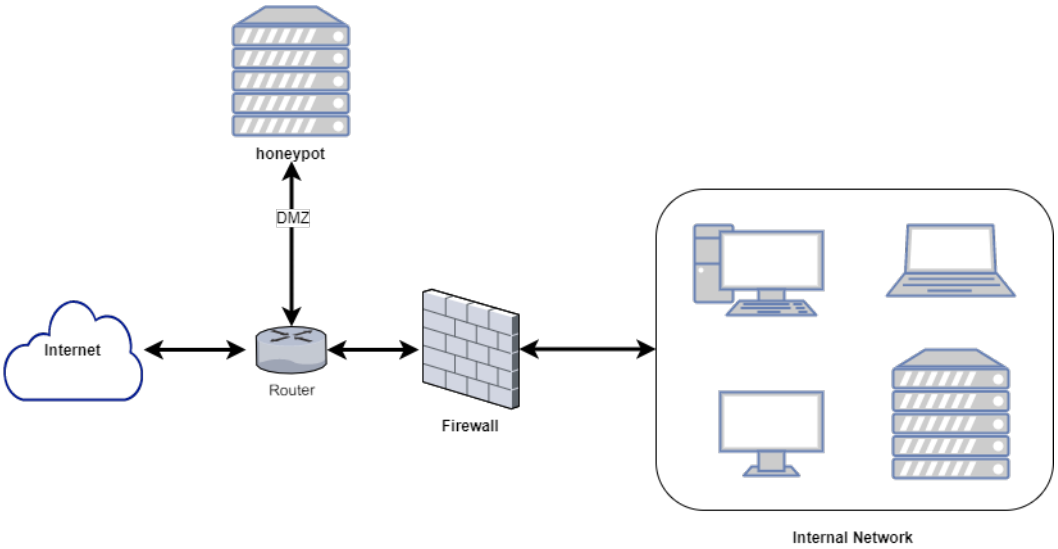


FIGURE 2.10: Honeypot use case (adapted from [69])

This kind of probes can be divided into two types, research and production honeypots. The production honeypots are typically low interaction meaning they are regularly ready-to-deploy solutions. These solutions are meant to be placed in a production environment but will generate less information about attacks than the research honeypots.

The research honeypots are less focused on protecting components or the network structure, instead, they are designed to understand what is the attackers' behaviour when making an attack, these honeypots tend to be more complex to maintain that the production ones. Honeypots, in general, can also be classified accordingly to its functionalities [70]:

- **Server honeypots** are active security devices that simulate being a component or infrastructure to avoid attackers from damaging the real infrastructure.
- **Client honeypots** are, just like server honeypots, active security devices but unlike the server, they are not traps for attackers, instead, they act as a client to interact with the server to analyse whether an attack occurred or not.

#### IDS

An **I**ntrusion **D**etection **S**ystem (IDS) can be defined as a device or software that monitors a network/system for malicious threats. Any detected activity will be either collected in a centralised SIEM system or notify the administrator. In the

case the collected information is sent to the SIEM this will be analysed and filtered by other external components, not directly related to the IDS, such as a cluster-computing framework [71].

There are a wide variety of IDS, some of them analyse the network traffic for a whole network topology others are just antivirus software that analyses a specific device looking for malicious software (browser hijackers, ransomware, keyloggers, backdoors, rootkits, trojan horses, worms, dialers, adware, and spyware) [72].

There are some specific IDS branches that are commonly used to classify the software such as NIDS and **H**ost-based **I**ntrusion **D**etection **S**ystem (HIDS). A HIDS is a system that monitors relevant operating system files and a NIDS is a system that analyses network traffic. Besides, we can also classify the systems by detection approach, having signature-based (that recognises known patterns, like malware) and anomaly-based in this case the traffic is analysed and we determine deviations from the regular traffic, generally recurring to data mining and machine learning algorithms.

Some IDS go further in the detection and can prevent the threat from happening, systems with this kind of response capabilities are referred to as an intrusion prevention system (IPS).

**(v)NIDS -** A network intrusion detection system (Figure 2.11) is a component placed strategically in a specific network point to monitor the traffic at that point. As previously mentioned this system can use pattern-based detection to identify unexpected behaviours and activities like denial-of-service, network scans or even man in the middle. There is also the possibility, for some of these systems, to prevent attacks whenever they are detected. In this situation, we have an intrusion prevention system. Finally, this component has the possibility to be virtualized giving us the opportunity to easily deploy multiple systems along the network topology.

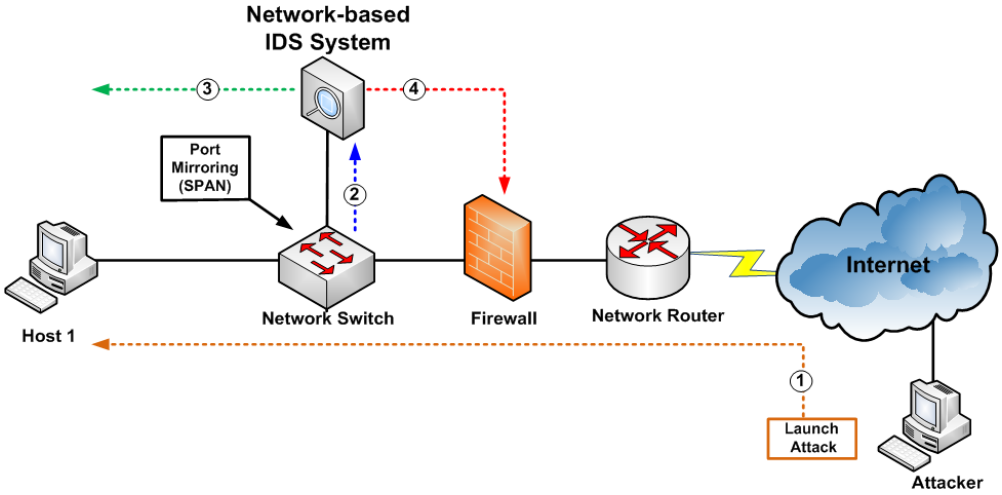


FIGURE 2.11: NIDS architecture (by [73])

**HIDS -** The host intrusion detection system is generally used to detect the same kind of threats the NIDS detects but at a host level. In most of the cases, a HIDS (Figure 2.12) is not only able to analyse the network traffic but also monitor important operating system files detecting changes and even avoiding them. One of the main advantages of this system is it being used simultaneously with a NIDS and if the

NIDS fails at detecting an anomaly or threat, there is still a chance the HIDS detects it.

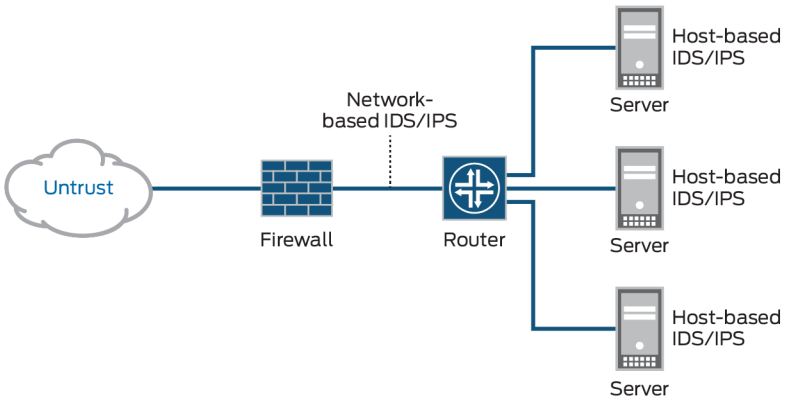


FIGURE 2.12: HIDS architecture (by [74])

**Shadow RTU**

A Shadow security unity is a device deployed in parallel with a PLC or remote terminal unit, being capable of intercepting its communications control channels and physical **I**nput/**O**utput (I/O) to continuously check and verify if there is any threat such as unauthorised accesses, program tampering, Denial-of-Service or even flooding [75]. This kind of devices does not need any major network changes in the existing control network and generally, this type of solutions are low-cost, which makes it an adequate to use in SCADA systems [76].

## 2.5 State of Art - a brief review

In this chapter were introduced the concepts and fundamentals of network management and monitoring, after, it was explained the evolution of the most used network management protocol SNMP and explained what were advantages and disadvantages of using it, with the previous information we could delineate what were the characteristics necessary to use in an IACS, this leads us to the comparison of some protocols mostly used in IoT, CoAP, MQTT and MQTT-SN, and some configuration management protocols like LwM2M, TR-069, NETCONF and some MQTT variations. Besides we also introduced some tools using these configuration protocols. After that it was shown an evolution of SCADA systems, to understand why it is necessary a new secure protocol to manage devices in this system, and what were the threats that appeared for not using a secure protocol. Then we introduced the concept of probe and gave some examples of probes since our work will focus on these agents.

# Chapter 3

# System Overview

In this chapter we started by introducing the ATENA project architecture. After, we understood this architecture we developed an architecture to be integrated within the ATENA architecture, allowing us to define the requirements for the management architecture. Finally, after deciding the architecture and requirements we explained what technologies we have used to implement the solution.

## 3.1 Reference Architecture

Within the ATENA project, there are multiple international partners each one with a specific set of objectives and tasks. This set of tasks can be visualised in Figure 3.1.

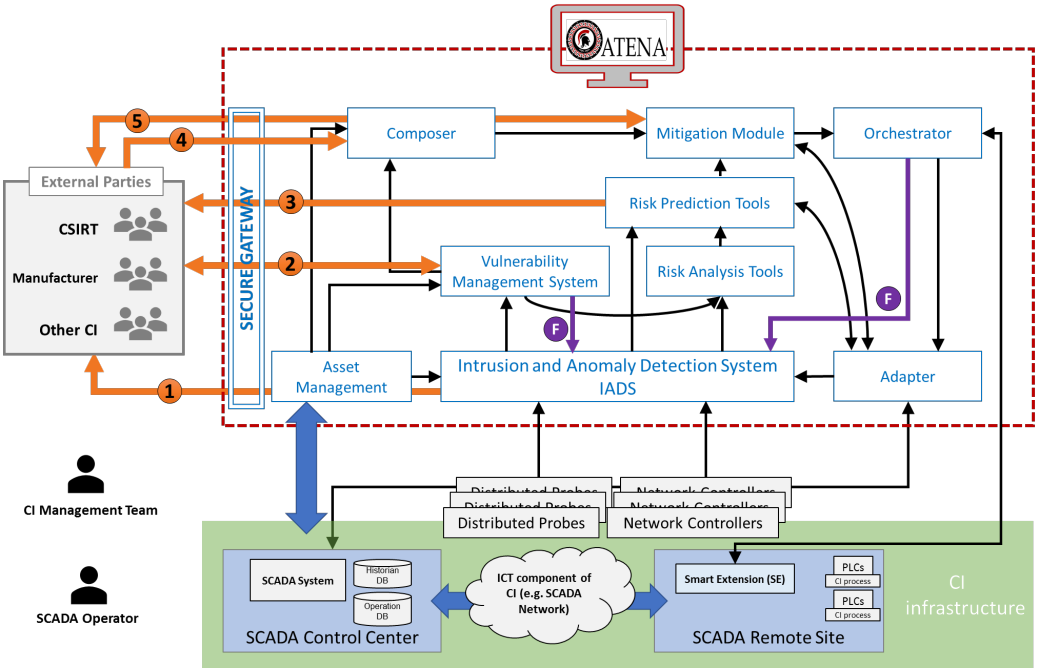


FIGURE 3.1: Incident and vulnerability management with ATENA modules (by [1])

LCT, as one of the ATENA project partners, is responsible for the intrusion and anomaly detection system, consequently, there is a set of tasks assigned. This set of tasks is represented in Figure 3.2 as the architecture of the IADS, where we have multiple components with the objectives of collecting, aggregating, filtering and analysing data in order to detect anomalous behaviour in the monitored IACS. In addition, we also need a way to manage these components in the infrastructure.

Of course, these tasks are not as simple as the actions described, for instance, there are multiple components only to analyse the data with multiple mechanisms to filter, aggregate (and replicate) and process the data using the analytics platform ( the information from the security probes is fed into the analytics platform, built on a lambda architecture template containing both online stream processing - fast path related with analysis of near real time stream of events consumed directly from the distributed streaming platform topics while implementing a set of Complex Event Processing mechanisms tighten to a small time-window - and batch processing - slow path related with machine-learning and anomaly detection mechanisms, typically slow and computational expensive algorithms - capabilities).

In this particular set of tasks, our work consists of the development of the platform management solution (a server component for managing and registering the agents), part of the monitoring (exposing an API) and the data collection part (and the necessary mechanisms for the encoding and transmission of events). The developed agents will work as wrappers to other software, in this way both the management/monitoring and the data collecting solution are generic and mutable for future usage.

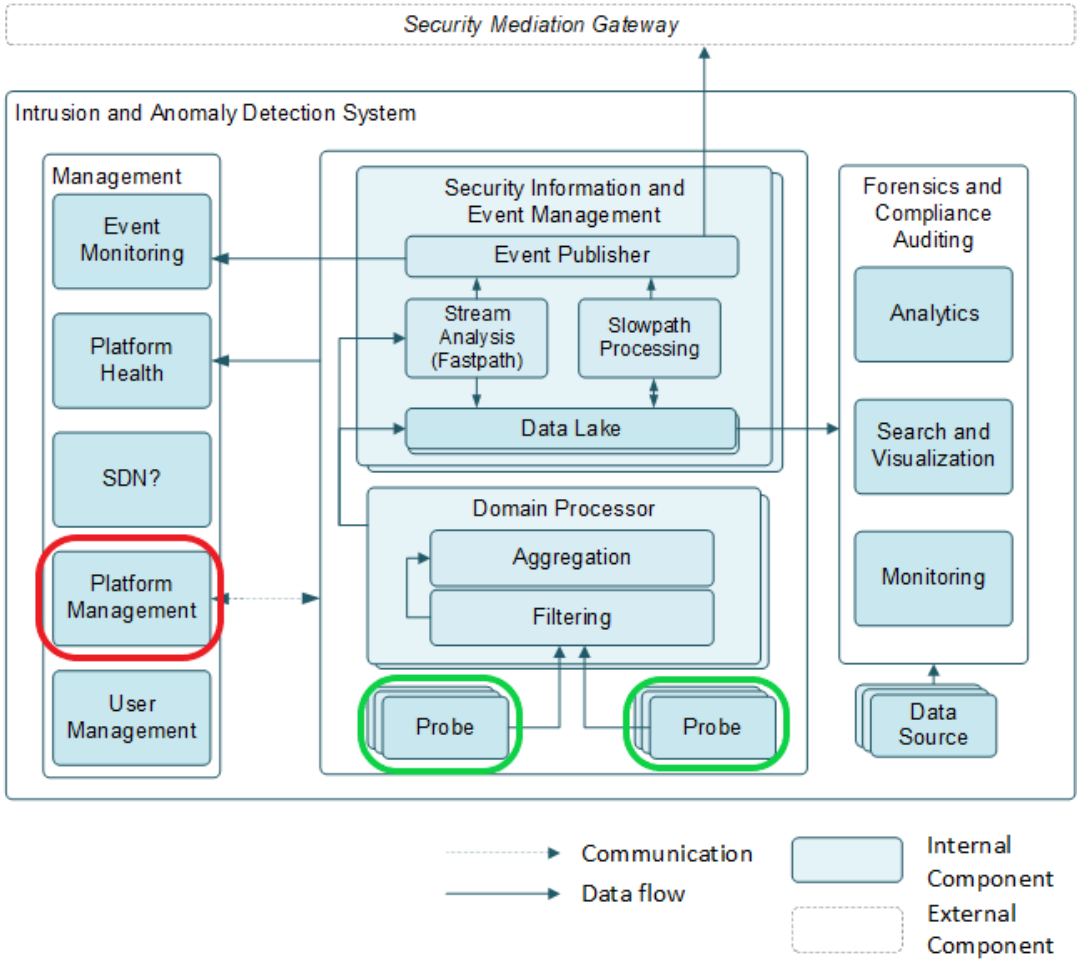


FIGURE 3.2: IADS Architecture

There are two main components marked in the figure 3.2 clarifying the previously mentioned tasks, although only these two particular components are developed, it is necessary to communicate with other components, but this will be explained in the proper components. A more specific diagram with the respective

hardware is presented in Figure 3.3

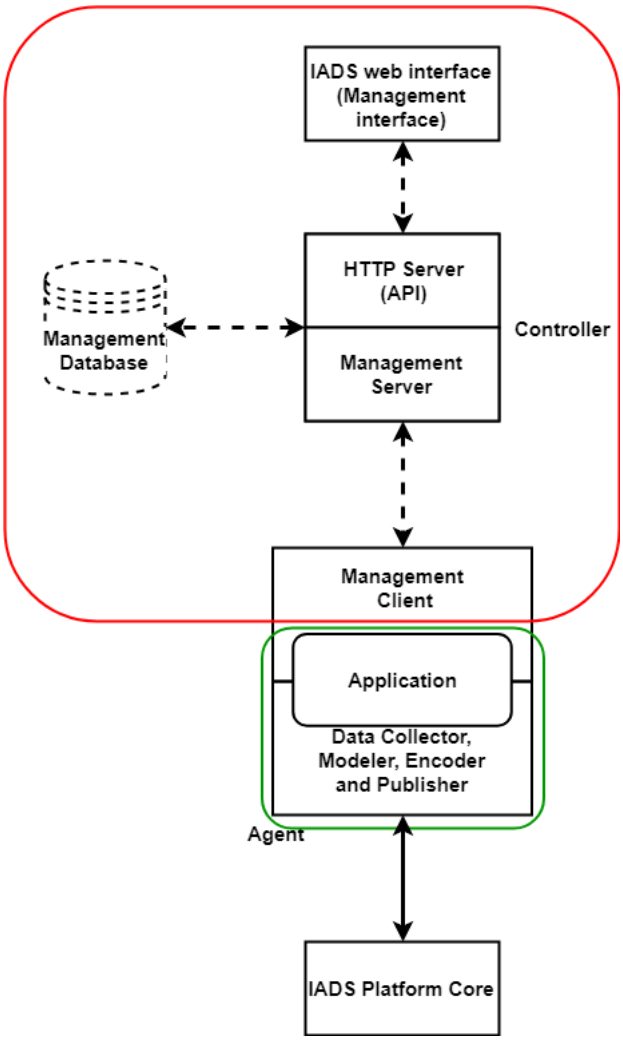


FIGURE 3.3: IADS Platform Management architecture and probe integration

In this second figure we have the component corresponding to the red segment in Figure 3.2, this component will be responsible for managing and monitoring every deployed probe. Obviously, the managing and monitoring features, as stated in Chapter 2, go a little further allowing client registration and others. Another task that can be done using this management component is configuring every other component in the infrastructure, from the domain processor to the security information and event management.

Finally, the green marked component in Figure 3.2, is the component representing the network probes. In the following Figure 3.4 is presented what is the designed architecture for this type of components. We can divide this solution to better explain each component:

- **Network capture**, this component acquires all the data to be filtered and analysed, this is usually a mirror port configured in the router to the machine responsible for the probe;
- **Security detection system**, this component in most cases uses a third-party software because of its complexity. Some examples of this components are

NIDS, Honeypots, among other kinds of security agents. These components send alerts, notifications or reports to the next component;

- In the **system log server** we have a solution based on third-party software, with the advantage that almost every computer has a built-in server of this kind. The Syslog server brings us some advantages when it comes to data since it will store the received logs locally, in case of some network failure or agent error the data will be preserved and sent after when the agent recovers;
- The **Developed Agent**, is not a third party solution, and its main features are collecting the untreated data from the Syslog server, building a new message, in a specially developed data format, with the necessary content collected from the data and sending this message encoded in that specific data model.

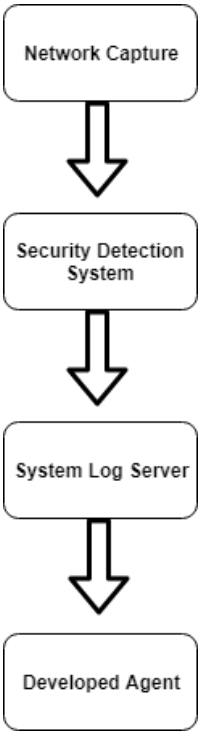


FIGURE 3.4: Generic Developed Probe Architecture

## 3.2 Requirements

The proposed and approved requirements for probes management platform will be presented in the Appendix A.

There are two actors in these requirements the System Admin and the Security Admin. The System Admin has full administration and control over all the cyber-physical IADS except the virtual infrastructure, the Security Admin has full administration access only to the security components of the cyber-physical IADS except the virtual infrastructure.

Table 3.1 is a summary of the requirements presented in the appendix, which explains what is expected from the management system. The data collector does not have any special requirements since part of the solution is already tested and implemented by other members of the team, the only specific requirement is to collect the

data from a generic generator (system logs server) and fill the data in the data model used in the project.

| ID | Requirement |
|---|---|
| **MP_FR1** | **Provisioning a Probe Component –** The system must provide a way for the actor to provision the resources needed for probe deployment |
| **MP_FR2** | **Deploy a Probe Component –** The system must provide a way for the actor to deploy the resources needed for probe configuration |
| **MP_FR3** | **Enable a Probe Component –** The system must provide a way for the actor enable a probe |
| **MP_FR4** | **Configure a Probe Component –** The system must provide a way for the actor be able to configure a probe |
| **MP_FR5** | **Disable a Probe Component –** The system must provide a way for the actor disable a probe |
| **MP_FR6** | **List Probes Components –** The system should provide a way for the actor with all resources needed to view all existent probes |
| **MP_FR7** | **Remove a Probe Component –** The system must provide a way for the actor to remove a probe |
| **MP_FR8** | **Manage Rules –** The system must provide a way for the actor to manage the rules for a probe |
| **MP_FR9** | **Probe Status –** The system should provide a way for the actor to view the probe's status |
| **MP_FR10** | **Create Rule –** The system must provide a way for the actor to create a new rule for a probe |
| **MP_FR11** | **Enable Rule –** The system must provide a way for the actor to enable a rule for a probe |
| **MP_FR12** | **Modify Rule –** The system must provide a way for the actor to modify a rule |
| **MP_FR13** | **Disable Rule –** The system must provide a way for the actor to disable a rule |
| **MP_FR14** | **List Rules –** The system should provide a way for the actor to list all existent rules |
| **MP_FR15** | **Remove Rule –** The system must provide a way for the actor to remove a rule |

TABLE 3.1: Management Probes functional requirements summary

## 3.3 Adopted Technologies and data formats

In this section, we will explain the technologies and data formats used in both management and events solution, based on the knowledge acquired in Chapter 2.

### 3.3.1 Management solution

After analysing the protocols and technologies in Chapter 2 we decided to avoid the available managing solutions and create our own managing protocol and set of tools.

The main reason to take this choice is related with the complexity of the managing protocols available, both NETCONF and TR-069 are overqualified choices and focused in managing network devices. LwM2M is a suitable choice because of its simplicity and low resource consumption. However, it has two major problems, it runs over UDP making it problematic when working over some networks and limits the usage of components in external networks, the other problem is related with the lack of tools/ implementations and poor documentation of the existing one (Eclipse Leshan).

Therefore we decided to implement our own protocol over a messaging protocol MQTT which is widely used for telemetry and as great documentation and support. This allowed us to build a perfectly suitable solution to the IADS management problem without extra overhead. However this protocol uses TCP it is usually faster and more reliable than LwM2M/CoAP due to the TCP dynamic window size which allows less packet loss than CoAP, which leads to fewer re-transmissions and therefore better performance.

But this does not mean the MQTT is a problem free solution. The first problem found was the need to build a solution from scratch (not only an implementation but also the protocol), MQTT cannot achieve the lightness of CoAP since it runs over TCP and typically holds a connection open (which could be a problem not only in power consumption but also if there is a high packet loss ratio on the network).

The integration of MQTT capabilities for existing and new probes will be based on a generic management agent that provides the management API frontend, integrated with a customisable coupling layer which translates the management operations into native configuration backend modifications for the probe, also being able to perform operations on its runtime components (for instance, to restart a component).

The language used to develop this task was Python 3 which is not a programming language with good performance when compared to a compiled language, but it is a language with great support and extensive set of libraries which eases the work of developing a protocol over MQTT, while exposing an API, that needs to contain a changelog database. Besides all these features, the management client also needs a file watcher that notifies the client if any configuration file is changed. Not forgetting all these features needed to be cross-platform.

- **HTTP Server:**
  - **Adopted Technologies:** Python3 Flask and JSON
  - **Data formats:** Defined payload (Listing 4.3 and 4.4)
- **Management Database:**
  - **Adopted Technologies:** Python3 SQLAlchemy and SQLite 3

- **Management Server:**
  - **Adopted Technologies:** Python3, Python3 pyAesCrypt and eclipse Paho-MQTT
  - **Data formats:** Defined payload (Listing 4.3 and 4.4)
- **Management Client:**
  - **Adopted Technologies:** Python3, Python3 watchdog and eclipse Paho-mqtt
  - **Data formats:** (Listing 4.3 and 4.4)

### 3.3.2 Events solution

This section will be divided by the 3 last components of Figure 3.4 and after subdivided into Adopted technologies and data formats. It is necessary to explain that most of the solutions used are well documented and widely used software. All the decisions made had in account some particular aspect like stability and scalability to avoid any bottleneck in the data flow. Rsyslog (a system log server) can handle up to 250K messages per second [77] with a relatively low budget computer. Snort performance is also known to be very efficient, although it can become a real mess if we want to monitor every type of attack (without any type of tuning in the rules) or even if we have to monitor hundreds of devices at once. When it comes to the programming language decision, it is all related to both performance and libraries compatibility, since it was necessary some compatibility with libraries like Avro (encoding).

- **Security detection system:**
  - **Adopted Technologies:** Snort
  - **Data formats:** Undefined and irregular data format
- **System log server:**
  - **Adopted Technologies:** Rsyslog
  - **Data formats:** RFC 3164 and RFC 5424
- **Developed Agent:**
  - **Adopted Technologies:** Golang and Avro
  - **Data formats:** RFC 3164, RFC 5424 and CPIDS events (developed for ATENA)

## 3.4 System overview - a brief review

In this chapter were introduced both architectures of the ATENA project and the developed solutions, after that, we analysed the architecture, we made a brief overview of the necessary requirements to develop this solution, and finally, the technologies used to implement these architecture and requirements.

This brief planning allowed us to develop the solutions explained in the next chapter.

# Chapter 4

# Development / Implementation

This Chapter marks the beginning of the second part of this report. Here, after studying the technologies being used in other projects and deciding what technologies to use in our development, we will address the practical components of our work. We intend to explain everything we implemented, the decisions made as well as any deviation in the implementation when compared to the initially planned implementation. Since we developed a management solution from scratch we decided to explain the created protocol to try to facilitate the implementation's comprehension. All the protocol diagrams will be presented in the Appendix B in full size in order to be easier to read.

