

Do Le Tuan Minh

# LONG TERM TESTING FOR RAILWAY'S LIGHT CONTROL UNIT

M. Sc. Thesis  
Faculty of Information Technology and Communication Sciences  
Examiner: Prof. Karri Palovuori  
Examiner: University Lecturer Erja Sipilä  
May 2024

# ABSTRACT

Do Le Tuan Minh: Long Term Testing for Railway's Light Control Unit  
M.Sc. Thesis  
Tampere University  
Master's Degree Programme in Information Technology, Embedded Systems  
May 2024

---

Nowadays, railway transport is becoming common and increasing in use in daily life. The railway industry also received advancement in technology, such as LED replacing phosphorous light. As such, the Light Control Unit (LCU), an embedded system used in railway carriage, utilizes PWM technique to control LED lighting became a demanding component. This device adjusts LED lighting depending on various conditions, ensures passenger comfort and safety. However, as an embedded system, LCU suffers operational stresses over an extended period, such as voltage fluctuations, temperature variations, and mechanical vibrations, which can impact the LCU performance and operational lifetime. Moreover, faults might not be detected at the testing phase, only after a long duration. As such, LCU requires addition testing procedures to detect and correct potential hardware and software failures.

This Thesis was done in the Rail Division of Teknoware Oy. The goals are to create a testing framework at the very end of testing process to monitor performance of the LCU in real time, detecting failures that have not been encountered during the testing process, evaluating if the performance met the requirements, and contribute to the company's testing procedure.

The framework continuously monitors CPU status, memory usage, temperature, and voltage, with data logged and transmitted in real-time via syslog protocol to a designated test PC. This monitoring system was developed as a systemd service, installed in the LCU's debugging Yocto Project's image as a recipe. The LCUs connect to the train's network through communication protocol, using simulation software. Through remote Syslog server, log messages are saved in the Test PC and separated by the original IP addresses. Regular expressions are used to detect desired fault event messages and push to a Microsoft Teams channel, preventing Information overload in the channel, which leads to channel inactive. The data from each LCUs can be analyzed, visualized and evaluated. The code for this work has been merged and maintained in Teknoware's Git repository.

Keywords: Embedded Linux, Yocto Project, Systemd Service, Syslog-ng, Light Control Unit, Embedded System, Pulse-Width Modulation, long-term testing, embedded system testing.

The originality of this thesis has been checked using the Turnitin Originality Check service.

# PREFACE

This work was carried out at Teknoware Oy, Lahti from spring 2023 to spring 2024.

My study at Tampere University is nearly at the end, after 4 years. It was difficult, since the study started during Covid-19 time, and I must spend most of my study time remotely in my country, and some thesis topic changing. It is time that I show my gratitude to everyone who has supported me during the work of this thesis.

First, I would like to thank Professor Karri Palovuori and University Lecturer Erja Sipilä as my supervisor from the University, for their guidance, advice, and support during my writing.

I would like to express my gratitude to my supervisors Jouko Nikula, Mati Alava and Markku Vorne, in Teknoware, who kindly granted me the topic and guided me through the project. Without them, my progress would have been much more difficult.

I would like to send my love and grateful to my family in Vietnam, they always be my motivation to move forward to be better, with their care and support. In addition, I would give my appreciation to my group of friends in Tampere, they helped me a lot when I first came back to Tampere after Covid-19 time, cheered me up and gave me spirit during my hard time finding a job. With their support, I finally landed my job in Teknoware and being part of this work. Finally, I would like to show my appreciation to my friends in Lahti. They helped me during I first settled in Lahti and with me during my desperate time.

Lahti, 30 May 2024

Do Le Tuan Minh

## **USE OF AI IN THESIS**

*I have utilised AI tools in my thesis:*

- No
- Yes

*The AI tools utilised in my thesis and their purposes are described below:*

*Names and versions of AI tools: ChatGPT-4 & 4o, Quillbot v20.1.0*

*Purpose of using AI tools: Quillbot was used to improve language, grammar checking and citation generated from URL. ChatGPT was used to help brainstorming, finding relevant academic sources, generate demonstration figures in theoretical part.*

*Sections where AI tools were used: Quillbot was used in all sections. ChatGPT was used in 2.2, 2.4, 2.5, 2.6, 2.7, figure of 2.1.1, 2.4.1, 2.5, 2.6 and 2.7, finding reference for 4.1.3 and 4.1.4*

*I acknowledge that I am fully responsible for the entire content of my thesis, including the parts generated by AI, and accept accountability for any violations of ethical standards in publications.*

# CONTENTS

<i>USE OF AI IN THESIS</i> .....	III
1. INTRODUCTION.....	1
2. THEORETICAL BACKGROUND.....	3
2.1 Embedded Systems .....	3
2.1.1: Introduction .....	3
2.1.2: Embedded System Software .....	3
2.2 Long-Term Testing in Embedded Systems.....	4
2.2.1 Testing in Embedded System and Long-term Testing .....	4
2.2.2: Challenges of Long-Term Testing .....	5
2.3 Lighting Control Unit in Railway Carriage .....	6
2.3.1: Introduction .....	6
2.3.2 Specification of the LCU used for research .....	8
2.4 Yocto Project .....	11
2.4.1 Introduction .....	11
2.4.2 Bitbake .....	11
2.4.3 Meta-data.....	12
2.5 Systemd Service .....	14
2.6 Syslog-ng .....	15
2.7 Sysfs .....	16
3. RESESARCH METHODOLOGY .....	18
3.1 Infrastructure and approach .....	18
3.2 System-Monitoring Service.....	20
3.2.1 Syslog-ng configuration.....	21
3.2.2 Monitoring CPU and Memory consumption .....	23
3.2.3 Monitoring Hardware status.....	25
3.2.4 Logging System's status.....	27
3.2.5 Systemd setup .....	29
3.2.6 Yocto recipe setup.....	31
3.3 Test PC test environment setup .....	33
3.3.1 Syslog-ng server setup.....	34
3.3.2 Syslog-ng Regex configuration.....	35
3.3.3 Teams Channel set up. ....	36
3.3.4 Teams Channel message forwarding script.....	37
3.3.5 Train network setup.....	43
4. RESULTS AND ANALYSIS.....	45
4.1 Memory Consumption .....	49
4.2 CPU Consumption.....	54
4.3 CPU and LCU temperature .....	58
4.4 Voltage.....	62
4.5 LCU reboot, systemd service reboot and train network disconnect.....	63
5. CONCLUSIONS.....	64
REFERENCES.....	66



## LIST OF SYMBOLS AND ABBREVIATIONS

ADC	Analog to Digital Converter
ALT	Accelerated Life Testing
BSP	Board support packages
CPU	Central Processor Unit
DHCP	Dynamic Host Configuration Protocol
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
LCU	Light Control Unit
LED	Light-emitting diode
MB	Megabytes
MCU	Microcontroller unit
NTC	Negative temperature coefficient
OS	Operating System
PC	Personal Computer
PID	Process identifier
PWM	Pulse Width Modulation
regex	regular expression
SSH	Secure Shell
TCP	Transmission Control Protocol
TRDP	Train Real-time Data Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB	Universal serial bus
UTC	Coordinated Universal Time
I2C	Inter-Integrated Circuit
VDC	Volts of direct currents
Syslog-ng	System Logger Next Generation

# 1. INTRODUCTION

Nowadays, railway transport is having important role in modern life and contributes significantly to economic growth. According to Worldwide railway organization (UIC), the number of passengers has increased every year for the last decades, with an average of 3.5% compared to 2019 in 2023 [1]. Many reasons explained this trend, including high speed trains, low cost, good service in interior carriage and lighting is one of the core elements. One of the significant advances in railway technology is using LED lighting, a rising trend in modern rolling stock lighting systems, to achieve energy and maintenance cost reduction, also increase passenger comfort and safety. LED proved to be more superior compared to traditional fluorescent lighting system, as Ishii et al. discussed the benefit of using LED such as, saving 40 to 60 % power consumption, emitting less carbon dioxide, no flickering and most importantly, significantly increase life cycle of the unit [2]. Thus, the introduction of the Light Control Unit (LCU) that controls LED lighting plays a vital role in the management of lighting operations for interior lighting applications in railway carriages. LCU can manually or dynamically adjust lighting levels of LED, using Pulse Width Modulation (PWM), by observing time of day, occupancy, and external light levels. Due to the discussed benefit and importance of LCU, it is essential to conduct testing in long duration to ensure their reliability and functionality remain consistent.

Unlike regular software programs, embedded systems function for extended periods from years to decades, with software and firmware updates during operational time. Embedded systems usually encounter changes environment stressor, such as temperature, humidity, electromagnetic or even physical vibration. Additionally, despite thorough test plans and procedures, errors, defects, or performance issues and deterioration may emerge over time that were not discovered during initial testing or shorter testing time. Moreover, although having a webserver interface for maintenance tasks, this feature currently works in a local network, causing difficulties in real-time monitoring and error detecting from a remote location. In addition, it is interesting to observe the behavior of the system after prolonged duration, since in fact, the train system usually works for the duration of the journey and stops after the train arrives at the station. Therefore, an addition testing phase is required to assess the LCU's performance over time, identifying any hardware or software issues. The process is

implemented once the software successfully clears all unit tests and is ready for final release.

This thesis aims to monitor and evaluate the functionality and reliability of LCUs after certain operational time, adding an additional step to the testing process, through several goals:

- Designing a systemd service to log the status of the LCU, designing a custom Yocto recipe for integration.
- Establishing a syslog-ng server to receive log messages from the LCUs regardless of distance on a Test PC.
- Creating Teams Channel to instantly notify when any faults happen.
- Detecting and analyzing the results in both hardware and software. If no faults were detected, analyze the potential cause that might harm the performance.
- Evaluating the LCU's performance to ensure it meets operational requirements, performance can be used as benchmarks for future products.

Chapter 2, Theoretical Background provides an overview of the theoretical and tools used in the development of the project. First, Embedded Systems provides introduction and the software that operates within these systems. Next section discusses Testing in Embedded Systems, Long-Term Testing in Embedded Systems highlighting the significance and challenges. Interior Lighting in Railway Carriages, providing an overview and the role of LCUs, introduces the LCU product at Teknoware, briefly explaining its hardware and software architecture. In addition, Yocto Project section studies the tool for creating custom Linux distribution and its components such as Poky, Bitbake, and how to configure for a build. Finally, several technologies needed for this work such as Systemd Service, Syslog-ng, Sysfs are also discussed.

Chapter 3, Research Methodology and Materials section describe the architecture of the test framework, approach, implementation of the system, including system-monitoring services development, and test PC environment setup.

Chapter 4, Results and Analysis, demonstrates the result of the work and evaluates the LCU's performance and the issue that might harm the LCU in each category. This chapter also discusses the weaknesses of design and offers recommendations.

Finally, Chapter 5 summaries accomplished goals, contributions, and suggestions to further improve the system and its application in the future.

## 2. THEORETICAL BACKGROUND

### 2.1 Embedded Systems

#### 2.1.1: Introduction

An embedded system is a system that combines computer hardware and software to carry out a specific or limited function, unlike general-purpose computers, which are multi-purpose systems. According to ARM, embedded system is a “*microprocessor-based computer system typically implemented as a component of a larger electrical or mechanical system*” [3].

Saini discussed that embedded systems containing many functions of a computer on a single device [4]. Embedded systems contain Central Processor Unit (CPU), memory, and I/O peripherals, power supply and operating software. Embedded systems usually have constrained resources, operate with limited resources, memory and power consumption compared to general purpose computers [3]. Embedded systems are usually designed to be operated continuously for long duration [5]. ARM discussed that “*there are billions of embedded system devices used across many industries including medical and industrial equipment, transportation systems, and military equipment*” [3]. Examples of embedded systems are as common as smart watches, household devices, robots, medical equipment, to more complex systems such as cars and aerospace shuttle.

In this work, the LCU is an embedded system. The LCU contains a center processor, I/O peripherals in light sensors, PWM ports and dedicated in lighting control purpose in the railway carriage.

#### 2.1.2: Embedded System Software

Embedded system software is specialized software that is installed or loaded into devices that enable embedded systems to function. Due to embedded system’s limited resources, embedded systems software usually light weight and highly optimized. According to ARM, “*embedded systems programming instructions are stored in read-only memory or flash memory chips*” [3]. Embedded system software can be bare-metal, which is OS-independent software and runs directly in the hardware, software running on the OS and real-time OS, which is more time-critical software. Saini discussed that Linux and Windows are two popular operating systems for implementing embedded

systems [4]. The LCU in this work is an embedded Linux system, with Yocto Project used to create a customized Linux distribution.

According to Siemens [6], Embedded system software can be divided into the following categories:

- The Operating System (OS) lets other software run on computer devices. OS also resource manager, responsible for memory allocation, storage devices, I/O peripherals, network connections, task management, and process scheduling.
- Firmware is programmed into a hardware component, operates without an OS or device drivers. The firmware provides instructions that enable the device to interact with other hardware components and execute fundamental functionalities as intended.
- Middleware is a software layer between the operating system and application programs.
- Application software provides interaction with middleware and firmware. Each application is unique, while the operating system and firmware remain consistent between different devices.

## **2.2 Long-Term Testing in Embedded Systems**

This subchapter discusses embedded software testing, long-duration testing and difficulties when conducting the test.

### **2.2.1 Testing in Embedded System and Long-term Testing**

Testing is an important step in embedded system development, which evaluates the integration between hardware and software, ensures they work together as intended. According to an online study by Intechhouse, there are several types of embedded testing, including unit testing, integration testing, system testing, acceptance testing, stress testing [7]. Embedded testing aims to detect defections, errors, or issues, decrease maintenance costs, and improve product performance. According to Bajer et al, the need to test embedded systems is lowering risk and verifying that the product satisfies its requirements and operates properly in various conditions [8]. Bajer et al. also discussed that effective testing should minimize the defective life cycle and the necessity of including testing into the development process from the beginning [8]. According to Vahid Garousi et al., lacking properly and adequately testing could leads to life-threat situations in critical system, or delays which causes lost in business profit [9].

Long-term testing for embedded systems can be referred to several types or steps of testing. Firstly, reliability testing assesses a system's ability to fulfill its intended function repeatedly and without failure over time [10]. Ryan Aalund et al. discussed that embedded systems are usually deployed and exposed to harsh conditions such as extreme temperatures, humidity, dust, and corrosive substances in prolonged time, as such ensure the reliability of embedded system is essential [11]. On the other hand, soak testing, or endurance testing, runs a system under a significant amount of load for a long time to detect problems such as memory leaks, resource depletion, crashes, freezes, and performance issues that are not detected in short-duration tests [12]. Soak testing should be run continuously, as long as possible, the duration of the test based on the system [13]. This test is ideal to conduct before a release, after big changes, expecting high usage load or during performance testing [13]. In addition, stress testing pushes the embedded system's usual working limits by putting the system into harsh conditions or running it outside of normal operational settings to ensure the system can handle unexpected situation [7]. At this stage, when bugs and long-terms effects happen, these symptoms need to be decided if they are acceptable and tolerable without impacting predefined requirements, as it is nearly impossible to produce bug-free programs. In this work, long-term testing involves the tested system continuing to operate for long, predefined time, automatically monitoring CPU, memory, temperature, and input voltage and alerts when exceeded thresholds.

### **2.2.2: Challenges of Long-Term Testing**

As a part of embedded system testing, reliability testing for long time not only inherits the challenges of embedded system testing but also facing unique difficulties.

One of the main difficulties in long-term testing for embedded systems is replicating real-world operating conditions. Operating in some extreme conditions could lead to unexpected behavior from the system, which could be ignored during the testing phase, or affecting the operation lifetime of the system. Research by Texas instruments stated that the effective lifespan of processors declines exponentially as temperature increases, for instance, functioning at 105°C and above may decrease the lifespan of embedded processors from 10 years to 2 years [14]. Furthermore, simulating or creating these conditions can be challenging, requires significant investment in specific equipment, such as heat, humidity or electromagnetic chambers, thus difficult to produce. Maria Hernandez discussed that the environment often requires specialized chambers and

equipment to create repeatable, controlled stress conditions [15]. Banerjee et al. argues that the actual surroundings that embedded software operates are usually unavailable during testing [16].

According to Vahid Garousi et al., embedded systems are usually limited to no Graphical User Interface (GUI), causing difficulty in observing internal state of the systems [9]. This drawback requires development of specialized instrumentation and probing techniques for testing [9].

Furthermore, after certain operation time, embedded system suffers degradation and aging effects. This happened when the gradual decline in the physical and functional state of hardware parts over time because of continuous use, environmental, and material wear and might not reflect the correct system behavior. Ryan Aalund et al. notes that heat might accelerate aging and cause impact in system reliability [11]. Hardware components are degraded over time, and software can output unpredict behavior. Vikas Chandra discussed that as CPUs age, they may experience progressive degradation in performance characteristics, which can result from various factors, including increased leakage currents and reduced transistor switching speeds, leading to slower processing times and diminished overall system responsiveness [17]. Vikas Chandra also discussed that aging effects can create delays in instruction execution, increasing latency and lowering real-time efficiency, which might interrupt time-sensitive processes [17].

Embedded systems usually have limited resources. Noack argues that embedded systems often have limited processing power, memory, and energy resources, which is difficult to perform long-term monitoring and data processing on the device [18].

Test duration is one of the biggest challenges for soak testing. Firstly, it is difficult to determine the exact testing time. The duration might be several hours, or even days or months, which create planning and scheduling difficult [12]. On the other hand, soak testing usually runs for long time, thus too long for strict deadline projects [12]. Yumei Wu et al. investigated how challenges in reproducing the embedded system's operating environment may result in incorrect testing time during which failures occur; even if the environment could be constructed, the test might damage the system. [19].

## **2.3 Lighting Control Unit in Railway Carriage**

### **2.3.1: Introduction**

According to Teknoware, discussed in the product introduction, LCU is an embedded system, specialized in managing and controlling the interior lighting of trains, trams and other rolling stock, which optimizes energy consumption, enhances passenger comfort,

and ensures compliance with lighting standards [20]. LCUs can operate independently or integrate into broader train control systems, through various interfaces such as power supply, PWM outputs, and Ethernet connections, and the size of the carriage. Teknoware provides wide variation of LCUs, depending on many aspects, for instance, size of the carriage, location of lighting installation and customer's requirements. Core feature of LCU is lighting control including the intensity, color temperature, and RGB color of carriage lighting through dimming control using on PWM technology and the ability to connect with the train controlling network for better management. Architecture wise, Teknoware provides 2 types of LCUs: microcontroller based, and Linux based. The LCU used for this work was Linux based LCU using Yocto Project.

Dimming control is the core feature of the LCU. Dimming control using PWM technic to adjust light output levels to desired intensities. PWM rapidly controls the power supply to the lighting, switching the LED on and off with pre-defined frequency, affecting the output brightness. According to Teknoware, LCU can manage the intensity and color of LED lights to create ambient environments suitable for different scenarios, by using a webpage interface [20]. This interface allows users to adjust lighting through pre-defined scenarios, such as going to tunnels, stations, daylight and so on, overriding PWM values of existing scenarios or creating a custom scenario. Moreover, PWM control can be used to control RGB LEDs to create dynamic visual experiences or situational guidance, for example, creating Aurora effects, or seasonal themes of a holiday. PWM can also be used to control the LED's color temperature, as color temperature significantly affects passenger comfort or mood. PWM and LED control is a rising trend in railway lighting control as LED has been far superior in benefit compared to traditional lighting control such as longer lifespan, consume less energy or less flickering, as discussed by Teknoware and Ishii et al. [20][2].

Some LCU varieties are equipped with active lighting control features or Smart Lighting, by integrating light sensors. Light sensors continuously monitor surrounding light illumination, provide real-time data to the LCU, dynamically adjust the lighting level of the LEDs throughout carriage, matching the ambient conditions, such as daylight or entering tunnels. According to Teknoware, *“Active lighting control uses sensors to adjust the interior lighting of a vehicle according to external factors, such as daylight. This means that the LEDs are only used to the exact degree they are needed. This makes for a more comfortable journey for passengers and extends the lifecycle of the lights. The result Reduced costs and increased efficiency”* [20]. Active lighting control's biggest advantage is the LCU optimizes the use of light, archiving desired lighting level with

minimal energy and maximum LED lifetime. Depending on the scenario that controlling the light, the LCU can switch between Smart Lighting and Dimming Control easily.

