FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Automatic Discovery And Capability For IIoT Sensors

Miguel Cardoso Ribeiro Gouveia



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Ana Cristina Costa Aguiar

October 25, 2026

© Miguel Cardoso Ribeiro Gouveia, 2026

Automatic Discovery And Capability For IIoT Sensors

Miguel Cardoso Ribeiro Gouveia

Mestrado em Engenharia Eletrotécnica e de Computadores

October 25, 2026

Resumo

A *Industrial Internet of Things* (IIoT) recorre cada vez mais a redes de sensores e nós de processamento para suportar a automação e a tomada de decisões em ambientes industriais. À medida que estas redes crescem, a configuração manual dos dispositivos torna-se um obstáculo que limita a escalabilidade, aumenta o risco de erro humano e atrasa a resposta operacional à mudança. Esta dissertação, realizada no âmbito do projeto Sensitive Industry em parceria com a Bosch, aborda este problema no contexto da Digital MilkRun Application na fábrica da Bosch em Ovar, uma aplicação que utiliza sensores de câmara e nós de processamento dedicados para rastrear e otimizar o transporte intrafabril, mas que, antes deste trabalho, dependia inteiramente de configuração manual da rede, sem qualquer infraestrutura de gestão ou monitorização de suporte.

Este trabalho propõe e implementa um sistema autónomo de gestão e monitorização de rede que automatiza o *on-boarding* de sensores de câmara e nós de processamento. A solução centraliza os serviços de DNS, DHCP e Gestão de Endereços IP (DDI), juntamente com a monitorização e o controlo de acessos, num único Raspberry Pi. Os *hook points* do Kea DHCP são utilizados para descoberta de dispositivos a ingressar na rede, os quais são depois sondados para determinar a sua capacidade — sensor de câmara ou nó de processamento — e configurados automaticamente em conformidade: os sensores de câmara têm a sua palavra-passe de acesso definida através da respetiva API web, enquanto os nós de processamento têm o seu servidor SSH e o agente de monitorização Zabbix configurados através de *playbooks* Ansible. O controlo de acesso aos nós de processamento é garantido através de uma autoridade certificadora que assina as chaves públicas dos utilizadores, restringindo o acesso SSH a utilizadores autorizados. Os dispositivos recém-descobertos ou desligados, juntamente com as suas capacidades, são comunicados a um orquestrador externo via MQTT.

O sistema foi implementado num ambiente laboratorial que reproduz a topologia de rede da Digital MilkRun Application e validado através de testes funcionais dos fluxos de trabalho de *on-boarding* dos sensores de câmara e dos nós de processamento. Os resultados confirmam que ambos os tipos de dispositivos podem ser automaticamente detetados, configurados e reportados sem intervenção manual para além da sua ligação física e da reserva de IP, demonstrando a auto-configuração como uma propriedade autónoma do sistema. Foram identificadas limitações relativamente à latência de sincronização, ao tempo de deteção de dispositivos inativos e à complexidade da automação da palavra-passe da câmara, as quais são discutidas juntamente com possíveis direções para trabalho futuro.

Palavras-chave: IIoT • rede autonómica • gestão de rede • monitorização de rede • automatização de rede • descoberta de dispositivos • auto-configuração • controlo de acesso

Abstract

The Industrial Internet of Things (IIoT) increasingly relies on networks of sensors and processing nodes to support automation and decision-making in industrial environments. As these networks grow, the manual configuration of devices becomes a bottleneck that limits scalability, increases the risk of human error, and slows down operational response to change. This dissertation, carried out within the Sensitive Industry project in partnership with Bosch, addresses this problem in the context of the Digital MilkRun Application at the Bosch Ovar factory, an application that uses camera sensors and dedicated processing nodes to track and optimize intra-factory transportation, but which, prior to this work, relied entirely on manual network configuration with no supporting management or monitoring infrastructure.

This work proposes and implements an autonomic network management and monitoring system that automates the on-boarding of camera sensors and processing nodes. The solution centralizes DNS, DHCP and IP Address Management (DDI) services, alongside monitoring and access control, on a single Raspberry Pi. Kea DHCP hook points are used for device discovery of devices joining the network, which are then probed to determine their capability - camera sensor or processing node - and automatically configured accordingly: camera sensors have their access password set through their web API, while processing nodes have their SSH server and Zabbix monitoring agent configured through Ansible playbooks. Access control to processing nodes is enforced through a certificate authority that signs user public keys, restricting SSH access to authorized users. Newly discovered or disconnected devices, along with their capabilities, are communicated to an external orchestrator via MQTT.

The system was deployed in a laboratory environment mirroring the Digital MilkRun Application's network topology and validated through functional testing of the camera sensor and processing node on-boarding work-flows. Results confirm that both device types can be automatically detected, configured, and reported without manual intervention beyond their physical connection and IP reservation, demonstrating self-configuration as an autonomic property of the system. Limitations were identified regarding synchronization latency, host-down detection time, and the complexity of the camera password automation, which are discussed together with directions for future work.

Keywords: IIoT • autonomic network • network management • network monitoring • network automation • device discovery • self-configuration • access control

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a better and more sustainable future for all. It includes 17 goals to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

The specific Sustainable Development Goals mentioned have the following names:

SDG 8 Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all

SGD	Target	Contribution	Performance Indicators and Metrics
8	8.2	Achieve higher levels of economic productivity through diversification, technological upgrading and innovation, including through a focus on high-value added and labour-intensive sectors	Annual growth rate of real GDP per employed person

Acknowledgements

I want to give a very special dedication to my mother, Sónia da Costa Reis Cardoso. She couldn't be present at the defense of my Bachelor's final project dissertation and won't be able to attend the defense of this Master's dissertation, but was and always will be with me every step of the way. I also want to thank my dad, Feliz Alberto Ribeiro Gouveia, and brother, Martim Cardoso Ribeiro Gouveia, for providing me an environment that helped me work on my dissertation.

I would like to thank my supervisor Prof. Ana Cristina Costa Aguiar for her support and guidance in this work. Additionally I want to acknowledge the collaboration and contributions to the Sensitive Industry project from fellow student colleague Guilherme Coelho and professor André Restivo.

A huge thank you to everyone I trained Judo and went to the gym with for allowing me an opportunity to unwind from the pressure and stress that comes with this final stage of the Master's course. I want to thank all my friends and extended family who directly or indirectly accompanied and supported my journey developing and writing for this project.

Miguel Cardoso Ribeiro Gouveia

Declaration of honour

I declare that this dissertation is my original work and has not previously been submitted or assessed in any other course or curricular unit, whether at this or any other institution. References to other authors (statements, ideas, thoughts), as well as the use of published works fully complies with the rules of attribution and are appropriately cited in the text and bibliography, in accordance with reference standards. I am aware that plagiarism and self-plagiarism constitute academic misconduct.
I further declare that I have not used Artificial Intelligence tools to produce any part(s) of this dissertation or, when such tools were used, their use and results are clearly indicated and were carefully reviewed for the detection of errors, inaccuracies, unfounded attributions, or other forms of content and argument bias, as well as for the determination of authorship.
I further declare that I have included studies carried out with multiple co-authors, having specified, with transparency and rigour, the contribution of each co-author, provided information on the possible existence of any other dissertation in which the same article has been included, reproduced the work in full with the permission of the journal in which it was published, and obtained the explicit written consent of all co-authors for the inclusion of the article.

Miguel Cardoso Ribeiro Gouveia



"No man has the right to be an amateur in the matter of physical training. It is a shame for a man to grow old without seeing the beauty and strength of which his body is capable"

Socrates

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 2
- 1.3 Objectives . . . 2
- 1.4 Contributions . . . 3
- 1.5 Document Structure . . . 3

2 Technical Background **4**
- 2.1 Introduction . . . 4
- 2.2 Industrial Internet of Things . . . 4
  - 2.2.1 Principles . . . 5
  - 2.2.2 Challenges . . . 6
- 2.3 Autonomic Networks . . . 7
  - 2.3.1 Maturity Levels . . . 8
- 2.4 Network Automation . . . 9
- 2.5 Script-driven Automation Software Solutions . . . 10
  - 2.5.1 Ansible . . . 10
  - 2.5.2 Salt . . . 11
- 2.6 Network Management: DDI . . . 11
  - 2.6.1 DNS . . . 12
  - 2.6.2 DHCP . . . 13
  - 2.6.3 dnsmasq . . . 14
  - 2.6.4 IPAM . . . 15
- 2.7 Network Monitoring . . . 17
  - 2.7.1 Nagios . . . 18
  - 2.7.2 Zabbix . . . 19
- 2.8 Security Management . . . 20
- 2.9 ONVIF . . . 21
  - 2.9.1 Overview . . . 21
  - 2.9.2 WS Discovery . . . 22
  - 2.9.3 Profile S . . . 23

3 Solution Design **24**
- 3.1 Introduction . . . 24
- 3.2 Digital MilkRun Application . . . 24
  - 3.2.1 System Architecture . . . 24
  - 3.2.2 Camera Sensor On-Boarding . . . 26
- 3.3 Constraints of Digital MilkRun Application . . . 27

3.4 Solution Proposal . . . 28
3.5 System Requirements . . . 29
3.6 Hardware . . . 30
3.7 Software . . . 31

4 Implementation **33**
4.1 Introduction . . . 33
4.2 System Architecture . . . 33
4.3 Access Control . . . 34
4.4 Automation . . . 35
4.4.1 Synchronization of Databases . . . 37
4.4.2 Host Discovery, Probing and Expiration . . . 37
4.4.3 MQTT Broker and Message Format . . . 41
4.4.4 Ansible and Processing Node Configuration . . . 42
4.4.5 Camera Configuration . . . 44
4.5 Host On-Boarding Work-flow . . . 45

5 Results and Discussion **48**
5.1 Introduction . . . 48
5.2 Laboratory Set up . . . 48
5.3 Functional Testing . . . 51
5.4 Kea Hooks log . . . 52
5.4.1 FT-NHOB-001 . . . 52
5.4.2 FT-NHOB-002 . . . 53
5.5 Known Hosts csv file Changes . . . 54
5.5.1 FT-NHOB-001 . . . 54
5.5.2 FT-NHOB-002 . . . 55
5.6 MQTT Message log . . . 55
5.6.1 FT-NHOB-001 . . . 55
5.6.2 FT-NHOB-002 . . . 56
5.7 Ansible Inventory Changes . . . 56
5.8 System Evaluation . . . 57

6 Conclusions **59**
6.1 Future Work . . . 60

References **62**

A System Requirements **65**

B Functional Tests **67**

List of Figures

2.1 IIoT Reference Architecture [10] 5
2.2 Detailed MAPE-K model for the ACL [5] 8

3.1 Digital MilkRun System Architecture 25
3.2 Camera On-Boarding Activity Diagram 26

4.1 DDI Raspberry Pi services 34
4.2 Access Control Sequence Diagram 35
4.3 Activity Diagram for synchronizing services databases 37
4.4 Activity Diagram for Kea Hooks script 38
4.5 Activity Diagram for New Device Probing script 39
4.6 Activity Diagram for Expired Device script 40
4.7 Activity Diagram for Ansible Playbook of SSH server set up 43
4.8 Activity Diagram for Ansible Playbook of Zabbix Agent set up 44
4.9 Activity Diagram for Camera Password Setup 45
4.10 Sequence Diagram for implemented camera on-boarding work-flow 46
4.11 Sequence Diagram for implemented processing node on-boarding work-flow . . 47

5.1 Laboratory Network Set Up with two plug and play switches 49
5.2 Laboratory Network Set Up with PoE Smart Switch 50
5.3 Isolated Laboratory Network Set Up 51

List of Tables

2.1 Maturity Levels for IoT System 9

A.1 System Requirements 66

B.1 Functional Test Case for Camera On-boarding 68
B.2 Functional Test Case for Processing Node On-boarding 69

List of Acronyms

ACL	Autonomic Control Loop
ACLs	Access Control Lists
AJAX	Asynchronous Javascript and XML
API	Application Programming Interface
BOOTP	Bootstrap Protocol
CA	Certificate Authority
CLI	Command-line Interface
CMS	Communication Management System
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
CV	Computer Vision
DDI	DNS, DHCP and IPAM
DHCP	Dynamic Host Configuration Protocol
DMS	Device Management System
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
FCAPS	Fault, Configuration, Accounting, Performance, Security
FOV	Field of View
GDPR	General Data Protection Regulation
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IIoT	Industrial Internet of Things
IP	Internet Protocol
IPAM	Internet Protocol Address Management
IPFIX	Internet Protocol Flow Information Export
ISC	Internet Systems Consortium
KMS	Knowledge Base Management System
MAC	Medium Access Control
MQTT	Message Queuing Telemetry Transport
NSD	Name Server Daemon
NUC	Next Unit of Computing

ONVIF	Open Network Video Interface Forum
OS	Operating System
OSPF	Open Shortest Path First
PHP	Hypertext Preprocessor
PKI	Public Key Infrastructure
PoE	Power over Ethernet
PTZ	Pan-Tilt-Zoom
PXE	Preboot eXecution Environment
RBAC	Role-Based Access Control
RCP	Remote Control Protocol
RDNSS	Recursive DNS Server
REST	Representational State Transfer
RPi	Raspberry Pi
RTSP	Real-Time Streaming Protocol
SBC	Single Board Computer
SDG	Sustainable Development Goals
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SO	Smart Object
SOAP	Simple Object Access Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TMS	Task Management System
VLAN	Virtual Local Area Network
VMS	Video Management Software
VPN	Virtual Private Network
WS	Web Services
WSDL	Web Sevice Description Language
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Context

The evolution of the Industrial Internet of Things (IIoT) has revolutionized industrial operations by embedding intelligence into processes, enabling unprecedented levels of automation and efficiency [1]. Among these advancements, the use of networks of sensors has emerged as a key enabler for optimizing complex industrial workflows. In particular, networks of camera sensors hold immense potential for addressing challenges in transportation logistics within industrial environments, such as the movement of forklifts and carriages from warehouses to production lines and their parking [2].

Transportation within industrial facilities is a critical operational component that directly influences productivity, cost-efficiency, and workflow continuity. Traditional methods for managing intra-facility transportation often rely on predefined routes, manual oversight, or basic automation that lacks adaptability to dynamic conditions. This rigidity can lead to inefficiencies such as congestion, delays, and suboptimal utilization of resources. In response to these challenges, Internet of Things (IoT) based Warehouse Management is adopted.[3].

In an IIoT network, the scalability is a critical factor [4]. On-boarding of sensors and processing nodes in the network can be a bottleneck and their configuration a time consuming task impacting the operation of the facilities. An autonomic network incorporates self-configuring, self-healing, and self-optimizing capabilities to ensure the seamless integration and continued functionality of its components, in this case camera sensors and processing nodes [5] [6].

When camera sensors or processing nodes are added, removed or replaced, the autonomic system must automatically detect, configure, and integrate them into the operational framework without manual intervention. This capability is important for maintaining robustness and scalability in dynamic industrial environments where changes to the sensor infrastructure are inevitable. For

an autonomic network with those objectives to operate efficiently and securely there must exist a network management and monitoring infrastructure adapted to the industrial environment [7].

1.2 Motivation

The Sensitive Industry project, done in conjunction with Bosch and FEUP, more specifically Instituto de Telecomunicações and Centro de Investigação em Sistemas e Tecnologias (SYSTEC) at FEUP, aims to develop an industrial management system to support integrated decision making in the context of the Internet of Things. The applicable management systems should, in this sense, accompany this development trajectory based on pillars of flexibility and adaptability. The project proposes to develop a new generation of orchestrated systems for industrial management, capable of enhancing the efficiency of industrial operations and supporting decision making [8].

The industrial partner of the Sensitive Industry project, Bosch Security Systems, provides the application context in the form of the Digital MilkRun Application. This application, comprised of multiple layers of sensors capturing data and machines processing the generated data of the sensors, has no network management and monitoring infrastructure and relies solely on manual configuration of camera sensors and processing nodes.

This dissertation provides the infrastructure for the orchestrator operation, allowing the efficient addition, removal and replacement of sensors and processing nodes, avoiding manual configuration. The infrastructure also includes security by access control to the machines of the Digital MilkRun Application through public key cryptography and certificate authorities. The orchestrator functionality itself is outside the scope of this work.

1.3 Objectives

The objectives of this dissertation are:

- to design, develop and validate a network management and monitoring system to be deployed in the IIoT infrastructure at the Bosch Ovar factory [1];
- to streamline the on-boarding process of camera sensors and processing nodes;
- to add security with access control for processing nodes;
- to contribute to the Sensitive Industry project.

[1] later Bosch Ovar factory was acquired by Keenfinity

1.4 Contributions

This dissertation explores the design, implementation, and evaluation of an autonomic network management and monitoring system in a camera sensor and processor nodes network for IIoT applications. By addressing these objectives, this research aims to contribute to the growing body of knowledge on IIoT-driven automation and demonstrate the potential of autonomic network management systems for sensor and processor nodes networks in maintaining operational continuity and efficiency.

The findings from this study are expected to provide actionable insights for industries seeking to enhance their adaptability and resilience in an increasingly dynamic and competitive landscape. Finally, with the upgrade of Bosch Ovar's system, the project intends to fulfill the United Nations Sustainable Development Goals (SDG) 8, specifically SDG 8.2.

1.5 Document Structure

This document is structured as follows: Chapter 2 introduces the fundamental concepts relevant to this project, covering the principles and challenges of the Industrial Internet of Things, the theory of Autonomic Networks and the MAPE-K control loop, Network Automation and the script-driven solutions considered, the DDI paradigm for network management (DNS, DHCP and IPAM), Network Monitoring platforms, Security Management fundamentals, and the ONVIF standard used by the camera sensors. Chapter 3 details the design of the proposed solution, starting from the characterization of Bosch's Digital MilkRun Application and its constraints, followed by the solution proposal, the derived system requirements, and the hardware and software components selected for the implementation. Chapter 4 describes the implementation of the system, detailing the overall architecture centralized on the DDI Raspberry Pi, the certificate-based access control mechanism, and the automation developed for database synchronization, host discovery/probing/expiration, MQTT notification of the orchestrator, Ansible-based configuration of processing nodes, and automated camera sensor configuration, concluding with the resulting on-boarding work-flows. Chapter 5 presents the laboratory set-up used to validate the system, the functional testing methodology, and the results obtained for the camera sensor and processing node on-boarding test cases, along with a discussion of the system's benefits and limitations. Finally, Chapter 6 presents the conclusions of this dissertation, along with directions for future work.

