

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**

# **A Tool for Large-Scale Workflow Control in Edge-based Industry 4.0 Applications**

**Rui Pedro de Oliveira Reis**



**FEUP** FACULDADE DE ENGENHARIA  
UNIVERSIDADE DO PORTO

Master in Electrical and Computer Engineering

Supervisor: Pedro Miguel Salgueiro Santos

Co-Supervisor: Luís Miguel Pinho de Almeida

February 28, 2023



# **A Tool for Large-Scale Workflow Control in Edge-based Industry 4.0 Applications**

**Rui Pedro de Oliveira Reis**

Master in Electrical and Computer Engineering

Approved in oral examination by the committee:

Chair: Paulo José Lopes Machado Portugal

External Examiner: Sérgio Ivan Lopes

Supervisor: Pedro Miguel Salgueiro dos Santos

Co-Supervisor: Luís Miguel Pinho de Almeida

February 28, 2023



# Abstract

The Industry 4.0 is a set of fundamental changes of the industrial and logistical processes enabled by data computing, e.g., 5G, IoT, Machine Learning, among others. In order to intertwine the devices managed in the workflow, a tool for large-scale workflow control is necessary. This shall control device connection, monitoring and management of the services used in workflows. The main focus is to create a tool to enable the large scale interaction between devices while having a GUI (Graphical User Interface) to change the parameters of the deployed workflows.

To be able to implement such workflow, this thesis makes a theoretical background study to know where similar types of frameworks have been applied, in order to create a scalable tool. After knowing how the studied IoT frameworks perform, a comparison is made to decide which one better suits the matter in hands.

Afterwards a description of the system takes place. In order to understand how the tool runs an explanation of how developed tool is done. It is necessary to understand how it communicates with the chosen framework, how to navigate in its interface and what is the functionalities that it provides.

Then the tool is validated and tested. The tool is validated once all the listed functionalities are tested and working. The tests that were done were about the time responses of the tool. How long it took to update all the information related to each device that is linked with the tool.

In conclusion it is presented some possible future works to improve the robustness of the developed large scale control tool.

**Keywords:** Edge, Industry 4.0, IoT, large-scale, workflow.



# Resumo

A Indústria 4.0 é um conjunto de mudanças fundamentais permitidas pela computação de dados dos processos logísticos e industriais, e.g., 5G, IoT, *Machine Learning*, entre outros. De forma a ligar todos os dispositivos geridos na *workflow*, é necessária uma ferramenta de controlo de *workflows* em larga escala. Isto deverá controlar as conexões dos dispositivos e monitorizar e gerir os serviços nas *workflows*. O foco principal é criar uma ferramenta que possibilite a interação em larga escala entre dispositivos com a disposição de uma interface gráfica para mudar os parâmetros das *workflows* como se desejar.

Para a implementar tal *workflow*, esta tese faz um estudo teórico para saber que tipos de plataformas já foram usados e aplicados em situações idênticas, de forma a criar uma ferramenta de controlo em larga escala. Depois de saber a performance das plataformas de IoT estudadas, é feita uma comparação para decidir qual se adequa melhor para este problema.

Seguidamente a descrição do sistema é feita. De forma a entender como a ferramenta corre, uma explicação sobre a ferramenta desenvolvida é feita. É necessário entender como esta comunica com a plataforma escolhida, como navegar na sua interface e que funcionalidades esta providenciam.

Depois a ferramenta é validada e testada. A ferramenta é validada assim que todas as funcionalidades listadas forem testadas e funcionarem. Os testes que foram feitos foram em relação do tempo de resposta da ferramenta. Quanto tempo demoraria esta a atualizar toda a informação relacionada com cada dispositivo ligado à ferramenta.

Em conclusão, são apresentadas algumas possíveis melhorias, de forma a tornar a ferramenta de controlo em larga escala mais robusta

**Keywords:** *Edge*, Indústria 4.0, IoT, escala, *workflow*.





# Acknowledgements

To my teachers and supervisors Pedro Santos and Luís Almeida for giving me this opportunity to work under their wing. Specially professor Pedro Santos, that helped me a lot throughout the development of this dissertation, over the course of the year.

To my parents, without them I wouldn't be where I am. They helped me with everything I needed, did not matter if it was economical or emotional problems, there were always there for me.

To my brother and cousins that I have grown with and followed me on this journey.

To my grandmother Erminda, who raised me to become the person that I am today.

To all my friends, who always helped me with whatever I needed and made me laugh so much throughout my life.

Thank you very much!

Rui Reis



*“Be not afraid of greatness. Some are born great, some achieve greatness,  
and others have greatness thrust upon them.”*

William Shakespeare



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Motivation and goals . . . . .	2
1.3	Structure . . . . .	2
<b>2</b>	<b>Related Work and Background</b>	<b>3</b>
2.1	Industry 4.0 . . . . .	3
2.1.1	Characteristics of Industry 4.0 . . . . .	4
2.1.2	SCADA . . . . .	5
2.1.3	Examples of Industry 4.0 Use-cases . . . . .	6
2.2	Industrial Frameworks . . . . .	7
2.2.1	Arrowhead . . . . .	7
2.2.2	AUTOSAR . . . . .	11
2.2.3	BaSys . . . . .	12
2.2.4	FIWARE . . . . .	13
2.2.5	Node-RED . . . . .	14
2.2.6	Kubernetes . . . . .	15
2.2.7	Discussion . . . . .	17
2.3	ML Workflows . . . . .	18
2.3.1	Introduction to Machine Learning . . . . .	18
2.3.2	Overview of Machine Learning flows . . . . .	19
2.3.3	Applications in the MIRAI examples . . . . .	20
2.4	Visualization Technologies . . . . .	21
2.4.1	Java Spring Boot . . . . .	21
2.4.2	Grafana . . . . .	23
2.5	State-of-the-Art: Application of Integration Frameworks . . . . .	24
2.5.1	Industrial Predictive Maintenance Application . . . . .	24
2.5.2	Integrating an Electric Vehicle Supply Equipment . . . . .	25
2.5.3	Smart Cities . . . . .	26
2.5.4	Node-RED in Industrial Environment . . . . .	26
<b>3</b>	<b>System Implementation</b>	<b>29</b>
3.1	Large-Scale Edge Management Tool (LEM tool) . . . . .	29
3.1.1	Goal . . . . .	29
3.1.2	List of Functionalities . . . . .	29
3.1.3	User Interface . . . . .	30
3.1.4	Integration with other Frameworks . . . . .	32
3.2	Architecture & System components . . . . .	32

3.2.1	Classes Description . . . . .	34
3.2.2	Information Model . . . . .	36
3.2.3	Monitoring Functionality . . . . .	38
3.2.4	Update Functionality . . . . .	39
3.2.5	Deploy Functionality . . . . .	40
3.2.6	Edit Functionality . . . . .	41
3.2.7	Delete Functionality . . . . .	43
3.2.8	Control Functionality . . . . .	44
3.3	Interface with Node-RED . . . . .	46
3.3.1	Communicating with Node-RED . . . . .	46
3.3.2	Limitations of Node-RED . . . . .	48
3.3.3	Monitor Node-RED flows . . . . .	49
<b>4</b>	<b>Validation</b>	<b>53</b>
4.1	Integration of the pipelines developed by MIRAI . . . . .	53
4.1.1	Flow Description . . . . .	53
4.1.2	Integration with LEM tool . . . . .	54
4.2	Validation of Deploy and Delete Functionalities . . . . .	55
4.3	Monitoring - Update Time of Flow Status . . . . .	58
4.3.1	Local Setup Results . . . . .	58
4.3.2	Remote Setup Results . . . . .	63
<b>5</b>	<b>Conclusion</b>	<b>67</b>
5.1	Future Work . . . . .	68
<b>A</b>	<b>Response Times Tables</b>	<b>69</b>
A.1	Local Results per Flows . . . . .	69
A.1.1	1 Flow on n Devices . . . . .	69
A.1.2	5 Flow on n Devices . . . . .	71
A.1.3	10 Flows on n Devices . . . . .	73
A.1.4	15 Flows on n Devices . . . . .	75
A.1.5	20 Flows on n Devices . . . . .	77
A.2	Local Results per Devices . . . . .	79
A.2.1	Flows on 1 Device . . . . .	79
A.2.2	Flows on 4 Devices . . . . .	81
A.3	Remote Results per Devices . . . . .	82
A.3.1	Flows on 1 Device . . . . .	82
A.3.2	Flows on 4 Devices . . . . .	86
<b>B</b>	<b>Traffic Capture</b>	<b>91</b>

# List of Figures

2.1	Possible Industry 4.0 workflow adapted from [5]	4
2.2	Arrowhead local cloud architecture adapted from [3]	7
2.3	Core Systems of Arrowhead Framework adapted from [9]	9
2.4	Cloud service workflow adapted from [3]	10
2.5	AUTOSAR classic platform architecture adapted from [14]	11
2.6	AUTOSAR adaptive platform architecture adapted from [15]	12
2.7	BaSys functional block diagram adapted from [8]	13
2.8	FIWARE architecture adapted from [19]	14
2.9	Node-Red architecture adapted from [24]	15
2.10	Kubernetes architecture adapted from [27]	16
2.11	Machine learning methods adapted from [29]	19
2.12	Standard ML workflow adapted from [31]	19
2.13	Java Spring architecture adapted from [37]	22
2.14	Grafana charts adapted from [38]	23
2.15	Arrowhead cloud architecture in the article [39]	24
2.16	EVSE's station controller in article [40]	25
2.17	System's architecture from [22]	27
3.1	Graphical User Interface Web based	30
3.2	Deploy tab in GUI	31
3.3	Deploy tab in GUI - Delete Option	31
3.4	Deploy tab in GUI - Edit Option	32
3.5	Control tab in GUI	32
3.6	System's Architecture	33
3.7	UML class diagram	35
3.8	Database UML Diagram	36
3.9	Database Table Relations	38
3.10	UML Monitoring Sequence	39
3.11	UML Update Sequence	39
3.12	UML Deploy Device Sequence	40
3.13	UML Deploy Flow Sequence	41
3.14	UML Edit Device Sequence	42
3.15	UML Edit Flow Sequence	42
3.16	UML Delete Flow Sequence	43
3.17	UML Delete Device Sequence	44
3.18	UML Control Sequence	45
3.19	Node-RED API Methods	46
3.20	Small example of JSON code	47

3.21	URL for POST method . . . . .	47
3.22	Response from Node-RED . . . . .	47
3.23	URL for PUT method . . . . .	48
3.24	GET method for JSON string of all existing flows . . . . .	48
3.25	GET method for JSON string of a specific flow . . . . .	48
3.26	Node-RED Node ID . . . . .	49
3.27	URL with the last time the service was executed . . . . .	49
3.28	Set of nodes . . . . .	49
3.29	URL with the flow status . . . . .	50
3.30	Information contained in the timer Template node . . . . .	50
4.1	ML model pipeline developed by MIRAI . . . . .	54
4.2	Adapted pipeline for monitoring . . . . .	55
4.3	Raspberry Pi Setup . . . . .	55
4.4	Node-RED response to Deploy . . . . .	56
4.5	Node-RED response to Delete . . . . .	56
4.6	Node-RED response to Update . . . . .	57
4.7	Interface response to Edit . . . . .	57
4.8	Interface response to Stop . . . . .	57
4.9	Interface response to Start . . . . .	57
4.10	Interface response to Device Disassociation . . . . .	58
4.11	NTP Status in device . . . . .	58
4.12	local Setup . . . . .	59
4.13	Local Setup - Multiple Devices with 1 Flow . . . . .	59
4.14	Local Setup - Multiple Devices with 5 Flows . . . . .	60
4.15	Local Setup - Multiple Devices with 10 Flows . . . . .	60
4.16	Local Setup - Multiple Devices with 15 Flows . . . . .	61
4.17	Local Setup - Multiple Devices with 20 Flows . . . . .	61
4.18	Local Setup - 1 Device with Multiple Flows . . . . .	62
4.19	Local Setup - 4 Devices with Multiple Flows . . . . .	62
4.20	Remote Setup . . . . .	63
4.21	Remote Setup - 1 Device with Multiple Flows . . . . .	64
4.22	Remote Setup - 4 Devices with Multiple Flows . . . . .	64



# List of Tables

2.1	Accessibility (from [8]) . . . . .	18
3.1	Example of data in Devices Table . . . . .	37
3.2	Example of data in Nodes Table . . . . .	37
3.3	Table of Services . . . . .	37
3.4	Example of data in Flows table . . . . .	37
A.1	Local Setup - Response Time 1 Flow with 1 Device . . . . .	69
A.2	Local Setup - Response Time 1 Flow with 2 Devices . . . . .	70
A.3	Local Setup - Response Time 1 Flow with 3 Devices . . . . .	70
A.4	Local Setup - Response Time 1 Flow with 4 Devices . . . . .	71
A.5	Local Setup - Response Time 5 Flows with 1 Device . . . . .	71
A.6	Local Setup - Response Time 5 Flows with 2 Devices . . . . .	72
A.7	Local Setup - Response Time 5 Flows with 3 Devices . . . . .	72
A.8	Local Setup - Response Time 5 Flows with 4 Devices . . . . .	73
A.9	Local Setup - Response Time 10 Flows with 1 Device . . . . .	73
A.10	Local Setup - Response Time 10 Flows with 2 Devices . . . . .	74
A.11	Local Setup - Response Time 10 Flows with 3 Devices . . . . .	74
A.12	Local Setup - Response Time 10 Flows with 4 Devices . . . . .	75
A.13	Local Setup - Response Time 15 Flows with 1 Device . . . . .	75
A.14	Local Setup - Response Time 15 Flows with 2 Devices . . . . .	76
A.15	Local Setup - Response Time 15 Flows with 3 Devices . . . . .	76
A.16	Local Setup - Response Time 15 Flows with 4 Devices . . . . .	77
A.17	Local Setup - Response Time 20 Flows with 1 Device . . . . .	77
A.18	Local Setup - Response Time 20 Flows with 2 Devices . . . . .	78
A.19	Local Setup - Response Time 20 Flows with 3 Devices . . . . .	78
A.20	Local Setup - Response Time 20 Flows with 4 Devices . . . . .	79
A.21	Local Setup - Response Time 1 Device with 2 Flows . . . . .	79
A.22	Local Setup - Response Time 1 Device with 3 Flows . . . . .	80
A.23	Local Setup - Response Time 1 Device with 4 Flows . . . . .	80
A.24	Local Setup - Response Time 4 Device with 2 Flows . . . . .	81
A.25	Local Setup - Response Time 4 Device with 3 Flows . . . . .	81
A.26	Local Setup - Response Time 4 Device with 4 Flows . . . . .	82
A.27	Remote Setup - Response Time 1 Device with 1 Flow . . . . .	82
A.28	Remote Setup - Response Time 1 Device with 2 Flows . . . . .	83
A.29	Remote Setup - Response Time 1 Device with 3 Flows . . . . .	83
A.30	Remote Setup - Response Time 1 Device with 4 Flows . . . . .	84
A.31	Remote Setup - Response Time 1 Device with 5 Flows . . . . .	84

A.32 Remote Setup - Response Time 1 Device with 10 Flows . . . . .	85
A.33 Remote Setup - Response Time 1 Device with 15 Flows . . . . .	85
A.34 Remote Setup - Response Time 1 Device with 20 Flows . . . . .	86
A.35 Remote Setup - Response Time 4 Devices with 1 Flow . . . . .	86
A.36 Remote Setup - Response Time 4 Devices with 2 Flows . . . . .	87
A.37 Remote Setup - Response Time 4 Devices with 3 Flows . . . . .	87
A.38 Remote Setup - Response Time 4 Devices with 4 Flows . . . . .	88
A.39 Remote Setup - Response Time 4 Devices with 5 Flows . . . . .	88
A.40 Remote Setup - Response Time 4 Devices with 10 Flows . . . . .	89
A.41 Remote Setup - Response Time 4 Devices with 15 Flows . . . . .	89
A.42 Remote Setup - Response Time 4 Devices with 20 Flows . . . . .	90



# Abbreviations

AH	Arrowhead
AI	Artificial Intelligence
AOP	Aspect Oriented Programming
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BaSys	Basic System
BSW	Basic Software
CAN	Controller Area Network
CB	Context Broker
CBGE	Context Broker Generic Enabler
CPS	Cyber-Physical Systems
CSV	Comma-Separated Values
DDoS	Distributed Denial of Service
DI	Dependency Injection
DNS-SD	Domain Name System - Service Discovery
ECU	Electronic Control Unit
ETSI	European Telecommunications Standardization Institute
EVSE	Electric Vehicles Supply Equipment
GE	Generic Enabler
GSD	Global Service Discovery
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
ICN	Inter-Cloud Negotiation
ICT	Information and Communications Technology
IIoT	Industrial Internet of Things
IoC	Inversion of Control
IoT	Internet of Things
IoS	Internet of Service
IT	Information Technology
J2EE	Java 2 Platform, Enterprise Edition
JPA	Java Persistence API
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LD	Linked Data
LEM	Large-Scale Edge Management
LIN	Local Interconnect Network
MES	Manufacturing Execution System
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
MVC	Model-View-Controller
NGSI	Next Generation Service Interface

NTP	Network Time Protocol
PDM	Predictive Maintenance
PID	Proportional–Integral–Derivative
PLC	Programmable Logic Controller
QoS	Quality of Service
REST	Representational State Transfer
REST WS	Restfull WebServices
RTDB	Real-time Database
RTE	Runtime Environment
SCADA	Supervisory Control and Data Acquisition
SOA	Service-Oriented Architecture
SoS	System of Systems
SSH	Secure Shell
TCP/IP	Transmission Control Protocol/Internet Protocol
URL	Uniform Resource Locators
VFB	Virtual Function Bus
VM	Virtual Machine
VPN	Virtual Private Network



# Chapter 1

## Introduction

This introductory section presents a brief contextualization. In section 1.1 there is a contextualization on where the project is based, with a brief explanation related to distributed Industry 4.0 applications. In the section 1.2, there is a brief explanation, to know why there was the need to come up with this project and also the main goals. The structure of this report will be explained in the section 1.3.

### 1.1 Context

Since the beginning of this century, there has been a keen interest in using Internet of Things (IoT) applications to leverage on cloud infrastructures to attack storage and computational limitations as well as constraints at the end and edge nodes. Although this is the most common approach, there are many problems associated to it, because edge devices have limited communication capabilities. This originates a new problem, due to cost constraints or bandwidth limitations, which is the data storage in the network. These limitations might lead to important data discarding [1].

The birth of end and edge nodes with improved computational and storage abilities enabled the performance of highly resource demanding computations of systems. This is done locally at the edge, only requiring the cloud backend to communicate, store and process results from multiple edge devices. Furthermore, the increase of computation capabilities made the collaboration of neighbouring edge devices possible, supporting each other with their available computational resources before unloading them to the cloud backend.

This greatly supported the development of Distributed Artificial Intelligence (AI) technologies. Although local AI platforms are good for this matter, they are typically based on high performance hardware, with expensive and high-power consuming processors [2]. A way to support AI technologies is the use of embedded computing devices in a parallel and distributed architecture, to overcome many of the difficulties mentioned above. By using these embedded systems, the computation and storage can be located near the data source, surpassing the bandwidth and latency

problems. In case there is a failure point, the tasks can be done by a free neighbour edge device, making the workflow robust.

## 1.2 Motivation and goals

This thesis was born from the need to create a framework or leverage from existing solutions in order to answer all problems that MIRAI project [1] is a part of. MIRAI is a research group that focuses on AI approach for edge devices and it has set their minds on five main uses cases provided by owners of companies from three different countries, Belgium, Portugal and Turkey. These itemized cases cover the domains of renewable energy management, Internet provisioning for households, road traffic management, water consumption management and continuous auto configuration of Industrial Controllers at Edge for the dyeing of textiles [1].

The main issue is that currently there is no suitable framework up to meet all use case requirements. With this in mind, MIRAI project aims to build a framework that can horizontally distribute applications among the edge devices, in addition to the vertical computing of a cloud. In order for that to happen, this thesis was born, where a framework must be created or leveraged from existing options to suit these expectations.

The goal to this project is to develop a tool to monitor and control multiple devices where you can deploy sets of flows in a large scale. This tool will be in charge of multiple connected devices, setting up workflows that will run ML modules. In order to manage and provide reports of the operations performances there is the need to develop an User Interface to complement the backend operations and give the user the opportunity to easily monitor, control and deploy flows or devices at his own will.

## 1.3 Structure

This preparation for thesis is divided into four chapters. The first one 1 being the this current one where it includes the motivation for this thesis as well its goals.

The second chapter 2 is dedicated to the related work and background, including a brief explanation of the theme where the thesis is settled as well as the components that can be used to reach the goals. A study about where this type of related work as been applied is done as well.

The third chapter 3 talks about the developed tool, its architecture and its components and how it integrates with the framework deployed in the target devices.

The fourth chapter 4 talks about the integration of the tool with the services developed by MIRAI and how it behaves, in order to obtain results and validate its development.

The fifth and final chapter 5 presents the conclusions, possible improvements to the tool and future works.



## Chapter 2

# Related Work and Background

IoT has been growing faster and faster, playing an important role in our daily lives in terms of industry. Currently the number of connected devices has already surpassed the world population and new device integration approaches are being introduced with a higher frequency [3]. The IoT adoption has reached the industry domain, and it has been massively growing in this area. It has been coined Industrial IoT and is often presented as part of Industry 4.0. It enables to connect multiple different devices where these share data between each other.

Many frameworks have been presented in order to manage the link between IoT devices. A particular challenge is the management of data for Machine Learning workflows. ML techniques are necessary to optimize many industrial processes, to achieve the removal of a big percentage of human interaction in the workflow.

This chapter is divided in four main sections. In section 2.1 it is provided a background to where this thesis will be settled upon. In the section 2.2, there is done a study of multiple IoT frameworks and a comparison between them, to understand which framework should be used for this task ahead. In the section 2.3 it is done a study about machine learning algorithms as well as its workflows. In the section 2.4 there will be a theoretical background based on three visual frameworks that can possibly be integrated as the GUI in the project. In the section 2.5 it is done a bibliographical study about similar projects and why and how they implemented these frameworks and for what purposes.

### 2.1 Industry 4.0

Industry 4.0, of which Industrial Internet of Things (IIoT) is part of, has a set goal of facilitating better performance, lower costs and higher quality in many fields of industry. In order to achieve Industry 4.0 there needs to be an integration of multiple components and systems collaborating towards a common purpose, such as Cyber-physical systems (CPS), IoT, cloud and cognitive computing.

### 2.1.1 Characteristics of Industry 4.0

At the creation of an Industry 4.0 system, some aspects have to be considered: the system must possess interoperability, technical assistance, information transparency and decentralized decision making [4]. With all these aspect together, it is possible to allow advanced manufacturing hardware and sophisticated software to collaborate effectively to optimize operations and automate manufacturing processes offering advanced capabilities like automation of tasks customable and adaptive devices and machines, reduction of human interaction with machines via digital sensors, controls and automated decisions, improvement of measurement and monitoring procedures, collection and storage of real-time data across various areas of a manufacturing plant, introduction of intelligent algorithms, easy integration of different technology models in the manufacturing industry, among others.

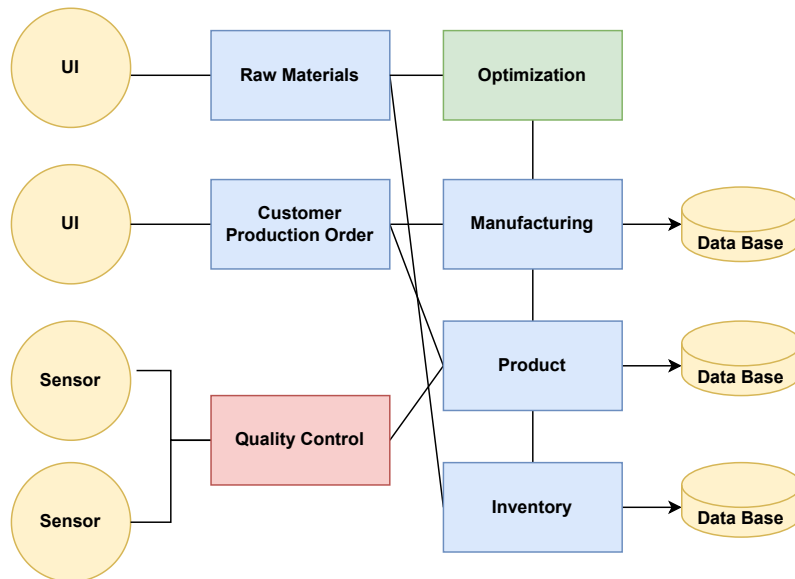


Figure 2.1: Possible Industry 4.0 workflow adapted from [5]

The Industry 4.0 success is mainly acquired from the revolutionary hardware and software technology innovations (figure 2.1 shows a possible workflow). This is enabled due to the collective advancements in various Information and Communications Technology (ICT) fields like:

1. IIoT to allow the connection between multiple manufacturing devices and machines in a network;
2. Internet of Service (IoS) to allow Internet to provide services for manufacturing related systems and organizations;
3. Manufacturing CPS to make useful interactions between cyber world and physical world easier;
4. Cloud manufacturing to allow scalable computation, data storage and smart services on demand;

5. Fog manufacturing to have real-time control and low latency support;
6. Manufacturing data analysis for intelligent decision making according to the gathered data.

In order to create an Industry 4.0 framework there are some design principles that must be followed. These principles answer to the main requirements imposed by an IIoT application and they are **interoperability** (the ability to connect and communicate with multiple different devices), **service oriented** (ability to present manufacturing process functions as a set of services), **decentralization** (ability of systems to make their own decisions), **real-time capability** (ability to collect and analyze data instantaneously), **modularity** (ability of flexible changing, expanding and enhancing singular modules to correspond new requirements existing in processes or build new ones) and **virtualization** (ability to monitor processes by making virtual copies that can be used to simulate and measure environments) [4], [5].

In industrial context, one concept is important. The term *pipeline* or *flow* refers to the sequence of steps and/or infrastructure required to one or more transformation from raw materials to a finished product. The terms can be used interchangeably, although sometimes *pipeline* may refer more to the infrastructure and *flow* to an instance of use of a pipeline, in which specific materials or data flow through the pipeline. Examples of pipelines/flows are a production line of automobiles, in which every station transforms its input into something that grows in a vehicle. Pipelines and flows exist equally in software-based applications, such as video editing or machine learning.

### 2.1.2 SCADA

SCADA systems were made to monitor and control industrial and critical infrastructure functions, like the ones mentioned in 2.1.3. It is not a full control system but rather a supervisory one. This is a software package that is placed above the hardware and it is interfaced via PLCs. This system usually controls plants that may have thousands of inputs/outputs channels [6].

SCADA products are multi-task and based upon RTDB (real-time database) that are located in one or multiple servers. Data acquisition is the servers' domain and these are responsible for handling a set of parameters which they are usually connected to. SCADA has two types of communications which are *internal* or to devices.

The **internal communication** is server-client or server-server and it's generally done in a publish-subscriber manner or event-trigger and uses the protocol TCP/IP (Transmission Control Protocol/Internet Protocol) for communication [7].

In the **devices communication** there is a controller polling rate done by the data servers and defined by the user. This polling rate can differ depending on the parameters. The controllers respond to these polls and they pass the requested parameters to the data servers. A very common approach is the use of data stamps so it is easier to track when this data was polled. If the controller and communication protocol allow the request of unsolicited data, the products will support it.

SCADA is scalable, which means it can track multiple devices at will. The products achieve scalability by having multiple data servers connected to multiple controllers. Each data server is responsible for handling a sub-set of process variables and it possesses their own configuration.

### 2.1.3 Examples of Industry 4.0 Use-cases

Since the goal of this project is motivated by the MIRAI examples, there is the need to know a bit about them. They are in total five use cases and they are applied in different fields [1].

- Distributed energy assets have been increasing in the past years and electricity production has become increasingly oscillating. For example the solar panels need to be able to adapt faster than ever, in order to obtain the most energy resource. The solution that needs to be implemented in the **Distributed Renewable Energy Systems** use case is the management of solar and wind assets, by monitoring, report and improve their performance as well as organizing their maintenance. Thanks to MIRAI, the aim is to provide optimised control of the renewable energy plant assets as well as provide real-time monitoring of the status.
- Since Distributed Denial of Service (DDoS) attacks are one of the main problems related to availability and security of internet providers, NOS (a Portuguese telecommunication company) has made a partnership with MIRAI to come up with a solution for this problem. The solution of the **Secure Internet Provisioning** use case is to identify malicious traffic, differentiate it from the normal one and block it. By the use of ML modules and framework deployment is it possible to reach this goal.
- Road accidents are a problem in a daily basis. Stress can cause a driver to be distracted, making him propitious to have an accident. By the use of sensors for traffic monitoring as well as the use of cameras, it is possible to forecast if a driver is dangerous or not in that moment. MIRAI's goal in **Traffic Management** use case, is to treat the data received by the pedal sensors and cameras in order to know if a driver is suited to be driving at that time.
- Water leakage problems greatly affect houses if the leakage is not detected rapidly. What MIRAI is set to do in **Water Management** use case, is to monitor and control water's flow in order to prevent or quickly detect a leakage to avoid building damages.
- In the textile field, the dyeing process takes five to twelve hours depending on many process parameters such as desired colour, fabrics, chemicals, among others. Since Proportional–Integral–Derivative (PID) parameters are tuned by technicians according to their experience during the installation of the dyeing machine, it is hard to control the used resources, such as energy, steam, water, chemical, dye and time. In **Continuous Auto Configuration of Industrial Controllers at Edge** use case, MIRAI needs to tune the PID according to the output of the AI algorithm working on the process controllers, which are the IoT devices operating at the edge.