To communicate and exchange data with the train's central management network, LCUs can interface with the train's central management network by utilizing communication protocols. Example of communication protocols used in train carriage are Ethernet, Train Real-time Data Protocol (TRDP), an IP-based protocol using User Datagram Protocol (UDP) to transmit data, which is one of the most common used in industry. LCUs from Teknoware are equipped with 2 different Ethernet ports. Local Maintenance Ethernet with static IP address port for maintenance engineers, without connect to the train network. The remaining Ethernet port uses specific protocol to establish communication with the train network, with DHCP server to assign IP to the LCU and transmit data with UDP. In this work, the Ethernet port with DHCP server was used. Demonstration of how LCU works, and connectivity reflects real life scenario will be discussed in chapter 3.

### 2.3.2 Specification of the LCU used for research

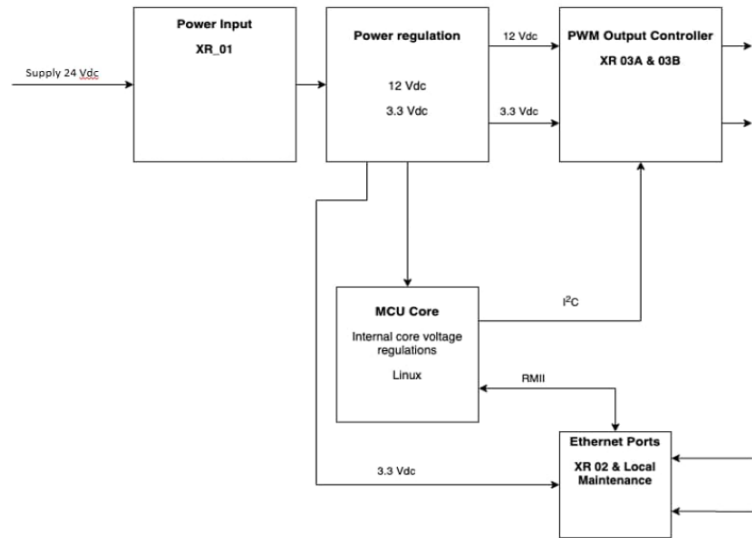
This subchapter describes the hardware and software specification of the LCU used in this work.

#### 2.3.2.1 Hardware



**Figure 1.** *Teknoware Light Control Unit*

Figure 1 demonstrate the LCU manufactured by Teknoware and used for this work. The LCU can be powered by 24 V input voltage. The LCU is equipped with power supply and two PWM ports, a local maintenance, and a train network ethernet port.

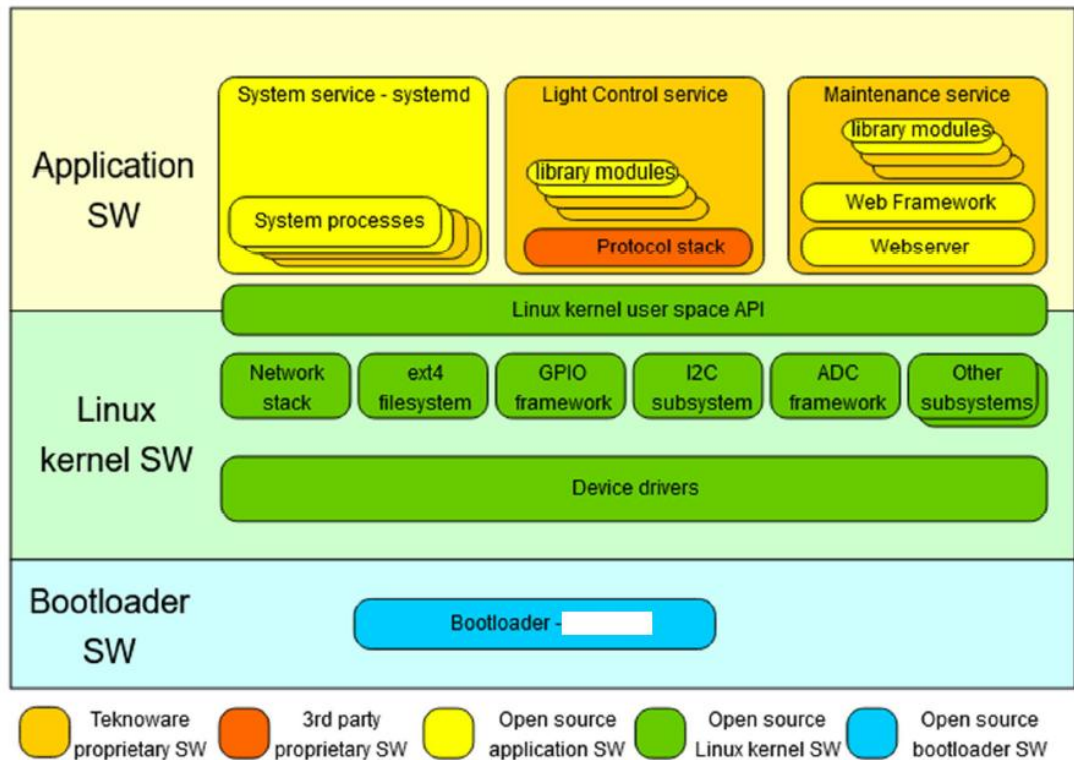


**Figure 2.** LCU hardware function block diagram.

The hardware of the LCU contains the following components, from Figure 2 hardware block diagram:

- **The Power Input Block (XR\_01)** receives 24 Volt direct current (VDC) power supply input and distributes the power into the system.
- **The Power Regulation Block** regulates the supplied voltage to conform to the system specifications, producing two separate outputs: 12 VDC and 3.3 VDC. This block provides power for PWM power control and the MCU core.
- **The Microcontroller Unit (MCU) Core** is the brain of the system. The core is a Linux system.
- **The PWM Output Controller (XR 03A & 03B)** receives voltage regulated from power regulation block and modulates the intensity of the lights by PWM signal.
- **Ethernet Ports (XR 02 & Local Maintenance)** used to establish network communication to train network or external devices and maintenance tasks. XR 02 is used as DHCP client and Local Maintenance used static IP.
- **Inter-Integrated Circuit (I<sup>2</sup>C) Bus** serial connection for enabling the MCU to control the PWM Output Controller.
- **Reduced Media-Independent Interface (RMII)** used for network communications, connecting the MCU core with external systems through Ethernet interfaces.

### 2.3.2.2 Software



**Figure 3.** LCU software architecture diagram.

The software architecture of the LCU contains the following main components, as shown in Figure 3:

- **Bootloader SW** starts the operating system upon device startup, by loading the Linux kernel to memory and initiates the Linux kernel.
- **Linux kernel SW:** The LCU's operating system is the Linux kernel. Linux manages memory, processes, subsystems, and devices drivers. The Linux kernel also provides user space API for user applications to access kernel services and hardware devices.
- **Linux device drivers:** The Linux device drivers control the device hardware.
- **Application SW:** The application SW contains three main services: system service, light control service and maintenance service.
  - The `systemd` starts processes upon LCU booting. In addition, the system service monitors critical application services, requires services to update status, and reboots the devices if any services fail to update the status with a defined time. The system logs are written by `system-journald` service and viewed with `journalctl`.

- `lcuserver` is the most important service of the system, which controls the PWM outputs. In addition, this service can detect faults from the lighting operation or internal software error and react accordingly.
- The maintenance service provides application webserver, which can update the software and LCU configuration, retrieve device status information and logs and display status in the web GUI.

## 2.4 Yocto Project

Teknoware's LCU are Linux based embedded system, as such, understanding the Yocto project is essential for this work. This subchapter provides briefly the structure of Yocto and how it works.

### 2.4.1 Introduction

The Yocto Project is an open-source project, which provides tools and build customized Linux-based distribution for embedded systems, regardless of hardware architecture [21]. Some components Yocto project build an image includes layers from OpenEmbedded, Bitbake build system and meta-data. Georg et al. discussed that Yocto Project is an open source collaborating project, which gathers multiple hardware vendors under a common framework [22]. A Yocto build can be customized to include only the essential components for the specific embedded device, thus significantly reducing the size of the image itself. Rakshitha et al. discussed the Yocto Project's build system, its effectiveness in developing customized binary images optimized for specific hardware setups, which is important for devices with limited processing power and storage capacity [23].

### 2.4.2 Bitbake

BitBake is a Python-based tool provided in Yocto Project for automatically executing and scheduling tasks for compiling and assembling custom Linux distributions. According to Yocto project, Bitbake is responsible for parsing metadata, generating lists of tasks from metadata and executing these tasks [24].

BitBake divides the build process into separate tasks, which are executed in order, based on the configuration files, including fetching source code, configuring, compiling, and packaging. Bitbake also supports parallel tasks execution, caches build outputs and metadata, to speed up the building process by allowing multiple tasks to run and reduce task redundancy [24]. During the build, Bitbake can check syntax from the source code,

provide detailed log information of the build, or detect changes in the recipes or layers and continue the build from these changes. Bitbake is invoked to start the build by using `bitbake <recipe-name>` command. From the Yocto Project Mega-Manual, Bitbake can fetch source code from numerous places such as Git and can check syntax and detect changes in files between builds automatically [24].

### 2.4.3 Meta-data

According to Yocto project, metadata includes recipes, configuration files, and other information related to the build instructions, parts needed to be built and required software needed for the build [24]. This subchapter details some of the most important metadata.

#### 2.4.3.1 Recipe

The recipe is a set of instructions and meta-data that BitBake, the Yocto Project's build system. Recipes are files with the `.bb` extension. Bitbake builds recipes into software packages, which are later used to create software images.

The recipe includes several important configurations for the build. `SRC_URI` variable specifies location of the source code, either from a Uniform Resource Locator (URL) of a remote version control repository or a local directory path. `DEPENDS` variable defines the build dependencies, which describes other software packages that needed to be built or installed before the building of the current package. Recipes use tasks such as `do_fetch`, `do_compile`, and `do_install` to give instructions and guidance for the build, including configuring, compiling and packaging [24]. The `do_fetch` method retrieves the source code, whereas `do_compile` handles the compilation process like `make`, `do_install` copies, packs and stores files to the target build directory [24]. Packages built from recipes can finally be assembled into an image recipe. Bitbake can execute image recipe files, which can be deployed to the targeted hardware platform by flashing tools, boot loaders, or specific procedures provided by the hardware manufacturer.

To add a new recipe to the Yocto Project, a `.bb` file in a layer folder must be created. Existing recipes can be modified by editing the variables in the `.bb` file. To extend or override the functionality of the recipe and keep the base implementation, `.bbappend` file is needed.

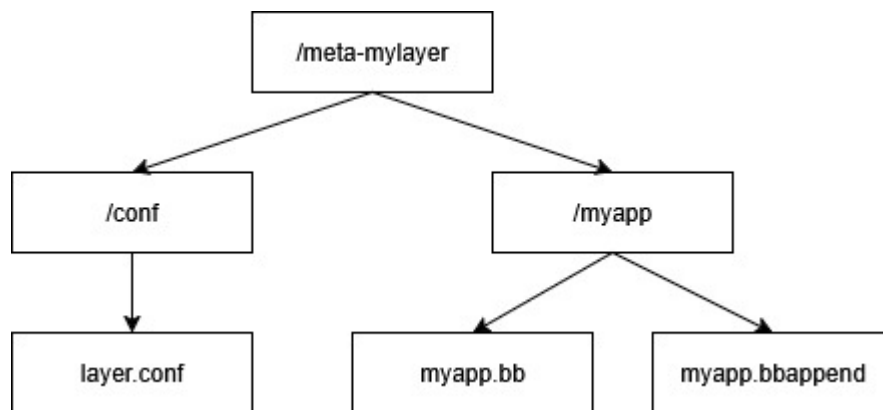
The work of this project creates recipe for monitoring LCU's health and adds this recipe to an image recipe of meta-teknoware layer. Detailed implementation will be discussed in chapter 3.

### 2.4.3.2 Layer

Layer is a collection of related recipes [24]. Layers are directories containing instructions, configurations, and metadata that guide build process [21]. Georg et al. point out that each Yocto project is supported by layers from the OpenEmbedded project, allows embedded Linux development [22]. Layers should be grouped into related or similar uses or purposes, for instance, Board Support Package (BSP), GUI, OS configuration or application [24]. Layers' main components include:

- **Recipes:** Instructions for building packages ( `.bb` files).
- **Configurations:** includes recipes's path, layer's priority and dependencies on other layers (`layer.conf`).
- **Overrides:** overriding exist recipes using `.bbappend` files.

Figure 4 shows the example of basic structure of a layer, with a layer `mylayer` and its `myapp` recipe:



**Figure 4.** *mylayer layer*

To add a layer into a Yocto Project build, use command `bitbake-layers <layer-name>` and add it to the `bblayers.conf` configuration file. The order of layers in `bblayers.conf` affects the priority of configuration settings and recipe overrides or dependencies execution order.

### 2.4.3.3 Configuration file

BitBake configuration files allow modification and managing the build environment and process. Most configuration files have `.conf` extension.

- `bblayers.conf` locate in the build directory `/conf` folder, includes the layers that BitBake uses in the build steps. The build directory is created when command `source`, which is used to set up build environment [24].

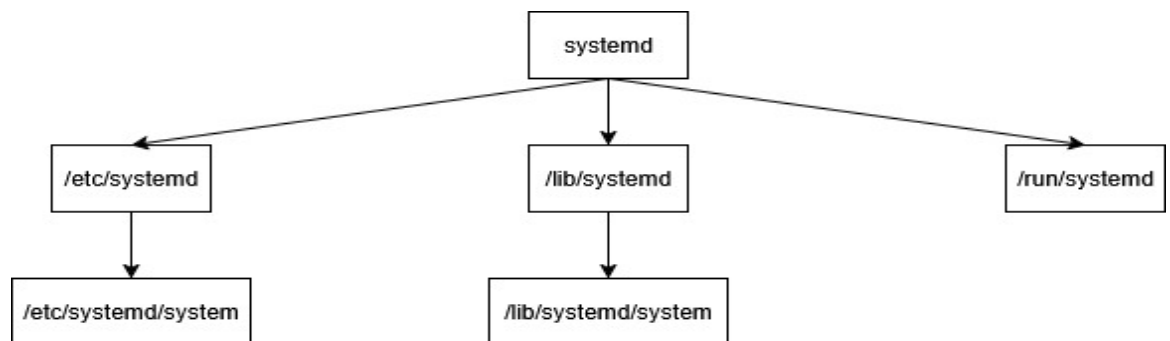
`bblayers.conf` includes location path of layers and their build priorities. New layers need to be added to this file so they will be included in new build [24].

- `local.conf` also found in the `/conf` directory, is used to define local build settings for build environment [24]. Example of data inside `local.conf` including the target machine (`MACHINE`), location to access downloaded file (`DL_DIR`), and package type (`PACKAGE_CLASS`) [24].
- Target machine configuration file, located in a BSP layer. The configuration files, labeled with the target hardware's name (e.g., `conf/machine/my-machine.conf`), define the hardware-specific build parameters. The `MACHINE` variable in the `local.conf` file points to a specific machine configuration file that instructs BitBake on building the specified target device [24].

## 2.5 Systemd Service

Systemd is a system and service manager for Linux OS during system runtime.

Figure 5 demonstrate the file system of the system.



**Figure 5.** Systemd file structure.

Systemd unit files have `.service` extension, which contains information about how the process is managed by the system. The first daemon starts at boot up and the last one finishes at shutdown. Freedesktop discussed that systemd service is defined by unit files that describe how the service should be managed and executed by the systemd manager [25]. Based on literature in the field from Freedesktop, systemd services operate by processing these unit files and dependencies to start and manage services correctly, by reading the unit file and checks dependencies before executing the specified commands when starting a service [25].

DigitalOcean discussed that the unit file contains the following sections [26]:

- `[Unit]`: This section contains an overview of the service, describes the service relationship with other services, for instance, `requires` defines if the service

needs other services to run, `before` and `after` defines if the service runs before or after other service, and `PartOf` determines if the service is part of other service.

- `[Service]`: section defines necessary for the service's operation, like which program or command to start, restart behavior, resource limitations, and logging configuration.
- `[Install]`: This section is optional, specify target under which the service should be installed.

When installed to the system, the service file locates in `/lib/systemd/system` directory. Service functions can be configured with unit files in `/etc/systemd/system` directory.

Systemd provides several tools to monitor the system's status and manage how a system works. According to Freedesktop, the system software package includes different tools for managing systems, such as `systemctl`, or `journalctl`, each have specific tasks, for instance, dependency management, automatic restart, resource control, and integration with `journald` for logging and monitoring [25]. The `systemctl` command to start, stop, reboot, activate, check status, or disable systemd services and units. DigitalOcean discussed systemd's logging system is `journald`, gathers log data from many places and stores data in order, provides runtime logging and debugging [26] [27]. `Journald` can be invoked in command line environment with `journalctl` command.

Existing systemd service's functionality can be extended or overridden configurations for existing systemd services without modifying the original service unit files. This is done by placing a configuration file in a directory named after the service unit, with a `.conf.d` extension, and then creating a `.conf` file with specific changes. The implementation of a system service will be further discussed in chapter 3.

## 2.6 Syslog-ng

System Logger Next Generation (syslog-ng) is an open-source version of the syslog protocol for Unix systems. According to CrowdStrike, syslog-ng extends regular logging by additional features such as TCP encryption, advance filtering and logging to database [28]. From syslog-ng github, syslog-ng became default logging system for some distribution like Debian, SUSE, Gentoo [29]. Syslog-ng 's advantage to traditional logging methods by separate log messages into specific fields instead of treating them as

unstructured text. Additionally, syslog-ng can handle a variety of input and output formats, such as JavaScript Object Notation (json), unstructured text, and furthermore. Syslog-ng configuration file can be used to configure the source, destination and format of the log. *Sources* is the origin of log messages, could be logs in the local system or logs from servers on a remote network. *Filters* define rules to sort through log messages and adjust data depending on the content, severity, or other categories. *Destinations* are the final destinations for log messages.

According to Tsunoda and Keeni, syslog has been commonly used for gathering log messages via the network to monitor and manage network operations [30]. During system operations, messages are generated by operating systems, processes, and applications to report their status or event occurrences [30]. A syslog application manages these messages by sending and/or receiving them [30]. Syslog-ng collects logs from many sources, analyzes, and then transmits them to various destinations, by configuration files. In this work, the. Huang et al. addressed that syslog allows system events to be reported by gathering these events and stores them in log files on either the local system, a remote system, or both [31]. Barry and Loyola also discussed that syslog is very important for critical systems, since they generate large amounts of log, it is valuable to be collected and inspected from the hardware-based logging [32].

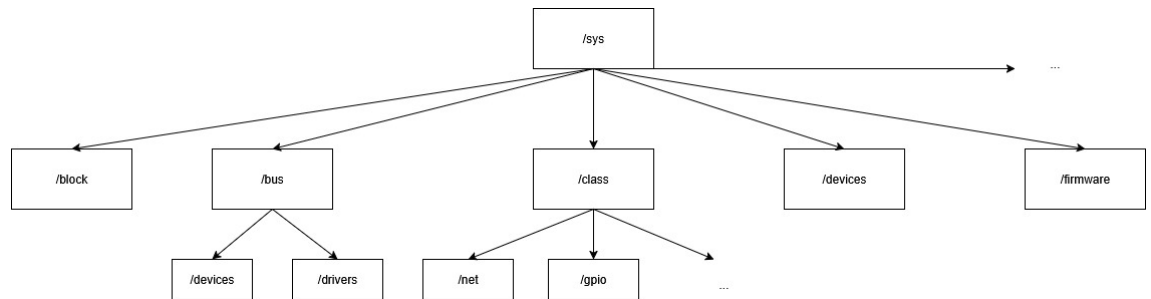
Syslog-ng was chosen for this project as logging system because syslog-ng has been installed in Yocto's image of the LCU. In addition, syslog-ng is highly configurable, allows customized log message format, and simple in streaming the messages with common network protocol, such as UDP in this work.

## 2.7 Sysfs

Mochel studied that sysfs is a file system in Linux that describes kernel information such as devices, kernel modules, filesystems, and other kernel components [33]. It is used for system configuration and device management, allowing user-space programs to interact with the kernel objects and its data structures. According to Wang et al., sysfs is an important component in Linux for managing and configuring kernel objects, providing a hierarchical and user-friendly interface for interaction between user space and kernel space, using tools such as echo or shell scripts [34]. According to Mochel, sysfs acts as a connection between the kernel's internal device model and user-space processes [33]. The sysfs filesystem has root directory at `~/sys`. Apart from information retrieving, Mochel highlighted how sysfs allow user-space applications to connect with the kernel

by executing basic file I/O activities, giving a simple mechanism to directly modify device attributes [33].