Chapter 2

Technical Background

2.1 Introduction

This chapter presents the fundamental concepts and technologies relevant to the design of an autonomic network management and monitoring system for IIoT sensors. It begins by introducing the Industrial Internet of Things, its defining principles and the challenges inherent to industrial network environments. Building on this, the chapter describes the concept of Autonomic Networks, detailing the self-CHOP properties, cognitive networking principles, the MAPE-K reference model for the autonomic control loop, and the maturity levels that characterize the evolution of cognitive IoT systems. The chapter then addresses Network Automation, outlining its main categories and the script-driven automation software solutions, Ansible and Salt, considered for this dissertation. Following this, the chapter covers Network Management through the DDI paradigm, examining DNS, DHCP and IPAM concepts along with the corresponding software solutions evaluated, namely NSD, PowerDNS and BIND 9 for DNS, Kea DHCP and dnsmasq for DHCP, and phpIPAM, NetBox and Infoblox for IPAM. Network Monitoring is then discussed, presenting its underlying principles and the Nagios and Zabbix platforms considered for the system. The chapter also introduces the fundamentals of Security Management, covering authentication, access control, and the mechanisms used to ensure integrity, non-repudiation and availability. Finally, the ONVIF standard is presented, with particular focus on WS-Discovery and Profile S, given its relevance to the camera sensors used in the Digital MilkRun Application. The chapter closes with a summary of the concepts covered and their relevance to the solution proposed in the following chapters.

2.2 Industrial Internet of Things

IIoT is designed for heavy-duty tasks, such as manufacturing or monitoring. It uses more precise and durable devices, actuators and sensors. IoT and IIoT share the same core principles such as data management, network, security and cloud. However, IIoT devices generate massive amounts of data, requiring data streaming, big data, machine learning or artificial intelligence practices.

Loss of the generated data in IIoT is critical. The data itself should be more precise, continuous and sensitive than in IoT [9]. A reference architecture for IIoT can be visualized in figure 2.1.

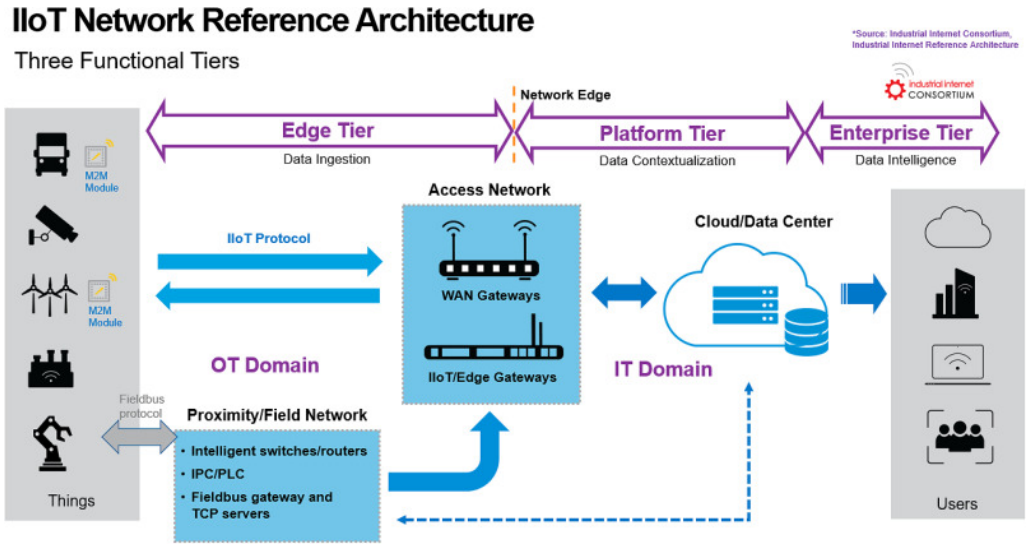


Figure 2.1: IIoT Reference Architecture [10]

IIoT deployments are generally large scale, following the principles described in the next section.

2.2.1 Principles

IIoT can be seen as having seven principles [9]:

- **Big amount of analog data -** Numerous sensors produce analog data, which must be digitized for analysis and storage;
- **Perpetual connectivity -** IIoT devices are always online, providing three main benefits: real-time monitoring, the ability to push updates and fixes continuously, encouraging purchase and adoption;
- **Real-time data streaming -** Safety mechanisms in industries like nuclear power constantly generate data, requiring real-time streaming for immediate monitoring. Any delay could lead to critical issues, so timely data streaming and aggregation are vital;
- **Data insights -** IIoT data is analyzed to answer essential questions about goals and desired outcomes;
- **Time to vs. depth-of -insight trade-off -** A trade-off exists between how quickly data insights are available and their depth. For instance, immediate monitoring is crucial in a nuclear power plant, while long-term scientific data analysis may take years;
- **Visibility from Big data -** After data is collected and stored in large data environments, it should be accessible for analysis and other uses whenever needed;

- **Edge computing -** Advanced computing and analytics are moving to the edge (closer to the source), reducing latency, bandwidth costs, and improving security and reliability while adhering to data sovereignty and compliance.

These principles entail several challenges, described in the next section.

2.2.2 Challenges

Industrial networks also face their own set of challenges [9]:

- **Wireless Coexistence -** Ensures that devices using different wireless standards can operate without interference. This issue can lead to signal problems, packet drops, and data loss if devices share frequencies;
- **Latency -** Refers to the delay between a command and its execution. In industrial settings, low latency is essential, especially for real-time responses with high-speed machinery;
- **Interoperability -** Encompasses the ability of different systems to connect and exchange data seamlessly. Cross-layer interoperability addresses network layer compatibility using tools like Software-Defined Networking (SDN), while cross-system interoperability allows systems with different architectures to communicate through semantic models or AI-based tools;
- **Sensor Data Streaming -** Involves planning for scalability, data durability, and fault tolerance. This ensures data integrity and efficient analysis in real-time streaming contexts, like cybersecurity;
- **Safety -** Focuses on protocols to prevent accidents and respond to emergencies in industrial facilities. Modern IIoT safety methods include computer vision for anomaly detection, wearable devices, cybersecurity measures, and AI-driven robotic systems for autonomous decision-making;
- **Security and Privacy -** Critical in IIoT, involving protection against cyber-attacks, with regulations like General Data Protection Regulation (GDPR) in place. Security strategies include:
  - Intrusion Detection - Uses anomaly or misuse detection to identify unauthorized access;
  - Intrusion Prevention - Employs encryption, access control, and other methods to prevent attacks;
  - Runtime Security Monitoring - Observes system states to identify threats in real-time and can activate a "reversionary" system to maintain essential functions if a threat occurs.

2.3 Autonomic Networks

Autonomous IIoT combines IIoT with autonomic computing and cognitive networks principles. Autonomic computing is the self management of single entities and/or of whole systems based on four self-* properties, also known as self-CHOP properties [11] [12][13]:

- **Self-configuration -** Automatic configuration of the entity/system;
- **Self-healing -** Detection, diagnosis and repair of entity/system components independent of human intervention;
- **Self-optimization -** Continuous seeking of better optimization of the entity/system by adjusting tuning options;
- **Self-protection -** Automatic defense mechanisms against threats to the entity/system.

Cognitive networks are designed around self-awareness and context-awareness. Their core elements, known as cognitive agents, operate within a cognition loop that leverages past experiences to enhance current performance. Each agent begins the loop by sensing current network conditions (context-awareness), then, based on this data, its own status, and predefined policies, it creates and selects an optimal plan (self-awareness). The agent then acts on this plan, evaluates the outcomes, and completes the loop. This continuous cognition loop across all agents creates a collective intelligence that improves overall network performance [13].

Autonomicity in an autonomic network may be achieved via a distributed architecture composed of autonomic managers and their managed components. The autonomic managers accomplish self-management or self-adaptation through the autonomic control loop (ACL) process or feedback loop. This loop monitors the environment and the managed system to detect possible faults (self-heal), changes in operation conditions (self-optimize) and changes in system goals (self-configure) [14]. IBM proposed a reference model for the ACL, named MAPE-K model (Monitor-Analyze-Plan-Execute over a shared Knowledge base), showed in great detail in figure 2.2.

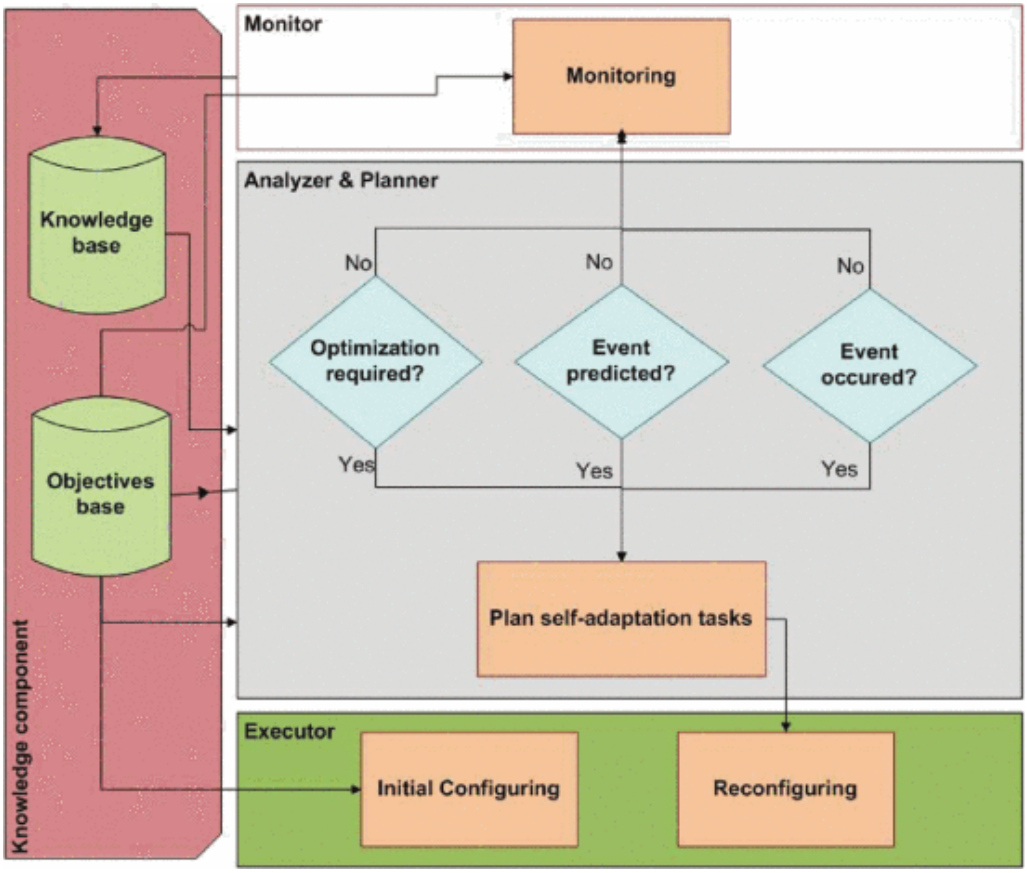


Figure 2.2: Detailed MAPE-K model for the ACL [5]

The MAPE-K model consists of:

- **High-level Objective Base -** A repository of high-level goals or instructions for the system. Also provides initial configuration and guidance for other ACL components;
- **Knowledge Base -** Location where collected data, also referred too as knowledge, is stored;
- **Monitor -** Responsible for tracking important data for the system's self-properties;
- **Analyzer -** Analyzes the knowledge taking into account the system's goals. It verifies system performance, predicts future system states and detects events;
- **Planner -** Develops plans for the system adaptation based on the Analyzer's results;
- **Executor -** Carries out the plan created by the planner, communicating with other autonomic managers if needed.

2.3.1 Maturity Levels

Maturity levels can help describe the evolution of cognitive IoT systems. As presented in table 2.1, an Autonomic IoT system is comprised of 4 maturity levels [15].

Table 2.1: Maturity Levels for IoT System

No.	Maturity Level	Services	Human Tasks	Automated Processes
1	Monitoring Management	Visualization Business Context Control Corruption Detection	Analysis Decision taking	Monitoring
2	Predictive Management	Context Evolution Prediction	Analysis validation Decision taking	Monitoring Analyzing
3	Prescriptive Management	Business Recommendations Judgment and Reasoning	Recommendations Procedural-knowledge filling	Monitoring Analyzing Planning
4	Autonomic Management	Context Evolution Management	Business rules filling	Monitoring Analyzing Planning Executing Discovering processes

Starting at the **Monitoring Management Level**, it enables human-IoT interaction by collecting and displaying real-time data, allowing for automated monitoring of changing conditions. Next up, the **Predictive Management Level** employs data mining and machine learning to analyze trends and predict anomalies. Above that, the **Prescriptive Management Level** enables IoT-driven business decisions based on behavioral insights and expert rules. Finally, in the **Autonomic Management Level**, the system adapts autonomously to dynamic contexts. It proactively adjusts management processes.

2.4 Network Automation

According to IBM, network automation refers to the application of software-driven processes to configure, manage, test, deploy, and operate both physical and virtual network infrastructure with minimal human intervention. Its emergence responds to a structural mismatch: while networking hardware and protocols have advanced considerably, day-to-day network administration has historically remained dependent on manual, command-by-command intervention. This reliance on manual configuration introduces well-documented risks — inconsistent policy application, slower response to demand fluctuations, and a higher likelihood of human error, which in turn contributes to outages and security exposures. Automation is therefore positioned not merely as an efficiency measure but as a response to the operational limits of manual network management in environments characterized by scale and volatility [16].

There are several categories of network automation depending on function:

- **Orchestration Automation -** Coordinates interdependent systems and services — including virtual network functions, load balancers, and firewalls — to maintain coherent operation across distributed applications;
- **Provisioning Automation -** Integrates new devices into a network automatically, applying the required configuration without manual setup;

- **Script-driven Automation -** Executes predefined tasks in response to triggers, historically implemented in languages such as Perl and Tcl and increasingly in Python;
- **Software-based/Intelligent Automation -** Relies on administrative interfaces and templates rather than manual scripting;
- **Intent-based Automation -** Uses AI/ML to interpret business or performance objectives and adjusts network policy enforcement to meet defined service levels;
- **Security Automation -** Applies AI and related tools to vulnerability scanning, intrusion detection, incident response, and compliance monitoring;
- **Load-balancing and Failover Automation -** Manages traffic distribution and failover response to sustain application performance under variable load.

2.5 Script-driven Automation Software Solutions

Script-driven automation, alongside with Perl, Tcl and Python implementation, can also be implemented with software solutions such as Ansible and Salt.

2.5.1 Ansible

Ansible is an open-source automation tool that, through simple human-readable scripts named playbooks, automates tasks. The user declares de desired state of a local or remote system in the playbook and ansible ensures the system remains in that state. Common use cases for Ansible are [17]:

- Elimination of repetition and simplification of workflows;
- Management and maintenance of system configuration;
- Continuous deployment of complex software;
- Rolling of zero-downtime updates.

Ansible is also designed around the principles of:

- **Agentless Architecture -** Low maintenance overhead by avoiding the installation of additional software across IT infrastructure;
- **Simplicity -** Automation playbooks use straightforward YAML syntax for code that reads like documentation. Ansible is also decentralized, using SSH with existing OS credentials to access remote machines;

- **Scalability and Flexibility -** Easily and quickly scale the automated systems through a modular design that supports a large range of operating systems, cloud platforms, and network devices;
- **Idempotence and predictability -** When the system is in the state the playbook describes, Ansible does not change anything, even if the playbook runs multiple times.

2.5.2 Salt

Salt is an open-source python-based remote execution framework built to consolidate the best ideas from remote execution tools into something faster and more flexible and specifically engineered to scale from a handful of servers to thousands. Its dual-purpose platform functions as a [18]:

- **Configuration Management System -** enforces defined states on remote nodes (e.g., ensuring packages are installed, services running);
- **Distributed Remote Execution System -** runs commands/queries across nodes individually or via arbitrary selection criteria.

Salt's architecture can be both:

- **Agent -** With a topology of a simple server/client model, scalable from small local networks to multi-datacenter, international deployments;
- **Agentless -** Executing salt commands and states over SSH

2.6 Network Management: DDI

DDI stands for DNS, DHCP, and IPAM, three essential network services that, when integrated into a single solution, provide centralized visibility, control, and automation [19].

The benefits of a centralized DDI solution are:

- Single-pane management of DNS, DHCP, and IPAM;
- Real-time updates between services (e.g., DHCP leases automatically updating DNS);
- Conflict prevention and better data consistency;
- Automation of routine tasks like adding/deleting host or IP records;
- Improved security, resiliency and auditability.

2.6.1 DNS

The Domain Name System (DNS) provides a way to name resources across diverse networks and systems. Users interact with DNS through resolvers, which retrieve information like IP addresses or mail server data associated with domain names. To the user, DNS appears as a unified system, while resolvers handle the complexity of its distributed nature [20].

Resolvers query name servers to locate data or get referrals to other servers, gradually building knowledge of the DNS hierarchy. Name servers store two types of data: authoritative data, which defines specific portions (zones) of the domain tree, and cached data, temporarily stored by resolvers to speed up repeated queries. This structure separates user interaction and fault handling (handled by resolvers) from data management and synchronization (handled by name servers).

2.6.1.1 NSD

Name Server Daemon (NSD) is a high-performance, authoritative-only DNS server developed by NLnet Labs. Designed for speed, reliability, and security, it excels in serving large volumes of DNS queries, making it ideal for top-level domains and root servers. NSD includes simdzone, a fast zone file parser that leverages modern CPU instructions for faster zone loading [21].

By design, NSD does not perform recursive queries or caching. For recursive DNS, NLnet Labs provides Unbound. NSD is open source under the BSD license, widely available as packages, and supported by a security patch policy. Commercial support with SLAs is also available.

2.6.1.2 PowerDNS

PowerDNS provides three base products: authoritative server, recursive server and DNSdist.

The PowerDNS Authoritative Server enables providers to host domain names efficiently, delivering fast responses to recursive resolvers. It supports authoritative DNS services using a wide range of major databases and offers exceptional scalability for both customized setups and large-scale hosting environments. Additional features include automated reporting, real-time analytics, and geographic load balancing [22].

PowerDNS Recursor is a high-performance, low-latency DNS caching resolver that ensures accurate and fast responses to subscriber DNS queries. Designed for speed, reliability, and security, it enhances overall internet performance, playing a vital role in delivering a smooth and safe user experience [23].

Finally, DNSdist is an advanced, DNS-aware load balancer that sits in front of recursive servers to filter, protect, and distribute DNS traffic efficiently. While tightly integrated with PowerDNS

Recursor, DNSdist also enhances any DNS resolver by adding critical features like encryption and traffic management [24].

2.6.1.3 BIND 9

BIND 9 is a full DNS protocol implementation that can function as an authoritative server, resolver, and stub resolver (on supported systems), with flexible configuration options. While large operators often separate these roles, smaller setups can use BIND 9 to perform multiple DNS functions on a single system [25].

It offers a wide range of example configurations, including basic setups, forwarding, advanced use cases, and secure configurations.

2.6.2 DHCP

The Dynamic Host Configuration Protocol (DHCP) is a client-server protocol that automatically provides IP addresses and other configuration parameters to hosts on a network. DHCP servers manage the allocation of IP addresses and deliver network configuration data to clients [26].

There are three IP address allocation methods in DHCP:

- **Automatic allocation -** permanently assigns an IP address;
- **Dynamic allocation -** assigns an IP for a limited time or until released;
- **Manual allocation -** a fixed IP is set by an administrator and delivered by DHCP.