## 2.2 Industrial Frameworks

To develop a tool for large scale workflow control, there is the need to look into the most suitable framework for this work that can be ran in edge devices. The criteria used to select the framework is based on the following steps [8].

- i) The industrial character and focus on the Industry 4.0 objectives;
- ii) The provisioning of architectural and technical solutions for industrial contexts beyond individual IoT solutions;
- iii) The targeting of System of Systems (SoS) applications based on the IoT;
- iv) The reputation of the consortia members and support from the large projects;
- v) Their future potential and emergence;
- vi) The marked evolution from the cloud to the edge.

We now review six selected frameworks for system integration and management.

### 2.2.1 Arrowhead

Arrowhead (AH) framework is based on a Service-Oriented Architecture (SOA). Its goal is to offer automation capabilities, like security, real-time control and engineering simplicity without disabling IIoT and device data sharing at the service level.

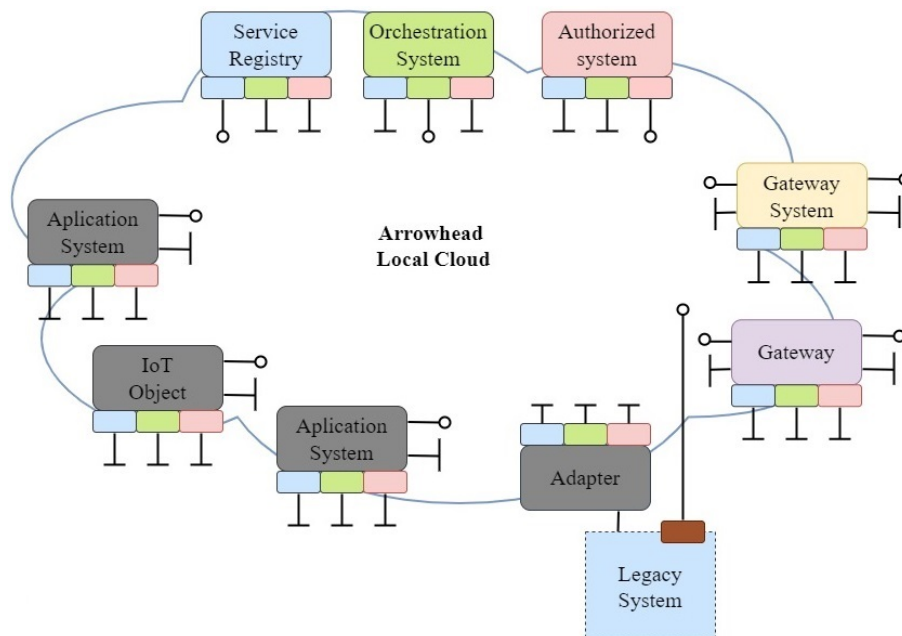


Figure 2.2: Arrowhead local cloud architecture adapted from [3]

This framework consists of local clouds, devices, services and systems. The local cloud is an important concept to designate the communication and a computational environment capable of

providing main services for the development of automation tasks in a safe way. The figure 2.2 demonstrates an Arrowhead architecture.

AH framework's goal is to overcome the existing problem of system and device diversity by granting a common infrastructure with standard interfaces for services' provision, in order to reach full interoperability. A key point in the IoT scenario is the interoperability between different systems and devices, to attain a high number of connected devices. This number has already surpassed the world population [3].

There are three mandatory core systems, which are the Service Registry, Orchestration System and Authorization System [9].

The **Service Registry**, as the name says, provides registration capabilities and works as a storage unit. It saves all the information related to the registry, like the service description, communication protocols, interfaces, among others. It's also in charge of looking up services requested by other services. This lookup operation uses a Domain Name System - Service Discovery (DNS-SD) protocol [10].

The **Orchestration System** is responsible for the coordination of the services' interoperability throughout the framework. Once a request by the consumer is made, this system needs to answer it with the available services on the Service Registry. The Orchestrator selects the best service producer that meets what is needed by the request. It is also in charge of load balancing and fault tolerance on the side of the service producer.

The **Authorization System** manages the correct flow of the services by granting rights and permissions for that to happen.

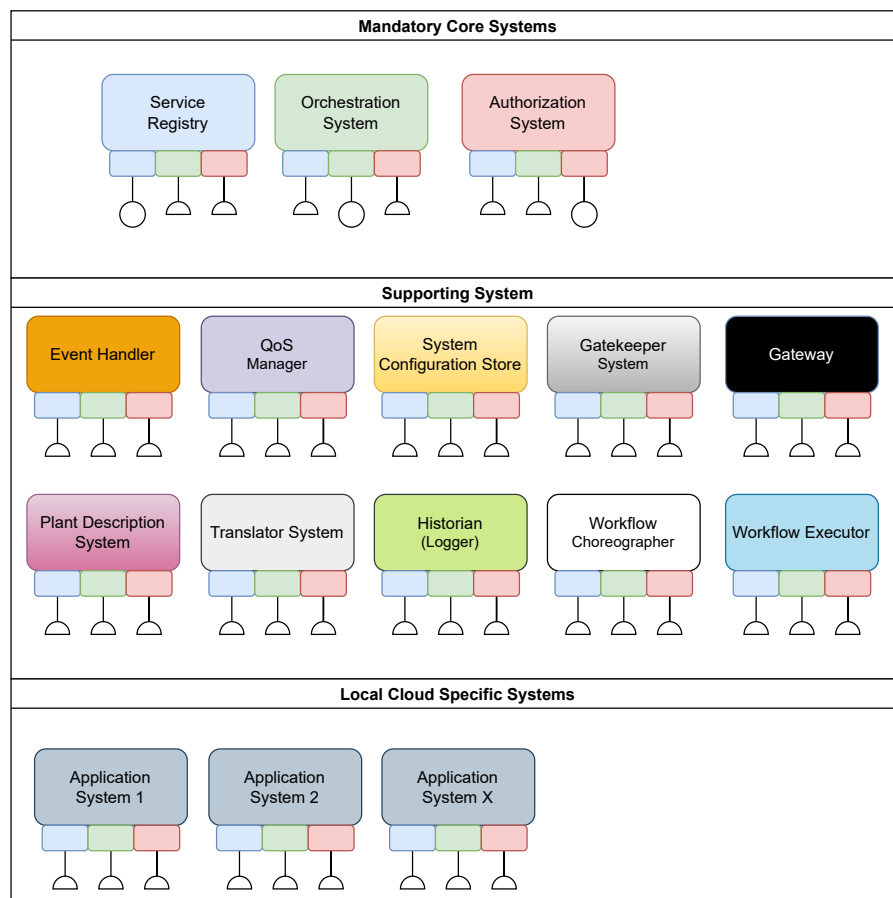


Figure 2.3: Core Systems of Arrowhead Framework adapted from [9]

In the AH local cloud there are a number of optional supporting services. These are the Event Handler, Quality of Service (QoS) Manager, System Configuration Store, Gatekeeper System, Gateway System, Plant Description System, Translator System, Historian (Logger), Workflow Choreographer and Workflow executor [9]. The two main supporting services are the **Gatekeeper System** and the **Gateway System**. Although they are not mandatory, these two are often deployed because they are in charge of the communication between different local clouds.

It is the **Gatekeeper System** responsibility to manage the control information for the inter-cloud communication in the orchestration processes but it isn't directly in charge of the data flow between consumer and producer. This system contains two services. The Global Service Discovery (GSD) that locates the best fitting service in the nearby clouds and the Inter-Cloud Negotiation (ICN) that is responsible for the establishment of a mutual trusted connection between the two different local clouds. This system works side by side with the Orchestration Systems of both local clouds.

The **Gateway System** works as a mediator between the producer and the consumer by establishing and managing a session between them. It creates a connection (where all the traffic will flow) between the producers and consumer.

The **Event Handler** is required to circulate status and event information, the **Workflow Choreographer** is necessary to trigger the next step in the process execution and the **Plant Description System** is to keep track of SoS or plant related metadata [11]–[13].

The Arrowhead framework service is defined as what is exchanged between producer and consumer, the flow of exchange can be observed in the figure 2.4.

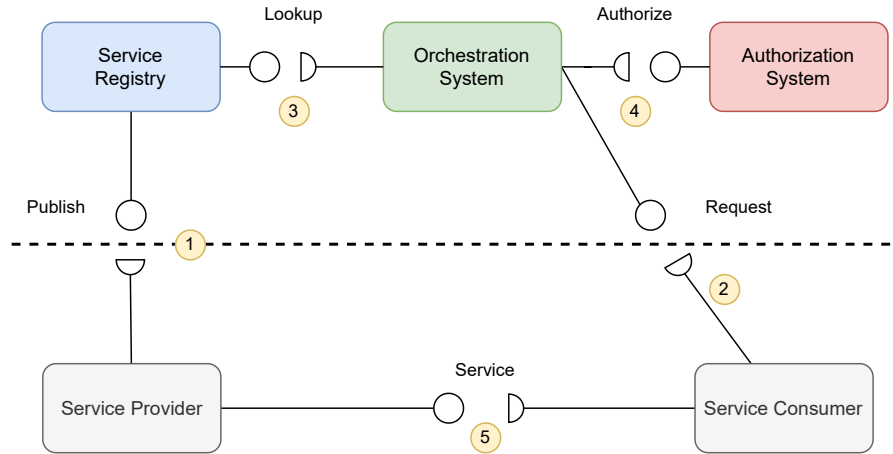


Figure 2.4: Cloud service workflow adapted from [3]

As you can see in the figure 2.4, the flow can be explained in five steps.

In the **first step**, the service provider by publishing the metadata, which contains information about the producer endpoint, authorization information, protocols, service interface, among others, in the Service Registry, becomes accessible to all Arrowhead users.

In the **second step**, sends information about itself and which type of service he wants to consume to the Orchestration System.

In the **third step**, the Orchestration System send the information requested by the consumer to the Service Registry and tries to provide a service lookup.

In the **fourth step**, the Orchestration System queries the Authorization System to see if he consumer has the rights to use the service provided.

Last but not least, in the **fifth step**, if the request by the service consumer matches with at least one registered service on the Service Registry and it has the rights to use it, the Authorization System permits the exchange and the Orchestration System provides the service lookup information to the consumer. The information contained is the same as the one provided in the metadata from the Service Provider to the Service Registry. The Service Consumer can now start making use of the available service.

Once all these steps are met, the flow between producer and consumer is performed in a point-to-point topology, the Arrowhead Framework is no longer involved.



### 2.2.2 AUTOSAR

Automotive Open System Architecture (AUTOSAR) is a framework divided in multiple layers for intelligent mobility, providing standards for Electronic Control Units (ECUs). The specifications of AUTOSAR differ from other high-level oriented frameworks [8]. This framework application scope is automotive ECUs, with a great interaction with hardware, connected to the vehicle networks such as Controller Area Network (CAN), Local Interconnect Network (LIN) and Ethernet and they are run in microcontrollers with real-time features. AUTOSAR has two different types of platforms, the adaptive one and the classic one.

The AUTOSAR classic platform architecture has three different software layers that are run on a microcontroller: the Application, Runtime Environment (RTE) and Basic Software (BSW). The **Application layer** is mostly hardware independent. The communication in the framework is done between software components and the access to **BSW** is via **RTE** (full interface for applications). The **BSW** is separated in three major layers and complex drivers which are ECU and microcontroller abstractions and Services. Infrastructure for the system, memory and communication services are functional groups of the Service layer.

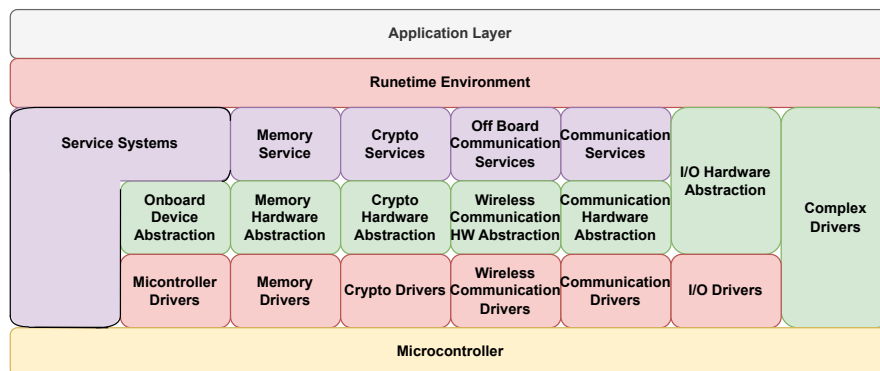


Figure 2.5: AUTOSAR classic platform architecture adapted from [14]

AUTOSAR introduced an harmonized methodology approach for the development of automotive software to complement its software architecture [14]. This was required by the need to overcome collaboration difficulties between different parties involved in automotive projects. This methodology defines the dependencies of multiple activities on products and it supports them by using its tools.

AUTOSAR adaptive platform implements the AUTOSAR Runtime for Adaptive Applications, where two types of interfaces are available which are services and Application Programming Interfaces (API). This platform is subsists of functional clusters that are grouped in services. These clusters assemble functionalities of the adaptive platform, define clustering of requirements specification and describe the behavior of the software from the point of view of the application and the network.

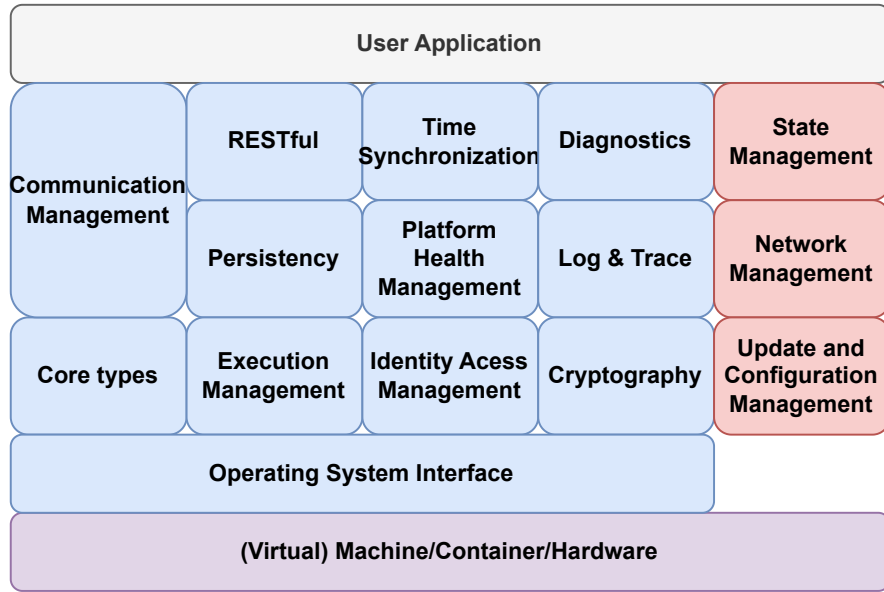


Figure 2.6: AUTOSAR adaptive platform architecture adapted from [15]

AUTOSAR extends the already existing methodology in order to have a common approach for both classic and adaptive platforms [15].

### 2.2.3 BaSys

In the present time, manufacturing facilities are designed for the mass production of identical assets. Although manufacturing systems have a certain flexible, these usually come associated with high cost. In order to fight that, there was a developed a researched project called **BaSys 4.0**, which is the development of basic systems in the Industry 4.0.

BaSys 4.0 defines a reference architecture for production systems that allows the transition to Industry 4.0. To implement this concept Basic System (BaSys) created the open source platform known as Eclipse BaSyx that is handled in Java, C++ and C# [16].

BaSys design was founded on three central pillars which are the creation of a structured RTE, process planning and the use of digital twins [8]. A digital twin is a virtual model designed to accurately reflect a physical object [5].

One of the major goals of BaSys 4.0 for Industry 4.0 is to address the shifting of production processes. In attempt to fight this problem a new concept has emerged known as changeable production. The idea of this concept is to address unplanned changes of production processes, such as unaware steps that are required for the development of a product that were not known before the creation of the production line. Changeable production enables manufacturing of different products, dynamically add and remove production resources and device capabilities, change of devices for another compatible one and moving of software components. BaSyx components are divided in four levels [17].

1. Field level that consists of automation devices, sensors and actuators without the use of a specific BaSys conforming interface.
2. Device level offers a conforming interface to BaSys 4.0 by the use of automation devices.
3. Middleware level reuses Industry 4.0 components that implement necessary generic capabilities for the production lines. A good example of these capabilities are registry and discovery services, Asset Administration Shell providers and protocol gateways.
4. Plant level contains high level plant components to manage, monitor and optimize the production.

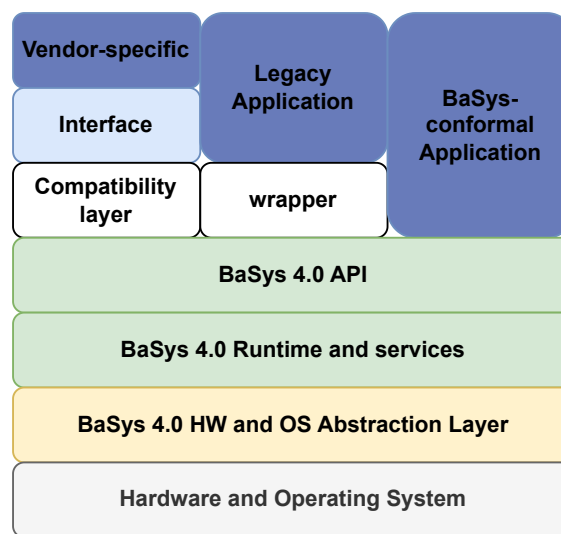


Figure 2.7: BaSys functional block diagram adapted from [8]

Connectivity between the shopfloor (end-to-end digitization of production) and the Information Technology (IT), changeable production processes, great deal of data analysis of production processes and prognostic maintenance are some of the components that BaSys provide to implement in Industry 4.0 [17]. The functional block diagram of BaSys can be seen in the figure 2.7.

#### 2.2.4 FIWARE

FIWARE is an open source platform and it offers basic modules to develop and create IoT applications. By combining various open source modules it enables its use in various sectors. This platform has a growing community which means it will be continuously improved [18]. The only mandatory component of any FIWARE platform is a Context Broker Generic Enabler (CBGE), supplying a cornerstone function necessary in any smart application, such as information management, update performing and granting access to data. FIWARE Context Broker (CB) exports NGSI that is used for the integration of components to update or consume context information. FIWARE NGSI API specifications have evolved, now aligning with European Telecommunications

Standardization Institute ETSI NGSI-LD standard (European Telecommunications Standardization Institute Next Generation Service Interface-Linked Data) [19].

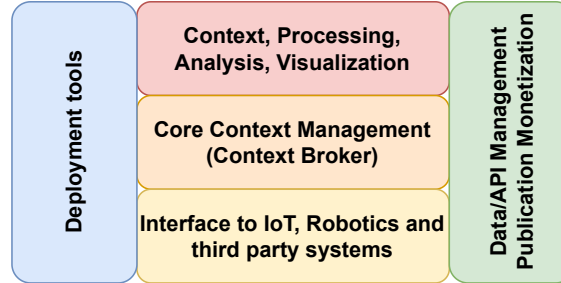


Figure 2.8: FIWARE architecture adapted from [19]

FIWARE CB is the main component in charge of gathering, managing and processing contextual information and it enables the systems to perform updates and access the context state. The interactions between CB and the additional platform components are done by the NGSI restful API [8]. FIWARE GE has its eyes set on dealing with three main topics [19].

**Interfacing with the IoT, robots and third-party systems**, to capture updates and translate required actuation on the context information.

**Context data/API management, publication and monetization**, supporting the usage control and granting the opportunity to earn some extra income from part of the managed context data.

**Process, analysis and virtualization of context information** by implementing an expected smart behaviour on the application and assisting the end user in making smart decisions.

Some of the FIWARE (architecture can be seen in figure 2.8) domains are inserted in smart energy, Wilma module (smart plants) and Green routing [8], [20], [21].

### 2.2.5 Node-RED

Node-RED is an open source development tool that is flow oriented and it was developed by IBM (International Business Machines Corporation) Emerging Technology for the integration of IoT devices, online services and APIs [22]. It's JavaScript based, built on a Node.js platform, providing a browser interface for flow editing. It has multiple node options that are represented by different icons. The browser interface gives two options to the user. This tool allows the developers to create flows for data processing, controlling services or event triggered alarms at will, by connecting input and processing nodes to output nodes [23]. The runtime supports inter-node and inter-flow communication which allows the possibility of having multiple flows running at the same time communicating between themselves. Node-red architecture can be seen in the following figure 2.9.

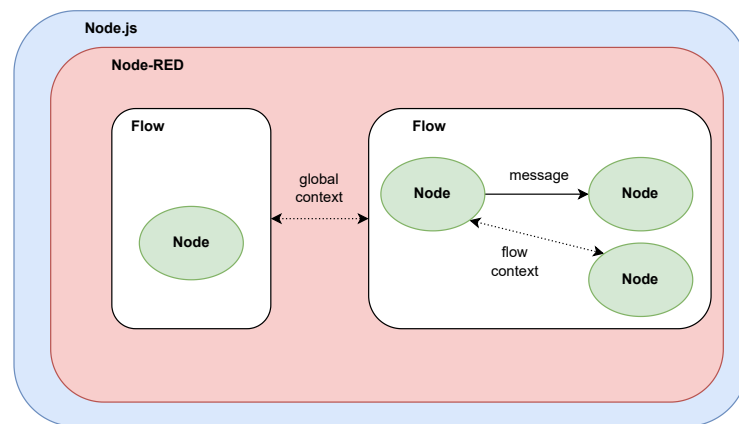


Figure 2.9: Node-Red architecture adapted from [24]

A **node** is a reactive Node.js application that is triggered by receiving a message on an input port and sends the results of its computations to an output. Each node has a unique ID, this means that two nodes cannot have the same identification.

A set of connected nodes is known as a **flow**. You either build the flow from scratch by dragging, dropping and wiring the nodes, making a flow, or by importing a JavaScript code containing the flow information. These flows are stored using JSON (JavaScript Object Notation).

**Contexts** provide a shared communication channel between the multiple different nodes without the use of explicit messages that pass through a flow. This means that the visible wiring between nodes in the user interface is a partly depicted information that exists in a flow. Node-RED defines three scope levels for the contexts. The node, the flow and the global. The node scope possesses information only visible to the node that sets the value. The flow scope possesses information visible to all nodes in the flow where the value is set. The global allows all nodes in any flow to have access to that value. Using a global scope is good in case there is a flow monitoring the input values from a sensor and there is another flow updating its status. This is the best way of sharing information of a certain set of values [24].

In terms of security, this tool relies on the user. This means that Node-RED platform needs to be ran on a trusted network, ensuring that the data is processed in an user controlled environment. The official documentation includes security patterns that include basic authentication mechanisms responsible for the access to the existing nodes and wires [24],[25].

This framework enables multi instances creation. This means that it is possible to create multiple instances of Node-RED in the same device by the use of multiples ports.

### 2.2.6 Kubernetes

Kubernetes is an orchestrator that oversees containerized applications. These are applications that are run in containers. A container is a form of operation system virtualization. Kubernetes enables the configuring and deployment of applications while automatically managing their life-cycles, storage and service discovery [26]. Its clusters primarily consist of sets of nodes (Master

and Worker). The Master contains the API and it is responsible of the authentication and authorization on an operation level.

Kubernetes is based on seven components. Etcd, API server, Scheduler, Controller Manager, container runtime, Kubelet and Kubernetes Proxy [27]. Its architecture can be seen in the figure 2.10.

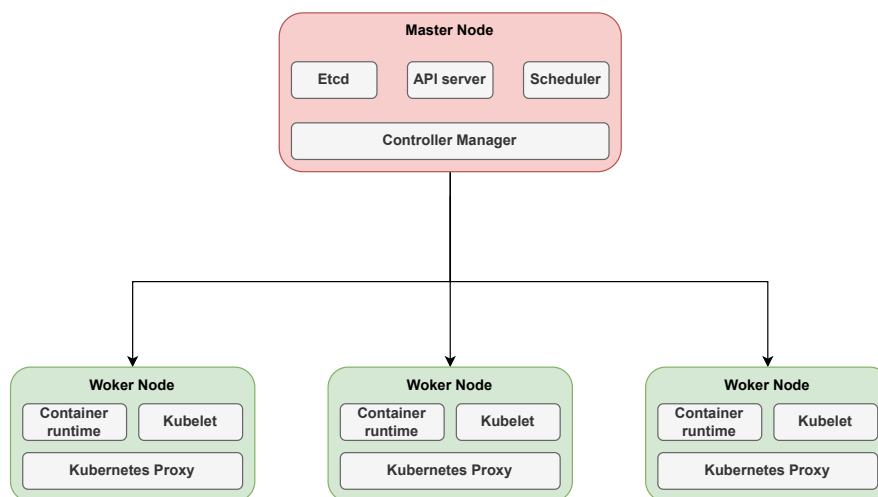


Figure 2.10: Kubernetes architecture adapted from [27]

As it can be seen in the figure, Master node is consisted of Etcd, API server, Scheduler and Controller Manager and Worker node consists of container runtime, Kubelet and Kubernetes Proxy. The Master components are in charge of automating tasks in charge of the management of the deployed application instances' life-cycle, these instances are known as Pods [28].

**Etcd** is a key value storage. It stores the organized data that can be used by each node in the group. It's a high access key that can be passed on among the various groups and can only be open by the API server due to the its sensitive data.

The **API server** updates an interface. So it's required the use of different instruments and libraries to communicate with it. This API uses Kubeconfig to make the GUI reactive.

The **Scheduler** is an administration in charge of handling the existing tasks. It tracks the burden of the workload usage and it disperses into multiple hubs if necessary until the end of the task.

The **Controller Manager** is a non-ending loop in charge of controlling the system. This controller watches the status of the cluster through the API server and if required makes changes to attain the desired state.

The **Container runtime** is the component responsible for the running the containers in the system. This is where the applications will be ran.

**Kubelet** is a running agent in each node of the cluster. It ensures that the containers are running in a node. It makes sure that the specifications of a Pod are running and its status is good.

**Kubernetes Proxy** that runs in every node and helps the outer host in getting access to the administrations. It maintains all the network rules on the nodes. These rules allow network communication from networks inside or outside the cluster to the Pods.

### 2.2.7 Discussion

After a study done in [8], it is possible to make a comparison between all four mentioned frameworks. This study was made based their features and functional principles and it can be divided in five key characteristics: real-time features, runtime features, centralized/distributed approach, hardware requirements and QoS.

**Arrowhead**, **AUTOSAR Node-RED** and **Kubernetes** present real-time behaviour, making them a good option in this key. The same can't be said about **FIWARE**: although it has fast responses, it does not support real-time behaviour. **BaSys** considers real-time no longer essential at the process level, so instead of having this kind of behaviour, it uses Programmable Logic Controllers (PLC) to control the real-time production steps.

All the frameworks mentioned above have runtime features. **Arrowhead** and **Kubernetes** possess runtime functionalities the orchestration system is capable of providing dynamic orchestration and authorization in runtime patterns. It also has more systems capable of working in runtime such as Event Handler and Data Manager. **AUTOSAR** has the RTE layer, that is in charge of the communication between software components and their scheduling, and it is one of its key components. It can also trace errors and dynamically link services and clients in runtime. **BaSys** also possesses a RTE Virtual Function Bus (VFB). The runtime and service layer can provide sets of services in order to manage functional components as well as run hardware independent code. **FIWARE** has some features that includes runtime monitoring attributes of the object store and can add a security runtime component to verify obedience with the data sharing policies. **Node-RED** does not possess an orchestrator.

**AUTOSAR**, **BaSys** and **FIWARE** have flexible and expandable systems but they are centralized. **Arrowhead** framework has a distributed system where all functionalities and main services are distributed among different core systems instead of having an unique middleware. This framework uses different local clouds and it is capable of communicating securely with each other. **Kubernetes** is ran in containers which provides a great large-scale workflow control. **Node-RED** can be deployed in multiple devices but needs something on the top of the architecture to that controls in which devices it is working.

In terms of hardware requirements **AUTOSAR** was designed to fit the resource constraints of the devices and **Arrowhead** can be deployed in a large range of devices, the same goes for **Kubernetes** and **Node-RED**. **FIWARE** and **BaSys** require a certain amount of computational power, which imposes restrictions in small devices.

