Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Master's Degree in Computer Science with Specialization in Software Engineering - 60 credits

EMULATION OF NETWORK DEVICE BEHAVIOUR FOR ROBOT CONTROLLER TESTING

Muhamed Opacin
mon22012@student.mdu.se

Examiner: Wasif Afzal
Mälardalen University, Västerås, Sweden

Supervisors: Eduard Paul Enoiu
Mälardalen University, Västerås, Sweden

Company supervisor: Pär Muhr,
Linus Lyckehult,
Gustav Borgersen
ABB Robotics & Discrete Automatics, Västerås, Sweden

June 14, 2023

Abstract

The testing of software for robot controllers has become increasingly difficult as robotic systems become more complex. As the complexity of the systems increases, the number of hardware systems that the robot relies on also grows. This poses a challenge in testing robot controllers, which is crucial to ensure that robots function safely and effectively in their intended applications. While simulation can be used as a platform for software testing, it is not feasible to simulate everything in a virtual environment, especially when test cases require physical connections to hardware for input and output signals sent to robot controllers. Therefore, the objective of this thesis is to replicate I/O device network communication in order to enhance virtual testing processes. The approach employed involves capturing real-time network traffic, modifying and rebuilding it, and subsequently replaying it. The work examines existing academic research on these approaches and technologies, and investigates the specific challenges in the testing process by conducting research within a company leading globally in industrial robot development. A conceptual model is proposed, and a prototype is developed. The solution demonstrates potential in addressing the current challenges in robot controller testing by enabling network capture, modification, and level 4 network traffic replay. However, experimental results reveal various limitations, such as significant delays in generating responses. Therefore, further research and development are required if the solution is to be implemented in a real-world setting.

List of Figures

List of Tables

Table of Contents

1. Introduction

Robotics is a rapidly evolving field that involves the design, development, and use of robots to perform a variety of tasks. Robots are machines that can carry out complex operations autonomously or with human assistance. They can be used in a wide range of applications, from manufacturing and logistics to healthcare and space exploration [6]. With advancements in artificial intelligence, machine learning, and sensor technology, robots are becoming more intelligent and capable, their potential uses are expanding and they are increasingly being integrated into various industries, revolutionizing the way work is done [7].

Robotic software development is a specialized field within the broader realm of software engineering that focuses on creating software systems to control and operate robots. It is a crucial component of robotics, as it enables robots to perform tasks and interact with their environment and typically consists of implementing various modules, including perception, planning, and control [8]. Developing software for robots presents unique challenges, such as dealing with noisy and incomplete sensor data, ensuring real-time performance, and addressing safety concerns. However, advancements in software engineering, particularly in the area of software testing and verification, are helping to address these challenges.

Overall, the development of robust and reliable robotic software is essential for creating effective and useful robots that can operate safely and efficiently in a variety of environments. Robotic systems must be thoroughly tested before they are put into service because of the risks and expenses involved with failures. Their qualities present particular testing challenges, and usually require significant field testing in the real world [9].

In industrial settings, robots are typically used alongside various types of equipment, such as machine tools, conveyors, sensors, and fixtures, to perform specific tasks. For the entire system to function correctly, the actions of the robot must be synchronized with those of the other equipment, and each component must perform its designated function in the appropriate sequence [6].

Equipment that is commonly used in modern robotic systems are I/O (input/output) modules (devices). In robot controllers, I/O devices are used to interface with the robot and other components of the cell for purposes like receiving input from a sensor indicating that a part is in place, and then use this information to initiate a specific robot action. Similarly, the controller may send output signals to actuators to control the movement of the robot or other equipment in the cell [10]. By using I/O devices to interface with the robot and other components, the controller can coordinate and synchronize all aspects of the cell's operation, enabling efficient and reliable performance.

As robots become more complex due to technological advances in areas such as artificial intelligence, machine learning, and computer vision [7], they generally require more I/O devices which in turn require a lot of physical hardware. This can present a barrier in the development, but mostly in testing of robot controllers which is vital to ensure that robots behave safely and effectively in their intended applications. The use of simulation as a platform for more sophisticated software testing of robots is now possible because of the simulation tools that are already accessible, which would be advantageous to all parties associated with the robot [11].

However, emulation of I/O devices for software testing is also an area that is being studied and explored in both academic and industrial research. Emulating I/O devices in robot cells can allow software developers and testers to simulate various scenarios and test the robot controller's responses to different inputs, without the need for physical hardware. This can help identify potential issues or errors in the controller's behavior, and refine its performance before it is deployed in a real-world setting.

This thesis work aims to find an approach to emulate I/O devices in virtual environments for the general purpose of robot controller testing. It is done in collaboration with ABB Robotics, a

company that specializes in the development, testing and manufacturing of industrial robots. The company already possesses a software that serves as a partial emulator, however physical hardware and ethernet connection cables are still required. Therefore, the aim of this thesis work is to propose a model for an emulator designed to mimic the communication between the robot controller and I/O devices and eliminate the need for any hardware during the testing process. This requires good understanding of the network protocols that lie beneath the communication, thorough network traffic analysis using capture tools and software, as well as research into approaches for seamless mimicking of network packets through a virtual interface.

The thesis work aims to answer the following research questions:

RQ1: What existing approaches, if any, can be used to emulate network communication and behaviour of I/O devices inside robot cells in a virtual environment?

RQ2: Does the approach to emulating I/O device behaviour impact performance of test cases that are run on robot cells?

The rest of the work is split into sections as follows. Section 2. will give background information that is needed to understand the area and the task. A brief intro into the robotic system and robot cell, emulation, network protocols and traffic replay is presented. Section 3. describes the scientific and research methods that have been applied to this project. Section 4. describes the current research in the field and its limitations. Section 5. will look into the current process and approaches in the company, as well as provide motives for such a project. Section 6. outlines the proposed model to the emulation approach. Section 7. will present evaluation and results of the proposed model. Finally, the report will conclude in Section 8. and 9. where the results of the thesis work will be discussed and future work presented.

2. Background

2.1. Industrial Robots

An industrial robot is one that has been created to automate labor-intensive manufacturing operations, as those needed by an assembly line that is continually in motion. They are positioned in permanent locations across an industrial facility as integral parts, and all other worker duties and operations revolve around them.

The robot is made up of several connections that are connected by electric motors which activate them. Robots can be outfitted with tools that can be used for a variety of tasks, like for instance, welding, gluing, painting etc [11].

The setup and programming of actions and sequences for an industrial robot are commonly facilitated by establishing a connection between the robot controller and a laptop, desktop computer, or an internal or Internet network. A workcell, or robot cell, is a robot and a group of equipment or accessories. A single computer or PLC serves as both the controller and "integrator" for the numerous devices [12]. Programming is required to control how the robot communicates with the other machines in the cell, as well as how they move about the cell and synchronize with one another.

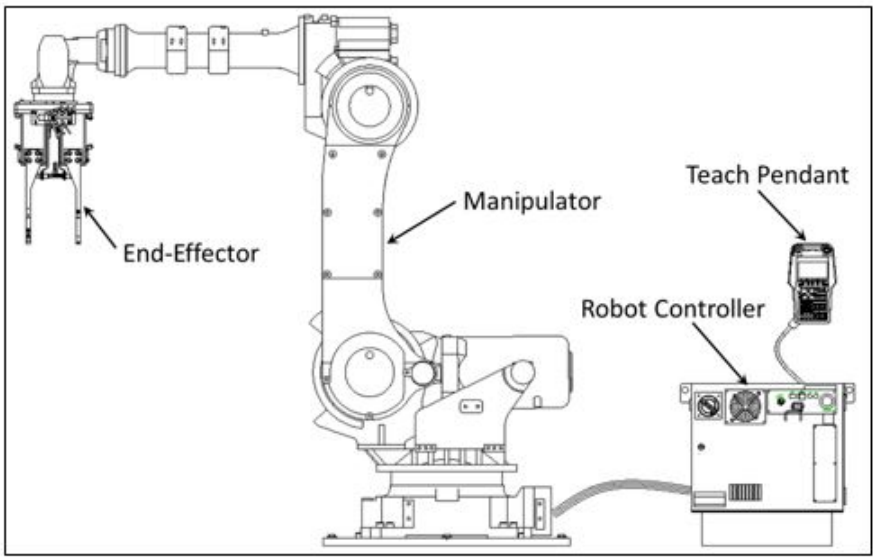

Figure 1: Basic components of an industrial robot system [1]

As shown in Figure 1, the main components of an industrial robot are:

- *Manipulator.* Refers to the robot's arm and its base. It includes a structural frame that is constructed and equipped to support various mechanical components, such as joints, actuators, sensors, control valves, limiting devices, and guides.
- *Robot Controller.* Serves as the central processing unit of the robot and is responsible for its overall functioning. It is an information-processing and communication device that controls the movements and actions of the robot. In industrial settings, the majority of robots are equipped with controllers that are based on computer or microprocessor technology.
- *End-Effector.* Used to perform the necessary tasks for a given application. Its specific function may vary depending on the application at hand, with some robots possessing a single function, such as spot welding or spray-painting.
- *Teach pendant.* Handheld device that allows users to manually move the robot to a certain position and teach two basic entities: positional data and procedure.

2.2. I/O Devices

Input/output (I/O) devices are devices that allow robots to interact with their environment by receiving inputs and providing outputs. In industrial robot systems, I/O devices facilitate the exchange of data between the robot controller and the mechanical components, either sending data from the controller to the peripherals or receiving data from them. In addition, I/O devices can serve to connect multiple robot controllers or link multiple robots to a central computer [10].