## 4.1 Application and Probe integration

The Figure 4.1 is a simplified architecture of the management solution. It represents the necessary elements to implement full data collection and management support. The Application represents the component of the IADS platform, either a probe or other component of the system, that needs to be integrated with the global management architecture. The probes collect evidence and feed the IADS platform with security events based on that evidence. This is a simplified view since the data flows are only represented by lines in this diagram, omitting the needed brokers to support that data flow. The brokers work as messages queues that implement a publish/subscribe mechanism to support that data flow. There are two kinds of connectors in the figure, the dashed lines that represent the management flow and the solid lines that represent the data events flow.

In the management part, there are three major blocks: the **controller** used to provide the integration between the agents and the IADS Web interface; the **management database** used to store the changes in the managed client's, named the **agents**; and the **agent** used to connect the applications with the IADS platform.

An agent has a management client and a data collector (Data collector, Modeller, Encoder and Publisher on Figure 4.1), that run on the same machine as the application, although they are distinct software components. The agent connects any application - either third-party software or one developed within the ATENA project – to the global management system and to the security events processing system. Those applications all have their particularities in the way they are configured and how their data is generated. To abstract the differences between the various applications – management and security events generation - the agent is a wrapped application composed by the management client and the data collector. The main objective is to avoid, as much as possible, interacting directly with the application. In this way, we have a unified and unique way to collect all data and manage every application.

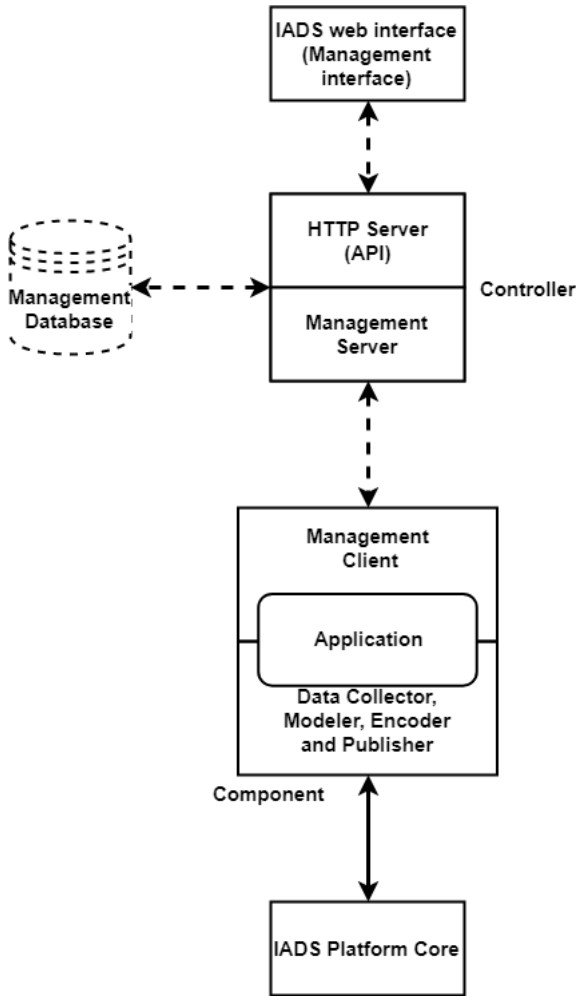


FIGURE 4.1: IADS Management architecture and probe integration

This new **component** - composed by the agent and the application – works as a probe if, for instance, snort or tcpdump are used as applications to collect network information. The component – with the absence of the data collection part of the agent – is also used with other applications, that are not probes, to provide the management capabilities to those applications. The use of the agent with applications like the Apache Kafka [78] or Spark [2] software (used in the **D**istributed **I**ntrusion **D**etection **S**ystem (DIDS) platform), will enable their partial configuration within the IADS platform as any other component.

The security events use the data flow to reach the DIDS or IADS platform core, while the management data uses the management flow to reach the controller. Although they are different flows, the management client can be used to control the data collector, as well as other components, as explained in the next section.

The data collector needs data to stream security events to the DIDS, therefore an application collecting data is mandatory. As an example, a probe application that is configured to transmit data to a Syslog server (Rsyslog server) reaches the IADS platform via the data collector. The data collector reads the data from that server, parses each log using regex and adds some additional information: probe identification, the data, a timestamp, severity, and checksum among others, to build the security event messages. Finally, the message is encoded using Apache Avro [79] and transmitted, by the data collector, to the IADS platform core.

The data collector is modular and generic. This means that for different applications the needed customisations are minimal and limited to the read part of that data collector, that should be adapted to the data source and format. The encoding and the build of the security event message parts remain the same.

## 4.2 Management and monitoring solution

To fulfil the demand requirements for the IADS of managing the components in a uniformed and unique way a new framework was needed. To implement this solution it was necessary to ensure that either probes and other IADS components could be managed and configured using the same procedure. This means that, despite the existing differences between applications, it was needed to abstract those differences with the used controller (see Figure 4.1). As some of the applications used are third party components, and no change can be made to how they load their configuration files, a methodology was developed to overcome these limitations. Also, a simple and new management protocol was developed, from scratch, as well as the necessary management client to manage each component and management server to orchestrate them.

Several managing protocols were analysed like LwM2M, TR-069 and NetConf, but between their large amount of features, most of them were not suitable for the required approach, conducting to the decision of developing a lighter protocol focused just on those needed requirements.

The decision of which messaging protocol to use - reducing the list by selecting only protocols with reliable message delivery - leads to protocols using UDP with QoS or TCP. Another requirement defined was that the management protocol should be as light as possible and this narrowed our options to CoAP and MQTT. The use of MQTT was preferred because of its simplicity, good documentation and support community. Other components of the global ATENA architecture are also using MQTT which reinforced this decision.

The use of MQTT has two mandatory components in the system: a broker (MQTT mosquito [26] is used) and multiple clients (python version of Eclipse Paho [22] is used) – one for each agent and one for the server - the controller. These building blocks have the following roles:

- The **management client**, from each agent, exposes the available configurations to the controller and modifies the necessary configuration files in each application.
- The **controller** is a server, responsible to expose all the clients' configurations to the IADS Web interface through a REST API. It is responsible to control which clients are connected and to communicate with them via the MQTT broker. It can be seen as a middleware between the IADS Web Interface and each component form the IADS platform.
- The **broker** work as a messaging queue system, having no direct interaction with the management. It is the bridge between clients and server.

  The flow of the developed management protocol is explained by describing the steps from provisioning to management. A general overview of the protocol can be visualised in Figure 4.2.

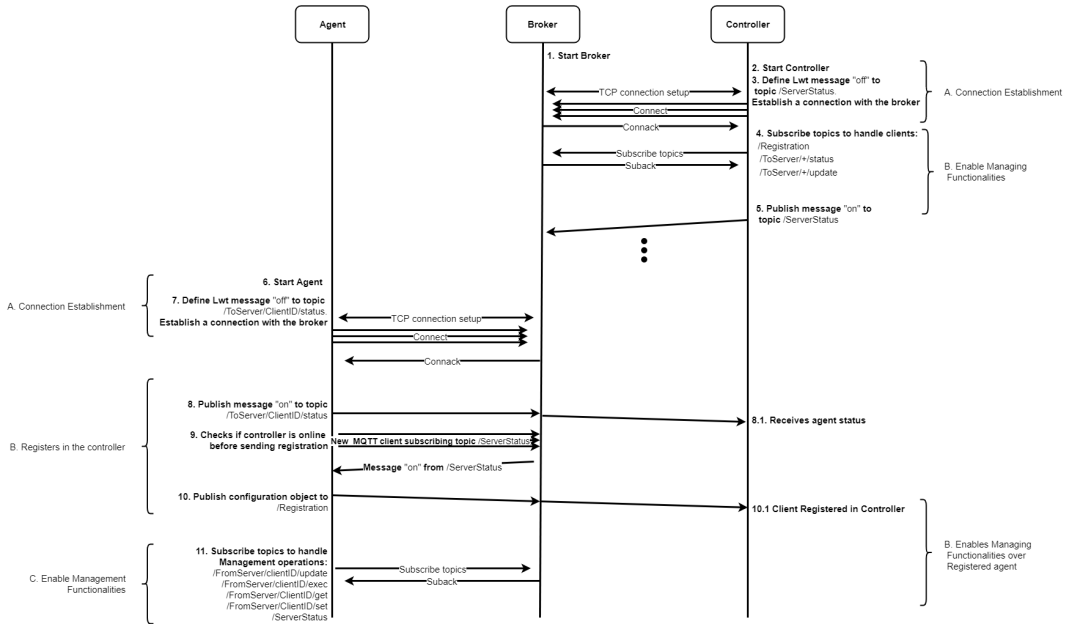


FIGURE 4.2: Registration process on MQTT management solution

### 4.2.1 Provisioning

The first step is called provisioning, where the administrators define what are the used configurations files in each component/application. This is a manual process and uses a manifest file, designed for that purpose. This operation occurs before starting the agent. This manifest file is an encrypted file, composed by topics each one with a respective topic description, keys and values. The topic corresponds to a group of configurations, usually related to one specific application or software. Inside each topic, multiple keys (which are aliases for the application parameters) can be defined, each one with multiple fields. These fields correspond to:

- **Value** - the default value assigned to this key.
- **Description** - information about this key-value pair explaining for instance what they do, this information is exposed to the Web Interface informing the user what is the functionality of the parameter.
- **File path** - file to where this configuration is being mapped.
- **Permission type** - flag with the permission level for the key - read, write or exec. All keys, except exec type ones, are readable. Only the write type keys are changeable and only the exec type are executable. Some parameters, that won't change along the process read, don't need to be updated and can be requested directly from the controller. This avoids some I/O operations.
- **Service** - applications running could have a specific service associated, for instance, snort has snortd service. This represents the service that needs to be restarted, in order to apply the configuration changes. This field can be empty.
- **Regex** - this regex value is a regular expression used to parse the original parameter in the file and match it, to the configuration alias (the key). For instance, a key will have a value of 127.0.0.1 and the regex will indicate the original parameter (IP) in the file referred in the path.

This manifest file also allows comments usage if starting a line with a hash. The defined separator between keys is, also, the hash character. There is a specific topic called static reserved for the management client configuration. A sample decrypted manifest file is shown in Figure 4.1. This manifest allows to map this parameters as alias in our manifest, abstracting and hiding the real configurations and file paths from the platform users.

```
# this is a comment
[Topic1] # Topic information
Key1 = Value # Description # File Path # Permission type #
    Service # Regex
Key2 = Value1 # Description # File Path # Permission type #
    Service # Regex

[Topic2] # Topic information
# this is another comment
Key1 = Value # Description # File Path # Permission type #
    Service # Regex
```

LISTING 4.1: Sample of a manifest file mapping

After the provisioning, the second step, is the deployment of the component. This can be done either manually or using **S**oftware **D**efined **N**etwork (SDN). The agent's deployment implies that both broker and controller are configured and with a connection established between them. Either way will end up with a component connected to the configured network, that will automatically register in the controller.

### 4.2.2 Connection

The connection process between broker and agent/server is similar, we create a TCP connection between the two instances than the MQTT connection is initiated through a client sending a CONNECT message to the broker. The broker responds with a CONNACK.

The content of the CONNECT message has the following payload:

```
clientID                    "mqtt1"
cleanSession                "true"
lastWillTopic (optional)    "/ServerStatus"
lastWillQos (optional)      "2"
lastWillMessage (optional)  "off"
lastwillRetain (optional)   "true"
keepAlive (optional)        "60"
```

LISTING 4.2: Sample of a CONNECT payload

- **clientID** - the client identifier is an unique identifier of each MQTT client. This identifier is only set by the user on the agent, in the server case, the clientID is generated automatically.
- **cleanSession** - a cleanSession indicates if the client wants to establish a persistent session or not. The persistent session (cleanSession set to false) indicates

that the broker will store all subscriptions and all missed messages, when QoS is 1 (at least once) or 2 (exactly once). In case cleanSession is set to true the broker won't store clients' information which is useful to ensure a proper restart of the client.

- **lastWillTopic** - indicates the topic where the last will is published.
- **lastWillQos** - indicates the QoS of the last will message.
- **lastWillMessage** - sets the payload of the last will message.
- **lastwillRetain** - sets if the last will message should be persistent in the broker. This functionality permits the status mechanism usage in the protocol (Example of an usage of LWT as status, Figure 4.3). As mentioned in Section 2.2.3, if the client is shutdown the last will message is published and remains sustained in the broker. The absence of a message in this topic indicates the client was not even started.
- **keepAlive** - is the time interval defined in seconds to send regular PING Request messages to the broker. The broker responds with a PING Response allowing for both sides to determine if the other is still alive and reachable.

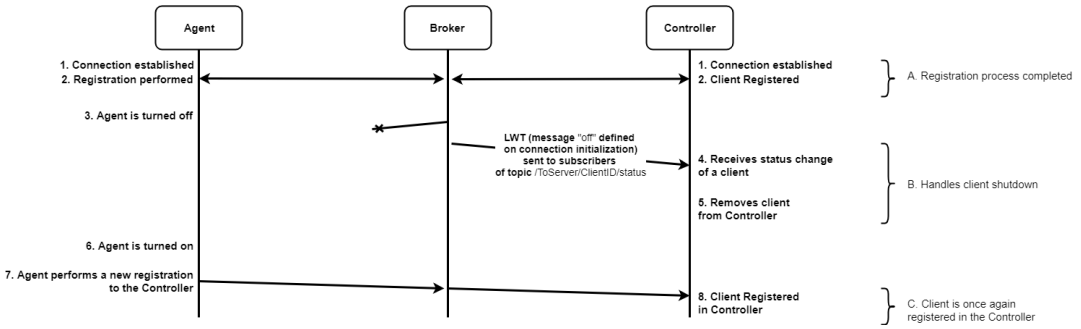


FIGURE 4.3: Client disconnection and re-connection on MQTT Management solution

After the connection is established between broker and agent/server the next step is to change the status of this component and to subscribe the necessary topics.

### 4.2.3 Handling the management components

In order to ensure the server is running, it will automatically set the last will message to a sustained message "off", on the connection process, and immediately after connecting to the broker it will publish the sustained message "on" to the /ServerStatus. In this way, we ensure that the server sets its status when it starts and at the same time ensure that the broker will update the status if the server is unexpectedly shutdown. The server also subscribes the /Registration, /ToServer/+/status and /ToServer/+/update (initially we defined the multilevel subscription /ToServer/#. We changed this subscription in order to optimise resources since the /ToServer/# subscribes every sub-ramification topic that starts with /Toserver). The /Registration topic is the topic that will handle the client registration, the /ToServer/+/status and /ToServer/+/update are responsible for updating the clients in the controller, and they are one direction communication topics.

The second step, consists of starting the agent (the management client) of that component, as an MQTT client. This will load an object with every key-value pair, of the manifest file, and update these values, according to the path where each one is mapped. After loading this configuration object, a connection to the MQTT broker will be established. If the broker is not available a retry loop will be started until successfully connected. When connected the same procedure to set the client status is used - as explained for the server - only changing the topic from /ServerStatus to /ToServer/<clientid>/status. The <clientid> is variable, unique and defined in the manifest. After the connection of the agent, the broker is checked to find if the controller is running. This same procedure is also used whenever there is a message transmission.

The third step consists on sending the Management Client object to the controller, to perform the registration. Before sending this object, the client checks if the server is up - in case of a negative response, the client stays stuck waiting for the server to come up – and then the client sends the information to the server. This object only contains the necessary information from the manifest: key, value, description and permission type. The client will subscribe the topics /FromServer/<clientid>/<method> (being the methods [update,exec,get,set]) and /ServerStatus to handle every server request or operation. When this task is finished the Client is ready to handle the management requests.

These management requests come from the controller, which will expose a REST API that allows the manipulation of the agents from the IADS Web interface. The MQTT operations of update, get, set and exec are mapped to the REST API.

### 4.2.4 Request types

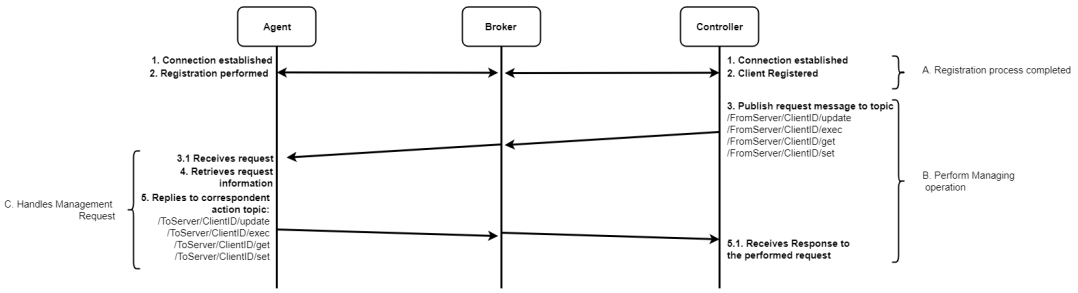


FIGURE 4.4: Request process on MQTT management solution

Once again we initially planned to have a topic that would allow agents to be remotely shutdown in cases of spoofing but this option was invalidated since it was not the right security measure (both the agents were shutdown instead of only the one spoofing). This operation was used for instance when a client with the id "client1", already registered in the system, and another client tried to register in this server with the same id. This delete operation would publish a message to the MQTT broker, directed to the "client1", in the topic /FromServer/<clientid>/delete and both clients were disconnected. This procedure ensured the original client would not suffer unwanted operations, and also that the other client does not collect information about the original client.

The update operation consists of a request from the server to the topic /FromServer/<clientid>/update, the agent will open every configuration file mentioned in the manifest updating the object and send it to the server. This process is similar to the registration process although this starts from the server, unlike registration.

The operations of get, set and exec have the same behaviour. The controller makes a request of one of these types, and waits for a response to appear. While the response is not received, this request is stuck waiting for the client response (if the response takes too long the request will timeout). Once the agent receives the request, it proceeds to perform the requested operation and retrieves information to the server. In case the operation is not valid or cannot be performed, the payload message will have a details parameter (with value 200 in case of success or 400 in case of failure) and a value parameter where we will retrieve the results or in case of error a message explaining what went wrong.

```
{"client": "",
 "file": [
        {"characteristic":
         {"test": {"description": "avro enconding status",
                           "details": "200",
                           "mode": "write",
                           "value": "127.0.0.1"
                          }
         },
         "description": "server information",
         "name": "server",
        }
 ],
 "timestamp": ""
}
```

LISTING 4.3: Sample of a payload

Listing 4.3 presents one example payload message. This payload corresponds to the set operation performed in the request 4.4. Besides this payload message the controller also sends other fields in each message to control the replies from the clients. We can say that each request has a body with a message, request type, timestamp and message id. The timestamp and message id are only used for flow control and to ensure the message replied goes to the right request. The message has the content of each request and finally, the request type is a mechanism to ensure the request message is in the right topic.

```
{"topic": [{
                "topic_name": "server",
                "key": [{
                        "name": "test",
                        "value": "127.0.0.1"
                }]
        }]
}
```

LISTING 4.4: Sample of a payload JSON request

There is one request type (Figure 4.5) that functions as a callback in the server. This request occurs only from the agent side and it is automatically triggered when a file contained in the manifest file is changed. To perform this operation, a thread is watching every file present in the manifest file and whenever there is a change, an

update is sent to the controller. The filewatch operation is handled with the open-source package Watchdog [80]. This package is a cross-platform library, which provides a solution to the cross-platform and generic requirements on our solution.

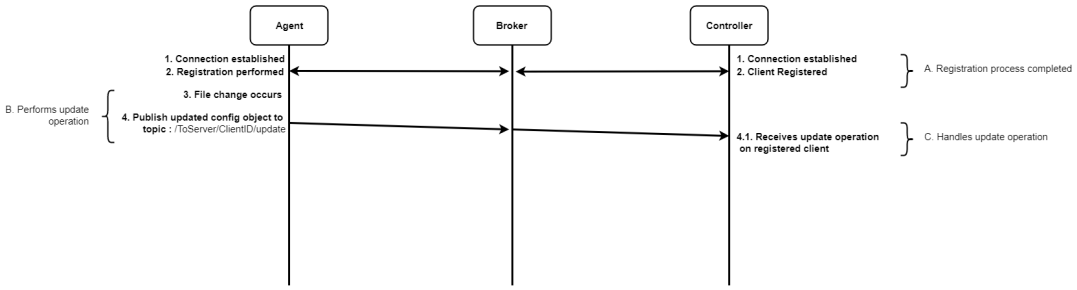


FIGURE 4.5: Client file change notification process MQTT on management solution

### 4.2.5 Failover mechanisms

To ensure this management solution works continuously and handles power surges and other unexpected stops, it was mandatory to implement some failover mechanisms and to create a service in the operating system to handle reboots and other unexpected behaviours. The failover mechanisms were developed for two circumstances: broker failure (Figure 4.6) and controller failure (Figure 4.7).

When there is an unexpected broker failure, both agents and controller stay stuck in a MQTT internal package loop trying to reconnect. When they are able to establish the connection with the broker, we obtain two different connections, normal connection and registation cycle. Both connect to the broker and both subscribe the necessary management topics, but the agents are stuck until the controller sends the message on to the topic /ServerStatus. Only when that happens, the client can send the configuration object to the controller and finish the registration (Figure 4.2).

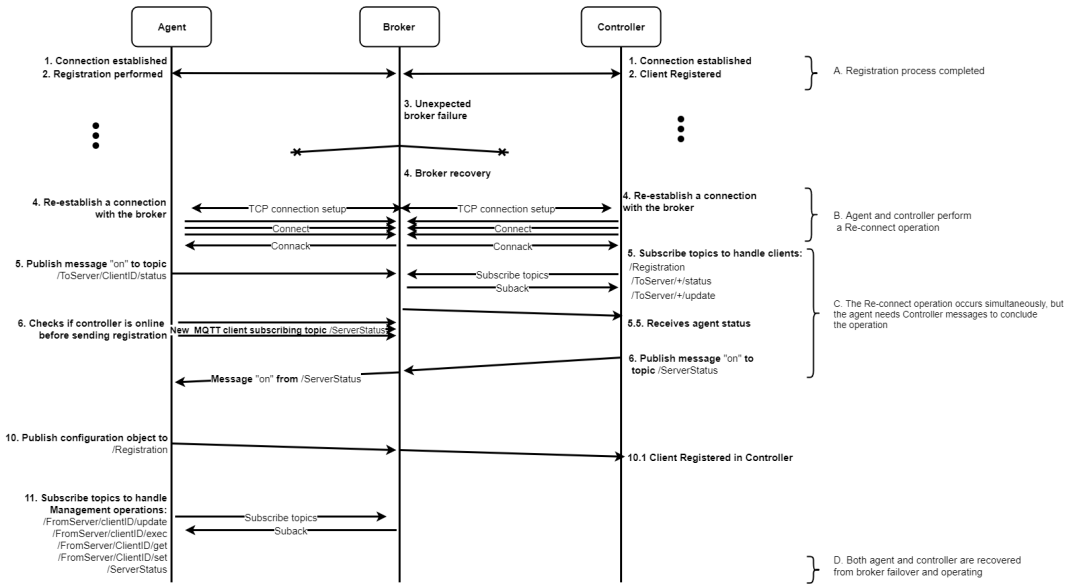


FIGURE 4.6: Broker failover on MQTT management solution

When there is an unexpected controller failure, the agents receive the last will message "off" in topic /ServerStatus and wait until there is a new message "on" in

this topic to perform the reconnect operation. This operation will work a normal registration (Figure 4.2)

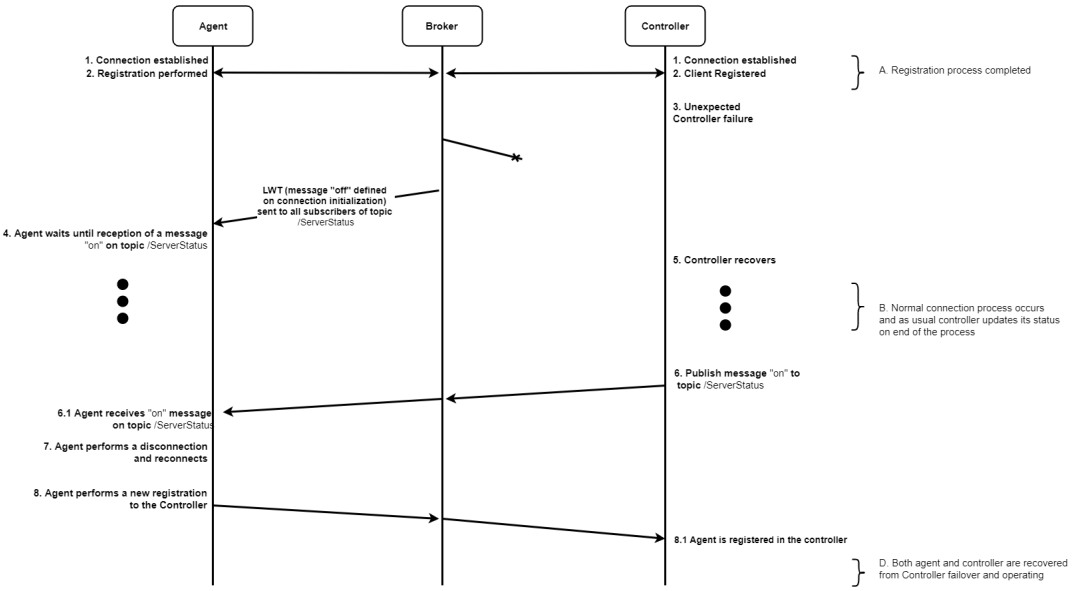


FIGURE 4.7: Controller failover on MQTT management solution

### 4.2.6 Security aspects

The use of a username and password on the MQTT broker avoids its use by unknowns. Furthermore, the usage of certificates to encrypt the connection using TLS avoids eavesdropping or even man-in-the-middle attacks. Adding this encryption in the connection adds some overhead in all transmissions.

Using an encrypted connection with asymmetric encryption, to further improve the security the payloads could be also encrypted, adding a second encryption and ensuring more security to the transmissions.

The payload can be encrypted using Python's encryption library Fernet package. This package receives the user message, a 256 bits key and the current timestamp. Next, it produces a token which has the message encrypted. This message cannot be read or decrypted without the 256 bits key since it was encrypted using Advanced Encryption Standard (AES-128) in Cipher Block Chaining (CBC) mode.