**Arrowhead** provides a QoS manager core system, **AUTOSAR** has an Ethernet driver that is support by QoS, **FIWARE** possesses a defined networking block that presents QoS, **BaSys** does not have any tool that provides QoS, which means it has to ensure QoS from other protocols.

Table 2.1: Accessibility (from [8])

	<b>Specification</b>	<b>Code</b>	<b>Tutorial</b>	<b>Examples</b>	<b>Orchestrator</b>
Arrowhead	Easy	Easy	Medium	Easy	Yes
AUTOSAR	Easy	Difficult	Medium	Medium	Yes
BaSys	Difficult	-	-	Medium	Yes
FIWARE	Easy	Easy	Easy	Easy	Yes
Kubernetes	-	-	-	-	Yes
Node-RED	-	-	-	-	No

Analysing all the information mentioned above in the table 2.1, **Arrowhead** and **Node-RED** are the most suitable frameworks for this project. Although **FIWARE** presents the best results in the table, it requires a high computational power, making this framework, not suitable. Even though there is not much information about **Node-RED** in terms of accessibility, the ML modules developed under MIRAI project were conceived in this framework and since this one does not have an orchestrator, it makes it suitable for the project.

## 2.3 ML Workflows

Looking at the MIRAI project, if we look deep into the five use cases, ML techniques are used among all of them, in data forecast (power values output data forecast), cybersecurity (detection of anomalies), traffic management (object detection for collision prevention), faster water leakage detection (detection of anomalies) and dye machine operations (parameter estimation from historical data) [1]. Machine learning techniques play a pivotal role in Industry 4.0 due to their fast responses are adaptive environment to multiple sets of data.

### 2.3.1 Introduction to Machine Learning

ML is an artificial intelligence technique that trains machines, helping them learn from their experience, adapting to the situation, by using different algorithms for their training. It does not need human assistance nor complicated mathematical equations and it can work in dynamic networks.

ML techniques are grouped in supervised, unsupervised and reinforcement and they can be applied in multiple situations. As you can see in the figure 2.11 shows different machine learning algorithms [29].



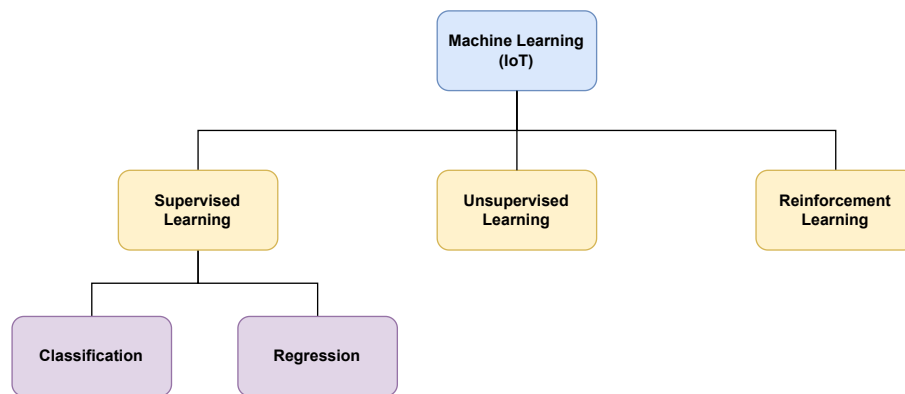


Figure 2.11: Machine learning methods adapted from [29]

**Supervised learning** the output is classified based on the input, by using a data set for training the algorithm. This method can perform as classification or regression. In Classification learning, the ML output identifies the class to which inputs belong to. Regression learning refers to where the output of the learning is a real or continuous value depending on the input variables.

In **Unsupervised learning**, there is no output for the given input variables. A big portion of the data is unlabeled, so the system tries to find out the similarities among this data set classing them into different groups as clusters.

**Reinforcement learning** allows the machine to learn from interactions with the environment by performing actions to maximize the feedback. There are no predefined actions in any particular task while the machine uses trial and error methods, where they obtain an answer with a positive or negative reinforcement.

### 2.3.2 Overview of Machine Learning flows

A typical ML workflow consists of six different steps, which are: data gathering, cleaning and pre-processing data, representing data, machine learning modules, learners and model evaluation [30].

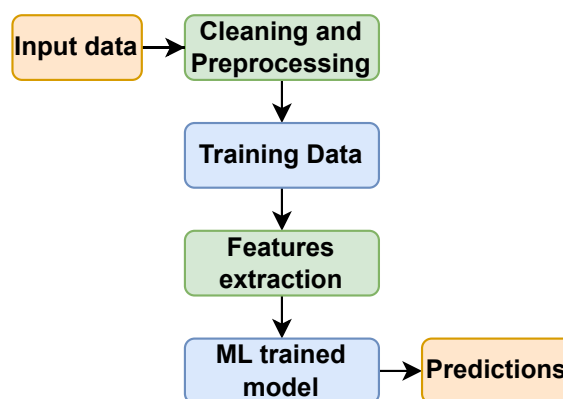


Figure 2.12: Standard ML workflow adapted from [31]

In data gathering there needs to be a dataset entry that will be trained. This gathering can be done with a direct input or can be fetched from databases (normally happens when working with a cloud). After obtaining the dataset, this vector of values needs for be cleaned and pre-processed. In this step there are four points (known as the "four V's") that are considered crucial in big data workflows and they are relevant in machine learning methodologies [31].

1. Volume is the amount of available data;
2. Variety is known as the diversification of data in form and meaning;
3. Veracity, is the understanding of the uncertainties linked with each data point;
4. Velocity, is how fast the data needs to be generated and treated (not a problem in workflows that don't require real-time features).

Data has to be clean and homogeneous before being used. A way for that to happen is to detect possible errors or inconsistent data points and removing from for the dataset training. Without this step it would be harder to build an accurate predictor via machine learning. Once the that is clean and homogeneous there is the need to encode the data into a set of specific variables that will be manipulated by the ML algorithm. Normally this data is in a raw format and needs to be converted into a suitable format for a learning procedure. The data is transformed and passes through a rescaling, normalization or binarization procedure in order to bring it to a state where the algorithm can easily parse the information. The effect of the pre-processing of data needs to be studied carefully because some algorithms can deploy better results without or with excessive pre-processing.

After representing the data, there is the need to apply the ML models mentioned in the section 2.3.1. The following step is the use of learners, this is a crucial step because each learner plays a key role in the accuracy of the prediction. In the evaluation model, there is a study to check if the predictions have a good performance. After the training of the dataset, there is usually given a subset for tests to check if the output values correspond to the expected ones.

In the figure 2.12 it is possible to observe a standard ML workflow that is used for the implementation of ML modules.

### 2.3.3 Applications in the MIRAI examples

In MIRAI there is the need to apply ML workflows in order to obtain better results for each use case [1]. In the **Distributed Renewable Energy Systems** use case, there is the need for ML techniques for forecasting values. In terms of solar energy, looking at the irradiation behaviour it is possible to extract the higher amounts of power/energy at peak times [32]. By predicting the best time to withdraw energy, it is possible to arrange a schedule, wasting less sources and extending the lifetime of the station. These modules can be applied in many other types of renewable energies, such as wind, biomass, wave, among others.

In the second use case, **Secure Internet Provisioning**, ML techniques can be used to identify malicious internet traffic, in order to prevent attacks from unknown sources. Looking at the bandwidth feature, there are minor differences between normal and malicious traffic. By using a ML model it is possible to detect these minor differences [33].

In the third use case, **Traffic Management**, with help ML techniques it is possible to predict the future movement of vulnerable road users based on their trajectory. By using video based motion classifiers it is possible to evaluate the physical state of motion when starting or stopping a car [34]. With this in mind, it will be possible to predict the area that will have higher traffic intensity.

In the fourth use case, **Water Management**, there is the need to use ML models to detect leakages as soon as possible. At the moment, the leakage detection is based on a statistical analysis and it is only possible to reach a high accuracy after three hours. With the use of a device with long lifetime running a ML trained model, it is possible to detect a water leakage directly without the need to send all data on a high temporal resolution [1], [35].

In the last use case, **Continuous Auto Configuration of Industrial Controllers at Edge**, the use of ML techniques will be needed to provide resource efficient deployment. These algorithms will be in permanently updating themselves and accounting the constraints, like insufficient steam sources [1].

## 2.4 Visualization Technologies

A GUI is an interface that displays the data of a system. It makes an application practical, easy and efficient to use [36]. This section has information about three possible frameworks that can be used as a GUI for this project. In 2.4.1 introduces Java Spring Boot, in 2.1.2 introduces SCADA (Supervisory Control and Data Acquisition) and in 2.4.2 introduces Grafana.

### 2.4.1 Java Spring Boot

Spring Framework is based in Java/J2EE (Java 2 Platform, Enterprise Edition) and it is an application development framework. It is one of the most popular Java frameworks with a 30% of share usage. This framework possesses features that enable an efficient development from simple web to complex enterprise applications. It has four main concepts which are, IoC (Inversion of Control), DI (Dependency Injection), AOP (Aspect Oriented Programming) and JPA (Java Persistence API) [37]. Spring framework can be seen in the figure 2.13.

**IoC** is a general concept where the control of the flow is made by external sources instead of being controlled by the programmer. As the name says the control is inverted.

The **DI** is a key concept in Spring and in Google Guice. This concept is a pattern in a form of IoC.

The improvement of modularity and the structure of code is the concept that **AOP** is based on.

The **JPA** is in control of the database. This concept maps the object state to the database columns and issues the queries across the objects.

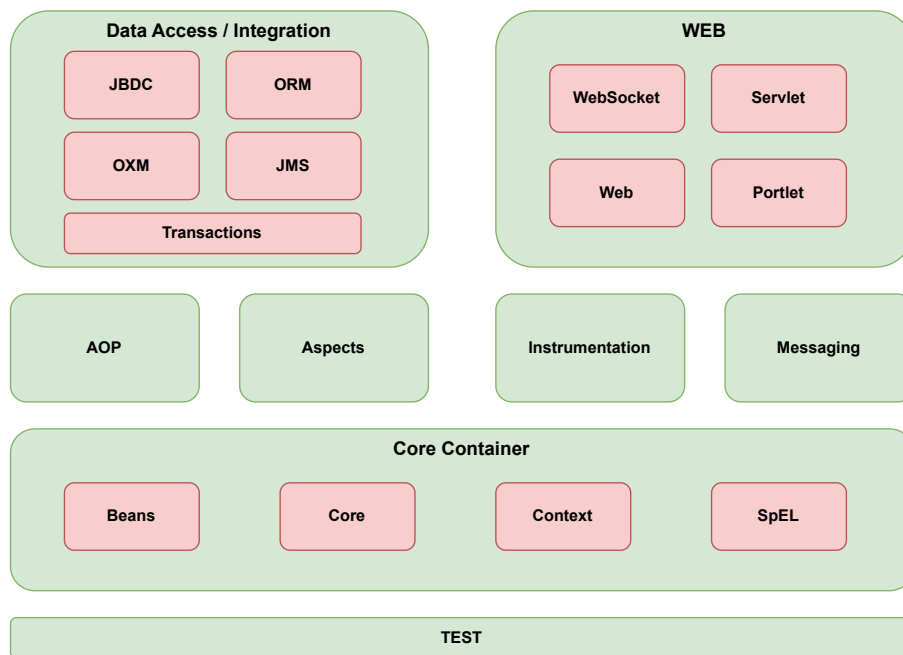


Figure 2.13: Java Spring architecture adapted from [37]

There are many important modules such as Data Access and Core Container, but if the need is the development of a web application the Web module becomes the most important one. This module consists of 4 modules, Web, WebSocket, Portlet and Servlet. In the Servlet it is contained the definition of the two most used concepts in the current days which are the MVC (Model-View-Controller) and the REST WS (Restfull WebServices).

The Spring Web flow and MVC are two powerful web frameworks that allow the development of multi-layered applications. The Spring MVC is built on DispatcherServlet and passes the requests to the @Controller that is based on a @RequestMapping annotations. The Spring Web Flow builds reusable web controller modules complementing Spring MVC by containing rich page navigation rules.

The REST WS is an architecture that provides interoperability between the computer systems on the internet. It is one of the most used techniques for data provisioning to multiple types of data consumers. The REST WS is supported by the Spring framework and its class annotation is @RestController. In order to gain access to the methods that provide functionalities it is required to use an unique @RequestMapping annotation. The ResponseEntity class can be used as a method and it results in an entire set of HTTP status code, body and headers. It possesses multiple constructors that carry the information sent as a HTTP Response.

The Spring Boot was designed to simplify the development of a Spring application. It has four main concepts. The Automatic configuration, the Dependencies starter, the Command-Line-Interpreter and the Actuator.

The **Automatic Configuration** configures the applications as standard Spring applications. The **Dependencies starter** automatically integrates the dependencies that are needed for the project.

The **Command-Line-Interpreter** allows the application to be controlled through the console. The **Actuator** provides information about the events that happen inside the application.

This framework provides tons of getting started examples for the development of applications and allows the user to select between two types of projects. Either Gradle or Maven.

### 2.4.2 Grafana

Grafana is a tool that allows the user to visualize multiple series of data chronologically. It possesses a graphical overview of the collected and generated data and highlights the most important data, known as KPIs (Key Performance Indicators). Some of the KPIs that are included in Grafana are latency, user data rates of up and downlinks communications, among others [38].

This software provides some advantages which are:

- Flexible quick graphs with multiple options;
- Dynamic and reusable dashboards;
- Greatly extensible, enabling the use of many dashboards and plugins that are available in the official library;
- Enables collaboration by allowing the data and dashboards to be exchanged between teams.

These advantages provide the user with a very useful application that allow efficient management of data and services, providing a global knowledge of the situation.

Additionally, Grafana possesses multiple dashboard representations, some can be seen in the figure 2.14

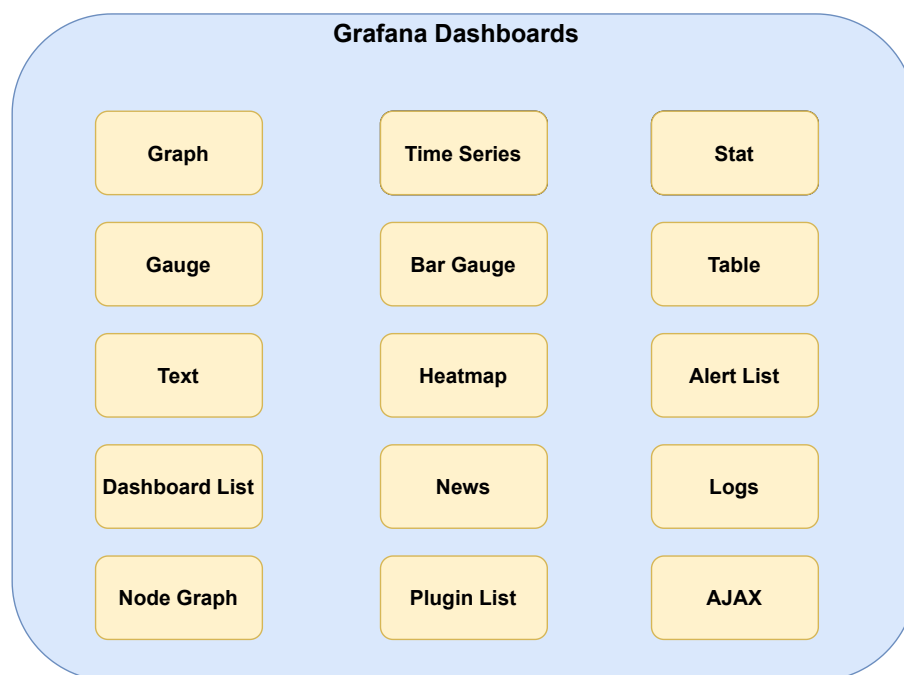


Figure 2.14: Grafana charts adapted from [38]

## 2.5 State-of-the-Art: Application of Integration Frameworks

During the course of the years, there has been a great number of applications in the industry that require the use of IoT frameworks and ML solutions. In this section, we will focus on the use of the Arrowhead and Node-RED frameworks since these are the two that show most potential for utilization in the MIRAI project, as explained in the section 2.2.7. Arrowhead has greatly expanded in this area and can be applied in multiple fields to facilitate the production of components or to just monitor tasks [3]. Node-RED has become widespread and is being used in the industry area, this is because this programming tool is flow-based and is great for automation systems management [22].

### 2.5.1 Industrial Predictive Maintenance Application

The article [39] mentions that EUREKA-ITEA has developed a project called "A Smart Predictive Maintenance Approach based on Cyber Physical Systems" with the objective of acquiring manufacturing data to provide diagnosis and prognostic information making the feasible technology financially viable. The project improves overall equipment effectiveness of the manufacturing industry by making use of low cost sensors in order to produce a cost efficient Predictive Maintenance (PDM) model. A primary technique used in PDM is data science (analysis and modelling of measurement data) and when this is combined with the knowledge domain, it is possible to acquire automatic and reliable diagnostics, mechanisms of failure detection and supported decision making. In order to reach this goal, it is required to deploy an IoT framework, that's where Arrowhead comes to play.

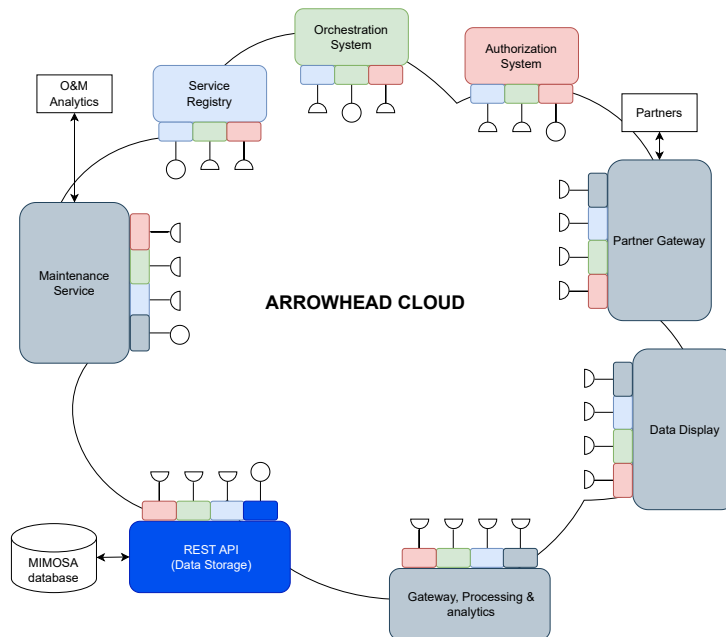


Figure 2.15: Arrowhead cloud architecture in the article [39]

This project makes use of two IoT frameworks, MIMOSA, to deploy data models in the edge devices and in the cloud, and Arrowhead for delivery of services (architecture can be seen in 2.15). Since Arrowhead enables interoperability between different devices while providing features like QoS and security, it is a good framework for this case. The nodes of the edge devices are used to capture and provide data as a producer, while providing data acquisition services that measure vibration acceleration from the bearing. After this data is obtained, they are stored in a local MIMOSA database that has previously been filled with metadata. The data consumer is a separate node in the local cloud, that receives data with JSON Representational State Transfer (REST) API. After the data analysis the information is available to be monitored in a GUI.

Also the use of ML techniques were used in this project to help predicting results of maintenance by estimating the possible future failures of the system.

### 2.5.2 Integrating an Electric Vehicle Supply Equipment

The biggest challenge of electric vehicle initiatives is the overcoming of electrical energy storage barriers. Batteries energy density is not still high enough to compete with fossil fuels, that is why charging and supplementary maintenance issues in the electric mobility. Electric Vehicles Supply Equipment (EVSE) have physical access to the electric vehicles and can also communicate with service providers of ICT systems. EVSEs are organized into networks and accessed remotely. They need to operate in a service oriented way due to the stakeholders request. One of the biggest tasks is to create a dynamic and trusted infrastructure of charging stations and managing centers that allows information exchange ICTs [40]. EVSE are important endpoints of an electrical grid, that produces and stores energy locally, because their energy consumption is controllable.

Arrowhead is a framework that is suited for this problems mentioned above because it can be a networking platform of electric mobility. The local cloud concept and the service oriented architecture allows safe and dynamic networking for the electric mobility domain. In this case Arrowhead is used to control a charging station.

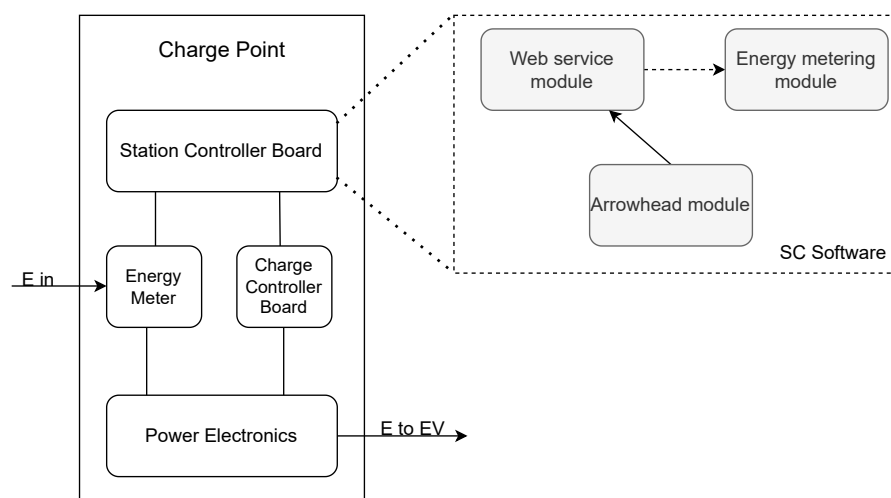


Figure 2.16: EVSE's station controller in article [40]

A charging station consists of power electronics, an energy meter, a charge controller board and a station controller board. The charger's information related to the charging transactions is sent to the Arrowhead cloud in form of a web service. This framework is in charge of controlling the energy in the charging station, by controlling the data about power consumption. Arrowhead in this case is used to provide services that the Open Charge Point Protocol does not possess.

### 2.5.3 Smart Cities

Smart cities are composed by a multiple number of different companies' systems and managed by different operators. A great challenge is the efficient integration of all these systems into one but with new technologies a this collaboration can be reached. This collaboration can be enabled by a SOA due to its interoperability services. Since Arrowhead possesses a SOA it is an IoT framework that suits the job.

In this article [41] in order to deploy a smart city they separate the smart city concept into three systems: Street light, car heating and system integration.

For the street light system an Arrowhead interface was developed for light controllers, allowing the integration of the street light system into the Arrowhead network. The light control system acts as a service producer. Each street light can be monitored and controlled via a RESTful interface. The light controller publishes its endpoints in the Arrowhead Service Registry in order to be discovered and consumed by other devices in the same network. Uses the authorisation system to verify if the consumer is allowed to use the services. The orchestration system is not used because the light controller does not need to consume any application producer services.

The car heating system only uses service registry and authorisation system as well. The authorisation system is used to verify if the consumer is allowed to use engine block heaters. The service registry system is used to publish the endpoint of the heater controller in order to be discovered by other systems in the Arrowhead network. The Arrowhead service interface can be used for requesting energy consumption data and providing external information like for example the temperature outside.

The integration system is what we call a system of systems. This will integrate the car heating and street light systems into a network of systems with additional sensors and control. These sensors can monitor, the outdoor temperature and the brightness level, and will be used to integrate both systems. The integration control can define the brightness level on each street light which means the energy consumption is controlled. In terms of the car heating system, the control of the outdoor temperature value is used to optimize the engine block heating time to save up energy.

### 2.5.4 Node-RED in Industrial Environment

There is the need of storing a huge amount of data collected by smart devices. For industrial systems, the use of relational SQL data storage has been the main solution until now. The problem with big data means that it is not possible to store such amount of data in small database systems, unless there is a major investment in the infrastructure. In order to avoid this problem, the use



of a Cloud to store data is a good possibility, which is a good cost efficient method that does not need a big hardware investment. There's a lot of cloud hosting services on the market designed to rent as much space required. Another important feature that cloud systems possess is their scalability. Since companies in the industrial environment are disinclined to see their data outside of the factory, it affects the spread of this technology in the industry. Even this being said, the spread of IIoT helps a lot in the growth of cloud services, due to the fact that these devices can easily be connected to cloud systems which already provide data analysis and visual platforms in order to control the stored data.

The developed application can monitor a general factory machine and environment using Node-RED. It uses multiple sensors to store the machine's operating data, such as speed, status, hours of work, and the changes in the environment, such as temperature, humidity, among others. The system's architecture can be seen in the figure 2.17.

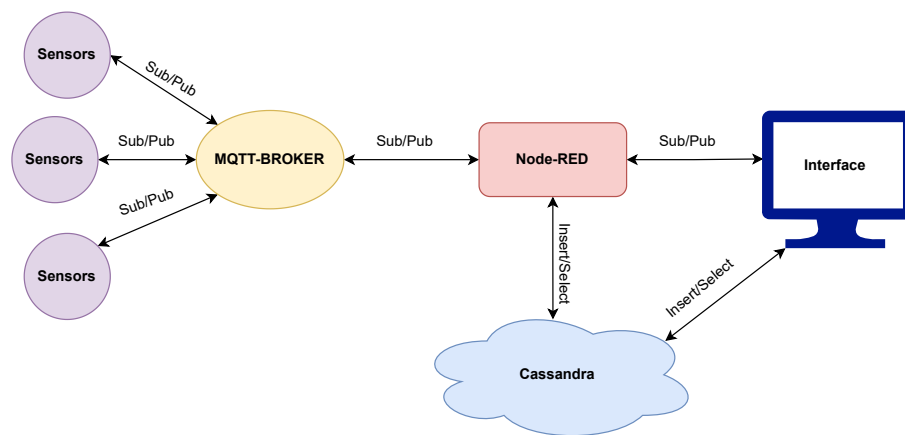


Figure 2.17: System's architecture from [22]

The interface displays the data in real-time via a MQTT (Message Queuing Telemetry Transport) broker and stored in a Cassandra database. With the use of the Node-RED dashboard, a simple interface was created to provide a visual aid to the user.

To have a good amount of IoT sensor data, it was used a python code to generate industrial machine and environment values to simulate the use of sensors. With the use of the MQTT broker it was possible to manage the data, generating a code that not only publishes but also receives the real-time data.

In conclusion, the Node-RED framework can be used to develop an IIoT applications prototype as well as fetching and displaying data via the MQTT broker.



## Chapter 3

# System Implementation

This chapter contextualizes the developed project. The section 3.1 provides a general overview of the developed tool (its goal, a list of its functionalities and the user interface). The section 3.2 presents the project's architecture as well as its components, how they are linked between them and a detailed explanation of the system's functionalities. The section 3.3 presents all the considerations needed when working with Node-RED as well as an insight of how the framework operates.

### 3.1 Large-Scale Edge Management Tool (LEM tool)

The development of the tool is required to meet all expectations that MIRAI has, in order to address all the use cases mentioned in the section 2.1.3. For this, the tool needs to monitor, deploy and control multiple devices that will be running ML algorithms previously developed by MIRAI. The system needs to be able to communicate with a large number of devices at a quite fast pace, with the minimum delay possible.

#### 3.1.1 Goal

The goal of this tool is to control a large number of devices, independently of which framework is currently running, all devices. For this, the system needs to be adaptable when receiving data in order to accept any type of framework that will be running in the devices. Theoretically speaking this is a good aim, but this dissertation will focus on only one framework. This is due to the fact that MIRAI only developed pipelines in Node-RED.

#### 3.1.2 List of Functionalities

The developed tool possesses multiple functionalities responsible for the good behaviour of the *LEM tool*. These are the following:

- Monitoring;
- Deploy Devices;
- Deploy Flows;
- Device Control (Start);
- Device Control (Stop);
- Delete Devices;
- Delete Flows;
- Edit Devices;
- Edit Flows;
- Information Update;

The explanation of all of these functionalities can be seen in detail in the section [3.2](#).

### 3.1.3 User Interface

The **User Interface** displays all the information regarding the *LEM tool*. This means if the user wants deploy, control or just check all the information from any device, he can do it in here. It also provides the options delete and edit flows or devices.

This interface is coded in Java Spring Boot. Java Spring Boot also has default functions that communicate directly with the database, allowing information exchange from the interface to the database. This choice was done taking in consideration that the Core system and the HTTP connections were being coded in Java. This component provides a web based interface as you can see in the figure [3.1](#).

ID	Name	IP
2	exampleDevice	127.0.0.1
3	VM2	192.168.107.137

Figure 3.1: Graphical User Interface Web based

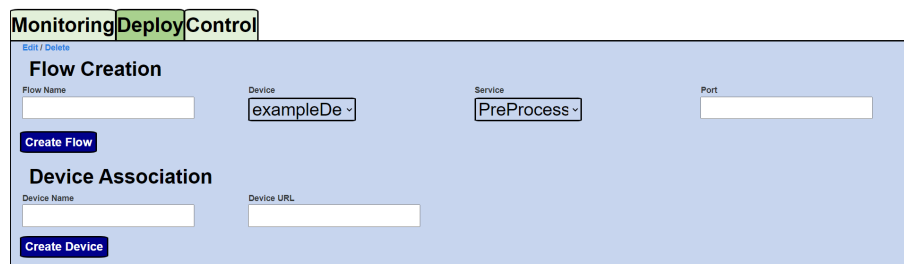
As it can be seen in the figure above 3.1, the interface has three tabs. Monitoring, Deploy and Control.