I/O devices are used to control and monitor the various components of the system, such as sensors, actuators, and other peripheral devices [6]. For example, I/O devices can be used to monitor the temperature of a manufacturing process, detect the presence of objects in the robot's workspace, or control the flow of materials on a conveyor belt. They can also be used to control the movement and positioning of the robot's end-effector, such as a gripper or welding tool.

2.3. Emulation

An emulator is a piece of hardware or software that allows one computer system, referred to as the host, to mimic the actions of another, referred to as the guest. Typically, an emulator allows the host system to use peripherals or run programs created for the guest system [13].

In an emulation, the systems and processes that are imitated are run by the emulator. In this way, the emulation mimics the states, behaviors, and outcomes of process and system implementations as well as real-world models. In addition to imitating processes and yielding the same outcomes as the original systems, an emulation also needs the process's execution to be equal (or at least fairly comparable) to those systems.

For instance, an emulator can enable the use of a dated device on a modern computer system. In addition, the opposite can also be accurate - in older and simpler machinery, we can replicate complicated modern machines (or elements of them). Examples of emulations include video game emulators (i.e. running console or mobile games on desktops and vice versa), printer emulators and CPU emulators (i.e. QEMU).

The generalized process of constructing an emulator of a system is shown in Figure 2.

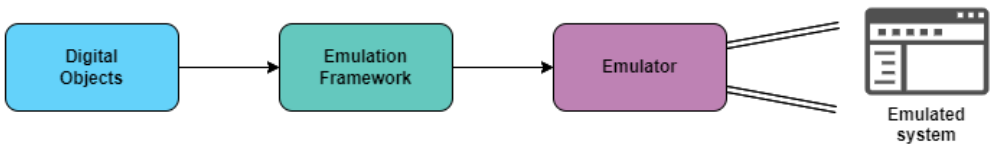

Figure 2: Process for constructing an emulator

As shown in the Figure 2, in order to create an emulator, we often need an emulation framework. All of the original system's hardware and software components must first be described as digital objects. The emulation framework receives and processes these objects and subsequently provides a corresponding emulator software. Finally, we may run the emulator to render and start the desired system [14].

It is also important to note that simulators and emulators, while similar, have distinct differences. A simulator focuses on a system model and attempts to replicate some of the conditions and processes that result in a certain outcome. An emulator offers a recreated setting where these conditions can be observed and these operations can be carried out as in the original system, which allows for additional investigation into system routines and characteristics.

2.4. OSI Model

The Open Systems Interconnection model (OSI model) is a conceptual model that "provides a common basis for the coordination of [ISO] standards development for the purpose of systems interconnection" [15]. Physical, Data Link, Network, Transport, Session, Presentation, and Application are the seven distinct abstraction layers into which the communications between computing systems are divided in the OSI reference model, shown on Table 1. The functions of each layer in the OSI model are clearly defined, and each layer communicates and interacts with the layers directly above and below it, unless there are no layers above or below the layer in question.

Layer			Protocol Data Unit (PDU)	Function
Host Layers	7	Application	Data	High-level protocols such as for resource sharing or remote file access, e.g. HTTP.
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media Layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Transmission of data frames between two nodes connected by a physical layer
	1	Physical	Bit, Symbol	Transmission and reception of raw bit streams over a physical medium

Table 1: OSI model Layer architecture [5]

2.5. The Internet Protocol Suite (TCP/IP)

The Internet protocol suite, also referred to as TCP/IP, is a framework for categorizing the collection of communication protocols used on the Internet and other comparable computer networks. The most popular information networking protocols are TCP/IP and OSI. The primary distinction is that OSI is a conceptual framework and not a real-world communication tool and it instead specifies how apps can talk to one another over a network. On the other hand, TCP/IP is frequently used in practice to create connections and communicate across networks [16].

TCP/IP establishes end-to-end communication that specify how data should be divided into packets, addressed, transmitted, routed, and received at the target. TCP/IP is used to describe how data is exchanged over the internet and is intended to make networks reliable by enabling them to autonomously recover from the failure of any device on the network with minimal central management requirements.

TCP/IP employs the client-server model of communication, where a user or machine (a client) requests a service from another computer (a server) on the network, such as sending a webpage. The TCP/IP suite of protocols is categorized as stateless [17], which means that since each client request is independent of earlier requests, it is regarded as fresh. Being stateless liberates network

routes, allowing for continuous use. However, the transport component itself is stateful [17]. It sends a single message and maintains its link until all of the message's packets have been received and assembled at the destination.

2.6. Transmission Control Protocol (TCP)

TCP, or Transmission Control Protocol, is a protocol used for communication between devices on a network. It is a connection-oriented protocol, meaning that before any data is exchanged between two devices, a virtual "circuit" or connection is established between them [18]. This connection ensures reliable delivery of data and error detection.

The process of establishing a TCP connection is known as the "*three-way handshake*". It involves three steps [19]:

1. The first step is the **SYN (synchronize) packet**. The device that initiates the connection sends a SYN packet to the device it wants to communicate with. This packet contains a random sequence number and other information needed to establish the connection.
2. The second step is the **SYN-ACK (synchronize-acknowledge) packet**. The receiving device responds to the SYN packet with a SYN-ACK packet, which contains an acknowledgement number that confirms the receipt of the initial SYN packet. The SYN-ACK packet also contains a random sequence number.
3. The third step is the **ACK (acknowledge) packet**. The device that initiated the connection sends an ACK packet to the receiving device, confirming the receipt of the SYN-ACK packet. The ACK packet also contains the acknowledgement number from the SYN-ACK packet.

Once the three-way handshake is complete, the TCP connection is established and data can be exchanged between the two devices.

To terminate a TCP connection, a device sends a **FIN (finish) packet** to signal that it has finished sending data. The other device responds with an **ACK packet** to confirm the receipt of the FIN packet [19]. The other device may also send its own FIN packet to signal that it has finished sending data. The first device responds with an ACK packet to confirm the receipt of the FIN packet. Figure 3 shows a typical TCP connection establishment and termination (without any data transfer).

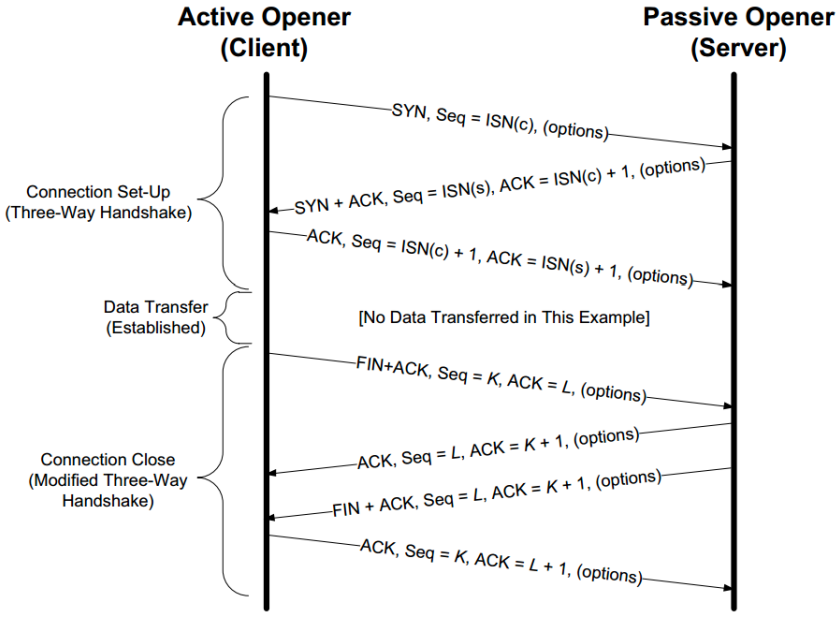

Figure 3: Standard TCP connection establishment and termination [2]

2.7. Internet Protocol (IP)

The Internet Protocol (IP) is a network layer protocol that is responsible for the transmission of data packets between hosts in a network. IP operates in a best-effort delivery mode, which means that it does not guarantee delivery or provide error checking and recovery [18]. IPv4 is the fourth version of the IP protocol and is still widely used today, despite the emergence of IPv6. IPv4 addresses are 32-bit binary numbers, commonly expressed with four numbers ranging from 0 to 255 separated by dots. This allows for approximately 4.3 billion unique addresses, which are becoming scarce due to the explosive growth of the internet.

The general format of an IPv4 packet includes a header and a payload. The header is divided into several fields, including the version of the IP protocol being used, the length of the header, the type of service, the total length of the packet, the source and destination IP addresses, and several other flags and fields [20]. The payload of the packet contains the actual data being transmitted.

2.8. EtherNet/IP Protocol

EtherNet/IP is one of the leading industrial protocols in the United States and is widely used in a range of industries including factory, hybrid and process. It is an application layer protocol that is transmitted within a TCP/IP packet. Therefore, EtherNet/IP is nothing more than the manner that data is structured within a TCP or UDP packet [21].

Every device connected to an EtherNet/IP network presents its data to the network as a collection of data values known as attributes that are collected together into groups of attributes known as objects. These objects can be *required* objects and are defined by the EtherNet/IP standard, and they can be *application* objects which are specific to the device in question. Ethernet/IP devices exchange two different types of messages: implicit or I/O messages and explicit messages.

EtherNet/IP is part of the Common Industrial Protocol (CIP). CIP defines the object structure and specifies the message transfer. CIP protocol over Ethernet is EtherNet/IP [22].

2.9. Wireshark

Wireshark[1] is a widely used network protocol analyzer tool that is used to capture and analyze network traffic in real-time. It is commonly used for network troubleshooting, security analysis, and network protocol development.