At this point we already have a consistent protocol, but we still have some failures. For instance, whenever someone makes a mistake in the configuration there is not a way to revert the configuration. To solve this reliability problem we used SQLAlchemy [81] and SQLite [82] in the controller level to implement a changelog database. This database allows the registry of every modification made, the agents connected or even when the changes occurred. All actions are logged on the database: whenever there is a component registration, disconnection or even a request from the controller to the agent. In case the server reboots, the database is checked and all registered agents are added to the server and requests are sent to check which ones are still online. That new status is logged on the database providing a consistent view of the overall system.

The Flask [83] framework is used to implement an HTTP server that exposes the REST API available to the IADS Web interface.

### 4.2.7 Web interface

This component has one specific function, handling the requests that come from the user interface, convert the **J**ava**S**cript **O**bject **N**otation (JSON) payload requests (if their JSON schema was correctly validated) into managing objects and perform the operations to retrieve the required information, again as a JSON object. To ensure this HTTP server functionality we use Flask [83], a microframework for Python based on Werkzeug [84] and Jinja 2 [85]. This API will expose the following endpoints:

- **/list** - this will be the first request used to obtain the list of components (agents) registered with the controller as well as its present configurations.
- **/list/<clientid>** - this request is used to obtain a component (agent) present configurations without communicating with the agent.
- **/<clientid>/get** - after knowing the agent is registered in the controller, we can perform operations focused on the specific agent. The get operation retrieves a particular configuration parameter directly from the agent. This operation forces an update on the selected parameter.
- **/<clientid>/set** - another operation available after knowing the agent's id (clientid) is the set operation. It allows to change the parameters according to the permission level they have. If the parameter has write permissions on the Manifest file, it can be changed, otherwise, this operation is not allowed.
- **/<clientid>/exec** - just like the set operation we can only perform this operation in parameters that have the exec permission. This endpoint allows us to execute some predefined operation in our component.
- **/<clientid>/update** - finally the update operation is a global operation since it is done to perform an (forced) update to the whole agent object updating every configuration parameter: value, description and information to its present state.

## 4.3 Events

As explained at the beginning of this chapter, one of the objectives of our agents was to create a data flow that could reach the DIDS. Therefore, an application to collect data is mandatory. This data collecting task is granted due to a system log server. In this case, Rsyslog that joined with its module "omprog" permits to execute an external program, where it is executed in a new thread. So every time a new log surges rsyslog creates a new thread that writes into a unix socket. This unix socket is opened in our data collector application that will receive the messages.

### 4.3.1 Data collector

On a new message received our developed client will try to extract information from this message. Since Rsyslog uses a default message payload, we can easily extract the information using some regular expressions and map them to the wanted data model. In this data model, there are some specific fields that receive dynamic

information, where it is mapped using regular expressions that are stored in a configuration file. After getting this data model filled, the next step is to send the information to a distributed streaming platform that will filter duplicates, aggregate results and perform other data operations in the rest of the DIDS components.

## 4.4 Development overview

In this chapter, we explained all the implemented solutions, both management and event collector solutions. We started the chapter by explaining how and why we integrated both solutions in the ATENA platform, then we described the connection mechanisms, the provisioning, all used payloads, the request types and how they are made, how the agents connect/disconnect to the controller, how the controller handles the agents, the failover mechanisms, the security measures and the HTTP server and its endpoints. When it comes to the event collector solution, we explained how the data was collected, handled and sent to the distributed streaming platform.

From the functionalities implemented we can identify the requirements they are mapped to. For instance, the creation of rules to be displayed on the platform are made in the manifest file and can only be performed before starting the management client.

The manifest file is used in MP_FR1 and MP_FR2 functional requirements corresponding to the provisioning and deployment tasks. The deployment can be performed either manually or automatically via SDN.

MP_FR6 and MP_FR14 can be performed by using the endpoints /list and /list /<clientid> where we can list both probes/ components and rules defined inside of them.

The rest of the functional requirements can be performed by interacting with the management clients and performing the necessary requests. For instance, we can activate or deactivate a component by changing a configuration, but we can also perform this tasks by executing a script on the client. The same process can occur if we want to configure/ manage some rule in the probe. A special case is the monitoring one where we can only get the current status of some component/ rule by performing an execute-type request.

In the next chapter, we will test and validate the previously mentioned solutions, trying to cover both the functional and non-functional requirements, and analysing the obtained results.

# Chapter 5

# Tests and validation

Validation plays an important role in the development and implementation of any software. Although it is the final step in any project, any implemented solution is susceptible to bugs, therefore fixes and new validations must be performed. With this, validation is needed so our work may be deployed into a production environment.

Since our agents are composed by a management and an event collector part, we divided our validation into two sets of tests: One where we validate our event collector solution and other where we validate our management solution.

This Chapter has three sections. First, we present the testbed scenarios by detailing the hardware and used components. Next, we show a list of tests, consisting of the expected results of the validation. Finally, the results obtained after performing the validation, where we include any unusual or unexpected behaviour. These sets of tests, predictions and results are presented as a list, showing the performed test to validate the functional requirements, and a different list regarding non-functional requirements.

## 5.1 Scenarios

Although both set of tests have different scenarios, they were both virtualised using the same server, a **Dell PowerEdge R710** with an *12 CPUs x Intel(R) Xeon(R) CPU X5660 @ 2.80GHz* and *191.99 GB RAM*. This server uses as bare metal (type 1) hypervisor **Vmware ESXi** and any virtualised machine used in the scenarios uses CentOS 7 with Kernel Linux 3.10.0-229.7.2.el7.x86_64 as guest operating system.

### 5.1.1 Agent Management

Our management solution has multiple components (*MQTT Broker*, agent, controller) and the controller is divided into 3 other components (MQTT client, database and HTTP server) therefore we decided to create two distinct scenarios to validate the management solution.

In the first scenario (Figure 5.1) we use the same hypervisor and server, however with a new virtual machine, with **8 vCPUs**, **8GB of RAM** and **20GB of storage capacity on Non-SSD drives, working in RAID 10**. This machine is running *Eclipse Mosquitto* [26], a *MQTT Broker*, "1, 10 and 20" management agents and a controller (both developed in Python3).

In the first scenario (Figure 5.1) we have all components on the same machine and perform some functional tests as well as non-functional using bash scripting to perform and collect the results. This tests focus on the managing capabilities of the solution and not the payloads sent or the API, which is the focus of the second set of tests.

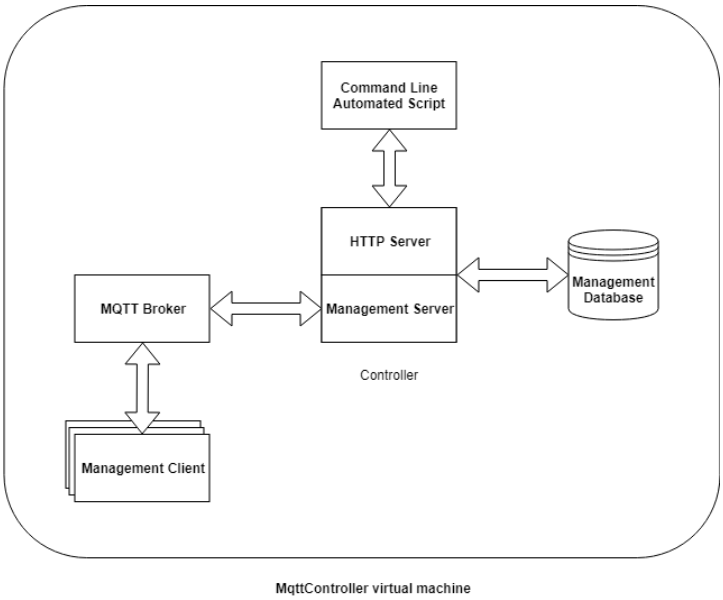


FIGURE 5.1: First Scenario - Management

In the second scenario, the used virtual machine has the same hardware allocation as the virtual machine referred to in the previous scenario.

In this scenario (Figure 5.2) we also do both functional and non-functional tests, but this time we use Postman [86] to automate the tests. In this scenario we created "1, 10 and 20" new management agents on the virtual machine referred in the previous scenario and moved the other components to a new virtual machine, creating a more realistic scenario where the agents are in a different machine from the broker and controller.

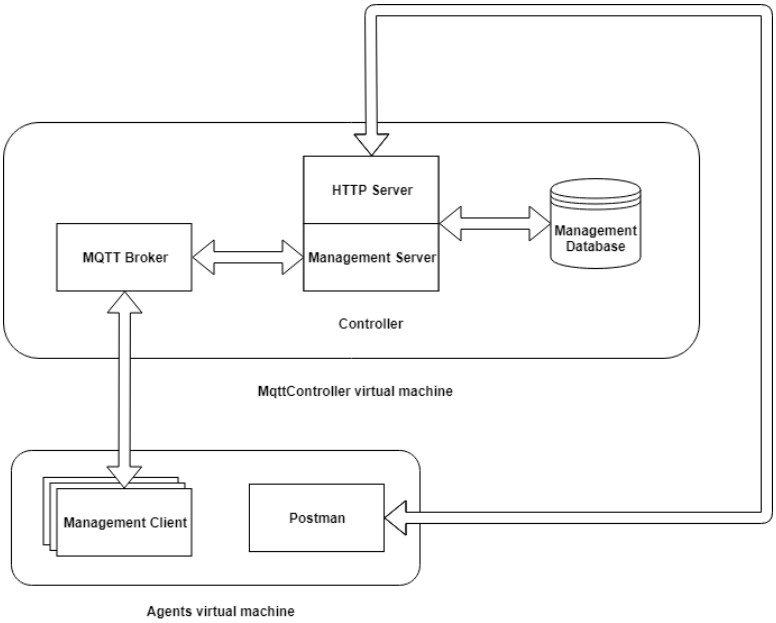


FIGURE 5.2: Second Scenario - Management

### 5.1.2 Event Collector

We also consider a third scenario (Scenario 3 - Events) where we used a virtual machine with **2 vCPUs**, **6GB of RAM** and **26GB of storage capacity on Non-SSD drives, working in RAID 10**.

In this event collector solution the machine is running *Rsyslog* system for log processing and our developed event collector/encoder/sender solution to collect, convert to the ATENA data model, encode in AVRO encoding and send this data to a distributed streaming platform.

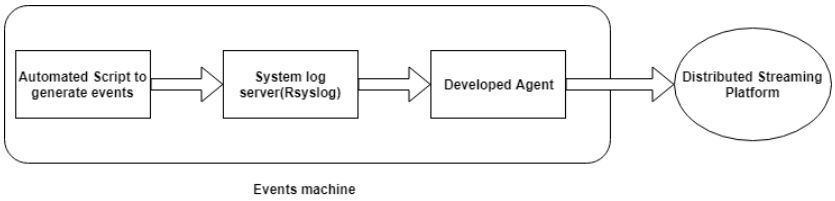


FIGURE 5.3: Third Scenario - Events

## 5.2 Test list

All the tests were repeated 5 times in order to guarantee better precision (increase confidence in collected data, by better calculating the uncertainty) in the tests and avoid some possible outliers. Therefore the measured results will have standard deviation and mean values.

### 5.2.1 Functional

The set of functional tests are divided into two section (Agent Management and Event Collector). The objective of these tests is to ensure that after a certain input the system retrieves the expected output, if the output as the predicted value the tested functionality is validated.

#### Agent Management

The following tests were designed to evaluate the behaviour and functional characteristics of the Agent Management components, accordingly with the previously identified requirements.

| **Test 1 - Add agents to controller** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service;<br>2. Start the controller as an operating system service;<br>3. Start a timer to control the operation, this timer uses the operating system call *time*;<br>4. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially. This agents' manifest files were all created using a python script;<br>5. Stop the timer, this timer uses the operating system call *time* and the result is obtained by subtracting the value in this timer with the value in the first timer;<br>6. Query the database to see if the agents were successfully initialised, we used the command line shell to obtain this results.<br>7. Assert the output to verify the test, to make this step we used a combination between grep and count Linux commands, piping the result from grep command to call and finally comparing the result with an if statement . |
| **Objective** | This test verifies if the agents are successfully registered in the controller and consequently verifies if the database collects the agent registration information. By asserting the database for the registered agents, we can verify that the controller registered the agents correctly. |
| **Expected Result** | Database contains the number of added agents in its tables. |

TABLE 5.1: Test 1 (Scenario 1 - Management)

| **Test 2 - Remove all added agents from the controller** | |
|---|---|
| **Description** | 1. Run Test 1 script;<br>2. Start a timer to control the operation, this timer uses the operating system call *time*;<br>3. Remove the same number of agents (1, 10 or 20) added in the test1, using a loop to initialise the agent making the program call sequentially;<br>4. Stop the timer, this timer uses the operating system call *time* and the result is obtained by subtracting the value in this timer with the value in the first timer;<br>5. Query the database to see if the agents were successfully removed, we used the command line shell to obtain this results.<br>6. Assert the output to verify the test, to make this step we used a combination between grep and count Linux commands, piping the result from grep command to call and finally comparing the result with an if statement. |
| **Objective** | This test verifies if the previously added agents are successfully removed from the controller and consequently verifies if the database contains the agent status information. By asserting the database for the offline agents, we can verify that the controller removed the agents correctly. |
| **Expected Result** | Database contains no online agents in its tables. |

TABLE 5.2: Test 2 (Scenario 1 - Management)

| **Test 3 - Controller failover** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service;<br>2. Start the controller as an operating system service;<br>3. Added a certain number of agents;<br>4. Start a timer to control the operation, this timer uses the operating system call *time*;<br>5. Stopped the controller service;<br>6. Performed a 2 minute wait using the Linux command sleep.<br>7. Started the controller service;<br>8. Performed an HTTP request to the API using the curl command;<br>9. Stop the timer, this timer uses the operating system call *time* and the result is obtained by subtracting the value in this timer with the value in the first timer;<br>10. Query the database to see if the agents were successfully initialised, we used the command line shell to obtain this results.<br>11. Assert the output to verify the test, to make this step we used a combination between grep and count Linux commands, piping the result from grep command to call and finally comparing the result with an if statement. |
| **Objective** | This test verifies if the agents can successfully recover in case the controller fails. By performing a request to the API we ensure the components are working and verify that the agents recovered correctly from the controller failure. |
| **Expected Result** | Database contains the number of added agents in its tables. |

TABLE 5.3: Test 3 (Scenario 1 - Management)

| **Test 4 - Change an agent configuration (local configuration change)** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service;<br>2. Start the controller as an operating system service;<br>3. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially;<br>4. Start a timer to control the operation, this timer uses the operating system call *time*;<br>5. Change one configuration file (to trigger the file-watcher functionality).<br>6. Performed an HTTP request to the API to verify the agent configuration changed, using for that the curl command;<br>7. Stop the timer, this timer uses the operating system call *time* and the result is obtained by subtracting the value in this timer with the value in the first timer;<br>8. Assert the output of the curl command to verify the test, to make this step we used a combination between grep and count Linux commands, piping the result from grep command to call and finally comparing the result with an if statement . |
| **Objective** | This test verifies if the controller is aware of a configuration file change made in an agent it is controlling. |
| **Expected Result** | Database contains the changed configuration registered. |

TABLE 5.4: Test 4 (Scenario 1 - Management)

| **TEST5 - Broker failover** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service;<br>2. Start the controller as an operating system service;<br>3. Added a certain number of agents;<br>4. Start a timer to control the operation, this timer uses the operating system call *time*;<br>5. Stopped the Eclipse Mosquitto broker service;<br>6. Performed a 2 minute wait using the Linux command sleep.<br>7. Started the Eclipse Mosquitto broker service;<br>8. Performed an HTTP request to the API using the curl command;<br>9. Stop the timer, this timer uses the operating system call *time* and the result is obtained by subtracting the value in this timer with the value in the first timer;<br>10. Query the database to see if the agents were successfully initialised, we used the command line shell to obtain this results.<br>11. Assert the output to verify the test, to make this step we used a combination between grep and count Linux commands, piping the result from grep command to call and finally comparing the result with an if statement. |
| **Objective** | This test verifies if the agents can successfully recover in case the Eclipse Mosquitto broker fails. By performing a request to the API we ensure the components are working and verify that the controller and registered agents recovered from a broker failure correctly. |
| **Expected Result** | Database contains the number of added agents in its tables. |

TABLE 5.5: Test 5 (Scenario 1 - Management)

| **TEST6 - Does the controller parse the message payload (JSON) correctly (multiple parameters)** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service on the machine mqttcontroller;<br>2. Start the controller as an operating system service on the machine mqttcontroller;<br>3. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially on the machine with the agents;<br>4. Postman makes a POST request to the controller using the same valid JSON with multiple configuration parameters (some valid some invalid);<br>5. On response postman asserts HTTP status code and validates the JSON response schema. |
| **Objective** | In this test we used Postman [86] to automate the tasks, this test is performed for one of the requests type in the API (get configuration, change configuration, and execute configuration). This test verifies if an agent can successfully respond to the requests made. Therefore validating if a configuration is successfully changed, if a configuration is successfully executed, or if a configuration is successfully obtained. |
| **Expected Result** | This test verifies if the HTTP status code is 200 and validates the JSON response schema. |

TABLE 5.6: Test 6 (Scenario 2 - Management)

| **TEST7 - Does the controller parse the message payload JSON correctly (one parameter)** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service on the machine mqttcontroller;<br>2. Start the controller as an operating system service on the machine mqttcontroller;<br>3. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially on the machine with the agents;<br>4. Postman makes a POST request to the controller using the same valid JSON;<br>5. On response postman asserts HTTP status code and validates the JSON response schema. |
| **Objective** | In this test we used Postman [86] to automate the tasks, this test is performed for each of the allowed agents requests in the API (get configuration, change configuration, and execute configuration). This test verifies if the agents can successfully respond to the requests made. |
| **Expected Result** | This test verifies if the HTTP status code is 200 and validates the JSON response schema. |

TABLE 5.7: Test 7 (Scenario 2 - Management)

| **TEST8 - Does it handle a request to an unknown agent** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service on the machine mqttcontroller;<br>2. Start the controller as an operating system service on the machine mqttcontroller;<br>3. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially on the machine with the agents;<br>4. Postman makes a POST request to the controller using a valid JSON with multiple configuration parameters to an unknown agent;<br>5. On response postman asserts HTTP status code. |
| **Objective** | In this test we used Postman [86] to automate the tasks, this test is performed for each of the allowed agents requests in the API (get configuration, change configuration, and execute configuration). This test verifies if the agents can successfully handle the requests of an non-existent agent. |
| **Expected Result** | This test verifies if the HTTP status code is 400 agent not registered in the controller. |

TABLE 5.8: Test 8 (Scenario 2 - Management)

| **TEST9 - Test invalid configuration types in valid request** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service on the machine mqttcontroller;<br>2. Start the controller as an operating system service on the machine mqttcontroller;<br>3. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially on the machine with the agents;<br>4. Postman makes a POST request to the controller using a valid JSON with invalid types of variables for the request;<br>5. On response postman asserts HTTP status code. |
| **Objective** | In this test we used Postman [86] to automate the tasks, this test is performed for each of the allowed agents requests in the API (get configuration, change configuration, and execute configuration). This test verifies if the agents can successfully respond to the requests made. For instance, the get configuration request will receive variables of type set and exec, but only the get and set types are value for this request. When it comes to the change configuration only the variables of type set are allowed so the get and exec types will not work. In the case of execute configuration only the exec type can be executed, so the other two types will not work. |
| **Expected Result** | This test verifies if the HTTP status code is 200 and the payload contains the value invalid parameter type. |

TABLE 5.9: Test 9 (Scenario 2 - Management)

| **TEST10 - Does the controller handle invalid parameters** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service on the machine mqttcontroller;<br>2. Start the controller as an operating system service on the machine mqttcontroller;<br>3. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially on the machine with the agents;<br>4. Postman makes a POST request to the controller using valid JSON with invalid/non-existent parameter;<br>5. On response postman asserts HTTP status code and validates the JSON response schema. |
| **Objective** | In this test we used Postman [86] to automate the tasks, this test is performed for each of the allowed agents requests in the API (get configuration, change configuration, and execute configuration). This test verifies if the agents can successfully respond to the requests made. |
| **Expected Result** | This test verifies if the HTTP status code is 200 and validates the JSON response schema, the JSON has the normal syntax response, but the value has invalid parameter as payload. |

TABLE 5.10: Test 10 (Scenario 2 - Management)

| **TEST11 - Does the controller handle invalid malformed JSON requests** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service on the machine mqttcontroller;<br>2. Start the controller as an operating system service on the machine mqttcontroller;<br>3. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially on the machine with the agents;<br>4. Postman makes a POST request to the controller using two invalid JSON requests testing invalid structure and invalid JSON keys;<br>5. On response postman asserts HTTP status code. |
| **Objective** | In this test we used Postman [86] to automate the tasks, this test is performed for each of the allowed agents requests in the API (get configuration, change configuration, and execute configuration). This test verifies if the agents can successfully handle malformed requests (3 kinds of malformed JSON). |
| **Expected Result** | This test verifies if the HTTP status code is either 400 (bad request) or 500 (server error - request timeout). In this test, there are two possible behaviours in the requests: either the JSON schema validator catches the malformed JSON and replies with 400, or the JSON passes the validator and when reaching the client will be rejected, making the HTTP server reply with a timeout after 20 seconds. |

TABLE 5.11: Test 11 (Scenario 2 - Management)

| **TEST12 - Does the controller handle malformed url** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service on the machine mqttcontroller;<br>2. Start the controller as an operating system service on the machine mqttcontroller;<br>3. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially on the machine with the agents;<br>4. Postman makes a POST request to the controller using the same valid JSON;<br>5. On response postman asserts HTTP status code. |
| **Objective** | In this test we used Postman [86] to automate the tasks, this test is performed for each of the allowed agents requests in the API (get configuration, change configuration, and execute configuration). This test verifies if the agents can successfully handle malformed URLs. |
| **Expected Result** | This test verifies if the HTTP status code is 404. |

TABLE 5.12: Test 12 (Scenario 2 - Management)

| **TEST13 - List agents** | |
|---|---|
| **Description** | 1. Start the Eclipse Mosquitto broker as an operating system service on the machine mqttcontroller;<br>2. Start the controller as an operating system service on the machine mqttcontroller;<br>3. Start a certain number of agents (1, 10 or 20), using a loop to initialise the agent making the program call sequentially on the machine with the agents;<br>4. Postman makes a GET request to the API;<br>5. On response postman asserts HTTP status code and the JSON schema. |
| **Objective** | In this test we used Postman [86] to automate the tasks. This test verifies if the agents can successfully store the agents' configuration and retrieve them. |
| **Expected Result** | This test verifies if the HTTP status code is 200 and JSON schema is valid. |

TABLE 5.13: Test 13 (Scenario 2 - Management)

**Event Collector**

We decided to run only one test instance for this part of the agent. It consists of a stress test, where we validate the functional requirements of the agent by generating logs and asserting the result in the distributed streaming platform. Also performing a non-functional test evaluating the CPU and RAM usage along the saturation test.

| TEST14 - Does the event publisher handle logs | |
|---|---|
| **Description** | 1. Start the Rsyslog server service;<br>2. Start events agent service;<br>3. Generate random logs until the CPU or RAM are stressed(if possible);<br>4. Track time, CPU and RAM (using operating system functions like time and ps) to understand how the resources are consumed in this agent.<br>5. Compare the number of encoded messages sent with the number of logs generated. |
| **Objective** | This test verifies if the agents can successfully collect system logs, collect the requested data and send it in a encoded payload. |
| **Expected Result** | The produce messages are correctly grabbed and converted. |

TABLE 5.14: Test 14 (Scenario 3 - Events)

### 5.2.2 Performance (non-functional) evaluation

The previously mentioned tests performed were all timed. This allowed us to understand if there was any significant variation in performance according to the number of agents registered and if there was any significant variation in performance according to the number of messages generated in the events. It also allowed us to study (and also help predicting) the scalability of the developed solutions.

## 5.3 Results and observed limitations

In this section we present the results obtained in the tests. These results are presented using two kinds of charts, box plot and bar chart. In the bar chart, we intended to compare the average time of the 5 runs among the multiple categories tested (number of agents and in some cases type of request). In the box plot we can perceive the deviations among runs (identified with points), identify outliers and also compare the average times (identified with a cross) among the multiple categories tested.

In an overall perspective, the functional tests went as expected as shown in Table 5.15, but when it came to non-functional tests results some anomalous results were obtained. These behaviours are mostly explained by the fact that the management client was designed to run in a standalone mode. In other words, having multiple management clients in the same computer was necessary to perform formal testing, but this created a considerable resource overhead, due to the number of threads that had to be spawned. Besides, one of the used features (file watcher) has operating system resource limitations, not allowing us to perform management tests with 100 clients in the same machine, which led us to change the planned test to "1, 10 and 20" agents.

The tables containing the test results are presented in Appendix C.

| Functional Test results | |
|---|---|
| **Test Name** | **Passed** |
| **Test 1** | ✓ |
| **Test 2** | ✓ |
| **Test 3** | ✓ |
| **Test 4** | ✓ |
| **Test 5** | ✓ |
| **Test 6** | ✓ |
| **Test 7** | ✓ |
| **Test 8** | ✓ |
| **Test 9** | ✓ |
| **Test 10** | ✓ |
| **Test 11** | ✓ |
| **Test 12** | ✓ |
| **Test 13** | ✓ |
| **Test 14** | ✓ |

TABLE 5.15: Test results summary

### Test 1 - Add agents to controller

In the first test performed the results went as expected since each client needs to load both manifest file, mapped configurations, and spawn threads for watch modifications in those files. Besides, the server can only register one client at a time queuing the other requests, this meaning that besides the registration process time we have the waiting time in queue, therefore it was expected that the increasing number of agents would lead to a major operation time.

This effect can be clearly seen in Figure 5.4 according to the number of agents spawned.