The **Monitoring** tab, allows the user to select between three tables what type of information he wants to see. The tables that are shown here are the ones previously explained in the section 3.2.1.4.

The **Deploy** tab provides two options for the user. Either create a new flow on an edge device or associate a device to the system.

The flow creation allows the user to choose any name for the flow, select in which device it wants to be deployed, which type of service wants the device to run and in which port will this flow run.

The device association just needs the name that the user wants to give to the device and the associated URL (Uniform Resource Locators). The deploy tab can be seen in the figure 3.2.



The screenshot shows the 'Deploy' tab selected in the top navigation bar. Below the tabs, there are two main sections: 'Flow Creation' and 'Device Association'. The 'Flow Creation' section has a 'Flow Name' text input, a 'Device' dropdown menu with 'exampleDe' selected, a 'Service' dropdown menu with 'PreProcess' selected, and a 'Port' text input. There is a 'Create Flow' button below these fields. The 'Device Association' section has a 'Device Name' text input and a 'Device URL' text input, with a 'Create Device' button below them.

Figure 3.2: Deploy tab in GUI

As it can be seen, this tab also provides two other options, the delete and edit. In the delete option, it is possible to either delete a flow from a device or simply delete the device association from the system. In the edit option it is possible to replace the names of any created flow as well as their service and edit the name of a device. In the figure 3.3 it can be seen the delete option's page and in the figure 3.4 edit's page.



The screenshot shows the 'Delete Options' section within the 'Deploy' tab. It contains two sub-sections: 'Flow Delete' and 'Device Delete'. The 'Flow Delete' section has a 'Flow Name' dropdown menu with 'Flow' selected and a 'Submit' button. The 'Device Delete' section has a 'Device Name' dropdown menu with 'exampleDe' selected and a 'Submit' button.

Figure 3.3: Deploy tab in GUI - Delete Option

The screenshot shows the 'Deploy' tab selected in the top navigation bar. Below it, the 'Edit Options' section contains two forms. The 'Flow Edit' form has a 'New Flow Name' text input, a 'Flow Name' dropdown menu with 'Flow' selected, and a 'Service' dropdown menu with 'PreProcess' selected. Below these is a 'Submit' button. The 'Device Edit' form has a 'New Device Name' text input, a 'Device Name' dropdown menu with 'exampleDe' selected, and a 'Submit' button below it.

Figure 3.4: Deploy tab in GUI - Edit Option

In the **Control** tab, it is possible to start or stop any instance of Node-RED at will on any associated device. All it needs to be done is select the device, the port where the instance wants to be run or is currently running and what control option it's wanted. The figure 3.5 show the control page in the GUI.

The screenshot shows the 'Control' tab selected in the top navigation bar. Below it, the 'Control Options' section contains a form with a 'Device Name' dropdown menu with 'exampleDe' selected, a 'Port' text input, and a 'Control' dropdown menu with 'Start' selected. Below these is a 'Submit' button.

Figure 3.5: Control tab in GUI

### 3.1.4 Integration with other Frameworks

This tool was developed in a general overview to be ran no matter what framework is running in the devices. In order to use any framework at choice, there is somethings that need be taken into consideration.

The *LEM tool* currently communicates with the devices using a HTTP protocol to access certain endpoints in order to fetch required information, such as the timestamp of when the service was last executed as well as the status of the device.

The developer needs to meet these requirements. If an endpoint is given to the tool, the system can run multiple frameworks. Looking into the section 3.2, it is in the `Controller` class where the connections to these endpoints are made. The `Core` class also needs access to the endpoints in order to update the information on the system.

## 3.2 Architecture & System components

In order to develop such a tool for large scale control, the system was be divided into four major parts. The `Core`, the `Database`, the `Controller` and the `GUI`. This system requires to be scalable, this means that the tool can either control one or multiple devices at will.

The **Core** is in charge of updating all the tables from the database, as well as running the Java Spring Boot Application, where the `Controller` and the `GUI` components are allocated.

The **Database** is there to store all the data. The database has multiple tables, such as *flows*, *devices* and *nodes*. In these tables, all the information is saved in order to control the system.

The **graphical user interface** is where the user can monitor all the devices visually, as well as deploy multiple devices or flows and control the Node-RED service in each device. It also has the ability to edit certain information of the flows or devices and it allows the user to delete created flows or associated devices.

The **Controller** is in charge of controlling the full system. It's the only component that is linked with all the others. This one fetches and updates data in the database, it updates the information in the GUI and it is run in the Core component. It is also in charge of the communications with the edge devices. It forces the deploy of a service and it can also delete the service from the device, if the user decides to.

In the **edge devices** is where the *services* are running. The system is prepared to work with multiple frameworks as long as the communication parameters between the MES and the edge devices are the same.

In the figure 3.6 it is possible to see the architecture that was developed for the creation of the tool.

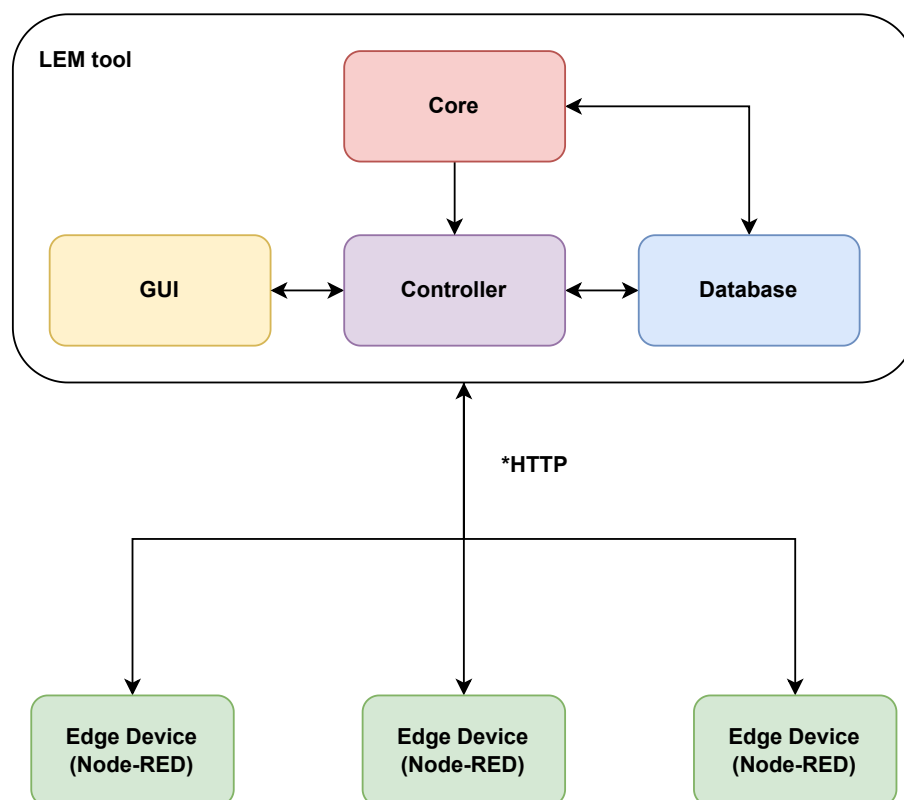


Figure 3.6: System's Architecture

Even though the Core is the main component where the Java Spring Boot is ran, in order for this to communicate with GUI, it needs the help of the Controller. The Controller can communicate

with the GUI, receiving the submits from it and accessing the database to display the information in the interface.

### 3.2.1 Classes Description

The system is composed by fifteen classes. There are four entities, four repositories, one list of services for the interface composed by all the repositories, one class to communicate with the edge devices, two classes to control the Node-RED services, one class to control the system, one class to generate IDs for the nodes in a new Node-RED flow and one class that runs the project.

The four Entity classes has a direct link with the database and these are:

- Flows Entity;
- Devices Entity;
- Nodes Entity;
- Services Entity;

The **Entities** are classes that contain parameters ready to accept the information of each field from the database tables. The way these fetch information is via their repository that possesses multiple functions to fetch or input information in each database table.

The **GUIServices** class is in charge of connecting and updating the database. The functions inside this class are used by the **Controller** and **Core** classes. It has multiple functions that fetches data.

The **Http Connection** class makes enables the connection between the MES and the edge devices that run Node-RED. This class can POST, PUT and DELETE flows in a target device and it is linked with the **Json String** class in order to generate new flows with different node IDs.

The classes **commands** and **SSH Connection** are in charge of starting and stopping the Node-RED in the edge devices.

The **Controller** class is composed by multiple classes that allow the control of the system. This class is in charge of updating all the information in the GUI as well as creating and modelling the browser pages from the interface.

The **Core** class is where the system in ran. This class starts the GUI as well as updates the database every ten seconds. This class creates multiple threads depending on how many devices are connected to the system. As previously mentioned in the section 3.2.1.2, the number of threads is proportional to the number of associated devices in the tool.

The UML class diagram can be seen in the figure 3.7.



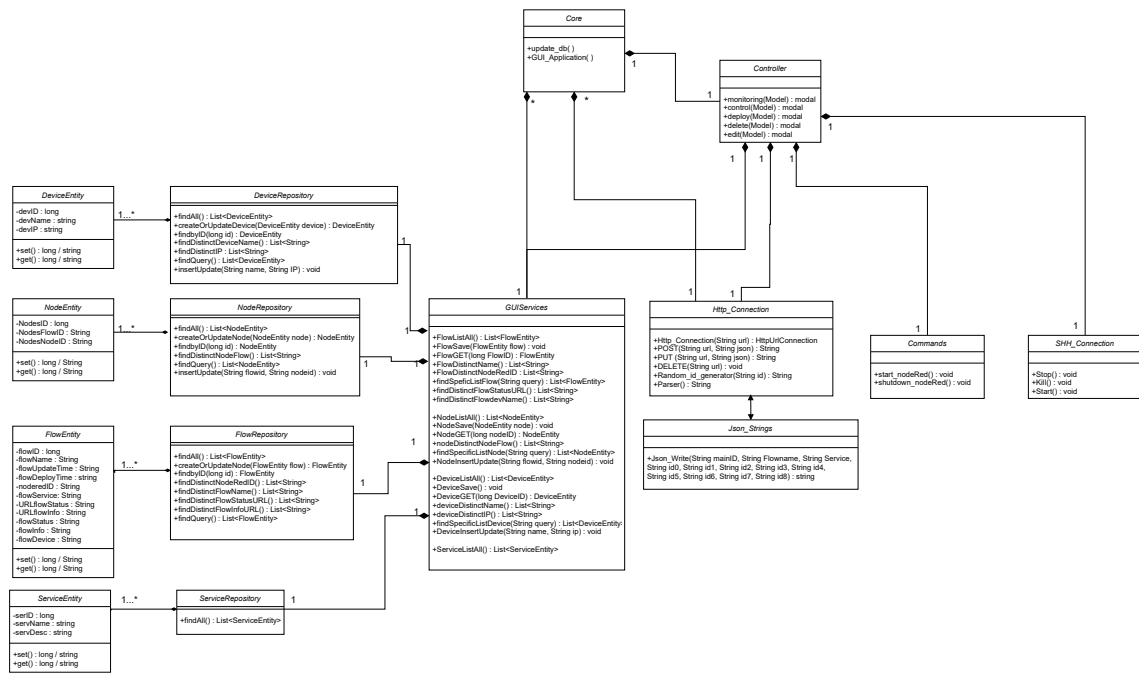


Figure 3.7: UML class diagram

### 3.2.1.1 GUI

The GUI is the component that is in charge of displaying all the information to the user. This component is in charge of delivering a logical view in order to monitor, control and deploy all the information more easily. It is the link that connects a user to the tool in a graphical way.

It allows information submits that can associate devices or create flows. It directly interacts with the `Controller` component.

### 3.2.1.2 Core

The `Core` is where the `GUI` and `Controller` run. This component starts the interface and updates all information of the database. Every ten seconds the system checks how many devices are associated, creates a number of threads proportional to the number of devices and updates the flow field with the live information.

The threads are created to speed up the devices' updates so it doesn't overwhelm the tool when it's controlling multiple flows at once. In every thread it's made a connection to check the status of each `flow` in each associated `device`. If this connection does not return a status code of 200, it is considered that the `flow` is not running. The way the components work between each other is described in the 3.2.4 section.

### 3.2.1.3 Controller

The `Controller` serves as a middle man between `GUI` and the `DB`. This component defines all type of information that the `GUI` displays. It accepts the submit inputs from the interface and based

on the input it decides what type of information it will display, as well as creating new instances in the DB when a flow or device is deployed.

This component is based on an event trigger approach. Every time a submit is made from the GUI's side, it starts a step of procedures depending on what was asked to do. The communications made by this component are explained in detail on multiple sections below that explain the functionalities listed in 3.1.2.

### 3.2.1.4 Database

The **DB** stores all the information required to run the *LEM tool*. This information is fetch by the class *GUIservices* and it is used by the *Controller* class. The *Controller* class takes this fetched data and sends it to the GUI component, enabling the access to the information for the user.

## 3.2.2 Information Model

All the system's information is stored in the database, in order to keep track of all *flows*' status, associated *devices* and *nodes* that exist in each flow.

This project's database has three tables that are required to fully monitor the system. These are the *devices*, *flows*, *nodes* and one auxiliary table, the *services* tables.

In the figure 3.8 it is possible to see the field description of each table.

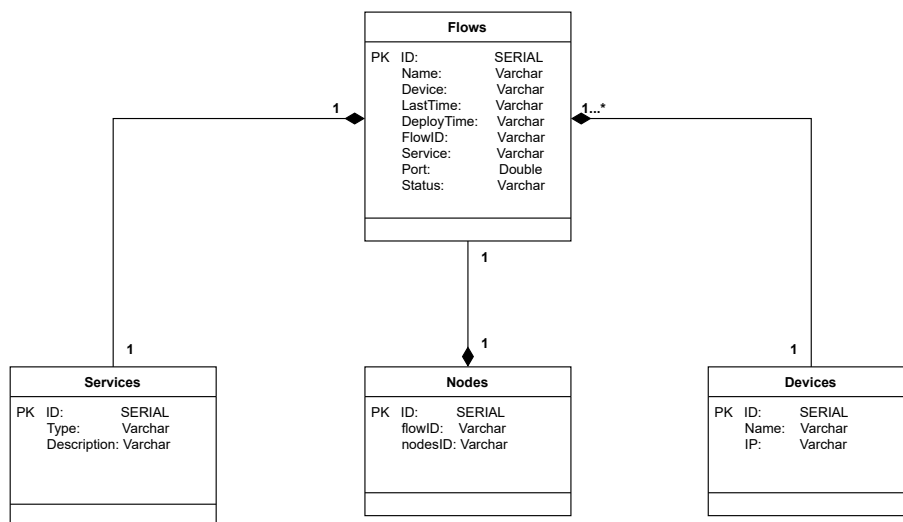


Figure 3.8: Database UML Diagram

The table of *devices* is composed by the *ID*, the *Name* and the *IP*. An example of the information inside this table can be seen in the table 3.1.

Table 3.1: Example of data in Devices Table

ID	Name	IP
1	Device A	162.154.107.151
2	Device D	151.103.123.145
3	Device B	174.192.143.128

In the table of `nodes`, the *ID* that has the same specifications as the *ID* field in the `devices` table, the *ID* of a flow and the *ID* of all the nodes inside the flow. Looking into the table 3.2 it is possible to see an example of a `Nodes` table.

Table 3.2: Example of data in Nodes Table

ID	FlowID	NodeID
1	51638b50621f7dd3	51638b50621f7dd3.PX1CI
2	51638b50621f7dd3	51638b50621f7dd3.jTtEO
3	a9e8cccf72b284bc	a9e8cccf72b284bc.JXhk1

The table of `services` has the *ID* field that is exactly the same format as the others, the *type* that says which service it is and the *description* that describes the service. Looking into the table 3.3 there is a detailed version of the service table. This table should be editable, with the options of adding or deleting services at will.

Table 3.3: Table of Services

ID	Type	Description
1	Preprocessing	Filters the data, in order to select the correct one for training
2	Training	Training is used to apply concepts such as denoising and segmentation
3	Evaluation	Evaluation is used to see if the trained model is optimal or needs more training
4	Inference	Inference is the trained model that will receive the input data and produce an output

The table of `flows` has the *ID*, the flow *Name*, the *device name* where the flow is running, the *deploy time*, the last initialized service time (*Last Update Time*), the *ID* of the *NodeRED* which is the same as the *flowID* of the table 3.2, the *port*, the *status* and the *service*. It is possible to see an example of the table in 3.4.

Table 3.4: Example of data in Flows table

ID	Name	Dev	DeployTime	LastRunTime	FlowID	Service	Port	Status
1	Flow1	VM1	2023/01/12 17:50:03	2023/01/16 18:32:03	51638b	Infer	1880	Dead
2	Flow2	VM2	2023/01/13 18:15:22	2023/01/16 18:32:34	40d721	Train	1881	Alive

The three tables that are required to fully monitor were created in order to create a decent structure necessary for the good behaviour of the *LEM tool*. When a **flow** is created, it is necessary to save its information in the `flows` and `nodes` table.

The flow name, device name, deploy time, flow ID, service and port are instantly saved in the `flows` table. Once the service in the flow starts, it is necessary to save the execution time. The status is constantly updated.

The **nodes** information is instantly saved in the `nodes` table upon creation. It is essential to keep track of all the nodes ID, in order to check if the flow is properly deployed.

The *devices* information is saved in the `devices` table once a *Name* and *IP* attributed to a machine.

In the figure 3.9 it is possible to see how the fields of each table are intertwined.

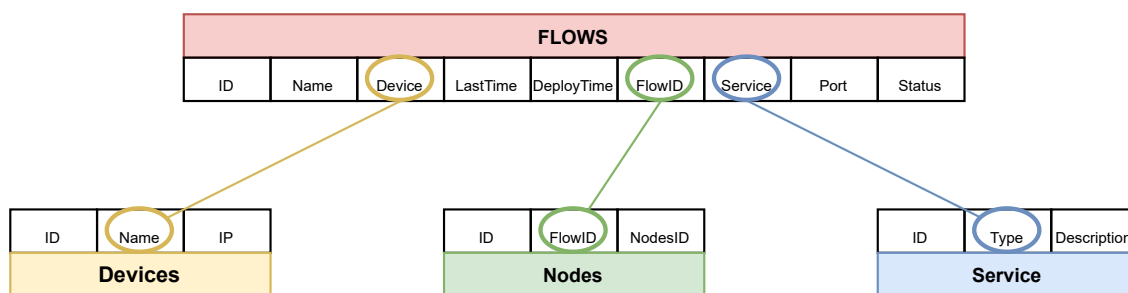


Figure 3.9: Database Table Relations

These tables were created to track everything from the moment a device is added to the *LEM tool* until the service is deployed and running in the edge device.

### 3.2.3 Monitoring Functionality

The way the components interact with each other differs depending on which GUI's tab is being used. In the **Monitoring** functionality 3.1, there are two communication sequence that include all components in the system. One of them is updating the system every ten seconds and the other is fetching and displaying the data in the web browser.

Taking a close look into the figure 3.10, it is possible to perceive how the data is displayed. When the Monitoring tab is open, the `Controller` fetches all the information from the `Database` and displays it in the interface. Once a submit button is pressed, the `Controller` fetches the data chosen by the user and prints it also the in graphical interface.

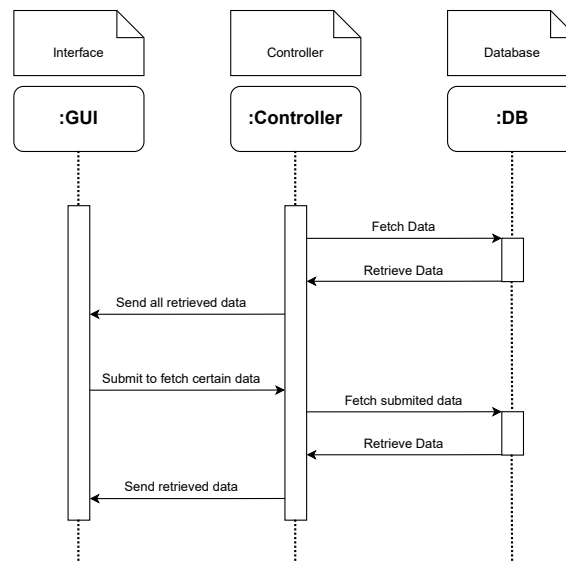


Figure 3.10: UML Monitoring Sequence

### 3.2.4 Update Functionality

The updating sequence can be seen in the figure 3.11. In here there is a loop of every ten seconds. The `Core` component fetches the number of existing devices in the system and then creates the same number of threads. Each thread is responsible to update each *device*. These threads check the status of Node-RED in the device and fetch the last time of when the service was run. Once this information is attained, they update the `Database`.

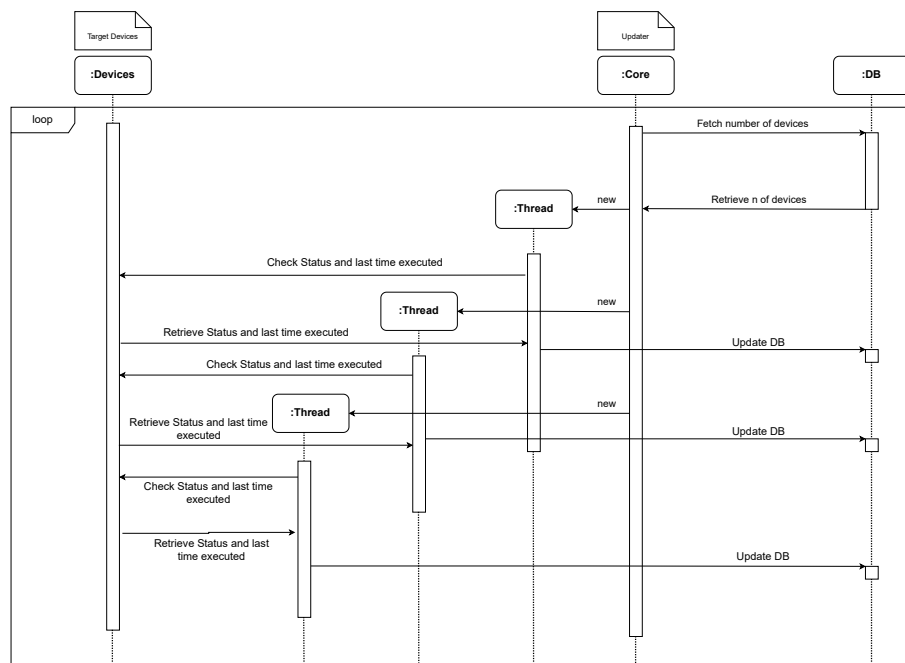


Figure 3.11: UML Update Sequence

### 3.2.5 Deploy Functionality

In the **Deploy** functionality happens two sequence depending on what the user wants to deploy. If it is a device deployment, the Controller, Database and GUI start a sequence, if it is a flow deployment, the edge device is added to the equation.

#### 3.2.5.1 Deploy Device Functionality

In the device deployment, once the user submits a name and URL in the GUI, the information is sent to the Controller. This component communicates with the Database to check if a device with the same name or same URL exists. If it doesn't, it saves the information and sends a deployment confirmation to the GUI, if it does, it sends a message saying that there is also a device with the same name or same URL. This sequence can be seen in the figure 3.12.

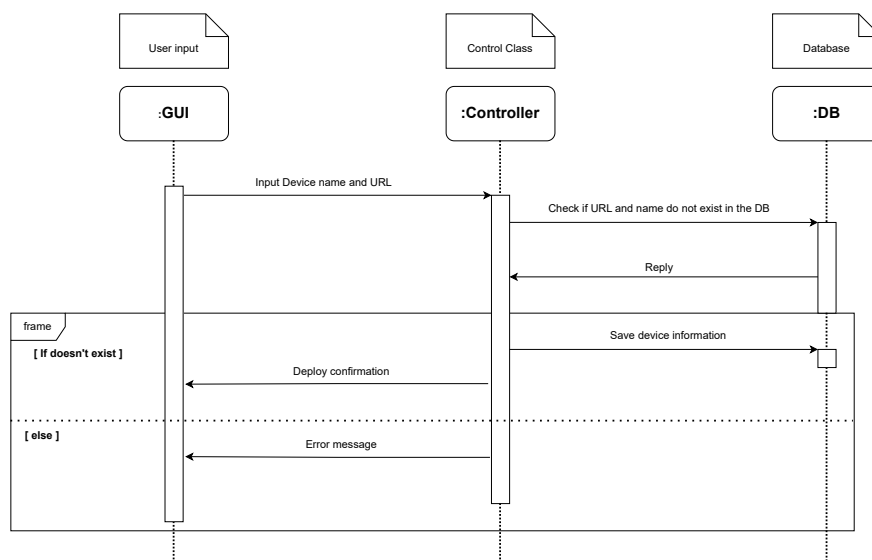


Figure 3.12: UML Deploy Device Sequence

#### 3.2.5.2 Deploy Flow Functionality

In the flow deployment, the user submits the desired flow name, the service, the device where he wants to deploy and the port of the device. Once this information is submitted, the controller checks if this information already exists in the Database.

If it exists sends a message saying to swap the submitted data, if it doesn't this component checks the connection with the device.

If the connection is OK, **POSTS** a flow in the device and the Node-RED in the device replies with the ID of the created flow. Then the Controller generates random IDs for each node in the flow and sends a **HTTP PUT** into the device with the desired service flow. Once the device replies with everything OK, the Controller saves the information in the Database and sends a message to the GUI confirming the flow was correctly deployed.

If the connection with the device fails, the `Controller` sends a message saying the Node-RED in the device is down. The sequence can be seen in the figure 3.13.

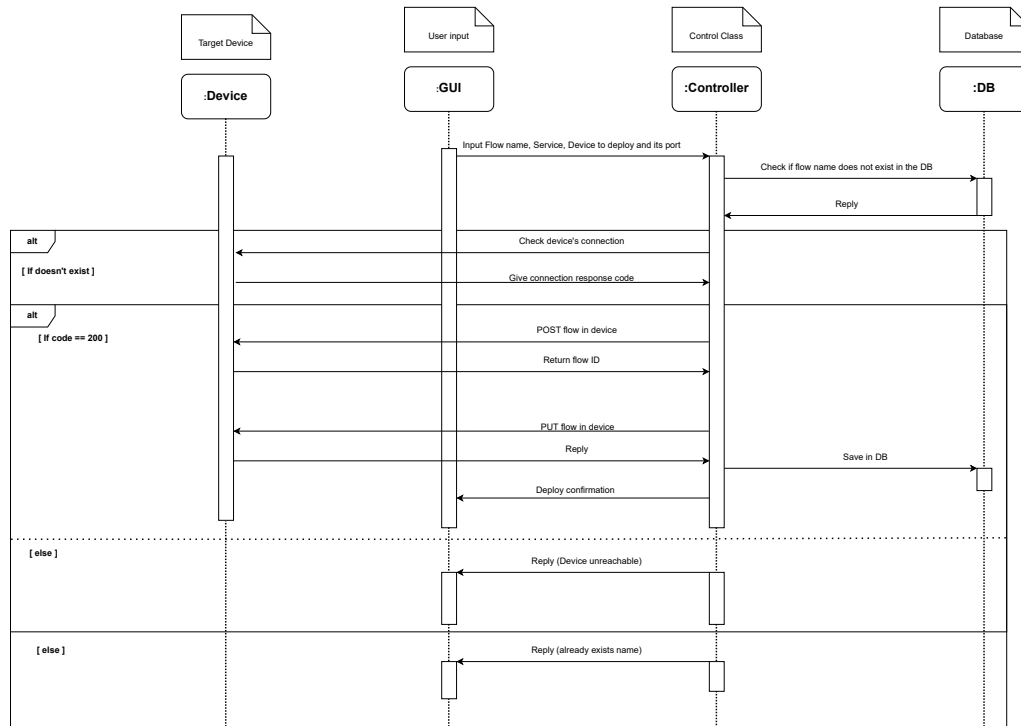


Figure 3.13: UML Deploy Flow Sequence

### 3.2.6 Edit Functionality

In the edit functionality it is possible to either edit the flow or the device. The edit device has the `Controller`, the `GUI` and the `Database` component communicating in a sequence. The edit flow adds a new component in the sequence that is the `edge device`.

#### 3.2.6.1 Edit Device Functionality

In the **edit device** the user select which *device* wants to edit, selects a new name and submits this information. The `GUI` sends this data to the `Controller`, then the data associated to the *device* is fetch from the `Database` and then updates the information with a new *device* name and send the confirmation to the `GUI`. This sequence can be seen in the figure 3.14.