The main functionalities of Wireshark include [23]:

1. **Network Traffic Capture**: Wireshark allows users to capture and examine network traffic in real-time from multiple network interfaces.
2. **Protocol Analysis**: It can decode and dissect a wide range of network protocols, including TCP/IP, HTTP, DNS, and many others. Wireshark provides detailed information about the network traffic, including packet contents, source and destination addresses, and timing information.
3. **Filtering and Search**: Wireshark provides powerful filtering and search capabilities, allowing users to easily search and isolate specific packets or traffic patterns.
4. **Packet Reconstruction**: Wireshark can reassemble packets that have been fragmented across multiple network segments, allowing users to analyze complete network transactions.
5. **Graphical Analysis**: Wireshark provides a range of visualization and graphical tools to help users identify patterns and trends in network traffic.

[1] https://www.wireshark.org/docs/wsug_html_chunked/

2.10. Network Traffic Replay

Network traffic replay is the process of reproducing previously recorded network traffic in a controlled environment for the purpose of analyzing or testing network-based systems and applications. This involves capturing network packets from a live network or from stored packet capture files, and then replaying them on a network test bed or simulation environment [24]. Traffic capture and replay refers to the practice of recording all requests, regardless of protocol or content, with the aim of replaying them at a later time or in a new setting.

Network capture and replay can be visualized using a DVR device analogy. In Figure 4, the traffic that has been captured is duplicated in a format that can be played back later or even altered for usage in various settings.

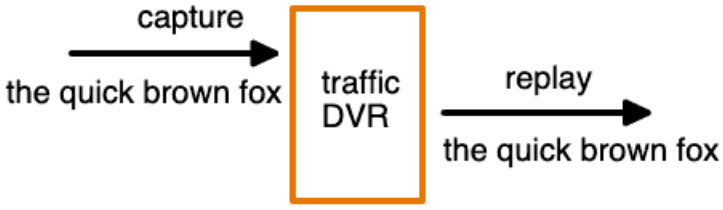

Figure 4: Visualization of traffic replay through DVR analogy [3]

Traffic replay implementations vary depending on the objectives and method used, and there are several levels based on available resources. Capture is just as crucial as replay because it affects what can be done during the replay phase. Different levels of network capture and replay are [3]:

- **Level 1**. Network layer 4 is typically where level 1 capture operates. The same bytes are replayed after being collected. The traffic that was recorded and replayed are identical. This level is helpful for accurately simulating traffic. Replayed traffic is one-to-one, therefore there is no need to comprehend protocol or composition. Network packets and raw data are the only things that can be analyzed. Figure 4 accurately represents how level 1 capture operates.
- **Level 2**. Level 2 and higher utilize network layer 7. This level might offer introspection into the body, headers, and URL of an HTTP request.
- **Level 3**. Level 3 offers manual request rewriting. Request information can be manually changed. URLs in HTTP can be changed. It is possible to modify, add, or remove headers. With the aid of a script or configuration that is provided, the traffic DVR permits modification during replay.
- **Level 4**. Finally in level 4, the intelligence layer that sits on top of the traffic allows for automatic detection and rewriting of traffic or the creation of automated suggestions without the need for human participation. Although it is the hardest to attain, Level 4 offers the biggest benefits to apps during replay. Figure 5 visualizes level 4 capture and replay using the same DVR analogy as before.

Figure 5: Visualization of level 4 traffic replay through DVR analogy [3]

Network traffic replay is a valuable tool for security professionals to test the effectiveness of security measures, such as firewalls and intrusion detection systems, by replaying known attack patterns. It can also be used to reproduce real-world network conditions to test the performance and reliability of network-based applications and services.

3. Method

In this thesis work, the research will employ a multi-methodology research approach based on the framework proposed by Nunamaker et al. [4]. The research will follow an iterative process, building upon previous findings to guide subsequent steps and expand the investigation. Such research cycle is presented in Figure 6.

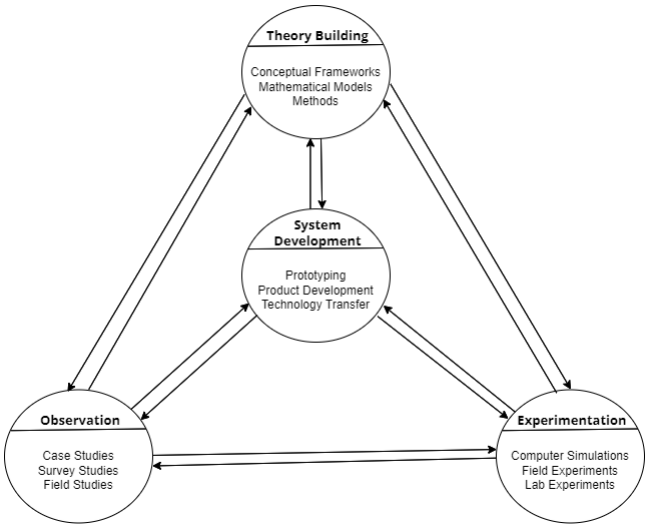

Figure 6: Multi-method research approach [4]

The initial stage of "theory building" proves its significance by providing potential insights that may challenge existing information and predictions. These insights then offer valuable guidance for future model design, experiment design, and practical implementation. The subsequent stage, "experimentation," involves conducting simulations, laboratory experiments, and field experiments. The design of these experiments is driven by the established theories and systems development, and their results contribute to the refinement of these theories and systems. This phase bridges the gap between theory building and observation by providing validation. The "observation" stage serves to shape relevant questions and refocus the investigation, employing methodologies such as case studies, field studies, and sample surveys. The final stage is the multi-stage process of "system development."

Given the thorough research process described previously and adopting a comparable multi-methodology approach for this thesis, a flowchart was constructed that better aligns with the specific case of the thesis. This flowchart presents the specific phases that must be followed, as illustrated in Figure 7.

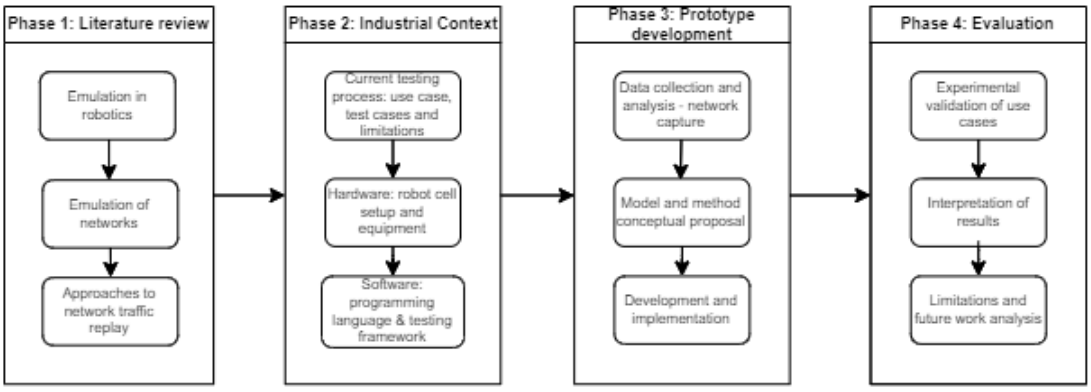

Figure 7: Methodological procedures applied on the system under study

The four phases shown in the Figure are:

- **State of art literature review**. This implies a thorough analysis of the work that is currently done in the field and gaining a deeper understanding of it. This will encompass analysis of the subject matter and background to get a better understanding. Research into networks, protocols, network packets and network communication will need to be done. Current approaches of emulation in the robotic industry, as well as beyond, will be reviewed. Following that, publications that cover emulation of networks or network virtualization will be compiled. The approaches in these works will be reviewed.
- **Industrial context**. This phase will cover research within the company that provides context to the current process of testing robot controller software. This will include insight into the current setup of robot cells, relevant technologies, programming languages, development frameworks and testing frameworks used, analysis of the equipment and devices utilized in the process. The outcome from this research will address clear limitations of the current process and establish a motive for the project. An approach will be chosen that from the first phase that fits the industrial context and will serve as an answer to **RQ1**.
- **Prototype development**. This encompasses system development and the creation of a prototype. Any relevant data analysis will be performed in this phase. Following that, a conceptual model of the prototype will be generated. This is followed by the actual implementation and development of the model and will result in a solution prototype for the project.
- **Evaluation**. The evaluation of results takes place by conducting multiple experiments across various use cases: recognition of network request, generation of network responses and mimicking of the actual communication. A performance review will be performed for the solution. The outcomes of this phase will serve as the answer to **RQ2**. Finally, limitations, academic and practical contribution of the work and future work will be discussed.

4. Related Work

As this study is new, there has been no prior investigation into the emulation of I/O devices for testing robot controllers or within the robotics industry. As a result, there are no publications specifically addressing this matter. Thus, the literature review focuses on exploring publications that highlight different aspects of emulation - emulation of network communication and emulation in robotic systems. The results are presented below.

4.1. Application of Emulation in Robotics Industry

Emulation is a valuable tool in the robotics industry for testing and validating robot controller software and hardware before deployment. It allows engineers to simulate the behavior of a robot in a virtual environment, which can save time and resources compared to physical testing. Furthermore, emulation can aid in the development of new robotic systems by allowing engineers to test and validate different hardware and software configurations before building physical prototypes. This can help to identify potential design flaws or performance issues early in the development process, ultimately leading to more efficient and effective robot systems.

The paper "Emulation of robots interacting with environment" by F. Aghili and J.C. Piedboeuf [25] presents a framework for emulating the behavior of a robot interacting with its environment using a virtual environment. The authors demonstrate the effectiveness of their approach by applying it to several real-world robotic applications, including obstacle avoidance, grasping and manipulation, and path planning. The results of the paper show that the proposed emulation framework can accurately model the behavior of a real robot in a virtual environment. The authors also demonstrate that the framework can be used to test and evaluate the performance of robot control algorithms and to develop and refine robot control strategies.