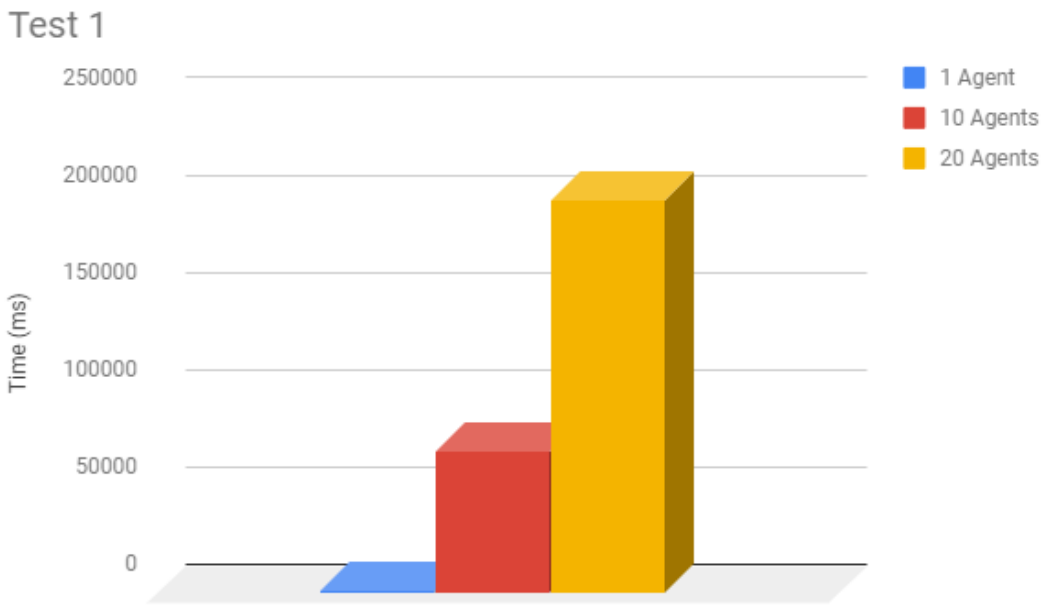


FIGURE 5.4: Test 1 - bar chart with results average

According to this graph, the average time for 1 agent is roughly 2 seconds, with 1 minute and 13 seconds for 10 agents and 3 minutes and 22 seconds for 20 agents, revealing a trend towards exponential time behaviour. But once again there are two aspects to pay attention to: the number of agents per machine was inadequate, and according to Figure 5.5 these tests need more runs to be properly approved since there are some outliers in the tests performed that may have biased the global output results.

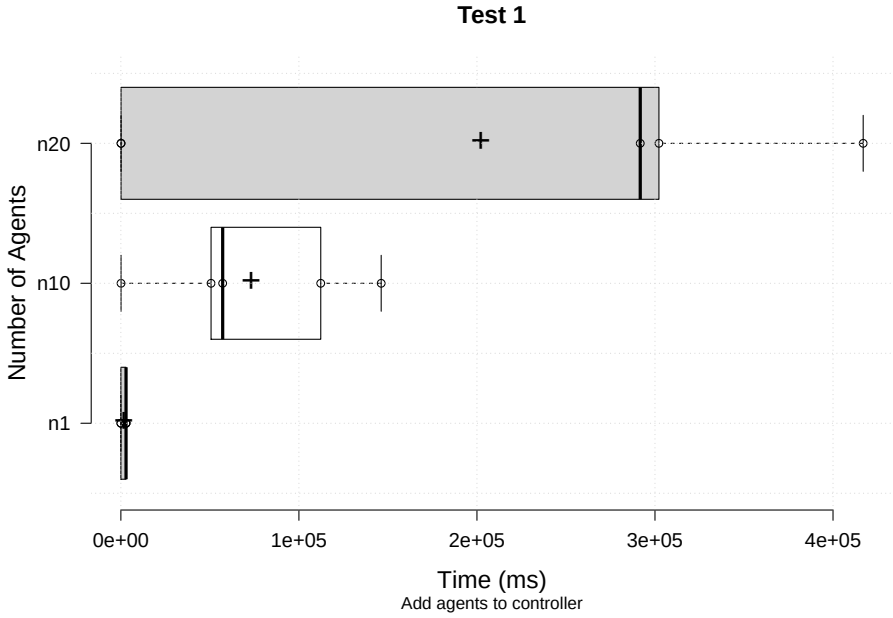


FIGURE 5.5: Test 1 - box plot 5 runs

### Test 2 - Remove all added agents from the controller

In this second test, the results had two problematic runs, in the test case for 10 agents. That led to this problematic Figure 5.6 where the mean says we had approximately the same time to remove 10 and 20 agents. This problem is related to a problematic client shutdown that took longer than expected.

In Figure 5.7 we can distinguish the two problematic runs that led to this unexpected result, but we can also see the other 3 runs with similar times. In this specific case (10 agents) we had an average time of 285 ms and a standard deviation of 217 ms meaning there were runs with almost twice the time than others.

The reason why these global results are so fast when compared to the ones in test one is that the controller does not actually remove the clients from the queue only change its status to off leading to faster shutdown times and faster reconnects.

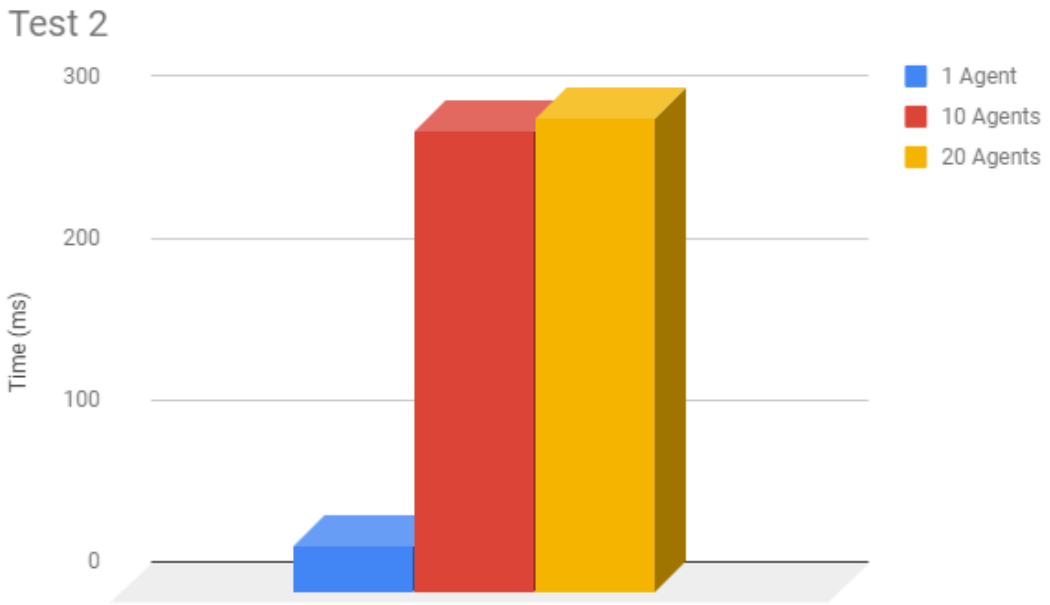


FIGURE 5.6: Test 2 - bar chart with results average

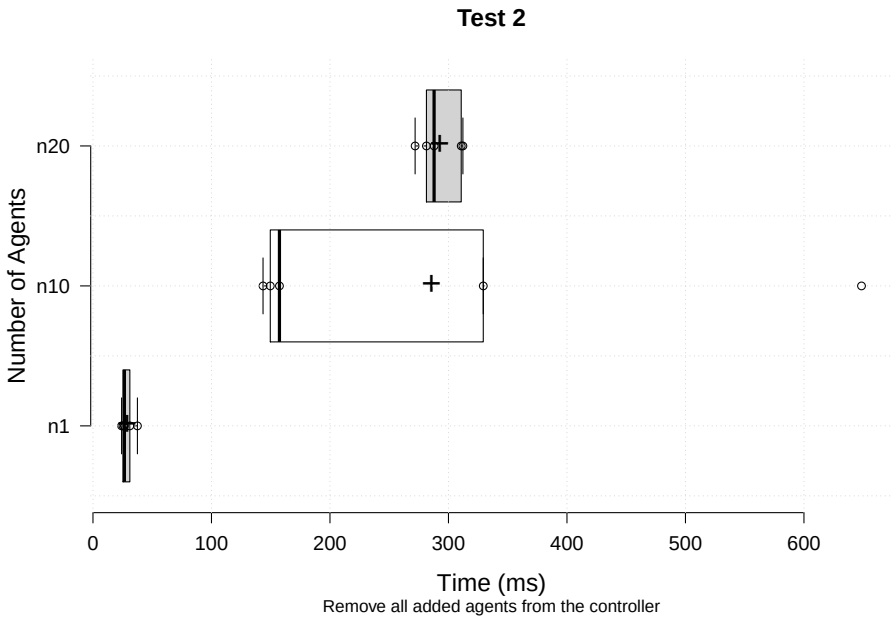


FIGURE 5.7: Test 2 - box plot 5 runs

### Test 3 - Controller failover

The third test also had an expected behaviour, since we stopped the controller and started it once again. This process stuck the clients until there was a controller status change. When that occurs, the clients resend the registration request to the controller. According to the results (see Figure 5.9) in this test the deviation from

the average result was inferior to 10% of the average thus meaning there was no exaggerated differences between results.

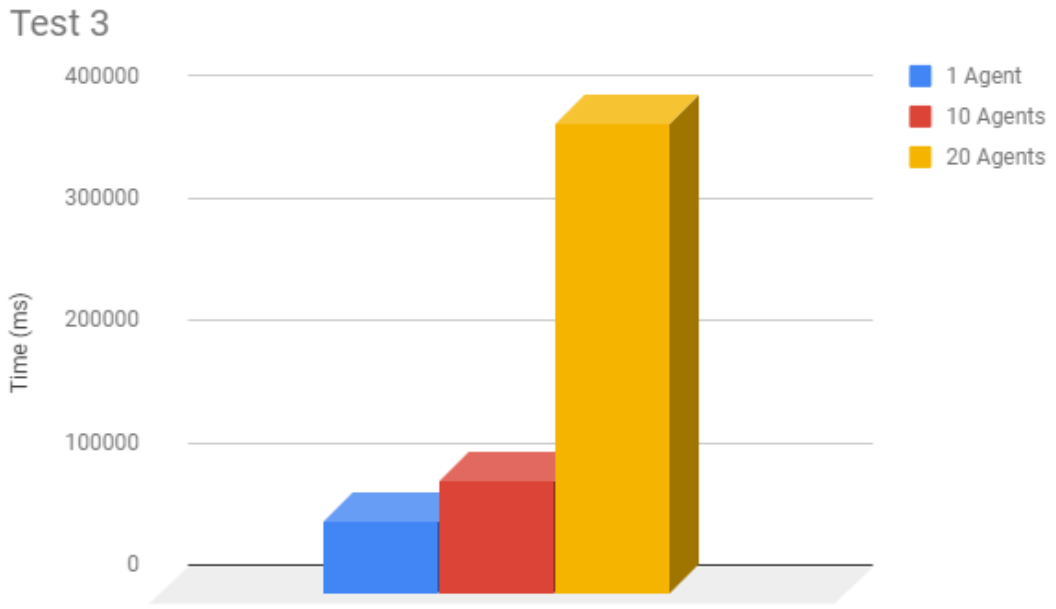


FIGURE 5.8: Test 3 - bar chart with results average

Besides, there was an expected growth between test cases "1, 10 and 20", this growth is exponential, which may be related once again with the chosen test scenario.

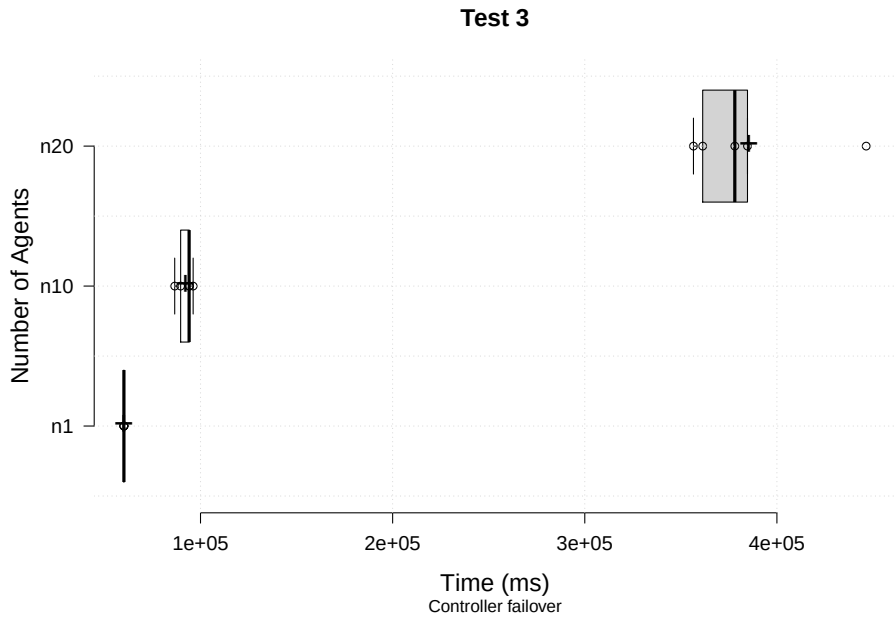


FIGURE 5.9: Test 3 - box plot 5 runs

### Test 4 - Change an agent configuration (local configuration change)

In this test we changed the configuration of the agent without recurring to the management system, in other words, we changed the configuration locally and waited for that change to be detected by the management system. This test (Figure 5.11) was expected to have no difference among test cases since only one of the clients is performing operations

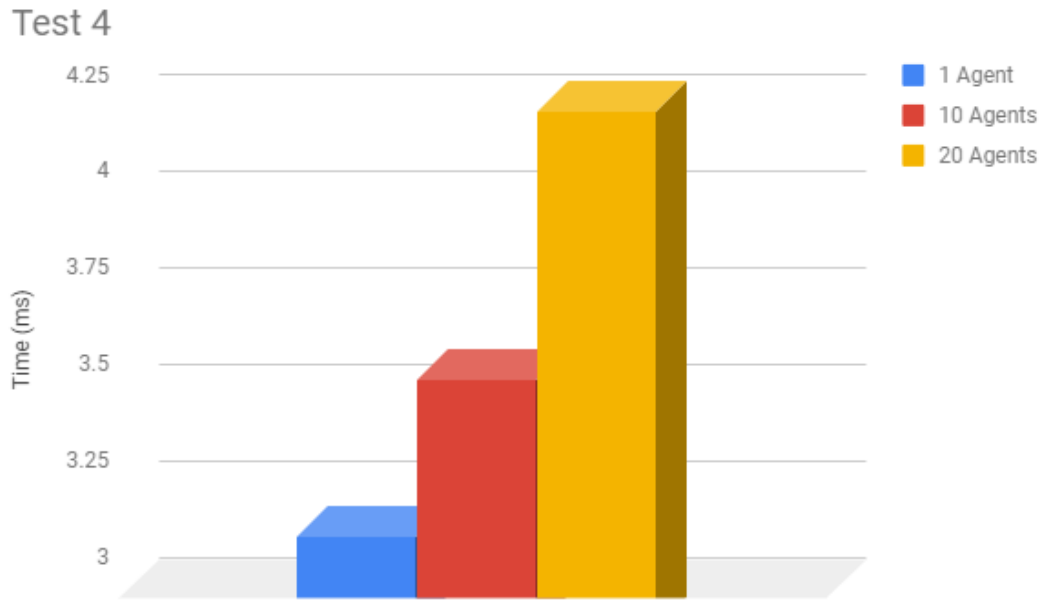


FIGURE 5.10: Test 4 - bar chart with results average

In both Figures 5.11 and 5.11 it appears to exist a huge difference among runs however we are only talking about 2ms of maximum difference among all runs. Therefore the difference between test cases is almost null.

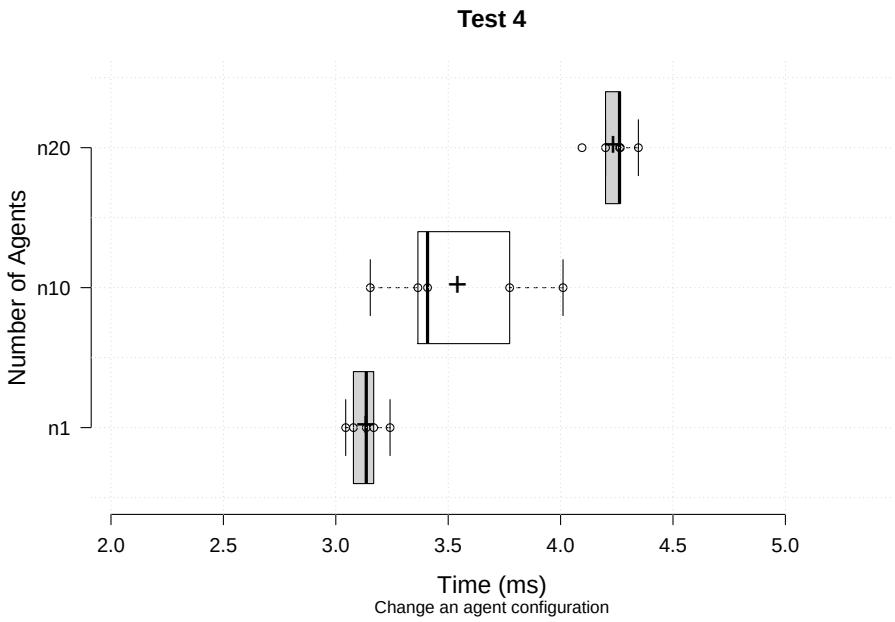


FIGURE 5.11: Test 4 - box plot 5 runs

### Test 5 - Broker failover

Test 5 is the last of the management tests performed using bash scripts and it is also the first test where the results didn't go as expected. Despite the fact the agents performed as expected, there was an unforeseen issue with the test execution. In Figure 5.12 we can see that there was almost no difference between the test cases. But as we explained in last section this test has 3 important steps: stop the broker, wait 120 seconds and start the broker. The problem here is that the 120 seconds wait was counted. Therefore, according to Figure 5.13 there was a maximum deviation among all runs of nearly 100ms, which is unrealistic.

There is an explanation for this unexpected behaviour, it is related to the waiting time chosen for this test. We decided to go with 120 seconds because it is the defined MQTT keepalive (recurrent operation that assures that the connection is open and both client and broker are connected) value. According to the specification, the keepalive was not able to reach the other end and consequently, the client should try to reconnect. The real problem here was that the agents didn't have time for the keepalive to be triggered, and since there was no operation between the broker stop and start, the agent didn't notice the broker went gone, consequently they recovered without any problem.

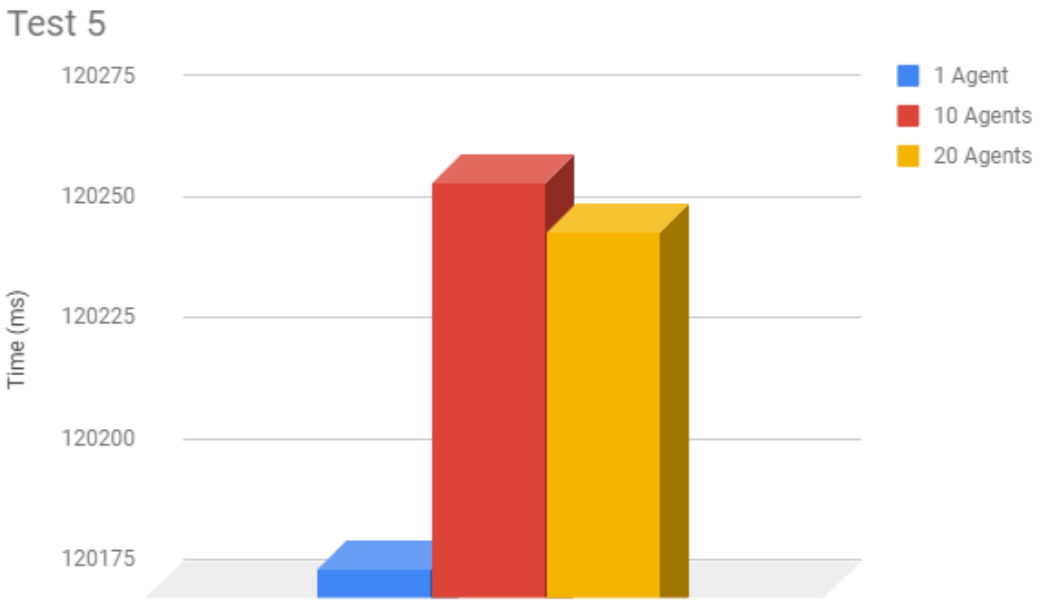


FIGURE 5.12: Test 5 - bar chart with results average

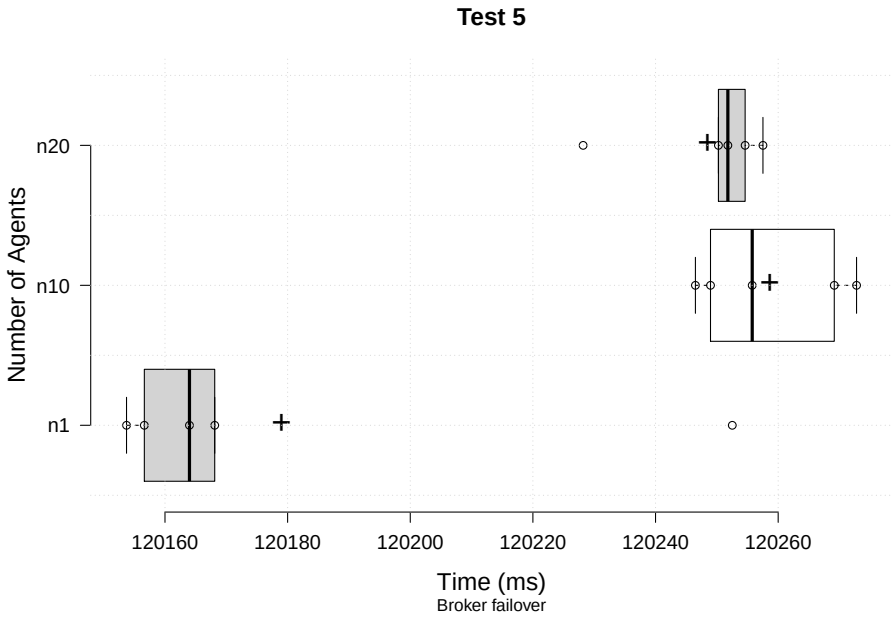


FIGURE 5.13: Test 5 - box plot 5 runs

### Test 6 - Does the controller parse the message payload (JSON) correctly (multiple parameters)?

The sixth test marks (Figure 5.14) the change of the used scenario in the management tests and consequently the drop of scripting in favour of the Postman software. Postman allowed us to test the API and understand which operation took more time

to perform.

In this test, we could clearly see that the execution operations took longer than the other operations of reading and changing multiple configurations, which is somehow expected since we need to wait for each execution to finish before retrieving the results. The time each operation took in this test is not only acceptable but can be considered to be a good result since every one took less than 5 seconds to execute, which is the time a user would take only to perform a login in the remote machine.

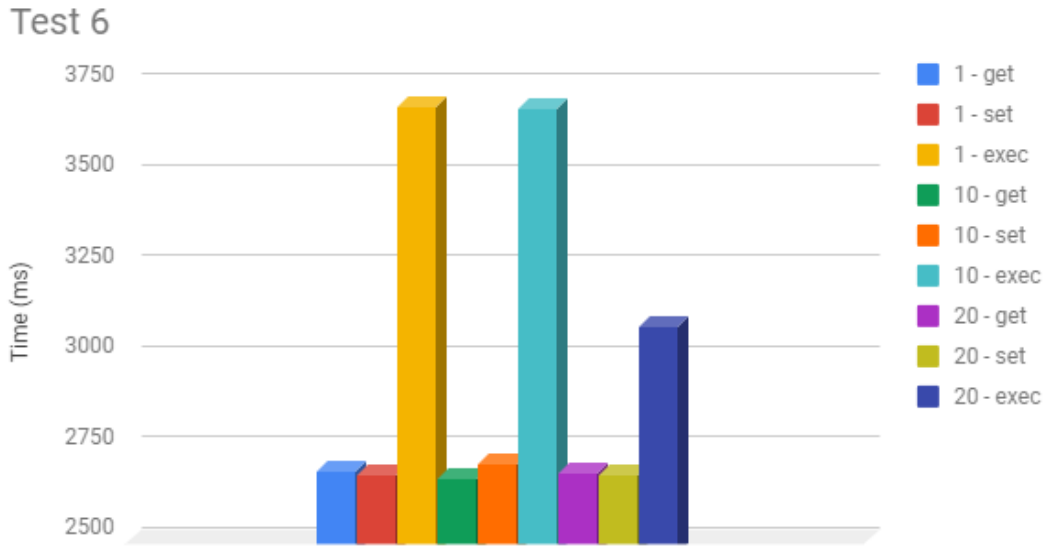


FIGURE 5.14: Test 6 - bar chart with results average

Figure 5.15 shows that they were 3 test cases where the results somehow diverged [set - 10 agents, execute - 10 agents and execute 20 - agents]. These results can be explained by some hardware and network overhead that occurred during the tests.

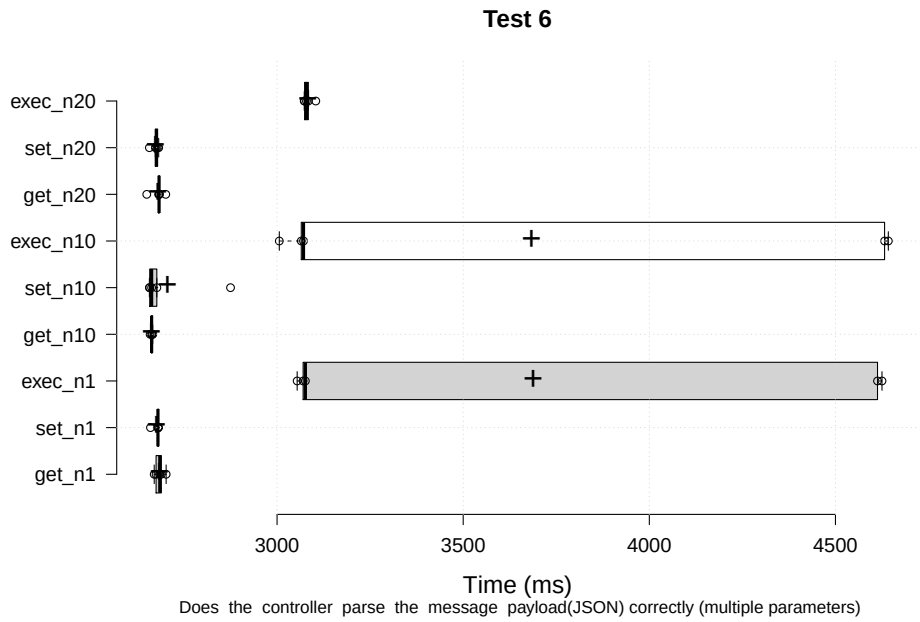


FIGURE 5.15: Test 6 - box plot 5 runs

### Test 7 - Does the controller parse the message payload JSON correctly (one parameter)?

In this test we can confirm the expected stable behaviour obtained in the above test, with a maximum standard deviation (Figure 5.17) of 88 ms among runs of the same type and getting a maximum difference between all run of 200ms each represents a deviation of 10% between the minimum value and the max value which is acceptable as well.

The difference presented in Figure 5.16 among test types is not stable, in other words, in the test cases with 10 and 20 agents the execution time was higher, but in the test with only one agent, the set parameter operation was the longer one.