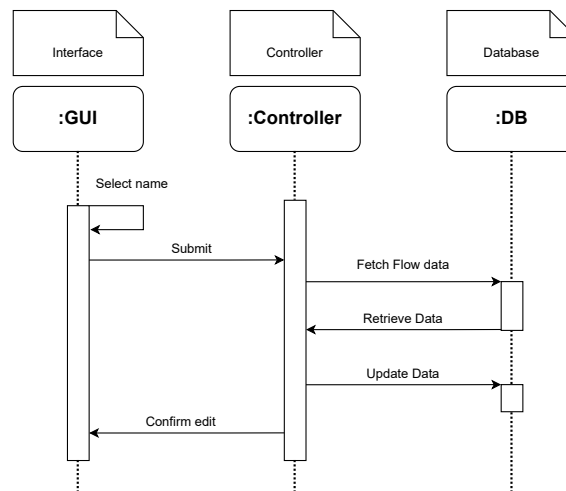


Figure 3.14: UML Edit Device Sequence

### 3.2.6.2 Edit Flow Functionality

The **edit flow** proceeds the same way at start. The new data is submitted from the **GUI** to the **Controller**, the **Controller** fetches the information from that *flow*, checks the connection with the device where this *flow* is deployed and if the connection is down sends a message to the **GUI** warning the user that the service is down. If the connection is up, it edits the *flow* data in the device, updates the information in the **Database** and sends the confirmation to the **GUI**. The sequence can be seen in the figure 3.15.

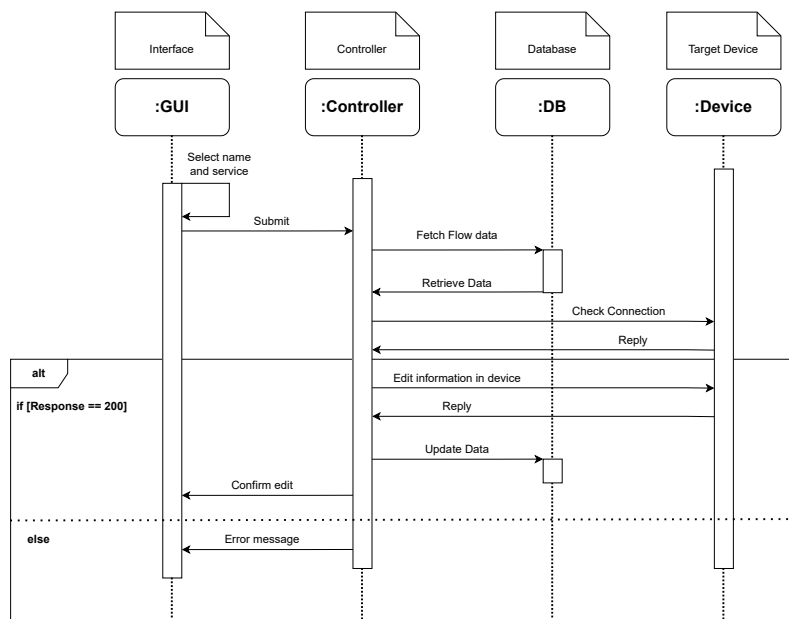


Figure 3.15: UML Edit Flow Sequence



### 3.2.7 Delete Functionality

Like the edit, the delete option can either dissociate a device or delete a flow. Both choices use four components in their communicating sequence. The GUI, Controller, edge device and Database. Both sequences are similar.

#### 3.2.7.1 Delete Flow Functionality

In the flow one, it is selected a flow and once the information is submitted, the Controller fetches the information associated to the chosen flow. Then it checks if the Node-RED in the device is working fine. If it is, it sends a **HTTP Delete** into the device and receives a reply confirming the delete and deletes the entry in the Database. The confirmation is redirected to the GUI, so the user knows the flow was deleted. If the status of the device is not OK, the GUI prints an error message warning the user. Check figure 3.16 in order to see the flow delete sequence.

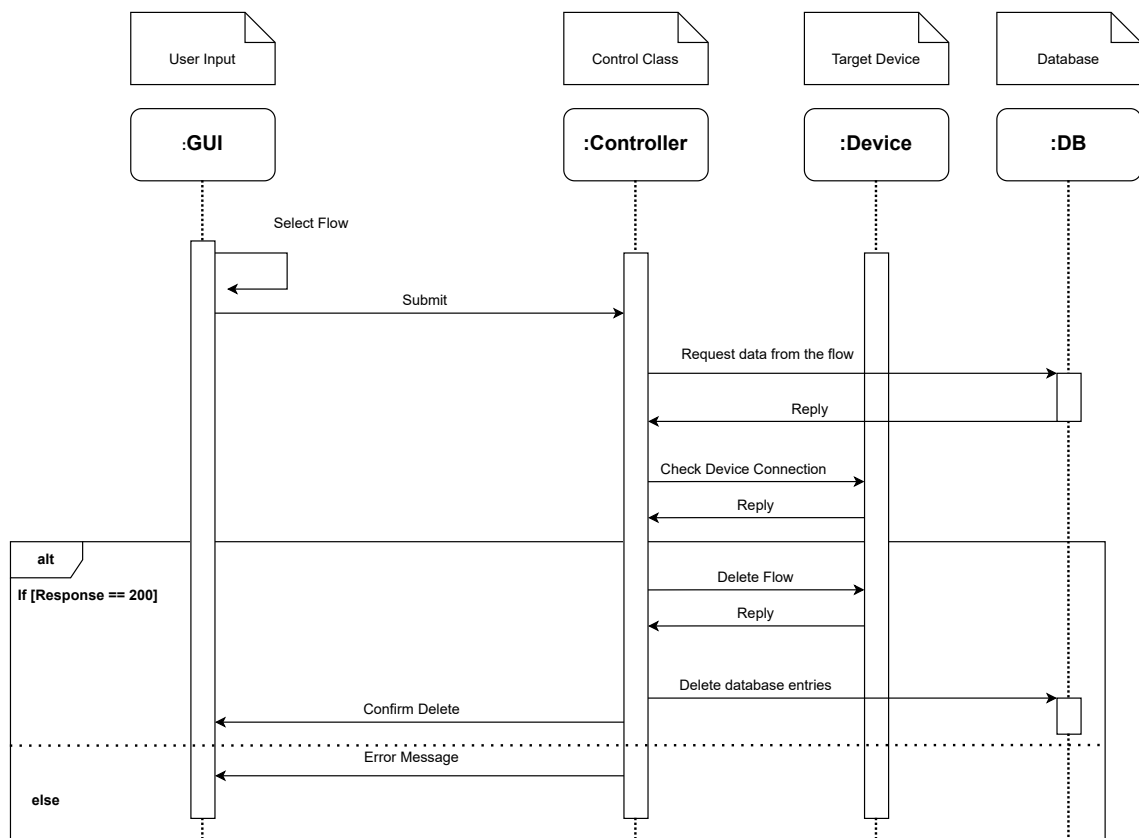


Figure 3.16: UML Delete Flow Sequence

### 3.2.7.2 Delete Device Functionality

In the device delete, the sequence is almost the same, the only difference is that the device is deleted from the `Database` after deleting all existing flows inside it. It is possible to see the sequence in the figure 3.17.

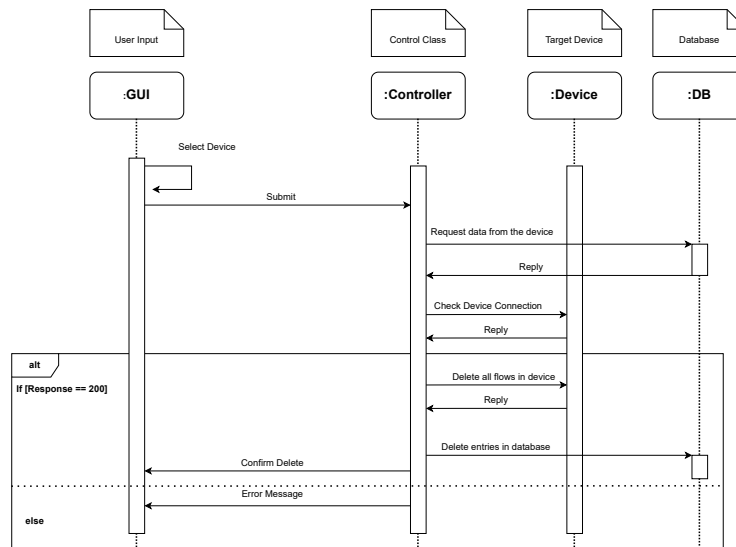


Figure 3.17: UML Delete Device Sequence

### 3.2.8 Control Functionality

In the **Control** functionality, there is only one sequence. After the user selects the *device*, *port* and control option (**Start** or **Stop**), submits it. The `GUI` communicates with the `Controller` and this component has three options. If the service is already running and the user decides to start it, it is print an error message in the `GUI` warning the user. The same goes if the service is down.

If the service is running and the user decides to stop it, the `Controller` sends a **SSH** (Secure Shell) command to the device, killing the Node-RED process *ID* of that device. If the service is stopped and the user decides to start it, it sends a **SSH** command starting the Node-RED. In figure 3.18 it is possible to see this sequence.

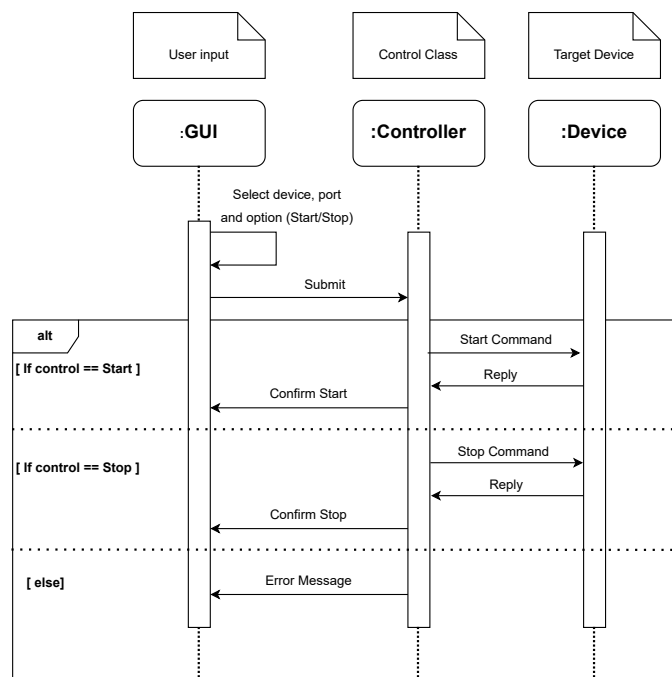


Figure 3.18: UML Control Sequence

### 3.3 Interface with Node-RED

Node-RED possesses an API that allows the user to communicate with Node-RED using its endpoints. The way to communicate with this application is by the use of HTTP protocol. It is required to have the *device URL* and *port* before choosing the method and the endpoint.

In the figure 3.19 is it possible to see a list of all methods that the Node-RED API provides.

Endpoint	Description
<b>GET</b> /auth/login	Get the active authentication scheme
<b>POST</b> /auth/token	Exchange credentials for access token
<b>POST</b> /auth/revoke	Revoke an access token
<b>GET</b> /settings	Get the runtime settings
<b>GET</b> /flows	Get the active flow configuration
<b>POST</b> /flows	Set the active flow configuration
<b>POST</b> /flow	Add a flow to the active configuration
<b>GET</b> /flow/:id	Get an individual flow configuration
<b>PUT</b> /flow/:id	Update an individual flow configuration
<b>DELETE</b> /flow/:id	Delete an individual flow configuration
<b>GET</b> /nodes	Get a list of the installed nodes
<b>POST</b> /nodes	Install a new node module
<b>GET</b> /nodes/:module	Get a node module's information
<b>PUT</b> /nodes/:module	Enable/Disable a node module
<b>DELETE</b> /nodes/:module	Remove a node module
<b>GET</b> /nodes/:module/:set	Get a node module set information
<b>PUT</b> /nodes/:module/:set	Enable/Disable a node set

Figure 3.19: Node-RED API Methods

#### 3.3.1 Communicating with Node-RED

In order to deploy a flow there is the need to input JSON code using the endpoints previously mentioned above. The `Controller` component possesses a class called `JSON_strings` that generates the necessary code to be posted in the endpoint once the generate *flow* button is submitted. In the figure 3.20 it is possible to see the format of the JSON code required to generate the flow.

```
[
  {
    "id": "b6f84af62f886671",
    "type": "inject",
    "z": "a36b4a3b94c6d8c4",
    "name": "",
    "props": [
      {
        "p": "payload"
      },
      {
        "p": "topic",
        "vt": "str"
      }
    ],
    "repeat": "",
    "crontab": "",
    "once": false,
    "onceDelay": 0.1,
    "topic": "",
    "payload": "",
    "payloadType": "date",
    "x": 320,
    "y": 80,
    "wires": [
      []
    ]
  }
]
```

Figure 3.20: Small example of JSON code

When creating a flow, it is required to access the endpoint */flow* with a *POST* method as in figure 3.21.

**POST**    **192.168.107.152 : 1880 / flow**

Method      Device URL      Port    Endpoint

Figure 3.21: URL for POST method

If the *POST* is successful, this framework returns the ID of the created *flow*. The response can be seen in the figure 3.22.

```
{
  "id": "51638b50621f7dd3",
}
```

Figure 3.22: Response from Node-RED

Once the *LEM tool* receives the *ID* of the flow, it uses a *PUT* method to inject the JSON code containing all the details of the *flow* that the user desires to deploy. In the figure 3.23 it is possible to see the endpoint that the user needs to access in order to deploy the flow properly.

**PUT**    **192.168.107.152 : 1880 / flow / 51638b50621f7dd3**

Method    Device URL    Port    Endpoint    Flow ID

Figure 3.23: URL for PUT method

If the user needs the JSON code of all flows, the URL for the HTTP GET method is in the figure 3.24, but in order to attain the information of only one flow the URL is represented in 3.25.

**GET**    **192.168.107.152: 1880 / flows**

Method    Device URL    Port    Endpoint

Figure 3.24: GET method for JSON string of all existing flows

**GET**    **192.168.107.152 : 1880 / flow / 51638b50621f7dd3**

Method    Device URL    Port    Endpoint    Flow ID

Figure 3.25: GET method for JSON string of a specific flow

### 3.3.2 Limitations of Node-RED

Even though Node-RED is suited to work with flows, there are some limitations to it.

1. This framework does not allow the user to force a specific ID on a flow;
2. The debugger does not pass information outside the Node-RED;
3. It is not possible to import a new flow code if there is one already with the same ID on a node;
4. If the PUT method is used incorrectly, the framework completely bugs and does not allow the user to delete the bugged flow manually;

Since it is not possible to force an ID for a flow, the coder needs to take this point into consideration. When using the method *POST* in the URL seen in the figure 3.21, the response from the Node-RED is the random generated ID. By having this ID, it is now possible to generate random IDs to all the nodes in the flow. The back end code takes the flow ID and adds five random characters in front it for each node. Then it used the *PUT* method seen in the figure 3.23 to inject the JSON code in Node-RED. The node ID structure can be seen in the figure 3.26.

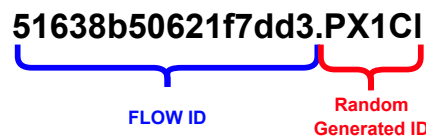


Figure 3.26: Node-RED Node ID

Each service has its own code that can be imported, but there is the need to tweak the JSON string a bit, so it does not repeat any node ID, since the PUT methods does not check if the node IDs already exist, to avoid the problem mentioned in the point four the limitations.

As mentioned, the debugger does not pass any type of information outside the framework. There are two ways of retrieving the value of any variable of the Node-RED. Write the value into a text file and store it in the device or convert all variables into global variables, to be accessible to any flow and POST this variable in a new URL.

In this project, it was opted to use the POST way, in order to create flexibility in the communication. Even though the goal of the project does not focus on the data that is being worked in the flow, it was required to use this POST method to retrieve the time of when the service triggers. Every time the executes, this timer is posted in an URL that can be seen in the figure 3.27.

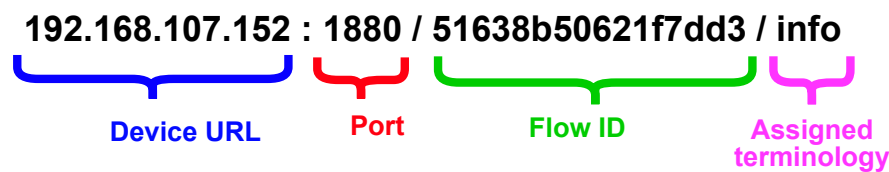


Figure 3.27: URL with the last time the service was executed

### 3.3.3 Monitor Node-RED flows

In order to monitor a *flow* with the *LEM tool*, the system requires two sets of three nodes (figure 3.28). These nodes provide support by posting the data contained in the `Template` node in a HTML page.

Once these two sets are deployed, the *LEM tool* uses a **HTTP GET** method in order to fetch the data that is visible in the URLs. The URLs that need to be accessed can be seen in the figures 3.27 and 3.29.

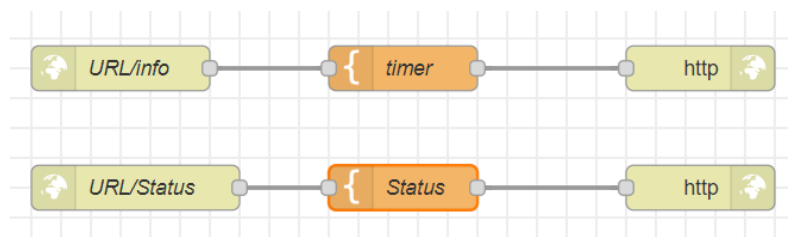


Figure 3.28: Set of nodes

**192.168.107.152 : 1880 /51638b50621f7dd3 / status**

Device URL      Port      Flow ID      Assigned Terminology

Figure 3.29: URL with the flow status

These sets are composed by two nodes that are responsible for the HTTP connection with the browser page, which are the two on the edges, and one node that has the information which is the one in the middle. The middle node is the `Template` which contains the information that wants to be passed, in the figure 3.30 it can be seen what information is being passed in the URL/info (looking closely, it is possible to see that the variable is global)

**Edit template node**

Delete Cancel Done

**Properties**

Name timer

Property msg. payload

Template Syntax Highlight: mustache

```
1 {{global.06164c6b5a1d8fd7time}}
```

Format Mustache template

Output as Plain text

Enabled

Figure 3.30: Information contained in the timer Template node



In order to know the time of execution and the status, these two sets need to exist in the flow, otherwise the *LEM tool* does not track the execution time nor the status of the flow. Does not really matter how another framework performs, as long it is possible to provide an access point to their status and time of execution.



## Chapter 4

# Validation

This chapter aims to explain the results attained with the developed tool as well as their validation. The section 4.1.2 explains how the developed project integrates the ML pipelines developed under the MIRAI project. The section 4.3 presents how the devices are managed as well as their response time.

### 4.1 Integration of the pipelines developed by MIRAI

The pipeline developed by MIRAI captures the traffic in the network and decides if the traffic is normal or not. In order to make this flow work it is required to have the following requisites in the target device, *Python2 with numpy library*, *Node-RED* and *Tstat* installed.

#### 4.1.1 Flow Description

The developed pipeline purpose is to capture network traffic and make decisions based on machine learning models, explained in detail in [42]. The pipeline can be seen in the figure 4.1, where it can be seen that is composed by fourteen nodes, which are:

- 5 Debuggers;
- 2 Execution;
- 1 Watch;
- 2 Function;
- 2 Inject;
- 2 Change.

The **debuggers** only exist for the user to look at the console while the flow is running in order to get feedback in the Node-RED platform.

The **Execution** nodes are the most important ones, composed by the *Tstat* and the *Python2* nodes. The *Tstat* calls the Tstat command to generate files that contain network traffic information in CSV (comma-separated values) files. Each line in the file represents a network entry containing features used in inference and training models.

The *Python2* executes a python script that reads line by line the traffic logs and runs a ML *training* model outputting an *inference* that provides information to whether the traffic is malicious or normal. An example of the output of this node can be seen in the figure B in the appendix.

The **watch** node is providing a path to where the traffic logs are allocated and checks if the folder suffers changes, in this case the folder is at /tmp/tgen. If a new log file is created by *Tstat*, the watch node will detect it. This value will be read by one of the function nodes (*watch /tmp/tgen*).

The **function** nodes are of different importance. One of them just converts the inference value into a string, enabling the user to see the output in the Node-RED console. The other one **watches** if there is any modification in the logs, filtering the changed information and passing it to the *python* node.

The **change** nodes exist to kill the process and convert the time of execution into a global variable.

The **inject** nodes release the command to run or stop the flow.

In the figure 4.1, it is possible to see the pipeline developed under the MIRAI project.

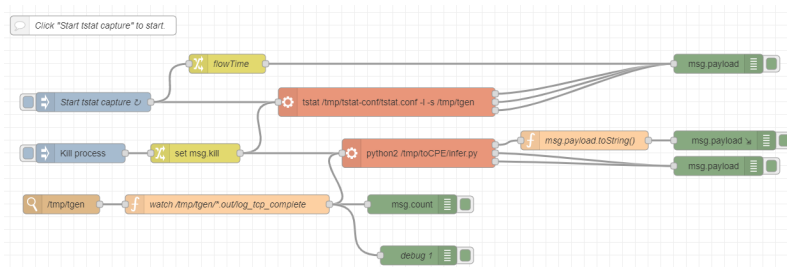


Figure 4.1: ML model pipeline developed by MIRAI

### 4.1.2 Integration with LEM tool

As previously mentioned in the section 3.3, in order to enable the monitoring of flows in the devices by *LEM tool* it is required to implement the two sets of nodes.

In order to deploy this flow, it was required to fetch its JSON code, edit it, in order to create random IDs for the flow and for each node and add the two sets of three nodes in order to communicate with the flow via HTTP.

In the figure 4.2 it is possible to see the integration of the two sets in the pipeline developed under the MIRAI project.

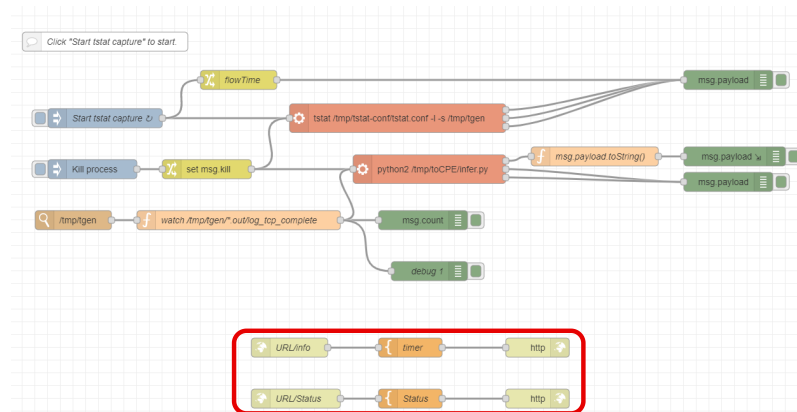


Figure 4.2: Adapted pipeline for monitoring

## 4.2 Validation of Deploy and Delete Functionalities

We validated the deployment and delete functionalities of a flow in a Raspberry.

The setup that was used was a laptop running the *LEM tool* and the target device was a Raspberry Pi with Node-RED installed. Both devices were connected to the same network. In figure 4.3 it is possible to see the used setup.

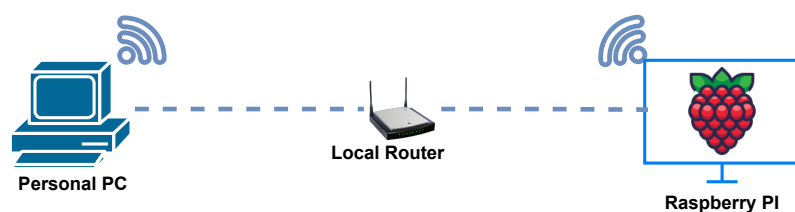


Figure 4.3: Raspberry Pi Setup

In this target device all the functionalities associated with flows were validated. In the figure 4.4 it is possible to see the Node-RED response after the *LEM tool* forces a flow to be created in its environment.

```
-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----
9 Feb 20:02:43 - [warn] Encrypted credentials not found
9 Feb 20:02:44 - [info] Server now running at http://127.0.0.1:1881/
9 Feb 20:02:44 - [info] Starting flows
9 Feb 20:02:44 - [info] Started flows
9 Feb 20:03:05 - [info] Updated flows
9 Feb 20:03:05 - [info] Adding flow: Name [3f67abea7f82638d]
9 Feb 20:03:06 - [info] Updated flows
9 Feb 20:03:06 - [info] Updated flow: Name [3f67abea7f82638d]
```

Figure 4.4: Node-RED response to Deploy

In the figure 4.5 shows the Node-RED response to a flow delete.

```
-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.


You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----
9 Feb 20:02:43 - [warn] Encrypted credentials not found
9 Feb 20:02:44 - [info] Server now running at http://127.0.0.1:1881/
9 Feb 20:02:44 - [info] Starting flows
9 Feb 20:02:44 - [info] Started flows
9 Feb 20:03:05 - [info] Updated flows
9 Feb 20:03:05 - [info] Adding flow: Name [3f67abea7f82638d]
9 Feb 20:03:06 - [info] Updated flows
9 Feb 20:03:06 - [info] Updated flow: Name [3f67abea7f82638d]
9 Feb 20:08:53 - [info] Updated flows
9 Feb 20:08:53 - [info] Removed flow: Name [3f67abea7f82638d] 
```

Figure 4.5: Node-RED response to Delete

The edit functionality response (figure 4.6) can be seen in the terminal as a flow has been updated.

```
-----
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
-----

9 Feb 20:20:17 - [warn] Encrypted credentials not found
9 Feb 20:20:17 - [info] Server now running at http://127.0.0.1:1881/
9 Feb 20:20:17 - [info] Starting flows
9 Feb 20:20:17 - [info] Started flows
9 Feb 20:20:26 - [info] Updated flows
9 Feb 20:20:26 - [info] Updated flow: Flow [842de936f3915fc3] ←
```

Figure 4.6: Node-RED response to Update

The Start and Stop functionalities, as well as the device delete/deploy are seen only in the GUI as a response message, since the terminal does not give any type of response to these. In the figures 4.7 - 4.10 it is possible to see the printed messages from the developed interface validating the tool.

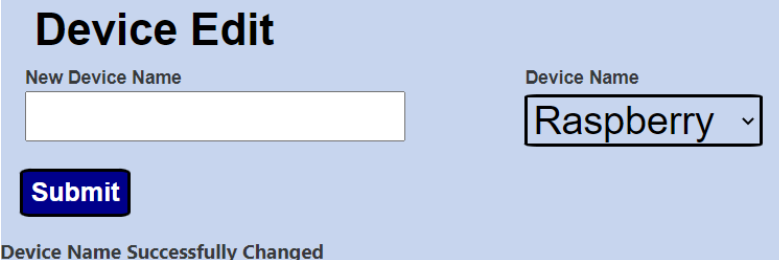


Figure 4.7: Interface response to Edit

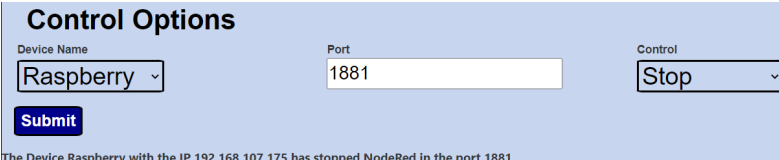


Figure 4.8: Interface response to Stop

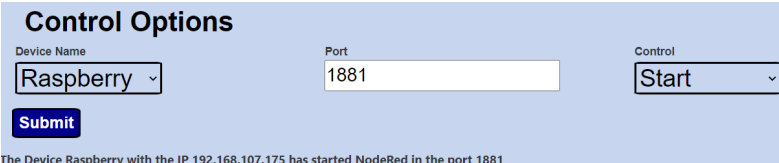


Figure 4.9: Interface response to Start



Figure 4.10: Interface response to Device Disassociation

With these tests it was possible to validate the *LEM tool*, since every functionality is working without any problem and it was deployed on an edge device.

### 4.3 Monitoring - Update Time of Flow Status

These tests check how long it takes for the tool to update all devices' information. In order to test scalability, the tests go from one to four devices and from one to twenty flows per device. It is required to have a NTP (Network Time Protocol) to sync both machines. The way to check if this protocol is running is by using a command in the terminal called `timedatectl`. In the figure 4.11 it is possible to see the status of the protocol inside the device.

```
rui@rui-VirtualBox:~$ timedatectl
    Local time: seg 2023-02-06 15:07:21 WET
   Universal time: seg 2023-02-06 15:07:21 UTC
          RTC time: seg 2023-02-06 15:07:20
        Time zone: Europe/Lisbon (WET, +0000)
System clock synchronized: yes
            NTP service: active
          RTC in local TZ: no
```




Figure 4.11: NTP Status in device

#### 4.3.1 Local Setup Results

In this test the laptop with the server was using the same network, which allowed faster responses than the previous one. The laptop was connected via wireless to the local network and communicated with a desktop which was connected to the same network via Ethernet. This target device had four VMs running, simulating multiple devices. This setup can be seen in the figure 4.12.