Figure 6 shows the file structure of sysfs.



**Figure 6.** *sysfs file system.*

The following lists important folders of sysfs file system[39]:

- `/sys/bus`: data of various types of device buses.
- `/sys/class`: devices registered to the system, based on types, for instance, `/sys/class/net` for network devices, `/sys/class/gpio` for General-Purpose Input/Output (GPIO) pins.
- `/sys/devices`: device trees.
- `/sys/firmware`: firmware-specific objects and attributes

Sysfs' operating around Kernel objects, or kobject. Mochel studied that sysfs exports kernel data structures by kobjects, as a hierarchy of virtual files and directories [33]. According to Mochel, kobjects form the backbone of sysfs, providing representation of kernel objects and their interactions [33]. Each kobject contains data on a single device, driver, or subsystem, kobject's name, parent-child connections with other kobjects. Kobjects can handle several actions, such as registering with the sysfs filesystem, creating and managing attributes, and establishing parent-child relationships. User-space programs communicate with sysfs using regular file system operations such as open, read, write, and close. The kernel converts these operations into interactions with the appropriate kobjects. Sysfs file system data retrieve and used in user application will be discussed in subchapter 3.2.3.

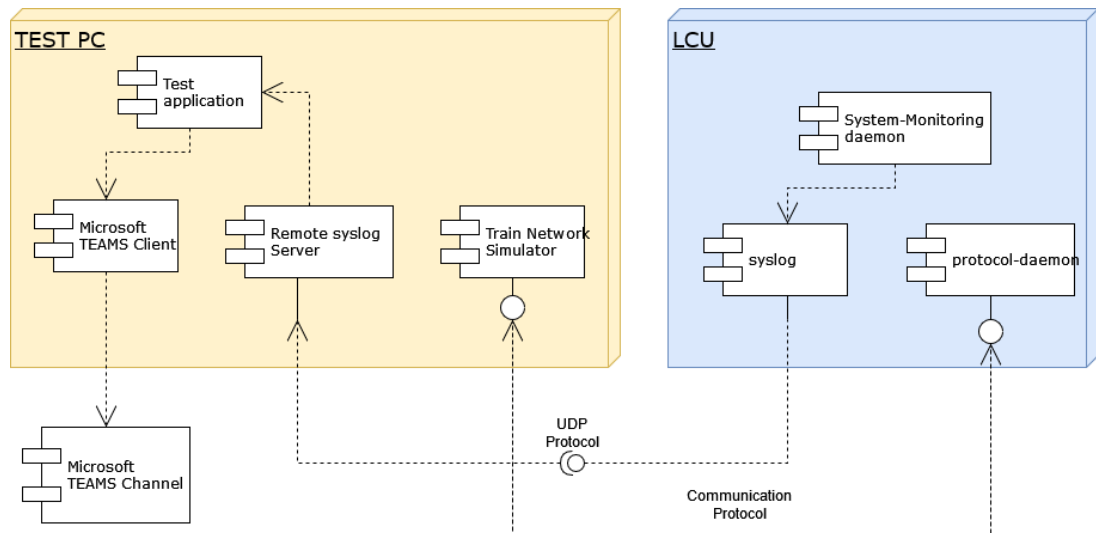
## 3. RESESARCH METHODOLOGY

### 3.1 Infrastructure and approach

The testing framework of this work contains the following components:

- LCU: a total of 6 LCUs were used in this test. Each LCU was installed in its images with extra systemd services supporting the test environment. syslog-ng service was used for local log saving and protocols to transfer log messages to the syslog-ng server. system-monitoring service monitors the LCUs' status.
- Test PC: The computer served as a syslog-ng server, which received the log messages sent from the LCUs through the syslog protocol. Moreover, the Test PC contained a regular expression (regex) configuration file, which was used to filter desired log messages. Furthermore, software was used to simulate the train network, communicating with LCUs. Finally, a continuously running script sent the filtered log message to the Microsoft Teams Channel.
- Router Hub: The MikroTik RB2011iL-IN is a network router, provides 10 Ethernet ports. This router connects LCUs and the Test PC through Ethernet cables.
- Microsoft Teams Channel: a Teams Channel is used to notify faults happening from the LCUs. The Channel connects with the Test PC through a webhook.

Figure 7 describes the block diagram of the software architecture of testing environment, demonstrating the components, their relations, and the direction flow of the data.



**Figure 7.** LCU system monitoring block diagram.

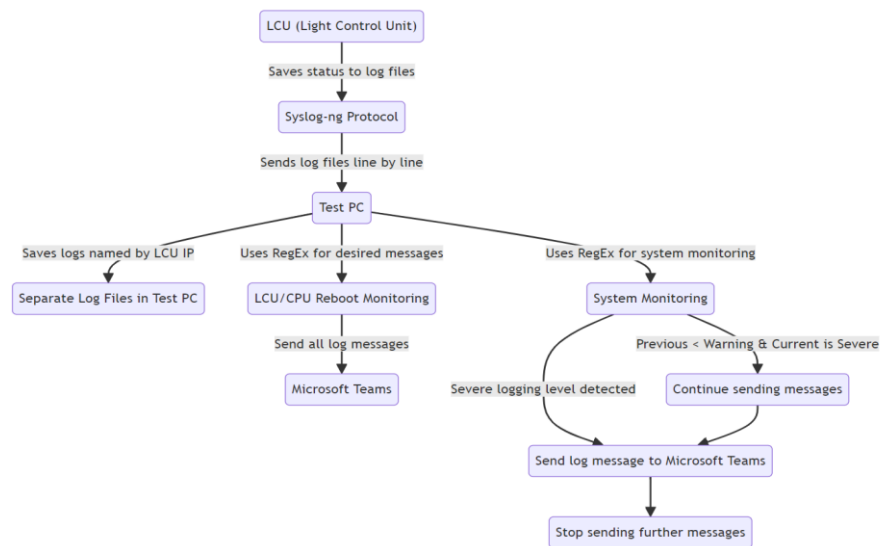
In this project, a combination of real-time monitoring and long-term data analysis was developed. LCUs continuously monitor their internal health status. This monitoring includes CPU utilization, memory consumption, LCU health, CPU temperature, and voltage levels. The collected data points are then logged within the LCUs for further analysis. The severity level of each logged message is determined by predefined thresholds established for each monitored category.

Through the syslog protocol, these log files are transmitted line by line to the test PC on the same network, serve as syslog-ng server. The test PC is configured to parse and separate the incoming LCU log data based on the source LCU's Internet Protocol (IP) address. To fulfil this, the LCUs and the test PC must be within the same network range, by connected through a router or hub. As the environment and real train control system are either hard to reproduce or not available, the train network simulators run on the test PC continuously send signals and receive the LCU's status through network protocol. Moreover, the test also uses `random_data` option from the train network simulator, which sends random train network data to the LCU, thus the LCUs continuously reports `lcludata` back to the train network. This stress tests the tenacity of the LCUs to process large amounts of data for long duration.

Even though the LCU has a web GUI, which is only available for local network accessible. This means only the computer connects with the LCUs through Ethernet interface can use this GUI. To mitigate this obstacle, the Teams message channel is needed to receive the LCU's internal status from afar. The test PC is a bridge between the LCUs and Microsoft Teams. It passes the received LCU logs to identify specific message regex pattern relevant to system status using predefined regex specified within a configuration file.

When the channel is spammed with logs at a time, the channel stops receiving messages and makes detecting fault messages more difficult. As such, logic prevents LCU status event messages flooding to the Teams channel was implemented. However, for LCU reboots or systemd service resets events, the test PC transmits all detected log messages of these events to the Teams channel without restriction of log level. This work was developed in both application layer, in Python and in Yocto layer creation and configuration. This work took log messages lines from recipes and used some methods developed by Teknoware, such as LCU reboot reason service, train network simulator, target machine hardware configuration and communication protocols.

Figure 8 summaries the workflow of the testing environment and how log messages are transferred and processed:

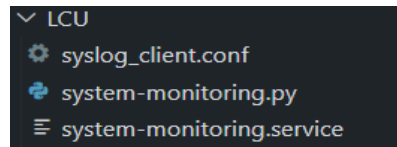


**Figure 8.** System Monitoring workflow.

For data analyzing purposes and resolving the lack of GUI, a Python script was created to read the log files from the LCU and plot variety of graph types to visualize the LCU's performance over the period of the testing.

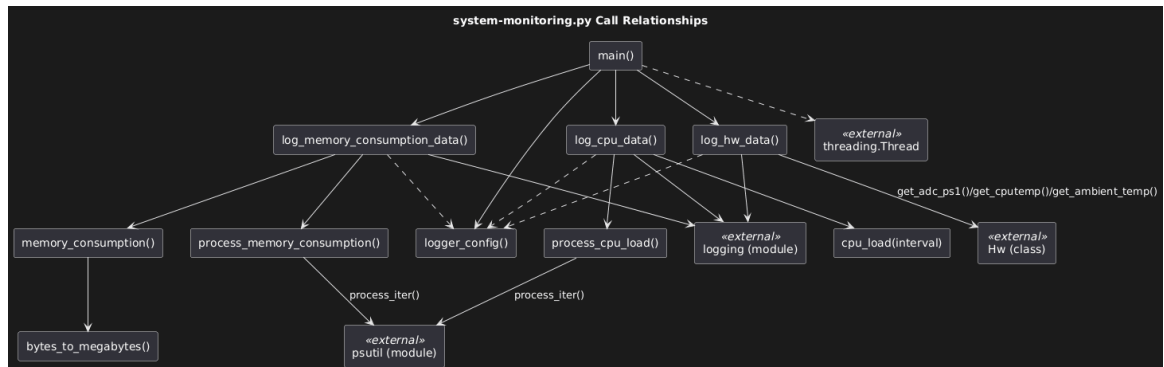
### 3.2 System-Monitoring Service

System-Monitoring is a Python-written systemd service, integrated into the LCU's Linux image using a Yocto project recipe. Figure 9 shows the files used for setting up the system-monitoring service in the LCU.



**Figure 9.** *system-monitoring service files*

- `file://system-monitoring.py`: The Python script containing the LCU monitoring algorithm, explained in subchapter 3.2.1 to 3.2.4. Figure 10 shows the relations of methods inside this file.



**Figure 10.** *system\_monitoring.py method relations*

- `file://system-monitoring.service`: The systemd service unit file that configures how the service runs. This file is discussed in subchapter 3.2.5
- `file://syslog_client.conf`: This configuration file specifies how logs are handled and processed, the destination of log messages, and the protocol for transferring them. This file is discussed in subchapter 3.2.5

This section covers the implementation steps for the service, describes how system-monitoring service works, setting the LCU as a syslog-ng client and installing the service as a Yocto recipe to the LCU's image.

### 3.2.1 Syslog-ng configuration

The `logger_config()` method sets up loggers writing to system, using `logging.config` package. A dictionary in json format that controls how the logging system operates, such as logging format, handlers, and default logging level, which is then parsed into the `logging.config.dictConfig` function. Default logging level allows only messages with higher level than default to write to `journald`. `logger_config()` takes the name of the logger and default logging level as function arguments.

**Program 1. logger config method**

```

1  def logger_config(name, loglevel):
2      """
3      syslog configuration
4      """
5      log_config = {
6          "version": 1,
7          "disable_existing_loggers": False,
8          "formatters": {
9              "journal": {
10             "format": '%(levelname)s \
11             [SYSTEM_MONITORING] %(message)s',
12             }
13         },
14         "handlers": {
15             "journal": {
16                 "class": "systemd.journal.JournalHandler",
17                 "level": loglevel,
18                 "formatter": "journal",
19                 "SYSLOG_IDENTIFIER": name
20             }
21         },
22         "loggers": {
23             name: {
24                 "handlers": ["journal"],
25                 "level": loglevel
26             }
27         }
28     }
29
30     logging.config.dictConfig(log_config)
31     logger = logging.getLogger(name)
32     return logger

```

The dictionary has the following parameters to configure syslog-ng's behavior:

- **Formatters:** the formatting layout for log entries, which includes the logging level (`%(levelname)s`), name of the system service (`[SYSTEM_MONITORING]`), and message content (`%(message)s`).
- **Handlers:** defines the handlers that will process log messages. The configuration includes the handler class (`systemd.journal.JournalHandler`), the logging level threshold (`level`), the corresponding formatter above, and the name of the logger. `systemd.journal.JournalHandler` class sends log messages from the Python application to the systemd journal, which discussed in subchapter 2.6.
- **Loggers:** set up specific loggers. The logger configuration uses the "journal" handler and default `loglevel`.

### 3.2.2 Monitoring CPU and Memory consumption

To monitor CPU and Memory consumption, `psutil` (python system and process utilities) package for Python was used. `psutil` is a package which can retrieve system and running process's information, such as CPU, memory, disks, network [35]. This package is mainly used for system monitoring, profiling and managing processes [35].

`psutil.virtual_memory()` method responsible for observing systems virtual memory usage in byte as a tuple. Support method `bytes_to_megabytes()` converts byte to megabyte. From this tuple, total physical memory, available memory can be given to processes and memory occupied by processes can be extracted. Method `memory_consumption()` packs memory information to a list.

**Program 2.** *psutil.virtual\_memory() method*

```

1  def memory_consumption():
2      """
3      find memory consumption(virtual) of the system
4      """
5      memory_stats = psutil.virtual_memory()
6
7      total_mem = bytes_to_megabytes(memory_stats.total)
8      avail_mem = bytes_to_megabytes(memory_stats.available)
9      used_mem = bytes_to_megabytes(memory_stats.used)
10     mem_usage_per = memory_stats.percent
11
12     return [total_mem, avail_mem, used_mem, mem_usage_per]
```

The percentage of memory consumed by processes can be calculated by method (1):

$$\frac{total - available}{total} \times 100 \quad (1)$$

`psutil.cpu_percent()` returns the current universal system CPU utilization in percentage. This method takes the time interval between measurements in seconds and a Boolean argument to determine whether CPU consumption of overall CPU usage in average over the system or individual cores if the CPU has multiple cores. In this project, sample rate was set to 5 seconds and monitoring CPU usage over all cores.

**Program 3.** *cpu\_load method*

```

1  def cpu_load(interval):
2      """
3      find cpu load average of available CPUs
4      """
5      cpu_load = psutil.cpu_percent(interval=interval)
6      return cpu_load
```

`psutil.process_iter()` method gets data on all processes running in the system. This method returns a Process object containing information on each running process in

the local machine. `process_memory_consumption()` and `process_cpu_load()` use `psutil.process_iter()` to construct running process's dictionaries.

**Program 4. process memory consumption object**

```

1  def process_memory_consumption():
2      """
3      find memory consumption of running processes,
4      with name, pid and command
5      """
6      process_memory_dict = dict()
7      processes = psutil.process_iter(['pid', 'name', \
8          'cmdline', 'memory_info'])
9
10     for p in processes:
11         try:
12             p_info = p.info
13             pid = p_info['pid']
14             process_name = p_info['name']
15             command = [c for c in p_info['cmdline'] if c.strip()]
16             memory_usage = p_info['memory_info'].rss / 1024 / 1024
17             if memory_usage > 0:
18                 if process_name in process_memory_dict:
19                     process_memory_dict[process_name].append((pid, \
20                         ' '.join(command), memory_usage))
21                 else:
22                     process_memory_dict[process_name] = [(pid, \
23                         ' '.join(command), memory_usage)]
24             except (psutil.NoSuchProcess, psutil.ZombieProcess):
25                 pass
26
27     return process_memory_dict

```

**Program 5. process CPU consumption object**

```

1  def process_cpu_load():
2      """
3      find cpu load of processes across all CPUs,
4      with name, pid and command
5      """
6      cpu_process_dict = dict()
7      processes = psutil.process_iter(['pid', 'name', \
8          'cmdline', 'cpu_percent'])
9
10     for p in processes:
11         try:
12             p_info = p.info
13             pid = p_info['pid']
14             cpu_percent = p_info['cpu_percent']
15             process_name = p_info['name']
16             command = [c for c in p_info['cmdline'] if c.strip()]
17
18             if cpu_percent > 0:
19                 if process_name in cpu_process_dict:
20                     cpu_process_dict[process_name].append((pid, \
21                         ' '.join(command), cpu_percent))
22                 else:
23                     cpu_process_dict[process_name] = [(pid, \

```

```

24         ' '.join(command), cpu_percent)]
25     except (psutil.NoSuchProcess, psutil.AccessDenied):
26         pass
27     return cpu_process_dict

```

From `psutil.process_iter()` objects, the process's name, process identifier (PID), command executed the process in the system and CPU usage, memory consumption can be retrieved. This information was stored in a dictionary with the process 'name as key and other data as values, packed in a list. If the process name is not found in the dictionary, a new list with the current process's information is created and assigned as the value for that process name key in the dictionary. If the process name already exists as a key in the dictionary, a new list is appended to the existing list associated with the process name, allows grouping processes with the same name but different PID and command line.

### 3.2.3 Monitoring Hardware status

Each LCUs from Teknoware contains a python file called `machine_board_name.py` to describe the hardware of the LCU, defining GPIO and mapping GPIOs to PWM ports, and Analog to Digital Converter (ADC) for PWM control, by reading the data from `sysfs` file system and making calculation of voltage and temperature based on these data. This file was contributed by Marku Vorne and software team members. The voltage and temperature values depend on the ADC values from the corresponding files taken from the `/sys` directory. ADC is a converter, turning analog signals, for example, voltage or sensors reading, into digital values that LCU's software can process. After the ADC value is obtained, it needs to be converted to analog value for human to read. This work used some methods from this file to calculate temperature and voltage. `machine_board_name.py` is installed to the image, by using a recipe and machine configuration file in `/conf`.

The information of voltage and temperature are store in the following locations of `sysfs` file system:

- **Voltage ADC value:** `/sys/bus/iio/devices/iio:device0/in_voltage0_raw`
- **Voltage ADC scale:** `/sys/bus/iio/devices/iio:device0/in_voltage_scale`
- **LCU temperature ADC value:** `/sys/bus/iio/devices/iio:device1/in_voltage1_raw`
- **LCU temperature ADC scale:** `/sys/bus/iio/devices/iio:device1/in_voltage_scale`
- **CPU temperature:** `/sys/devices/virtual/thermal/thermal_zone0/temp`

`_read_adc()` method used to calculate voltage value, using raw ADC value and ADC scale read from sysfs files as inputs. Raw ADC is the digital value obtained from the ADC hardware, while the ADC scale is the scale factor, which converts the raw ADC value back to voltage. The result of the calculations, which are the product of raw ADC and scale, was in millivolts (mV).

The internal voltage was the result of the input voltage that passed through the voltage divider circuits, so the voltage is enough for the CPU. The voltage divider contains 22000 and 1500 Ohms resistors. The internal voltage needs to be converted back to original value; by reversing the voltage divider formula (2), the result is in V:

$$V_{internal} = \frac{22000+1500}{1500 \times 1000} \times adc\ value \times scale\ factor \quad (2)$$

CPU temperature can be obtained directly from the corresponding sysfs file and divided by 1000.

LCU temperature can be calculated, using Negative temperature coefficient (NTC) thermistor to measure the LCU's ambient temperature. NTC thermistor resistance drops when the temperature rises, as such, a list of NTC thermistor resistances at different temperatures was provided in the thermistor datasheet. The temperature curve of the NTC in the LCU ranges from -55 to 125 °C. Based on this curve, a list of voltage corresponding to the temperature can be calculated by (3), as NTC is part of voltage divider circuit:

$$V_{NTC} = V_{in} \times \frac{R_{NTC} \times rescof}{R_{NTC} \times resconf + R_{div}} \quad (3)$$

$R_{NTC}$  is the resistance of the thermistor,  $R_{div}$  is the resistance of the resistor used in the measurement circuit and  $rescof$  is the resistor coefficient from the list of NTC resistance of temperature curve.

`get_ambient_temp()` method was created to calculate the voltage of the thermistor, based on the raw ADC value. Afterward, linear interpolation for temperature estimation based on ADC value and  $V_{NTC}$ . First, voltage was calculated by `_read_adc()`. Then the method iterates through  $V_{NTC}$  (`ntc_voltage_list` in the code) to find the first voltage index in the list that is lower than voltage calculated by `_read_adc()`.