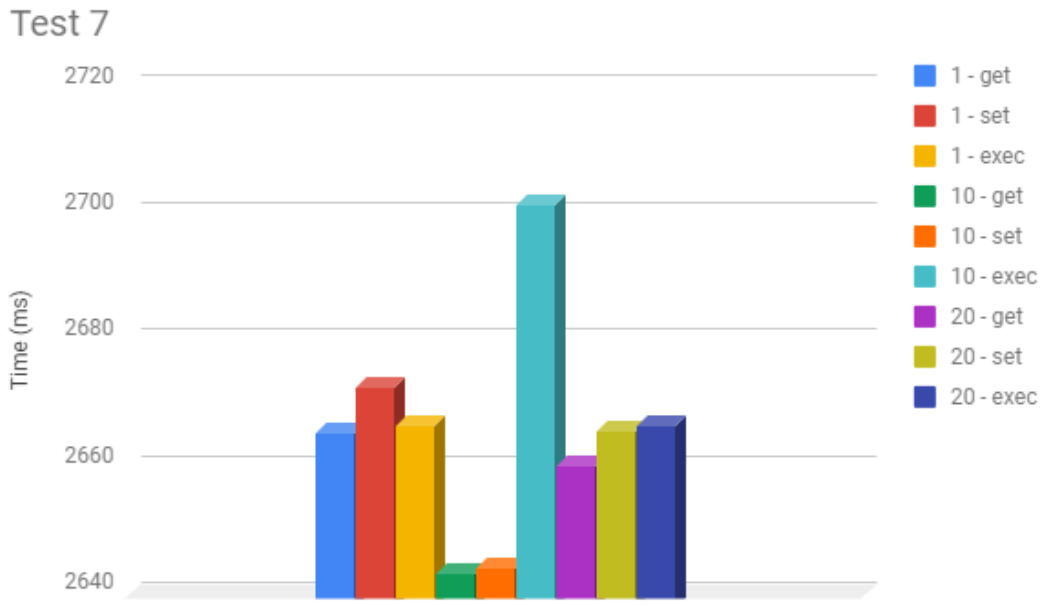


FIGURE 5.16: Test 7 - bar chart with results average

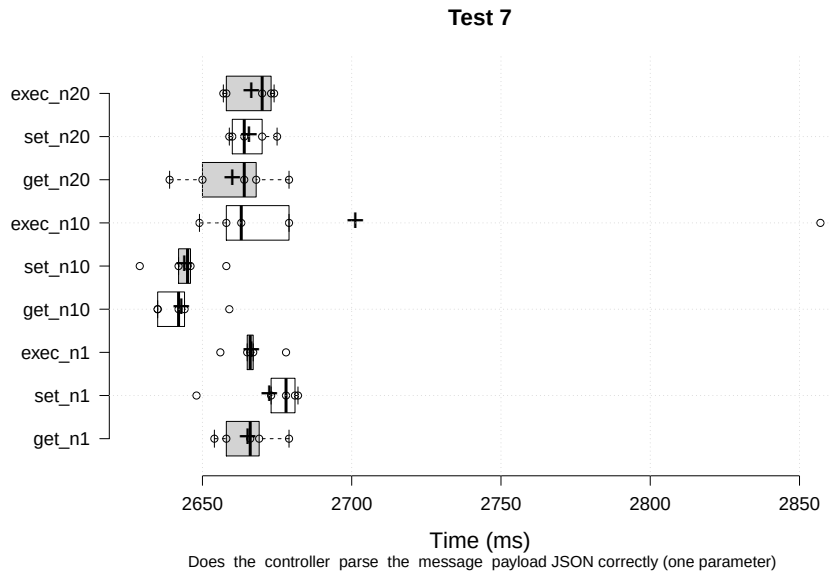


FIGURE 5.17: Test 7 - box plot 5 runs

### Test 8 - Does it handle a request to an unknown agent

This test might raise some questions about its relevance, especially when compared with tests 6 and 7. Eventually, the reader might be questioning about the time differences between such tests and this one. There is a simple explanation for that: in the other two tests the HTTP client does a request that needs to go to the controller

after retrieving the information from the client, returning it to the controller and finally send it to the HTTP client. In test 8 what happens it that the message arrives to the controller, which rejects it since the client is not registered, reducing the response times as visible in Figure 5.18.

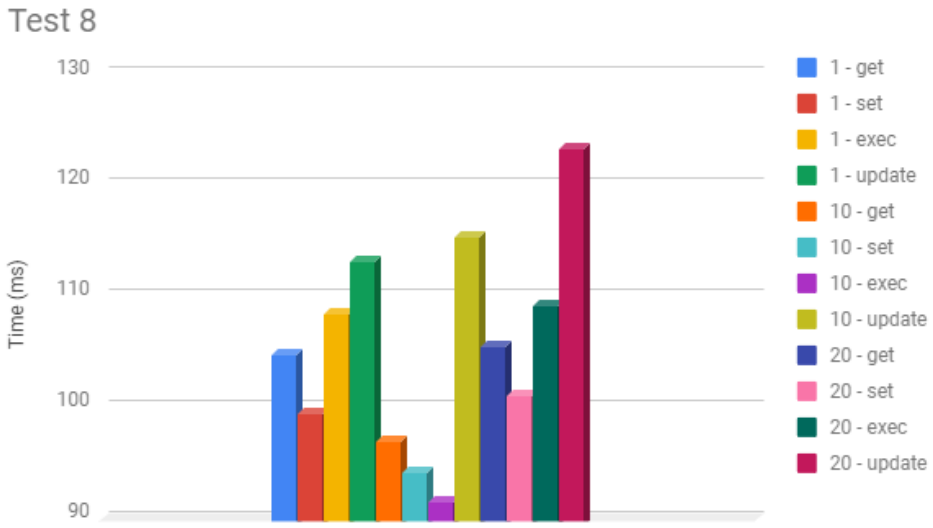


FIGURE 5.18: Test 8 - bar chart with results average

Once again the differences visible in Figure 5.19 are mostly related to the used scale and the differences between runs are virtually none.

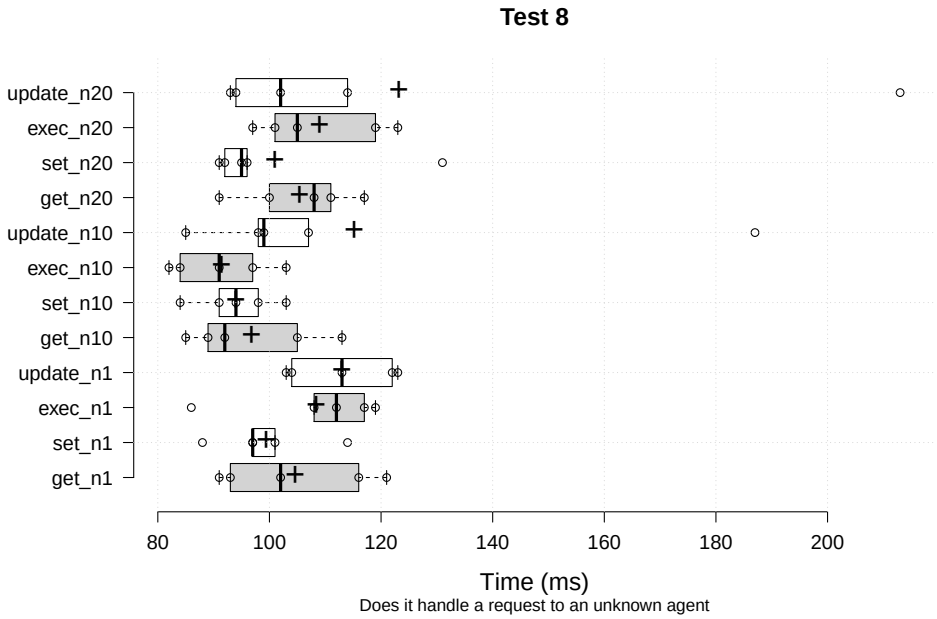


FIGURE 5.19: Test 8 - box plot 5 runs

### Test 9 - Test invalid configuration types in valid request

In this test(Figure 5.21), we measured the time it took for the agents to refuse a request with a parameter that can not perform the requested operation. Once again there were no unexpected results, the tests behaved stably and the results went as expected. There were, however, some outliers, but once again the standard deviation does not represent even 10% of the average level and this deviation can be ignored.

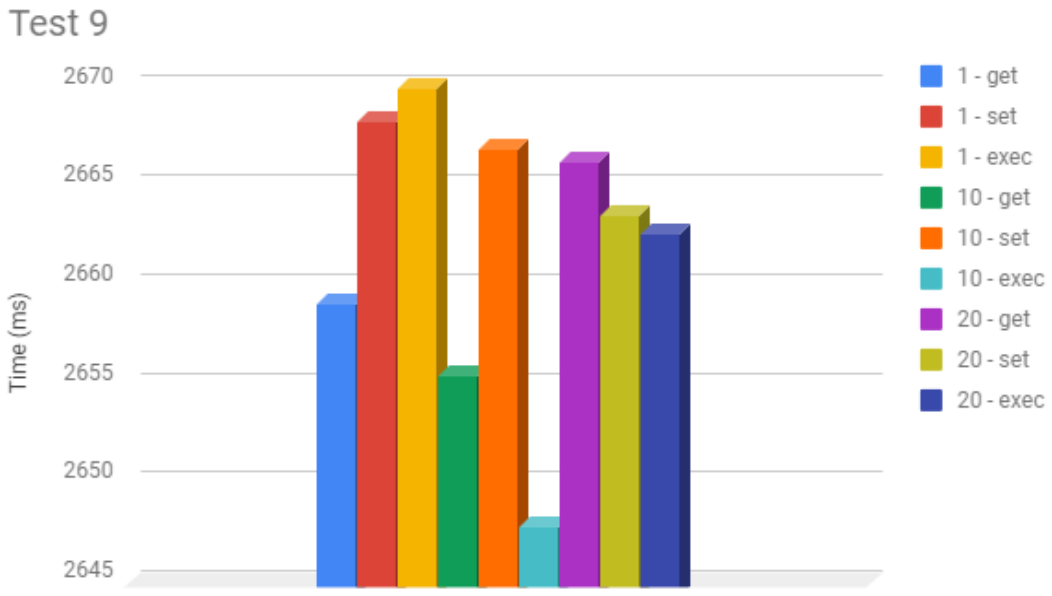


FIGURE 5.20: Test 9 - bar chart with results average

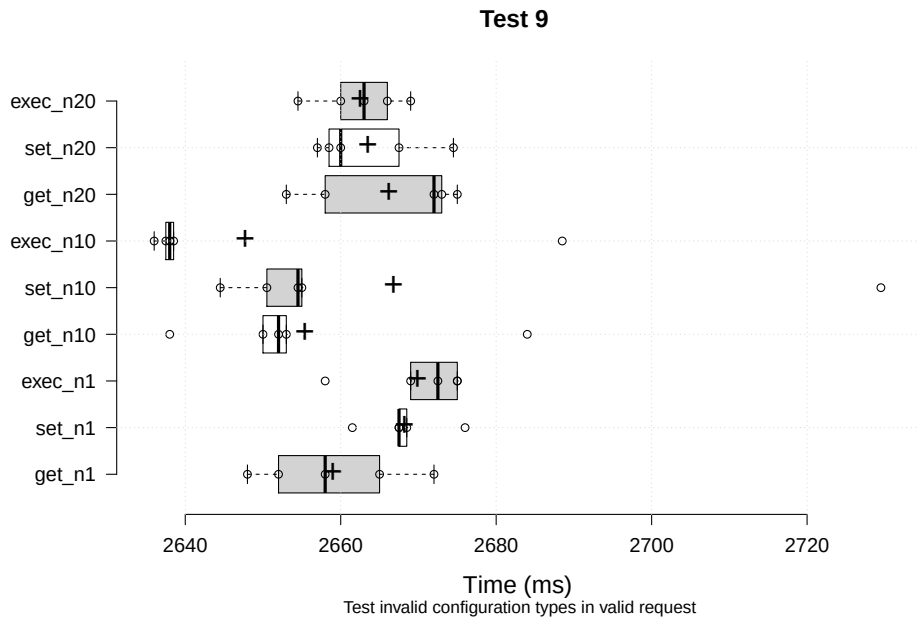


FIGURE 5.21: Test 9 - box plot 5 runs

### Test 10 - Test invalid configuration types in valid request

The tenth test (Figure 5.23) is similar with the ninth but there is a significant difference between them, since the parameters sent in the test payload do not exist in the manifest, forcing the agent to check every saved parameter of the manifest to ensure it does not exist. Although this test maintained the request time below 5 seconds, the set operation took more time than the other two test cases to be performed and it was an unexpected behaviour that we are unaware of the cause.

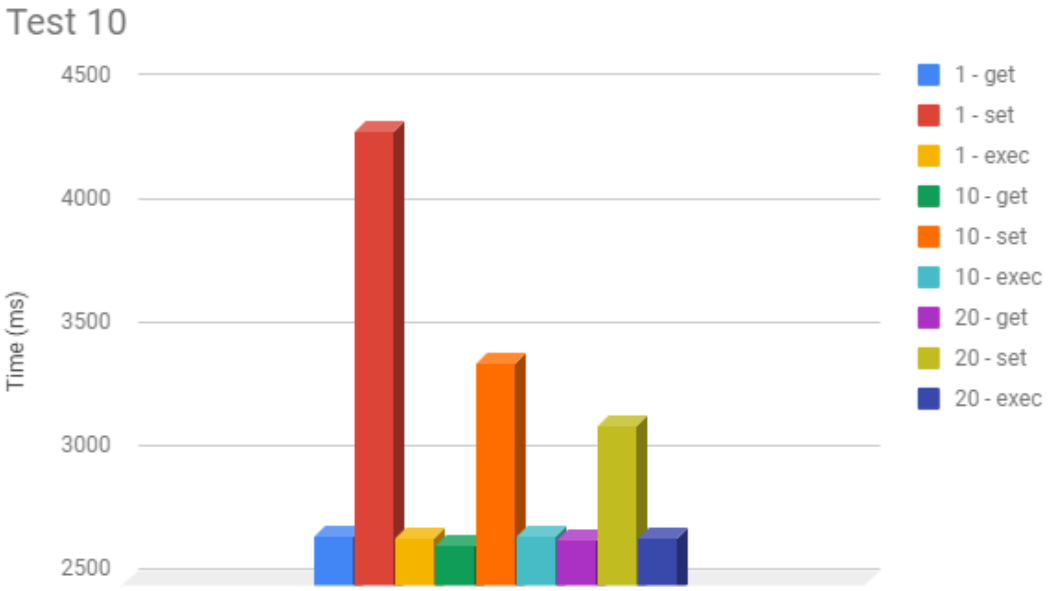


FIGURE 5.22: Test 10 - bar chart with results average

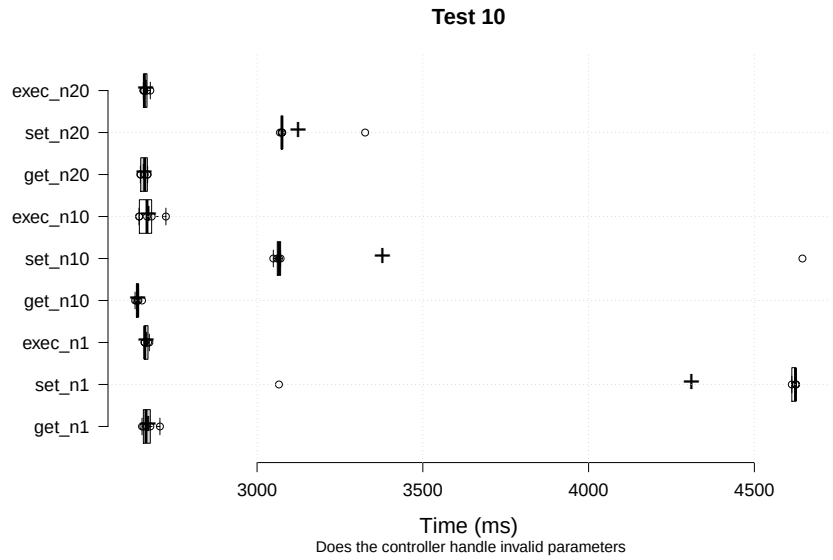


FIGURE 5.23: Test 10 - box plot 5 runs

### Test 11 - Test invalid configuration types in valid request

Test 11 (Figure 5.25) is possibly the one with strangest results, however, they make sense, but only after a brief explanation of what happened. For each run we made 3 tests to ensure most of the possibilities were covered. In these 3 tests, there were two outputs, the result 404 (taking about 100ms) and the 500 result (taking about 20 seconds to timeout). In the three test cases [get, set, exec] there were two situations: the payloads used in get where two of the results gave the response of

500 and one 404, while the set and exec operations had two results 404 and one 500. Briefly, this means that the run values are the average of this values and that is the reason why get has twice the runtime of the other tests.

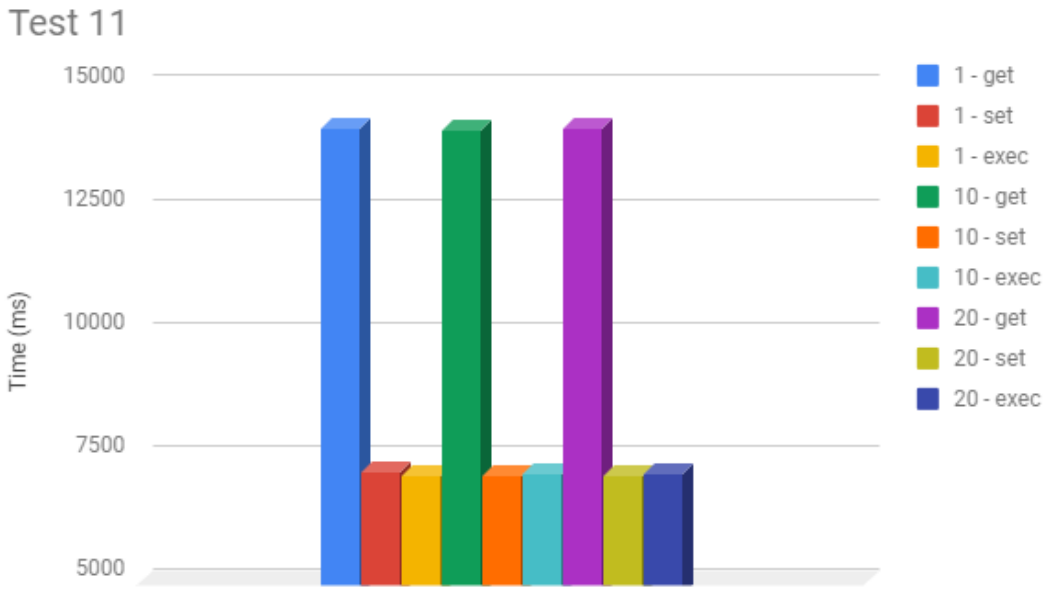


FIGURE 5.24: Test 11 - bar chart with results average

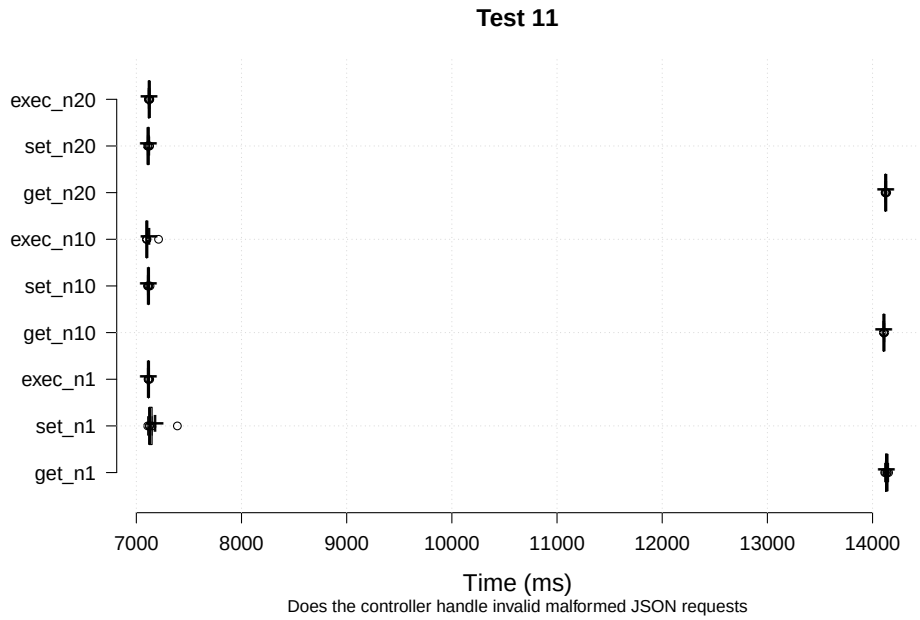


FIGURE 5.25: Test 11 - box plot 5 runs

### Test 12 - Does the controller handle malformed url

Just like test 8, this test (Figure 5.27) is faster than the usual requests, the reason for that is the controller automatically rejects the invalid request without needing to communicate with the agents. The overall results of this test were precise and there was no major deviation between runs.

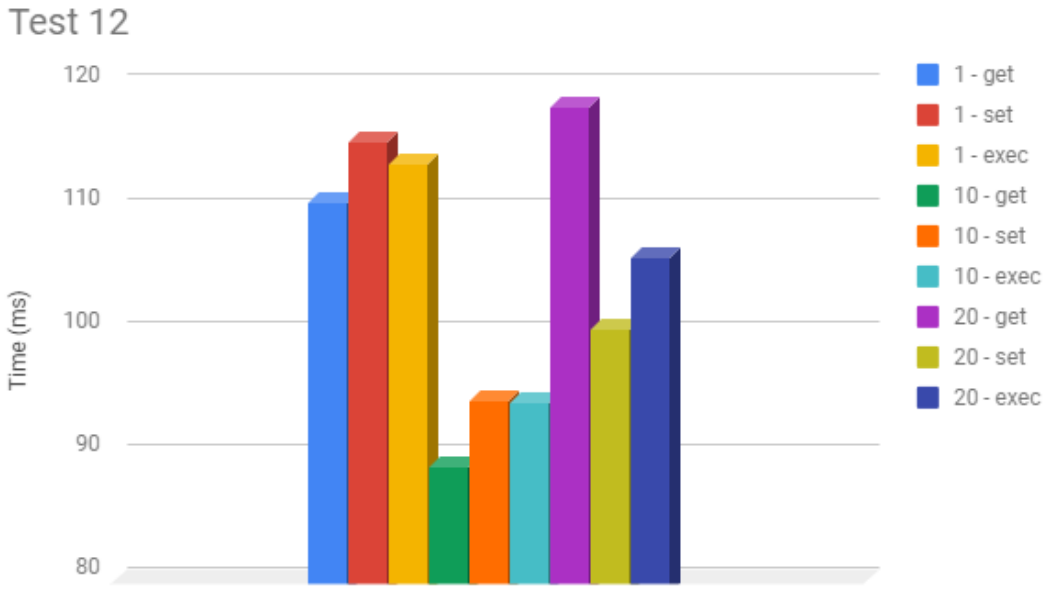


FIGURE 5.26: Test 12 - bar chart with results average

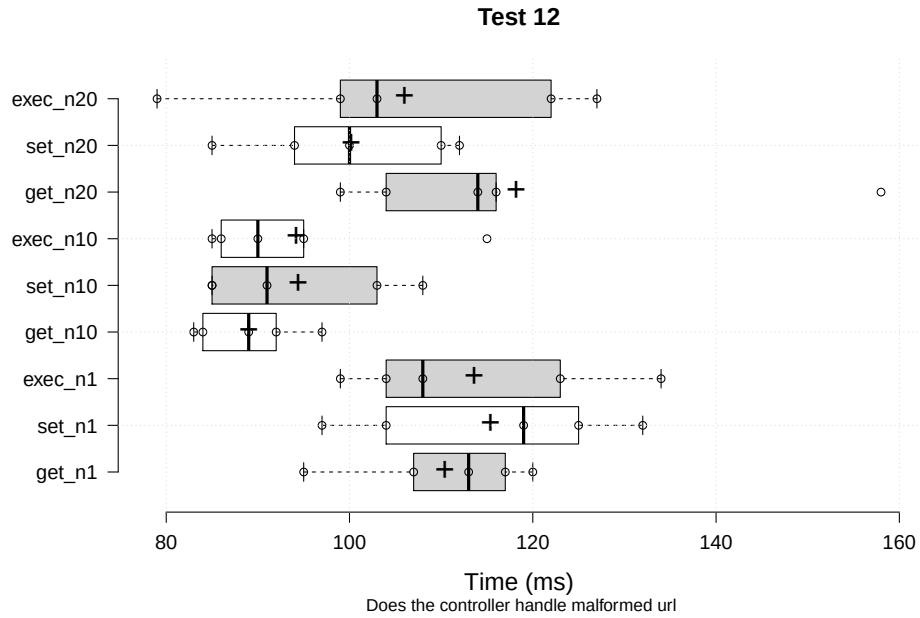


FIGURE 5.27: Test 12 - box plot 5 runs

### Test 13 - List agents

The last of the management tests (Figure 5.29) is just like tests 12 and 8, but it is only communicating with the controller, therefore the runtime results are faster than the global test results. Once again there were no unexpected values and the standard deviation was once again below 10% of the average value.