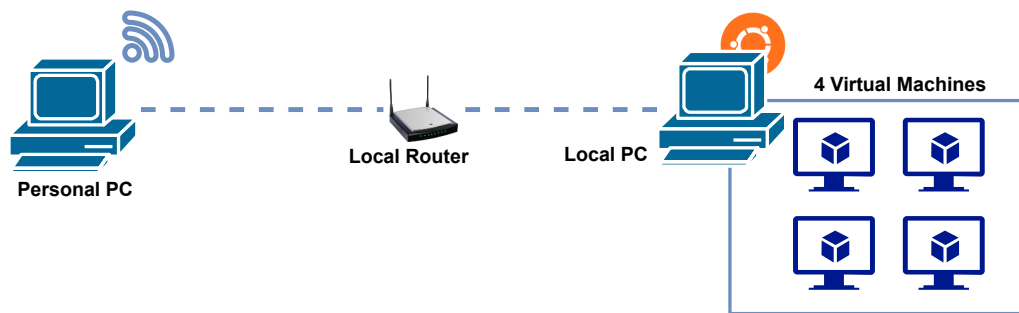


Figure 4.12: local Setup

### Impact of Multiple Devices

We evaluate the impact of having one or multiple devices has on the status collection time of the *LEM tool*. In the figures from 4.13 to 4.17 it is possible to see the time differences.

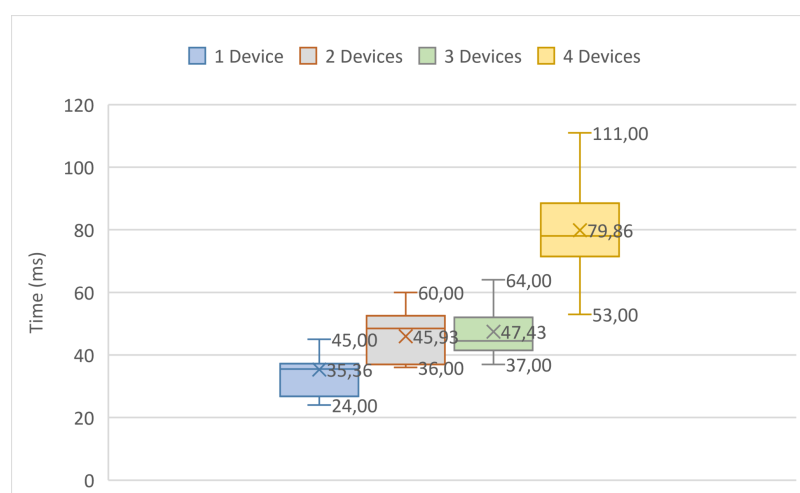


Figure 4.13: Local Setup - Multiple Devices with 1 Flow

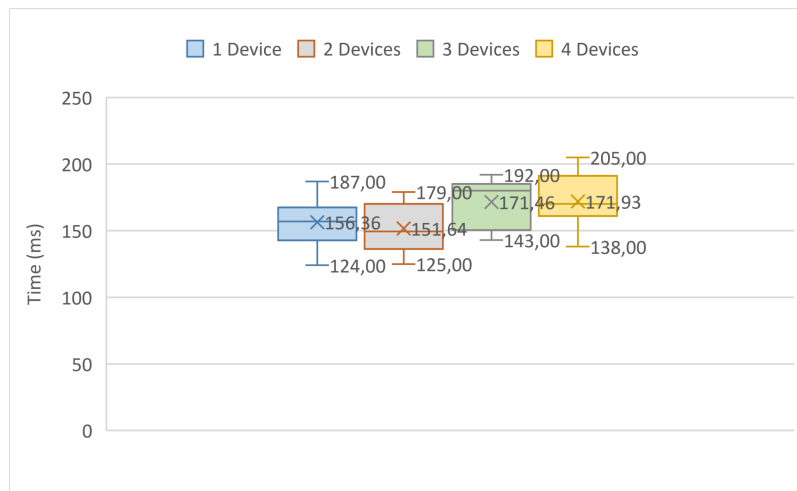


Figure 4.14: Local Setup - Multiple Devices with 5 Flows

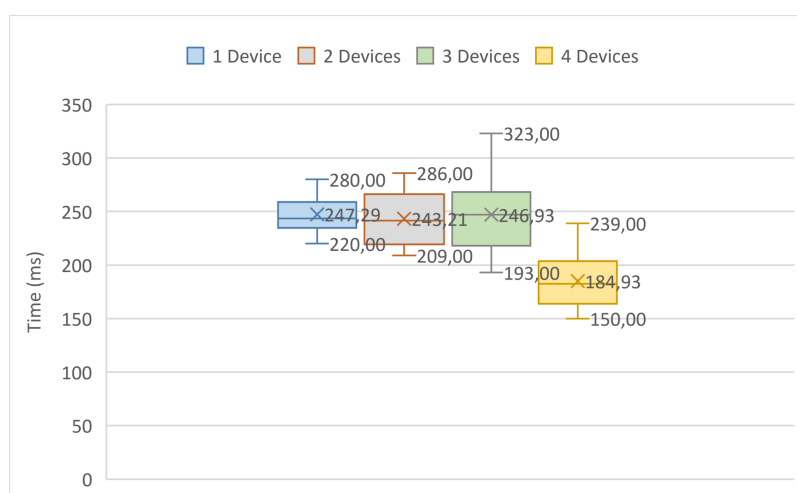


Figure 4.15: Local Setup - Multiple Devices with 10 Flows

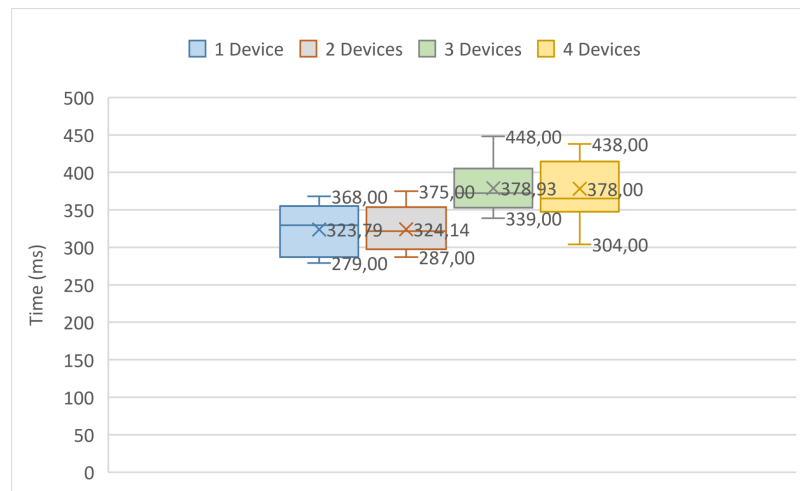


Figure 4.16: Local Setup - Multiple Devices with 15 Flows

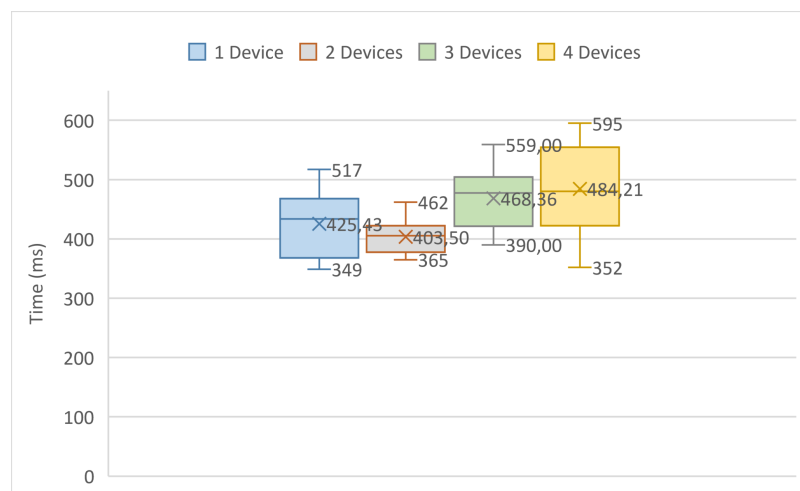


Figure 4.17: Local Setup - Multiple Devices with 20 Flows

The responses obtained from this test indicates that having one device or four can originate a variety of outputs. It is clear that more flows lead to higher response time (observe medians across Figures 4.13 to 4.17). This may be caused (or not) by higher load in the VM-hosting desktop; or Network traffic volume becomes larger as more flows exist, leading to increased delay.

Interestingly enough, an increase in the number of devices may not necessarily lead to a higher response due to the parallel thread approach of the *LEM tool* to probe the flow status in different

devices.

### Impact of Number of Flows

The quantity of existing flows in a device causes an impact in the time response of the *LEM tool*. If the device possesses multiple flows (figures 4.18 - 4.19) it increases the monitoring time exponentially.

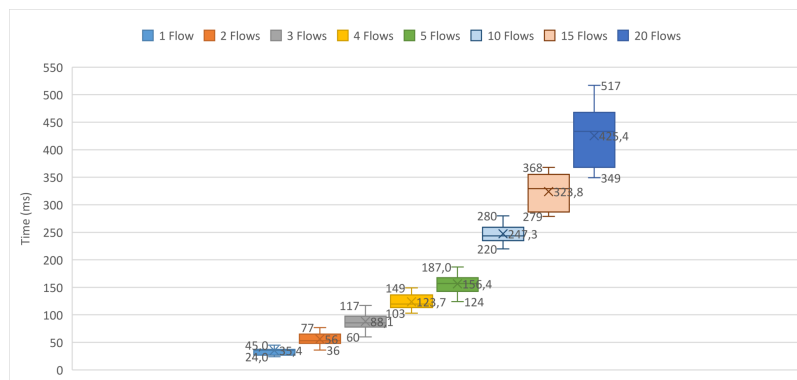


Figure 4.18: Local Setup - 1 Device with Multiple Flows

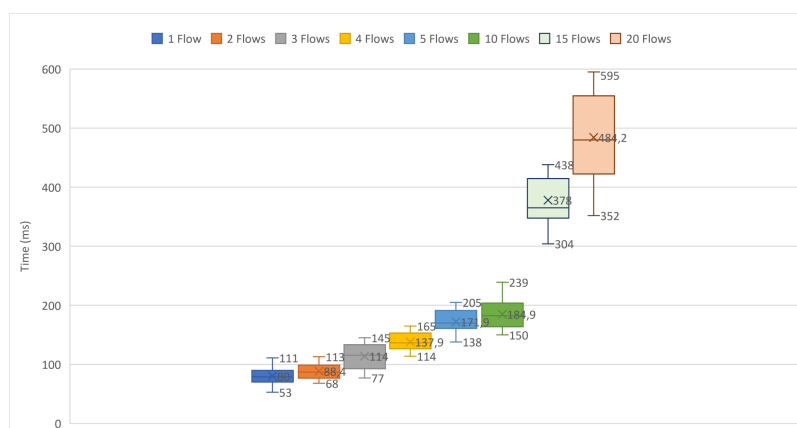


Figure 4.19: Local Setup - 4 Devices with Multiple Flows

As it can be seen in the previous figures, there is a different impact whether you are using 10 or 15 flows. There is a huge leap in time response.

Comparing medians in Fig. 4.18 and Fig.4.19, we observe that the increase of number of machines leads to some difference in response time, but with inferior impact than that of the increase of flows.

### 4.3.2 Remote Setup Results

In this test, the server was located far from the devices with a more or less 24 Km distance in a straight line. The laptop with where the *LEM tool* was running, was connected to the user's Remote internet, which connected to the local network via VPN (Virtual Private Network) to gain access to the local computer which was connected via Ethernet. The target device was running four VMs in order to simulate different devices. The setup can be seen in the figure 4.20.

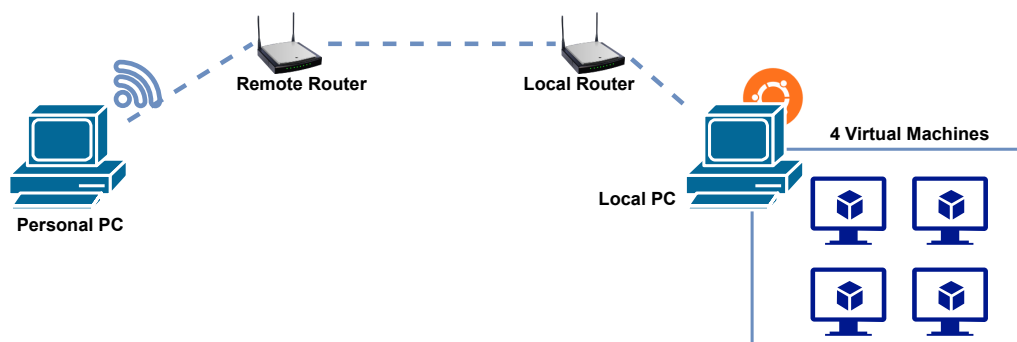


Figure 4.20: Remote Setup

The response time of the *LEM tool* with the increase of flows. It was tested for one and four connected devices. Possible causes are the same as mentioned before (network traffic volume becomes larger as more flows exist; more flows cause higher load in the VM-hosting desktop).

In figures 4.21 and 4.22 it is possible to see the response times with the increase of flows in each device.

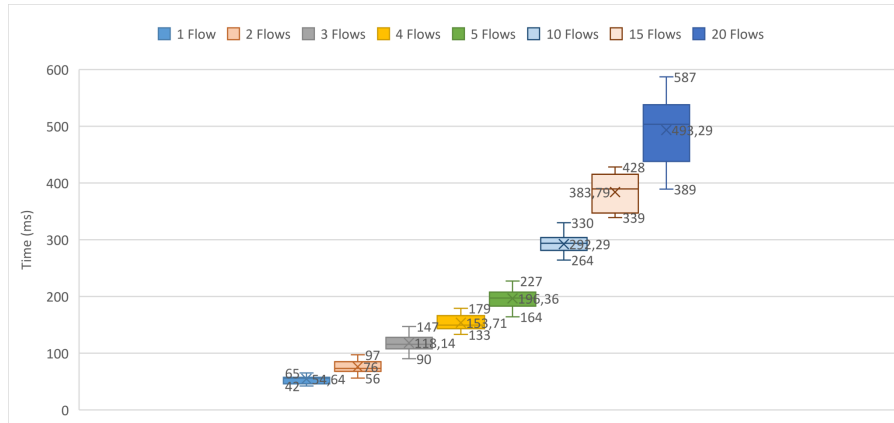


Figure 4.21: Remote Setup - 1 Device with Multiple Flows

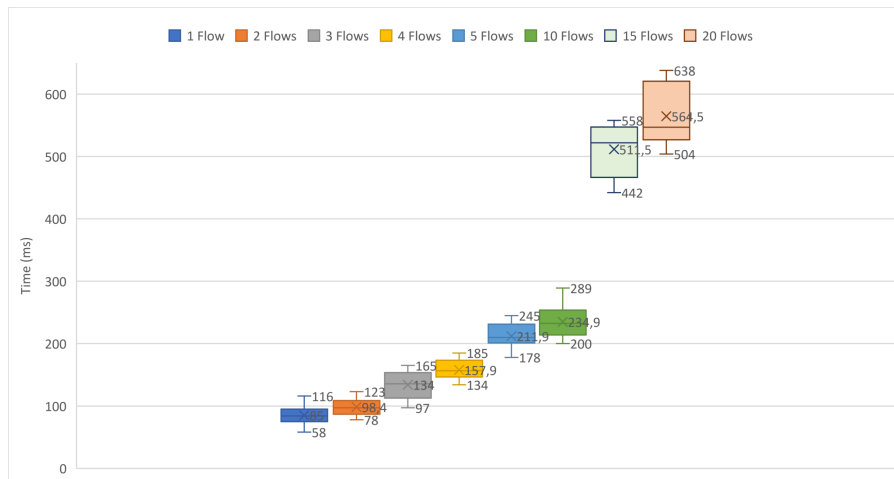


Figure 4.22: Remote Setup - 4 Devices with Multiple Flows

Looking into the tables in the appendix [A](#) it is possible to see that the time responses of the local and remote tests behave similar in the performed tests. The main difference is the time

difference between these two, being the remote test the slowest one.





## Chapter 5

# Conclusion

The need of workflows in edge devices has grown deeply in the last few years in order to avoid storage and computational issues. With this, the deployment of multiple IoT devices began to take place in the industrial environment. Controlling and monitoring a low amount of devices is easily done, but have a tool that is scalable, enabling the large scale control needs a lot of testing. Therefore this dissertation was born, with the goal of building a large scale monitoring and controlling tool of workflows in deployed edge devices.

MIRAI wants to solve some known problems (section 2.3.3 with the use of ML models. These models are running in workflows that are deployed in edge devices. This is where the *LEM tool* comes into play, to track all the workflows in each device associated to the system. This tool provides some features such as monitoring, deploy and control of the devices and flows.

In the literature review it was possible to study the frameworks that could suit the development of this tool was done. It was also possible to understand a bit more about how ML models work and how the user of the developed tool could gain visual access to the controlling system. System architecture and technological decisions were made, that provided the basis for the implementation stage with Node-RED ending up to be the used framework.

Once the development of the *LEM tool* was completed, tests were performed. The tool ran in a laptop while communicating with a Raspberry and a desktop that was simulating four different computers with the use of VMs. The Raspberry was a nice choice to validate the tool. The local setup as the graphs show, has fast responses with a low quantity of flows in each device, but if this value is increased, the process becomes slower. For last the remote setup behaved similarly to the local setup but was overall slower, this is due to the fact that both machines did not share the same network.

## 5.1 Future Work

Even though the project was successfully implemented there is still space to grown. There are some features that could be added to the tool to become more robust.

**Auto pairing:** Every time a device is associated with the *LEM tool*, the tool could be able to read if there is already any type of flow deployed in the device and fetch the data, to start monitoring.

**Frameworks:** Node-RED was the used framework for this project. It would be interesting to see the tool controlling the edge devices while these were running different frameworks.

**Enabling User:** Adding a feature where the user decides to create a new service, write down the required code on the tool and deploy it to the device.

**Reactive Interface:** It would be a great improvement to the tool to make the interface responsive instead of having to refresh to see the updated information.

To conclude, this project still has a lot of things that can be improved to become a more solid tool. Bear in mind, that anything added to the tool should be critically tested so it does not interfere with what already exists.

# Appendix A

## Response Times Tables

### A.1 Local Results per Flows

#### A.1.1 1 Flow on n Devices

Table A.1: Local Setup - Response Time 1 Flow with 1 Device

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 19:43:54.065	2023-02-08 19:43:54.102	37
2	2023-02-08 19:44:05.068	2023-02-08 19:44:05.106	38
3	2023-02-08 19:44:27.072	2023-02-08 19:44:27.136	64
4	2023-02-08 19:44:38.075	2023-02-08 19:44:38.101	26
5	2023-02-08 19:44:49.077	2023-02-08 19:44:49.101	24
6	2023-02-08 19:45:00.079	2023-02-08 19:45:00.111	32
7	2023-02-08 19:45:11.082	2023-02-08 19:45:11.118	36
8	2023-02-08 19:45:33.086	2023-02-08 19:45:33.123	37
9	2023-02-08 19:45:44.089	2023-02-08 19:45:44.121	32
10	2023-02-08 19:46:06.093	2023-02-08 19:46:06.119	26
11	2023-02-08 19:46:39.107	2023-02-08 19:46:39.134	27
12	2023-02-08 19:46:50.109	2023-02-08 19:46:50.154	45
13	2023-02-08 19:47:01.111	2023-02-08 19:47:01.146	35
14	2023-02-08 19:47:12.113	2023-02-08 19:47:12.149	36

Table A.2: Local Setup - Response Time 1 Flow with 2 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 19:39:47.969	2023-02-08 19:39:48.021	52
2	2023-02-08 19:39:58.972	2023-02-08 19:39:59.024	52
3	2023-02-08 19:40:09.975	2023-02-08 19:40:10.035	60
4	2023-02-08 19:40:20.977	2023-02-08 19:40:21.015	38
5	2023-02-08 19:40:42.982	2023-02-08 19:40:43.020	38
6	2023-02-08 19:40:53.984	2023-02-08 19:40:54.035	51
7	2023-02-08 19:41:04.986	2023-02-08 19:41:05.023	37
8	2023-02-08 19:41:15.988	2023-02-08 19:41:16.024	36
9	2023-02-08 19:41:37.995	2023-02-08 19:41:38.049	54
10	2023-02-08 19:41:48.998	2023-02-08 19:41:49.034	36
11	2023-02-08 19:42:00.000	2023-02-08 19:42:00.047	47
12	2023-02-08 19:42:11.004	2023-02-08 19:42:11.059	55
13	2023-02-08 19:42:33.010	2023-02-08 19:42:33.060	50
14	2023-02-08 19:42:44.016	2023-02-08 19:42:44.053	37

Table A.3: Local Setup - Response Time 1 Flow with 3 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 19:34:42.345	2023-02-08 19:34:42.390	45
2	2023-02-08 19:34:53.349	2023-02-08 19:34:53.392	43
3	2023-02-08 19:35:04.353	2023-02-08 19:35:04.397	44
4	2023-02-08 19:35:15.356	2023-02-08 19:35:15.425	69
5	2023-02-08 19:35:37.366	2023-02-08 19:35:37.412	46
6	2023-02-08 19:35:48.371	2023-02-08 19:35:48.411	40
7	2023-02-08 19:35:59.375	2023-02-08 19:35:59.414	39
8	2023-02-08 19:36:10.380	2023-02-08 19:36:10.424	44
9	2023-02-08 19:36:32.395	2023-02-08 19:36:32.440	45
10	2023-02-08 19:36:43.398	2023-02-08 19:36:43.435	37
11	2023-02-08 19:36:54.400	2023-02-08 19:36:54.451	51
12	2023-02-08 19:37:05.410	2023-02-08 19:37:05.452	42
13	2023-02-08 19:37:16.414	2023-02-08 19:37:16.478	64
14	2023-02-08 19:37:38.422	2023-02-08 19:37:38.477	55

Table A.4: Local Setup - Response Time 1 Flow with 4 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 16:14:43.448	2023-02-08 16:14:43.501	53
2	2023-02-08 16:15:14.454	2023-02-08 16:15:14.527	73
3	2023-02-08 16:15:45.460	2023-02-08 16:15:45.540	80
4	2023-02-08 16:16:16.463	2023-02-08 16:16:16.556	93
5	2023-02-08 16:16:47.467	2023-02-08 16:16:47.534	67
6	2023-02-08 16:17:49.482	2023-02-08 16:17:49.557	75
7	2023-02-08 16:18:20.484	2023-02-08 16:18:20.567	83
8	2023-02-08 16:18:51.491	2023-02-08 16:18:51.568	77
9	2023-02-08 16:19:22.496	2023-02-08 16:19:22.550	54
10	2023-02-08 16:19:53.499	2023-02-08 16:19:53.578	79
11	2023-02-08 16:20:24.508	2023-02-08 16:20:24.617	109
12	2023-02-08 16:22:28.592	2023-02-08 16:22:28.703	111
13	2023-02-08 16:23:30.635	2023-02-08 16:23:30.722	87
14	2023-02-08 16:24:01.647	2023-02-08 16:24:01.724	77

### A.1.2 5 Flow on n Devices

Table A.5: Local Setup - Response Time 5 Flows with 1 Device

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:09:23.976	2023-02-08 18:09:24.133	157
2	2023-02-08 18:09:45.986	2023-02-08 18:09:46.131	145
3	2023-02-08 18:09:56.989	2023-02-08 18:09:57.145	156
4	2023-02-08 18:10:07.991	2023-02-08 18:10:08.172	181
5	2023-02-08 18:10:29.995	2023-02-08 18:10:30.152	157
6	2023-02-08 18:10:40.997	2023-02-08 18:10:41.160	163
7	2023-02-08 18:10:52.001	2023-02-08 18:10:52.188	187
8	2023-02-08 18:11:03.005	2023-02-08 18:11:03.132	127
9	2023-02-08 18:11:14.007	2023-02-08 18:11:14.165	158
10	2023-02-08 18:11:25.009	2023-02-08 18:11:25.133	124
11	2023-02-08 18:11:36.011	2023-02-08 18:11:36.147	136
12	2023-02-08 18:11:58.015	2023-02-08 18:11:58.176	161
13	2023-02-08 18:12:09.018	2023-02-08 18:12:09.200	182
14	2023-02-08 18:12:42.024	2023-02-08 18:12:42.179	155

Table A.6: Local Setup - Response Time 5 Flows with 2 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:02:12.785	2023-02-08 18:02:12.961	176
2	2023-02-08 18:02:23.787	2023-02-08 18:02:23.925	138
3	2023-02-08 18:02:34.791	2023-02-08 18:02:34.955	164
4	2023-02-08 18:02:45.794	2023-02-08 18:02:45.953	159
5	2023-02-08 18:02:56.797	2023-02-08 18:02:56.923	126
6	2023-02-08 18:03:07.799	2023-02-08 18:03:07.968	169
7	2023-02-08 18:03:18.801	2023-02-08 18:03:18.947	146
8	2023-02-08 18:03:29.803	2023-02-08 18:03:29.956	153
9	2023-02-08 18:03:40.806	2023-02-08 18:03:40.985	179
10	2023-02-08 18:03:51.808	2023-02-08 18:03:51.933	125
11	2023-02-08 18:04:02.814	2023-02-08 18:04:02.958	144
12	2023-02-08 18:04:13.818	2023-02-08 18:04:13.949	131
13	2023-02-08 18:04:46.823	2023-02-08 18:04:46.963	140
14	2023-02-08 18:04:57.825	2023-02-08 18:04:57.998	173

Table A.7: Local Setup - Response Time 5 Flows with 3 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 17:57:43.098	2023-02-08 17:57:43.285	187
2	2023-02-08 17:57:54.105	2023-02-08 17:57:54.293	188
3	2023-02-08 17:58:05.108	2023-02-08 17:58:05.288	180
4	2023-02-08 17:58:16.116	2023-02-08 17:58:16.299	183
5	2023-02-08 17:58:27.119	2023-02-08 17:58:27.267	148
6	2023-02-08 17:58:38.122	2023-02-08 17:58:38.304	182
7	2023-02-08 17:58:49.127	2023-02-08 17:58:49.303	176
8	2023-02-08 17:59:11.133	2023-02-08 17:59:11.284	151
9	2023-02-08 17:59:22.136	2023-02-08 17:59:22.322	186
10	2023-02-08 17:59:55.148	2023-02-08 17:59:55.314	166
11	2023-02-08 18:00:06.152	2023-02-08 18:00:06.336	184
12	2023-02-08 18:00:28.158	2023-02-08 18:00:28.350	192
13	2023-02-08 18:00:39.160	2023-02-08 18:00:39.303	143
14	2023-02-08 18:00:50.150	2023-02-08 18:00:50.300	150

Table A.8: Local Setup - Response Time 5 Flows with 4 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 17:53:02.050	2023-02-08 17:53:02.223	173
2	2023-02-08 17:53:35.089	2023-02-08 17:53:35.231	142
3	2023-02-08 17:53:46.095	2023-02-08 17:53:46.286	191
4	2023-02-08 17:53:57.115	2023-02-08 17:53:57.282	167
5	2023-02-08 17:54:08.118	2023-02-08 17:54:08.281	163
6	2023-02-08 17:54:19.123	2023-02-08 17:54:19.328	205
7	2023-02-08 17:54:30.127	2023-02-08 17:54:30.312	185
8	2023-02-08 17:54:41.132	2023-02-08 17:54:41.305	173
9	2023-02-08 17:54:52.147	2023-02-08 17:54:52.302	155
10	2023-02-08 17:55:14.153	2023-02-08 17:55:14.317	164
11	2023-02-08 17:55:25.157	2023-02-08 17:55:25.349	192
12	2023-02-08 17:55:36.161	2023-02-08 17:55:36.325	164
13	2023-02-08 17:55:58.180	2023-02-08 17:55:58.375	195
14	2023-02-08 17:56:09.182	2023-02-08 17:56:09.320	138

### A.1.3 10 Flows on n Devices

Table A.9: Local Setup - Response Time 10 Flows with 1 Device

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:37:31.421	2023-02-08 18:37:31.689	268
2	2023-02-08 18:37:42.424	2023-02-08 18:37:42.672	248
3	2023-02-08 18:37:53.426	2023-02-08 18:37:53.667	241
4	2023-02-08 18:38:15.430	2023-02-08 18:38:15.666	236
5	2023-02-08 18:38:48.436	2023-02-08 18:38:48.715	279
6	2023-02-08 18:39:10.440	2023-02-08 18:39:10.720	280
7	2023-02-08 18:39:21.442	2023-02-08 18:39:21.688	246
8	2023-02-08 18:39:32.444	2023-02-08 18:39:32.693	249
9	2023-02-08 18:39:43.447	2023-02-08 18:39:43.687	240
10	2023-02-08 18:39:54.452	2023-02-08 18:39:54.708	256
11	2023-02-08 18:40:16.456	2023-02-08 18:40:16.690	234
12	2023-02-08 18:40:27.457	2023-02-08 18:40:27.687	230
13	2023-02-08 18:40:49.460	2023-02-08 18:40:49.680	220
14	2023-02-08 18:41:11.464	2023-02-08 18:41:11.699	235