#### **Program 6. ambient temperature calculation**

```

1 def get_ambient_temp(self) -> int:
2     ''' get ambient temperature in celcius'''
3     # convert from mV to V
4     adc_value = self.get_adc_temp_sensor() / 1000
5
```

```

6      # If the measured voltage is higher than the highest value
7      # in the NTC curve, return -55°C
8      if adc_value >= self.ntc_voltage_list[0]:
9          return -55
10
11     index = 0
12     found = None
13
14     # Search for the first index in ntc_voltage_list
15     # where adc_value is greater than the curve value
16     for index, voltage in enumerate(self.ntc_voltage_list):
17         if adc_value > voltage:
18             found = index
19             break
20
21     if found is None:
22         # temperature is out of the curve range
23         return 125
24
25     # Calculate the temperature based on the index
26     temp = -55 + index * 5
27     # Interpolate between points
28     temp -= 5 * (adc_value - self.ntc_voltage_list[index]) / \
29     (self.ntc_voltage_list[index - 1] - \
30     self.ntc_voltage_list[index])
31
32     return int(round(temp))

```

The calculation of temperature based on linear interpolator (4):

$$T = T_{low} - (T_{low} - T_{high}) \times \frac{V_{ADC} - V_{low}}{V_{high} - V_{low}} \quad (4)$$

where  $T_{low}$  is  $-55^{\circ}\text{C} + \text{index} * 5^{\circ}\text{C}$ ,  $T_{high}$  is  $-55^{\circ}\text{C} + (\text{index} - 1) * 5^{\circ}\text{C}$ ,  $V_{low}$  is `ntc_voltage_list[index]` and  $V_{high}$  is `ntc_voltage_list[index - 1]`. The difference between  $T_{low}$  and  $T_{high}$  is  $-5^{\circ}\text{C}$ , because the difference between temperature in each temperature in the temperature curve is  $5^{\circ}\text{C}$ .

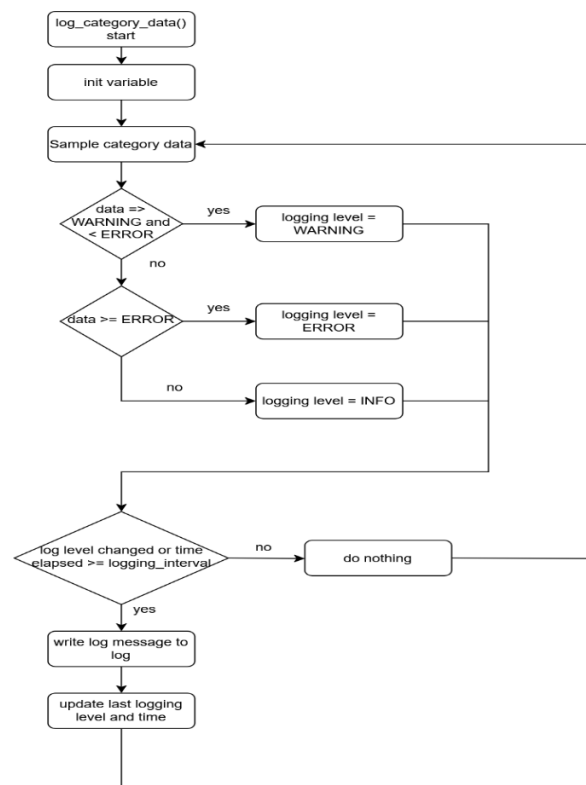
### 3.2.4 Logging System's status

`log_cpu_data()`, `log_memory_consumption_data()`, `log_hw_data()` generate log messages of the LCU status of the corresponding category, save the lines to local log file, which was specified in the `syslog-ng` recipes, and continuously sends log messages to the `syslog-ng` server. These methods generate log messages every 5 minutes. The level of log message is based on the current value of the LCU status if it exceeds the predefined thresholds. Logging methods use `logging` package for Python. Table 1 shows details of thresholds of the LCU status:

Table 1. *LCU status threshold*

Metric	Warning Threshold	Error Threshold
Memory usage (%)	25.0	35.0
CPU temperature (°C)	45.0	50.0
LCU temperature (°C)	35.0	40.0
Supply voltage (V)	11.0	10.0
CPU consumption (%)	80.0	90.0

Figure 11 describes the algorithm, which `log_cpu_data()`, `log_memory_consumption_data()` and `log_hw_data()` use to generate log message and determine level of the messages.



**Figure 11.** *system-monitoring logging algorithm.*

According to the diagram, the logging level of the message was determined by the following description :

- `WARNING_THRESHOLD_STATUS` : If data value is above this threshold but below the `ERROR` threshold, the level is set to `logging.WARNING`.
- `ERROR_THRESHOLD_STATUS` : If data value is at or above this threshold, the level is set to `logging.ERROR`.

- Otherwise, if the data value is below `WARNING_THRESHOLD_STATUS`, the level is set to `logging.INFO`.

The log message will be written by the following condition, to prevent log spamming:

- `current_logging_level != last_logging_level`: logging if the level has changed from the previous event.
- `elapsed_time >= LOGGING_INTERVAL`: logging happens every `LOGGING_INTERVAL` (5 minutes), even if the level stays the same .

`log_cpu_data()`, `log_memory_consumption_data()` and `log_hw_data()` run in the `main()` method, by creating threads for each method. In addition, the log object created from `logger_config()` is initialized in `main()` method, by creating logger name `system-monitoring-service`, and allows log message with INFO level and above to be logged into files and transfer through syslog-ng protocol.

### 3.2.5 Systemd setup

To set up LCU as a syslog-ng client, system monitoring service file and syslog-ng configuration file need to be created.

`system-monitoring.service` determines how system-monitoring should be managed by the system. After installation, this file locates in `~/lib/systemd/system`

#### **Program 7.** *system-monitoring service file*

```

1  [Unit]
2  Description= LCU status monitoring service
3  After = network.target
4
5  [Service]
6  SyslogIdentifier=system-monitoring-service
7  Type = simple
8  ExecStart=/usr/bin/system-monitoring.py
9
10 [Install]
11 WantedBy=multi-user.target

```

This file is configured as follows:

- `[Unit]` Section:
  - **After**: specifies dependencies on the service. system-monitoring service should start after the network.target service initialized.
- `[Service]` Section:

- **SyslogIdentifier:** describes a unique identifier that is used in journal entries. In this case, `system-monitoring-service` will be used in the log to represent the message generated from system monitoring service.
- **Type:** type of service, set to `simple`, which begins when the system boots up and runs continuously.
- **ExecStart:** the command to execute when starting the service, execute the `system-monitoring.py` script, located at `/usr/bin/system-monitoring.py`.

`syslog_client.conf` is an addition support for `syslog-ng` service, extending `syslog-ng` features without overriding it, as `syslog-ng` is also a `systemd` service. `syslog_client.conf` can be added to `~/etc/syslog-ng/conf.d`. This configuration defines how log messages should be sent to remote syslog servers, by specifying the source, destination and filter out messages before sending, in this case, messages with level “debug” will not be sent.

**Program 8.** *extension for syslog-ng service in the LCU*

```

1  block root longterm_syslog(host(), port(514), transport("udp")) {
2    log {
3      source(s_system);
4      source(s_internal);
5
6      filter {
7        not level(debug) and
8        not filter(f_loopback)
9      };
10
11     destination {
12       syslog(`host`, port(`port`),
13       transport(`transport`));
14     };
15   };
16 };
17
18 # Uncomment this line to enable sending log
19 # to designate syslog-ng server
20 # longterm_syslog(host("192.168.88.251"));

```

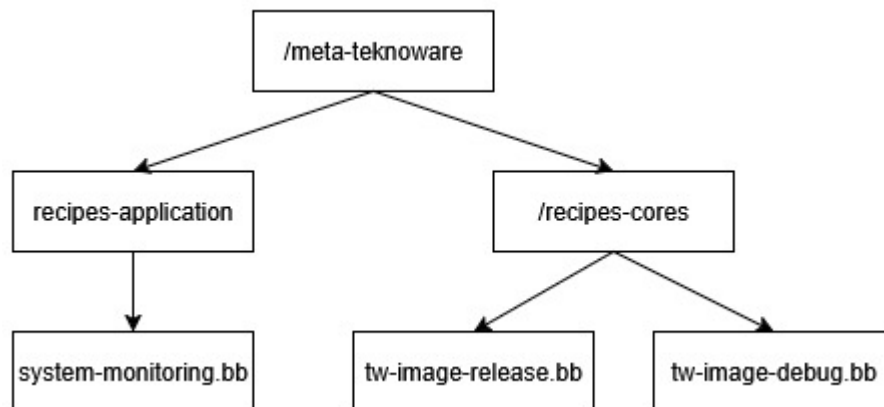
This block accepts the following arguments :

- **host():** defines the hostname or IP address of the remote syslog server. The IP address of the Test PC (192.168.88.251) was provided as the destination of the log messages.
- **port () :** defines the port on the remote server to receive syslog messages. Port 514 was used as default argument, which is the standard syslog port).

- `transport()`: sets the transport protocol for sending logs. User Datagram Protocol (UDP) was chosen as default protocol.

### 3.2.6 Yocto recipe setup

A new Yocto recipe was created and added to image of the LCU as a `system-monitoring.bb` file. This recipe gathers the files into packages and installs the packages to the target machine. Figure 12 shows the location of `system-monitoring.bb` in meta-teknoware layer.



**Figure 12.** *system-monitoring* recipe in meta-teknoware layer

#### Program 9. *system-monitoring* recipe file

```

1  DEPENDS = "systemd"
2
3  FILESEXTRAPATHS:append:="${BSPDIR}/applications/system-
4 monitoring/LCU:"
5
6  SRC_URI += "file://system-monitoring.py"
7  SRC_URI += "file://system-monitoring.service"
8  SRC_URI += "file://syslog_client.conf"
9
10 do_install () {
11     install -d ${D}${bindir}
12     install ${WORKDIR}/system-monitoring.py \
13     ${D}${bindir}/system-monitoring.py
14
15     install -d ${D}${systemd_unitdir}
16     install -d ${D}${systemd_unitdir}/system
17     install -m 0644 ${WORKDIR}/system-monitoring.service \
18     ${D}${systemd_unitdir}/system/system-monitoring.service
19
20     install -d ${D}${sysconfdir}/syslog-ng/conf.d
21     install -m 644 ${WORKDIR}/syslog_client.conf \
22     ${D}${sysconfdir}/syslog-ng/conf.d/syslog_client.conf
23 }
24
25 FILES:${PN} = " \
26     ${bindir}/system-monitoring.py \

```

```

27         ${systemd_unitdir}/system/system-monitoring.service \
28         "${sysconfdir}/syslog-ng/conf.d/syslog_client.conf \
"

```

This recipe file has been configured as follows:

- **DEPENDS:** system-monitoring is a systemd service, as such systemd must be present in the build environment for this recipe to work.
- **FILESEXTRAPATHS\_append:** files from `~/applications/system-monitoring/LCU`.
- **SRC\_URI:** source files required for building the service, described in Figure 9
- **do\_install:** steps to install the software onto the target system. Firstly, directories were created using `install -d` for binaries (`${bindir}`), systemd service files (`${systemd_unitdir}/system`), and syslog-ng configuration files (`${sysconfdir}/syslog-ng/conf.d`). Afterward, files in **SRC\_URI** were copied into these directories.
- **FILES\_\${PN}** directs the location where the files are installed.

`system-monitoring` can be installed to Yocto debugging image `updateimg-debug` by using `IMAGE_INSTALL` in the image's recipe. There are 2 different building images for LCU: `updateimg-debug` and `updateimg-release`. `updateimg-debug` contains `updateimg-release`, with addition of debugging tools, such as ssh connection.

**Program 10.** *install system-monitoring to device's image*

```

1  # system monitoring service
2  IMAGE_INSTALL += "system-monitoring"

```

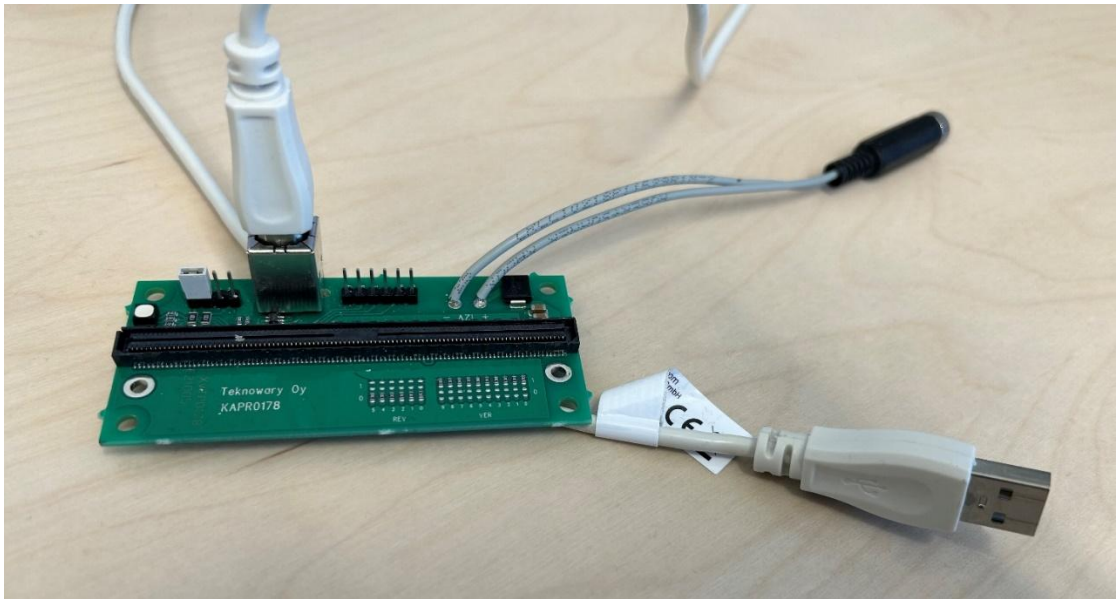
To build the image, Yocto build environment needs to be initialized. At `~/yocto`, where `setup-environment` located, using the following command to initiate a building environment for the target machine LCU:

```
source setup-environment LCU
```

The build environment for the target machine is `~/yocto/build`. To build the image, run the following command in `~/yocto/build`:

```
bitbake updateimg-debug
```

Bitbake can be used to install bootloader and flash environment for the image. The newly built image can be loaded into the LCU using flash device. The CPU card of the LCU can be detached from the LCU and inserted into the flash device, as shown in Figure 13.



**Figure 13.** *image flashing device.*

The flashing device is needed when the built image's target machine is different from the current target machine. The flash environment requires several files generated from the built image and flash executable file, the LCU's device tree, bootloader, image, and flash script. The flasher script is responsible for bundling required files together for the flash environment and copies these files to a dedicated location.

After the flash environment has been initiated, connect the flashing device to the PC through USB cable and run the file `tw-image-debug-LCU-flash.bat` to flash the image to the CPU card. This file runs the flash execution command, which instructs the flash environment to load the files in the flash environment onto the device's eMMC storage. When the flashing is finished, the CPU card can be removed and inserted back into the LCU and boot again. After the first installation, software update can be done by loading `.swu` file of the same target machine, generated from `bitbake`, to the CPU card, by using the LCU's maintenance web GUI.

### **3.3 Test PC test environment setup**

This section describes how the test PC was set up as a syslog-ng server, set up the train network simulator on the test PC and configure the Teams Channel to receive log messages from the test PC. In this work, syslog-ng was used in Linux environment, as such a virtual machine with Ubuntu distribution was set up in the Test PC.

### 3.3.1 Syslog-ng server setup

syslog-ng package needs to be installed in the test PC to use the syslog-ng service, with the following command in Ubuntu:

```
sudo apt-get install syslog-ng -y
```

server\_syslog.conf was used to extend functionality for the syslog-ng server. This file should be put under the path `~/etc/syslog-ng/conf.d/`, as the extension of the existing `syslog-ng.conf`.

**Program 11.** *syslog-ng extension file in the test PC*

```

1  @define allow-config-dups 1
2  source s_network {
3      udp(ip(0.0.0.0) port(514));
4  };
5
6  template t_syslog {
7      template("${YEAR}-${MONTH}-${DAY}T${HOUR}:${MIN}:${SEC}Z \
8      ${LEVEL} ${MSGHDR}${MSG}\n");
9  };
10
11 #filter f_filter {
12 # not match("tw-portmonitor" value("MESSAGE"));
13 #};
14
15 destination d_local {
16     file("/var/log/LCU_${HOST}.log" template(t_syslog));
17 };
18
19 log {
20     source(s_network);
21     #filter(f_filter);
22     destination(d_local);
23 };

```

The following parameters have been configured to let Test PC serve as syslog-ng server:

- `source s_network`: define the source to receive UDP on port 514, from any IP address (0.0.0.0).
- `template t_syslog`: the format of the log messages, which will be saved in the Test PC.
- `destination d_local`: set the file names and destinations of log messages. The log messages received from the LCUs are saved in the path `~/var/log/LCU_<hostname>.log`. `hostname` is the IP address of the corresponding LCU. As a result, there are 6 LCU log files saved in the test PC.

`template(t_syslog)` specifies that the `t_syslog` template defined earlier will be used to format the log messages before writing them to the file.

- `log`: Defines the main log processing pipeline, calling the above code blocks.

After the configuration is finished, `syslog-ng` server can be started by using the following command:

```
systemctl enable syslog-ng
```

```
systemctl start syslog-ng
```

### 3.3.2 Syslog-ng Regex configuration

`regex.conf` filters log messages obtained from the LCUs, before sending these events to Teams Channel, using regex. If new events need to be monitored, new regex can be added to this file.

#### **Program 12.** *regex configuration file*

```

1  #configuration for regular expression when reading log files.
2  [description]
3  version_format = 3
4  version = 1.0.0.1
5
6  [lcu_regex]
7  lcu_reboot_regex = .* resetreason-startup\.sh .* - - reset reason.*
8  service_reboot_regex = .* systemd .* - - Stopping .*
9  train_network_disconnect = .* protocol-daemon .* - - *closed .*
10
11 [system_monitoring_regex]
12 memory_regex = .*- - (\w+) \[SYSTEM_MONITORING\] Total memory.*
13 CPU_regex = .*- - (\w+) \[SYSTEM_MONITORING\] CPU load average.*
14 temp_regex = .*- - (\w+) \[SYSTEM_MONITORING\] Voltage.*
15
16 [log_files_path]
17 log_path = /var/log/LCU_*.log

```

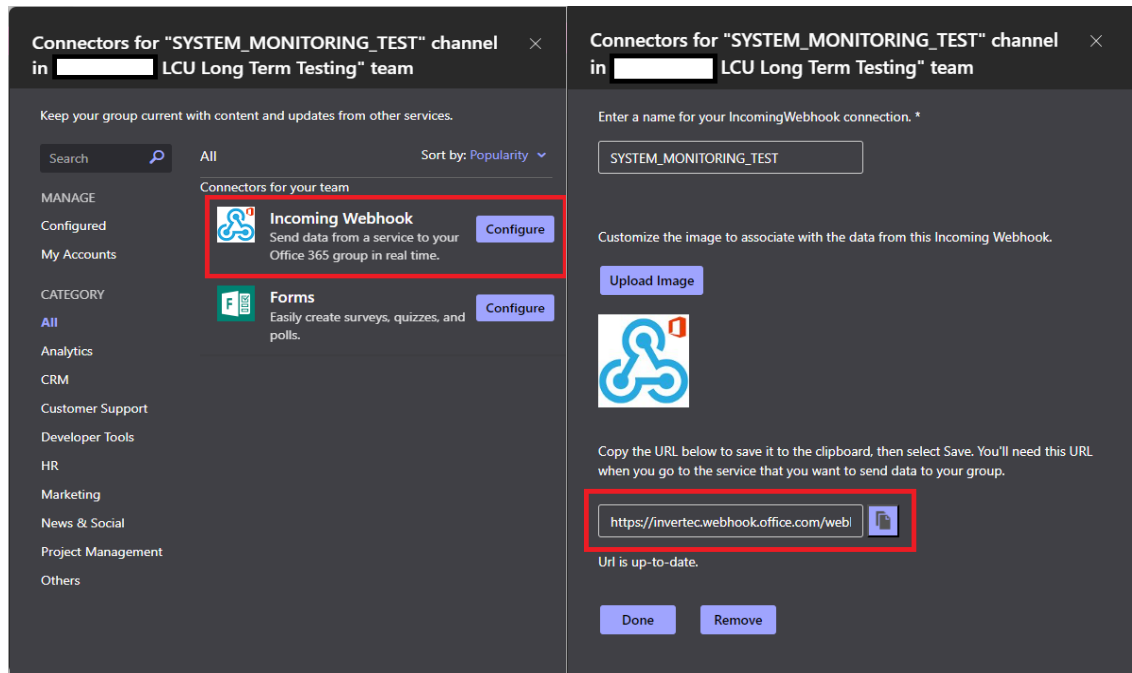
The regex-es are split into different categories:

- `[lcu_regex]`: This section is currently used for regexs related to LCU reboot or any system services reboot. The log lines matched the regexs from this section can be sent to Teams without restrictions.
  - o `lcu_reboot_regex`: targets lines containing "resetreason-startup.sh" and "reset reason", which identify L reboot event with its reset reason, either by rebooting manually or force rebooting by the watchdog.
  - o `service_reboot_regex`: searches for lines with "systemd" and "Stopping", identifying systemd services restarts.