Aghili and Piedboeuf also published a paper [26] four years prior where they propose a method for hardware-in-the-loop (HIL) simulation of robots interacting with their environment. Specifically, they focus on simulating contact dynamics between the robot and its environment, which is essential for accurate modeling and control of physical interactions. The proposed method involves using an emulated environment, which allows for real-time simulation of contact dynamics. The authors also propose a novel algorithm for solving contact dynamics equations in real-time, which is critical for achieving accurate and stable simulation results.

Ghalazman et al. [27] present a method for robot learning from human demonstrations in the presence of dynamic obstacles. The proposed approach combines dynamic motion primitives with Gaussian mixture regression to learn the desired trajectories for the robot to avoid obstacles and achieve its task. The method is tested on a robotic arm and a mobile robot in simulation and real-world experiments. The results show that the proposed approach outperforms other learning from demonstration methods in terms of success rate, smoothness of motion, and obstacle avoidance. One limitation of this work is that the method requires a large amount of demonstration data to effectively learn the desired trajectories, which may not always be feasible in real-world applications. Additionally, the approach assumes that the robot has full knowledge of the environment and the motion of the obstacles, which may not always be the case in practical scenarios.

The lack of found literature indicates that emulation is still a developing technology in the robotics industry. Current limitations make it hard to emulate and sometimes even simulate robots, robotic systems or even peripheral equipment. The papers that were reviewed show the benefits of emulation in the industry but don't relate much to the specific problems in this thesis work other than being concentrated in the same field.

4.2. Application of Emulation for Network Virtualization

The previous section had a focus in literature on emulation in the robotics industry in general, whereas this section will explore the emulation of networks. Emulation can be widely used in

computer network virtualization to create virtual environments for testing network protocols, applications, and configurations. Network emulation involves creating a virtual network with the same characteristics as a real network, such as bandwidth, latency, packet loss, and other network impairments. This virtual network can then be used to test and evaluate network applications, protocols, and configurations in a controlled and repeatable environment.

The thesis "Emulation of industrial Fieldbus modules for Virtual Commissioning" [22] proposes a method for emulating industrial fieldbus modules to facilitate the process of virtual commissioning. The approach aims to save time and costs in the commissioning phase of industrial automation systems. The proposed method emulates fieldbus modules using virtual controllers and interfaces, allowing for the virtual commissioning of automation systems before the physical systems are built.

Another thesis by Huang Benjamin [28] focuses on the communication between a virtual emulation system and a programmable logic controller (PLC) using the Modbus/TCP protocol. The goal of the research was to develop a communication interface between a virtual emulation system and a real PLC to facilitate the testing and validation of control systems. The communication interface between the two systems was implemented using a Modbus/TCP library for the virtual emulation system and a Modbus/TCP module for the PLC. The communication interface was successfully implemented, and various tests were conducted to validate the communication performance and reliability.

Maeda et al. [29] propose a network emulator that can capture and replay network traffic for testing and developing networked applications in a controlled environment. The results of their experiments show that the emulator can accurately reproduce a wide range of network conditions and that it can be used to test the performance and robustness of networked applications.

Cheng [30] performs network emulation by generating new traffic using generative adversarial networks which can be applied for network traffic replay, intrusion detection testing, and other security applications. Overall, the work provides a novel and promising approach to generating synthetic network traffic that can be used for a variety of purposes. However, further research is needed to evaluate the performance and scalability of the approach in large-scale network environment.

Matousek et al. [31] performed a similar emulation and proposed a method for generating precise IPv4 and IPv6 packets. The method is designed to enable precise control over the content and behavior of the generated packets, and to facilitate testing and validation of network devices and protocols. Nicolae et al. [32] performed packet generation for the LLDP protocol for the purpose of different testing scenarios.

Other works also address the problem of network device emulation and network traffic virtualization. Baba et al. [33] propose a network emulator and demonstrate the effectiveness of it by presenting several case studies and comparing the emulation results with those obtained from real-world experiments. The paper presents a useful tool for researchers and developers working on sensor networks. Niculaescu et al. [34] perform packet generation to get a better view of the network topology for the purpose of network troubleshooting.

The literature extracted shows there isn't a unique approach to emulation of network communication. However, many publications were related to approaches in network traffic capture, packet modification and subsequent replay. This is why a comprehensive study into the current research of these approaches was also performed.

4.3. Approaches to Network Traffic Replay

TCPReplay[2] is a tool for replaying network traffic from pcap files. It can be used to test network devices, applications, and services by replaying previously recorded traffic. However, TCPReplay

[2]https://tcpreplay.appneta.com/

supports only a limited number of protocols and replays network traffic exactly as it was recorded, without the ability to modify the traffic. It also cannot generate new traffic on its own, which limits its use in certain testing scenarios. Consequently, research into the removal of TCPReplay's limitations has been done.

Wu et al. [35] propose an alternative traffic replay method where experimental results show that the traffic generated by their method is completely consistent with original traffic. The method is based on interactive sequence and timestamp, which may not be applicable in all network scenarios or traffic patterns.

Similarly, Vam et al. [36] propose an original method to split traffic prior to replay using IP aggregation leading to high-speed backbone traffic replaying. Qiao et al. [37] present both a traffic capture and a replay system which interprets the detailed logic design of UDP pipeline on FPGA.

Toll et al. [38] also present a traffic replaying tool that allows the synchronous replaying of network traffic with multiple endpoints and connections. The resulting implementation is able to accurately replay connections within a maximum transmission rate but struggles with deviations from regular TCP connections, like packet loss or connection reset.

No unique and universal approach exists for network traffic replay and deeper understanding of the communication and its transmitted packets, protocols, layers, headers and payloads is needed for an effective and efficient replay method. The methods from these publications can not be directly applied to this thesis work, but they form a basis and show foundation on how they were used in different circumstances. The current research also lacks in proposing an approach which combines both the real-time capture of packets and their subsequent modification that would mimic real-time responses from network devices.

5. Industrial Context

5.1. ABB Robotics

ABB Robotics is one of the world's leading suppliers of robotics and machine automation, offering products such as industrial, collaborative and autonomous mobile robots, robot controllers and accessories, as well as robot cells.

In the R&D Software Test department, team IOTA's mission is to deliver software and lab tools and equipment, so that other software developer teams can test the robot controller software called Robotware. One such tool is the so-called TestEngine which acts as an interface to the robot controller software, while another tool is used to schedule and run tests automatically every night. The team builds and develops robot cells which are later used for running test cases. The robot cells contain hardware with an abstraction layer in software which can communicate with the hardware.

5.2. Setup in Robot Cells

Certain subsets of test cases are used to determine whether the robot controller can handle heavy network traffic. For that purpose, many I/O devices are utilized within robot cells to generate that traffic. Test cases are run on a PC agent which is connected to a switch that allows communication with both the robot controller and the I/O devices. The test cases usually contain instructions, in the form of configuration files, that tell the robot controller software how to interact and communicate with the I/O devices. This scheme of the setup in the robot cell is shown on Figure 8.

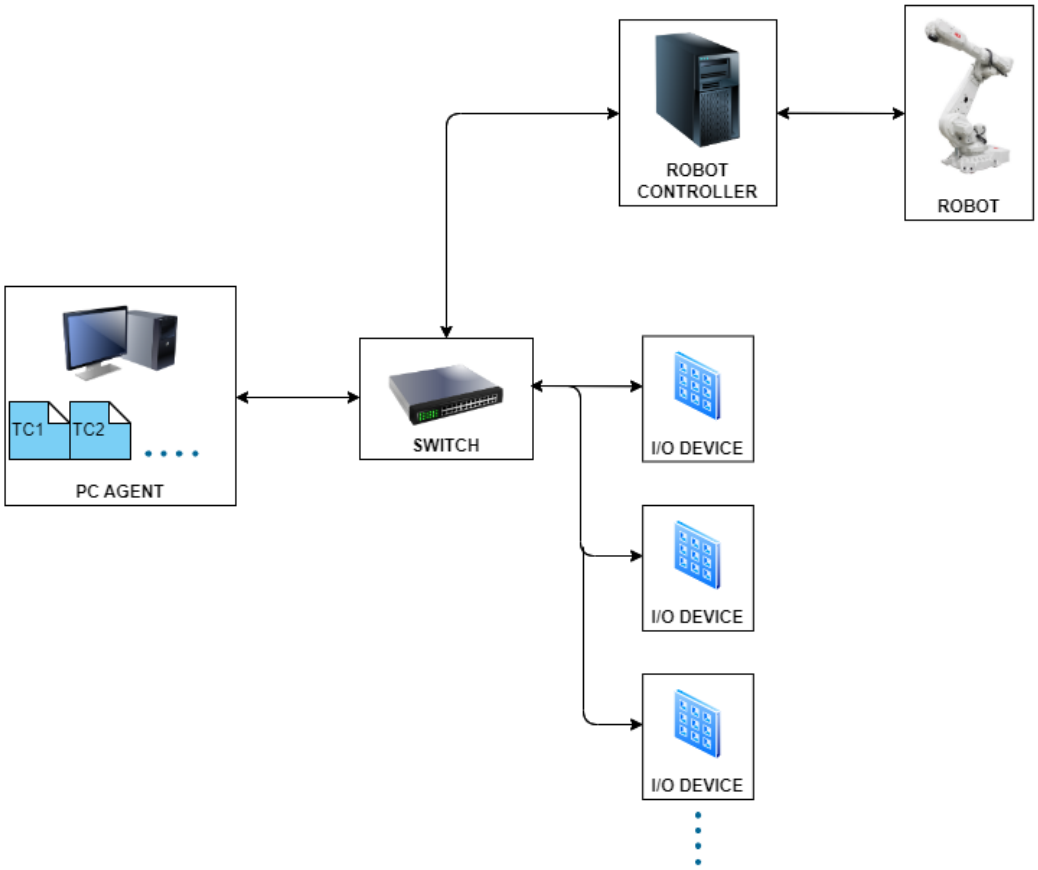

Figure 8: Robot cell setup

5.3. ABB Scalable I/O Device

ABB's scalable I/O [3] devices are designed to provide flexible and reliable input/output (I/O) capabilities for industrial automation systems. They are used to connect various sensors, actuators, and other field devices to the control system. The scalable I/O devices are modular, which means they can be easily added or removed as needed, allowing for easy expansion or modification of the system. They come in a variety of form factors, from compact to distributed I/O, to suit different application requirements The devices use different communication protocols such as Profibus, Profinet, Modbus, EtherNet/IP, and DeviceNet to connect with different types of controllers and systems, but the thesis work will focus on the devices using EtherNet/IP. They also feature built-in diagnostics and advanced safety functions to ensure reliable operation and reduce downtime.