Table A.10: Local Setup - Response Time 10 Flows with 2 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:32:52.958	2023-02-08 18:32:53.210	252
2	2023-02-08 18:33:14.962	2023-02-08 18:33:15.235	273
3	2023-02-08 18:33:36.967	2023-02-08 18:33:37.203	236
4	2023-02-08 18:33:47.981	2023-02-08 18:33:48.213	232
5	2023-02-08 18:33:58.989	2023-02-08 18:33:59.255	266
6	2023-02-08 18:34:09.991	2023-02-08 18:34:10.249	258
7	2023-02-08 18:34:20.993	2023-02-08 18:34:21.202	209
8	2023-02-08 18:34:54.012	2023-02-08 18:34:54.259	247
9	2023-02-08 18:35:16.015	2023-02-08 18:35:16.232	217
10	2023-02-08 18:35:27.017	2023-02-08 18:35:27.237	220
11	2023-02-08 18:35:38.019	2023-02-08 18:35:38.286	267
12	2023-02-08 18:35:49.021	2023-02-08 18:35:49.238	217
13	2023-02-08 18:36:00.022	2023-02-08 18:36:00.308	286
14	2023-02-08 18:36:11.023	2023-02-08 18:36:11.248	225

Table A.11: Local Setup - Response Time 10 Flows with 3 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:28:23.896	2023-02-08 18:28:24.219	323
2	2023-02-08 18:28:56.910	2023-02-08 18:28:57.161	251
3	2023-02-08 18:29:07.918	2023-02-08 18:29:08.207	289
4	2023-02-08 18:29:29.940	2023-02-08 18:29:30.215	275
5	2023-02-08 18:29:40.942	2023-02-08 18:29:41.193	251
6	2023-02-08 18:30:02.960	2023-02-08 18:30:03.226	266
7	2023-02-08 18:30:13.962	2023-02-08 18:30:14.205	243
8	2023-02-08 18:30:24.965	2023-02-08 18:30:25.219	254
9	2023-02-08 18:30:35.969	2023-02-08 18:30:36.188	219
10	2023-02-08 18:30:46.971	2023-02-08 18:30:47.195	224
11	2023-02-08 18:30:57.976	2023-02-08 18:30:58.191	215
12	2023-02-08 18:31:19.981	2023-02-08 18:31:20.223	242
13	2023-02-08 18:31:30.983	2023-02-08 18:31:31.176	193
14	2023-02-08 18:31:41.985	2023-02-08 18:31:42.197	212



Table A.12: Local Setup - Response Time 10 Flows with 4 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:17:56.988	2023-02-08 18:17:57.211	223
2	2023-02-08 18:18:52.025	2023-02-08 18:18:52.264	239
3	2023-02-08 18:19:36.034	2023-02-08 18:19:36.184	150
4	2023-02-08 18:19:47.036	2023-02-08 18:19:47.217	181
5	2023-02-08 18:19:58.040	2023-02-08 18:19:58.205	165
6	2023-02-08 18:20:09.042	2023-02-08 18:20:09.197	155
7	2023-02-08 18:20:20.045	2023-02-08 18:20:20.212	167
8	2023-02-08 18:20:31.047	2023-02-08 18:20:31.244	197
9	2023-02-08 18:20:42.049	2023-02-08 18:20:42.233	184
10	2023-02-08 18:20:53.052	2023-02-08 18:20:53.264	212
11	2023-02-08 18:21:04.054	2023-02-08 18:21:04.223	169
12	2023-02-08 18:21:15.056	2023-02-08 18:21:15.216	160
13	2023-02-08 18:21:26.058	2023-02-08 18:21:26.244	186
14	2023-02-08 18:21:37.061	2023-02-08 18:21:37.262	201

#### A.1.4 15 Flows on n Devices

Table A.13: Local Setup - Response Time 15 Flows with 1 Device

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:58:55.227	2023-02-08 18:58:55.582	355
2	2023-02-08 18:59:17.231	2023-02-08 18:59:17.587	356
3	2023-02-08 18:59:39.236	2023-02-08 18:59:39.515	279
4	2023-02-08 18:59:50.238	2023-02-08 18:59:50.606	368
5	2023-02-08 19:00:01.240	2023-02-08 19:00:01.593	353
6	2023-02-08 19:00:12.242	2023-02-08 19:00:12.590	348
7	2023-02-08 19:00:23.244	2023-02-08 19:00:23.580	336
8	2023-02-08 19:00:34.245	2023-02-08 19:00:34.535	290
9	2023-02-08 19:00:45.247	2023-02-08 19:00:45.534	287
10	2023-02-08 19:00:56.249	2023-02-08 19:00:56.572	323
11	2023-02-08 19:01:07.253	2023-02-08 19:01:07.556	303
12	2023-02-08 19:01:18.255	2023-02-08 19:01:18.538	283
13	2023-02-08 19:01:29.256	2023-02-08 19:01:29.543	287
14	2023-02-08 19:01:40.258	2023-02-08 19:01:40.623	365

Table A.14: Local Setup - Response Time 15 Flows with 2 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:54:13.474	2023-02-08 18:54:13.827	353
2	2023-02-08 18:54:24.476	2023-02-08 18:54:24.829	353
3	2023-02-08 18:54:35.478	2023-02-08 18:54:35.834	356
4	2023-02-08 18:54:46.479	2023-02-08 18:54:46.779	300
5	2023-02-08 18:54:57.484	2023-02-08 18:54:57.791	307
6	2023-02-08 18:55:19.490	2023-02-08 18:55:19.847	357
7	2023-02-08 18:55:52.495	2023-02-08 18:55:52.823	328
8	2023-02-08 18:56:03.497	2023-02-08 18:56:03.872	375
9	2023-02-08 18:56:14.498	2023-02-08 18:56:14.787	289
10	2023-02-08 18:56:25.500	2023-02-08 18:56:25.802	302
11	2023-02-08 18:56:36.502	2023-02-08 18:56:36.830	328
12	2023-02-08 18:56:47.504	2023-02-08 18:56:47.819	315
13	2023-02-08 18:56:58.505	2023-02-08 18:56:58.792	287
14	2023-02-08 18:57:09.507	2023-02-08 18:57:09.795	288

Table A.15: Local Setup - Response Time 15 Flows with 3 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:49:12.906	2023-02-08 18:49:13.354	448
2	2023-02-08 18:49:23.909	2023-02-08 18:49:24.248	339
3	2023-02-08 18:49:34.912	2023-02-08 18:49:35.289	377
4	2023-02-08 18:49:45.915	2023-02-08 18:49:46.267	352
5	2023-02-08 18:49:56.922	2023-02-08 18:49:57.289	367
6	2023-02-08 18:50:18.927	2023-02-08 18:50:19.336	409
7	2023-02-08 18:50:29.930	2023-02-08 18:50:30.298	368
8	2023-02-08 18:50:51.947	2023-02-08 18:50:52.364	417
9	2023-02-08 18:51:02.949	2023-02-08 18:51:03.353	404
10	2023-02-08 18:51:13.951	2023-02-08 18:51:14.329	378
11	2023-02-08 18:51:24.953	2023-02-08 18:51:25.304	351
12	2023-02-08 18:51:46.956	2023-02-08 18:51:47.309	353
13	2023-02-08 18:51:57.960	2023-02-08 18:51:58.317	357
14	2023-02-08 18:52:08.965	2023-02-08 18:52:09.350	385

Table A.16: Local Setup - Response Time 15 Flows with 4 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 18:44:34.852	2023-02-08 18:44:35.765	409
2	2023-02-08 18:44:56.963	2023-02-08 18:44:57.372	431
3	2023-02-08 18:45:18.992	2023-02-08 18:45:19.423	362
4	2023-02-08 18:45:52.008	2023-02-08 18:45:52.370	438
5	2023-02-08 18:46:03.010	2023-02-08 18:46:03.448	387
6	2023-02-08 18:46:14.018	2023-02-08 18:46:14.405	342
7	2023-02-08 18:46:25.020	2023-02-08 18:46:25.367	436
8	2023-02-08 18:46:36.026	2023-02-08 18:46:36.462	357
9	2023-02-08 18:46:47.033	2023-02-08 18:46:47.390	358
10	2023-02-08 18:47:09.041	2023-02-08 18:47:09.399	406
11	2023-02-08 18:47:20.044	2023-02-08 18:47:20.450	368
12	2023-02-08 18:47:31.047	2023-02-08 18:47:31.415	348
13	2023-02-08 18:47:42.051	2023-02-08 18:47:42.399	346
14	2023-02-08 18:47:53.054	2023-02-08 18:47:53.400	304

### A.1.5 20 Flows on n Devices

Table A.17: Local Setup - Response Time 20 Flows with 1 Device

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 19:19:40.786	2023-02-08 19:19:41.206	420
2	2023-02-08 19:19:51.789	2023-02-08 19:19:52.224	435
3	2023-02-08 19:20:02.794	2023-02-08 19:20:03.268	474
4	2023-02-08 19:20:24.798	2023-02-08 19:20:25.242	444
5	2023-02-08 19:20:35.800	2023-02-08 19:20:36.162	362
6	2023-02-08 19:20:57.803	2023-02-08 19:20:58.320	517
7	2023-02-08 19:21:08.805	2023-02-08 19:21:09.276	471
8	2023-02-08 19:21:19.807	2023-02-08 19:21:20.274	467
9	2023-02-08 19:21:30.808	2023-02-08 19:21:31.243	435
10	2023-02-08 19:21:52.811	2023-02-08 19:21:53.166	355
11	2023-02-08 19:22:03.816	2023-02-08 19:22:04.241	425
12	2023-02-08 19:22:14.818	2023-02-08 19:22:15.188	370
13	2023-02-08 19:22:25.819	2023-02-08 19:22:26.251	432
14	2023-02-08 19:22:47.822	2023-02-08 19:22:48.171	349

Table A.18: Local Setup - Response Time 20 Flows with 2 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 19:15:15.256	2023-02-08 19:15:15.676	420
2	2023-02-08 19:15:48.264	2023-02-08 19:15:48.696	432
3	2023-02-08 19:15:59.267	2023-02-08 19:15:59.696	429
4	2023-02-08 19:16:21.280	2023-02-08 19:16:21.696	416
5	2023-02-08 19:16:32.282	2023-02-08 19:16:32.677	395
6	2023-02-08 19:16:43.284	2023-02-08 19:16:43.662	378
7	2023-02-08 19:16:54.285	2023-02-08 19:16:54.747	462
8	2023-02-08 19:17:05.287	2023-02-08 19:17:05.702	415
9	2023-02-08 19:17:16.288	2023-02-08 19:17:16.699	411
10	2023-02-08 19:17:27.290	2023-02-08 19:17:27.668	378
11	2023-02-08 19:17:38.291	2023-02-08 19:17:38.656	365
12	2023-02-08 19:17:49.293	2023-02-08 19:17:49.670	377
13	2023-02-08 19:18:00.294	2023-02-08 19:18:00.694	400
14	2023-02-08 19:18:11.296	2023-02-08 19:18:11.667	371

Table A.19: Local Setup - Response Time 20 Flows with 3 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 19:10:13.221	2023-02-08 19:10:13.780	559
2	2023-02-08 19:10:24.225	2023-02-08 19:10:24.702	477
3	2023-02-08 19:10:35.237	2023-02-08 19:10:35.715	478
4	2023-02-08 19:10:46.239	2023-02-08 19:10:46.721	482
5	2023-02-08 19:11:19.260	2023-02-08 19:11:19.651	391
6	2023-02-08 19:11:30.262	2023-02-08 19:11:30.700	438
7	2023-02-08 19:11:52.267	2023-02-08 19:11:52.749	482
8	2023-02-08 19:12:03.269	2023-02-08 19:12:03.781	512
9	2023-02-08 19:12:14.271	2023-02-08 19:12:14.730	459
10	2023-02-08 19:12:25.272	2023-02-08 19:12:25.694	422
11	2023-02-08 19:12:36.274	2023-02-08 19:12:36.664	390
12	2023-02-08 19:12:47.276	2023-02-08 19:12:47.778	502
13	2023-02-08 19:12:58.297	2023-02-08 19:12:58.716	419
14	2023-02-08 19:13:09.299	2023-02-08 19:13:09.845	546

Table A.20: Local Setup - Response Time 20 Flows with 4 Devices

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 19:05:42.128	2023-02-08 19:05:43.645	517
2	2023-02-08 19:05:53.251	2023-02-08 19:05:53.826	575
3	2023-02-08 19:07:10.287	2023-02-08 19:07:10.726	439
4	2023-02-08 19:07:21.290	2023-02-08 19:07:21.810	520
5	2023-02-08 19:07:32.299	2023-02-08 19:07:32.862	563
6	2023-02-08 19:07:43.302	2023-02-08 19:07:43.897	595
7	2023-02-08 19:07:54.304	2023-02-08 19:07:54.713	409
8	2023-02-08 19:08:05.307	2023-02-08 19:08:05.848	541
9	2023-02-08 19:08:16.312	2023-02-08 19:08:16.754	442
10	2023-02-08 19:08:27.314	2023-02-08 19:08:27.718	404
11	2023-02-08 19:08:38.320	2023-02-08 19:08:38.763	443
12	2023-02-08 19:08:49.322	2023-02-08 19:08:49.674	352
13	2023-02-08 19:09:00.324	2023-02-08 19:09:00.751	427
14	2023-02-08 19:09:11.329	2023-02-08 19:09:11.881	552

## A.2 Local Results per Devices

### A.2.1 Flows on 1 Device

Table A.21: Local Setup - Response Time 1 Device with 2 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 17:06:23.816	2023-02-08 17:06:23.875	59
2	2023-02-08 17:06:34.818	2023-02-08 17:06:34.872	54
3	2023-02-08 17:06:45.82	2023-02-08 17:06:45.891	71
4	2023-02-08 17:06:56.822	2023-02-08 17:06:56.874	52
5	2023-02-08 17:07:07.825	2023-02-08 17:07:07.902	77
6	2023-02-08 17:07:18.826	2023-02-08 17:07:18.875	49
7	2023-02-08 17:07:29.828	2023-02-08 17:07:29.891	63
8	2023-02-08 17:07:40.831	2023-02-08 17:07:40.905	74
9	2023-02-08 17:08:13.836	2023-02-08 17:08:13.887	51
10	2023-02-08 17:08:24.838	2023-02-08 17:08:24.883	45
11	2023-02-08 17:08:35.841	2023-02-08 17:08:35.877	36
12	2023-02-08 17:08:57.844	2023-02-08 17:08:57.895	51
13	2023-02-08 17:09:19.852	2023-02-08 17:09:19.896	44
14	2023-02-08 17:09:30.853	2023-02-08 17:09:30.911	58

Table A.22: Local Setup - Response Time 1 Device with 3 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 17:26:16.957	2023-02-08 17:26:17.042	85
2	2023-02-08 17:26:38.962	2023-02-08 17:26:39.048	86
3	2023-02-08 17:26:49.964	2023-02-08 17:26:50.081	117
4	2023-02-08 17:27:00.967	2023-02-08 17:27:01.041	74
5	2023-02-08 17:27:11.969	2023-02-08 17:27:12.063	94
6	2023-02-08 17:27:22.971	2023-02-08 17:27:23.059	88
7	2023-02-08 17:27:33.973	2023-02-08 17:27:34.062	89
8	2023-02-08 17:27:44.975	2023-02-08 17:27:45.084	109
9	2023-02-08 17:27:55.977	2023-02-08 17:27:56.037	60
10	2023-02-08 17:28:17.982	2023-02-08 17:28:18.056	74
11	2023-02-08 17:28:28.984	2023-02-08 17:28:29.064	80
12	2023-02-08 17:28:50.987	2023-02-08 17:28:51.103	116
13	2023-02-08 17:29:23.994	2023-02-08 17:29:24.073	79
14	2023-02-08 17:29:34.996	2023-02-08 17:29:35.079	83

Table A.23: Local Setup - Response Time 1 Device with 4 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 17:46:25.671	2023-02-08 17:46:25.801	130
2	2023-02-08 17:46:47.677	2023-02-08 17:46:47.823	146
3	2023-02-08 17:46:58.679	2023-02-08 17:46:58.828	149
4	2023-02-08 17:47:20.686	2023-02-08 17:47:20.823	137
5	2023-02-08 17:47:31.689	2023-02-08 17:47:31.804	115
6	2023-02-08 17:47:42.693	2023-02-08 17:47:42.816	123
7	2023-02-08 17:48:04.698	2023-02-08 17:48:04.814	116
8	2023-02-08 17:48:15.700	2023-02-08 17:48:15.805	105
9	2023-02-08 17:48:26.702	2023-02-08 17:48:26.833	131
10	2023-02-08 17:48:37.705	2023-02-08 17:48:37.821	116
11	2023-02-08 17:48:48.708	2023-02-08 17:48:48.844	136
12	2023-02-08 17:49:21.714	2023-02-08 17:49:21.825	111
13	2023-02-08 17:49:43.718	2023-02-08 17:49:43.821	103
14	2023-02-08 17:49:54.720	2023-02-08 17:49:54.834	114

## A.2.2 Flows on 4 Devices

Table A.24: Local Setup - Response Time 4 Device with 2 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 16:32:48.810	2023-02-08 16:32:48.893	83
2	2023-02-08 16:33:09.814	2023-02-08 16:33:09.927	113
3	2023-02-08 16:33:51.823	2023-02-08 16:33:51.911	88
4	2023-02-08 16:34:33.842	2023-02-08 16:34:33.943	101
5	2023-02-08 16:34:54.850	2023-02-08 16:34:54.937	87
6	2023-02-08 16:35:15.855	2023-02-08 16:35:15.964	109
7	2023-02-08 16:35:57.883	2023-02-08 16:35:57.981	98
8	2023-02-08 16:36:39.893	2023-02-08 16:36:39.968	75
9	2023-02-08 16:37:42.912	2023-02-08 16:37:42.980	68
10	2023-02-08 16:38:45.920	2023-02-08 16:38:45.997	77
11	2023-02-08 16:41:33.975	2023-02-08 16:41:34.051	76
12	2023-02-08 16:41:54.978	2023-02-08 16:41:55.061	83
13	2023-02-08 16:43:18.994	2023-02-08 16:43:19.081	87
14	2023-02-08 16:44:01.008	2023-02-08 16:44:01.100	92

Table A.25: Local Setup - Response Time 4 Device with 3 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 17:11:48.308	2023-02-08 17:11:48.444	136
2	2023-02-08 17:11:59.312	2023-02-08 17:11:59.427	115
3	2023-02-08 17:12:10.331	2023-02-08 17:12:10.440	109
4	2023-02-08 17:12:21.336	2023-02-08 17:12:21.426	90
5	2023-02-08 17:12:32.339	2023-02-08 17:12:32.452	113
6	2023-02-08 17:12:43.342	2023-02-08 17:12:43.435	93
7	2023-02-08 17:12:54.347	2023-02-08 17:12:54.439	92
8	2023-02-08 17:13:27.367	2023-02-08 17:13:27.483	116
9	2023-02-08 17:14:00.389	2023-02-08 17:14:00.524	135
10	2023-02-08 17:14:22.406	2023-02-08 17:14:22.551	145
11	2023-02-08 17:14:33.412	2023-02-08 17:14:33.533	121
12	2023-02-08 17:14:44.424	2023-02-08 17:14:44.557	133
13	2023-02-08 17:14:55.428	2023-02-08 17:14:55.549	121
14	2023-02-08 17:15:06.430	2023-02-08 17:15:06.507	77

Table A.26: Local Setup - Response Time 4 Device with 4 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-08 17:32:40.839	2023-02-08 17:32:40.973	134
2	2023-02-08 17:33:02.851	2023-02-08 17:33:03.005	154
3	2023-02-08 17:33:13.855	2023-02-08 17:33:14.008	153
4	2023-02-08 17:33:24.859	2023-02-08 17:33:24.989	130
5	2023-02-08 17:33:46.876	2023-02-08 17:33:47.007	131
6	2023-02-08 17:33:57.898	2023-02-08 17:33:58.037	139
7	2023-02-08 17:34:30.910	2023-02-08 17:34:31.075	165
8	2023-02-08 17:35:03.937	2023-02-08 17:35:04.082	145
9	2023-02-08 17:35:14.943	2023-02-08 17:35:17.102	159
10	2023-02-08 17:35:25.948	2023-02-08 17:35:26.070	122
11	2023-02-08 17:35:36.952	2023-02-08 17:35:37.094	142
12	2023-02-08 17:35:47.957	2023-02-08 17:35:48.085	128
13	2023-02-08 17:36:09.967	2023-02-08 17:36:10.082	115
14	2023-02-08 17:36:20.970	2023-02-08 17:36:21.084	114

### A.3 Remote Results per Devices

#### A.3.1 Flows on 1 Device

Table A.27: Remote Setup - Response Time 1 Device with 1 Flow

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 12:13:54.065	2023-02-09 12:13:54.122	57
2	2023-02-09 12:14:05.068	2023-02-09 12:14:05.126	58
3	2023-02-09 12:14:27.072	2023-02-09 12:14:27.156	84
4	2023-02-09 12:14:38.075	2023-02-09 12:14:38.121	46
5	2023-02-09 12:14:49.077	2023-02-09 12:14:49.121	44
6	2023-02-09 12:15:00.079	2023-02-09 12:15:00.121	42
7	2023-02-09 12:15:11.082	2023-02-09 12:15:11.128	56
8	2023-02-09 12:15:33.086	2023-02-09 12:15:33.143	57
9	2023-02-09 12:15:44.089	2023-02-09 12:15:44.141	52
10	2023-02-09 12:16:06.093	2023-02-09 12:16:06.139	46
11	2023-02-09 12:16:39.107	2023-02-09 12:16:39.154	47
12	2023-02-09 12:16:50.109	2023-02-09 12:16:50.174	65
13	2023-02-09 12:17:01.111	2023-02-09 12:17:01.166	55
14	2023-02-09 12:17:12.113	2023-02-09 12:17:12.169	56



Table A.28: Remote Setup - Response Time 1 Device with 2 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 11:06:23.816	2023-02-09 11:06:23.895	79
2	2023-02-09 11:06:34.818	2023-02-09 11:06:34.892	74
3	2023-02-09 11:06:45.82	2023-02-09 11:06:45.911	91
4	2023-02-09 11:06:56.822	2023-02-09 11:06:56.894	72
5	2023-02-09 11:07:07.825	2023-02-09 11:07:07.922	97
6	2023-02-09 11:07:18.826	2023-02-09 11:07:18.895	69
7	2023-02-09 11:07:29.828	2023-02-09 11:07:29.911	83
8	2023-02-09 11:07:40.831	2023-02-09 11:07:40.925	94
9	2023-02-09 11:08:13.836	2023-02-09 11:08:13.907	71
10	2023-02-09 11:08:24.838	2023-02-09 11:08:24.903	65
11	2023-02-09 11:08:35.841	2023-02-09 11:08:35.897	56
12	2023-02-09 11:08:57.844	2023-02-09 11:08:57.915	71
13	2023-02-09 11:09:19.852	2023-02-09 11:09:19.916	64
14	2023-02-09 11:09:30.853	2023-02-09 11:09:30.931	78

Table A.29: Remote Setup - Response Time 1 Device with 3 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 11:26:16.957	2023-02-09 11:26:17.072	115
2	2023-02-09 11:26:38.962	2023-02-09 11:26:39.078	116
3	2023-02-09 11:26:49.964	2023-02-09 11:26:50.111	147
4	2023-02-09 11:27:00.967	2023-02-09 11:27:01.071	104
5	2023-02-09 11:27:11.969	2023-02-09 11:27:12.093	124
6	2023-02-09 11:27:22.971	2023-02-09 11:27:23.089	118
7	2023-02-09 11:27:33.973	2023-02-09 11:27:34.092	119
8	2023-02-09 11:27:44.975	2023-02-09 11:27:45.114	139
9	2023-02-09 11:27:55.977	2023-02-09 11:27:56.067	90
10	2023-02-09 11:28:17.982	2023-02-09 11:28:18.086	104
11	2023-02-09 11:28:28.984	2023-02-09 11:28:29.094	110
12	2023-02-09 11:28:50.987	2023-02-09 11:28:51.133	146
13	2023-02-09 11:29:23.994	2023-02-09 11:29:24.103	109
14	2023-02-09 11:29:34.996	2023-02-09 11:29:35.109	113

Table A.30: Remote Setup - Response Time 1 Device with 4 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 11:46:25.671	2023-02-09 11:46:25.831	160
2	2023-02-09 11:46:47.677	2023-02-09 11:46:47.853	176
3	2023-02-09 11:46:58.679	2023-02-09 11:46:58.858	179
4	2023-02-09 11:47:20.686	2023-02-09 11:47:20.853	167
5	2023-02-09 11:47:31.689	2023-02-09 11:47:31.834	145
6	2023-02-09 11:47:42.693	2023-02-09 11:47:42.846	153
7	2023-02-09 11:48:04.698	2023-02-09 11:48:04.844	146
8	2023-02-09 11:48:15.700	2023-02-09 11:48:15.835	135
9	2023-02-09 11:48:26.702	2023-02-09 11:48:26.863	161
10	2023-02-09 11:48:37.705	2023-02-09 11:48:37.851	146
11	2023-02-09 11:48:48.708	2023-02-09 11:48:48.874	166
12	2023-02-09 11:49:21.714	2023-02-09 11:49:21.855	141
13	2023-02-09 11:49:43.718	2023-02-09 11:49:43.851	133
14	2023-02-09 11:49:54.720	2023-02-09 11:49:54.864	144

Table A.31: Remote Setup - Response Time 1 Device with 5 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 13:09:23.976	2023-02-09 13:09:24.173	197
2	2023-02-09 13:09:45.986	2023-02-09 13:09:46.171	185
3	2023-02-09 13:09:56.989	2023-02-09 13:09:57.185	196
4	2023-02-09 13:10:07.991	2023-02-09 13:10:08.212	221
5	2023-02-09 13:10:29.995	2023-02-09 13:10:30.192	197
6	2023-02-09 13:10:40.997	2023-02-09 13:10:41.200	203
7	2023-02-09 13:10:52.001	2023-02-09 13:10:52.228	227
8	2023-02-09 13:11:03.005	2023-02-09 13:11:03.172	167
9	2023-02-09 13:11:14.007	2023-02-09 13:11:14.205	198
10	2023-02-09 13:11:25.009	2023-02-09 13:11:25.173	164
11	2023-02-09 13:11:36.011	2023-02-09 13:11:36.187	176
12	2023-02-09 13:11:58.015	2023-02-09 13:11:58.216	201
13	2023-02-09 13:12:09.018	2023-02-09 13:12:09.240	222
14	2023-02-09 13:12:12.024	2023-02-09 13:12:12.219	195

Table A.32: Remote Setup - Response Time 1 Device with 10 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 13:37:31.421	2023-02-09 13:37:31.739	318
2	2023-02-09 13:37:42.424	2023-02-09 13:37:42.722	298
3	2023-02-09 13:37:53.426	2023-02-09 13:37:53.717	291
4	2023-02-09 13:38:15.430	2023-02-09 13:38:15.716	286
5	2023-02-09 13:38:48.436	2023-02-09 13:38:48.765	329
6	2023-02-09 13:39:10.440	2023-02-09 13:39:10.770	330
7	2023-02-09 13:39:21.442	2023-02-09 13:39:21.738	296
8	2023-02-09 13:39:32.444	2023-02-09 13:39:32.743	299
9	2023-02-09 13:39:43.447	2023-02-09 13:39:43.687	240
10	2023-02-09 13:39:54.452	2023-02-09 13:39:54.748	296
11	2023-02-09 13:40:16.456	2023-02-09 13:40:16.720	264
12	2023-02-09 13:40:27.457	2023-02-09 13:40:27.747	290
13	2023-02-09 13:40:49.460	2023-02-09 13:40:49.730	270
14	2023-02-09 13:41:11.464	2023-02-09 13:41:11.749	285

Table A.33: Remote Setup - Response Time 1 Device with 15 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 13:58:55.227	2023-02-09 13:58:55.642	415
2	2023-02-09 13:59:17.231	2023-02-09 13:59:17.647	416
3	2023-02-09 13:59:39.236	2023-02-09 13:59:39.575	339
4	2023-02-09 13:59:50.238	2023-02-09 13:59:50.666	428
5	2023-02-09 12:00:01.240	2023-02-09 12:00:01.653	413
6	2023-02-09 12:00:12.242	2023-02-09 12:00:12.650	408
7	2023-02-09 12:00:23.244	2023-02-09 12:00:23.640	396
8	2023-02-09 12:00:34.245	2023-02-09 12:00:34.595	350
9	2023-02-09 12:00:45.247	2023-02-09 12:00:45.594	347
10	2023-02-09 12:00:56.249	2023-02-09 12:00:56.632	383
11	2023-02-09 12:01:07.253	2023-02-09 12:01:07.616	363
12	2023-02-09 12:01:18.255	2023-02-09 12:01:18.598	343
13	2023-02-09 12:01:29.256	2023-02-09 12:01:29.603	347
14	2023-02-09 12:01:40.258	2023-02-09 12:01:40.683	425