Dynamic allocation is especially useful for temporary devices or managing limited address pools. DHCP uses the Bootstrap Protocol (BOOTP) message format to maintain compatibility with existing BOOTP clients and to support relay agents, allowing centralized server deployment without needing a DHCP server on every network segment.

2.6.2.1 Kea DHCP

Kea DHCP is the most recent open-source, full featured, standards-based DHCP server. Key features are [27]:

- **Modular and Extensible -** Uses separate daemons for DHCPv4, DHCPv6, and DDNS. Features are added via optional “Hooks Modules” (some provided, or custom ones in C++);
- **Live Reconfiguration via REST API -** Configuration is in JSON and can be updated live using API commands—no restart needed (unlike ISC DHCP);

- **Backend Integration -** Leases, reservations, and config data can be stored separately from the server, using MySQL or PostgreSQL backends. This allows integration with provisioning systems, IPAMs, and centralized configuration;
- **Web Dashboard (Stork) -** Stork provides a centralized GUI to monitor and manage multiple Kea servers using deployed agents;
- **Open Source and Cross-Platform -** Kea is open source (MPL 2.0), developed openly on GitLab, and runs on Linux, Unix, and macOS. Pre-built packages are available for most OSes.

2.6.3 dnsmasq

Dnsmasq is a lightweight, versatile tool that provides DNS, DHCP, router advertisement, and network boot services, ideal for small or resource-constrained networks such as home routers, mobile hotspots, or virtual machines [28].

Key features:

- **DHCP**
  - Supports DHCPv4, DHCPv6, BOOTP, and Preboot eXecution Environment (PXE);
  - Handles static and dynamic leases, stateless DHCPv6, and PXE boot menus;
  - Includes a proxy PXE mode and a read-only Trivial File Transfer Protocol (TFTP) server for netboot;
  - DHCP-configured hostnames are automatically added to DNS;
- **DNS**
  - Acts as a local DNS server with forwarding and caching (A, AAAA, CNAME, PTR, DNSKEY, DS);
  - Supports /etc/hosts, DHCP-imported names, and custom record types;
  - Can run in authoritative mode, exporting local zones and supporting zone transfers;
  - Domain Name System Security Extensions (DNSSEC) validation protects against spoofing and poisoning;
  - Supports internationalized domain names and domain-specific upstream resolvers;
- **Router Advertisement**
  - Provides IPv6 auto configuration, optionally combined with DHCPv6;
  - Supports Recursive DNS Server (RDNSS), configurable M and O flags, and mapping DHCPv4 names to auto configured IPv6 addresses.

2.6.4 IPAM

IPAM refers to the coordination and administration of DNS and DHCP services, which are responsible for assigning and resolving IP addresses within a TCP/IP network. In essence, IPAM is used to plan, track, and manage the use of IP address space across a network. While DNS and DHCP often work together to handle these tasks, a complete IPAM solution integrates the two—ensuring that changes in one (e.g., a new IP lease issued by DHCP) are automatically reflected in the other (e.g., DNS updating with the new hostname-to-IP mapping) [29].

2.6.4.1 phpIPAM

phpIPAM is a web-based, open-source IPAM tool designed to be lightweight, modern, and user-friendly. Built with Hypertext Preprocessor (PHP), MySQL, jQuery, Asynchronous Javascript and XML (AJAX), and HTML5/CSS3, it offers comprehensive features for managing IP networks [30]:

- **IP Address Management -** Supports both IPv4 and IPv6, with automatic free space calculation and visual subnet display
- **Subnet and Section Management -** Organize IP space hierarchically with permission controls per user group
- **Automation -** Includes automatic subnet scanning and IP status checks
- **Integrations and Support**
  - PowerDNS integration;
  - Active Directory, LDAP, Radius authentication;
  - REST API for external integration;
- **Network Infrastructure Management**
  - Device and device type tracking;
  - RACK, VLAN, and VRF management;
  - NAT support;
  - Location tagging;
- **Data Import/Export -** Import subnets via RIPE, XLS, or CSV;
- **Utilities**
  - IP calculator;
  - Searchable IP database;
  - IP request system;

  - E-mail notifications;
  - Custom fields and changelogs;

- Multi-language support through translations.

2.6.4.2 NetBox

NetBox is an open-source network source of truth and infrastructure resource modeling tool originally developed at DigitalOcean in 2015. It is now maintained by NetBox Labs and a community of contributors. Designed specifically for network engineers and operators, NetBox serves as a central system for documenting and automating network infrastructure [31].

Key Features:

- **IPAM -** IPv4/IPv6 address management, automatic IP/prefix assignment, VRFs, VLANs, and ASNs;
- **DCIM -** Rack layouts (with SVG), device modeling, cable tracing (network, power, console), power distribution;
- **Connectivity -** VPN tunnels, wireless links, layer 2 overlays, FHRP groups (e.g., VRRP, HSRP);
- **Virtualization -** Virtual machines, clusters, and device contexts;
- **Data Modeling -** Custom fields, validation rules, reports, and in-UI scripts;
- **Extensibility -** Rich plugin system, REST API, event-driven webhooks;
- **User Management -** Role-based access, SSO support, object-level permissions;
- **Change Tracking -** Auto change logs, global search, application service bindings;
- **Other -** Tenant tracking, site/location hierarchy, configuration contexts.

2.6.4.3 Infoblox

Infoblox is a leading provider of DNS, DHCP, and IP Address Management (DDI) solutions. Its product is the first SaaS-based DDI platform built for modern hybrid and multi-cloud environments, including AWS, Azure, Google Cloud, and on-premises systems [32].

Key Features:

- **Unified DDI Management**
  - Centralized, policy-driven DNS, DHCP, and IPAM;

  - Fast, elastic, and low-maintenance network service delivery across cloud and on-prem environments;
  - Deep integration with Infoblox Security and other ecosystem tools;

- **Universal IPAM**
  - Unique in the industry;
  - Provides policy-based IP address allocation across distributed networks;

- **Monitoring with Network Workspace -** The Network Workspace offers real-time visibility and management of DDI services via six summary monitors;

- **Reliability and Resilience -** Infoblox ensures 24/7 service availability through DHCP Ping Before Offer, Fault-Tolerant DNS Caching and Portal-Based DHCP Management.

2.7 Network Monitoring

Monitoring is essential for managing computer systems. It involves collecting and consolidating data about system components' status and configuration. This information is used to support management functions under the Fault, Configuration, Accounting, Performance, Security (FCAPS) framework [33].

The most important types of monitored information are:

- **Status Information -** Indicates if components are active and functioning;
- **Configuration Information -** Tracks adjustable settings to detect misconfigurations;
- **Usage and Performance Statistics -** Includes throughput, users, latency, etc;
- **Error Information -** Details on faults and errors over time;
- **Topology Information -** Monitors network structure and changes.

Most monitoring systems follow a five-layer structure:

- **Data Collection -** Gathers raw data from system components;
- **Pre-Database Processing -** Filters, formats, and reduces data volume
- **Data Storage -** Stores processed data for historical analysis;
- **Post-Database Processing -** Analyzes data using various algorithms;
- **Reporting -** Presents data to administrators in actionable formats.

Data collection can be **passive**, this meaning the system is observed without introducing additional load, or **active**, which injects requests to measure performance or availability.

The most common active monitoring types are:

- **Applications -** Use logs (e.g. Apache logs) or proxies to gather data;
- **Servers/PCs -** Use agents or system commands to monitor performance. Agents must avoid synchronization to prevent overloads;
- **Networks -** Monitor via Simple Network Management Protocol (SNMP) traps, protocol participation (e.g., Open Shortest Path First (OSPF)), or packet sniffing using methods like port mirroring, network taps, or IP Flow Information Export (IPFIX).

Passive Monitoring:

- **Applications -** Send synthetic transactions to test performance. These may be read-only or designed not to alter application state;
- **Servers/PCs -** Agent-based, meaning server polls agents for data or Agent-less, where server logs into devices to run scripts (credential management is key here);
- **Networks -** Use SNMP polling, or tools like ping and traceroute to measure reachability and latency. These tools are easy to use but can be inefficient if overused.

2.7.1 Nagios

Nagios is a powerful, scalable IT infrastructure monitoring tool designed to ensure the reliability of business processes by detecting, alerting, and helping resolve system issues before they impact users [34].

Key features:

- **Monitoring -** Tracks system metrics, services, servers, network devices, and protocols;
- **Alerting -** Notifies admins of failures and recoveries via email, SMS, or scripts;
- **Response -** Alerts can be acknowledged, escalated, and acted on quickly;
- **Reporting -** Offers historical logs and availability reports for SLA verification;
- **Maintenance -** Allows scheduled downtimes to suppress unnecessary alerts;
- **Planning -** Capacity and trend reports help predict and prevent future failures.

2.7.2 Zabbix

Zabbix is a free, open-source, enterprise-grade monitoring platform developed by Zabbix SIA. It monitors networks, servers, VMs, applications, databases, cloud services, and more. Designed for both small and large environments, Zabbix provides extensive data collection, alerting, and visualization capabilities [35].

Key features [36]:

- **Comprehensive Monitoring**
  - Monitors availability, performance, services, websites, and hardware;
  - Supports SNMP, IPMI, JMX, VMware, and custom checks;
  - Polling and trap-based data collection;
- **Flexible Alerting**
  - Configurable escalation, recipient targeting, and notification channels (email, scripts, etc.);
  - Supports automated remote commands;
  - Uses macros to enhance message content;
- **Advanced Visualization**
  - Built-in graphs, custom dashboards, network maps, reports, and slideshow views;
  - Real-time data graphing and historical data storage in a database;
- **Web Monitoring -** Simulates user interaction to monitor website functionality and performance;
- **Discovery and Configuration**
  - Auto-discovery of network devices and services;
  - Agent auto-registration and host templating for scalable setup;
  - Easy-to-use web interface for configuration and management;
- **Extensibility**
  - Rich API for automation and third-party integration;
  - Templates to group and reuse monitoring checks;
  - Supports deployment of agents on Linux and Windows;
  - Zabbix Proxy allows remote and distributed monitoring;
- **Performance and Licensing**

  - Written in C for speed and low resource use;
  - Licensed under AGPL-3.0 and free to use;
  - Commercial support is available worldwide

2.8 Security Management

Security management aims to protect computer systems and networks from unauthorized access while ensuring reliable and continuous operation. Since absolute security cannot be guaranteed in any practical system, security mechanisms are designed to balance protection, usability, performance, and operational cost.

Information security is commonly defined through five fundamental principles:

- Authentication verifies the identity of users or devices requesting access to the system;
- Confidentiality ensures that information is accessible only to authorized entities;
- Integrity protects data and system configurations from unauthorized modification;
- Non-repudiation provides accountability by ensuring that actions can be traced to their origin through logging and auditing mechanisms;
- Availability ensures that authorized users can reliably access services, even in the presence of failures or malicious attacks.

Authentication is commonly implemented using passwords, biometric information, or cryptographic credentials. In larger distributed systems, asymmetric cryptography is widely adopted through a Public Key Infrastructure (PKI), where each entity possesses a public/private key pair and a trusted Certificate Authority (CA) issues digital certificates that bind public keys to identities. This approach enables secure identity verification without requiring the exchange of shared secret keys.

Once authentication is completed, access control determines the resources and operations that an authenticated entity is permitted to access. Common access control models include access control lists (ACLs), capabilities, role-based access control (RBAC), and white-list or black-list mechanisms, depending on the security requirements and administrative policies.

Maintaining system integrity involves preventing or detecting unauthorized software, devices, or configuration changes. This can be achieved through configuration validation, anomaly detection, or signature-based approaches such as white-listing trusted software or black-listing known malicious components. Similarly, non-repudiation relies on audit logs that record system events, enabling administrators to trace configuration changes and security incidents.

Availability focuses on protecting systems against disruptions, particularly denial-of-service (DoS) attacks. Common mitigation strategies include traffic monitoring, anomaly detection, redundancy, and capacity provisioning to ensure that legitimate users retain access to network services during adverse conditions.

2.9 ONVIF

ONVIF is a standard used by Bosch's camera sensors that allows the interoperability of IP-based physical security products and services [37]. The interoperability provided by ONVIF allows for:

- cost-efficient system setup;
- more efficient workflows;
- flexibility in system design;
- secure future system development.

2.9.1 Overview

ONVIF's core specification covers generic IP device use cases such as device discovery and configuration. The framework is built on Web Service (WS) standards such as [38]:

- XML for data description syntax;
- SOAP for message transfer, with HTTP as the underlying transport mechanism;
- WSDL for describing the services.

The devices (service providers), such as cameras, implement ONVIF services that can be used by the service requester (client) using WSDL as the base for the service requester implementation. Core functions of the ONVIF standard include [38]:

- Device discovery utilizing the WS-Discovery standard;
- Device management for managing device capabilities, network and system settings, security operations and firmware upgrades.

ONVIF also defines security requirements for [38]:

- Transport-level security;
- Message-level security;
- Port-based network security as IEEE 802.1X.

2.9.2 WS Discovery

WS-Discovery is a protocol that lets network devices/services find each other without needing pre-configured directories. Devices can have 3 core roles [39]:

- **Target Service -** An endpoint offering itself up to be found, for example a camera sensor;
- **Client -** An endpoint searching for a Target Device;
- **Discovery Proxy -** An optional intermediary that helps things scale

The protocol has two operating modes:

- **Ad Hoc Mode -** Everything happens over IP multicast on a well-known address (239.255.255.250 for IPv4, port 3702). No DNS or directory service required, the devices talk directly on the local network;
- ´**Managed Mode -** Once a Discovery Proxy is available, communication switches to unicast messages sent directly to that proxy, reducing multicast chatter and letting discovery scale beyond a single ad hoc segment.

WS-Discovery has four key messages:

- **Hello -** A Target Service broadcasts this when it joins the network (or when its metadata changes), announcing itself and its capabilities;
- **Bye -** Sent when a Target Service is about to leave the network, letting listeners know it's going away;
- **Probe -** A Client broadcasts this to actively search for services matching certain Types and/or Scopes. Matching Target Services respond directly with a **Probe Match**;
- **Resolve -** Used when a Client already knows a service's unique identifier (Endpoint Reference) and just needs its current network address. Matching Target Services respond with a **Resolve Match**.

This specification helps minimize network chatter by allowing Target Services to proactively send **Hello** messages when joining the network so Clients are able to build up a picture of the network just by listening, without needing to probe constantly. If a Discovery Proxy is present it can "suppress" multicast traffic by signaling its existence to a Client when he probes for Target Services, making the Client switch to sending unicast requests straight to the Discover Proxy.

2.9.3 Profile S

Out of all the profiles provided by ONVIF, the one to highlight is profile S. A device compliant to profile S includes user management, the sending of video data over IP to clients, support for Pan-Tilt-Zoom (PTZ), audio and metadata streaming. A client compliant to profile S can configure, request and control streaming of video data over IP from a given ONVIF device compliant to profile S, control of PTZ, receiving audio and metadata stream [40].

Chapter 3

Solution Design

3.1 Introduction

This chapter describes the design of the proposed solution. It starts by characterizing Bosch's Digital MilkRun Application, including its current system architecture and the existing camera sensor on-boarding process, in order to identify the constraints of the current setup that motivated this dissertation. Based on these constraints, the chapter presents the solution proposal for an autonomic network management and monitoring system. It then details the system requirements derived from this proposal, followed by the hardware and software components selected to implement it.

3.2 Digital MilkRun Application

The Digital MilkRun Application is Bosch's Ovar response to the challenges of transportation logistics within their industrial environment between the warehouse and production lines. Using data from camera sensors installed throughout designated routes between the warehouse and production lines and computer vision algorithms, the Digital MilkRun Application is able to determine the best routes to take carriages or tugs and the order by which they should be delivered/distributed.

3.2.1 System Architecture

The system architecture can be described as a total of three layers, as can be seen in figure 3.1:

- **Layer 1 -** Camera Sensors;
- **Layer 2 -** Image Anonymization;
- **Layer 3 -** MilkRun tracking and parking.

Layer 1 consists of the camera sensors that stream the raw video data of the routes and parking spots in the factory warehouse and production lines. The streams generated from this layer have their access restricted to only the DSO, Plant Manager, Tech Manager and Tech Lead of the Digital Team. Communicating and controlling the cameras is possible using the ONVIF protocol or Remote Control Protocol (RCP).

Layer 2 contains the machines that receive the raw video data stream from the camera sensors of Layer 1 and apply the anonymization algorithm to the images to hide personal information from operators that happen to be in the camera Field of View (FOV). Restrictions to this layer are the same as the ones for Layer 1.

Layer 3 corresponds to the machines that process the different tasks of the Digital MilkRun. As these machines do not have access to the raw stream video data, they get the frames from the machines of Layer 2 as a client, after Layer 2 applies the anonymization. The access levels of this layer are looser than the previous two layers.

All the network configurations are done manually and locally in each of the devices of Layer 1, 2 and 3. The IP address space is maintained in an excel sheet.

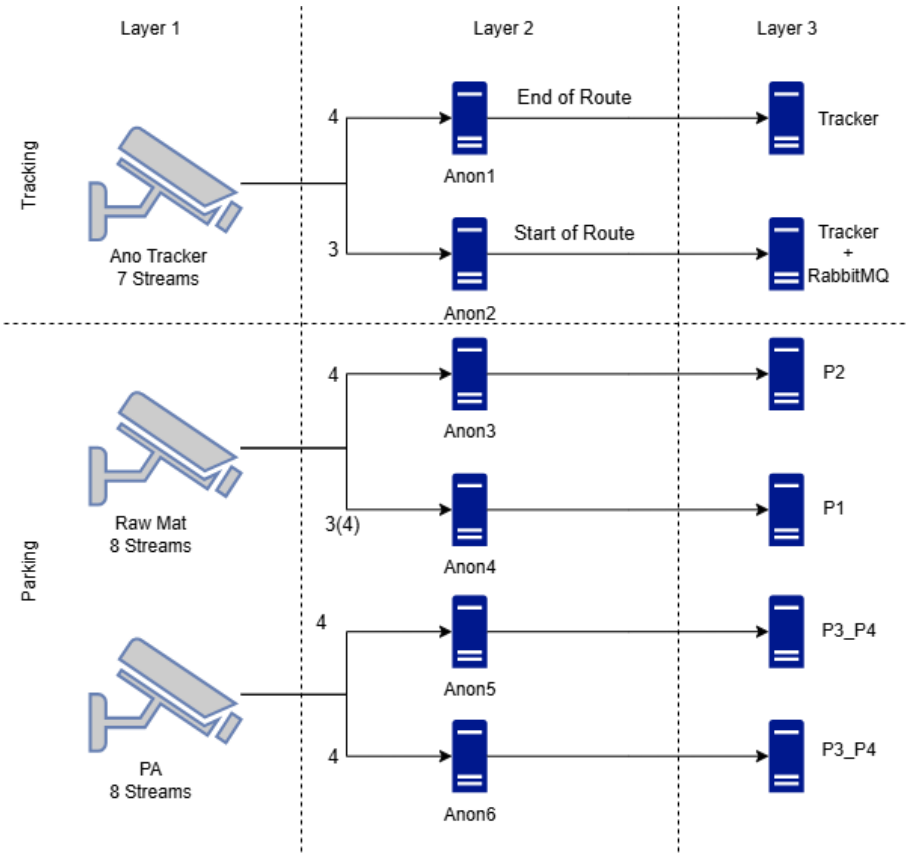


Figure 3.1: Digital MilkRun System Architecture

Both the Layer 2 and Layer 3 machines are Linux based and from now on will also be referred to as processing nodes.