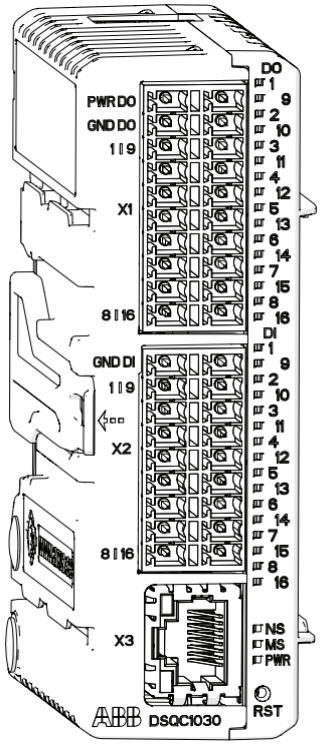

Figure 9: ABB scalable I/O device

The different devices and add-ons that are utilized:

- DSQC1030 - digital base and has 16 digital inputs and 16 digital outputs
- DSQC1031 - digital add-on and allows additional 16 digital inputs and 16 digital outputs
- DSQC1032 - analog add-on and has 4 analog inputs and 4 analog outputs
- DSQC1033 - relay add-on with 8 digital inputs and 8 relay outputs
- DSQC1042 - Safety digital base with 12 digital safe inputs and 4 digital safe outputs

Figure 10 shows how the different devices can be combined:

[3] Link to resource

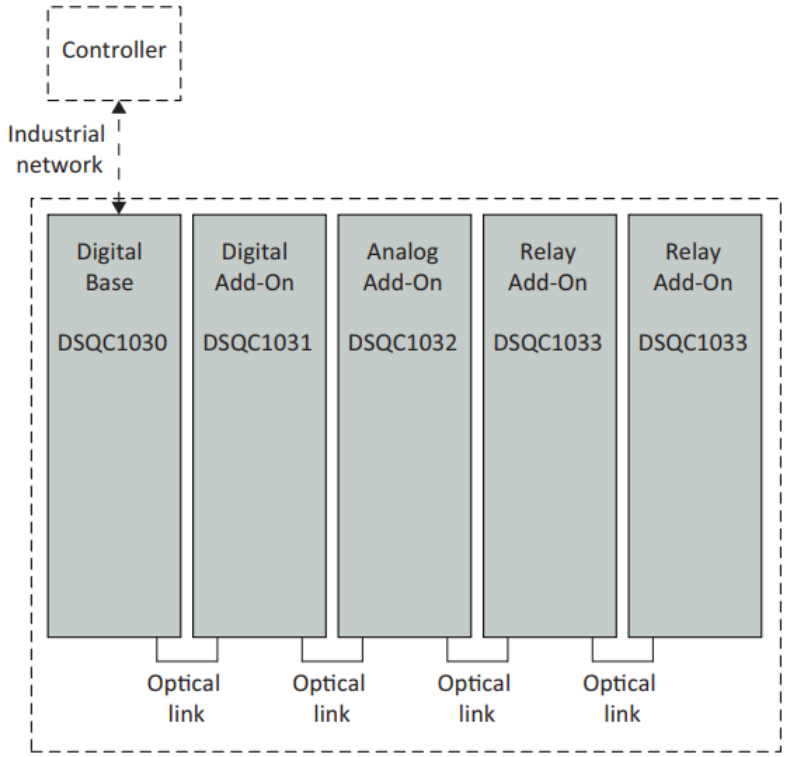

Figure 10: Combination of various devices

It is important to note that this thesis work, due to the scope of the project and complexity, will only focus on emulating communication with the digital base, i.e. with I/O devices that contain only the digital base. The I/O device, therefore, supports two different services (functionalities): one for setting a single attribute and one for getting a single attribute. Setting a single attribute means to modify it while the latter returns the contents of the specified attribute. Setting a single attribute is, therefore, an input signal and getting a single attribute an output signal.

5.4. Abstraction Layer

As mentioned, an abstraction layer in the software is present that allows for the current communication between the PC agent and the numerous I/O devices. This layer is written in C# and utilizes the EEIP library[4]. The library allows data exchange with Ethernet/IP devices, which are essentially the I/O devices used in ABB's robot cells. It also supports explicit and implicit messaging and provides a simple way to access Ethernet/IP devices without special knowledge about Ethernet/IP. The library does this by creating a client-server connection between the PC agent that is running the test cases and the I/O devices. Any data exchange is then handled by either the TCP protocol or EtherNet/IP protocol.

This abstraction layer can be considered as a partial emulator, given that it creates digital objects for the I/O devices and stores their attributes in said objects. Furthermore, it also mimics some of the devices' behaviour, i.e. when setting a single attribute to *true* on the I/O device, the digital objects perform this by doing the operation *bitwize or* of the first 8 bits against all zeros except bit value (provided that the index of the attribute is between 1 and 8). However, the robot controller and PC agent still need to be physically connected to the I/O devices via ethernet cables, whereas this thesis work aims to find an approach that would make the connections obsolete. One way to achieve this is to generate responses in the name of I/O devices and send them to the client in a seamless way.

[4]http://eeip-library.de/

5.5. Problems with Current Process

Unfortunately, the current approach has its limits. Namely, such configuration of the test process is highly dependable on the hardware (I/O devices and ethernet cables), meaning these test cases can only be performed if such hardware is installed at the site. As stated above, the hardware consists of many I/O devices and therefore requires significant amount of space at the site.

Apart from that, these hardware setups aren't cost-effective either. Just one I/O device is $400-$500, and sometimes dozens, if not hundreds, are needed in a robot cell. Furthermore, many I/O units that are used at the Västerås site in the testing process are no longer being manufactured. This creates limitations in the recreation of test cases in different sites and locations of the company across the world.

6. An Approach to Emulating I/O Device Communication with Robot Controller

This section will cover the proposed method and approach for emulation of I/O device communication with the robot controller during testing. Based on the literature review in Section 4. and research into the industrial context in Section 5., the core principle of the approach will be based on network traffic capture, modification and recreation and subsequent replay. The implementation should be preferably done in the C# programming language and .NET framework to align with the current software used in test cases.

6.1. Flow of the Emulation of I/O Device Communication

The emulation will need to perform and possess the functionalities of:

- **Real-time network capture**. The client-server connection that is set up can send requests any time, so the emulator needs to be able to listen and capture these requests real-time and process them accordingly.
- **Recognition of request packets in network traffic**. Recognition is important because the emulator needs to know which response to generate. If the request was a TCP SYN-request, then it needs to be able to know what SYN requests are and how to recognize them in a pool of other requests.
- **Import and modification of historical network capture data**. Assuming that responses in the communication follow a certain pattern, utilizing already captured data and only transforming relevant parts makes the process more efficient.
- **Subsequent generation of new response packets**. Once the responses are generated, they need to be sent to the client.
- **Sending of network packets through a virtual interface**. Given that no physical connection can be present during testing, network packets must be sent through a virtual interface.

As mentioned above, the emulator will utilize a virtual network interface (VNI) that is generated by a loopback adapter. A loopback adapter, also known as a loopback interface or loopback address, is a virtual network interface that allows a computer to communicate with itself over a network. It is often used for testing purposes or to provide a way for applications to communicate with themselves without the need for an actual network connection. This is needed due to the fact that the emulator needs to be independent of any ethernet cable connections to existing I/O devices.

Figure 11 presents a flowchart for the above mentioned process.