- o `train_network_disconnect`: looks for lines with "protocol-daemon" and "closed", indicates the disconnect between train simulator and the LCUs event.
- `[system_monitoring_regex]`: This section is currently used for regexs related to system-monitoring service. The log lines matched the regexs in this section always have `[SYSTEM_MONITORING]` identifier, only the lines with logging level of WARNING and above can be sent to Teams to avoid spamming.
  - o `memory_regex`: looking for lines containing "Total memory" that captures memory consumption related.
  - o `CPU_regex`: similar memory regex, this pattern targets lines with "CPU load average", capturing the CPU load event.
  - o `temp_regex`: This regex targets lines with Voltage, contains both temperature and voltage values.
- `log_path = /var/log/LCUs_*.log`: path that all log files start with "LCU\_" and end with ".log" within the `/var/log/` directory that syslog-ng save log messages in the test PC. \* is the wildcard matches the IP address of the LCU.

### 3.3.3 Teams Channel set up.

To connect the Teams Channel with `teams.py` message forwarding script, a webhook URL needs to be provided into the script. According Sumrak Jesse, webhook is a method that allows applications to send messages and information directly to other applications, such as Microsoft Teams channel, in real-time, by using Hypertext Transfer Protocol (HTTP) request [36]. Data through webhook is sent by HTTP POST request, in form of JSON format. In this project, webhook URL and the application exchanges data with each other using Json-format payload. Detail implementation will be described in section 3.3.4. Figure 14 shows the configuration in Team's GUI. After the setup is completed, a URL of the webhook is created. This URL is provided to the message forwarding script to set up connection between the script and the Channel.



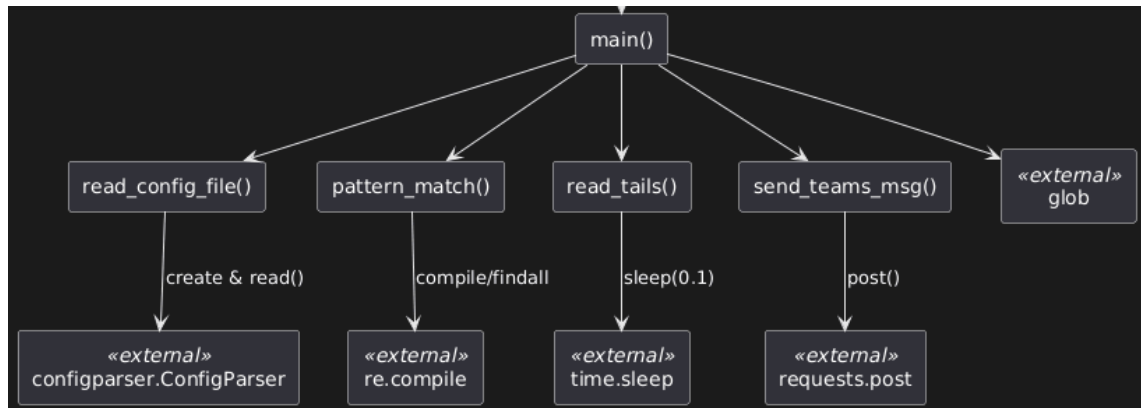
**Figure 14.** Teams Channel Webhook configuration

### 3.3.4 Teams Channel message forwarding script.

`teams.py` script was developed in Python to read log messages that came from the LCUs and forward the message to Teams Channel. This script has the following features:

- Reading configuration file `regex.conf` to retrieve regexes and log file's paths and finds log lines matched the regex.
- Reading the log files, found in `regex.conf` log file's paths and constantly monitoring if the end of each log file has new lines.
- Setup connection to transfer wanted log messages to Teams channel, by using HTTP request.

Figure 15 shows the method relations in `teams.py`



**Figure 15.** *teams.py method relations*

To support log messages filtering, utility methods `read_config_file()` and `pattern_match()` were created. `read_config_file()` takes a string representing the path to a configuration file as input. It utilizes the `configparser` library to parse the file by creating a `configparser` object and then uses the `read` method to read the configuration data from the specified file path and returns the parsed configuration data as a `configparser` object.

**Program 13.** *read\_config\_file method*

```

1 def read_config_file(inputpath: str):
2     """
3     read configuration file
4     """
5     config = configparser.ConfigParser()
6     config.read(inputpath)
7     return config
  
```

`pattern_match()` uses `re` package, which is regex support package, to check whether a given string matches a regex pattern provided as the `regex` argument, by parsing the regex into `re.compile` method to create a pattern object. Afterward, using `re.compile.findall` method to search for all occurrences of the pattern within the input string. The function returns a list containing all the matches if a match is found.

**Program 14.** *pattern\_march method*

```

1 def pattern_match(regex, line: str):
2     """
3     check if input string match regular expression
4     """
5     re_object = re.compile(regex)
6     match = re_object.findall(line)
7     if match:
8         return match
9     return None
  
```

`read_tails()` is a utility method for getting every new line from the log files stored in the test PC. It maintains a dictionary to store open file objects. For each file, the method

read and put the file pointer at the end using `seek(0, 2)`. If the new line is detected, the function yields a tuple containing the filename and the new line. If no new lines are found, `read_tails()` wait for a short duration (0.1 seconds) before continuing the loop, rechecking the files for updates.

**Program 15. `read_tail` method**

```

1  def read_tails(file_list):
2      """
3      read from the end of files and continue
4      if files were updated with new lines
5      """
6      # store open file's objects
7      files = dict()
8      for f in file_list:
9          files[f] = open(f, 'r')
10         # move file pointer to the end of the file
11         files[f].seek(0, 2)
12
13     while True:
14         for f in files:
15             line = files[f].readline()
16             # if no new lines, wait for 0.1 sec and continue
17             if not line:
18                 time.sleep(0.1)
19                 continue
20             yield (f, line)

```

`send_teams_msg()` forwards messages to Teams channel. The webhook URL, message content, message color, and an optional title are needed as arguments. In this work, the `requests` package was used to create HTTP requests for messages transmission. The method creates a JSON-formatted payload that follows the Microsoft Teams message card schema. This payload includes the message, color-coded of the severity level, title and log messages. `request.post` method transmits a POST request to the Teams webhook URL, containing the customized payload in JSON form with the headers. The function returns the HTTP response from Teams.

**Program 16. `send_team_message` method**

```

1  def send_teams_msg(webhook_url: str \
2      , message: str, color: str, title: str) -> int:
3      """
4      send message to TEAMS channel
5      """
6      headers = {
7          "Content-Type": "application/json"
8      }
9      # Define the payload with the message
10     payload = {
11         "@type": "MessageCard",
12         "@context": "http://schema.org/extensions",
13         "themeColor": color,
14         "title": title,

```

```

15         "text": message
16     }
17     # Send a POST request to the Teams webhook URL with the payload
18     response = requests.post(webhook_url, \
19                             json=payload, headers=headers)
20
21     return response.status_code

```

The main algorithm of `teams.py` is in the `main()` method, which assembles the above explained methods together, as shown in Figure 15. The `main()` method uses the following global variable.

**Program 17.** *color of message card in Teams channel depends on logging level*

```

1  # color of each logging level displayed in Teams message
2  logging_color = {
3      "DEBUG": '#ffffff',
4      "INFO": '#33cc33',
5      "WARNING": '#ffff00',
6      "ERROR": '#ff6600',
7      "CRITICAL": '#ff0000'
8  }

```

**Program 18.** *global variables used in main()*

```

1  # logging level can be sent to Teams channel
2  teams_message_severity = ["WARNING", "ERROR", "CRITICAL"]
3  # webhook of teams channel, change the webhook url if needed.
4  webhook_url = ""
5  # list of regex-es used for system monitoring
6  sysmor_regex_list = list()
7  # list of regex-es detects rebooted LCU/systemd service
8  lcu_regex_list = list()
9  # section in regex config file of system monitoring
10 sysmor_section = 'system_monitoring_regex'
11 # section in regex config file detecting rebooted LCU/systemd service
12 lcu_section = 'lcu_regex'
13 # dictionary of previous logging level of each regex in
14 sysmor_regex_list
15 previous_log_level = dict()

```

First, `main()` starts by reading the regex configuration file (`CONFIG_FILE`) to gather information on the log file path (`log_path`), LCU or service reboot (`lcu_section`) and system monitoring (`sysmor_section`). Afterwards, it takes regex from those segments and generates a list for each regex-es category.

**Program 19.** *reading regex configuration file and creating lists of regex*

```

1  # init variables
2  config = read_config_file(CONFIG_FILE)
3  log_path = config.get("log_files_path", "log_path")
4
5  files = glob.glob(log_path)
6  loglines = read_tails(files)

```

```

7
8 # init list of of regex based on section in config file
9 if lcu_section in config:
10     lcu_section_items = config.items(lcu_section)
11     for key, value in lcu_section_items:
12         lcu_regex_list.append(value)
13
14 if sysmor_section in config:
15     sysmor_section_items = config.items(sysmor_section)
16     for key, value in sysmor_section_items:
17         sysmor_regex_list.append(value)

```

Afterward, `previous_log_level()` stores past logging levels of events matching regex in `sysmor_regex_list`, for each IP address discovered in the log file directories. If the IP address is not found in `previous_log_level()`, creates new dictionary for that IP and `sysmor_regex_list` events inside `previous_log_level()` and sets the recent logging level as INFO. This list is needed to restrict messages with INFO level and below to be sent to Teams channel.

**Program 20. adding log to previous\_log\_level**

```

1 # init previous logging level
2 # to all regex for each IP addresses found
3 for reg in sysmor_regex_list:
4     for file in files:
5         ip_match = pattern_match(r'(\d+\.\d+\.\d+\.\d+)', file)
6         if ip_match:
7             ip_address = ip_match[0]
8             if ip_address not in previous_log_level:
9                 previous_log_level[ip_address] = dict()
10                previous_log_level[ip_address][reg] = 'INFO'

```

Next, the code searches each line extracted from the log files. The IP address is extracted to generate headers for Teams messages, depending on the type of monitoring (LCU or service reboot or system monitoring).

**Program 21. messege headers creation**

```

1 for file, line in loglines:
2     ip_match = pattern_match(r'(\d+\.\d+\.\d+\.\d+)', \
3     file)
4     if ip_match:
5         ip_address = ip_match[0]
6         sysmor_title = f"[SYSTEM MONITORING] : \
7         {ip_address}"
8         lcu_title = f"[LCU/SERVICE_REBOOT_MONITORING] : \
9         {ip_address}"

```

For regexes in LCU or services reboot related section, the log line is sent to Teams channel every time without restrictions. The logging level is set to INFO when displayed its color on teams.

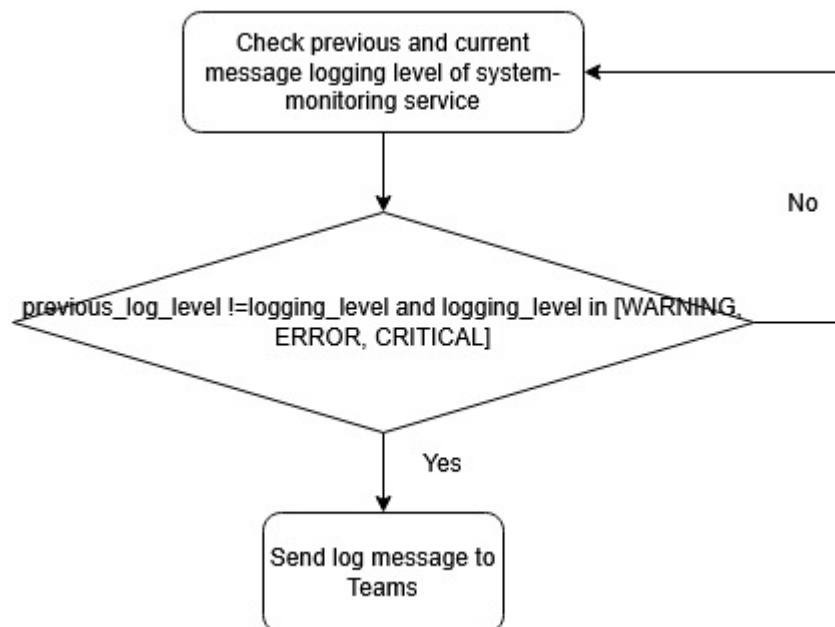
**Program 22.** send Teams messages related to LCU reboot/service reboot

```

1  for reg in lcu_regex_list:
2      match = pattern_match(reg, line)
3      if match:
4          send_teams_msg(webhook_url,line,logging_color["INFO"], \
5              lcu_title)

```

For regex-es in system monitoring section, the log line is only sent to Teams channel if the logging level of new message is different from previous message and only new message has sever logging level (WARNING, ERROR, ...) to avoid spamming. In addition, only messages with different log level than the previous log level of the same regex can be sent to Teams channel. The logging level took the color from `logging_color` in Program 17 to be displayed on Teams. Figure 16 summarized how to select message to Teams channel.



**Figure 16.** Select message from system-monitoring service to be sent to Teams channel

**Program 23.** send Teams message related to system-monitoring

```

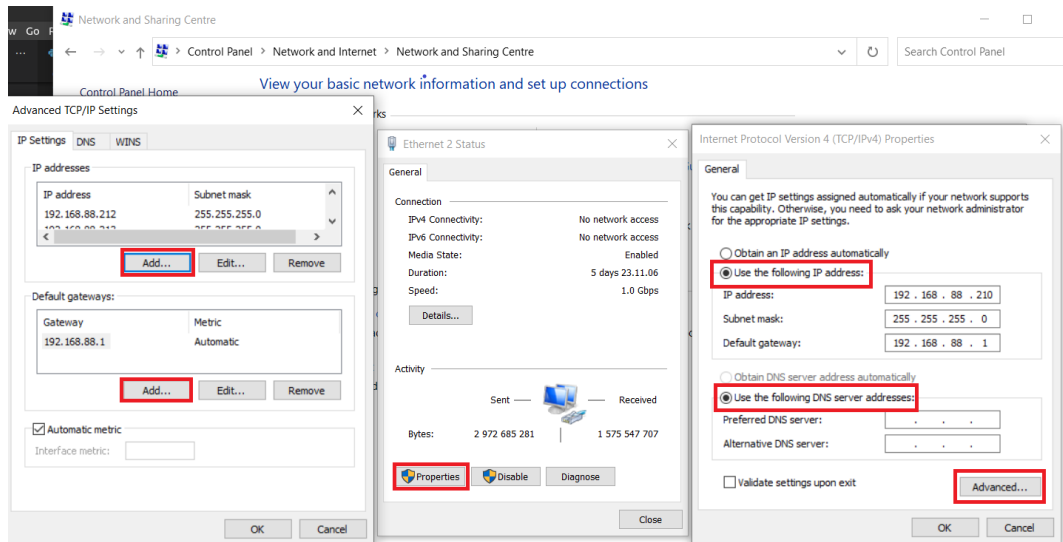
1  for reg in sysmor_regex_list:
2      match = pattern_match(reg, line)
3      if match:
4          logging_level = match[0]
5
6          if logging_level in logging_color:
7              color = logging_color[logging_level]
8          else:
9              pass
10
11         if logging_level != previous_log_level[ip_address][reg] \
12             and logging_level in teams_message_severity:

```

```
13         send_teams_msg(webhook_url, line, color, sysmor_title)
14         print(line)
15         print('')
16
17         previous_log_level[ip_address][reg] = logging_level
```

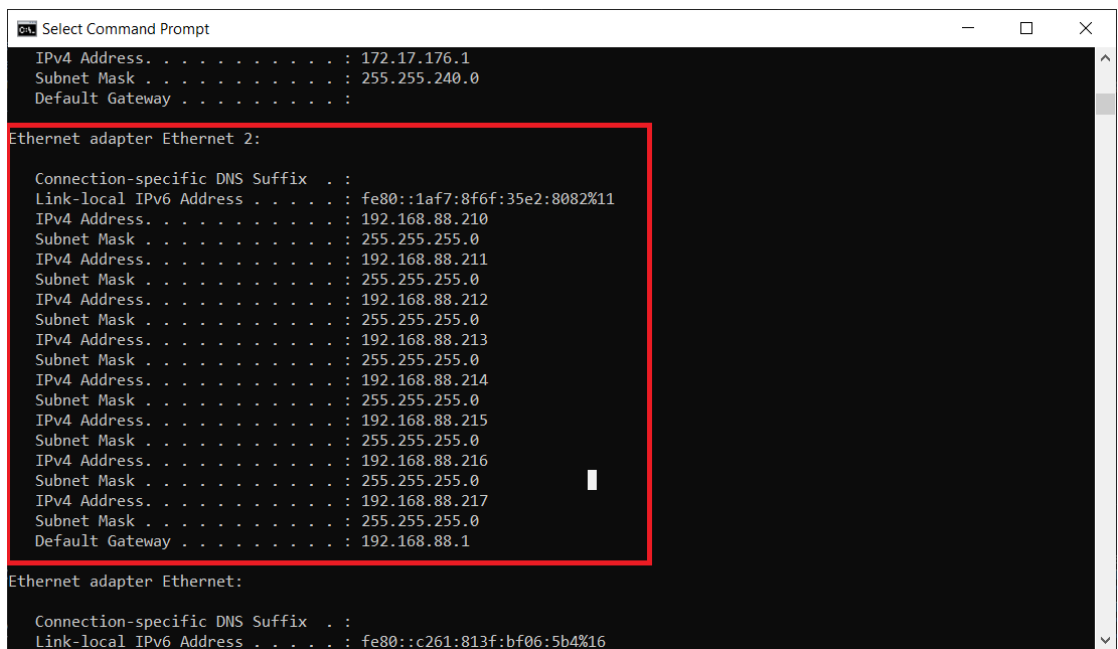
### 3.3.5 Train network setup

Train network simulator is a GUI application, developed by Teknoware, used internally to simulate connection between real train network and the LCU and how these components exchange data with each other, as the real train network hardware was not available. This application requires an ethernet setup between LCUs and the PC that the application is running. This software's interface is divided into 2 parts: LCU's status and Train network's status. LCU's status includes software and hardware version, PWM ports overloading status, current running scenario and so on. Train network status displays datetime, train's location, speed, car number and so on. In this project, the train simulator was configured to continuously generate random values of LCU's status and report back to the LCU to push the processing of the LCUs. The connection between the application and the LCU was established by communication protocol, which uses UDP channel to send and receive messages. This application allows connection between one PC's IP address and one LCU's IP address connected to the router with Dynamic Host Configuration Protocol (DHCP) server. Each LCUs were assigned with a unique IP address, provided by the router's DHCP server. To run multiple train simulators for each LCUs, it is necessary to set up multiple IP addresses on the PC's ethernet interface and bind these IPs to each LCUs when invoking the simulator software. Binding multiple LCU's IP addresses to a single PC's address causes the simulator crash. In addition, the test PC's IP address also needed to be configured within the range of the gateway provided so the test PC is in same local network as the LCUs. Figure 17 demonstrate the network interface configuration.



**Figure 17.** PC network interface configuration.

Following the ethernet setup, the PC contains six IP addresses to bound with six LCU's IP addresses. Figure 18 shows the lists of IP addresses of the LCUs after being successfully configured.