3.2.2 Camera Sensor On-Boarding

The cameras connect to a Virtual Local Area Network (VLAN). The process of setting up a camera involves requirements from Bosch's Ovar logistics team, maintenance team and Computer Vision (CV) team. An activity diagram of the current camera on-boarding process can be seen in figure 3.2.

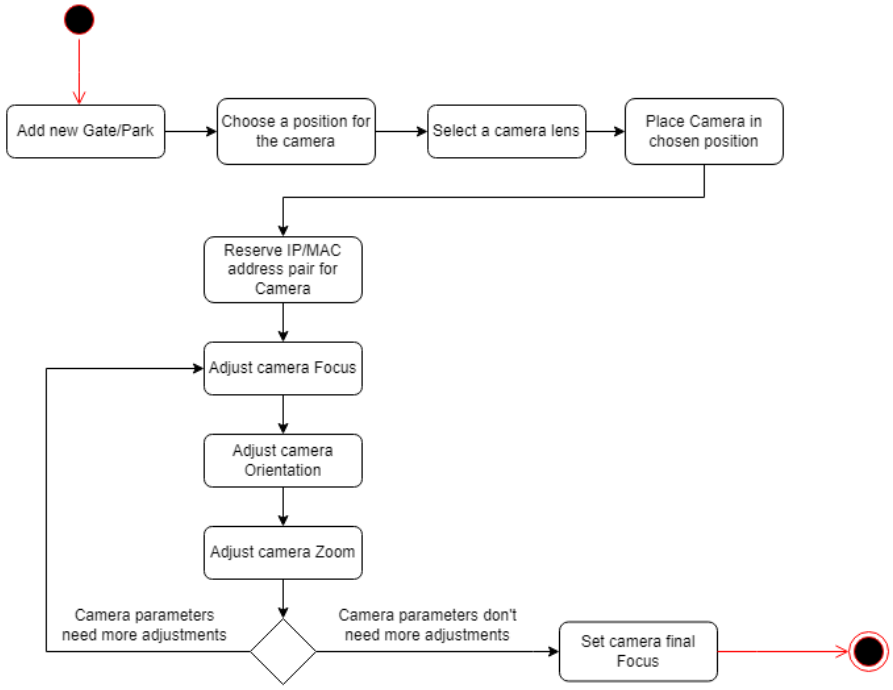


Figure 3.2: Camera On-Boarding Activity Diagram

Regarding logistics, there's a notification that a gate or park is being created or changed (a new warehouse is being added or the factory floor configuration has been changed). A set of zones are given to install the new camera in order to not affect production processes.

In the CV side, a lens with proper focal distance must be chosen to adapt to the distance from the target along with the pretended Field of View (FOV). For gate monitoring, the camera must be positioned to capture the MilkRun passing through, ensuring clear detection of tugs, carriages, and arucos. Ideally, the camera should be elevated and positioned on the left side of the gate, minimizing background noise in the field of view. Before production integration, the computer vision team should verify detection accuracy and retrain the AI model if necessary. For park monitoring, cameras should be roof-mounted with a slight inclination, covering all carriage positions. If arucos are obstructed, the logistics team must be notified.

Finally, maintenance utilizes the needed material, camera, lenses and mounting information such as position and height and where the network backstage is located that the camera will connect to for it's installation on the defined location. An IP in the network is also reserved for the camera's MAC address

During the camera installation, the maintenance and CV team should work together to properly install the camera in five steps:

1. **Position -** based on previous criteria, the maintenance operator installs camera on set position;
2. **Focus -** can be changed physically in the camera and in the camera application. In the application the focus needs to be in a median value so it can be properly manipulated after the installation. The lens focus should be changed by the maintenance operator until the CV team is able to analyze a focused enough stream;
3. **Orientation -** during analysis of the current camera application's FOV, the CV team should indicate what changes the maintenance operator should physically enact on the camera's orientation to cover the desired zone;
4. **Zoom -** can only be changed physically by the maintenance operator, requiring once again the coordinated effort between the CV and Maintenance teams. Could lead to the update of steps 2 and 3;
5. **Final Focus -** The CV team applies local focus through the camera application on the most relevant part of the image.

The Layer 2 processing nodes access the camera streams through the Real-Time Streaming Protocol (RTSP) under the user "service" and a password. This process is also done manually. The camera comes by factory default with DHCP enabled and without a password set for the user *service*, prompting a user to set one when connected to the camera's web API. The web API allows the use of a Remote Control Protocol (RCP) to operate the camera.

3.3 Constraints of Digital MilkRun Application

Based on the Digital MilkRun Application description, the following constraints were identified:

- limited network management infrastructure;
- lack of network monitoring;
- heavy manual configuration of devices;
- basic access control.

The network management is very limited without IPAM to properly centralize and manage the IP address space occupied, no DHCP to automatically configure the network settings of the devices and no DNS for name-to-address translation. Network monitoring is inexistent, removing the ability to properly assert the status, configuration information, usage and performance statistics of the devices in the network. Both the camera sensors of Layer 1 and processing nodes of Layers 2 and 3 require manual configuration of network settings and passwords and users for access control. The on-boarding process of a new processing node also doesn't seem to be properly formalized. The access control to the network devices is simple, only applying user and password authentication. In summary, the Digital MilkRun Application has no self-* properties of Autonomic Networks and has minimalist access control.

3.4 Solution Proposal

Targeting the previously identified constraints and giving the system self-configuration, it is proposed a system that is capable of network management by deploying available software solutions for DDI and automation capable of configuring both the camera sensors and the processing nodes once they are connected to the network so they are ready to execute their task. This will simplify the management and on-boarding of devices in the network of the Digital MilkRun Application. Access control to the processing nodes will be managed by a Certificate Authority.

In order to support an orchestrator responsible for the self-optimization of the Digital MilkRun Application, the system will have network monitoring in order to feed it the processing nodes and camera sensors performance information. This will also set the maturity of the IIoT system at the Monitoring Management Level. The orchestrator will also be notified of the devices entering and leaving the network along with their respective capabilities, this meaning if they serve the role of a processing node or a camera sensor.

The full management and monitoring system will be centralized into one machine and its control given to one user, the system administrator. The system administrator will be the sole entity responsible for:

- keeping the IP address space updated in order to accommodate the set up of the Digital MilkRun Application;
- addition of new processing nodes and camera sensors;
- issuing certificates, granting access to the processing nodes to other lower privileged users.

An on-boarding process for processing nodes will be formalized, defining a base standard operating system image with the necessary software elements in order for the device to be compatible with the new underlying network management and monitoring system. The on-boarding process

of the camera sensors will also be improved by allowing their password to be automatically set once connected to the network, allowing the camera to be added to the Digital MilkRun Application with initial factory default settings.

Finally, a small scale set up, located at Instituto de Telecomunicações laboratory in FEUP, will be built in order to mirror the network topology of the Digital MilkRun Application. I will be comprised of:

- the central machine hosting the management and monitoring system;
- camera sensors;
- processing nodes;
- switches.

3.5 System Requirements

In order for network management to be restricted only to the system administrator, the IPAM application must have login and authentication functionalities. Through the IPAM application the system admin will be able to add, remove or update hosts in the network with their IP and MAC address, along with their hostname and description. It is also necessary the system administrator is able to enforce access control through a certificate authority, being able to sign public keys with the issued certificate so lower privileged users can access the processing nodes.

The following tasks need to be automated by the system:

- synchronization between IPAM, DHCP, DNS and the Monitoring server;
- probing new hosts when they are connected to the network so their capabilities can be identified, this meaning if they are either a camera sensor or processing node;
- communicate the addition or removal of hosts to the orchestrator through MQTT along with their capabilities;
- setting up a password for new camera sensors;
- configuring new processing nodes SSH server to only allow SSH sessions with credentials signed by the CA;
- configure a monitoring agent in new processing nodes, enabling the monitoring server to collect monitoring data from those nodes.

With all of the previous topics as reference, the high level system requirements to be implemented in the project, as seen in table A.1 of Appendix A, are presented in 5 columns:

- **ID -** Gives the requirement an identification number for easier tracking and referencing;
- **Responsibility -** Determines which entity is responsible for executing the action in the requirement:
  - *System* is the automated part of the management and monitoring system itself;
  - *System Admin* is the user of the system with the highest level of permissions in it;
- **Want to be able to -** Action that we want to happen, executed by either of the entities previously mentioned;
- **So that -** Reason why we want that action to be executed;
- **Priority -** Priority of the requirement, divided into three categories:
  - *Must* is the highest level of requirement priority, indicating it is mandatory for the system;
  - *Should* is a medium level of requirement; priority, defining the requirement as important but not strictly essential for the system;
  - *Could* is the lowest level of requirement priority, meaning the requirement would be a nice-to-have desirable feature but with low impact on the system.

Requirements with ID 1 and 2 address the IPAM application functionality necessary for the system administrator to execute the tasks of managing the hosts in the network, while requirement 4 address the access control. On the system responsibility side, requirements with ID 3, 5, 6, 7, 8, 9, 10 and 11 address the automation tasks.

3.6 Hardware

The hardware used for this project was provided by both Bosch and IT FEUP and consists of:

- 4 Raspberry Pi (RPi) 5;
- 1 camera Bosch DINION IP Starlight 6000;
- 1 Camera Bosch DINION IP ultra 8000;
- 1 TL-SG2210MP TP-Link PoE Smart Switch;
- 1 TL-SG1005D TP-Link Switch;
- 1 TL-SG1005P TP-Link PoE Switch.

All devices will communicate via wired communication, being connected with ethernet cables. The TL-SG1005D TP-Link Switch and TP-Link PoE Switch are both plug-and-play unmanaged switches, meaning no configuration is needed for them to work, they automatically forward ethernet traffic between ports. The TL-SG2210MP TP-Link PoE Smart Switch does require configuration, however it will mostly be left on default, with all the ports assigned to one VLAN (VLAN 1), due to the fact this switch, like the previous two, is only needed to provide PoE to the cameras and forward traffic between the ports.

3.7 Software

A huge emphasis is put into enabling automation, this means that the software solutions picked for DDI contain at least API support, database backend or control over triggering events. Other important factors will be taken into account as well, such as being open-source or having a free license, compatibility with Raspberry Pi OS and ease of installation and maintenance. Network management will use:

- **DNS -** PowerDNS for being free, its suite of Authoritative and Recursive servers plus dnsdist for load balancing between them and being highly extensible (supports MySQL, PostreSQL, SQLLite, etc);
- **DHCP -** Kea DHCP due to it being free, designed for large scale networks, containing API support and having kea hooks to help with network automation;
- **IPAM -** phpIPAM because it's a free, open-source solution, has API support and can operate with MySQL as a backend.

Regarding network monitoring, Zabbix was chosen, due to it's free license and powerful features, including customizable dashboards, ability to use either zabbix agents or SNMP to collect host data, and API to help with tasks automation. Since it's currently unclear what data would be useful to the orchestrator, the place holder data being collected will be:

- CPU utilization;
- Load average (1 minute average);
- Load average (5 minutes average);
- Load average (15 minutes average);
- Number of running processes.

The PowerDNS, phpIPAM and Zabbix services require a database as backend for the proper functioning of the applications. For this end MariaDB, a fork of MySQL, was chosen due to being free and compatible with all 3 services.

For communicating with the orchestrator using MQTT, the Mosquitto broker is installed due to being lightweight and open-source. In order to assert the new devices capabilities, this meaning if the device is a processing node or a camera, the processing node will be identified by probing for an SSH server running in port 22. For the camera sensor, since all of them run an ONVIF service, they will be identified by attempting HTTP connections to common ONVIF URL paths.

Network automation will leverage ansible to periodically execute playbooks in the processing nodes through SSH in order to configure them. For automation tasks requiring synchronization of databases, host probing and orchestrator notification, python scripts will be developed for each of those automation tasks and they will be either executed by periodic tasks or event driven through the kea hooks handler calling them. In the specific case of kea hooks, the script handling them will be written in bash using the example provided by kea's documentation as a base.

Since the processing nodes and central machine hosting the network management and monitoring system are RPi 5s, they will run with the Raspberry Pi OS, an operating system similar and based in linux. All the RPi OS images used will have a high privileged user named *raspi*. This user will be used by both the system administrator to access and manage the devices and by the network management and monitoring system to carry out its automation tasks. Additionally, the base processing node Raspberry Pi OS image will have SSH enabled and by default accept connections with the public key of the *raspi* user of the central RPi machine, allowing SSH connections for the ansible playbooks. It will also come with the zabbix agent, assuming the Digital MilkRun Application network will not have internet connection, disallowing the installation of the agent during the processing node on-boarding.

Chapter 4

Implementation

4.1 Introduction

This chapter describes how the solution presented in Chapter 3 was implemented. It first details the overall system architecture, centralized in the DDI Raspberry Pi, and the access control mechanism built around a certificate authority. It then describes the automation developed to support the system's autonomic behavior, including the synchronization of the DDI and monitoring databases, the discovery, probing and expiration handling of hosts on the network, the MQTT broker and message formats used to notify the orchestrator, the Ansible playbooks used to configure processing nodes, and the process used to automatically configure camera sensors. The chapter concludes by describing the resulting host on-boarding work-flow for both camera sensors and processing nodes.

4.2 System Architecture

The Autonomic Network Management and Monitoring system's architecture has been centralized into one machine, named DDI RPi, as seen in figure 4.1. In this machine, the full stack of services required for DDI network management is implemented plus monitoring and a MQTT broker.

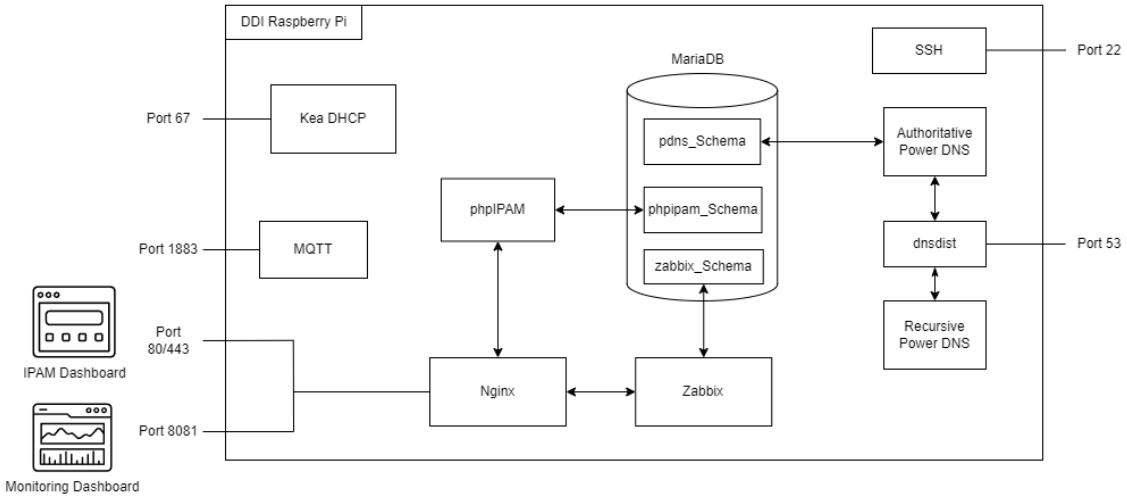


Figure 4.1: DDI Raspberry Pi services

DDI RPi comes with Kea DHCP running in default port 67, the powerDNS Authoritative and Recursive servers plus dnsdist running in port 53 suite and phpIPAM for the main DDI solution. Zabbix works as the monitoring server and both it and phpIPAM are hosted using Nginx with the IPAM Dashboard in default ports 80/443 for HTTP/HTTPS traffic and the Monitoring Dashboard in port 8081. Both dashboards are accessible to the system administrator only and is through the IPAM Dashboard that the hosts can be added, updated and removed in the network. The database schemas for the Authoritative Power DNS server, phpIPAM and Zabbix server are hosted in MariaDB. Mosquitto is the broker for MQTT running in default port 1883 and theSSH server runs in default port 22 so the system administrator can access the RPi DDI through it.

4.3 Access Control

The DDI RPi also acts as a Certificate Authority (CA), being capable of issuing certificates that control access to the other devices in the network by outside users. The access control process can be seen in figure 4.2.

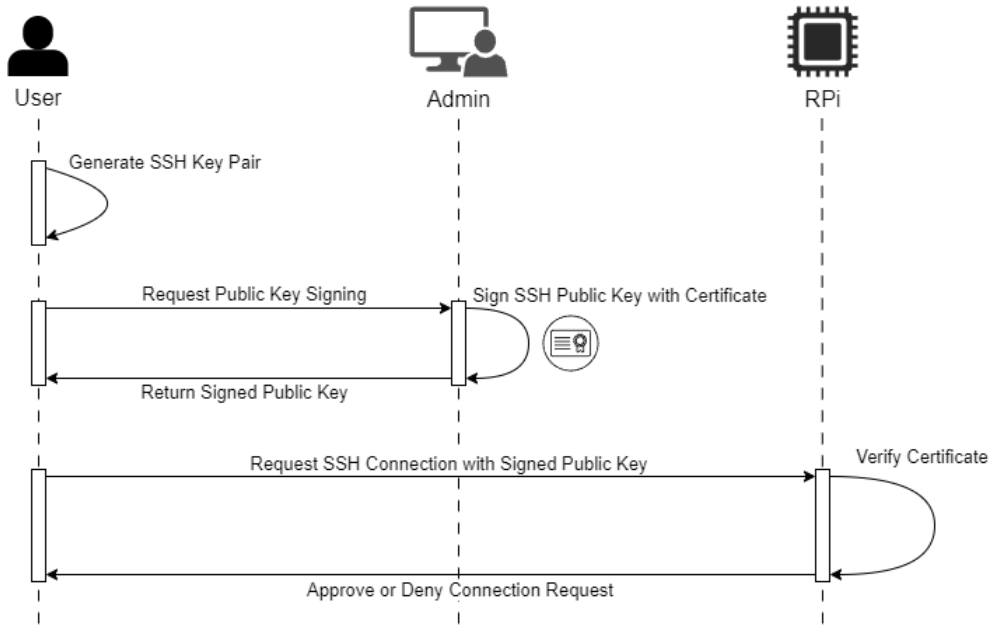


Figure 4.2: Access Control Sequence Diagram

In order for a user to access the processing node (RPi) in the network, he first needs to generate an SSH key pair. The public key of that key pair must then be signed by the Network Administrator/System Administrator with the CA and a new user created in the desired processing node. With the processing node connected to the network, it's SSH server should be automatically configured to accept connection requests with public keys signed by the networks CA.