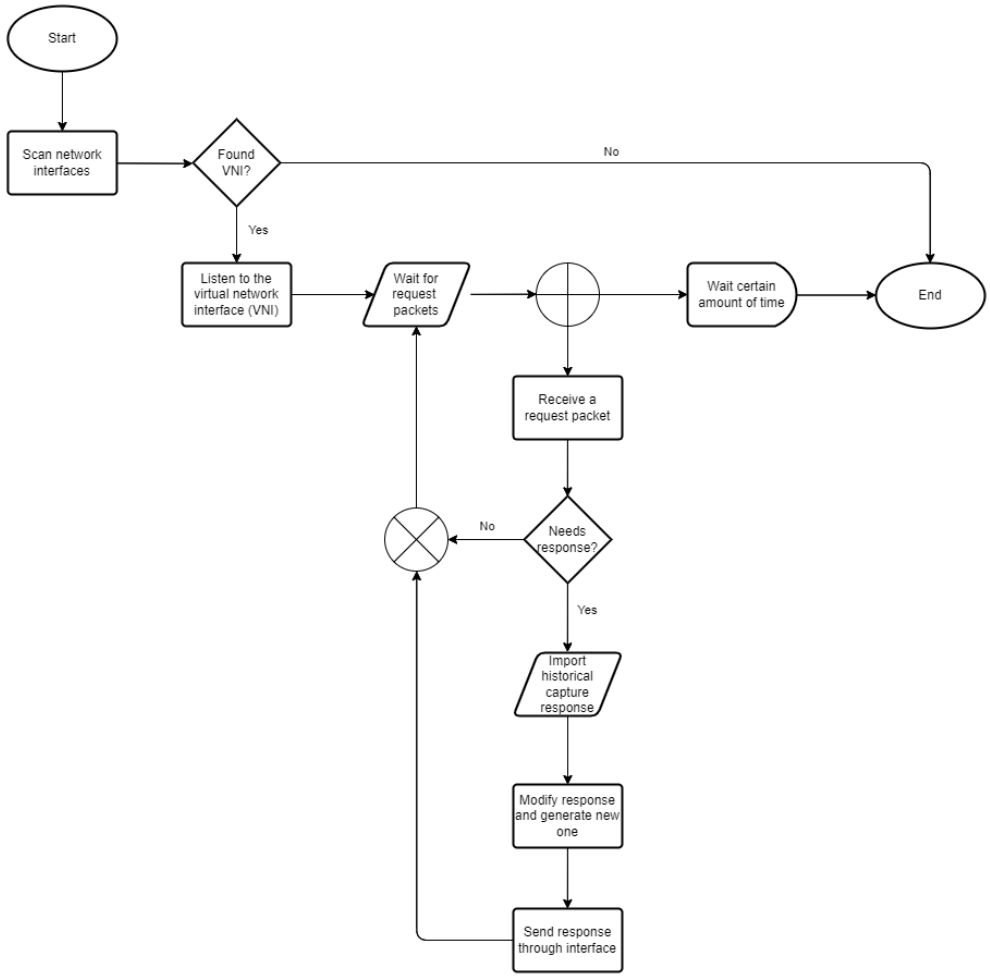

Figure 11: Flowchart of the proposed approach

As seen on Figure 11, the emulator will first scan the machine for available network interfaces and search for the virtual network interface on the loopback adapter. If it's not found, the emulation will end. Once the interface is found, it will start the process of listening to the interface and capturing any incoming traffic. If nothing arrives before a certain timeout, the emulation will end. However, if a packet arrives it will be captured and examined. If the emulator concludes that a response is not needed, it will continue to listen to new packets and simply ignore the captured one. If a response is needed, the emulator will proceed to response generation. First, a .pcap file will be imported containing a historical record of a previous response. Next, relevant modifications are going to be made. This will be described in later sections. Finally, the response will be sent to the interface and the emulator will start listening to the interface again for new packets.

During the interview process it was concluded that the solution should be implemented in C#. Therefore, this approach will be implemented with the help of Pcap.NET[5], a library for capturing and sending network packets using the .NET framework, written in C#. It is a wrapper around the libpcap/WinPcap library, which is a popular library for capturing live network traffic on Unix-like systems. Pcap.NET provides an easy-to-use API for capturing network packets in managed code. It supports a variety of capture options, such as filtering, timestamping, and promiscuous mode. It also supports sending packets on the network using the raw socket API.

Pcap.NET provides the framework for the following features:

[5]https://github.com/PcapDotNet/Pcap.Net

- Getting the list of Live Devices on the local host. The local host usually possesses multiple network interfaces and devices through which network traffic can be exchanged. This library is able to access all of them through name, description or IP address.
- Reading packets from Live Devices (Network Devices) and Offline Devices (Files). The library is capable of real-time live network capture, as well as import of packets from previous captures from files (.pcap files).
- Receiving statistics on full capture and statistics of packets instead of the full packets. The library allows easy access to metrics surrounding the network capture without the need to process the capture itself.
- Applying Berkley Packet Filters. This allows easier control over which packets get processes and handled, i.e. it is possible to listen to only TCP traffic from port 80.
- Sending packets to Live Devices directly. The library allows for direct intrusion of manually built or already captured packets (replay).
- Dumping network packets into .pcap files. It is possible to extract and export packets.
- Creation and build of network packets and customization of layers, headers and payloads

6.2. Network Traffic Capture

In order to gain understanding of the behaviour that needs to be emulated, communication between the robot controller software and the I/O devices needs to be analyzed. For this purpose, one I/O device was connected to a PC agent via an ethernet cable and the subsequent traffic was recorded. The PC agent acts as the client and the I/O device as the server. The traffic was captured using Wireshark, a free and open-source network protocol analyzer. The result of the capture is shown on Figure 12

No.	Time	Source	Destination	Protocol	Length	Info
4	14.038727	192.168.125.5	192.168.125.254	TCP	66	56603 → 44818 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5	14.041445	192.168.125.254	192.168.125.5	TCP	60	44818 → 56603 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1462
6	14.041597	192.168.125.5	192.168.125.254	TCP	54	56603 → 44818 [ACK] Seq=1 Ack=1 Win=64240 Len=0
7	14.043270	192.168.125.5	192.168.125.254	ENIP	82	Register Session (Req), Session: 0x00000000
8	14.044889	192.168.125.254	192.168.125.5	ENIP	86	Register Session (Rsp), Session: 0x00000001
9	14.048303	192.168.125.5	192.168.125.254	CIP	104	Assembly - Set Attribute Single
10	14.049508	192.168.125.254	192.168.125.5	CIP	102	Success: Assembly - Set Attribute Single
11	14.050724	192.168.125.5	192.168.125.254	CIP	102	Assembly - Get Attribute Single
12	14.050992	192.168.125.254	192.168.125.5	CIP	104	Success: Assembly - Get Attribute Single
13	14.051722	192.168.125.5	192.168.125.254	ENIP	78	Unregister Session (Req), Session: 0x00000001
14	14.051809	192.168.125.5	192.168.125.254	TCP	54	56603 → 44818 [FIN, ACK] Seq=151 Ack=119 Win=64122 Len=0
15	14.051973	192.168.125.254	192.168.125.5	TCP	60	44818 → 56603 [FIN, ACK] Seq=119 Ack=151 Win=32768 Len=0 MSS=1462

Figure 12: Network traffic capture in Wireshark

At first, a TCP three-way handshake occurs for the establishment of connection, where the client (PC agent) sends a SYN packet to the server (I/O device). Then, the server responds with SYN,ACK and the client sends an ACK response. The next interaction is done via Ethernet/IP and represents the session registration. One response from the server is expected. After the connection is established and session registered, the devices can proceed to exchange data which is done through CIP. One request and one response per service are expected. In the example in Figure 12, one *Set a Single Atribute* service is called and one *Get A Single Attribute* service is called. Finally, once the connection is complete, the client will send a FYN,ACK packet for the termination of connection and the server will send one back. The communication is further illustrated in Figure 13.

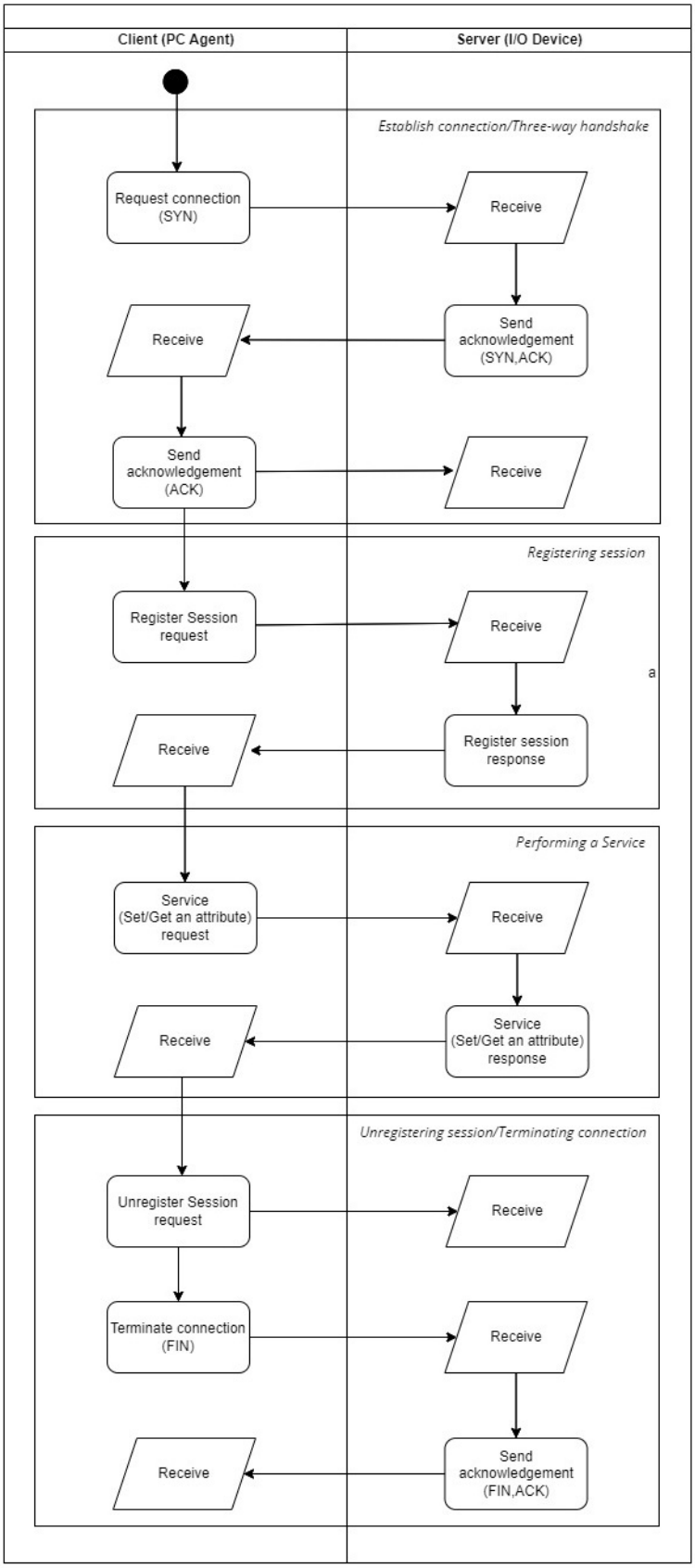

Figure 13: Flow of communication between devices

6.3. Analysis of Network Capture Requests

As mentioned in the previous section, throughout the communication, the client sends requests and expects responses from the server. The emulator, apart from generating responses to those requests, needs to know when to send them. In other words, it needs to recognize incoming request packets and know when to send each of the responses. This subsection will briefly analyze requests and aim to find patterns in them. It is also important to understand that while the emulation will listen to requests, it will also listen to the responses it generates. This why it is crucial to filter out any packets that shouldn't be processed so that the emulation doesn't generate responses when it doesn't need to.