**Figure 18.** PC network interface after configuration.

The train simulator of each LCUs needs to be bound with one of the unique IP addresses configured above. The binding can be done, by running the following command to start the train simulator:

```
python trainsimu.py -a < ip-address-of-lcu> -e <ip-address-of-testPC>.
```

## 4. RESULTS AND ANALYSIS

After the software of the LCU and test PC have been set up, the test environment of this project was established in the long-term testing room like Figure 19:



**Figure 19.** LCU long-term testing setup

From the figure, the LCUs and the Test PC were connected to the same router through an Ethernet cable. Furthermore, the LCUs were supplied with 24 V power. The Test PC was configured not to shut down to prevent the disruption of log message retrieval. The train simulator was configured to randomly generate train information and be sent to the LCUs. Every two weeks, the log files of the LCUs were taken and plotted to evaluate the performance. Once the trend of the performance enters an equilibrium state, the testing stops. The test ran for two months.

The thresholds of the test can be adjusted to determine the performance of the LCUs. After passing some two weeks of investigations without exceeding the thresholds, the performance of the LCU can be decided and normalized. The message format is JSON. The message received from the LCUs, created by the System-monitoring service, and saved to the Test PC's log file has the following format:

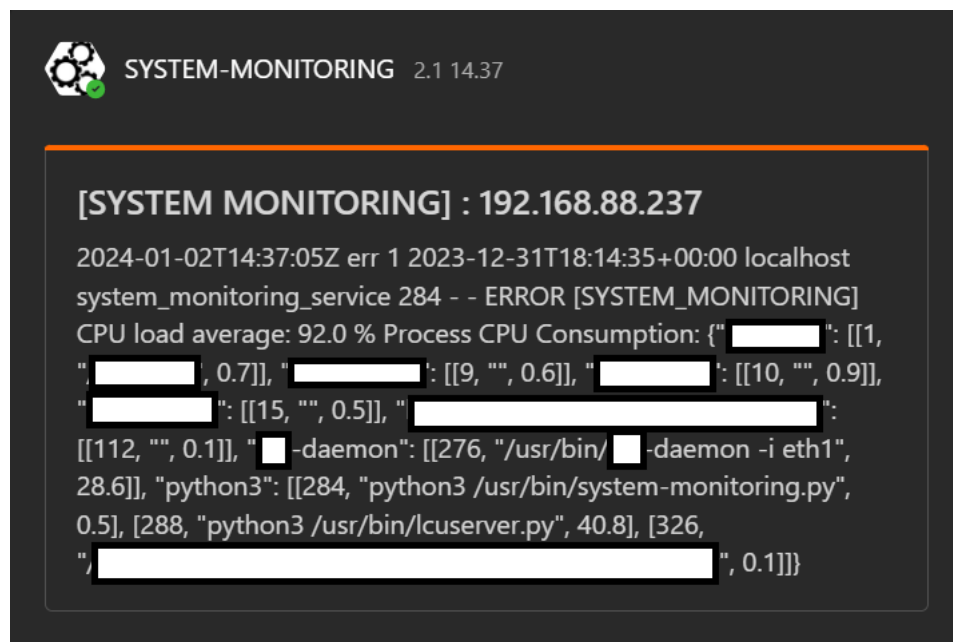
```
<test-pc-timestamp> <logging-level> <LCU-timestamp> localhost
system_monitoring_service<PID><logging-level>[SYSTEM_MONITORING]
<LCU-statistics><running-process-list-with-corresponding-
consumption>
```

- <test-pc-timestamp>: Test PC Timestamp (Coordinated Universal Time (UTC)) - the time log on the test PC.
- <logging-level>: Logging Level - severity level of the logged information (INFO, WARNING, ERROR).
- <LCU-timestamp>: LCU Timestamp (UTC) – the timestamp logged in the LCU. The LCU did not have a real-time clock component, as such the LCU synchronizes with real-time when it connects to the train simulator. This section and <test-pc-timestamp> section might be different.
- localhost system\_monitoring\_service <PID>: Source - Identifies the source of the log record, from system monitoring service running on the localhost (the test PC) with its corresponding process PID.
- <logging-level>: logging level from the format of syslog-ng in the LCUs.
- [SYSTEM\_MONITORING]: identifier of system-monitoring service event
- LCU-statistics: LCU Statistics section. This can include:
  - CPU Consumption (%): Overall CPU utilization by the LCU in percentage.
  - Memory Consumption (MB): Overall memory usage of the LCU in Megabytes.
  - Voltage (V): Voltage level measured by the LCU in Volt.
  - LCU Temperature (°C): Temperature of the LCU in degrees Celsius.
  - CPU Temperature (°C): Temperature of the CPU on the LCU in degrees Celsius.
- running-process-list-with-corresponding-consumption (Optional): Running Process List - This section is only present when LCU-statistics is either CPU or memory consumption. Hardware status does not have this section in the log message. It provides details about processes running on the LCU at the time of logging, including:
  - Name: Process name.

- PID: Process ID.
- Command Line: Command used to launch the process.
- Memory Consumption (MB) or CPU Consumption (%): Resource utilization (memory or CPU) associated with each process.

The following log message was taken from the log file of one LCU in the Test PC, named LCU\_192.168.88.237.log. The event indicates that the LCU's CPU consumption exceeded the ERROR level threshold, along with the value of the CPU consumption and the processes used CPU resources during that time, thus this event will be sent to Teams Channel. Figure 20 shows the message on Teams channel.

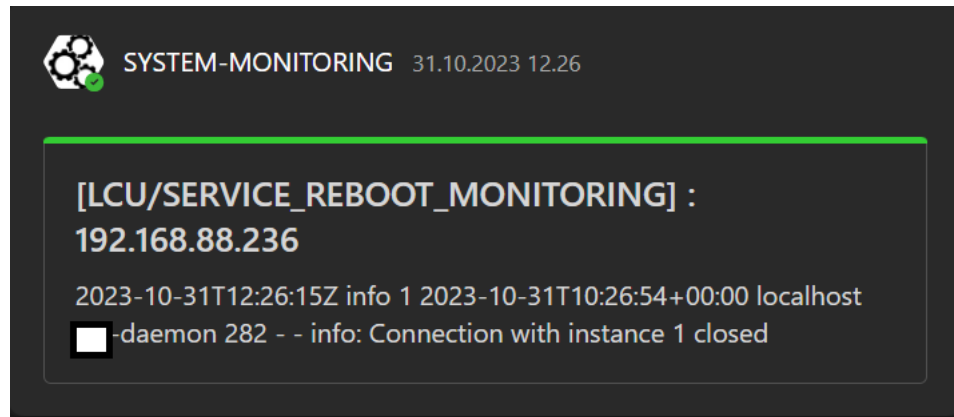
```
2024-01-02T14:37:05Z err 1 2023-12-31T18:14:35+00:00 localhost
system_monitoring_service 284 - - ERROR [SYSTEM_MONITORING] CPU load
average: 92.0 % Process CPU Consumption: {" process_1": [[1, cmd_1",
0.7]], " process_2": [[9, "", 0.6]], " process_3": [[10, "", 0.9]],
"process_4": [[15, "", 0.5]], " process_4": [[112, "", 0.1]], "protocol-
daemon": [[276, "/usr/bin/protocol-daemon -i eth1", 28.6]], "python3":
[[284, "python3 /usr/bin/system-monitoring.py", 0.5], [288, "python3
/usr/bin/lcuserver.py", 40.8], [326, "cmd_2", 0.1]]}
```



**Figure 20.** System Monitoring event notification in Teams.

The log message below indicates the connection between the LCU, and the train network simulator was down, shown in Figure 21:

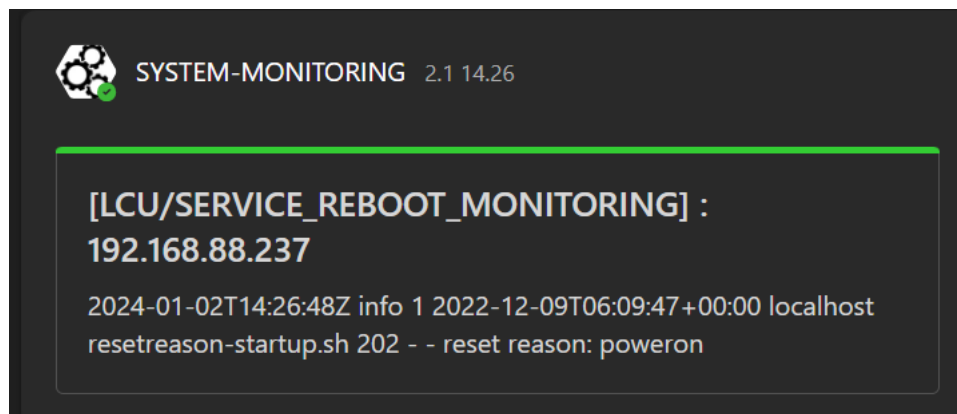
```
2023-10-31T12:26:15Z INFO 1 2023-10-31T10:26:54+00:00 localhost
protocol-daemon 282 - - INFO: Connection with instance 1 closed
```



**Figure 21.** Connection between LCU and train network closed event notification.

The following log message shows the LCU reboot, along with reason, shown in Figure 22:

```
2024-01-02T14:26:48Z INFO 1 2022-12-09T06:09:47+00:00 localhost
resetreason-startup.sh 202 - - reset reason: poweron
```



**Figure 22.** LCU reboot notification.

Overall, the proposed system successfully established connection between the LCUs, test computer and Teams channel and these components can exchange messages with each other. The performance of all LCUs were similar to each other, value and graph trending wise. The performance of all LCU will be evaluated to determine the reliability of the hardware and software of the LCU. The graph of the LCU with IP address 192.168.88.243 is used to analyze the trend of system.

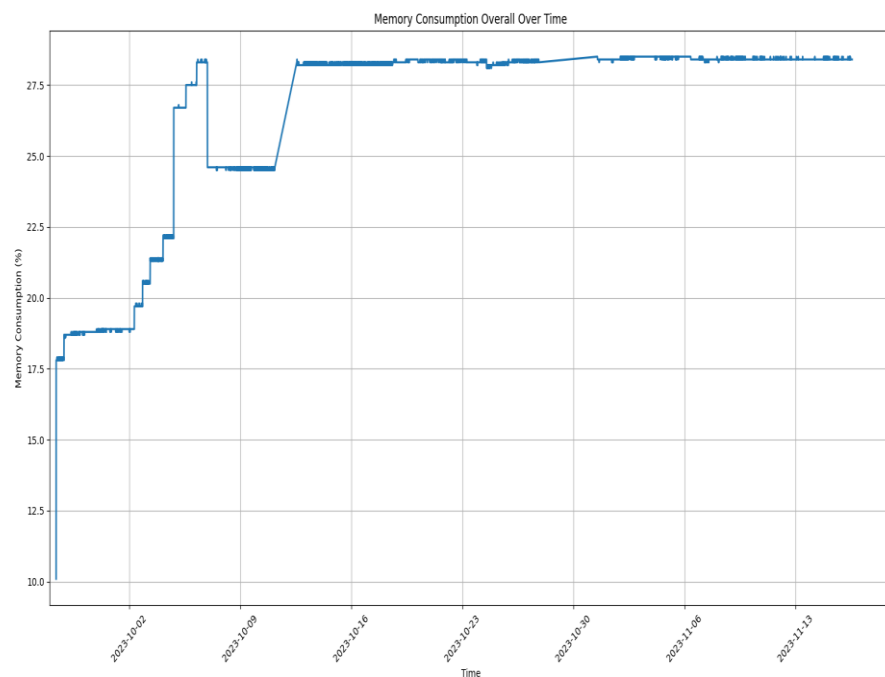
Table 2 shows the goals for each category. Most categories should perform under the mean and have lower than ceiling value, except for voltage is reversed.

Table 2. *Performance goals for LCUs*

<b>Metric</b>	<b>Mean(Higher or Lower)</b>	<b>Peak(Highest or Lowest)</b>
Memory usage (%)	40.0 (Lower)	50.0(Highest)
CPU temperature (°C)	60.0(Lower)	70.0(Highest)
LCU temperature (°C)	60.0(Lower)	70.0(Highest)
Supply voltage (V)	11.0(Higher)	11.0(Lowest)
CPU consumption (%)	60.0(Lower)	80.0(Highest)

## 4.1 Memory Consumption

The result of overall memory consumption of all processes is displayed in Figure 23 and Table 3.



**Figure 23.** Overall memory consumption of processes in LCU with IP address 192.168.88.243.

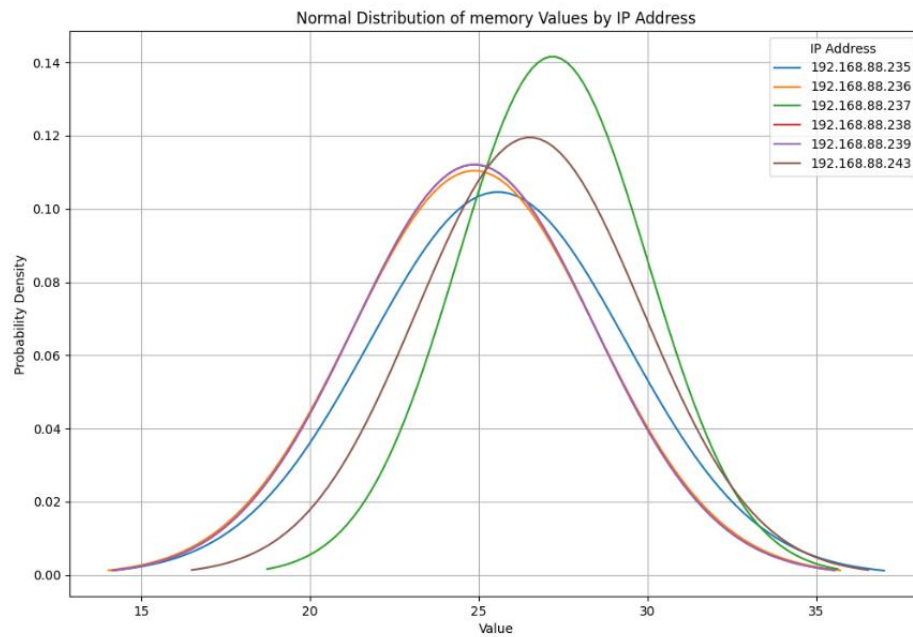
Table 3. *Overall memory Consumption in each LCUs.*

<b>LCU IP address</b>	<b>Min memory Consumption (%)</b>	<b>Max memory Consumption (%)</b>
192.168.88.243	10,1	28,5
192.168.88.239	16,9	30,6
192.168.88.238	17,9	28,6
192.168.88.237	9,7	28,4
192.168.88.236	10	28,5
192.168.88.235	9,9	31

Memory consumption monitoring 's main objective is detecting memory leaks in the LCU. Hongliang et al. explained that memory leaks happen when memory is dynamically allocated by malloc, calloc or new operators, is not deallocated after being used and cannot be reclaimed by the system afterward, leading to memory depletion [37]. Memory leak is a common issue in long time testing, since the occupied memory accumulates overtime if not detected. Memory leaks might negatively impact on the performance and reliability of the system, as the available memory exhausted. Feng Qin et al. discusses that leaks happen sometimes cause memory wasting or system slowdown due to increase paging, however, if the memory leak continues, the system might run out of memory, leading to system performance degradation or crashes [38]. Memory leaks gradually increase the memory consumption, which leads to low or exhausted available memory. Si Qin et al. claims that if this problem continue, it impacts other processes negatively, such as excessive paging, out of memory killer starts terminating processes to reclaim memory, or node reboot [39]. Si Qin et al. also argues that this problem is difficult to detect and only occurs in rare conditions and slowly, leading to bypassing detectors and testing [39].

According to Figure 24 and Table 3, the memory consumption of most LCUs ranges from 10% to slightly over 28,5% of 712,64 MB available memory, with an average of 24,86 to 27,17%. The graph shows a sharp rise in memory usage, with sudden spikes followed by stable periods, creating a staircase pattern. After rises in memory allocation, memory consumption reaches a consistent operation level without using more memory. The memory consumption in most LCUs mainly was under 30 % after the equilibrium in memory usage; only the LCU with an IP address of 192.168.88.239 was slightly higher than 30%.

A normal distribution graph was made to compare the similarities in performance between LCUs. The normal distribution of memory usage between the LCUs can be observed in Figure 24 and Table 4:



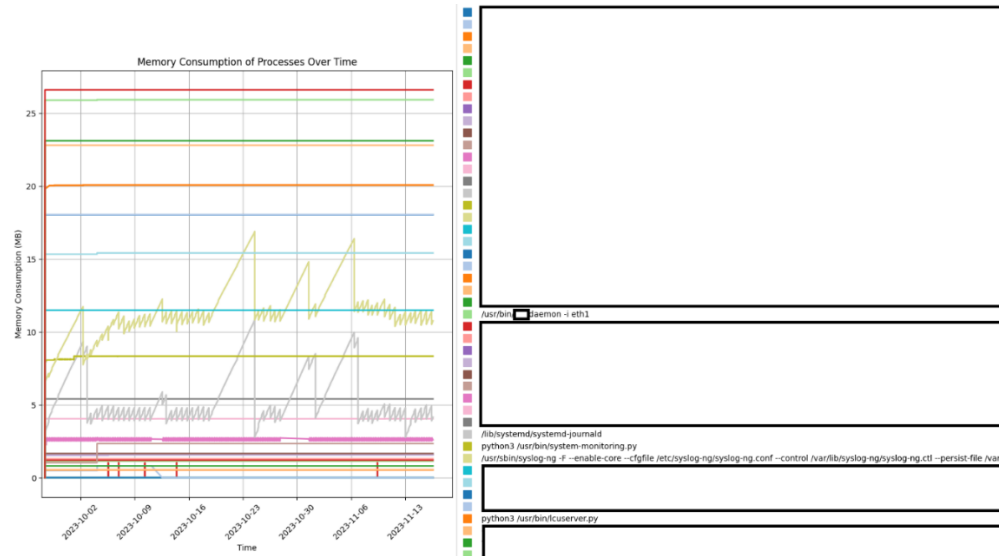
**Figure 24.** normal distribution of memory consumption of the LCUs

Table 4. Mean and standard deviation of memory consumption in each LCU

LCU IP address	Mean Memory Consumption (%)	Standard Deviation Memory Consumption
192.168.88.235	25,55	3,81
192.168.88.236	24,86	3,61
192.168.88.237	27,17	2,81
192.168.88.238	24,86	3,55
192.168.88.239	24,65	3,57
192.168.88.243	26,5	3,33

The normal distribution in Figure 24 and Table 4 shows that the LCUs' memory usage was similar, with very small differences in mean and standard deviation, meaning memory consumption in LCUs are stable without sudden spikes.