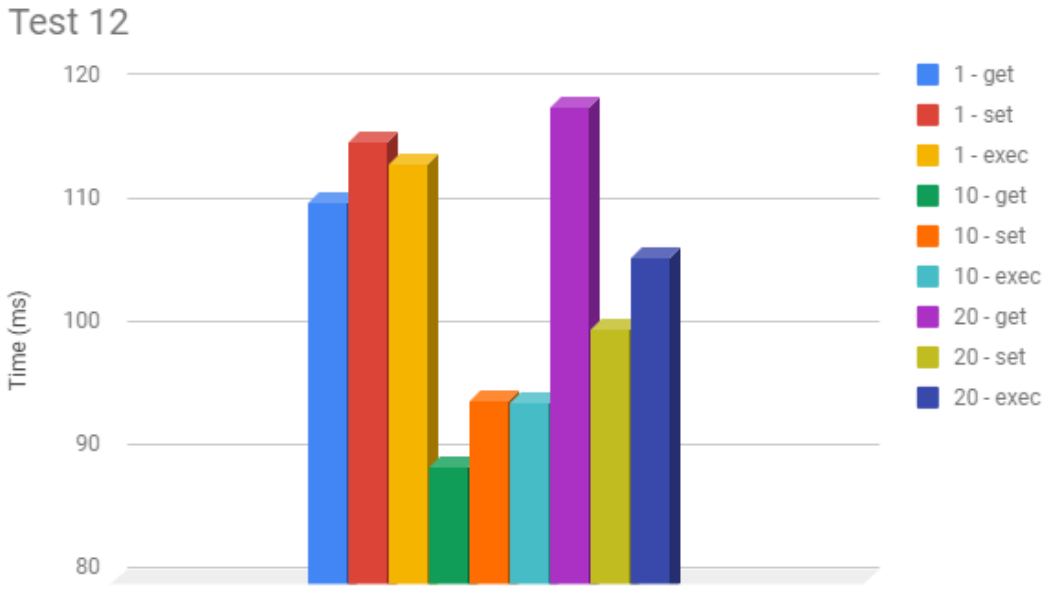


FIGURE 5.28: Test 13 - bar chart with results average

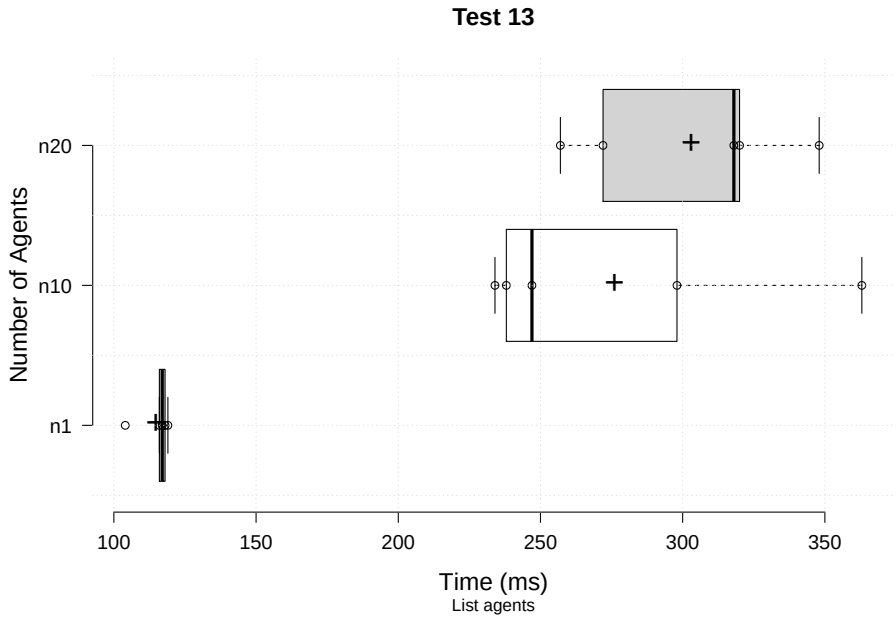


FIGURE 5.29: Test 13 - box plot 5 runs

### Test 14

Although test 14 was planned we could not perform it. However, the event solution was tested and used during the demonstration of the work package 4 in the project review mentioned in Chapter 6.

# Chapter 6

# Work Plan

This chapter describes the followed work plan and its evolution over the first and second semester.

## Work plan

To plan this thesis was necessary to have in mind the objectives and tasks already defined in Chapter 1. The first task to take care when defining the working schedule was to define the relevant deadlines in the thesis, therefore, we have the following milestones:

- **Milestone 1** (14/11/2017) - The first project review. The ATENA working team had to make a demonstration of the IADS platform (in the European Commission) and was mandatory to have a working prototype of the platform, therefore it was expected to have a working product from this thesis (a NIDS/virtual **N**etwork **I**ntrusion **D**etection **S**ystem (vNIDS) integration, explained in Chapter 4)
- **Milestone 2** (23/01/2018) - Intermediate report delivery
- **Milestone 3** (02/07/2018) - Final report delivery

Having the milestones defined, we automatically define 3 intervals or groups of tasks:

- **Group 1** - Tasks before the first milestone, these tasks were related to the development of the generic agent
- **Group 2** - Tasks between the first and second milestones, these tasks were related to the writing of the report
- **Group 3** - Tasks between the second and third milestones, these tasks will be related to the development of the management solution and completion of the report

Finally, inside those groups are particular tasks:

- **Group 1**
  - **Task 1** - Introduction to the ATENA Project and IADS architecture. Understanding the requirements for the agents and defining the agent architecture.
  - **Task 2** - First stage of implementation, configuration and deployment of the planned agent for the demonstration.

- **Group 2**
  - **Task 3** - Familiarisation with the state of the art (Management and lightweight messaging protocols), writing of the LwM2M section (used in the documentation of the ATENA project - Deliverable D4.3)
  - **Task 4** - Writing the first version of the report
- **Group 3**
  - **Task 5** - Define the management architecture and identifying and deciding the technologies to use
  - **Task 6** - The second stage of implementation, developing the management solution and integrating the developed agents in it.
  - **Task 7** - Testing and validating the implemented management solution and agents
  - **Task 8** - Writing the final report

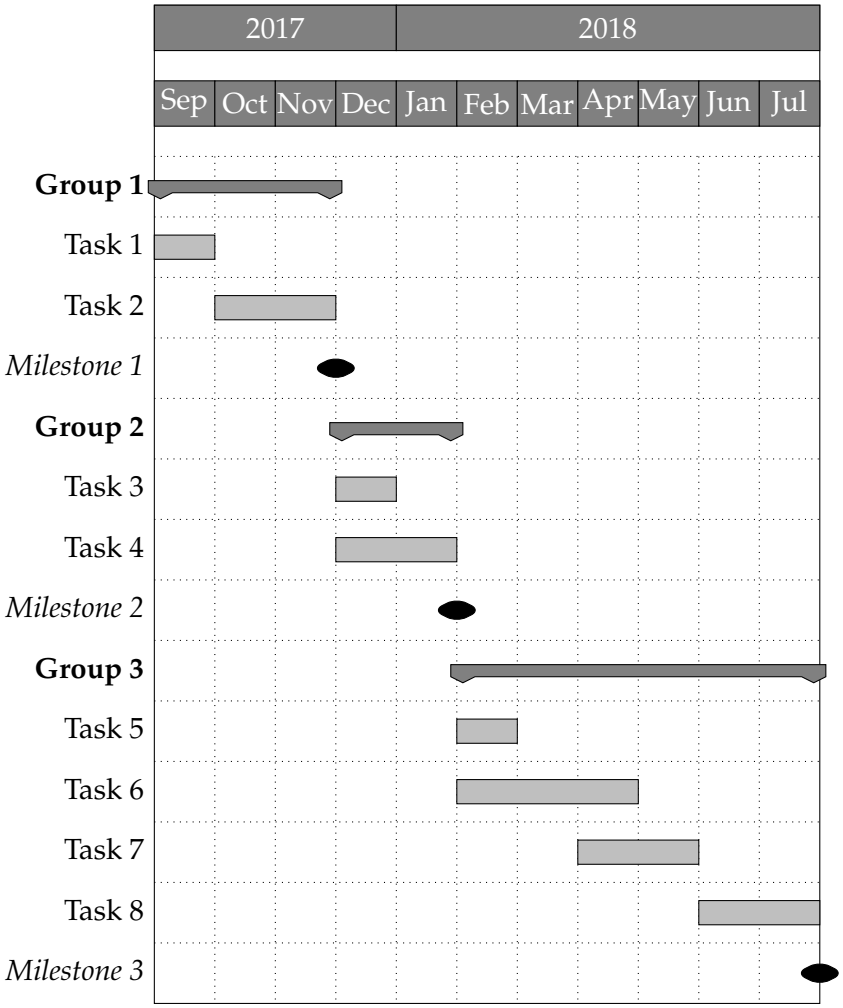


FIGURE 6.1: Gantt Chart

After concluding the planned tasks we can easily identify some planning deviations across the second semester. Most of these unexpected changes occurred due to the decision of developing a solution from scratch instead of using some predeveloped management solution. The original Tasks 6 and 7 were the ones affected.

Although Task 6 started on the planned date it took more time than expected leading to a big deviation and consequently a reduction of available time for testing.

We also added the **Task 9** that was not in the previous planning that corresponds to the time spent in contributions (Deliverables D4.4 and D4.5 referred in Chapter 7).

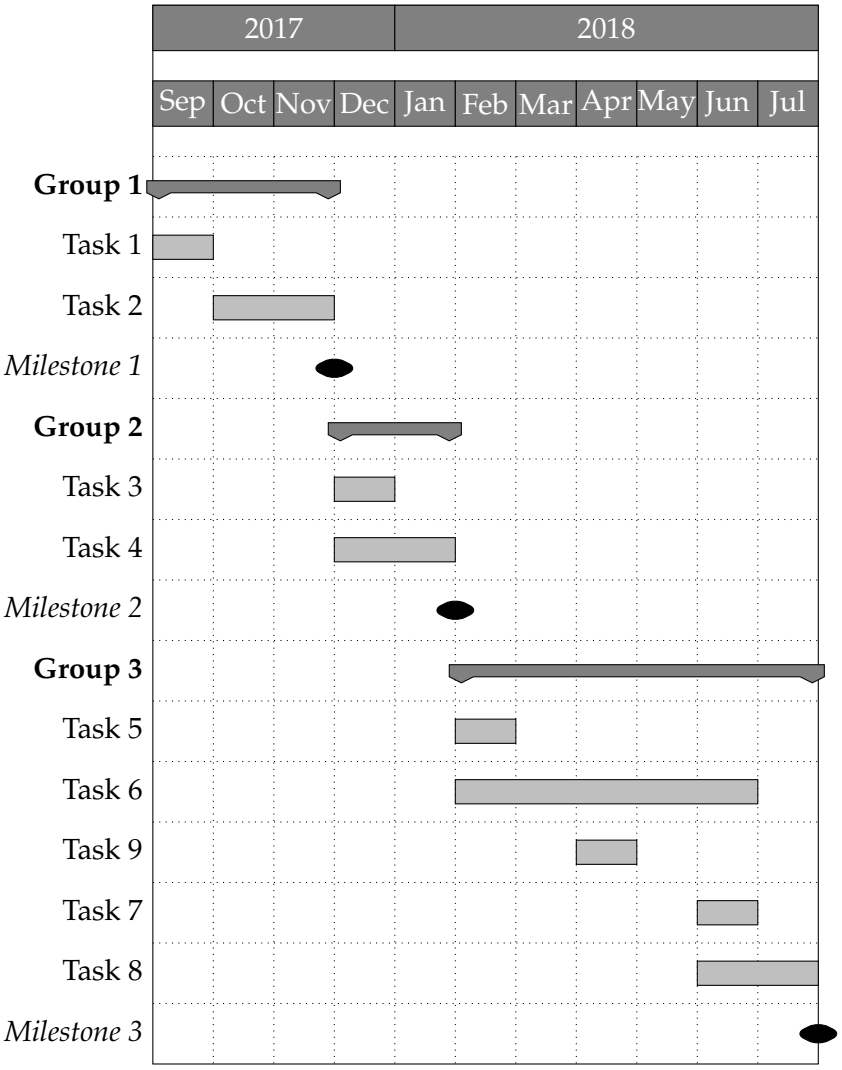


FIGURE 6.2: Gantt Chart updated

# Chapter 7

# Conclusions

The present document covered the proposed agent architecture, combining a management solution with a data collector solution. Since the main purpose of this component is to manage and monitor devices, the presented State-of-the-Art focused on the evolution of monitoring and management solutions, protocols and tools. We also introduced some of the scenarios where these tools can be used and why they need to be used.

The analysis of the State-of-the-Art leads to some conclusions. Until now there were three feasible ways to develop a management solution: 1) usage of standalone MQTT, together with topics to control the parameters; 2) usage of LWM2M (based on CoAP) which has defined objects and functions to control the agents. 3) usage of an implementation of LWM2M over MQTT, with the same methodology to control the agents but resorts to MQTT as the data protocol.

From the presented options we decided to implement our own protocol using standalone MQTT since it proved to be the solution with better documentation that fulfilled our requirements. This solution as well as the protocol was described in Chapter 6 and is already integrated with the IADS.

In addition, we also planned and designed a generic architecture for some probes, with an existing data collecting agent (NIDS and vNIDS) that proved to be efficient and reliable during the performed tests.

## 7.1 Contributions

During the involvement in the ATENA project, some contributions were made. They were made to both ATENA project documentation and to the first work package 4 demonstration (Project Review). The project review contribution was explained during the document and corresponds to the event collector developed. When it comes to the project documentation we made contributions in the deliverable D4.3 the Chapter 7, where we explained the agent management API. We also made contributions in the deliverable D4.4, Chapter 6 - management and monitoring framework - we contributed in the overview (6.1) section explaining the components of this framework, in the Application and Probe integration (6.2) we explained the management architecture, in Component Management (6.3) we explained how the protocol works and on the Middleware - Agents (6.5.2 and 6.6.2) we explained the API endpoints the requests they handle and what they respond. Finally, in the deliverable D4.5 we made a contribution to Chapter 4, Security Agent/Probe Management (4.7.1) section, where we explained why we based our management solution on the MQTT protocol.

## 7.2 Future Work

The work here presented will be continued, namely the data collector - the events part in the agent architecture - since there are multiple data model types. Although the management solution is already integrated with the platform and all the requirements implemented, there are some features that could enhance its performance and its management scope, for instance adding the capability of managing operating system environment variables, create and expose new endpoints allowing reverting the configurations to older versions, listing old agents or even old configurations. Another possibility is the creation of a websocket allowing real-time monitoring without any kind of request from the HTTP client.

# Appendix A

# Requirements

## Probes Management – Functional Requirements

The following requirements were specified before my integration in the ATENA team and the majority of the work is acknowledged to Miguel Freitas, Leonardo Toledo and the rest of the team. I have reviewed the requirements and made some changes.

| | |
|---|---|
| Use Case | MP_FR1: Provisioning a Probe Component |
| Primary Actor | System Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in provisioning a new probe component<br>- Security Admin: interest in being able to perform actions that require a probe component |
| Pre-Conditions | - The website is available<br>- Actor must be authenticated as System Admin<br>- The probe component must exist |
| Last Review | 20-12-2017 |
| Minimum Guarantees | The probe component is provisioned |
| Success Guarantees | The probe component is provisioned |
| Trigger | A need for a probe deployment exists |
| Process - Main Success Scenario | - 1: A need for a probe deployment exists<br>- 2: The system admin assures all the provisioning needs for the future probe component deployment. This might include:<br>a) Network configurations for the probe deployment (e.g., reserve an IP Address)<br>b) Initial configurations for a probe to be securely detected by the system<br>c) Hardware resources<br>d) Unique identifiers, secure parameters or any other parameter that unequivocally identifies the probe to ensure an only-manual deployable probe can be securely deployed<br>- 3: The probe component is provisioned |
| Exceptions | - 2:<br>a) The actor does not provide the correct provisioning needs for the probe component<br>a1) The use case ends and the probe cannot be deployed later |

Table 1: Use Case MP_FR1: Provisioning a Probe Component

| | |
|---|---|
| Use Case | MP_FR2: Deploy a Probe Component |
| Primary Actor | System Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in deploy a probe component<br>- Security Admin: interest in being able to perform actions that require a probe component |
| Pre-Conditions | - The website is available<br>- Actor must be authenticated as System Admin<br>- A probe was provisioned via use case MP_FR1 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A probe component is deployed |
| Trigger | The actor clicks on the "Probes" menu item on the platform |
| Process - Main Success Scenario | -1: The actor clicks on the "Probes" menu item on the platform<br>-2: The system presents the actor the following sub-menu items:<br>a) Manual Deployment<br>b) List probes<br>- 3: The actor clicks in one of the two first sub-menu items so he can deploy a probe. These probes can be deployed:<br>a) Manually via use case MP_FR10 |
| Exceptions | - |

Table 2: Use Case MP_FR2: Deploy a Probe Component

| | |
|---|---|
| Use Case | MP_FR3: Enable a Probe Component |
| Primary Actor | System Admin |
| Secondary Actors | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in enable a probe component<br>- Security Admin: interest in enable a probe component |
| Pre-Conditions | - The website is available<br>- The actor must be authenticated as System Admin or Security Admin<br>- The actor must have completed use case MP_FR6 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A probe component is enabled |
| Trigger | Actor selects the probe he wants to enable in the list of probes |
| Process - Main Success Scenario | - 1: Actor selects the probe he wants to enable in the list of probes<br>- 2: Actor clicks on "Enable" and a confirmation modal is opened<br>- 3: Actor confirms the intent by clicking on "Yes"<br>- 4: The specific probe is enabled<br>- 5: A success message is displayed by the system |
| Exceptions | - 3:<br>a) The actor does not confirm the action by clicking "Yes"<br>a1) The use case ends<br>- 4:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged<br>c) An error occurred due to security checks failing (e.g., wrong signatures)<br>c1) The system presents a popup message "Could not deploy probe due to failing security checks. Please check the provided information" and the exception is logged |

Table 3: Use Case MP_FR3: Enable a Probe Component

| | |
|---|---|
| Use Case | MP_FR4: Configure a Probe Component |
| Primary Actor | System Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in configure a probe component<br>- Security Admin: interest in being able to perform actions that require a probe component |
| Pre-Conditions | - The website is available<br>- Actor must be authenticated as System Admin<br>- The probe was deployed via use case MP_FR2<br>- The actor has fulfilled use case MP_FR6 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A probe component is configured |
| Trigger | Actor selects the probe he wants to configure from the list |
| Process - Main Success Scenario | - 1: Actor selects the probe he wants to configure from the list<br>- 2: Actor clicks on "Configure a Probe Component"<br>- 3: The system presents the actor with the list of available probe configuration fields<br>- 4: The actor configures the probe by filling the configuration fields<br>- 5: The actor clicks on "Save" and a confirmation modal is opened<br>- 6: The actor confirms the intent by answering "Yes"<br>- 7: The probe is configured<br>- 8: A success message is presented by the system |
| Exceptions | - 4:<br>a) Actor does not fill all required fields<br>a1) The system presents a popup message "You must fill all required fields!" along with some visual information regarding the fields missing<br>- 6:<br>a) Actor does not confirm the action by clicking "Yes"<br>a1) The use case ends<br>- 7:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged<br>c) An error occurred due to security checks failing (e.g., wrong signatures)<br>c1) The system presents a popup message "Could not deploy probe due to failing security checks. Please check the provided information" and the exception is logged |

Table 4: Use Case MP_FR4: Configure a Probe Component

| Use Case | MP_FR5: Disable a Probe Component |
|---|---|
| Primary Actor | System Admin |
| Secondary Actors | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in disable a probe component<br>- Security Admin: interest in disable a probe component |
| Pre-Conditions | - The website is available<br>- The actor must be authenticated as System Admin or Security Admin<br>- The actor must have completed use case MP_FR6 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A probe component is disabled |
| Trigger | Actor selects the probe he wants to disable in the list of probes |
| Process - Main Success Scenario | - 1: Actor selects the probe he wants to disable in the list of probes<br>- 2: Actor clicks on "Disable" and a confirmation modal is opened<br>- 3: Actor confirms the intent by clicking on "Yes"<br>- 4: The specific probe is disabled<br>- 5: A success message is displayed by the system |
| Exceptions | - 3:<br>a) The actor does not confirm the action by clicking "Yes"<br>a1) The use case ends<br>- 4:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged<br>c) An error occurred due to security checks failing (e.g., wrong signatures)<br>c1) The system presents a popup message "Could not deploy probe due to failing security checks. Please check the provided information" and the exception is logged |

Table 5: Use Case MP_FR5: Disable a Probe Component

| | |
|---|---|
| Use Case | MP_FR6: List Probes Components |
| Primary Actor | System Admin |
| Secondary Actor | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in view all existent probes<br>- Security Admin: interest in view all existent probes |
| Pre-Conditions | - The website is available<br>- Actor must be authenticated as System Admin or Security Admin |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | View all existent probes |
| Trigger | Actor clicks on "List Probes" |
| Process - Main Success Scenario | - 1: Actor clicks on "List Probes"<br>- 2: The system provides a list containing all the available probes |
| Exceptions | 2:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again later!" and the exception is logged<br>a2) The use case ends |

Table 6: Use Case MP_FR6: List Probes Components

| Use Case | MP_FR7: Remove a Probe Component |
|---|---|
| Primary Actor | System Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in remove a probe component<br>- Security Admin: interest in being able to perform actions that require a probe component |
| Pre-Conditions | - The website is available<br>- Actor must be authenticated as System Admin<br>- The actor has fulfilled use case MP_FR6 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A probe component is removed |
| Trigger | Actor selects the probe he wants to remove from the list |
| Process - Main Success Scenario | - 1: Actor selects the probe he wants to remove from the list<br>- 2: Actor clicks on "Remove Probe" and a confirmation modal is opened<br>- 3: Actor confirms the action by clicking "Yes"<br>- 4: The specific probe is primarily disable<br>- 5: The specific probe is removed<br>- 6: A success message is displayed by the system |
| Exceptions | - 3:<br>a) Actor does not confirm the action by clicking "Yes"<br>a1) The use case ends<br>- 4, 5:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged |

Table 7: Use Case MP_FR7: Remove a Probe Component

| Use Case | MP_FR8: Manage Rules |
|---|---|
| Primary Actor | System Admin |
| Secondary Actor | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in managing the rules for a probe<br>- Security Admin: interest in managing the rules for a probe |
| Pre-Conditions | - The website is available<br>- Actor must have completed the use case MP_FR13 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | Actor is able to manage the rules for a specific probe |
| Trigger | Actor finds the probe he wants to manage its rules |
| Process - Main Success Scenario | - 1: Actor finds the probe he wants to manage its rules<br>- 2: The actor clicks on "Configure probe component"<br>- 3: The system presents the actor with the configuration view of the probe<br>- 4: The actor finds the "Rules section". In this section he can:<br>a) List all installed rules via use case MP_FR19<br>b) Create rules via use case MP_FR15<br>c) Modify rule via use case MP_FR17<br>d) Enable a rule via use case MP_FR16<br>e) Disable a rule via use case MP_FR18<br>f) Remove a rule via use case MP_FR20 |
| Exceptions | 3:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged |

Table 8: Use Case MP_FR8: Manage Rules

| | |
|---|---|
| Use Case | MP_FR9: Probe Status |
| Primary Actor | System Admin |
| Secondary Actors | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in view the probe status<br>- Security Admin: interest in view the probe status |
| Pre-Conditions | - The website is available<br>- Actor has completed use case MP_FR6 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | View the actual status of a specific probe |
| Trigger | Actor selects the probe he wants to check the status |
| Process - Main Success Scenario | - 1: Actor selects the probe he wants to check the status<br>- 2: The system displays all the status information about the specific probe |
| Exceptions | - 2:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged |

Table 9: Use Case MP_FR9: Probe Status

| Use Case | MP_FR10: Create Rule |
|---|---|
| Primary Actor | System Admin |
| Secondary Actor | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in create a new rule for a probe<br>- Security Admin: interest in create a new rule for a probe |
| Pre-Conditions | - The website is available<br>- Actor has completed use case MP_FR8 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A new rule for a probe is created |
| Trigger | Actor clicks on "Create Rule" |
| Process - Main Success Scenario | - 1: Actor clicks on "Create Rule"<br>- 2: Actor fills all required fields to create a rule for that specific probe<br>- 3: Actor clicks on "Save" and a confirmation modal is opened<br>- 4: Actor confirms the action by clicking "Yes"<br>- 5: The rule is created<br>- 6: A success message is displayed by the system |
| Exceptions | - 3, 5:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged<br>- 5:<br>a) The rule has the wrong syntax<br>a1) The system presents an error message "The rule syntax is wrong. Please check it." |

Table 10: Use Case MP_FR10: Create Rule

| Use Case | MP_FR11: Enable Rule |
|---|---|
| Primary Actor | System Admin |
| Secondary Actor | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in enable a rule for a probe<br>- Security Admin: interest in enable a rule for a probe |
| Pre-Conditions | - The website is available<br>- Actor has completed use case MP_FR8 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A rule for a specific probe is enabled |
| Trigger | Actor clicks on "Enable Rule" |
| Process - Main Success Scenario | - 1: Actor clicks on "Enable Rule" and a confirmation modal is opened<br>- 2: Actor confirms the action by clicking "Yes"<br>- 3: The rule is enabled<br>- 4: A success message is displayed by the system |
| Exceptions | - 3:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged |

Table 11: Use Case MP_FR11: Enable Rule

| Use Case | MP_FR12: Modify Rule |
|---|---|
| Primary Actor | System Admin |
| Secondary Actor | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in modify a rule<br>- Security Admin: interest in modify a rule |
| Pre-Conditions | - The website is available<br>- Actor has completed use case MP_FR8 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A specific rule from a probe is modified |
| Trigger | The actor selects the rule he wants to modify |
| Process - Main Success Scenario | - 1: The actor selects the rule he wants to modify<br>- 2: Actor clicks on "Modify Rule"<br>- 3: Actor change all fields that he needs to modify<br>- 4: Actor clicks on "Save" and a confirmation modal is opened<br>- 5: Actor confirms the action by clicking "Yes"<br>- 6: The rule is modified<br>- 7: A success message is displayed by the system |
| Exceptions | - 4:<br>a) Actor does not fill all required fields<br>a1) The system presents a popup message "You must fill all required fields! Please check all required fields!"<br>- 5:<br>a) Actor does not confirm the action by clicking "Yes"<br>a1) The use case ends<br>- 6:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged |

Table 12: Use Case MP_FR12: Modify Rule

| Use Case | MP_FR13: Disable Rule |
|---|---|
| Primary Actor | System Admin |
| Secondary Actor | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in disable a rule for a probe<br>- Security Admin: interest in disable a rule for a probe |
| Pre-Conditions | - The website is available<br>- Actor must have completed the use case MP_FR8 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A rule for a specific probe is disabled |
| Trigger | The actor selects the rule he wants to disable |
| Process - Main Success Scenario | - 1: The actor selects the rule he wants to disable<br>- 2: Actor clicks on "Disable Rule" and a confirmation modal is opened<br>- 3: Actor confirms the action by clicking "Yes"<br>- 4: The rule is disabled<br>- 5: A success message is displayed by the system |
| Exceptions | - 3:<br>a) Actor does not confirm the action by clicking "Yes"<br>a1) The use case ends<br>- 4:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged |

Table 13: Use Case MP_FR13: Disable Rule

| | |
|---|---|
| Use Case | MP_FR14: List Rules |
| Primary Actor | System Admin |
| Secondary Actor | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in view all existent rules<br>- Security Admin: interest in view all existent rules |
| Pre-Conditions | - The website is available<br>- Actor has fulfilled the use case MP_FR8 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | View all existent rules |
| Trigger | Actor clicks on "Rules" |
| Process - Main Success Scenario | - 1: Actor clicks on "Rules"<br>- 2: Actor view all existent rules for that specific probe |
| Exceptions | - 2:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged |

Table 14: Use Case MP_FR14: List Rules

| Use Case | MP_FR15: Remove Rule |
|---|---|
| Primary Actor | System Admin |
| Secondary Actor | Security Admin |
| Scope | Probes (Sub-System) - Black-box |
| Level | User-Goal |
| Stakeholders and Interests | - System Admin: interest in removing a rule<br>- Security Admin: interest in removing a rule |
| Pre-Conditions | - The website is available<br>- Actor must have completed use case MP_FR8 |
| Last Review | 20-12-2017 |
| Minimum Guarantees | An error message is presented |
| Success Guarantees | A rule from a probe is removed |
| Trigger | Actor clicks on "Remove Rule" |
| Process - Main Success Scenario | - 1: The actor selects the rule he wants to disable<br>- 2: Actor clicks on "Disable Rule" and a confirmation modal is opened<br>- 3: Actor confirms the action by clicking "Yes"<br>- 4: The rule is disabled<br>- 5: The rule is removed<br>- 6: A success message is displayed by the system |
| Exceptions | - 3:<br>a) Actor does not confirm the action by clicking "Yes"<br>a1) The use case ends<br>- 4, 5:<br>a) A backend error occurred<br>a1) The system presents a popup message "An error occurred and the previous operation was not executed! Please try again!" and the exception is logged<br>b) The probe is not available<br>b1) The system presents a popup message "Could not connect to probe" and the exception is logged |

Table 15: Use Case MP_FR15: Remove Rule

# Appendix B

# Management Protocol

In this appendix we present the full sized version of each diagram presented in chapter 4

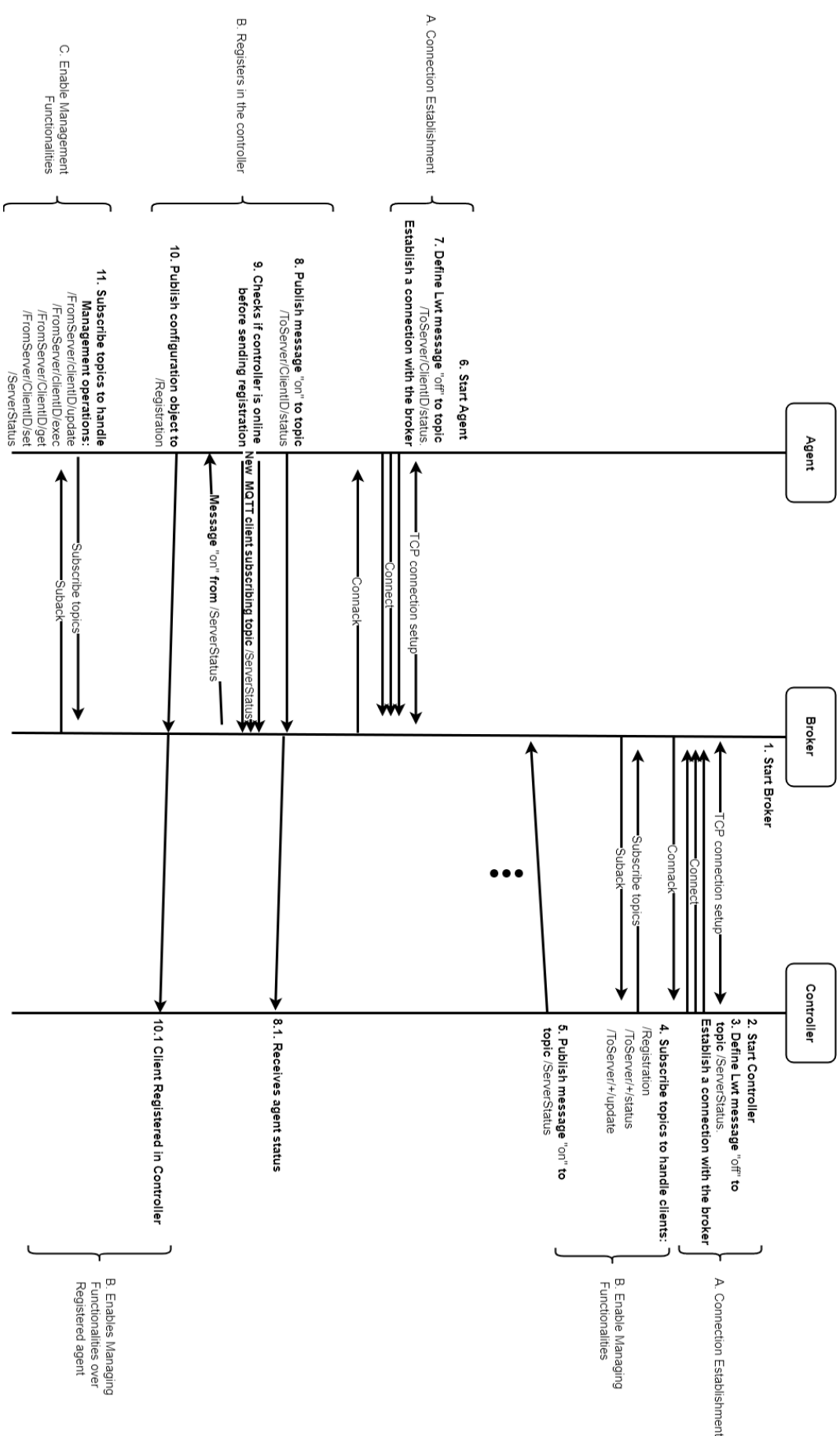


FIGURE B.1: Registration process on MQTT management solution

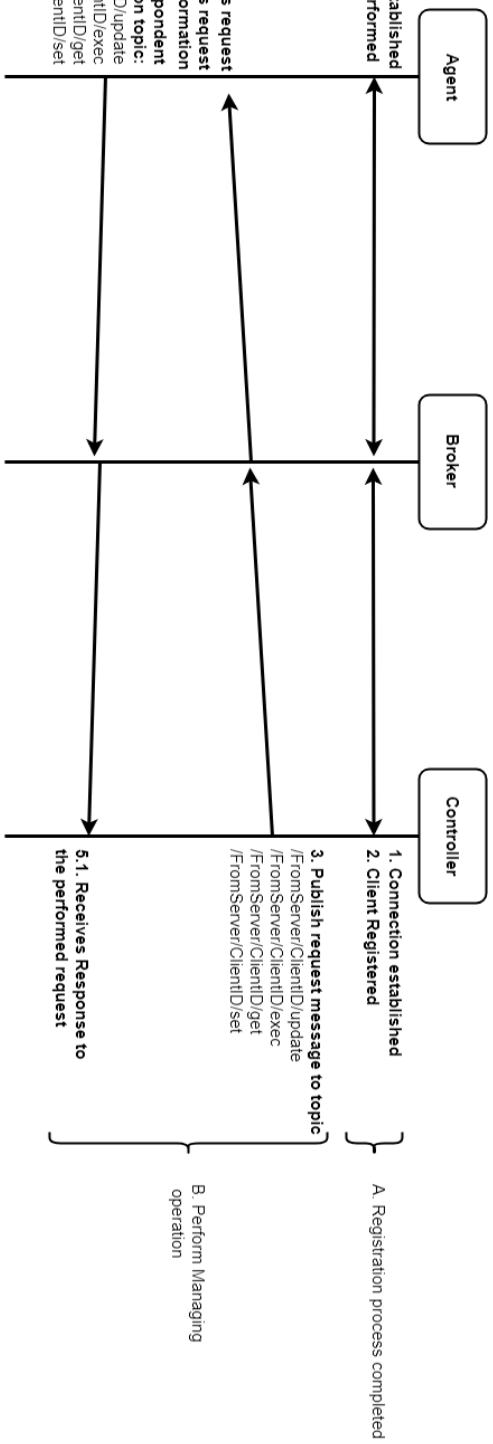


FIGURE B.2: Request process on MQTT management solution

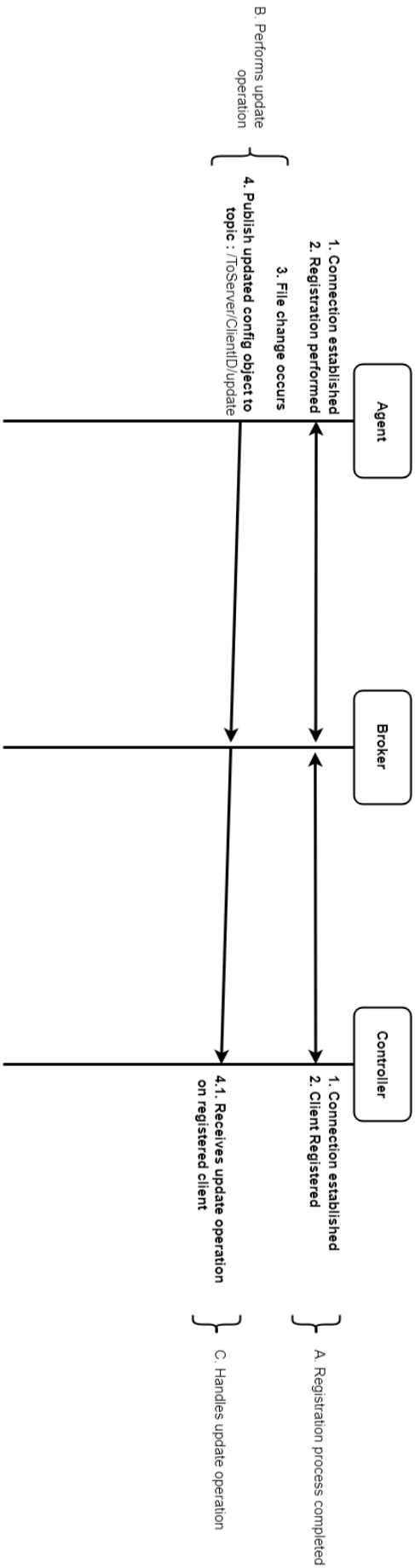


FIGURE B.3: Client file change notification process MQTT on management solution

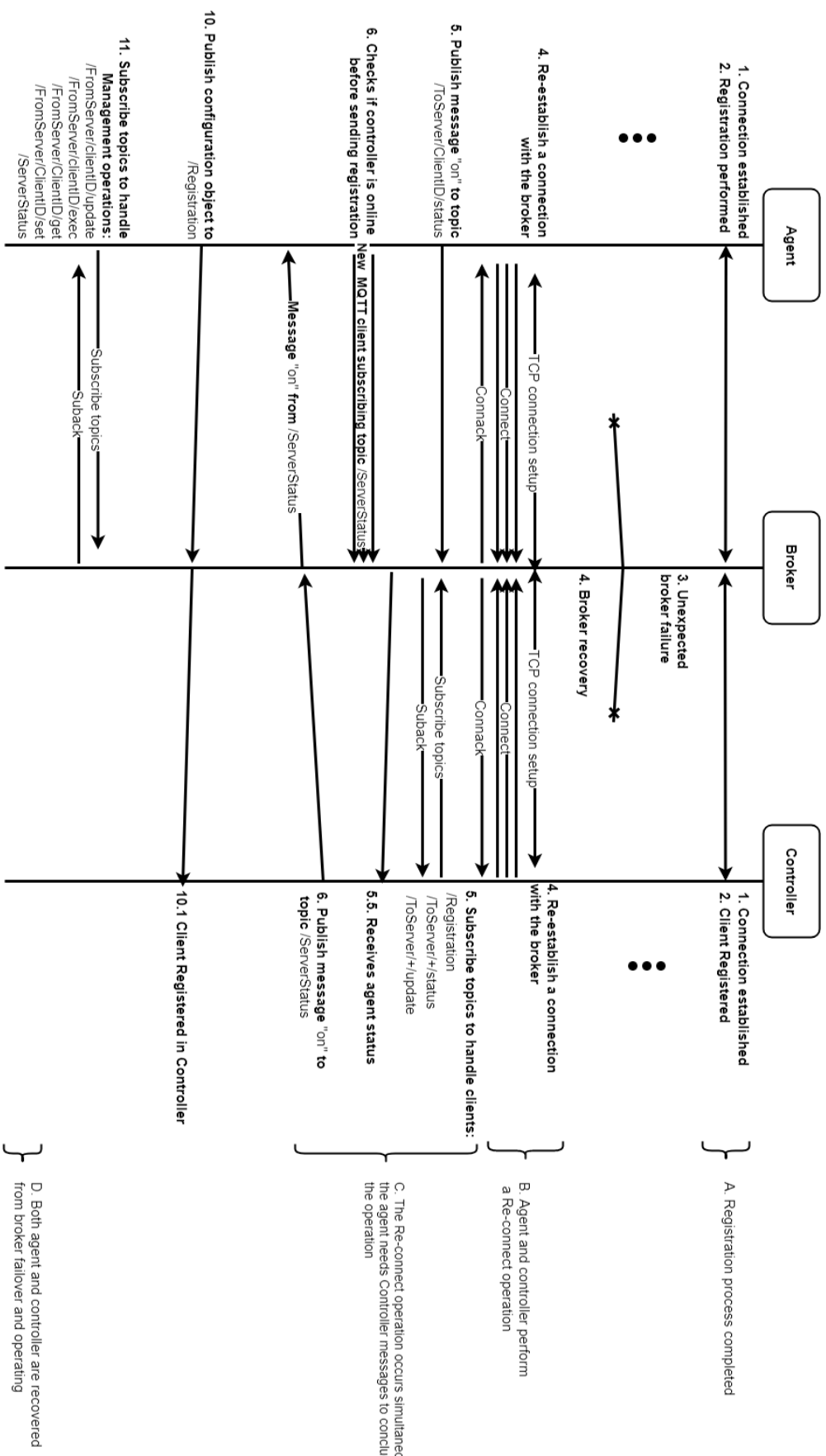


FIGURE B.4: Broker failover on MQTT management solution

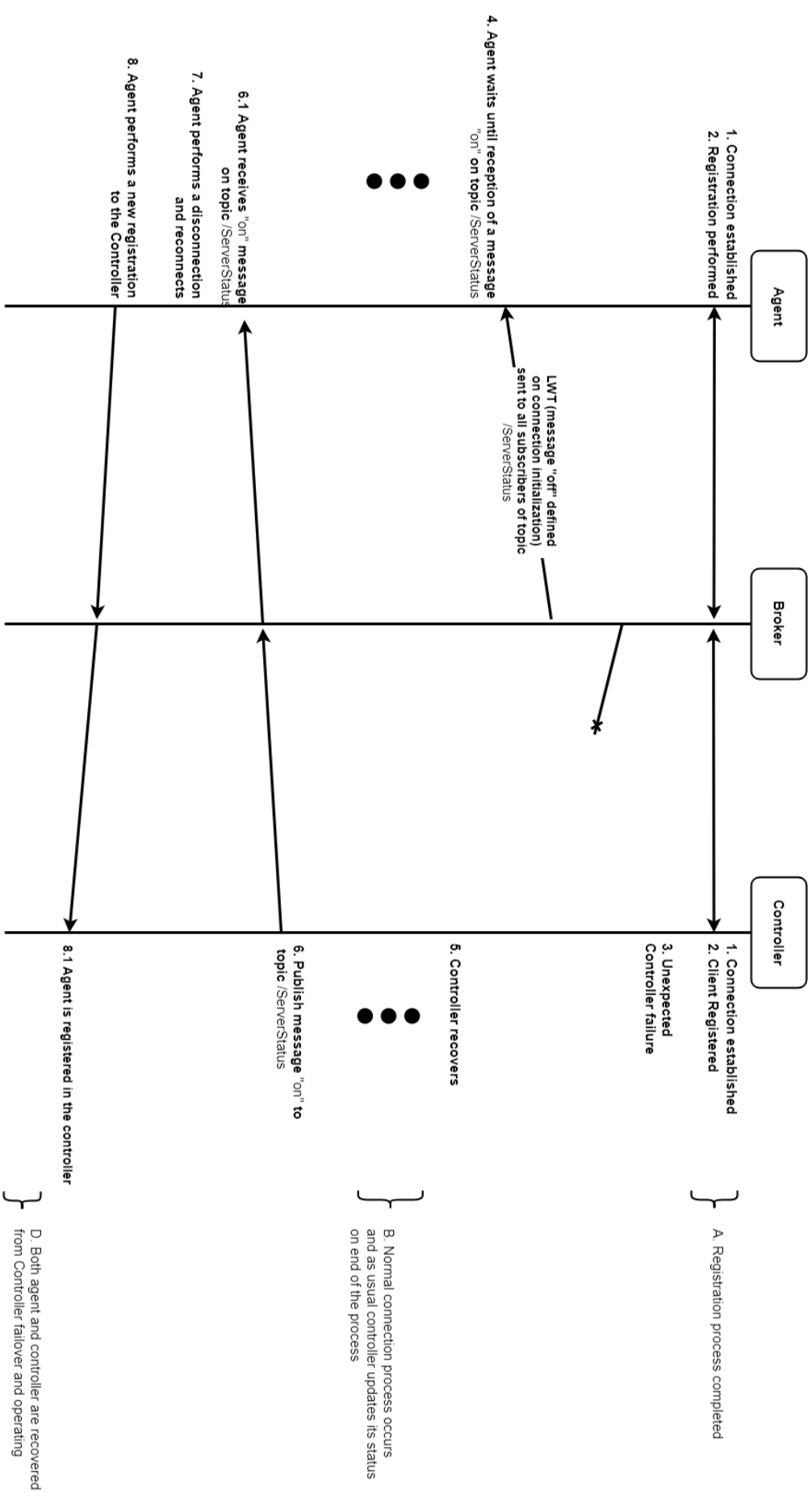


FIGURE B.5: Controller failover on MQTT management solution

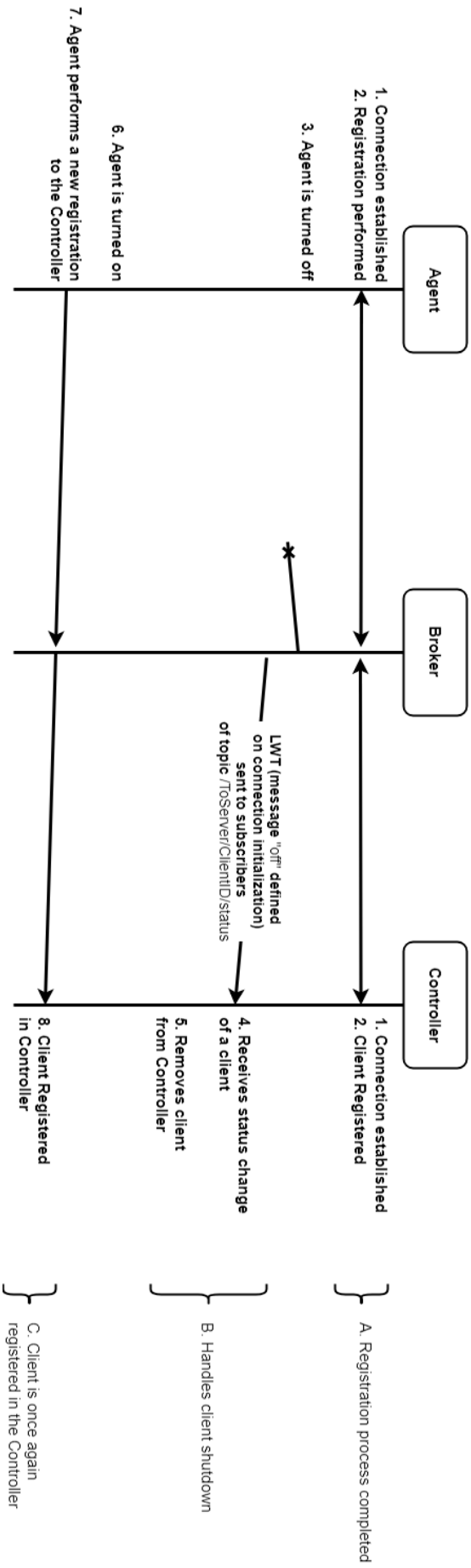


FIGURE B.6: Client disconnection and re-connection on MQTT Management solution

# Appendix C

# Results

In this appendix we present the tables with the results obtained in the tests. All the results are expressed in milliseconds.