Generally, the destination MAC address and destination IP address can be used to differentiate between packets sent from client and packets sent from the server. However, this wouldn't create a scalable solution. In scenarios where the emulation runs for multiple I/O devices, filtering out each MAC and IP address isn't effective. The following subsection will explore potential patterns in request packets.

6.3.1. Connection Establishment

Recognizing the request for establishment of connection can be done using the packet's TCP flag. However, the response will also have the SYN flag set, so this is not a sufficient condition. The request, unlike the response, will have the ACK flag unset and this can be used for the recognition. Figure 14 shows the state of the flags in the request.

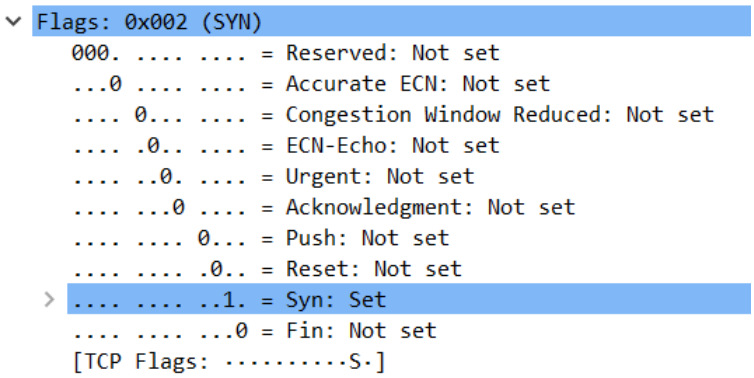

Figure 14: TCP flags in connection establishment request

6.3.2. Registering Session

When the session is being registered, **the Command field** and **Session Handle** in the Encapsulation Header of the Ethernet/IP layer can be examined. Namely, the command field is always 0x0065 and the session handle is 0x00000000 for these types of request. Figure 15 visualizes these fields.

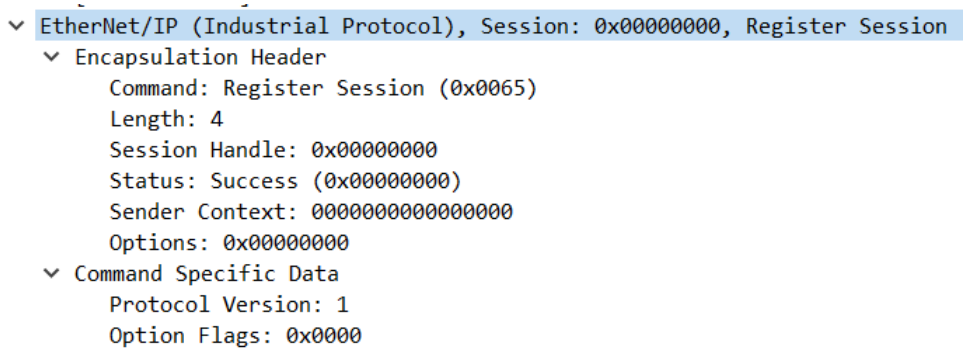

Figure 15: Encapsulation header in register session request

6.3.3. Set a Single Attribute & Get a Single Attribute

When the emulation needs to differentiate between the different services requests, the **Service** field in the Common Industrial Protocol layer can be utilized. For setting a single attribute it is always 0x10 and for getting a single attribute it is always 0x0e. Figures 16 and 17 showcase this.

```
Common Industrial Protocol
  Service: Set Attribute Single (Request)
      0... .... = Request/Response: Request (0x0)
      .001 0000 = Service: Set Attribute Single (0x10)
    Request Path Size: 3 words
  > Request Path: Assembly, Instance: 0x64, Attribute: 3
  Set Attribute Single (Request)
      Data: 1000
```

Figure 16: Service field in setting single attribute request

```
Common Industrial Protocol
  Service: Get Attribute Single (Request)
      0... .... = Request/Response: Request (0x0)
      .000 1110 = Service: Get Attribute Single (0x0e)
    Request Path Size: 3 words
  > Request Path: Assembly, Instance: 0x64, Attribute: 3
    Get Attribute Single (Request)
```

Figure 17: Service field in getting single attribute request

6.3.4. Connection Termination

The request should have the FIN and ACK flags set and have the header length be 20 bytes (unlike the response which has 24 bytes). Figure 18 showcases this.

```
0101 .... = Header Length: 20 bytes (5)
Flags: 0x011 (FIN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Accurate ECN: Not set
    .... 0... .... = Congestion Window Reduced: Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
    .... .... ..0. = Syn: Not set
  > .... .... ...1 = Fin: Set
```

Figure 18: Header length and TCP flags in connection termination request

6.4. Analysis of Network Capture Responses

A thorough analysis was performed on the network capture requests to understand which part of network packets need to be modified and transformed during emulation. Each network packet consists of multiple layers. The layer at the very top is the Ethernet layer. Beneath the Ethernet layer is the IPv4 layer, followed by the TCP layer. Each layer has a header and payload, the payload being the header of the protocol that follows. Because all the network packets in the communication share a similar datagram, similar modification are also made to all packets.

Figure 19 shows a datagram of the response to the connection establishment. However, this is also the datagram for the connection termination and most of the datagram for the other three

responses that need to be generated. In fact, modifications that are made in the Ethernet, IPV4 and TCP layers during connection establishment response generation, should also be made in these layers in other responses. In the Figure, the fields marked red need to be modified during response generation. The values of these field will be addressed in the later sections.

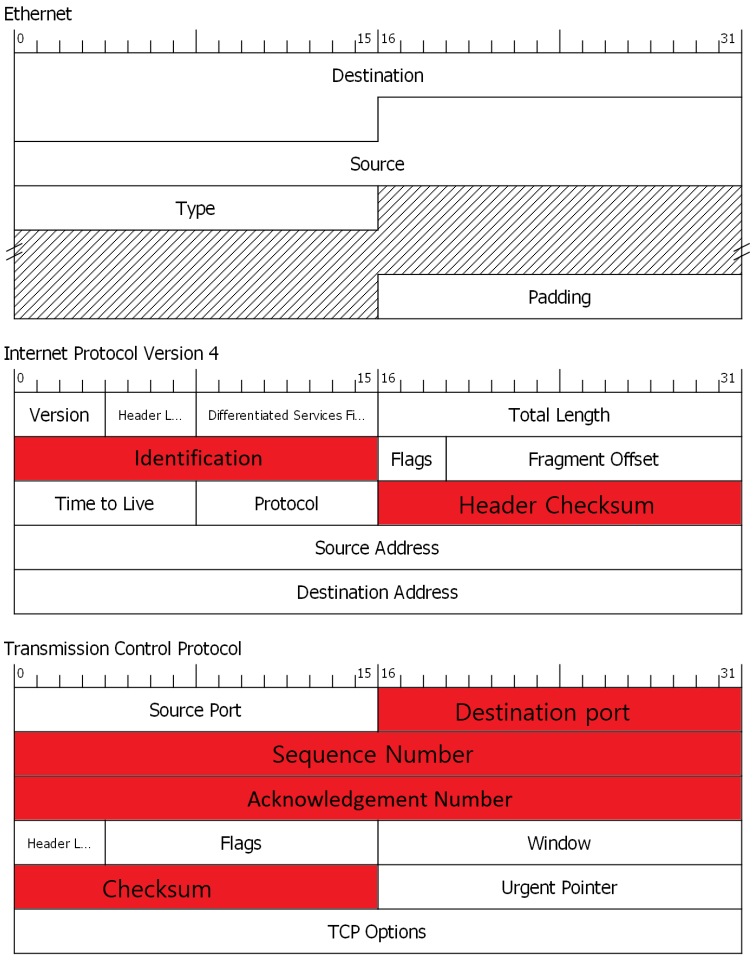

Figure 19: Datagram shared throughut all packets, red indicates fields that need to be modified before replay

The fields that are marked red are:

- In the IPv4 layer
  - **Identification**. This is value is used to identify a unique datagram [20]. We can choose any value for this field as long as it is unique for each packet. For this purpose a random number generator to generate unique values will be utilized.
  - **Header checksum**. This is value is used to verify the integrity of the IPv4 header in a packet. It is calculated over the 20-byte IPv4 header fields, including the source and destination IP addresses, protocol field, header length, and other fields [20].
- In the TCP layer
  - **Destination port**. This field identifies the receiving port number on the destination device that the TCP segment is intended for [19]. This value is the same as the request's source port number and can be obtained from there.
  - **Sequence number**. This field identifies the byte number of the first data octet in the TCP segment. It is used to keep track of the amount of data that has been transmitted and to detect missing or duplicated data. Laters section will explain how this field needs to modified for each of the responses

– **Acknowledgement number**. This field indicates the sequence number of the next expected byte of data from the sender. Later sections will explain what the value of this field is for each of the responses.
– **Checksum**. This field is used for error-checking of the TCP header [19] and will be automatically calculated during packet modification and build.