To further investigate the trend of the graph, memory usage of each process needed to be profiled. The Memory consumption of each process in the LCU can be seen in Figure 25 and Table 5:



**Figure 25.** Memory Consumption of all processes in LCU WITH IP address 192.168.88.243.

**Table 5.** Process memory consumption in each LCUs

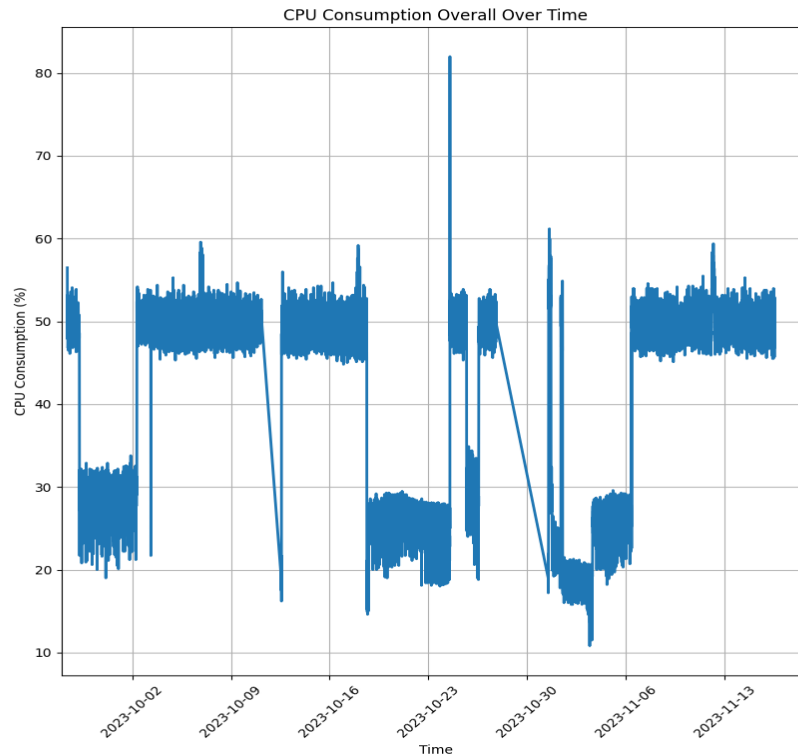
IP Address	Process	Min (MB)	Max(MB)	Mean(MB)
192.168.88.235	/lib/systemd/systemd-journald	2,80	12,51	6,36
	python3 /usr/bin/lcuserver	15,01	19,92	19,91
	//usr/sbin/syslog-ng	6,07	17,17	11,81
192.168.88.236	/lib/systemd/systemd-journald	3,03	12,58	6,79
	python3 /usr/bin/lcuserver	15,91	19,85	19,83
	/usr/sbin/syslog-ng	5,94	16,76	11,54
192.168.88.237	/lib/systemd/systemd-journald	2,71	11,66	5,38
	python3 /usr/bin/lcuserver	13,96	19,77	19,75
	/usr/sbin/syslog-ng	6,03	16,18	11,38
192.168.88.238	/lib/systemd/systemd-journald	2,49	11,71	6,316
	python3 /usr/bin/lcuserver	20,73	20,98	20,98
	/usr/sbin/syslog-ng	6,58	16,44	11,43
192.168.88.239	/lib/systemd/systemd-journald	2,49	11,71	6,316
	python3 /usr/bin/lcuserver	20,73	20,98	20,98
	//usr/sbin/syslog-ng	6,58	16,44	11,43
192.168.88.243	/lib/systemd/systemd-journald	2,76	10,84	5,437
	python3 /usr/bin/lcuserver	16,23	20,07	20,07
	/usr/sbin/syslog-ng	6,03	16,88	11,39

From Figure 25 and Table 5, most of the processes ran with stable linear shape and did not spike in memory usage. Most of the processes consume low to very low memory,

from below 1 to nearly 25 MB. However, two processes did not follow this trend, while creating a saw-tooth shape line and continuously fluctuating the memory consumption, `system-journald`, and `syslog-ng`. The saw-tooth shape graph reflects the continuation in allocating and deallocating memory. Some spikes in these processes' memory consumption are caused by high amounts of log messages at a time, thus more memory allocation. The trend in overall memory consumption and two logging processes caused by garbage collecting. RealPython discussed in an online study that Python manages memory using an automated garbage collection process, helps free up memory by detecting and disposing of objects that are no longer referenced or accessible by any part of the program [40]. The stability in memory use of the system came from the optimization of the garbage collector, balancing the allocation and deallocation, processes continue to generate logs and occupy memory of the system. From the description of memory leak, the graph pattern of memory leak is a constant increase in memory usage without a plateau section in a short time. In contrast, the graph in Figure 23 did not follow this pattern, instead memory usage in this graph increase in step and did not increase anymore, also there were periods when the graph had some long flat regions where memory usage stabilizes. The result showed low memory consumption, and as such, no memory leak was detected. The results satisfy the aim of having memory consumption on average under 40% and peak under 50%.

## 4.2 CPU Consumption

Figure 26 and Table 6 shows the overall CPU consumption throughout the LCU.



**Figure 26.** Overall CPU consumption in the LCU with IP address 192.168.88.243.

Table 6. CPU consumption of each LCUs.

LCU IP address	Min CPU Consumption (%)	Max CPU Consumption (%)
192.168.88.243	10,8	82,0
192.168.88.239	41,4	99,0
192.168.88.238	42,4	98,0
192.168.88.237	43,4	98,7
192.168.88.236	42,8	98,7
192.168.88.235	14,4	98,7

CPU consumption must be observed to prevent high CPU utilization. In this test, overall CPU consumption was monitored instead of each CPU. It is simpler to detect general trends and issues without the complexity of analyzing consumption from multiple cores individually.

From Table 6 and Figure 26, overall CPU consumption in the LCUs ranges from 10,8% to 98,7%, with an average of 40,98% to 54,03%. However, the graph's peak of 82% to 98,7% was a sudden spike, the peak of regular operation was between 60 to 65%. A sudden spike was detected when the SSH connection to the LCU, using PUTTY software, was made. Two sections described different trends of CPU usage in all LCUs:

one between 20% to 30% range and one between 40% to 60% for LCUs with IP addresses 192.168.88.235 and 192.168.88.243. For other LCUs, one section ranges from 45 to 50%, and the other ranges from 55% to 65%. From the log file, the first section was when the LCU was running in an ideal state, without connection to the train network, thus no data transmission. On the other hand, the second section had a connection between the LCU and the train network, with a high data transmission rate since the train network continued to send random data to the LCU and the LCU listened to messages from the train network, adjusted its parameters accordingly and sent its status back to the train network. The snip set below shows the CPU usage when the simulator started and crashed.

```
2023-10-02T04:44:08+00:00 localhost system_monitoring_service 286 - -
INFO [SYSTEM_MONITORING] CPU load average: 30.4 %

2023-10-02T04:49:08+00:00 localhost system_monitoring_service 286 - -
INFO [SYSTEM_MONITORING] CPU load average: 29.4 %

2023-10-02T04:49:28+00:00 localhost protocol-daemon 277 - - info:
Connection with instance 2 successfully opened

2023-10-02T04:53:39+00:00 localhost system_monitoring_service 286 - -
INFO [SYSTEM_MONITORING] CPU load average: 50.2 %

2023-10-02T04:58:40+00:00 localhost system_monitoring_service 286 - -
INFO [SYSTEM_MONITORING] CPU load average: 52.4 %

# After some times until the simulator crashed

2023-10-03T04:32:46+00:00 localhost system_monitoring_service 286 - -
INFO [SYSTEM_MONITORING] CPU load average: 48.3 %

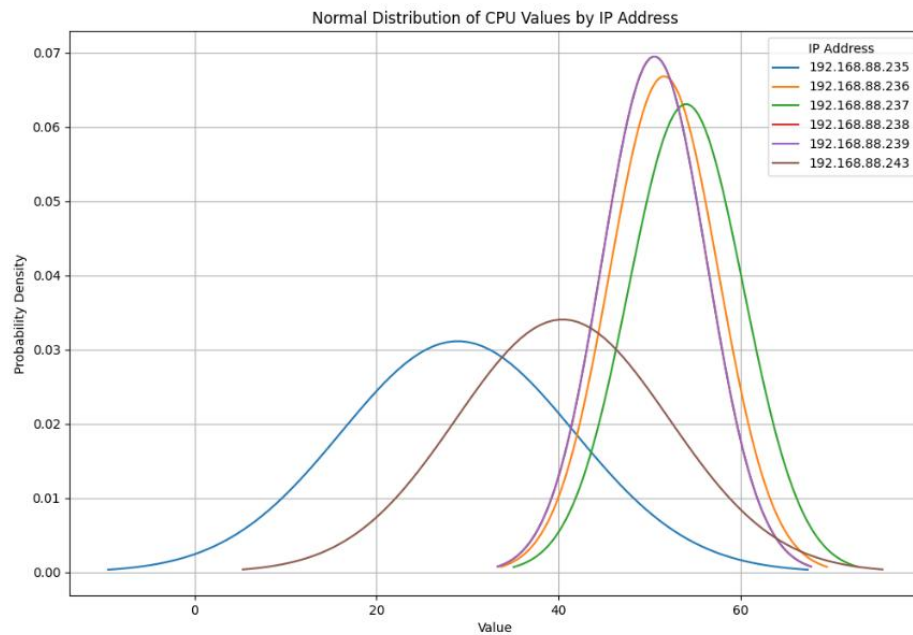
2023-10-03T04:37:46+00:00 localhost system_monitoring_service 286 - -
INFO [SYSTEM_MONITORING] CPU load average: 50.8 %

2023-10-03T04:39:46+00:00 localhost protocol-daemon-daemon 277 - - warn:
Connection with instance 1 timed out

2023-10-03T04:42:47+00:00 localhost system_monitoring_service 286 - -
INFO [SYSTEM_MONITORING] CPU load average: 22.3 %

2023-10-03T04:47:48+00:00 localhost system_monitoring_service 286 - -
INFO [SYSTEM_MONITORING] CPU load average: 21.7 %
```

A normal distribution graph was created to identify the different behaviors between LCUs. The following figure shows the normal distribution of each LCU CPU consumption:



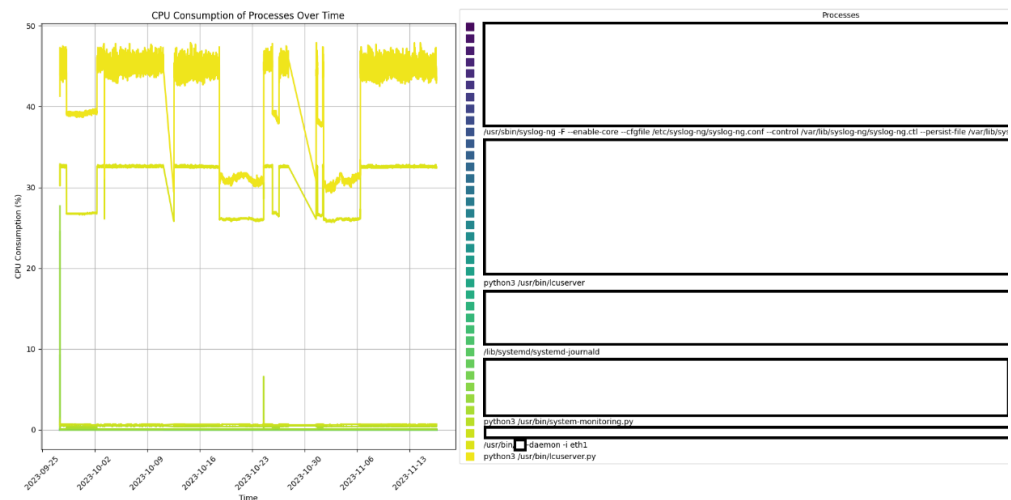
**Figure 27.** normal distribution of CPU usage in the LCUs

Table 7. mean and standard deviation of CPU consumption in each LCUs.

LCU IP address	Mean CPU Consumption (%)	Standard Deviation CPU Consumption
192.168.88.235	28,9	12,81
192.168.88.236	51,61	5,9
192.168.88.237	54,03	6,32
192.168.88.238	50,55	5,73
192.168.88.239	50,55	5,73
192.168.88.243	40,46	11,71

From Figure 27 and Table 7, LCUs with IP addresses 192.168.88.235 and 192.168.88.243 had similar trends of lower mean CPU consumption while having a wide range of values, while the other LCU had higher mean CPU consumption with more stable CPU usage. CPU of 192.168.88.235 and 192.168.88.236 varies significantly. The results satisfied the requirement of mean CPU consumption around 60% and peak CPU consumption under 80%. However, the 4 LCUs with mean over 50% should be optimized to be less or equal to 50%.

To investigate the CPU consumption graph further, CPU usage of each process needed to be profiled. The following figure demonstrates the CPU consumption of each process in the LCU:



**Figure 28.** CPU consumption of all processes in the LCU with IP address 192.168.88.243

**Table 8.** Process CPU consumption in each LCUs.

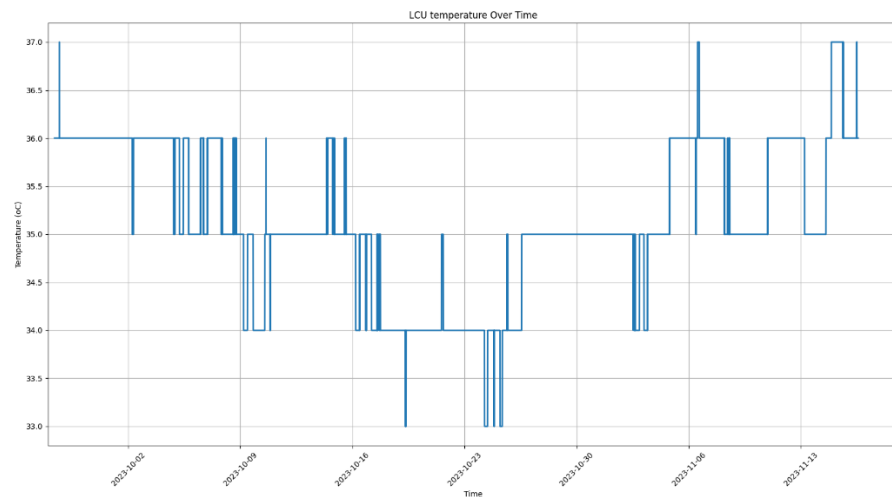
IP Address	Process	Min Value	Max Value	Mean Value
192.168.88.235	/usr/bin/lcuserver	28,4	47,3	33,93
	/usr/bin/protocol-daemon	25,3	32,4	27,65
192.168.88.236	/usr/bin/lcuserver	27,5	47,7	34,12
	/usr/bin/protocol-daemon	27,5	35,3	30,17
192.168.88.237	/usr/bin/lcuserver	27,7	47,7	36,07
	/usr/bin/protocol-daemon	27,4	35,0	31,43
192.168.88.238	/usr/bin/lcuserver	27,5	49,8	35,50
	/usr/bin/protocol-daemon	26,7	33,5	30,1
192.168.88.239	/usr/bin/lcuserver	27,5	49,8	35,50
	/usr/bin/protocol-daemon	26,7	32,7	29,7
192.168.88.243	/usr/bin/lcuserver	29,3	47,9	40,75
	/usr/bin/protocol-daemon	25,7	32,9	30,27

From Figure 28 and Table 8, the CPU consumption of `lcuserver` service and `protocol-daemon` had the highest CPU consumption. The graph shapes of these processes were similar to the overall CPU consumption graph shape, with two different consumption threshold sections as explained. `protocol-daemon` is a service to establish connection between the train computer and the LCU. As the transmission traffic increased, the CPU consumption of this service eventually accumulated. `lcuserver` is a core component of the LCU, which controls the PWM channels outputs, determining the train's LED light behavior. In addition, the LCU defined predefined scenario, which

sets the PWM value based on the situation (day, night...) or zone where the LCU was installed (vestibule, spotlight, ...). By randomly and continuously sending random configuration setting to the LCU, train simulator forced the `lcuserver` service to process numerous of tasks, such as adjusting PWM value of channels, error handling, and significantly increases the transmitting rate, thus soaring the LCU's CPU consumption. As such, `lcuserver` was the largest CPU consuming process.

### 4.3 CPU and LCU temperature

Figure 29 and Table 9 shows the LCU temperature within LCU with IP address 192.168.1.243

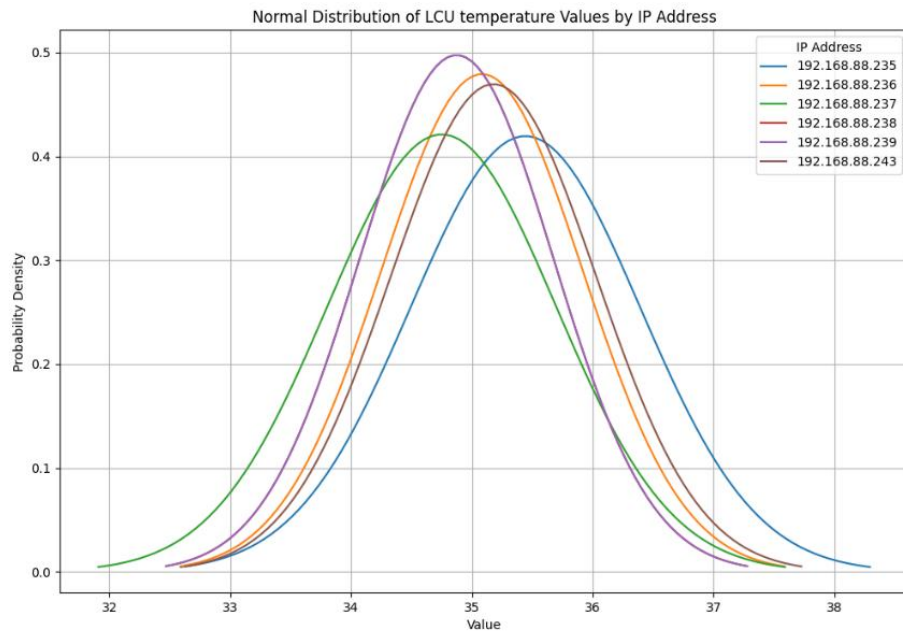


**Figure 29.** LCU temperature in LCU with IP address 192.168.88.243

Table 9. LCU temperature of each LCUs

LCU IP address	Min LCU temperature (oC)	Max LCU temperature (oC)
192.168.88.243	33	37
192.168.88.239	34	37
192.168.88.238	33	36
192.168.88.237	33	36
192.168.88.236	34	37
192.168.88.235	34	37

Figure 30 and Table 10 shows the normal distribution of LCU temperature.

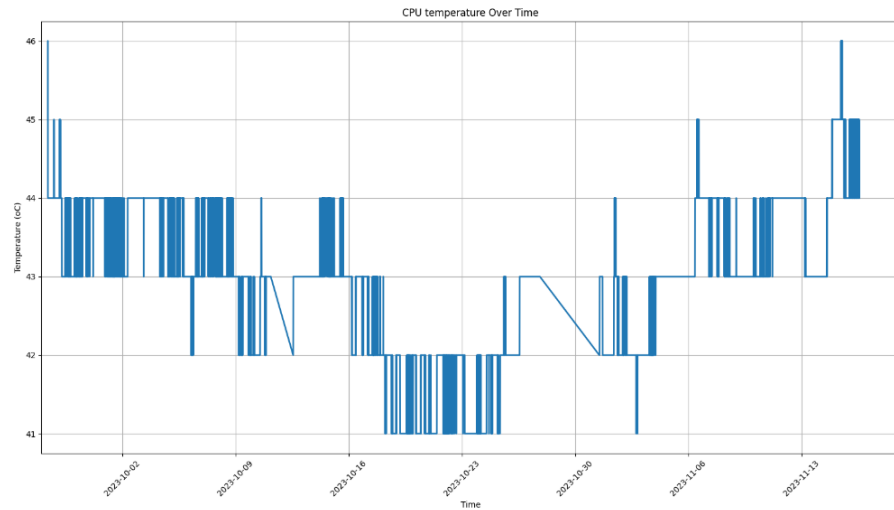


**Figure 30.** normal distribution graph of LCU temperature.

Table 10. mean and standard deviation of LCU temperature of each LCUs

LCU IP address	Mean LCU temperature (oC)	Standard Deviation LCU temperature (oC)
192.168.88.243	35,44	0,95
192.168.88.239	35,08	0,83
192.168.88.238	34,75	0,94
192.168.88.237	34,87	0,8
192.168.88.236	34,88	0,8
192.168.88.235	35,17	0,84

Figure 31 and Table 11 shows the CPU temperature of LCU with IP address 192.168.88.243

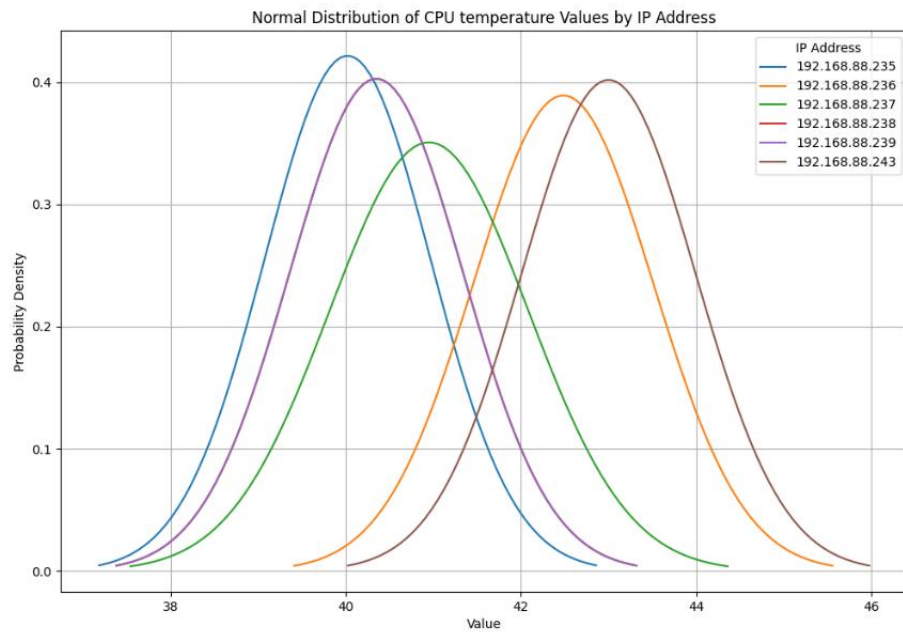


**Figure 31.** CPU temperature of LCU with IP address 192.168.88.243.

Table 11. CPU temperature of each LCUs.