4.4 Automation

With the services set up and running, the final implementation needed is to automate certain tasks to achieve the system described by the requirements. The following list contains the tasks that require automation:

- synchronization between phpIPAM, Kea, powerDNS records and Zabbix;
- probing new hosts when they request DHCP leases;
- setting up a password for new camera sensors;
- configuring new processing nodes SSH server to only allow SSH sessions with credentials signed by the CA;
- configure zabbix-agent in new processing nodes, enabling Zabbix server to collect monitoring data from those nodes.

The automation of these tasks will require scripts and those scripts will need to know important parameters:

- for phpIPAM:
  - API URL;
  - API session token;
- for Kea DHCP:
  - kea config file path;
  - cache file path;
- for powerDNS:
  - database host and name;
  - user and password;
  - DNS domain;
- for Zabbix:
  - API URL;
  - API session token;
  - monitoring items;
- for Ansible:
  - Ansible inventory file path;
- for camera:
  - password.

For automation regarding interaction with both phpIPAM and Zabbix it required their API URL in order to send requests to it and the API Token to authenticate the session, allowing the requests to be accepted by the services. Zabbix interaction uses an additional parameter for items to monitor in the hosts. This includes: CPU utilization, Load averages and number of running processes. In the case of Kea and Ansible, there are some files that may require writing to and/or reading from, such as the kea config file, the cache file with the last known IP/MAC reservations and the ansible inventory file. Updating records for powerDNS needs access to it's database hence the need to know where the database is being hosted, the database name and the credentials for access (user and password). The records also need to be updated with the correct domain name. Finally, the camera needs a password to be configured. All these parameters could be used by multiple scripts, so in order to centralize all these configuration parameters, the configuration file sp2-conf.ini was created at /etc/sp2 in RPi DDI. All the scripts written in Python access that file and read the parameters that they need for their execution.

4.4.1 Synchronization of Databases

In order to synchronize phpIPAM source of truth database with the Kea reservations configuration, powerDNS records database and Zabbix inventory database a python script was developed, capable of getting the phpIPAM database via API and using it to update the rest of the databases, as shown in the activity diagram of figure 4.3. The script runs every 2 mins using cron job, a functionality of Raspberry Pi OS that allows scheduling of tasks.

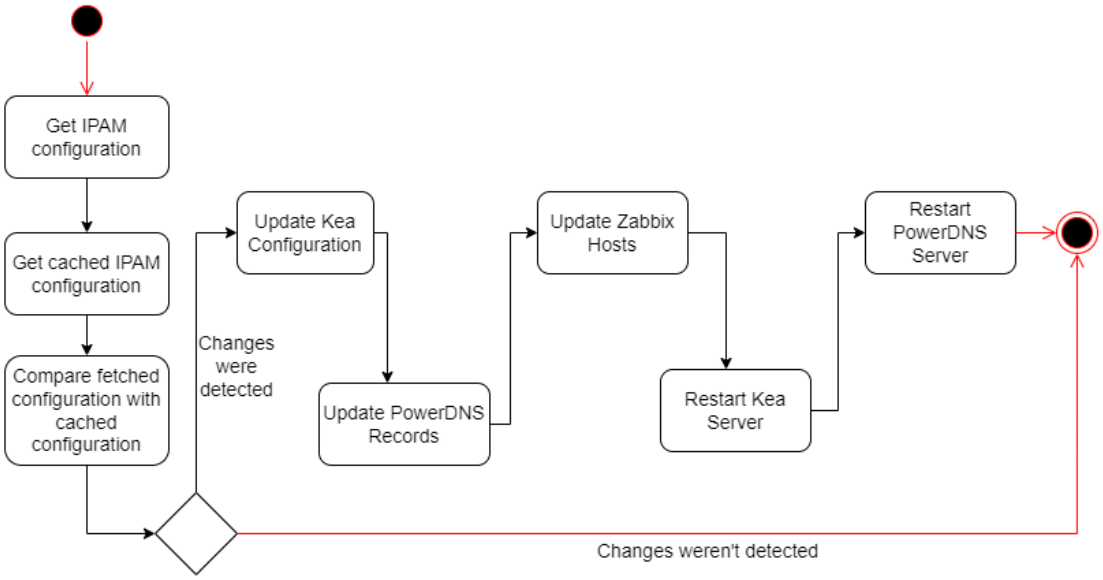


Figure 4.3: Activity Diagram for synchronizing services databases

The script starts by getting the IPAM configuration (source of truth) using phpIPAMs API. Following that up, it gets the previously cached IPAM configuration and compares it with the new one. If there is no cached IPAM configuration it will update the other services configurations and databases according with the most recently retrieved IPAM configuration. If there is a cached configuration, it will compare the retrieved IPAM configuration with the cached IPAM configuration and if changes are detected it will update the Kea reservations configuration, then the powerDNS Records, after that the Zabbix Hosts databse via the Zabbix API and finally it will restart the Kea and PowerDNS authoritative servers to ensure the changes take effect.

4.4.2 Host Discovery, Probing and Expiration

For probing new hosts when they request DHCP leases, there first needs to be a way to know there is a new host in the network requesting a DHCP lease. In order to determine that, the kea hooks functionality was used, which allows to run external scripts for specific packet processing hook points which in this case are the leases4_commited and lease4_expire hookpoints, as can be seen in the activity diagram in figure 4.4.

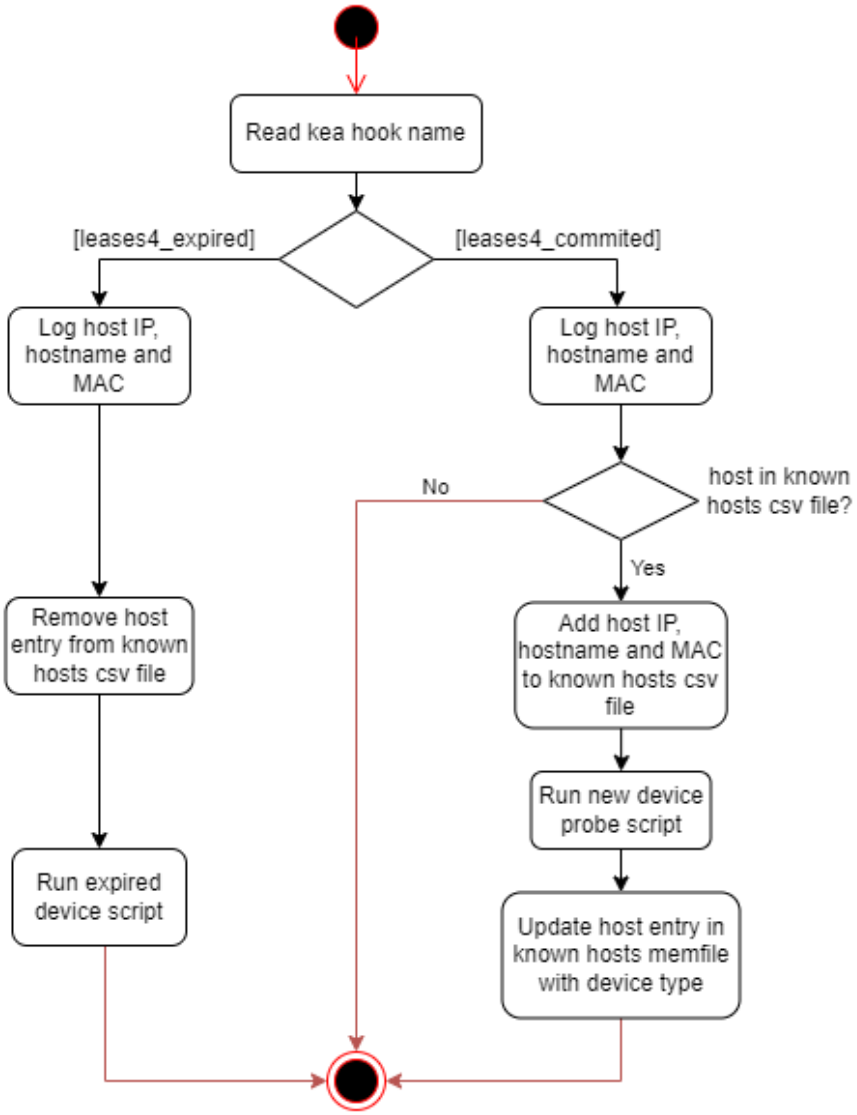


Figure 4.4: Activity Diagram for Kea Hooks script

In this case, the script was implemented in bash according to the kea documentation example. The script is always called by kea at one of the hook points and inherits all of the kea user privileges. When the script is called, the name of the hook point is passed as a parameter. Using this, the script can then determine which course of action to take: if it is going to analyze a committed lease (a new device has been discovered) or process the expired lease of a device (a device is down and no longer part of the network). In both branches the kea hooks script will log the host IP, hostname and MAC that are being analyzed and will call an external python script, either for a device whose DHCP lease expired or for probing a new device in the network. It will pass the device's hostname, IP and MAC addresses as arguments when calling those scripts.

The new device probing script, with the activity diagram as seen in figure 4.5, is responsible not only for detecting which type of device it is, but for configuring it as well in the case of it being a camera or updating the ansible inventory in the case of being a processor (Raspberry Pi).

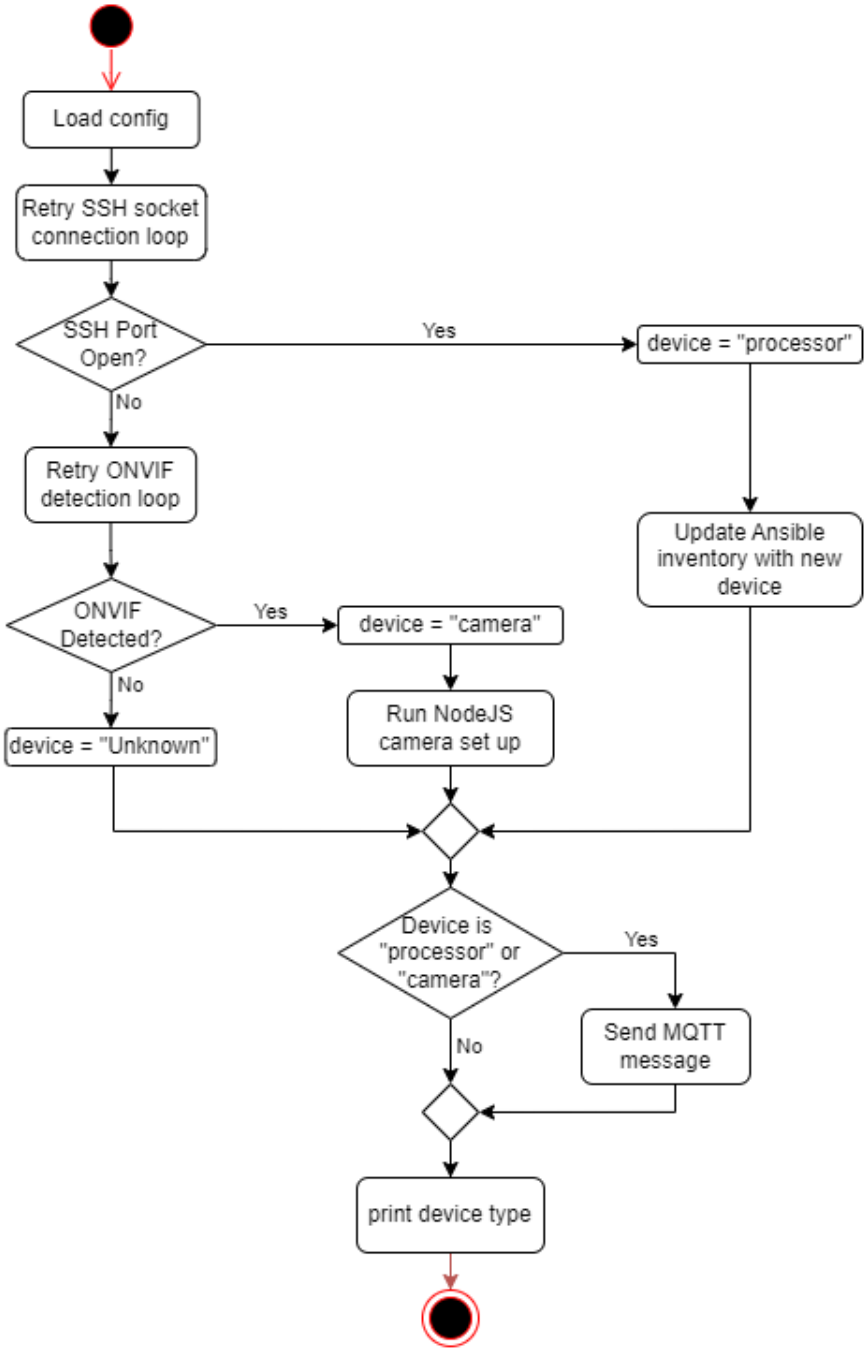


Figure 4.5: Activity Diagram for New Device Probing script

The probing script starts by loading the configuration from the set up main configuration file, retrieving important information for the scripts operation such as:

- the Ansible inventory path and the powerDNS domain in order to add the proper domain name of a discovered processor to the inventory file of Ansible for further automated tasks;
- the camera password to setup for the discovered camera sensor devices.

Once the probing script gets the proper configuration parameters, it starts by attempting a socket connection to the SSH port 22 of the provided IP through the command-line interface (CLI) arguments. It will retry that connection a total of 5 times. If it succeeds, it will add the proper domain name of the discovered processor to the inventory file of Ansible. If not, it will attempt HTTP connections to common ONVIF URL paths, retrying the connections to each URL path a total of 5 times. In case of positive ONVIF capabilities identification, the probing script will then attempt to configure the discovered Camera. In case the probing script wasn't successful in the identification of either a processing node or a camera sensor, it will mark the device as unknown. If the script positively identifies one of the two, it will send an MQTT message notifying about the new device discovered and it's type. At the end of the script, it will print the device type as either "unknown", "processor" or "camera". After the device probing script is finished, the kea hooks script gets it's printed result and uses it to update the known hosts csv file in order to keep track of all the devices and their types.

The expired device script is used primarily to send an MQTT message notifying about the down state of a host in the network, as can be seen in figure 4.6. In the case of the host also being in the Ansible inventory, it will be removed from it.

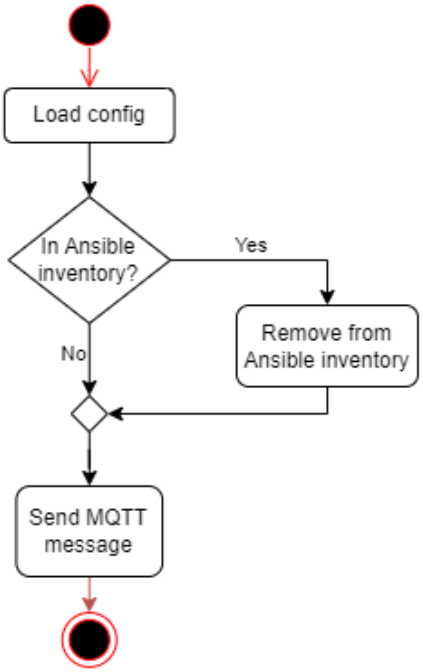


Figure 4.6: Activity Diagram for Expired Device script

4.4.3 MQTT Broker and Message Format

The mosquitto broker is configured to run on all interfaces of the RPi DDI. The MQTT message is JSON based and there are three possible formats, depending on the event causing the message and what device type it refers to. The messages are all published to the topic "network/host"

The first possible message is sent when a device stops renewing it's DHCP lease in the network. It contains 4 fields:

- **"event":"fallendevice" -** informs that the message contains information for a device that has disconnected from the network;
- **"hostname": device_hostname -** hostname of the device that has disconnected;
- **"ip": device_ip -** IP of the device that has disconnected;
- **"mac": device_mac -** MAC of the device that has disconnected.

Format example:

```
{
    "event": "fallendevice",
    "hostname": device_hostname,
    "ip": device_ip,
    "mac": device_mac
}
```

The second possible message is sent when a processing node is added to the network. This message contains 5 fields:

- **"event":"newdevice" -** informs that the message contains information for a device that has connected to the network;
- **"type": processor -** new field that specifies the type of device that connected to the network, in this case a processor (processing node);
- **"hostname": device_hostname -** hostname of the device that has connected;
- **"ip": device_ip -** IP of the device that has connected;
- **"mac": device_mac -** MAC of the device that has connected.

Format example:

```
{
    "event": "newdevice",
    "type": processor,
    "hostname": device_hostname,
    "ip": device_ip,
    "mac": device_mac
}
```

The third and last possible message is sent when a camera is added to the network. This message contains 6 fields:

- **"event":"newdevice" -** informs that the message contains information for a device that has connected to the network;
- **"type": camera -** specifies the type of device that connected to the network, in this case a camera sensor;
- **"hostname": device_hostname -** hostname of the device that has connected;
- **"ip": device_ip -** IP of the device that has connected;
- **"mac": device_mac -** MAC of the device that has connected;
- **"password": device_password -** new field that contains the password to access the "service" user of the camera.

Format example:

```
{
    "event": "newdevice",
    "type": camera,
    "hostname": device_hostname,
    "ip": device_ip,
    "mac": device_mac,
    "password": device_password
}
```

4.4.4 Ansible and Processing Node Configuration

The processing nodes require two configurations to be automated: the SSH server's and the Zabbix agent's. A cronjob set in the DDI RPi will periodically execute a playbook for each of the configurations.

SSH server configuration is important in order to implement access control to the processing nodes (RPis) in the network. The automated steps can be seen in figure 4.7

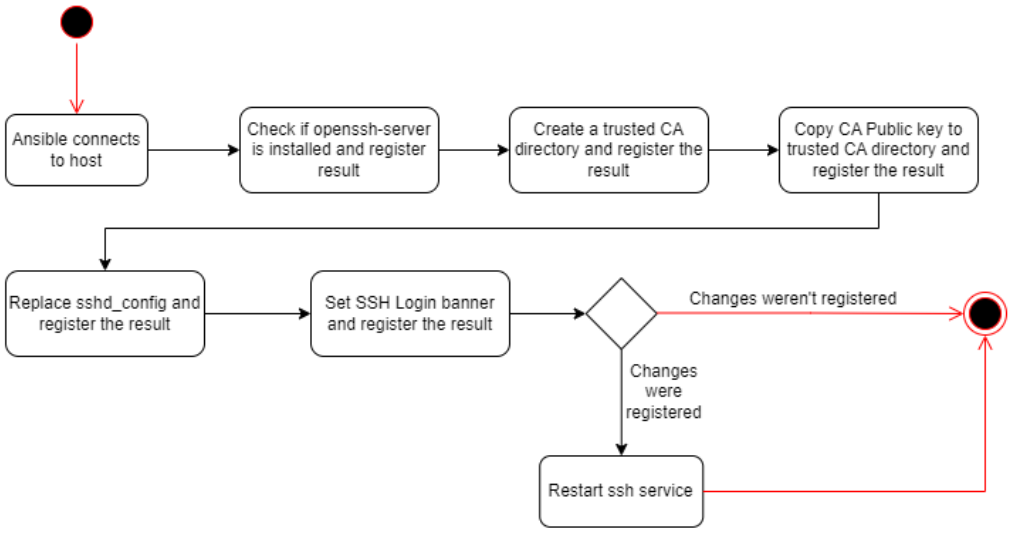


Figure 4.7: Activity Diagram for Ansible Playbook of SSH server set up

Once Ansible connects to the processing node it will first check if openssh-server is installed. This step can be skipped, since the processing nodes base image already comes with an SSH server. It will then create a trusted CA directory and copy the CA's public key into it. Next the SSH daemon config file (sshd_config) is replaced by a standard template with the desired settings and the SSH login banner is replaced by a custom one. Throughout these steps, Ansible will check if in fact a change was made, since the processing node could already be configured once it connected to the network and therefore no changes made. If no changes were made, Ansible will simply disconnect from the host. If changes were registered, it will restart the SSH server before disconnecting to apply the new settings.