| TEST 1 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 Agent** | 2905.689388 | 2905.021007 | 14.906303 | 2912.820201 | 13.820044 | 1750.451389 | 1584.827456 | 13.820044 | 14.906303 | 2905.021007 | 2912.820201 |
| **10 Agents** | 50657.60399 | 112269.6276 | 57181.78715 | 95.861163 | 146214.4338 | 73283.86274 | 56935.19573 | 95.861163 | 50657.60399 | 112269.6276 | 146214.4338 |
| **20 Agents** | 291726.7433 | 302227.3388 | 96.745678 | 91.641677 | 416945.9498 | 202217.6838 | 190938.368 | 91.641677 | 96.745678 | 302227.3388 | 416945.9498 |
| | | | | | | | | | | | |
| **TEST 2** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 Agent** | 37.505964 | 31.116816 | 26.516422 | 24.14535 | 25.341419 | 28.9251942 | 5.475521784 | 24.14535 | 25.341419 | 28.9251942 | 28.9251942 |
| **10 Agents** | 149.626059 | 143.517062 | 157.362393 | 329.37222 | 648.56039 | 285.6876248 | 217.2427049 | 143.517062 | 149.626059 | 329.37222 | 648.56039 |
| **20 Agents** | 281.400306 | 271.854573 | 287.867793 | 310.7768 | 312.186487 | 292.8171918 | 17.97213357 | 271.854573 | 281.400306 | 310.7768 | 312.186487 |
| | | | | | | | | | | | |
| **TEST 3** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 Agent** | 60129.66293 | 60146.87676 | 60141.0064 | 60131.70956 | 60124.99923 | 60134.85098 | 8.893113706 | 60124.99923 | 60129.66293 | 60141.0064 | 60134.85098 |
| **10 Agents** | 94177.08887 | 86573.95763 | 96164.89634 | 94067.72193 | 89661.43855 | 92129.02066 | 3911.173904 | 86573.95763 | 89661.43855 | 94067.72193 | 94067.72193 |
| **20 Agents** | 361370.4268 | 446464.3791 | 384688.4959 | 378088.292 | 356561.8461 | 385434.688 | 36027.34796 | 356561.8461 | 361370.4268 | 385434.688 | 385434.688 |
| | | | | | | | | | | | |
| **TEST 4** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 Agent** | 3.241405 | 3.168557 | 3.135569 | 3.044038 | 3.078362 | 3.1335862 | 0.07733106486 | 3.044038 | 3.078362 | 3.135569 | 3.1335862 |
| **10 Agents** | 3.153419 | 3.773452 | 4.010848 | 3.365259 | 3.408092 | 3.542214 | 0.3440756361 | 3.153419 | 3.365259 | 3.773452 | 3.542214 |
| **20 Agents** | 4.095275 | 4.265704 | 4.26214 | 4.199888 | 4.346046 | 4.2338106 | 0.09321934367 | 4.095275 | 4.199888 | 4.265704 | 4.346046 |
| | | | | | | | | | | | |
| **TEST 5** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 Agent** | 120252.5169 | 120163.9926 | 120168.1113 | 120156.6321 | 120153.7291 | 120178.9964 | 41.49535499 | 120153.7291 | 120156.6321 | 120168.1113 | 120178.9964 |
| **10 Agents** | 120246.5024 | 120272.7827 | 120248.9649 | 120255.7665 | 120269.148 | 120258.6329 | 11.82811882 | 120246.5024 | 120248.9649 | 120269.148 | 120269.148 |
| **20 Agents** | 120228.1918 | 120254.6114 | 120257.5174 | 120251.7943 | 120250.2552 | 120248.474 | 11.67277136 | 120228.1918 | 120250.2552 | 120254.6114 | 120251.7943 |
| | | | | | | | | | | | |
| **TESTE 6** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1 - get** | 2702 | 2685 | 2689 | 2675 | 2670 | 2684.2 | 12.51798706 | 2670 | 2675 | 2685 | 2684.2 |
| **1 - set** | 2681 | 2681 | 2680 | 2680 | 2660 | 2676.4 | 9.181503145 | 2660 | 2680 | 2680 | 2680 |
| **1 - exec** | 3076 | 3054 | 4625 | 4613 | 3070 | 3687.6 | 850.2965953 | 3054 | 3070 | 4613 | 4613 |
| **10 - get** | 2659 | 2666 | 2663 | 2665 | 2663 | 2663.2 | 2.683281573 | 2659 | 2663 | 2665 | 2665 |
| **10 - set** | 2657 | 2677 | 2663 | 2658 | 2875 | 2706 | 94.81033699 | 2657 | 2658 | 2706 | 2875 |
| **10 - exec** | 3071 | 4642 | 3065 | 4632 | 3006 | 3683.2 | 871.0738775 | 3006 | 3065 | 4632 | 4632 |
| **20 - get** | 2683 | 2684 | 2701 | 2682 | 2650 | 2680 | 18.50675552 | 2650 | 2682 | 2684 | 2682 |
| **20 - set** | 2679 | 2676 | 2657 | 2673 | 2682 | 2673.4 | 9.762171889 | 2657 | 2673 | 2676 | 2682 |
| **20 - exec** | 3079 | 3073 | 3104 | 3074 | 3084 | 3082.8 | 12.63724654 | 3073 | 3074 | 3084 | 3084 |
| | | | | | | | | | | | |
| **TESTE 7** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 - get** | 2658 | 2679 | 2666 | 2654 | 2669 | 2665.2 | 9.782637681 | 2654 | 2658 | 2669 | 2669 |
| **1 - set** | 2678 | 2673 | 2682 | 2648 | 2681 | 2672.4 | 14.08190328 | 2648 | 2673 | 2681 | 2681 |
| **1 - exec** | 2667 | 2678 | 2656 | 2666 | 2665 | 2666.4 | 7.829431652 | 2656 | 2665 | 2666.4 | 2666.4 |
| **10 - get** | 2635 | 2642 | 2659 | 2644 | 2635 | 2643 | 9.823441352 | 2635 | 2635 | 2644 | 2644 |
| **10 - set** | 2658 | 2645 | 2646 | 2629 | 2642 | 2644 | 10.36822068 | 2629 | 2642 | 2645 | 2644 |
| **10 - exec** | 2649 | 2663 | 2679 | 2658 | 2857 | 2701.2 | 87.77357233 | 2649 | 2658 | 2701.2 | 2857 |
| **20 - get** | 2679 | 2664 | 2650 | 2668 | 2639 | 2660 | 15.66843962 | 2639 | 2650 | 2664 | 2668 |
| **20 - set** | 2675 | 2670 | 2660 | 2659 | 2664 | 2665.6 | 6.804410334 | 2659 | 2660 | 2665.6 | 2665.6 |
| **20 - exec** | 2673 | 2670 | 2657 | 2658 | 2674 | 2666.4 | 8.264381405 | 2657 | 2658 | 2670 | 2674 |
| | | | | | | | | | | | |
| **TESTE 8** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 - get** | 121 | 93 | 102 | 91 | 116 | 104.6 | 13.46476884 | 91 | 93 | 104.6 | 116 |
| **1 - set** | 97 | 97 | 101 | 114 | 88 | 99.4 | 9.449867724 | 88 | 97 | 101 | 114 |
| **1 - exec** | 117 | 119 | 112 | 86 | 108 | 108.4 | 13.24009063 | 86 | 108 | 112 | 108.4 |
| **1 - update** | 123 | 103 | 122 | 104 | 113 | 113 | 9.513148795 | 103 | 104 | 113 | 113 |
| **10 - get** | 85 | 92 | 89 | 113 | 105 | 96.8 | 11.75584961 | 85 | 89 | 105 | 113 |
| **10 - set** | 91 | 94 | 98 | 84 | 103 | 94 | 7.176350047 | 84 | 91 | 98 | 103 |
| **10 - exec** | 82 | 91 | 97 | 84 | 103 | 91.4 | 8.792041856 | 82 | 84 | 97 | 103 |
| **10 - update** | 107 | 99 | 98 | 85 | 187 | 115.2 | 40.90476745 | 85 | 98 | 115.2 | 187 |
| **20 - get** | 91 | 111 | 108 | 100 | 117 | 105.4 | 10.11434625 | 91 | 100 | 111 | 117 |
| **20 - set** | 91 | 92 | 131 | 96 | 95 | 101 | 16.89674525 | 91 | 92 | 101 | 101 |
| **20 - exec** | 119 | 123 | 101 | 97 | 105 | 109 | 11.40175425 | 97 | 101 | 109 | 109 |
| **20 - update** | 213 | 102 | 94 | 93 | 114 | 123.2 | 50.89891944 | 93 | 94 | 114 | 123.2 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **TESTE 9** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 - get** | 2672 | 2658 | 2652 | 2665 | 2648 | 2659 | 9.695359715 | 2648 | 2652 | 2659 | 2665 |
| **1 - set** | 2661.5 | 2667.5 | 2676 | 2667.5 | 2668.5 | 2668.2 | 5.167204273 | 2661.5 | 2667.5 | 2668.5 | 2668.5 |
| **1 - exec** | 2675 | 2675 | 2669 | 2658 | 2672.5 | 2669.9 | 7.092249291 | 2658 | 2669 | 2672.5 | 2672.5 |
| **10 - get** | 2638 | 2652 | 2653 | 2650 | 2684 | 2655.4 | 17.08215443 | 2638 | 2650 | 2655.4 | 2684 |
| **10 - set** | 2655 | 2654.5 | 2650.5 | 2644.5 | 2729.5 | 2666.8 | 35.30155804 | 2644.5 | 2650.5 | 2666.8 | 2729.5 |
| **10 - exec** | 2636 | 2637.5 | 2638 | 2688.5 | 2638.5 | 2647.7 | 22.82706727 | 2636 | 2637.5 | 2647.7 | 2688.5 |
| **20 - get** | 2672 | 2653 | 2673 | 2675 | 2658 | 2666.2 | 9.984988733 | 2653 | 2658 | 2673 | 2675 |
| **20 - set** | 2658.5 | 2667.5 | 2674.5 | 2657 | 2660 | 2663.5 | 7.356969485 | 2657 | 2658.5 | 2667.5 | 2663.5 |
| **20 - exec** | 2663 | 2654.5 | 2669 | 2660 | 2666 | 2662.5 | 5.590169944 | 2654.5 | 2660 | 2666 | 2666 |
| | | | | | | | | | | | |
| **TEST 10** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 - get** | 2653 | 2657 | 2707 | 2665 | 2678 | 2672 | 21.77154106 | 2653 | 2657 | 2678 | 2678 |
| **1 - set** | 4613 | 3066 | 4624 | 4625 | 4626 | 4310.8 | 695.8841139 | 3066 | 4613 | 4625 | 4626 |
| **1 - exec** | 2675 | 2661 | 2661 | 2661 | 2671 | 2665.8 | 6.723094526 | 2661 | 2661 | 2665.8 | 2671 |
| **10 - get** | 2642 | 2653 | 2636 | 2640 | 2632 | 2640.6 | 7.924645102 | 2632 | 2636 | 2640.6 | 2640.6 |
| **10 - set** | 3061 | 3071 | 4645 | 3066 | 3049 | 3378.4 | 708.0980158 | 3049 | 3061 | 3378.4 | 3378.4 |
| **10 - exec** | 2668 | 2682 | 2725 | 2645 | 2644 | 2672.8 | 33.29714702 | 2644 | 2645 | 2682 | 2672.8 |
| **20 - get** | 2649 | 2648 | 2670 | 2661 | 2669 | 2659.4 | 10.54988152 | 2648 | 2649 | 2669 | 2669 |
| **20 - set** | 3326 | 3075 | 3076 | 3074 | 3069 | 3124 | 112.9535303 | 3069 | 3074 | 3076 | 3124 |
| **20 - exec** | 2658 | 2660 | 2657 | 2668 | 2678 | 2664.2 | 8.843076388 | 2657 | 2658 | 2668 | 2678 |
| | | | | | | | | | | | |
| **TESTE 11** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 - get** | 14115.33333 | 14136.66667 | 14152.33333 | 14123.33333 | 14139.66667 | 14133.46667 | 14.45798511 | 14115.33333 | 14123.33333 | 14139.66667 | 14139.66667 |
| **1 - set** | 7390 | 7122 | 7127 | 7150.333333 | 7107.666667 | 7179.4 | 118.7259683 | 7107.666667 | 7122 | 7150.333333 | 7179.4 |
| **1 - exec** | 7111.666667 | 7114.666667 | 7115.333333 | 7115.333333 | 7127 | 7116.8 | 5.900094161 | 7111.666667 | 7114.666667 | 7116.8 | 7127 |
| **10 - get** | 14104 | 14108 | 14109.66667 | 14105 | 14115.33333 | 14108.4 | 4.493204746 | 14104 | 14105 | 14109.66667 | 14115.33333 |
| **10 - set** | 7130.333333 | 7118.666667 | 7112.666667 | 7115 | 7106 | 7116.533333 | 8.989500048 | 7106 | 7112.666667 | 7116.533333 | 7116.533333 |
| **10 - exec** | 7099 | 7100.666667 | 7096.666667 | 7099.333333 | 7211.333333 | 7121.4 | 50.29490807 | 7096.666667 | 7099 | 7121.4 | 7211.333333 |
| **20 - get** | 14121 | 14134.33333 | 14126 | 14125 | 14119.66667 | 14125.2 | 5.752294228 | 14119.66667 | 14121 | 14126 | 14125.2 |
| **20 - set** | 7108 | 7120.333333 | 7126.333333 | 7105.666667 | 7111.666667 | 7114.4 | 8.693548055 | 7105.666667 | 7108 | 7120.333333 | 7114.4 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **20 - exec** | 7125.666667 | 7128 | 7114.666667 | 7116.666667 | 7123 | 7121.6 | 5.741660619 | 7114.666667 | 7116.666667 | 7123 | 7123 |
| | | | | | | | | | | | |
| **TESTE 12** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 - get** | 117 | 107 | 120 | 95 | 113 | 110.4 | 9.889388252 | 95 | 107 | 113 | 113 |
| **1 - set** | 119 | 132 | 104 | 125 | 97 | 115.4 | 14.57051818 | 97 | 104 | 125 | 125 |
| **1 - exec** | 108 | 134 | 104 | 99 | 123 | 113.6 | 14.50172404 | 99 | 104 | 123 | 123 |
| **10 - get** | 97 | 83 | 92 | 84 | 89 | 89 | 5.787918451 | 83 | 84 | 89 | 89 |
| **10 - set** | 108 | 103 | 85 | 85 | 91 | 94.4 | 10.57355191 | 85 | 85 | 94.4 | 94.4 |
| **10 - exec** | 90 | 115 | 95 | 85 | 86 | 94.2 | 12.27599283 | 85 | 86 | 95 | 94.2 |
| **20 - get** | 158 | 114 | 99 | 116 | 104 | 118.2 | 23.32809465 | 99 | 104 | 116 | 118.2 |
| **20 - set** | 112 | 85 | 100 | 110 | 94 | 100.2 | 11.23387734 | 85 | 94 | 100.2 | 110 |
| **20 - exec** | 127 | 79 | 99 | 122 | 103 | 106 | 19.26136028 | 79 | 99 | 106 | 122 |
| | | | | | | | | | | | |
| **TESTE 13** | | | | | | | | | | | |
| | | | | | | | | | | | |
| **test/run** | **1** | **2** | **3** | **4** | **5** | **average runs (MS)** | **STD runs (MS)** | **minimum** | **lower quartile** | **upper quartile** | **maximum** |
| **1 Agent** | 116 | 119 | 104 | 118 | 117 | 114.8 | 6.140032573 | 104 | 116 | 118 | 118 |
| **10 Agent** | 298 | 363 | 247 | 234 | 238 | 276 | 55.00454527 | 234 | 238 | 276 | 276 |
| **20 Agent** | 318 | 320 | 257 | 272 | 348 | 303 | 37.46998799 | 257 | 272 | 320 | 348 |

# Bibliography


[1] ATENA Consortium. "Advanced Tools to assEss and mitigate the criticality of ICT compoNents and their dependencies over Critical InfrAstructures (ATENA) - Proposal ID 700581". In: *Horizon 2020 Call : H2020-DS-2015-1 Topic : DS-03-2015 Type of action : IA* 700581 (2015), pp. 1–152.

[2] Apache. *Apache Spark™ - Unified Analytics Engine for Big Data*. URL: https://spark.apache.org/ (visited on 06/10/2018).

[3] Richard H. Caro. *Deterministic Modbus*. URL: https://control.com/thread/1026206682{\#}1026206682 (visited on 05/08/2018).

[4] By Vishal Prakash and Robert Casey. "SNMP – a new paradigm for SCADA By Vishal Prakash and Robert Casey". In: ().

[5] J. JD Case et al. "A simple network management protocol (SNMP)". In: *Internet request for comments* 1157 (1990), pp. 1–36. DOI: 10.17487/rfc1157. URL: http://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:A+Simple+Network+Management+Protocol+(SNMP){\#}0{\%}5Cnhttp://scholar.google.com/scholar?hl=en{\&}btnG=Search{\&}q=intitle:Simple+network+management+protocol+(SNMP){\#}0.

[6] William. Stallings and William. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley, 1999, p. 619. ISBN: 0201485346.

[7] J Case and M Rose. *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*. 1996, pp. 1–40. ISBN: 2147483648.

[8] B. Wijnen, D. Harrington, and R. Presuhn. "An Architecture for Describing SNMP Management Frameworks". In: (). URL: https://tools.ietf.org/html/rfc2271.

[9] R Frye et al. "RFC 3584 Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework". In: (2003), pp. 1–51. ISSN: 2070-1721. DOI: 10.17487/RFC3584. URL: https://tools.ietf.org/pdf/rfc3584.pdf.

[10] Ubizen AETHIS. "Security in SNMPv3 versus SNMPv1 or v2c". In: (2002), pp. 1–6.

[11] C. Bormann, C. Bormann, and A. Keranen. "RFC 7228 - Terminology for Constrained-Node Networks". In: (2014). URL: https://tools.ietf.org/pdf/rfc7228.pdf.

[12] Shahid Raza et al. "Lithe: Lightweight secure CoAP for the internet of things". In: *IEEE Sensors Journal* 13.10 (2013), pp. 3711–3720. ISSN: 1530437X. DOI: 10.1109/JSEN.2013.2277656.

[13] Ala Al-Fuqaha et al. "Toward better horizontal integration among IoT services". In: *IEEE Communications Magazine* 53.9 (2015), pp. 72–79. ISSN: 01636804. DOI: 10.1109/MCOM.2015.7263375. arXiv: arXiv:1011.1669v3.

[14] Dinesh Thangavel et al. "Performance evaluation of MQTT and CoAP via a common middleware". In: *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings* April (2014), pp. 21–24. ISSN: 978-1-4799-2843-9. DOI: 10.1109/ISSNIP.2014.6827678.

[15] Suhas Rao et al. "Implementing LWM2M in constrained IoT devices". In: *2015 IEEE Conference on Wireless Sensors, ICWiSE 2015* (2016), pp. 52–57. DOI: 10.1109/ICWISE.2015.7380353.

[16] Shashank Agrawal. "A survey on Internet of Things". In: *Abakos* 1.2 (2013), pp. 78–95. ISSN: 2316-9451. DOI: 10.5752/P.2316-9451.2013v1n2p78.

[17] Varun Wadhwa. *How to Implement Moscapsule MQTT client in ios Swift*. URL: http://www.oodlestechnologies.com/blogs/How-to-Implement-Moscapsule-MQTT-client-in-ios-Swift (visited on 01/10/2018).

[18] James Stansberry. *MQTT and CoAP: Underlying Protocols for the IoT*. 2015. URL: http://www.electronicdesign.com/iot/mqtt-and-coap-underlying-protocols-iot (visited on 12/27/2017).

[19] Justin Cooper. *Overview | Adafruit IO | Adafruit Learning System*. URL: https://learn.adafruit.com/adafruit-io/overview (visited on 04/05/2018).

[20] Paolo Patierno. *Who | M2Mqtt & GnatMQ*. URL: https://m2mqtt.wordpress.com/ (visited on 04/05/2018).

[21] Michael S. Klishin and Alex Petrov. *Machine Head, Clojure MQTT Client*. URL: http://clojuremqtt.info/ (visited on 04/05/2018).

[22] Eclipse Foundation. *Eclipse Paho - MQTT and MQTT-SN software*. URL: https://www.eclipse.org/paho/ (visited on 04/05/2018).

[23] wolfSSL Inc. *wolfSSL Embedded SSL/TLS Library | Now Supporting TLS 1.3*. URL: https://www.wolfssl.com/ (visited on 04/05/2018).

[24] HiveMQ. *HiveMQ 3.3 - User Guide*. URL: https://www.hivemq.com/docs/hivemq/latest/ (visited on 04/05/2018).

[25] HiveMQ. *MQTT Security Fundamentals | HiveMQ*. URL: https://www.hivemq.com/mqtt-security-fundamentals/ (visited on 04/05/2018).

[26] Eclipse Foundation. *Eclipse Mosquitto*. URL: https://mosquitto.org/ (visited on 04/05/2018).

[27] Pivotal. *RabbitMQ - RabbitMQ MQTT Adapter*. URL: https://www.rabbitmq.com/mqtt.html (visited on 04/05/2018).

[28] Andrea Selva. *What is Moquette?* URL: https://github.com/andsel/moquette (visited on 04/05/2018).

[29] Feng Lee. *EMQ - The Massively Scalable Open Source MQTT Broker*. URL: http://emqtt.io/ (visited on 06/07/2018).

[30] Andy Stanford-Clark and Hong Linh Truong. "MQTT For Sensor Networks (MQTT-SN) Protocol Specification". In: (2013). URL: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN{_}spec{_}v1.2.pdf.

[31] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. "MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks". In: *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)* (2008), pp. 791–798. DOI: 10.1109/COMSWA.2008.4554519. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4554519.

[32] Micro Widget Toolkit and Profile Specification. "ESR Consortium". In: ().

[33] Sathiskumar Palaniappan. *LWM2M over MQTT*. URL: https://github.com/sathipal/lwm2m{_}over{_}mqtt (visited on 05/04/2018).

[34] Seong Min Kim, Hoan Suk Choi, and Woo Seop Rhee. "IoT home gateway for auto-configuration and management of MQTT devices". In: *2015 IEEE Conference on Wireless Sensors, ICWiSE 2015* (2016), pp. 12–17. DOI: 10.1109/ICWISE.2015.7380346.

[35] Tiago Cruz. "A Management Framework for Residential Broadband Environments". PhD thesis. University of Coimbra, 2012.

[36] Sébastien Bolles and Olivier Beyler. *Modus-TR-069 download | SourceForge.net*. URL: https://sourceforge.net/projects/modus-tr-069/ (visited on 06/08/2018).

[37] Peter Lambrechtsen. *LibreACS download | SourceForge.net*. URL: https://sourceforge.net/projects/libreacs/ (visited on 06/08/2018).

[38] Dobrica Pavlinušić. *Perl CWMP*. URL: https://github.com/dpavlin/perl-cwmp (visited on 06/08/2018).

[39] Anders Norgaard. *Grail ACS*. URL: https://github.com/andersnorgaard/grailsacs.

[40] Axicom. *Google Code Archive - Long-term storage for Google Code Project Hosting.* URL: https://code.google.com/archive/p/open-tr069/ (visited on 06/08/2018).

[41] Serafeim Papastefanos. *spapas/CWMPClient — Bitbucket*. URL: https://bitbucket.org/spapas/cwmpclient (visited on 06/08/2018).

[42] Andranik Ghukasyan. *TR-069 D-Link download | SourceForge.net*. URL: https://sourceforge.net/projects/tr069dlink/ (visited on 06/08/2018).

[43] GenieACS Inc. *GenieACS*. URL: https://genieacs.com/ (visited on 06/08/2018).

[44] EasyCwmp - TR069 cwmp. *EasyCwmp - TR069 cwmp client open source - About*. URL: http://www.easycwmp.org/ (visited on 06/08/2018).

[45] Jarl André Hübenthal. *Freeacs*. URL: https://github.com/freeacs (visited on 06/08/2018).

[46] J Schoenwaelder. "RFC 6241 - Network Configuration Protocol ... NETCONF—". In: (2011). ISSN: 2070-1721. URL: http://www.rfc-editor.org/info/rfc6241..

[47] Juniper Networks. *Go-netconf*. URL: https://github.com/Juniper/go-netconf (visited on 06/08/2018).

[48] Giuseppe Palmeri. *Google Code Archive - Long-term storage for Google Code Project Hosting*. URL: https://code.google.com/archive/p/netconf4android/ (visited on 06/09/2018).

[49] CESNET. *CESNET | CESNET, zájmové sdružení právnických osob*. URL: https://www.cesnet.cz/?lang=en (visited on 06/09/2018).

[50] Cisco company. *ConfD Basic | Tail-f Systems*. URL: http://www.tail-f.com/confd-basic/ (visited on 06/09/2018).

[51] Andrew Foster. "Messaging Technologies for the Industrial Internet and the Internet of Things A Comparison Between DDS, AMQP, MQTT, JMS, REST and CoAP Version 1.2, November 2013". In: (). URL: http://www.prismtech.com/sites/default/files/documents/MessagingComparsionNov2013USROW{_}vfinal.pdf.

[52] NATIONAL COMMUNICATIONS SYSTEM. "Supervisory Control and Data Acquisition (SCADA) Systems". In: (2004). URL: https://scadahacker.com/library/Documents/ICS{_}Basics/SCADABasics-NCSTIB04-1.pdf.

[53] Michał Choraś et al. "Decision Aid Tool and Ontology-Based Reasoning for Critical Infrastructure Vulnerabilities and Threats Analysis". In: Springer, Berlin, Heidelberg, 2010, pp. 98–110. DOI: 10.1007/978-3-642-14379-3_9. URL: http://link.springer.com/10.1007/978-3-642-14379-3{_}9.

[54] Edvard. *3 generations of SCADA system architectures you should know about | EEP*. 2013. URL: http://electrical-engineering-portal.com/three-generations-of-scada-system-architectures (visited on 12/25/2017).

[55] Anam Sajid, Haider Abbas, and Kashif Saleem. "SPECIAL SECTION ON THE PLETHORA OF RESEARCH IN INTERNET OF THINGS (IoT) Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges". In: 4 (2016). DOI: 10.1109/ACCESS.2016.2549047. URL: file:///C:/Users/bas/AppData/Local/Temp/07445139.pdf.

[56] Occupytheweb. *What the Heck Was Stuxnet!? « Null Byte :: WonderHowTo*. URL: https://null-byte.wonderhowto.com/news/what-heck-was-stuxnet-0160816/ (visited on 12/25/2017).

[57] Positive Technologies. "SCADA Safety in Numbers". In: (2012), pp. 1–29. URL: https://www.ptsecurity.com/upload/corporate/ww-en/analytics/ICS-SCADA-Safety-Numbers-2012.pdf.

[58] Boldizsár Bencsáth et al. "The Cousins of Stuxnet: Duqu, Flame, and Gauss". In: *Future Internet* 4.4 (2012), pp. 971–1003. ISSN: 1999-5903. DOI: 10.3390/fi4040971. URL: www.mdpi.com/journal/futureinternet.

[59] Boldizsár Bencsáth. "Duqu, Flame, Gauss: Followers of Stuxnet". In: (). URL: http://www.crysys.hu/.

[60] FireEye. *Attackers Deploy New ICS Attack Framework "TRITON" and Cause Operational Disruption to Critical Infrastructure « Attackers Deploy New ICS Attack Framework "TRITON" and Cause Operational Disruption to Critical Infrastructure | FireEye Inc*. URL: https://www.fireeye.com/blog/threat-research/2017/12/attackers-deploy-new-ics-attack-framework-triton.html (visited on 12/25/2017).

[61] Jackson Kelly. *SCADA Security In A Post-Stuxnet World*. URL: https://www.darkreading.com/vulnerabilities---threats/scada-security-in-a-post-stuxnet-world/d/d-id/1138634 (visited on 12/25/2017).

[62] Siddharth Sridhar and G Manimaran. "Data Integrity Attacks and their Impacts on SCADA Control System". In: (). URL: https://s3.amazonaws.com/academia.edu.documents/38387560/shiv{_}6.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A{\&}Expires=1514229410{\&}Signature=IaYzuDV4jw_Y2fnEIcjj4SYktqn8{\%}3D{\&}response-content-disposition=inline{\%}3Bfilename{\%}3DData{_}Integrity{_}Attacks{_}and{_}their{_}Impacts..

[63] Bonnie Zhu, Anthony Joseph, and Shankar Sastry. "A Taxonomy of Cyber Attacks on SCADA Systems". In: (). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.224.8430{\&}rep=rep1{\&}type=pdf.

[64] Mercy Bere. "Initial Investigation of Industrial Control System ( ICS ) Security Using Artificial Immune System ( AIS )". In: (2015).

[65] Yongge Wang and Bei-Tseng Chu. "sSCADA: Securing SCADA Infrastructure Communications". In: (2004). URL: https://eprint.iacr.org/2004/265.pdf.

[66] C. M. Davis et al. "SCADA cyber security testbed development". In: *2006 38th Annual North American Power Symposium, NAPS-2006 Proceedings* (2006), pp. 483–488. DOI: 10.1109/NAPS.2006.359615. arXiv: 161.

[67] Alvaro A Cárdenas et al. "Attacks Against Process Control Systems: Risk Assessment, Detection, and Response". In: (). URL: https://pdfs.semanticscholar.org/ee2e/e3dca15c4836b07c7a0e2c265329a9298901.pdf.

[68] Teo Lawrence. *Network Probes Explained: Understanding Port Scans and Ping Sweeps | Linux Journal*. URL: http://www.linuxjournal.com/article/4234 (visited on 01/08/2018).

[69] Thiago Milani. *Honeypot para detecção e contenção de ataques e botnets*. URL: https://www.profissionaisti.com.br/2014/08/honeypot-para-deteccao-e-contencao-de-ataques-e-botnets/ (visited on 01/18/2018).

[70] Marcin Nawrocki et al. "A Survey on Honeypot Software and Data Analysis". In: (). URL: https://arxiv.org/pdf/1608.06249.pdf.

[71] networks Paloalto. *What is an Intrusion Detection System? - Palo Alto Networks*. URL: https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids (visited on 01/11/2018).

[72] Alan Henry. *The Difference Between Antivirus and Anti-Malware (and Which to Use)*. URL: https://lifehacker.com/the-difference-between-antivirus-and-anti-malware-and-1176942277 (visited on 01/11/2018).

[73] Ssj Jay. *Basics Of Intrusion Detection Systems*. URL: https://www.hackthis.co.uk/articles/basics-of-intrusion-detection-systems (visited on 01/18/2018).

[74] networks Jupiter. *What is IDS and IPS? | Juniper Networks*. URL: https://www.juniper.net/us/en/products-services/what-is/ids-ips/ (visited on 01/22/2018).

[75] Tiago Cruz et al. "Improving network security monitoring for industrial control systems". In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (2015), pp. 878–881. DOI: 10.1109/INM.2015.7140399. URL: http://ieeexplore.ieee.org/document/7140399/{\%}5Cnhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7140399.

[76] Jean-Loup Richet. *Cybersecurity Policies and Strategies for Cyberwarfare Prevention - Google Livros*. URL: https://books.google.pt/books?id=p7EyCgAAQBAJ{\&}pg=PA153{\&}lpg=PA153{\&}dq=shadow+rtu{\&}source=bl{\&}ots=jp4mkRDKY7{\&}sig=V6rR{_}wkBbWRKruZ6dRq{_}vOTJq28{\&}hl=pt-PT{\&}sa=X{\&}ved=0ahUKEwir66e6hNHYAhVMshQKHdmCCZEQ6AEIRDAE{\#}v=onepage{\&}q=shadowrtu{\&}f=false (visited on 01/12/2018).

[77] Rainer Gerhards. "rsyslog: going up from 40K messages per second to 250K". In: (). URL: http://www.gerhards.net/download/LinuxKongress2010rsyslog.pdf.

[78] Apache. *Apache Kafka*. URL: http://kafka.apache.org/documentation.html (visited on 11/03/2017).

[79] Apache. *Apache Avro™ 1.8.2 Documentation*. URL: https://avro.apache.org/docs/current/ (visited on 06/10/2018).

[80] Yesudeep Mangalapilly. *Watchdog — watchdog 0.8.2 documentation*. URL: https://pythonhosted.org/watchdog/ (visited on 06/10/2018).

[81] Michael Bayer. *SQLAlchemy - The Database Toolkit for Python*. URL: https://www.sqlalchemy.org/ (visited on 06/10/2018).

[82] SQLite Consortium. *SQLite Home Page*. URL: https://www.sqlite.org/index.html (visited on 06/10/2018).

[83] Armin Ronacher. *Welcome | Flask (A Python Microframework)*. URL: http://flask.pocoo.org/ (visited on 06/10/2018).

[84] Armin Ronacher. *Welcome | Werkzeug (The Python WSGI Utility Library)*. URL: http://werkzeug.pocoo.org/ (visited on 06/10/2018).

[85] Armin Ronacher. *Welcome to Jinja2 — Jinja2 Documentation (2.10)*. URL: http://jinja.pocoo.org/docs/2.10/ (visited on 06/10/2018).

[86] Inc. Postdot Technologies. *Postman | API Development Environment*. URL: https://www.getpostman.com/ (visited on 05/28/2018).