LCU IP address	Min CPU temperature (°C)	Max CPU temperature (°C)
192.168.88.243	41	46
192.168.88.239	39	45
192.168.88.238	40	46
192.168.88.237	39	43
192.168.88.236	40	45
192.168.88.235	38	42

Figure 32 and Table 12 shows the normal distribution of CPU temperatures of all LCUs.



**Figure 32.** normal distribution graph of CPU temperature.

Table 12. mean and standard deviation of each LCUs.

LCU IP address	Mean CPU temperature (oC)	Standard Deviation CPU temperature
192.168.88.243	40.01	0.94
192.168.88.239	42.48	1.02
192.168.88.238	40.94	1.13
192.168.88.237	40.34	0.99
192.168.88.236	40.33	0.99
192.168.88.235	43	0.99

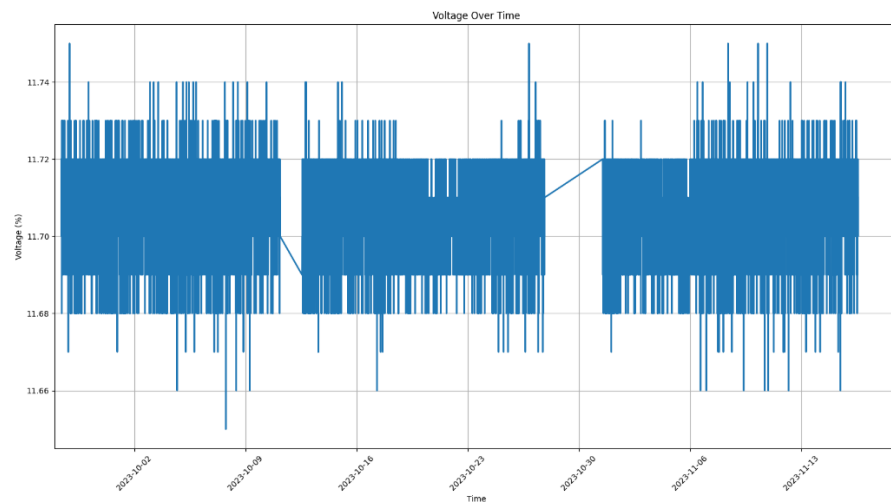
The test's purpose is to prevent mainly CPU overheating. As the main processing and calculating unit, the CPU requires the most power from the system, thus it is the hottest unit in the system. As the CPU's temperature increases, its surrounding's temperature will also rise. Moreover, the ambient temperature also might affect the CPU's temperature. The range of temperature from both LCU and CPU was not high, from 38 °C to 46 °C with an average of 40 °C to 42 °C for the CPU and 34,0 °C to 37,0 °C with an average of 35,31 °C for the LCU. The temperature fluctuated by a small margin, with changing in steps instead of gradual drift. This result indicates that the temperature of the system remained relatively low and stable for long continuous operation time. High temperature heavily affects the system's performance. Bach Matt argued that during high temperature, CPU maintains reduced frequencies leading to decreased processing power, which leads to performance reduction, or failure [41]. Narayanan and Xie

discussed that power leakage increases exponentially with temperature, also increase interconnect delay within electronic components [42].

From the normal distribution graphs, the LCU and CPU temperature of the LCUs were similar, with slight differences in mean and standard deviation. Overall, the LCU and CPU temperature satisfied the goals in Table 2.

## 4.4 Voltage

Figure 33 and Table 13 shows the inner voltage within the LCU with IP address 192.168.88.243.



**Figure 33.** Inner Voltage of the LCU with IP address 192.168.88.243.

Table 13. Inner voltage of LCUs.

LCU IP address	Min Voltage (V)	Max Voltage (V)	Average Voltage (V)
192.168.88.243	11,68	11,75	11,7
192.168.88.239	11,75	11,77	11,76
192.168.88.238	11,75	11,77	11,76
192.168.88.237	11,71	11,75	11,72
192.168.88.236	11,75	11,79	11,76
192.168.88.235	11,72	11,75	11,73

The test was conducted to detect voltage drops from the system. There are several reasons that might cause voltage drop. Firstly, insufficient power supply or variation in input voltage cannot deliver enough power for the system to operate. According to Fluke, many factors cause increase in resistance, for instance, high resistance wire, poor soldering connectors, significantly reducing the voltage over the system [43].

If the system voltage dropped below a threshold, the system might shut down due to not having enough power supply. Narayanan and Xie discussed that when voltage dropped, the error increased in SRAM memory [42]. Vahid Garousi et al. discussed that voltage drop can cause system resets, data loss, or unexpected behaviors, since the read and write operation was interrupted [9]. Force Technology discussed that prolonged undervoltage conditions can degrade components, potentially leading to reduce in its lifetime [44]. The result demonstrated that no voltage drop was found, the voltage had tiny fluctuations, with an average of 11.7 V. The voltage did not drop below the threshold of 10 V. The results from the LCU's voltage satisfied the voltage drop requirement in Table 2.

#### **4.5 LCU reboot, systemd service reboot and train network disconnect.**

No LCU reboot or service reboot was detected during the test. In the LCU software, numerous `systemd` services depend on the `lcuserver` service and are required for the `lcuserver` to operate correctly and vice versa, for instance, service that detects overheating, PWM ports failures or internal software failures. If any `lcuserver` dependencies or failures detected, `lcuserver` service fails and attempt to restart itself. In the LCU 's system, `watchdog` continuously monitor the status of `lcuserver` and other system's services. If any services fail, the service is given some time to restart. If the number of restarting times exceed limit, the LCU reboots. This event will be sent to Teams Channel since there was a problem with the system's software. In addition, `resetreason` were installed as a recipe in the LCU's Yocto image to explain the cause of the reboot. If the `watchdog` forced the LCU to reboot, the reason is `watchdog`, while if the reason was power supply related or manual reboot, the reason would be `power on`. During the test, no such log messages were detected.

Train network disconnection sometimes happens due to simulator crashes; thus, train network disconnection log messages were received. From the log, some missed messages from the LCU detected for some duration, hence counted as lost connection. Several "No data received" found. The LCU stopped sending the messages because of the timeout. If the timeout was set too low, slight delays or network issues can lead to the train simulator detecting a timeout. By increasing the timeout in the simulator, the connection allows more missing messages, avoiding the timeout restriction, letting the connection continue. After this change, the train simulator disconnection event was no longer detected and sent to Team Channel.

## 5. CONCLUSIONS

In this work, a testing framework had been successfully set up, which reports the status of the LCUs in real time and alerts the notification system in case of failures or performance issues. The Long-Term Testing environment for the LCU developed in this work has contributed to a crucial part of the company's testing process, especially in the final phase, after all functional testing is finished and the final release is approaching. Based on the testing result of this work, the testing team could decide the quality of the LCU by detecting and reacting to potential failures in advance. System-monitoring service has been installed to common image recipes and can be deployed in future projects. Recent projects have already applied the understanding about setting up Syslog server to enhance the remote logging capabilities of the LCU. The system was successfully established, the LCU and the simulated train network communicated with each other's, Teams were able to notify team members about faults during LCU's operation time, even from remote locations.

Although the LCU monitoring and alert system successfully handles real-time performance monitoring and immediate alerts, the system can be further improved. Firstly, expanding communication methods to cover different platforms, such as SMS or email, would widen user preferences and effectively transmit critical notifications. The list of processes in memory and CPU usage can be improved so it is more readable. As the LCU continues to get increasing features and devices connected to it, as such, additional events need to be monitored and added to the regex list. While having a web server for the system's information and maintenance, the web server can be upgraded to display CPU and Memory usage of the LCU and its runtime processes in real-time, including real-time graph display. A significant drawback in the system's design is when observing the LCU's status during power shortage. When there is a power outage, the LCU stops sending log messages to the Test PC, causing a gap in the monitoring system and determining the timeframe when the LCU powered off more difficult. Moreover, Teams is currently the only notification platform being used. Because Test PC needs a constant power supply to stay connected, Teams cannot issue notifications regarding the LCU's condition when power is lost. Therefore, users are not notified about the LCU's shutdown until the power is back on, and the test environment sends messages again, which might cause a sudden spike in the data graph, and missing time gaps. Moreover, this creates gaps in data exchange, which might create data gaps or huge spikes in the graphs. Moreover, obtaining logs from the LCUs or Test PC from a remote distance should be

considered. When faults happen, the teams.py script can be modified to send the current log file of the faulty LCU to Teams Channel, or a log downloading feature from a remote distance should be implemented in the future, either through Teams messages, or the web server's application. Furthermore, additional Teams Channel helps separate channels for each event. Finally, the script in test PC can be modified to monitor the method call times, CPU and memory usage in htop style, which displays the LCU's statistics in table style and makes data more readable.

## REFERENCES

- [1] “Traffic Trends Among UIC Member Companies in 2023 – UIC Communications.” UIC Communications, 19 June 2024, [uic.org/com/enews/article/traffic-trends-among-uic-member-companies-in-2023](http://uic.org/com/enews/article/traffic-trends-among-uic-member-companies-in-2023).
- [2] Ishii, Isao, et al. “LED Lighting System for Rolling Stock.” *Hitachi Review*, vol. 61, no. 7, 2012, pp. 301–02. [www.hitachihyoron.com/rev/pdf/2012/r2012\\_07\\_108.pdf](http://www.hitachihyoron.com/rev/pdf/2012/r2012_07_108.pdf).
- [3] Arm Ltd. “What Is Embedded System Design (ESD)?” *Arm | the Architecture for the Digital World*, [www.arm.com/glossary/embedded-system-design](http://www.arm.com/glossary/embedded-system-design).
- [4] Saini, D. K. (2012). Software Testing for Embedded Systems. *International Journal of Computer Applications*. <https://doi.org/10.5120/6192-8700>.
- [5] “GCSE COMPUTER SCIENCE CIE | TOPIC 3 EMBEDDED SYSTEMS.” *COMPUTER SCIENCE CAFÉ*, [www.computersciencecafe.com/315-computer-architecture-cie.html](http://www.computersciencecafe.com/315-computer-architecture-cie.html).
- [6] “Embedded Software | Siemens Software.” Siemens Digital Industries Software, [www.plm.automation.siemens.com/global/en/our-story/glossary/embedded-software/64121](http://www.plm.automation.siemens.com/global/en/our-story/glossary/embedded-software/64121).
- [7] “Testing Embedded Software: Ensuring Quality and Reliability - INTechHouse Blog.” InTechHouse, 1 Oct. 2024, [intechhouse.com/blog/testing-embedded-software-ensuring-quality-and-reliability](http://intechhouse.com/blog/testing-embedded-software-ensuring-quality-and-reliability).
- [8] M. Bajer, et al. "Embedded software testing in research environment. A practical guide for non-experts," 2015 4th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 2015, pp. 100-105, doi: 10.1109/MECO.2015.7181877.
- [9] Garousi, Vahid, et al. “Testing Embedded Software: A Survey of the Literature.” *information and Software Technology*, vol. 104, July 2018, pp. 14–45. <https://doi.org/10.1016/j.infsof.2018.06.016>.
- [10] “Reliability Testing Software Testing.” *GeeksforGeeks*, 27 Sept. 2025, [www.geeksforgeeks.org/software-testing/software-testing-reliability-testing/#what-is-reliability-testing](http://www.geeksforgeeks.org/software-testing/software-testing-reliability-testing/#what-is-reliability-testing).
- [11] R. Aalund and V. Philip Paglioni, "Enhancing Reliability in Embedded Systems Hardware: A Literature Survey," in *IEEE Access*, vol. 13, pp. 17285-17302, 2025, doi: 10.1109/ACCESS.2025.3534138
- [12] qodo. “What Is Soak Testing? Components, Advantages and Best Practices.” *Qodo*, 14 Feb. 2025, [www.qodo.ai/glossary/soak-testing/#:~:text=What%20is%20Soak%20Testing?%20Soak%20testing%20involves,users%20keep%20it%20active%20for%20long%20durations](http://www.qodo.ai/glossary/soak-testing/#:~:text=What%20is%20Soak%20Testing?%20Soak%20testing%20involves,users%20keep%20it%20active%20for%20long%20durations).
- [13] Testlio. “What Is Soak Testing? How to Design a Soak Test.” *Testlio*, 15 Nov. 2024, [testlio.com/blog/soak-testing-in-software-testing](http://testlio.com/blog/soak-testing-in-software-testing)

- [14] Texas Instruments, "SPRABX4B," Calculating Useful Lifetimes of Embedded Processors, [Online] <https://www.ti.com/lit/an/sprabx4b/sprabx4b.pdf>
- [15] Hernandez, Maria and Department of Electrical and Computer Engineering, Technical University of Munich, Germany. "Reliability Testing of Embedded Systems in Harsh Environments." *American Journal of Embedded Systems and VLSI Design*, journal-article, 2022, [australiansciencejournals.com/ajesvd](http://australiansciencejournals.com/ajesvd).
- [16] Banerjee, Abhijeet, et al. "On Testing Embedded Software." *Advances in computers*, 2016, pp. 121–53. <https://doi.org/10.1016/bs.adcom.2015.11.005>.
- [17] V. Chandra, "Monitoring reliability in embedded processors - A multi-layer view," 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2014, pp. 1-6, doi: 10.1145/2593069.2596682.
- [18] Noack, Tino. Real-Time Monitoring and Long-Term Analysis by Means of Embedded Systems. 2011. By TU Cottbus et al., journal-article, [ceur-ws.org/Vol-731/15.pdf](http://ceur-ws.org/Vol-731/15.pdf).
- [19] Yumei Wu, et al. "Embedded software reliability testing and its practice," 2010 International Conference On Computer Design and Applications, Qinhuangdao, China, 2010, pp. V2-24-V2-27, doi: 10.1109/ICCD.2010.5541047
- [20] "Lighting Control - Teknoware." Teknoware, 28 Nov. 2024, [www.teknoware.com/rolling-stock-lighting-and-interiors/interior-lighting-for-trains/lighting-control-solutions-for-rolling-stock](http://www.teknoware.com/rolling-stock-lighting-and-interiors/interior-lighting-for-trains/lighting-control-solutions-for-rolling-stock).
- [21] The Yocto Project. "Technical Overview - the Yocto Project." *The Yocto Project*, 6 Mar. 2025, [www.yoctoproject.org/development/technical-overview](http://www.yoctoproject.org/development/technical-overview).
- [22] Georg, J., et al. "BOARD BRING-UP WITH FPGA FRAMEWORK AND ChimeraTK ON Yocto." ICALEPCS2023, 2023, <https://accelconf.web.cern.ch/icalepcs2023/papers/tupdp025.pdf>.
- [23] Rakshitha, A.H., et al. "Software Configuration Using Jenkins and Yocto Project." 2023 7th International Conference on Inventive Computation Technologies (ICICT), IEEE, 2023, <https://ieeexplore.ieee.org/abstract/document/10334213/>.
- [24] Yocto Project. (n.d.). Mega manual <http://www.yoctoproject.org/docs/3.1/mega-manual/mega-manual.html>
- [25] "systemd.unit — Unit Configuration." [Freedesktop.org](http://freedesktop.org), 2024, <https://www.freedesktop.org/software/systemd/man/latest/systemd.unit.html>. Accessed 21 May 2024.
- [26] Ellingwood, Justin. "Understanding Systemd Units and Unit Files." *DigitalOcean*, 24 Jan. 2022, [www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files](http://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files).
- [27] Ellingwood, Justin. "Systemd Essentials: Working With Services, Units, and the Journal." *DigitalOcean*, 21 Apr. 2015, [www.digitalocean.com/community/tutorials/systemd-essentials-working-with-services-units-and-the-journal](http://www.digitalocean.com/community/tutorials/systemd-essentials-working-with-services-units-and-the-journal).

- [28] CrowdStrike. (2023). Working with syslog-ng <https://www.crowdstrike.com/en-us/guides/syslog-logging/working-with-syslog-ng/>
- [29] syslog-ng. (n.d.). syslog-ng [Source code repository]. GitHub. <https://github.com/syslog-ng/syslog-ng>
- [30] H. Tsunoda and G. M. Keeni, "Managing syslog," The 16th Asia-Pacific Network Operations and Management Symposium, Hsinchu, Taiwan, 2014, pp. 1-4, doi: 10.1109/APNOMS.2014.6996575.
- [31] Huang, J., et al. (2012). The design and implement of the centralized log gathering and analysis system. s2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE), 2, 268-273. <https://doi.org/10.1109/CSAE.2012.4272772>.
- [32] Irwin, Barry V. W., and Franco Loyola. "Towards Scalable Secure Syslog Compatible Remote Logging." International Conference on Intelligent and Innovative Computing Applications, vol. 2022, Dec. 2022, pp. 13–21. <https://doi.org/10.59200/iconic.2022.002>.
- [33] Mochel, Patrick. "The sysfs filesystem." Linux Symposium. Vol. 1. San Francisco, CA, USA: The Linux Foundation, 2005. [www.kernel.org/doc/ols/2005/ols2005v1-pages-321-334.pdf](http://www.kernel.org/doc/ols/2005/ols2005v1-pages-321-334.pdf).
- [34] Bei Wang, et al. "The comparison of communication methods between user and Kernel space in embedded Linux," International Conference on Computational Problem-Solving, Li Jiang, China, 2010, pp. 234-237. [ieeexplore.ieee.org/document/5696027](http://ieeexplore.ieee.org/document/5696027)
- [35] Psutil Documentation — Psutil 7.1.0 Documentation. [psutil.readthedocs.io/en/latest/index.html#](http://psutil.readthedocs.io/en/latest/index.html#).
- [36] Sumrak, Jesse. "What Is a Webhook (and How Does It Work)?" *Twilio*, 19 Aug. 2025, [www.twilio.com/en-us/blog/insights/whats-webhook](http://www.twilio.com/en-us/blog/insights/whats-webhook).
- [37] Liang, Hongliang, et al. "LeakGuard: Detecting Memory Leaks Accurately and Scalably." *arXiv.org*, 6 Apr. 2025, [arxiv.org/abs/2504.04422](http://arxiv.org/abs/2504.04422).
- [38] Qin, Feng, Jr., et al. "SafeMem: Exploiting ECC-Memory for Detecting Memory Leaks and Memory Corruption During Production Runs." *Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture (HPCA-11 2005)*, conference-proceeding, 2005, [pages.cs.wisc.edu/~shanlu/paper/28\\_qin-safemem.pdf](http://pages.cs.wisc.edu/~shanlu/paper/28_qin-safemem.pdf).
- [39] Si Qin, et al. "RESIN: A Holistic Service for Dealing with Memory Leaks in Production Cloud Infrastructure." *Afcafawda*, 2021, [www.cs.jhu.edu/~chlou/paper/resin-osdi22-preprint.pdf](http://www.cs.jhu.edu/~chlou/paper/resin-osdi22-preprint.pdf).
- [40] Python, Real. Garbage Collection | Python Glossary – Real Python. [realpython.com/ref/glossary/garbage-collection](http://realpython.com/ref/glossary/garbage-collection).
- [41] Bach, Matt. "Impact of Temperature on Intel CPU Performance." Puget Systems, 28 Oct. 2014, [www.pugetsystems.com/labs/articles/Impact-of-Temperature-on-Intel-CPU-Performance-606](http://www.pugetsystems.com/labs/articles/Impact-of-Temperature-on-Intel-CPU-Performance-606).

- [42] V. Narayanan and Y. Xie, "Reliability concerns in embedded system designs," in *Computer*, vol. 39, no. 1, pp. 118-120, Jan. 2006, doi: 10.1109/MC.2006.31
- [43] Fluke. "Diagnosing Voltage Drops: Electrical Automotive Troubleshooting." *Fluke Corporation*, 6 Sept. 2025, [www.fluke.com/en-us/learn/blog/automotive/electrical-automotive-troubleshooting?srsIid=AfmBOorhSrpUi71MqvbeDxHeNSRttyW8t\\_iHZ3JW2\\_a uVbzGSiO1-jvB](http://www.fluke.com/en-us/learn/blog/automotive/electrical-automotive-troubleshooting?srsIid=AfmBOorhSrpUi71MqvbeDxHeNSRttyW8t_iHZ3JW2_a uVbzGSiO1-jvB).
- [44] *Lifetime and Electrical Stressors*. [forcetechnology.com/en/articles/product-lifetime-electrical-stressors](http://forcetechnology.com/en/articles/product-lifetime-electrical-stressors)