The Zabbix Agent plays a key role in the processing node monitoring, since it is an agent responsible for acquiring information on the host and sending it back to the monitoring server in the central RPi (RPi DDI).

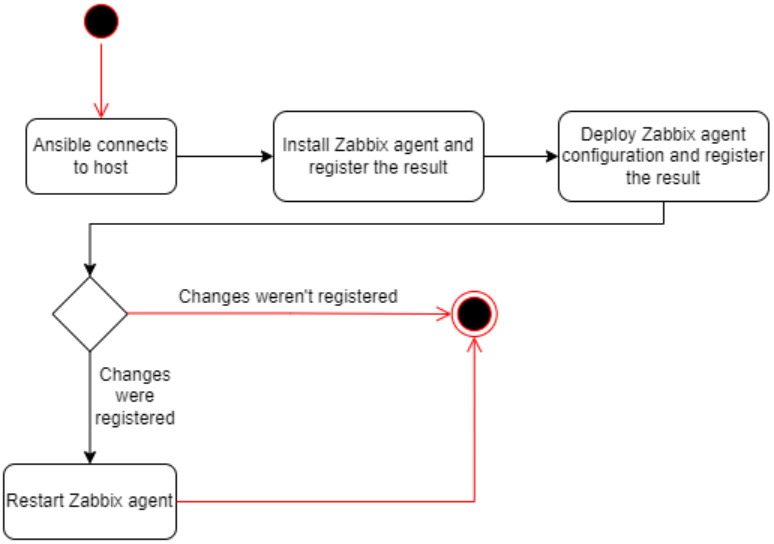


Figure 4.8: Activity Diagram for Ansible Playbook of Zabbix Agent set up

Ansible once again starts by connecting to the processing node and checking if the Zabbix agent is installed. Like for the SSH server configuration, this step can be skipped since the OS image comes with the Zabbix agent by default. Continuing the playbook execution, an updated template of the Zabbix agent configuration will be set in the processing node in order to deploy it. Throughout these steps, Ansible will check if in fact a change was made, since the processing node could already be configured once it connected to the network and therefore no changes made. If no changes were made, Ansible will simply disconnect from the host. If changes were registered, it will restart the Zabbix agent before disconnecting to apply the new settings.

4.4.5 Camera Configuration

In order to automate the setting of the camera sensor password figure 4.9 displays how NodeJS and a headless browser (puppeteer) is used to reverse engineer the camera's web API RCP implementation.

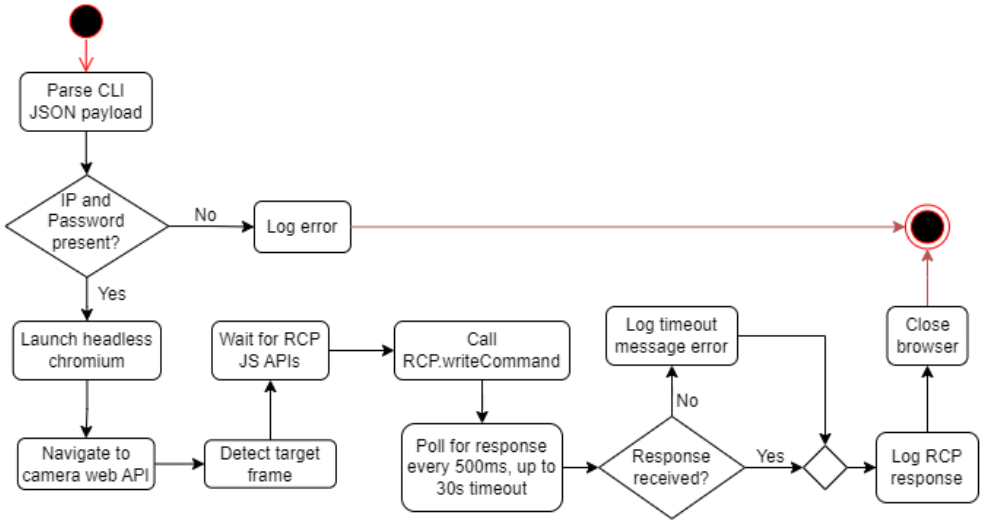


Figure 4.9: Activity Diagram for Camera Password Setup

The NodeJS script is called with a JSON payload passed through the CLI argument, so it starts by parsing that payload, containing the camera IP and the password to set. Following the argument parsing, it launches a headless browser in order to mimic standard human browser access to the web API and connects to it. Once the connection is established, it looks for the correct frame to interact with or falls back to the first frame. It polls until RCP javascript objects are available in order to send the password set command. Finally once the RCP objects are available it sends the configuration to set the password and awaits the result. Once the response is logged, it closes the headless browser and exits the program.

4.5 Host On-Boarding Work-flow

With the proposed system implemented, the manual labor for adding a new camera or processing node is greatly reduced. In the case of camera on-boarding it's only necessary to ensure the camera is factory reset so it doesn't have configurations set that can interfere with the system's work-flow. The only manual tasks for on-boarding a new camera are:

- factory resetting the camera (only if it was previously configured);
- reserving the IP/MAC address pair in IPAM Dashboard;
- defining a password to be set in the sp2 config file;
- collaboration between the maintenance and CV team regarding the camera optic settings;

In order to better understand the sequence of steps taken by the system to on-board the camera the figure 4.10 displays a sequence diagram of the steps taken by each participating component in RPi DDI.

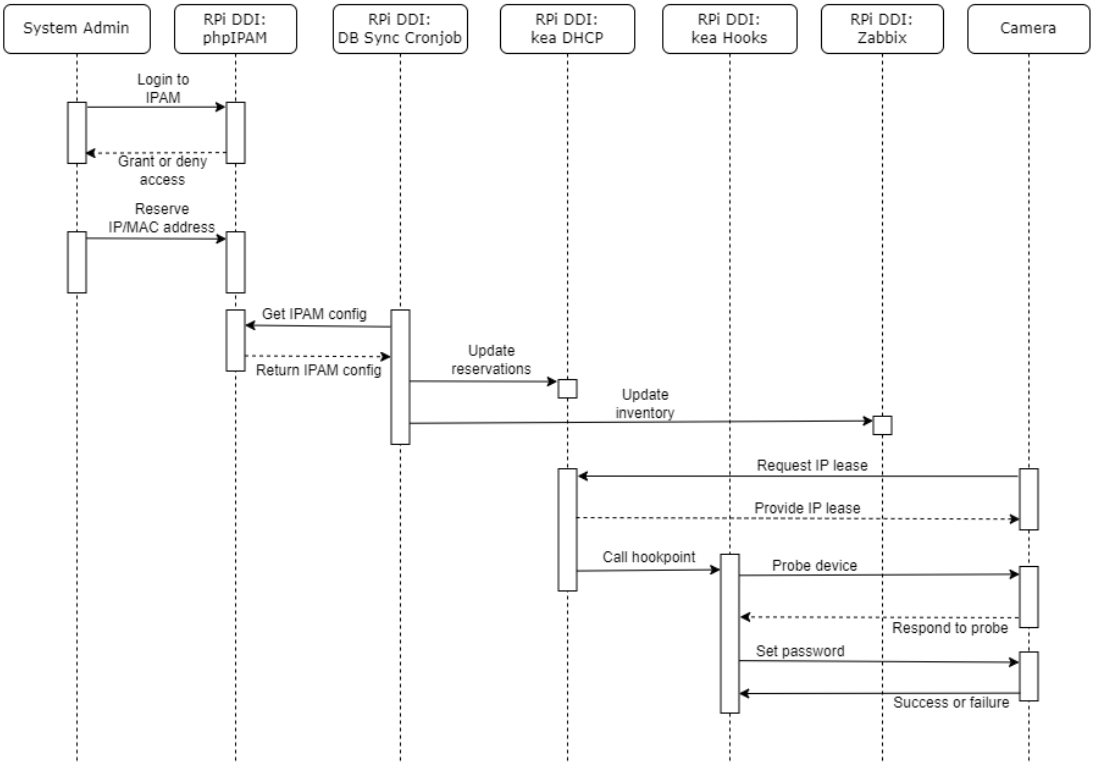


Figure 4.10: Sequence Diagram for implemented camera on-boarding work-flow

The System Admin starts by logging into phpIPAM to reserve the IP/MAC address. The periodic cronjob responsible for synchronizing the data bases then retrieves the IPAM config from phpIPAM and updates the respective data bases, including the Kea DHCP server reservations. Once the camera connects to the network, it will request an IP address through DHCP and the kea DHCP server will provide it. This in turn activates the kea hookpoint and executes the kea hooks script, leading to probing the new host (camera) to detect it's type. Once identified as a camera, a password for the user "service" is set, making the camera accessible using those credentials.

Similarly to the camera on-boarding, the processing node on-boarding only requires few manual tasks such as:

- flashing the RPi's microSD card with the base RPi OS image proposed;
- reserving the IP/MAC address pair in IPAM Dashboard.

The system entities involved in the processing node on-boarding are all the same as the ones for the camera on-boarding, however, there is an additional one - Ansible - as can be seen in the sequence diagram of figure 4.11.

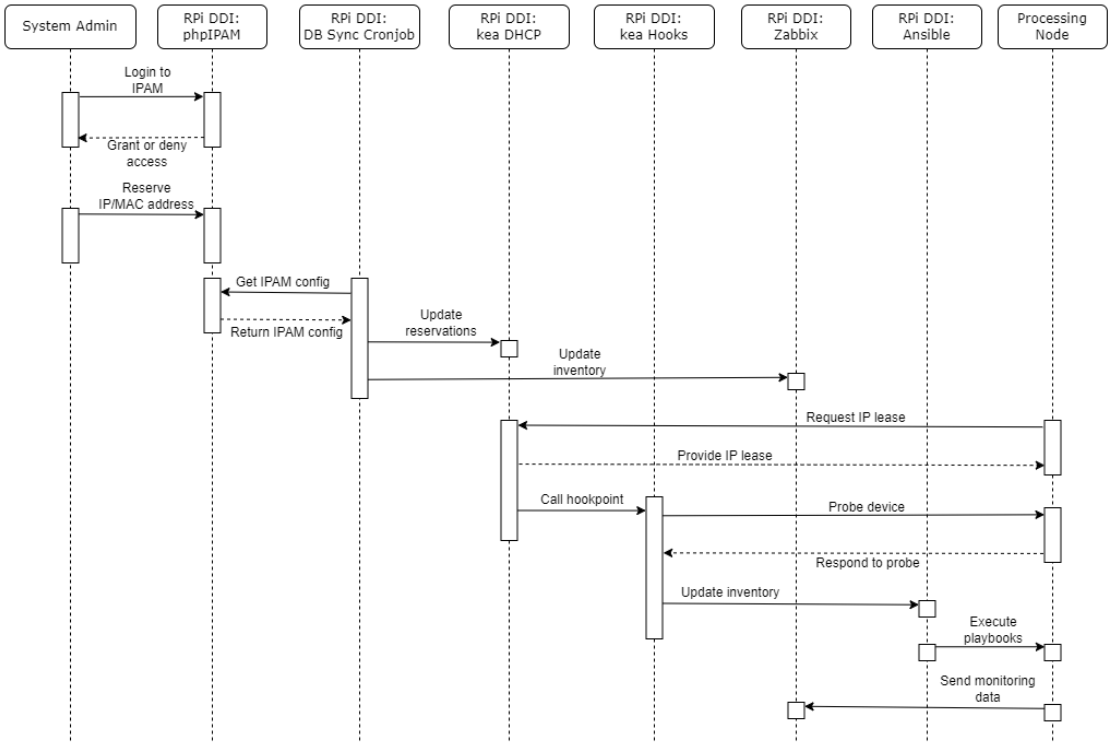


Figure 4.11: Sequence Diagram for implemented processing node on-boarding work-flow

The System Admin starts by logging into phpIPAM to reserve the IP/MAC address. The periodic cronjob responsible for synchronizing the data bases then retrieves the IPAM config from phpIPAM and updates the respective data bases, including the Kea DHCP server reservations. Once the processing node connects to the network, it will request an IP address through DHCP and the kea DHCP server will provide it. This in turn activates the kea hookpoint and executes the kea hooks script, leading to probing the new host (processing node) to detect it's type. Once identified as a processor, the Ansible inventory is updated and the cronjob that runs the ansible playbooks is executed. Upon the playbook finishing, the Zabbix agent configured in the processing node starts sending monitoring data to the Zabbix server in RPi DDI. The SSH server is also configure, making the processing node only accessible with public keys signed by the CA.

Chapter 5

Results and Discussion

5.1 Introduction

This chapter presents and discusses the results obtained from validating the implemented system. It begins by describing the laboratory set up used to develop, test and iterate on the solution. It then presents the functional testing methodology and the two functional test cases defined (see Appendix B) to validate the on-boarding work-flows for camera sensors and processing nodes. The evidence collected from executing these tests is then presented, including the Kea Hooks logs, the changes to the known hosts CSV file, the MQTT message logs, and the Ansible inventory changes. The chapter closes with an evaluation of the system, discussing its benefits as well as its current limitations.

The validation evidence acquired for the functional tests consists of:

- Kea Hooks log;
- Known Hosts csv file changes;
- MQTT Message log;
- Ansible inventory file changes.

5.2 Laboratory Set up

For the laboratory set up, as seen in figure 5.1, the subnet 10.227.16.224/28 was created, allowing for a total number of 14 usable hosts (10.227.16.225 - 10.227.16.238) and a broadcast address of 10.227.16.239. Address 10.227.16.225 was taken by the IT PO Switch, responsible for routing traffic into and out of the created subnet.

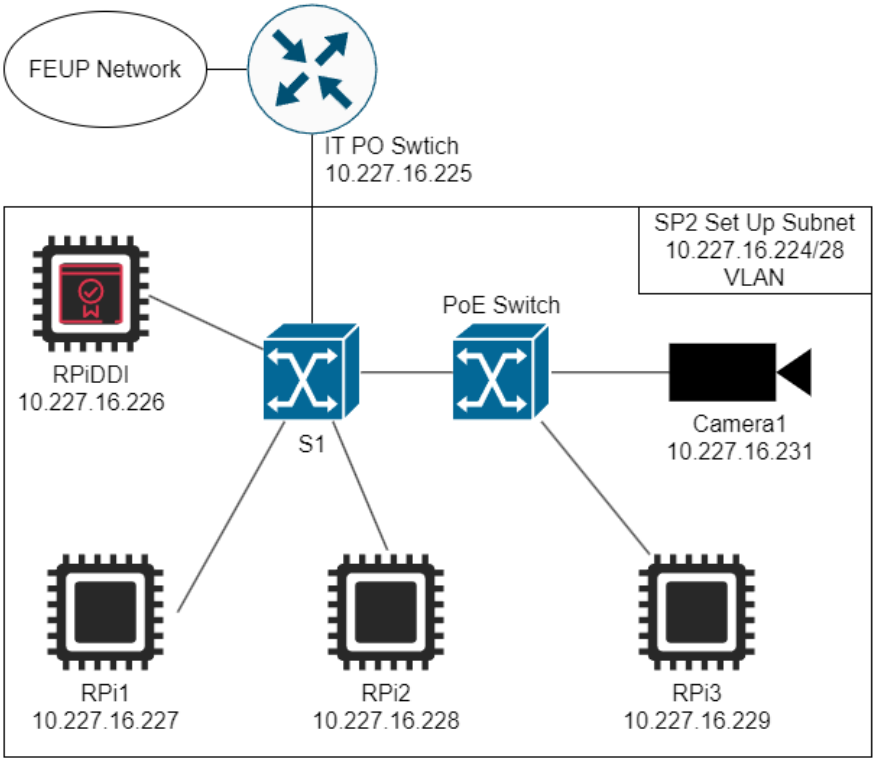


Figure 5.1: Laboratory Network Set Up with two plug and play switches

The initial subnet consisted of:

- 4 Raspberry Pi 5 (RPiDDI, RPi1, RPi2 and RPi3);
- 1 camera Bosch DINION IP Starlight 6000 (Camera 1);
- 1 TL-SG1005D TP-Link Switch (S1);
- 1 TL-SG1005P TP-Link PoE Switch (PoE Switch).

Later the laboratory set up was upgraded as can be seen in figure 5.2 for improved collaboration with other teams in the project and easier transport from the laboratory to the factory.

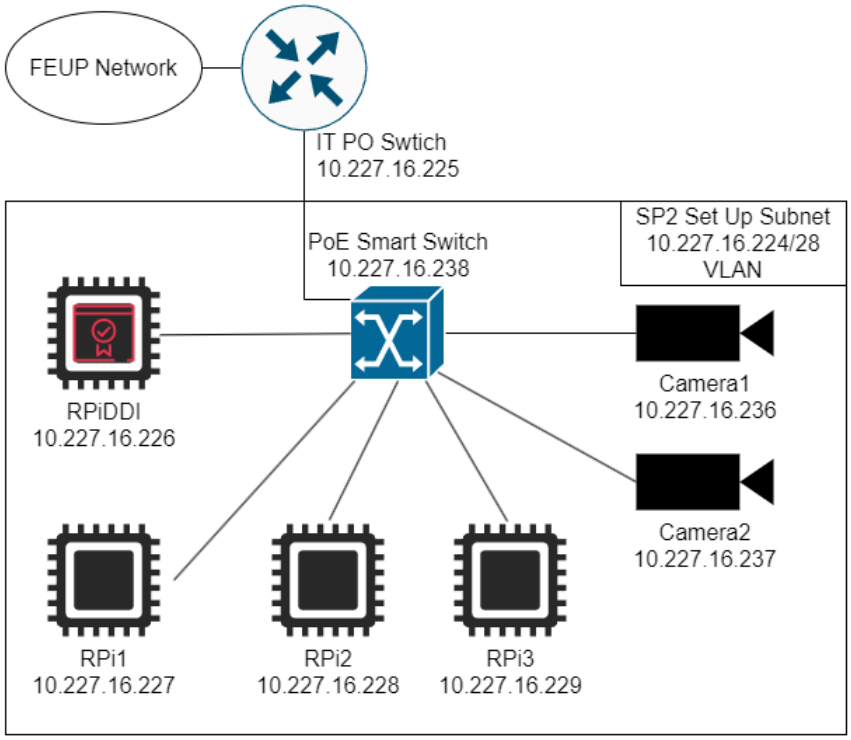


Figure 5.2: Laboratory Network Set Up with PoE Smart Switch

For this new set up, the S1 and PoE Switch were replaced by the PoE TP-Link Smart Switch model TL-SG2210MP and the Bosch DINION IP ultra 8000 (Camera 2) was added. Another variation of this network as displayed in figure 5.3 removes the connection between the PoE Smart Switch and the IT PO Switch, making the network totally isolated from traffic outside of it. In order to interact with the devices inside of it a Next Unit of Computing (NUC) mini computer was added. This was used to run the functional test for Camera On-Boarding.