Table A.34: Remote Setup - Response Time 1 Device with 20 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 12:19:40.786	2023-02-09 12:19:41.276	490
2	2023-02-09 12:19:51.789	2023-02-09 12:19:52.294	505
3	2023-02-09 12:20:02.794	2023-02-09 12:20:03.338	544
4	2023-02-09 12:20:24.798	2023-02-09 12:20:25.312	514
5	2023-02-09 12:20:35.800	2023-02-09 12:20:36.232	432
6	2023-02-09 12:20:57.803	2023-02-09 12:20:58.390	587
7	2023-02-09 12:21:08.805	2023-02-09 12:21:09.346	541
8	2023-02-09 12:21:19.807	2023-02-09 12:21:20.344	537
9	2023-02-09 12:21:30.808	2023-02-09 12:21:31.313	505
10	2023-02-09 12:21:52.811	2023-02-09 12:21:53.236	425
11	2023-02-09 12:22:03.816	2023-02-09 12:22:04.311	495
12	2023-02-09 12:22:14.818	2023-02-09 12:22:15.258	440
13	2023-02-09 12:22:25.819	2023-02-09 12:22:26.321	502
14	2023-02-09 12:22:47.822	2023-02-09 12:22:48.211	389

### A.3.2 Flows on 4 Devices

Table A.35: Remote Setup - Response Time 4 Devices with 1 Flow

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 07:04:43.448	2023-02-09 07:04:43.506	58
2	2023-02-09 07:05:14.454	2023-02-09 07:05:14.532	78
3	2023-02-09 07:05:45.460	2023-02-09 07:05:45.545	85
4	2023-02-09 07:06:16.463	2023-02-09 07:06:16.561	98
5	2023-02-09 16:07:37.467	2023-02-09 16:07:37.539	72
6	2023-02-09 07:07:49.482	2023-02-09 07:07:49.562	80
7	2023-02-09 16:08:40.484	2023-02-09 16:08:40.572	88
8	2023-02-09 07:08:51.491	2023-02-09 07:08:51.573	82
9	2023-02-09 07:09:22.496	2023-02-09 07:09:22.555	59
10	2023-02-09 07:09:53.499	2023-02-09 07:09:53.583	84
11	2023-02-09 07:10:24.508	2023-02-09 07:10:24.622	114
12	2023-02-09 07:12:28.592	2023-02-09 07:12:28.708	116
13	2023-02-09 07:13:30.635	2023-02-09 07:13:30.727	92
14	2023-02-09 07:14:01.647	2023-02-09 07:14:01.729	82

Table A.36: Remote Setup - Response Time 4 Devices with 2 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 07:22:48.810	2023-02-09 07:22:48.913	93
2	2023-02-09 07:23:09.814	2023-02-09 07:23:09.947	123
3	2023-02-09 07:23:51.823	2023-02-09 07:23:51.931	98
4	2023-02-09 07:24:33.842	2023-02-09 07:24:33.963	111
5	2023-02-09 07:24:54.850	2023-02-09 07:24:54.957	97
6	2023-02-09 07:25:15.855	2023-02-09 07:25:15.984	119
7	2023-02-09 07:25:57.883	2023-02-09 07:25:58.001	108
8	2023-02-09 07:26:39.893	2023-02-09 07:26:39.988	85
9	2023-02-09 07:27:42.912	2023-02-09 07:27:43.00	78
10	2023-02-09 07:28:45.920	2023-02-09 07:28:46.017	87
11	2023-02-09 07:31:33.975	2023-02-09 07:31:34.071	86
12	2023-02-09 07:31:54.978	2023-02-09 07:31:55.081	93
13	2023-02-09 07:33:18.994	2023-02-09 07:33:19.101	97
14	2023-02-09 07:34:01.008	2023-02-09 07:34:01.120	102

Table A.37: Remote Setup - Response Time 4 Devices with 3 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 07:41:48.308	2023-02-09 07:41:48.464	156
2	2023-02-09 07:41:59.312	2023-02-09 07:41:59.447	135
3	2023-02-09 07:42:10.331	2023-02-09 07:42:10.460	129
4	2023-02-09 07:42:21.336	2023-02-09 07:42:21.446	110
5	2023-02-09 07:42:32.339	2023-02-09 07:42:32.472	133
6	2023-02-09 07:42:43.342	2023-02-09 07:42:43.455	113
7	2023-02-09 07:42:54.347	2023-02-09 07:42:54.459	112
8	2023-02-09 07:43:27.367	2023-02-09 07:43:27.503	136
9	2023-02-09 07:44:00.389	2023-02-09 07:44:00.544	155
10	2023-02-09 07:44:22.406	2023-02-09 07:44:22.571	165
11	2023-02-09 07:44:33.412	2023-02-09 07:44:33.553	141
12	2023-02-09 07:44:44.424	2023-02-09 07:44:44.577	153
13	2023-02-09 07:44:55.428	2023-02-09 07:44:55.569	141
14	2023-02-09 07:45:06.430	2023-02-09 07:45:06.527	97

Table A.38: Remote Setup - Response Time 4 Devices with 4 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 08:02:40.839	2023-02-09 08:02:40.993	154
2	2023-02-09 08:03:02.851	2023-02-09 08:03:03.025	174
3	2023-02-09 08:03:13.855	2023-02-09 08:03:14.028	173
4	2023-02-09 08:03:24.859	2023-02-09 08:03:25.019	150
5	2023-02-09 08:03:46.876	2023-02-09 08:03:47.027	151
6	2023-02-09 08:03:57.898	2023-02-09 08:03:58.057	159
7	2023-02-09 08:04:30.910	2023-02-09 08:04:31.095	185
8	2023-02-09 08:05:03.937	2023-02-09 08:05:04.102	165
9	2023-02-09 08:05:14.943	2023-02-09 08:05:17.122	179
10	2023-02-09 08:05:25.948	2023-02-09 08:05:26.090	142
11	2023-02-09 08:05:36.952	2023-02-09 08:05:37.104	162
12	2023-02-09 08:05:47.957	2023-02-09 08:05:48.105	148
13	2023-02-09 08:06:09.967	2023-02-09 08:06:10.102	135
14	2023-02-09 08:06:20.970	2023-02-09 08:06:21.104	134

Table A.39: Remote Setup - Response Time 4 Devices with 5 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 08:23:02.050	2023-02-09 08:23:02.263	213
2	2023-02-09 08:23:35.089	2023-02-09 08:23:35.271	182
3	2023-02-09 08:23:46.095	2023-02-09 08:23:46.326	231
4	2023-02-09 08:23:57.115	2023-02-09 08:23:57.322	207
5	2023-02-09 08:24:08.118	2023-02-09 08:24:08.321	203
6	2023-02-09 08:24:19.123	2023-02-09 08:24:19.368	245
7	2023-02-09 08:24:30.127	2023-02-09 08:24:30.352	225
8	2023-02-09 08:24:41.132	2023-02-09 08:24:41.345	213
9	2023-02-09 08:24:52.147	2023-02-09 08:24:52.342	195
10	2023-02-09 08:25:14.153	2023-02-09 08:25:14.357	204
11	2023-02-09 08:25:25.157	2023-02-09 08:25:25.389	232
12	2023-02-09 08:25:36.161	2023-02-09 08:25:36.365	204
13	2023-02-09 08:25:58.180	2023-02-09 08:25:58.415	235
14	2023-02-09 08:26:09.182	2023-02-09 08:26:09.360	178

Table A.40: Remote Setup - Response Time 4 Devices with 10 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 08:37:56.988	2023-02-09 08:37:57.261	273
2	2023-02-09 08:38:52.025	2023-02-09 08:38:52.314	289
3	2023-02-09 08:39:36.034	2023-02-09 08:39:36.234	200
4	2023-02-09 08:39:47.036	2023-02-09 08:39:47.267	231
5	2023-02-09 08:39:58.040	2023-02-09 08:39:58.255	215
6	2023-02-09 08:40:09.042	2023-02-09 08:40:09.247	205
7	2023-02-09 08:40:20.045	2023-02-09 08:40:20.262	217
8	2023-02-09 08:40:31.047	2023-02-09 08:40:31.294	247
9	2023-02-09 08:40:42.049	2023-02-09 08:40:42.283	234
10	2023-02-09 08:40:53.052	2023-02-09 08:40:53.314	262
11	2023-02-09 08:41:04.054	2023-02-09 08:41:04.273	219
12	2023-02-09 08:41:15.056	2023-02-09 08:41:15.266	210
13	2023-02-09 08:41:26.058	2023-02-09 08:41:26.294	236
14	2023-02-09 08:41:37.061	2023-02-09 08:41:37.312	251

Table A.41: Remote Setup - Response Time 4 Devices with 15 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 09:04:34.352	2023-02-09 09:04:35.865	513
2	2023-02-09 09:04:56.963	2023-02-09 09:04:57.472	509
3	2023-02-09 09:05:18.992	2023-02-09 09:05:19.523	531
4	2023-02-09 09:05:52.008	2023-02-09 09:05:52.470	462
5	2023-02-09 09:06:03.010	2023-02-09 09:06:03.548	538
6	2023-02-09 09:06:14.018	2023-02-09 09:06:14.505	442
7	2023-02-09 09:06:25.020	2023-02-09 09:06:25.567	547
8	2023-02-09 09:06:36.026	2023-02-09 09:06:36.562	536
9	2023-02-09 09:06:47.033	2023-02-09 09:06:47.590	557
10	2023-02-09 09:07:09.041	2023-02-09 09:07:09.599	558
11	2023-02-09 09:07:20.044	2023-02-09 09:07:20.550	506
12	2023-02-09 09:07:31.047	2023-02-09 09:07:31.515	468
13	2023-02-09 09:07:42.051	2023-02-09 09:07:42.599	548
14	2023-02-09 09:07:53.054	2023-02-09 09:07:53.500	446

Table A.42: Remote Setup - Response Time 4 Devices with 20 Flows

<b>n</b>	<b>Start time</b>	<b>Stop time</b>	<b>Response Time (ms)</b>
1	2023-02-09 09:15:44.128	2023-02-09 09:15:44.751	623
2	2023-02-09 09:15:55.251	2023-02-09 09:15:55.889	638
3	2023-02-09 09:07:42.287	2023-02-09 09:07:42.7816	529
4	2023-02-09 09:17:19.290	2023-02-09 09:17:19.920	630
5	2023-02-09 09:08:10.299	2023-02-09 09:08:10.919	620
6	2023-02-09 09:17:47.302	2023-02-09 09:17:47.897	595
7	2023-02-09 09:08:24.304	2023-02-09 09:08:24.846	542
8	2023-02-09 09:18:05.307	2023-02-09 09:18:05.848	541
9	2023-02-09 09:08:36.312	2023-02-09 09:08:36.880	568
10	2023-02-09 09:08:47.314	2023-02-09 09:08:47.718	504
11	2023-02-09 09:18:38.320	2023-02-09 09:18:38.840	520
12	2023-02-09 09:18:49.322	2023-02-09 09:18:49.832	510
13	2023-02-09 09:19:00.324	2023-02-09 09:19:00.855	531
14	2023-02-09 09:19:11.329	2023-02-09 09:19:11.881	552



## Appendix B

# Traffic Capture

New traffic in log\_tcp\_complete.

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 42899 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 912 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 39685 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 36863 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 916 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 924 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 49769 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 804 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 42037 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 56457 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 809 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 807 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 109.49.147.28, port: 49872 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 55993 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 836 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 36797 is detected as NORMAL

Traffic from ip: 192.168.254.254, port: 59401 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 840 is detected as NORMAL

Traffic from ip: 192.168.254.254, port: 869 is detected as NORMAL

7 Feb 14:55:27 - [info] [debug:d49b538b.f0f42]

Traffic from ip: 192.168.254.254, port: 54295 is detected as NORMAL

# Bibliography

- [1] A. Hristoskova, N. González-Deleito, S. Klein, *et al.*, “An initial analysis of the shortcomings of conventional AI and the benefits of distributed AI approaches in industrial use cases”, in *Artificial Intelligence Applications and Innovations. AIAI 2021 IFIP WG 12.5 International Workshops*, I. Maglogiannis, J. Macintyre, and L. Iliadis, Eds., vol. 628, Series Title: IFIP Advances in Information and Communication Technology, Cham: Springer International Publishing, 2021, pp. 281–292, ISBN: 978-3-030-79156-8 978-3-030-79157-5. DOI: [10.1007/978-3-030-79157-5\\_23](https://doi.org/10.1007/978-3-030-79157-5_23). [Online]. Available: [https://link.springer.com/10.1007/978-3-030-79157-5\\_23](https://link.springer.com/10.1007/978-3-030-79157-5_23) (visited on 01/03/2022).
- [2] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, “A systematic literature review on hardware implementation of artificial intelligence algorithms”, en, *J Supercomput*, vol. 77, no. 2, pp. 1897–1938, Feb. 2021, ISSN: 0920-8542, 1573-0484. DOI: [10.1007/s11227-020-03325-8](https://doi.org/10.1007/s11227-020-03325-8). [Online]. Available: <https://link.springer.com/10.1007/s11227-020-03325-8> (visited on 02/09/2021).
- [3] R. Venanzi, F. Montori, P. Bellavista, and L. Foschini, “Industry 4.0 Solutions for Interoperability: A Use Case about Tools and Tool Chains in the Arrowhead Tools Project”, en, in *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*, Bologna, Italy: IEEE, Sep. 2020, pp. 429–433, ISBN: 978-1-72816-997-2. DOI: [10.1109/SMARTCOMP50058.2020.00089](https://doi.org/10.1109/SMARTCOMP50058.2020.00089). [Online]. Available: <https://ieeexplore.ieee.org/document/9239681/> (visited on 01/04/2022).
- [4] N. Mohamed and J. Al-Jaroodi, “Applying Blockchain in Industry 4.0 Applications”, en, in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA: IEEE, Jan. 2019, pp. 0852–0858, ISBN: 978-1-72810-554-3. DOI: [10.1109/CCWC.2019.8666558](https://doi.org/10.1109/CCWC.2019.8666558). [Online]. Available: <https://ieeexplore.ieee.org/document/8666558/> (visited on 02/18/2022).
- [5] M. Batty, “Digital twins”, en, *Environment and Planning B: Urban Analytics and City Science*, vol. 45, no. 5, pp. 817–820, Sep. 2018, ISSN: 2399-8083, 2399-8091. DOI: [10.1177/2399808318796416](https://doi.org/10.1177/2399808318796416). [Online]. Available: <http://journals.sagepub.com/doi/10.1177/2399808318796416> (visited on 02/16/2022).
- [6] C. Queiroz, A. Mahmood, and Z. Tari, “SCADASim—A Framework for Building SCADA Simulations”, en, *IEEE Trans. Smart Grid*, vol. 2, no. 4, pp. 589–597, Dec. 2011, ISSN:

- 1949-3053. DOI: [10.1109/TSG.2011.2162432](https://doi.org/10.1109/TSG.2011.2162432). [Online]. Available: <http://ieeexplore.ieee.org/document/6009221/> (visited on 01/30/2023).
- [7] A. Daneels and W. Salter, *Mc1i01.pdf*, en, 1999. (visited on 01/30/2023).
- [8] C. Paniagua and J. Delsing, “Industrial Frameworks for Internet of Things: A Survey”, en, *IEEE Systems Journal*, vol. 15, no. 1, pp. 1149–1159, Mar. 2021, ISSN: 1932-8184, 1937-9234, 2373-7816. DOI: [10.1109/JSYST.2020.2993323](https://doi.org/10.1109/JSYST.2020.2993323). [Online]. Available: <https://ieeexplore.ieee.org/document/9099983/> (visited on 01/04/2022).
- [9] D. Kozma, P. Varga, and F. Larrinaga, “Data-driven Workflow Management by utilising BPMN and CPN in IIoT Systems with the Arrowhead Framework”, en, in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain: IEEE, Sep. 2019, pp. 385–392, ISBN: 978-1-72810-303-7. DOI: [10.1109/ETFA.2019.8869501](https://doi.org/10.1109/ETFA.2019.8869501). [Online]. Available: <https://ieeexplore.ieee.org/document/8869501/> (visited on 02/04/2022).
- [10] S. Cheshire and M. Krochmal, “DNS-Based Service Discovery”, en, RFC Editor, Tech. Rep. RFC6763, Feb. 2013, RFC6763. DOI: [10.17487/rfc6763](https://doi.org/10.17487/rfc6763). [Online]. Available: <https://www.rfc-editor.org/info/rfc6763> (visited on 02/04/2022).
- [11] C. Hegedus, P. Varga, and A. Franko, “Secure and trusted inter-cloud communications in the arrowhead framework”, en, in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, St. Petersburg: IEEE, May 2018, pp. 755–760, ISBN: 978-1-5386-6531-2. DOI: [10.1109/ICPHYS.2018.8390802](https://doi.org/10.1109/ICPHYS.2018.8390802). [Online]. Available: <https://ieeexplore.ieee.org/document/8390802/> (visited on 02/04/2022).
- [12] P. Varga, D. Kozma, and C. Hegedus, “Data-Driven Workflow Execution in Service Oriented IoT Architectures”, en, in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin: IEEE, Sep. 2018, pp. 203–210, ISBN: 978-1-5386-7108-5. DOI: [10.1109/ETFA.2018.8502665](https://doi.org/10.1109/ETFA.2018.8502665). [Online]. Available: <https://ieeexplore.ieee.org/document/8502665/> (visited on 02/04/2022).
- [13] O. Carlsson, D. Vera, J. Delsing, B. Ahmad, and R. Harrison, “Plant descriptions for engineering tool interoperability”, en, in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, Poitiers, France: IEEE, Jul. 2016, pp. 730–735, ISBN: 978-1-5090-2870-2. DOI: [10.1109/INDIN.2016.7819255](https://doi.org/10.1109/INDIN.2016.7819255). [Online]. Available: <http://ieeexplore.ieee.org/document/7819255/> (visited on 02/04/2022).
- [14] *AUTOSAR classic platform*, <https://www.autosar.org/standards/classic-platform/>, Accessed: 2022-01-10.
- [15] *AUTOSAR adaptive platform*, <https://www.autosar.org/standards/adaptive-platform/>, Accessed: 2022-01-10.
- [16] *BaSys 4.0*, <https://www.basys40.de/>, Accessed: 2022-01-14.
- [17] *BaSyx*, <https://wiki.eclipse.org/BaSyx>, Accessed: 2022-01-14.

- [18] J. Plattner and J. Oberzaucher, “A MULTI-SENSORY APPROACH TO ACQUIRE AND PROCESS HEALTH AND LIFESTYLE INFORMATION”, en, p. 7, 2019.
- [19] FIWARE, <https://www.fiware.org/>, Accessed: 2022-01-16.
- [20] N. Kumar and D. P. Vidyarthi, “A Green Routing Algorithm for IoT-Enabled Software Defined Wireless Sensor Network”, en, *IEEE Sensors J.*, vol. 18, no. 22, pp. 9449–9460, Nov. 2018, ISSN: 1530-437X, 1558-1748, 2379-9153. DOI: [10.1109/JSEN.2018.2869629](https://doi.org/10.1109/JSEN.2018.2869629). [Online]. Available: <https://ieeexplore.ieee.org/document/8458421/> (visited on 02/16/2022).
- [21] L. Dantas, E. Cavalcante, and T. Batista, “A Development Environment for FIWARE-based Internet of Things Applications”, en, in *Proceedings of the 6th International Workshop on Middleware and Applications for the Internet of Things - M4IoT '19*, Davis, CA, USA: ACM Press, 2019, pp. 21–26, ISBN: 978-1-4503-7028-8. DOI: [10.1145/3366610.3368100](https://doi.org/10.1145/3366610.3368100). [Online]. Available: <http://dl.acm.org/citation.cfm?doid=3366610.3368100> (visited on 02/16/2022).
- [22] K. Ferencz and J. Domokos, “Using Node-RED platform in an industrial environment”, en,
- [23] M. Lekic and G. Gardasevic, “IoT sensor integration to Node-RED platform”, en, in *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, East Sarajevo: IEEE, Mar. 2018, pp. 1–5, ISBN: 978-1-5386-4907-7. DOI: [10.1109/INFOTEH.2018.8345544](https://doi.org/10.1109/INFOTEH.2018.8345544). [Online]. Available: <https://ieeexplore.ieee.org/document/8345544/> (visited on 01/17/2023).
- [24] M. M. Ahmadpanah, M. Balliu, D. Hedin, L. E. Olsson, and A. Sabelfeld, “Securing Node-RED Applications”, en, in *Protocols, Strands, and Logic*, D. Dougherty, J. Meseguer, S. A. Mödersheim, and P. Rowe, Eds., vol. 13066, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2021, pp. 1–21, ISBN: 978-3-030-91630-5 978-3-030-91631-2. DOI: [10.1007/978-3-030-91631-2\\_1](https://doi.org/10.1007/978-3-030-91631-2_1). [Online]. Available: [https://link.springer.com/10.1007/978-3-030-91631-2\\_1](https://link.springer.com/10.1007/978-3-030-91631-2_1) (visited on 01/17/2023).
- [25] *Node-Red: Securing node-red*. <https://nodered.org/docs/user-guide/runtime/securing-node-red/>, Accessed: 2023-01-17.
- [26] L. Larsson, W. Tärneberg, C. Klein, E. Elmroth, and M. Kihl, “Impact of etcd deployment on Kubernetes, Istio, and application performance”, en, *Softw Pract Exper*, vol. 50, no. 10, pp. 1986–2007, Oct. 2020, ISSN: 0038-0644, 1097-024X. DOI: [10.1002/spe.2885](https://doi.org/10.1002/spe.2885). [Online]. Available: <https://onlinelibrary.wiley.com/doi/10.1002/spe.2885> (visited on 01/19/2023).
- [27] N. Marathe, A. Gandhi, and J. M. Shah, “Docker Swarm and Kubernetes in Cloud Computing Environment”, en, in *2019 3rd International Conference on Trends in Electronics*

- and Informatics (ICOEI)*, Tirunelveli, India: IEEE, Apr. 2019, pp. 179–184, ISBN: 978-1-5386-9439-8. DOI: [10.1109/ICOEI.2019.8862654](https://doi.org/10.1109/ICOEI.2019.8862654). [Online]. Available: <https://ieeexplore.ieee.org/document/8862654/> (visited on 01/19/2023).
- [28] P. Martin, “Kubernetes Resources”, in *Kubernetes: Preparing for the CKA and CKAD Certifications*, Berkeley, CA: Apress, 2021, pp. 19–22, ISBN: 978-1-4842-6494-2. DOI: [10.1007/978-1-4842-6494-2\\_4](https://doi.org/10.1007/978-1-4842-6494-2_4). [Online]. Available: [https://doi.org/10.1007/978-1-4842-6494-2\\_4](https://doi.org/10.1007/978-1-4842-6494-2_4).
- [29] S. M. Tahsien, H. Karimipour, and P. Spachos, “Machine learning based solutions for security of Internet of Things (IoT): A survey”, en, *Journal of Network and Computer Applications*, vol. 161, p. 102 630, Jul. 2020, ISSN: 10848045. DOI: [10.1016/j.jnca.2020.102630](https://doi.org/10.1016/j.jnca.2020.102630). [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S1084804520301041> (visited on 01/05/2022).
- [30] D. Xin, L. Ma, S. Song, and A. Parameswaran, “How Developers Iterate on Machine Learning Workflows – A Survey of the Applied Machine Learning Literature”, en, *arXiv:1803.10311 [cs, stat]*, May 2018, arXiv: 1803.10311. [Online]. Available: <http://arxiv.org/abs/1803.10311> (visited on 02/17/2022).
- [31] S. Chibani and F.-X. Coudert, “Machine learning approaches for the prediction of materials properties”, en, *APL Materials*, vol. 8, no. 8, p. 080 701, Aug. 2020, ISSN: 2166-532X. DOI: [10.1063/5.0018384](https://doi.org/10.1063/5.0018384). [Online]. Available: <http://aip.scitation.org/doi/10.1063/5.0018384> (visited on 02/17/2022).
- [32] M. Pérez-Ortiz, S. Jiménez-Fernández, P. Gutiérrez, E. Alexandre, C. Hervás-Martínez, and S. Salcedo-Sanz, “A Review of Classification Problems and Algorithms in Renewable Energy Applications”, en, *Energies*, vol. 9, no. 8, p. 607, Aug. 2016, ISSN: 1996-1073. DOI: [10.3390/en9080607](https://doi.org/10.3390/en9080607). [Online]. Available: <http://www.mdpi.com/1996-1073/9/8/607> (visited on 02/17/2022).
- [33] R. Doshi, N. Apthorpe, and N. Feamster, “Machine Learning DDoS Detection for Consumer Internet of Things Devices”, en, in *2018 IEEE Security and Privacy Workshops (SPW)*, San Francisco, CA: IEEE, May 2018, pp. 29–35, ISBN: 978-1-5386-8276-0. DOI: [10.1109/SPW.2018.00013](https://doi.org/10.1109/SPW.2018.00013). [Online]. Available: <https://ieeexplore.ieee.org/document/8424629/> (visited on 02/17/2022).
- [34] S. K. Maurya and A. Choudhary, “Deep Learning based Vulnerable Road User Detection and Collision Avoidance”, en, in *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Madrid: IEEE, Sep. 2018, pp. 1–6, ISBN: 978-1-5386-3543-8. DOI: [10.1109/ICVES.2018.8519504](https://doi.org/10.1109/ICVES.2018.8519504). [Online]. Available: <https://ieeexplore.ieee.org/document/8519504/> (visited on 02/17/2022).
- [35] Y. Liu, X. Ma, Y. Li, Y. Tie, Y. Zhang, and J. Gao, “Water Pipeline Leakage Detection Based on Machine Learning and Wireless Sensor Networks”, en, *Sensors*, vol. 19, no. 23,

- p. 5086, Nov. 2019, ISSN: 1424-8220. DOI: [10.3390/s19235086](https://doi.org/10.3390/s19235086). [Online]. Available: <https://www.mdpi.com/1424-8220/19/23/5086> (visited on 02/17/2022).
- [36] B. J. Jansen, “The graphical user interface”, en, (visited on 01/29/2023).
- [37] Ž. Jovanović, D. Jagodić, and D. Vujičić, “JAVA SPRING BOOT REST WEB SERVICE INTEGRATION WITH JAVA ARTIFICIAL INTELLIGENCE WEKA FRAMEWORK”, en, *International Scientific Conference*, 2017. (visited on 01/26/2023).
- [38] E. Universitat Politècnica de València, “Universitat Politècnica de València”, en, *ing.agua*, vol. 18, no. 1, p. ix, Sep. 2014, ISSN: 1886-4996, 1134-2196. DOI: [10.4995/ia.2014.3293](https://doi.org/10.4995/ia.2014.3293). [Online]. Available: <http://polipapers.upv.es/index.php/IA/article/view/3293> (visited on 01/30/2023).
- [39] B. Bulut, H. Burak Ketmen, A. S. Atalay, O. Herkiloglu, and R. Salokangas, “An Arrowhead and Mimosa Based IoT Framework with an Industrial Predictive Maintenance Application”, en, in *2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, Kocaeli, Turkey: IEEE, Aug. 2021, pp. 1–5, ISBN: 978-1-66543-603-8. DOI: [10.1109/INISTA52262.2021.9548127](https://doi.org/10.1109/INISTA52262.2021.9548127). [Online]. Available: <https://ieeexplore.ieee.org/document/9548127/> (visited on 02/18/2022).
- [40] B. Peceli, G. Singler, Z. Theisz, C. Hegedus, P. Vargaz, and Z. Szepessy, “Integrating an Electric Vehicle Supply Equipment with the Arrowhead framework”, en, in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Florence, Italy: IEEE, Oct. 2016, pp. 5271–5276, ISBN: 978-1-5090-3474-1. DOI: [10.1109/IECON.2016.7793603](https://doi.org/10.1109/IECON.2016.7793603). [Online]. Available: <http://ieeexplore.ieee.org/document/7793603/> (visited on 02/20/2022).
- [41] J. Jokinen, T. Latvala, and J. L. Martinez Lastra, “Integrating smart city services using Arrowhead framework”, en, in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Florence: IEEE, Oct. 2016, pp. 5568–5573, ISBN: 978-1-5090-3474-1. DOI: [10.1109/IECON.2016.7793708](https://doi.org/10.1109/IECON.2016.7793708). [Online]. Available: <https://ieeexplore.ieee.org/document/7793708/> (visited on 02/21/2022).
- [42] N. Schumacher, “Cloud/Edge Machine Learning for Privacy-Preserving Network Trace Profiling”, en,