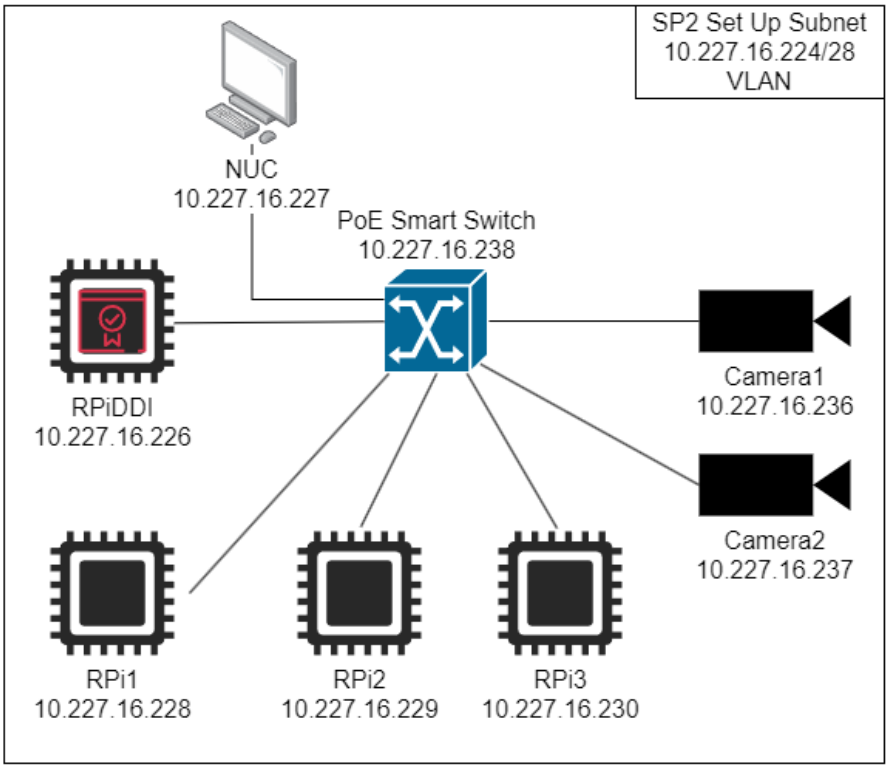


Figure 5.3: Isolated Laboratory Network Set Up

5.3 Functional Testing

Functional testing aims at validating two important host on-boarding workflows in the autonomic network:

- camera on-boarding;
- processing node on-boarding.

For camera on-boarding functional testing, the Functional Test Case FT-NHOB-001 of table B.1 was created and executed step by step. The environment used for this test was the isolated network with the PoE Smart Switch set up of figure 5.3, mimicking the factory production environment. The only devices in the subnet network initially are the PoE Smart Switch and RPi DDI.

For processing node on-boarding functional testing, the Functional Test Case of table B.2 in was created and executed step by step. It used a different environment of a network with the PoE Smart Switch connected to the IT PO Switch as seen in figure 5.2, being able to route internet traffic. The only devices in the subnet network initially are the PoE Smart Switch and RPi DDI. This is due to the need to install the Zabbix Agent package through the internet in order to properly configure the processing node for monitoring.

5.4 Kea Hooks log

The Kea Hooks logs output the kea hook points handlers activity, identifying devices hostnames, IP and MAC addresses, along with information about it's type and what changes are being made to the known hosts csv file. Log lines are numbered for easier interpretation and explanation.

5.4.1 FT-NHOB-001

The following terminal output presents the logs generated by the Kea Hooks in the context of the functional test FT-NHOB-001 (camera on-boarding):

```
1  Jan 02 21:38:19 RPiDDI kea-hooks[241809]: leases4_committed: LEASES4_SIZE
       =1
2  Jan 02 21:38:19 RPiDDI kea-hooks[241826]: lease4_renew: ip=10.227.16.238
       hostname=smartswitch hwaddr=1c:3b:f3:93:b6:25
3  Jan 02 21:38:19 RPiDDI kea-hooks[241830]:   -> lease[0]: ip=10.227.16.238
       hostname=smartswitch mac=1c:3b:f3:93:b6:25
4  Jan 02 21:38:19 RPiDDI kea-hooks[241842]:   -> host 10.227.16.238 /
       smartswitch / 1c:3b:f3:93:b6:25 already known, ignoring.
5  Jan 02 21:38:28 RPiDDI kea-hooks[241870]: leases4_committed: LEASES4_SIZE
       =1
6  Jan 02 21:38:28 RPiDDI kea-hooks[241888]:   -> lease[0]: ip=10.227.16.236
       hostname=camera1 mac=00:07:5f:af:d3:4d
7  Jan 02 21:38:28 RPiDDI kea-hooks[241910]: Added host -> ip=10.227.16.236
       hostname=camera1 hwaddr=00:07:5f:af:d3:4d type=unknown
8  Jan 02 21:38:34 RPiDDI kea-hooks[242076]: leases4_committed: LEASES4_SIZE
       =1
9  Jan 02 21:38:34 RPiDDI kea-hooks[242090]: lease4_renew: ip=10.227.16.238
       hostname=smartswitch hwaddr=1c:3b:f3:93:b6:25
10 Jan 02 21:38:34 RPiDDI kea-hooks[242095]:   -> lease[0]: ip=10.227.16.238
       hostname=smartswitch mac=1c:3b:f3:93:b6:25
11 Jan 02 21:38:34 RPiDDI kea-hooks[242107]:   -> host 10.227.16.238 /
       smartswitch / 1c:3b:f3:93:b6:25 already known, ignoring.
12 Jan 02 21:38:43 RPiDDI kea-hooks[242119]: leases4_committed: LEASES4_SIZE
       =1
13 Jan 02 21:38:43 RPiDDI kea-hooks[242137]: lease4_renew: ip=10.227.16.236
       hostname=camera1 hwaddr=00:07:5f:af:d3:4d
14 Jan 02 21:38:43 RPiDDI kea-hooks[242140]:   -> lease[0]: ip=10.227.16.236
       hostname=camera1 mac=00:07:5f:af:d3:4d
15 Jan 02 21:38:43 RPiDDI kea-hooks[242152]:   -> host 10.227.16.236 /
       camera1 / 00:07:5f:af:d3:4d already known, ignoring.
16 Jan 02 21:38:49 RPiDDI kea-hooks[242163]: leases4_committed: LEASES4_SIZE
       =1
17 Jan 02 21:38:49 RPiDDI kea-hooks[242178]: lease4_renew: ip=10.227.16.238
       hostname=smartswitch hwaddr=1c:3b:f3:93:b6:25
```

```
18 Jan 02 21:38:49 RPiDDI kea-hooks[242183]:   -> lease[0]: ip=10.227.16.238
       hostname=smartswitch mac=1c:3b:f3:93:b6:25
19 Jan 02 21:38:49 RPiDDI kea-hooks[242195]:   -> host 10.227.16.238 /
       smartswitch / 1c:3b:f3:93:b6:25 already known, ignoring.
20 Jan 02 21:38:51 RPiDDI kea-hooks[242196]: Detected device type for 00:07:5
       f:af:d3:4d: camera
21 Jan 02 21:38:51 RPiDDI kea-hooks[242208]: Updated host type -> ip
       =10.227.16.236 hostname=camera1 hwaddr=00:07:5f:af:d3:4d type=camera
```

The log can be overwhelming at first but interpretation is simplified if all the lines that do not include data about the added camera like it's host name (camera1), MAC address (00:07:5f:af:d3:4d) and/or IP address (10.227.16.236) are ignored, and so can be the lease4_renew log lines. This means focus needs to be on the lines 6, 7, 15, 20 and 21. Line 6 is the first instance of a committed DHCP IP lease to the new camera and line 7 logs the addition of the new host to the known hosts csv file and all the information known about the device so far. The type is still unknown since the probing script is still running at this time and still hasn't returned a result. Line 15 ignores the camera since it's either already being or has been probed. Finally line 20 logs the result of the probing script, being it the successful identification of the device as a camera and on line 21 we see another lease commit to the camera and this time the type is properly identified.

5.4.2 FT-NHOB-002

The following terminal output presents the logs generated by the Kea Hooks in the context of the functional test FT-NHOB-002 (processing node on-boarding):

```
1 Jan 02 22:04:39 RPiDDI kea-hooks[248487]: lease4_renew: ip=10.227.16.238
      hostname=smartswitch hwaddr=1c:3b:f3:93:b6:25
2 Jan 02 22:04:39 RPiDDI kea-hooks[248493]:   -> lease[0]: ip=10.227.16.238
      hostname=smartswitch mac=1c:3b:f3:93:b6:25
3 Jan 02 22:04:39 RPiDDI kea-hooks[248505]:   -> host 10.227.16.238 /
      smartswitch / 1c:3b:f3:93:b6:25 already known, ignoring.
4 Jan 02 22:04:49 RPiDDI kea-hooks[248518]: leases4_committed: LEASES4_SIZE
      =1
5 Jan 02 22:04:49 RPiDDI kea-hooks[248539]:   -> lease[0]: ip=10.227.16.229
      hostname=rpi-processor-1 mac=2c:cf:67:c7:ac:60
6 Jan 02 22:04:49 RPiDDI kea-hooks[248561]: Added host -> ip=10.227.16.229
      hostname=rpi-processor-1 hwaddr=2c:cf:67:c7:ac:60 type=unknown
7 Jan 02 22:04:49 RPiDDI kea-hooks[248563]: Detected device type for 2c:cf
      :67:c7:ac:60: processor
8 Jan 02 22:04:49 RPiDDI kea-hooks[248575]: Updated host type -> ip
      =10.227.16.229 hostname=rpi-processor-1 hwaddr=2c:cf:67:c7:ac:60 type=
      processor
9 Jan 02 22:04:54 RPiDDI kea-hooks[248584]: leases4_committed: LEASES4_SIZE
      =1
```

Same as with the kea hooks logs for the camera, the focus for these logs need to be on the messages that address the processing node hostname (rpi-processor-1), MAC address (2c:cf:67:c7:ac:60) and/or IP address (10.227.16.229), thus the lines 5, 6, 7 and 8. Line 5 is the first instance of a committed DHCP IP lease to the new processing node and line 6 logs the addition of the new host to the known hosts csv file and all the information known about the device so far. The type is still unknown since the probing script is still running at this time and still hasn't returned a result. Finally line 7 logs the result of the probing script, being it the successful identification of the device as a processor and on line 8 we see another lease commit to the processing node and this time the type is properly identified.

5.5 Known Hosts csv file Changes

The Known Hosts csv file keeps track of the current devices connected to the network and is edited by the kea hooks script. Each line contains information about the connected hosts such as their IP, hostname, MAC and type (camera or processor).

5.5.1 FT-NHOB-001

Known hosts csv file before adding the camera:

```
raspi@RPiDDI:~ $ sudo cat /var/lib/kea/known-hosts.csv
10.227.16.238,smartswitch,1c:3b:f3:93:b6:25,unknown
```

In the terminal output above, the only known host is the PoE smart switch, indicating the only hosts in the subnet network are the RPi DDI and the PoE Smart Switch.

Known hosts csv file after adding the camera:

```
raspi@RPiDDI:~ $ sudo cat /var/lib/kea/known-hosts.csv
10.227.16.238,smartswitch,1c:3b:f3:93:b6:25,unknown
10.227.16.236,camera1,00:07:5f:af:d3:4d,camera
```

After the camera connects, the known hosts csv file is updated with a line entry for the new host connected to the network. There we can see the camera's:

- **hostname -** camera1;
- **IP -** 10.227.16.236;
- **MAC -** 00:07:5f:af:d3:4d;
- **type -** camera.

5.5.2 FT-NHOB-002

Known hosts csv file before adding the processing node:

```
raspi@RPiDDI:~ $ sudo cat /var/lib/kea/known-hosts.csv
10.227.16.238,smartswitch,1c:3b:f3:93:b6:25,unknown
```

Just like in the previous functional test, the terminal output above contains only one known host line for the PoE smart switch.

Known hosts csv file after adding the processing node:

```
raspi@RPiDDI:~ $ sudo cat /var/lib/kea/known-hosts.csv
10.227.16.238,smartswitch,1c:3b:f3:93:b6:25,unknown
10.227.16.229,rpi-processor-1,2c:cf:67:c7:ac:60,processor
```

After the processing node connects, the known hosts csv file is updated with a line entry for the new host connected to the network. There we can see the camera's:

- **hostname -** rpi-processor-1;
- **IP -** 10.227.16.229;
- **MAC -** 2c:cf:67:c7:ac:60;
- **type -** processor.

5.6 MQTT Message log

MQTT messages are only sent when a device connects or disconnects from the network. By having a listener subscribed to the "network/host" topic we can see the messages being sent from the RPi DDI.

5.6.1 FT-NHOB-001

MQTT message log:

```
raspi@RPiDDI:~ $ sudo mosquitto_sub -h 10.227.16.226 -t network/host -v
network/host {"event": "newdevice", "type": "camera", "hostname": "camera1
    ", "ip": "10.227.16.236", "mac": "00:07:5f:af:d3:4d", "password": "
    aDV5CQtxp7ws)"}
```

In the terminal output we can see a JSON sent in the "network/host" topic. The message is for a new connected device as can be confirmed by the field "event": "newdevice" and the device is a camera as can be confirmed by the field "type: camera". The following fields contain the camera information such as:

- **hostname -** camera1;
- **IP -** 10.227.16.236;
- **MAC -** 00:07:5f:af:d3:4d;
- **password -** aDV5CQtxp7ws).

5.6.2 FT-NHOB-002

MQTT message log:

```
raspi@RPiDDI:~ $ sudo mosquitto_sub -h 10.227.16.226 -t network/host -v
network/host {"event": "newdevice", "type": "processor", "hostname": "rpi-
    processor-1", "ip": "10.227.16.229", "mac": "2c:cf:67:c7:ac:60"}
```

In the terminal output we can see a JSON sent in the "network/host" topic. The message is for a new connected device as can be confirmed by the field "event": "newdevice" and the device is a processing node as can be confirmed by the field "type: processor". The following fields contain the processing node's information such as it's:

- **hostname -** rpi-processor-1;
- **IP -** 10.227.16.229;
- **MAC -** 2c:cf:67:c7:ac:60.

5.7 Ansible Inventory Changes

The ansible inventory file contains the list of devices where Ansible will execute it's playbooks.

Ansible inventory before adding the processing node:

```
raspi@RPiDDI:~ $ cat ansible/ansible-project/inventory.ini
[control]
localhost ansible_connection = local

[processors]
```

As can be seen above, the processors section is empty due to the fact no processing node is currently connected to the network.

Ansible inventory after adding the processing node:

```
raspi@RPiDDI:~ $ cat ansible/ansible-project/inventory.ini
[control]
localhost ansible_connection = local

[processors]
rpi-processor-1.sp2.lan
```

After a processing node is connected to the network, the processors section has a new entry with the fully qualified domain name for the processing node with the hostname "rpi-processor-1". Ansible can use this fully qualified domain name to connect and execute the playbooks in the processor and configure it.

5.8 System Evaluation

The implemented system brings notable advantages. The Network Management DDI stack provides a unified interface for managing the available IP address space, making management easier, more reliable, resilient and auditable. In combination with the automated devices configuration, scalability is increased due to the easier on-boarding of devices and security is improved due to the automatic configurations enforcing the access control policy introduced with the system. Finally, the periodicity of the ansible playbooks instead of single event executions on processing node connection, ensure the devices configuration will always be the desired one even if manual changes are made, as long as those configurations do not affect the SSH server in a way that it doesn't allow the DDI RPi connections, blocking ansible from executing it's playbooks.

Despite it's benefits, the system still has room for improvement. The automatic tasks related to the databases synchronization are periodic due to the use of cronjobs, instead of event driven when the system administrator makes changes in the IPAM Dashboard. This affects the latency of the database synchronization updates since the changes could happen right after a polling cycle it will have to wait almost a full period. An event driven approach would execute the synchronization immediately after the change is made. However the low latency in this matter isn't a high criticality requirement so this would be a low priority improvement.

Another more critical latency issue is to do with how the system detects a host disconnecting. While trying to take advantage of kea's DHCP lease management and hookpoints a major flaw in this strategy was overlooked: the leases can take a very long time to expire making the time of host being down detection extremely long. This could negatively impact the orchestrators operation by

giving it an inaccurate list of available devices to orchestrate. A way to make it take less time is to shorten the lease time as was done in the implementation of the system, however it will cause the lease renewals to be more frequent. For networks with a small number of devices the effects can be neglected while for larger networks could lead to slower data transmission, packet loss and delays. This negatively impacts the scalability of the system. A more reliable fix would be to take advantage of the Zabbix Monitoring server and trigger an event when pinging a device fails after a given amount of tries.

Regarding host discovery and capability determination, not using WS-Discovery was a missed opportunity to simplify the management and monitoring system's development and improve it's architecture. Since the camera sensor devices already come with ONVIF incorporated and by default WS-Discovery, the processing node's OS image could also come with a service that runs the protocol as well. This way there would be a way for the devices to actively announce themselves and their capabilities without the RPi DDI actively probing them and without the need to use kea DHCP hookpoints. Instead, RPi DDI could have a running passive listener that would receive the Hello messages sent from the devices once they joined the network, process them to determine the devices capabilities and communicate them through MQTT to the orchestrator.

The method for the camera's password configuration is very complex and inefficient, requiring a lot of additional software (NodeJS, additional libraries and the puppeteer headless browser) for the reverse engineering. It also makes it harder to add more automated configuration options such as the camera's PTZ, zoom, focus and even SNMP for monitoring. Implementing an actual protocol solution based on the camera's defined RCP would be extremely beneficial for simplifying the camera's password configuration and introducing more autonomous configuration options.

Chapter 6

Conclusions

The primary motivation for this dissertation was clear: the manual effort required to on-board camera sensors and processing nodes into an IIoT network is costly, error-prone, and incompatible with the scalability demands of modern industrial environments. In the specific case of the Digital MilkRun Application, this manual effort was compounded by the complete absence of a network management or monitoring infrastructure, with IP addresses tracked in a spreadsheet and every device configured by hand.

Revisiting the objectives defined in Chapter 1, this dissertation designed, implemented and validated a network management and monitoring system suited for deployment in the Digital MilkRun Application's IIoT infrastructure; streamlined the on-boarding process for both camera sensors and processing nodes; introduced access control for processing nodes based on public key cryptography and a certificate authority; and, in doing so, contributed the network layer required to support the orchestrator being developed within the broader Sensitive Industry project.

By combining a centralized DDI stack (Kea DHCP, PowerDNS, phpIPAM) with a monitoring platform (Zabbix), a certificate-based access control mechanism, and both periodic and event-driven automation built around Kea hooks, Ansible and Python scripting, the system successfully eliminates the manual configuration steps that previously constrained the on-boarding of new devices. The functional tests FT-NHOB-001 and FT-NHOB-002, executed in a laboratory set-up mirroring the Digital MilkRun Application's network topology, demonstrated that both camera sensors and processing nodes can be automatically detected, identified, configured, and reported to an orchestrator upon joining the network, with no human intervention beyond the initial physical connection and IP reservation in phpIPAM. This validates the core autonomic property targeted by this work – self-configuration – and the Maturity Level 1 of Monitoring Management.

Beyond the technical contribution, the resulting system is expected to reduce the on-boarding time and error rate for new devices at the Bosch Ovar factory, directly supporting the productivity and technological-upgrading goals expressed in SDG 8.2, which this project was framed against.

At the same time, the System Evaluation in Chapter 5 identified concrete limitations of the current implementation – namely around synchronization latency, host-down detection time, and the complexity of the camera on-boarding automation – which are addressed as directions for future work below.

6.1 Future Work

Although the implemented system successfully executes the on-boarding work-flow for both device types, the evaluation carried out in Chapter 5 exposed several areas where the design could be improved. Future work should prioritize the following:

- **Event-driven database synchronization -** The synchronization between phpIPAM, Kea, PowerDNS and Zabbix currently runs on a fixed polling interval via cron job. This introduces avoidable latency between a change made by the system administrator in the IPAM Dashboard and its propagation to the rest of the system. Triggering synchronization directly from phpIPAM (e.g. via a webhook on record changes) would reduce this latency to near-zero without the overhead of constant polling. phpIPAM may need to be replaced by another IPAM solution with the needed functionalities;
- **Faster and more reliable host-down detection -** Relying on DHCP lease expiration to detect a disconnected host proved to be a significant limitation: lease times must be kept short to achieve timely detection, but doing so increases lease renewal traffic and could hurt scalability in larger deployments. A more robust approach would use Zabbix's active polling capabilities to trigger a "host down" event after a configurable number of failed pings, decoupling detection latency from DHCP lease configuration entirely;
- **Adoption of WS-Discovery for host discovery and capability determination -** The current probing approach, triggered through Kea hook points, actively attempts SSH and ONVIF connections to classify new devices. Since ONVIF-compliant camera sensors already implement WS-Discovery, extending the processing node's base OS image with a WS-Discovery service would allow all devices to announce their presence and capabilities proactively. The DDI Raspberry Pi could then passively listen for these announcements, simplifying the architecture and removing the dependency on DHCP hook points for discovery;
- **Native implementation of the camera's RCP -** The current password-configuration mechanism relies on a headless browser (Puppeteer) to reverse-engineer the camera's web API, which is fragile and adds unnecessary software dependencies. Implementing the RCP protocol natively would simplify this automation considerably and open the door to automating additional camera parameters, such as PTZ, focus, zoom, and SNMP-based monitoring, which are currently configured manually by the maintenance and CV teams;
- **Extending access control to camera sensors -** Access control in the current system is limited to processing nodes via SSH and the certificate authority. Extending an equivalent

mechanism, or at least stronger authentication, to camera sensors would close a security gap and align the whole device fleet with the same access control policy;

- **Increase autonomic properties and maturity levels -** The autonomic properties of the Digital MilkRun Application could be expanded beyond self-configuration with the integration of the orchestrator for self-optimization. For self-healing and self-protection, the deployment of other software/Artificial Intelligence (AI) solutions in RPi DDI should be studied. The introduction of AI should also be explored for building up the maturity level all the way to Autonomic Management;

- **Validation at production scale -** The system was validated in a small-scale laboratory environment with a limited number of hosts. Future work should evaluate its behavior under the device density and network conditions expected at the Bosch Ovar factory, particularly regarding the scalability concerns raised by short DHCP lease times and the periodic Ansible playbook executions;

- **Integration testing with the orchestrator -** As the orchestrator itself falls outside the scope of this dissertation, end-to-end integration testing between the implemented system and a functioning orchestrator remains an important next step to confirm that the MQTT messages and monitoring data provided are sufficient for the orchestrator's decision-making needs.

References

[1] D. Kumar, R. Kr Singh, R. Mishra, and S. Fosso Wamba, "Applications of the internet of things for optimizing warehousing and logistics operations: A systematic literature review and future research directions," *Computers & Industrial Engineering*, vol. 171, p. 108 455, 2022, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2022.108455 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835222004892

[2] A. A. Tubis and J. Rohman, "Intelligent warehouse in industry 4.0—systematic literature review," *Sensors*, vol. 23, no. 8, 2023, ISSN: 1424-8220. DOI: 10.3390/s23084105 [Online]. Available: https://www.mdpi.com/1424-8220/23/8/4105

[3] C. Lee, Y. Lv, K. Ng, W. Ho, and K. Choy, "Design and application of internet of things-based warehouse management system for smart logistics," *International Journal of Production Research*, vol. 56, no. 8, pp. 2753–2768, 2018. DOI: 10.1080/00207543.2017.1394592 eprint: https://doi.org/10.1080/00207543.2017.1394592. [Online]. Available: https://doi.org/10.1080/00207543.2017.1394592

[4] Y. Wu, H.-N. Dai, and H. Wang, "Convergence of blockchain and edge computing for secure and scalable iiot critical infrastructures in industry 4.0," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2300–2317, 2021. DOI: 10.1109/JIOT.2020.3025916

[5] Z. Movahedi, M. Ayari, R. Langar, and G. Pujolle, "A survey of autonomic network architectures and evaluation criteria," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 2, pp. 464–490, 2012. DOI: 10.1109/SURV.2011.042711.00078

[6] M. H. Behringer et al., *Autonomic Networking: Definitions and Design Goals*, RFC 7575, Jun. 2015. DOI: 10.17487/RFC7575 [Online]. Available: https://www.rfc-editor.org/info/rfc7575

[7] T. Rooney, *Dhcp-dns-ipam (ddi) automation*. Accessed: Jul. 27, 2025. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://assets.ctfassets.net/5vdtaoxce4nw/7M5oXCIQmcNbk54gLaSv3D/0f6f645cdfb36da270WP-DHCP-DNS-IPAM_DDI_Automation.pdf

[8] INL, *Sensitive industry - inl*. Accessed: Jul. 27, 2026. [Online]. Available: https://inl.int/projects/bosch-sensitive-industry/

[9] I. Butun, *Industrial IoT Challenges, Design Principles, Applications, and Security*. Springer Nature Switzerland AG, 2020, pp. 20–34. DOI: 10.1007/978-3-030-42500-5

[10] S. Munirathinam, "Chapter six - industry 4.0: Industrial internet of things (iiot)," in *The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases*, ser. Advances in Computers 1, P. Raj and P. Evangeline, Eds., vol. 117, Elsevier, 2020, pp. 129–164. DOI: https://doi.org/10.1016/bs.adcom.2019.10.010

[Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245819300634

[11] N. Samaan and A. Karmouch, "Towards autonomic network management: An analysis of current and future research directions," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 3, pp. 22–36, 2009. DOI: 10.1109/SURV.2009.090303

[12] N. Samaan and A. Karmouch, "Towards autonomic network management: An analysis of current and future research directions," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 3, pp. 22–36, 2009. DOI: 10.1109/SURV.2009.090303

[13] C. Savaglio, G. Fortino, and M. Zhou, "Towards interoperable, cognitive and autonomic iot systems: An agent-based approach," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016, pp. 58–63. DOI: 10.1109/WF-IoT.2016.7845459

[14] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing mape-k feedback loops for self-adaptation," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015, pp. 13–23. DOI: 10.1109/SEAMS.2015.10

[15] B. Athamena and Z. Houhamdi, "Cognitive and autonomic iot system design," in *2021 Eighth International Conference on Software Defined Systems (SDS)*, 2021, pp. 1–7. DOI: 10.1109/SDS54264.2021.9732121

[16] C. R. China, *What is network automation? | ibm*. Accessed: Jul. 27, 2026. [Online]. Available: https://www.ibm.com/think/topics/network-automation

[17] Ansible, *Introduction to ansible - ansible community documentation*. Accessed: Jul. 27, 2025. [Online]. Available: https://docs.ansible.com/projects/ansible/latest/getting_started/introduction.html

[18] S. Project, *Introduction to salt*. Accessed: Jul. 27, 2025. [Online]. Available: https://docs.saltproject.io/en/latest/topics/index.html

[19] B. Networks, *What is ddi? a solution for managing your network*. Accessed: Jul. 27, 2025. [Online]. Available: https://bluecatnetworks.com/glossary/what-is-ddi/

[20] P. Mockapetris, *Rfc 1034: Domain names - concepts and facilities*. Accessed: Jul. 27, 2025. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1034

[21] N. Labs, *Nlnet labs - nsd - about*. Accessed: Jul. 27, 2025. [Online]. Available: https://www.nlnetlabs.nl/projects/nsd/about/

[22] PowerDNS, *Powerdns authoritative server*. Accessed: Jul. 27, 2025. [Online]. Available: https://www.powerdns.com/powerdns-authoritative-server

[23] PowerDNS, *Powerdns recursor*. Accessed: Jul. 27, 2025. [Online]. Available: https://www.powerdns.com/powerdns-recursor

[24] PowerDNS, *Powerdns dnsdist*. Accessed: Jul. 27, 2025. [Online]. Available: https://www.powerdns.com/powerdns-dnsdist

[25] I. Internet Systems Consortium, *Introduction to dns and bind 9*. Accessed: Jul. 27, 2025. [Online]. Available: https://bind9.readthedocs.io/en/latest/chapter1.html#introduction

[26] R. Droms, *Rfc 2131 - dynamic host configuration protocol*. Accessed: Jul. 27, 2025. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2131#page-2

[27] I. Internet Systems Consortium, *Kea dhcp - isc*. Accessed: Jul. 27, 2025. [Online]. Available: https://www.isc.org/kea/

[28] S. Kelley, *Man page of dnsmasq*. Accessed: Jul. 27, 2025. [Online]. Available: https://thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html

[29] Infoblox, *What is ipam (ip address management)?* Accessed: Jul. 27, 2025. [Online]. Available: https://www.infoblox.com/glossary/ipam-ip-address-management/

[30] phpIPAM, *Phpipam ipam ip address management software*. Accessed: Jul. 27, 2025. [Online]. Available: https://phpipam.net/

[31] N. Labs, *Introduction to netbox | netbox documentation*. Accessed: Jul. 27, 2025. [Online]. Available: https://netboxlabs.com/docs/netbox/introduction/

[32] I. Infoblox, *[datasheet] infoblox universal ddi management*. Accessed: Jul. 27, 2025. [Online]. Available: https://insights.infoblox.com/resources-datasheets/infoblox-datasheet-infoblox-universal-ddi-management

[33] D. C. Verma, *Principles of Computer Systems and Network Management*. Springer US, 2010, pp. 111–122. DOI: 10.1007/978-0-387-89009-8

[34] L. Nagios Enternprises, *Overview | nagios open source*. Accessed: Jul. 27, 2025. [Online]. Available: https://www.nagios.org/about/overview/

[35] Z. SIA, *2 what is zabbix*. Accessed: Jul. 27, 2025. [Online]. Available: https://www.zabbix.com/documentation/7.0/en/manual/introduction/about

[36] Z. SIA, *3 zabbix features*. Accessed: Jul. 27, 2025. [Online]. Available: https://www.zabbix.com/documentation/7.0/en/manual/introduction/features

[37] ONVIF, *"onvif overview", onvif.org*. Accessed: Jan. 8, 2025. [Online]. Available: https://www.onvif.org/wp-content/uploads/2024/11/onvif-overview-20241108.pdf

[38] ONVIF, *"onvif core specification", onvif.org*. Accessed: Jan. 8, 2025. [Online]. Available: https://www.onvif.org/wp-content/uploads/2019/12/ONVIF_Profile_S_Specification_v1-3.pdf

[39] T. Nixon and A. Regnier, *Web services dynamic discovery (ws-discovery) version 1.1*. Accessed: Jan. 8, 2025. [Online]. Available: https://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html

[40] ONVIF, *"profile s", onvif.org*. Accessed: Jan. 8, 2025. [Online]. Available: https://www.onvif.org/wp-content/uploads/2019/12/ONVIF_Profile_S_Specification_v1-3.pdf

Appendix A

System Requirements

Table A.1: System Requirements

ID	Responsability	Want to be able to	So that	Priority	Status
1	System Admin	Login to the IP Address Management application	Only the system admin has the permissions to manage the network	Must	Done
2	System Admin	Manage IP addresses	Devices receive correct IPs, conflicts are avoided and everything is correctly mapped out for easier management and organization	Must	Done
2.1	System Admin	Add new subnet	Subnets and their corresponding hosts are organized and properly managed	Must	Done
2.1.1	System Admin	Add new host	An IP and MAC address pair ir reserved for the new host once it's added to the network, along with it's new domain name	Must	Done
2.1.1.1	System Admin	Input new host IP address	So that the IP is reserved for the host	Must	Done
2.1.1.2	System Admin	Input new host MAC address	So that no other host can accidentaly receive that IP	Must	Done
2.1.1.3	System Admin	Input new host hostname	It can be used for the hosts domain name	Must	Done
2.1.1.4	System Admin	Input new host description	It's easier to understand what's the host's role in the network	Could	Done
3	System	Retrieve subnet hosts information	IPs can be reserved for their respective MAC addresses and domain names be given for each host	Must	Done
3.1	System	Add IP/MAC addresses pairs to the DHCP server configuration	The IPs are reserved for each respective host (MAC address)	Must	Done
3.2	System	Add hosts domain name entry to the DNS authoritative server	The hosts can be found by their domain names, instead of IPs	Could	Done
3.3	System	Add hosts entry to Monitoring Server	The hosts performance can be monitored and information fed to the orchestrator	Must	Done
4	System Admin	Manage certificates	Access to the machines can be managed	Must	Done
4.1	System Admin	Issue certificates	Access to the machines can be granted	Must	Done
5	System	Discover Camera Sensors	Their capabilities are determined	Must	Done
5.1	System	Configure a new password for the Camera Sensor	The device is automatically configured and ready to be used	Must	Done
5.2	System	Comunicate the cameras capabilities to an orchestrator	The orchestrator can make it's own management	Should	Done
6	System	Configure SSH servers on hosts	The hosts are automatically configured and accessible to users	Must	Done
7	System	Configure the monitoring agent on hosts	The hosts can be automatically monitored by the monitoring system	Must	Done
8	System	Discover Raspberry Pis	Their capabilities are determined	Must	Done
8.1	System	Comunicate the RPis capabilities to an orchestrator	The orchestrator can make it's own management	Should	Done
9	System	Send discovered hosts information to an MQTT server	The orchestrator receives the discovered hosts information	Must	Done
10	System	Detect when a Raspberry Pi is down	The orchestartor is notified about the processing node being down	Must	Done
10.1	System	Communicate the down state of the Raspberry Pi via MQTT	The orchestrator can make it's own management	Must	Done
11	System	Detect when a Camera Sensor is down	The orchestrator is notified about the camera sensor being down	Must	Done
11.1	System	Communicate the down state of the Camera Sensor via MQTT	The orchestrator can make it's own management	Must	Done

Appendix B

Functional Tests

Table B.1: Functional Test Case for Camera On-boarding

Test Case ID	FT-NHOB-001		
Description	This test aims at validating the on-boarding procedure of a new ONVIF enabled IP Camera in the network		
Environment	Consult **Results and Discussion** chapter, **Laboratory Set up** section		
Preconditions	Camera must be factory reset Camera must not have been previously connected to the network for lesser than 1 min A default password is already configured in the configuration file for the Network Management and Monitoring (sp2-conf.ini)		
Input/Trigger	Connecting the Camera to the Network via Ethernet		
Step	**Procedure**	**Expected Result**	**Result Status**
1	Login to phpIPAM's admin account	Login is successful	Pass
2	Reserve IP address for Camera's MAC address	Camera's MAC and IP address is added to the subnet list of hosts	Pass
3	Reserve IP address for Camera's MAC address	Camera's MAC and IP address is added to the subnet list of hosts	Pass
4	Connect Camera to the networks Smart Switch via ethernet	Camera's and Smart Switches connected Ethernet Ports LEDs turn on	Pass
5	Connect to Camera's webpage through http	Camera's webpage turns up with a prompt requesting a sign-in password for the user "service"	Pass
6	Login to Camera's "service" user	Login is successful	Pass
Pass/Fail Criteria	Step 1 through 6 need to pass in order for the test to pass		
Overall Result Status	Pass		
Evidence Reference	Consult **Results** chapter		

Table B.2: Functional Test Case for Processing Node On-boarding

Test Case ID	FT-NHOB-002		
Description	This test aims at validating the on-boarding procedure of a new Processing Node in the network		
Environment	Consult **Results and Discussion** chapter, **Laboratory Set up** section		
Preconditions	Processing Node has Raspberry Pi OS installed Processing Node has SSH is enabled Processing Node has the Public key of RPi DDI in authorized hosts list Processing Node must not have been previously connected to the network for lesser than 1 min		
Input/Trigger	Connecting the Processing Node to the Network via Ethernet		
Step	**Procedure**	**Expected Result**	**Result Status**
1	Login to phpIPAM's admin account	Login is successful	Pass
2	Reserve IP address for Processing Node's MAC address	Processing Node's MAC and IP address is added to the subnet list of hosts	Pass
3	Reserve IP address for Processing Node's MAC address	Processing Node's MAC and IP address is added to the subnet list of hosts	Pass
4	Connect Processing Node to the networks Smart Switch via ethernet	Processing Node's and Smart Switches connected Ethernet Ports LEDs turn on	Pass
5	Attempt SSH connection to processing node without signed certificate	SSH Connection is rejected	Pass
6	Attempt SSH connection to processing node with signed certificate	SSH Connection is accepted	Pass
7	Login to Zabbix admin account	Login is successful	Pass
8	Navigate to Zabbix monitored hosts	Monitoring data for processing node is displayed	Pass
Pass/Fail Criteria	Step 1 through 8 need to pass in order for the test to pass		
Overall Result Status	Pass		
Evidence Reference	Consult **Results** chapter		