6.4.1. Connection Establishment

The entire datagram for this response is shown in Figure 19. All the modifications that need to be made during response generation are listed in the section above. What remains is to discuss the values of the fields **Sequence number** and **Acknowledgement number**.

Usually, sequence numbers are calculated by incrementing the previous sequence number by the number of data octets being sent. However, this response is a SYN-ACK response as part of a three-way handshake. Here, the sequence number is chosen at random [19]. The acknowledgment number, on the other hand, is calculated as the request's sequence number incremented by 1 [19].

6.4.2. Registering Session

Registering Session is done via EtherNet/IP protocol and so the packets differ from the packets in the previous section. The first three layers in the datagram are identical, but the packet has an extra layer - EtherNet/IP layer. The packet's EtherNet/IP layer datagram is shown in Figure 20.

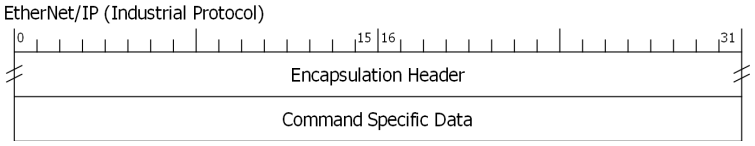

Figure 20: Datagram of the EtherNet/IP layer for Register Session

TCP **Sequence number** field will be set to the request's Acknowledgement number. TCP **Acknowledgement number** field will be obtained by incrementing the request's sequence number by 28. The number 28 is the TCP segment length of the request packet. Other fields, including fields in the EtherNet/IP layer stay the same for each generated response and can be used from previous recordings.

6.4.3. Set a Single Attribute

Setting a single attribute is done via CIP protocol. The first four layers in the datagram are identical as in the previous section. The packet also has an extra layer - the CIP layer. The datagram is shown on Figure 21.

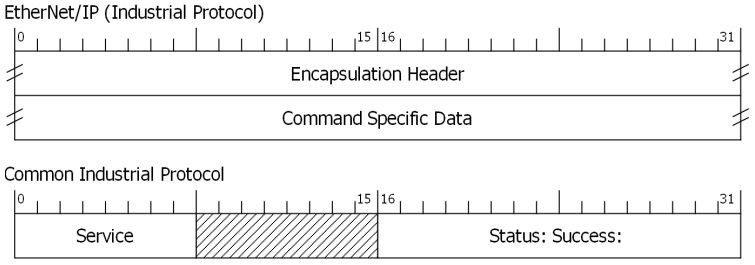

Figure 21: Datagram of the EtherNet/IP and CIP layer for Set A Single Attribute

TCP **Sequence number** field will be set to the request's Acknowledgement number. TCP **Acknowledgement number** field will be obtained by incrementing the request's sequence number by 50. The number 50 is the TCP segment length of the request packet. Other fields, including fields in the EtherNet/IP and CIP layer stay the same for each generated response and can be used from previous recordings.

6.4.4. Get a Single Attribute

This part of the communication is also done via CIP protocol. The first three layers are identical to other responses, while the EtherNet/IP layer (16 bits longer) and CIP layer are different. These layer are shown on diagrams in Figure 22.

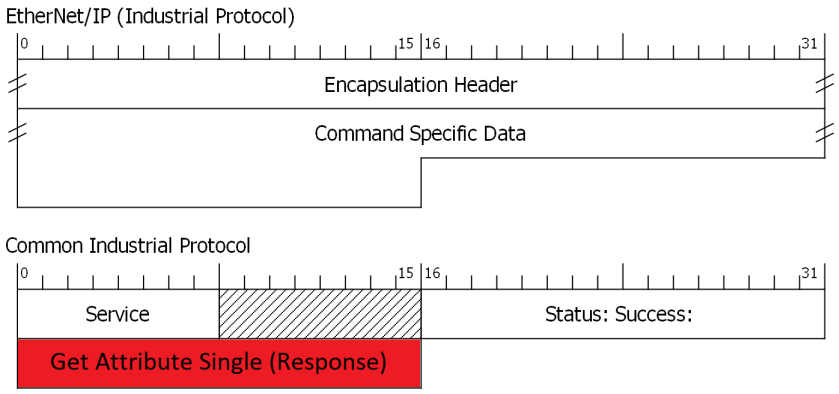

Figure 22: EtherNet/IP and CIP layer for Get A Single Attribute

TCP **Sequence number** field will be set to the request's Acknowledgement number. TCP **Acknowledgement number** field will be obtained by incrementing the request's sequence number by 48 which is the TCP segment length of the request packet. Fields in the EtherNet/IP layer stay the same.

However, the last field in the CIP protocol changes throughout different responses. This is because this service was the retrieval of the contents of the specified attribute. In new sessions this is always a 16-bit number with all zeros. In sessions that had prior modifications to attributes, this will be different. The emulator must, therefore, also store the values of the attributes in case a request for setting a single attribute is sent beforehand in the same session.

6.4.5. Unregistering Session and Connection Termination

The client will send a request for unregistering session through ENIP. No response to this is needed from the server. The parties will now move onto terminating their connection through TCP. The client will send a FIN request (TCP segment with the FIN bit set to 1). This requires a FIN, ACK response that the emulator needs to generate. The packet diagram was already shown in Figure 19.

TCP **Sequence number** field will be set to the request's Acknowledgement number. TCP **Acknowledgement number** field will be set to the request's sequence number.

6.5. Development of prototype

Development of the prototype was done in the programming language C# and .NET framework and utilized the library Pcap.NET. The first part of the implementation creates a packet sniffing program that will listen to and receive all packets on a specified interface/device. Once a packet has been received the program sends the packet to a handler function that determines what type of request it is and if a response needs to generated. The handler also serves as a filter to responses generated by the program itself (since those will be sent through the same interface). This process is demonstrated by the pseudocode in Algorithm 1.

Algorithm 1 Network capture and filtering

```
device = findNetworkInterface
if device == null then
    exit
end if
startListening
while packetReceived do
    if packet.Tcp.IsSyn and !packet.Tcp.IsAck then
        generateEstablishConnectionResponse(request)
    else if packet.Ethernet.Command == 0x0065 and packet.Ethernet.Command ==
0x00000000 then
        generateRegisterSessionResponse(request)
    else if packet.Cip.Service == 0x10 then
        generateSetAttributeResponse(request)
    else if packet.Cip.Service == 0x0e then
        generateGetAttributeResponse(request)
    else if packet.Tcp.IsFin and packet.Tcp.IsAck then
        generateTerminationConnectionResponse(request)
    else
        continue
    end if
end while
```

At first the program attempts to find the relevant network interface by storing it in an instance of LivePacketDevice. It then uses the PacketCommunicator to open a communication with the interface and calls the method ReceivePackets, which starts the process of capturing traffic. The method takes 2 parameters: one which indicates the number of packets to receive and a handler function. The received packet gets forwarded to the handler function which performs the filtering and calls upon the second part of the implementation which is generation of responses. In the pseudocode in Algorithm 2 a generalized version of the generation is shown. The process of generation of responses was described in detail in Section 6.4.

Algorithm 2 Response Generation

```
oldResponse = importRecording(fileName)
response.Ethernet = oldResponse.Ethernet
response.IpV4 = oldResponse.IpV4
response.IpV4.Identification = generateRandom()
response.Tcp = oldResponse.Tcp
response.Tcp.Source = request.Tcp.Destination
response.Tcp.Destination = request.Tcp.Source
response.Tcp.Sequence = calculateSequence(request)
response.Tcp.Acknowledgement = calculateAcknowledgement(request)
Build(response)
Send(response)
```

The generation of responses first needs to handle import of a historic capture. This is done by creating an instance of OfflinePacketDevice and using the PacketCommunicator to invoke the method ReceivePacket and store the packet in an instance of the class Packet. The program then proceeds to build the new response. It starts with the building of the Ethernet layer, proceeded by the IpV4 layer and TCP layer. It does so by utilizing the library's classes EthernetLayer, IpV4Layer and TcpLayer. Throughout the build, the request packet and the historic capture packet will be utilized to set all the fields in the layers to the correct values. Again, this process is described in detail in Section 6.4. Finally, once all layers are built, the response is sent into the network by invoking the SendPacket method of the PacketCommunicator. More details into the classes and methods can be found on Pcap.NET's wiki.

7. Experimental Validation of the Solution

This section will cover the experimental validation of the implemented solution based on the proposed model in Section 6.1.

First, validation of generated responses will be performed by playing requests individually and assessing if the packets were generated and assembled correctly. This in turn will also serve as a validation to request recognition. The requests will be taken from previously captured sessions and the response packets will be compared through Wireshark in hex format. Additionally, a console output in the solution will be generated as an additional step to validation.

Secondly, the performance of the solution will be assessed. Each request will be iterated 100 times and an average response time will be calculated. This will be compared to the non-emulation response time and implications will be discussed. The aim of this is to assess performance of test cases that would be run using this emulation.

Finally, an experimental validation of an industrial use case will be performed. The software in the abstraction layer will be run in parallel to the emulation. A real client-server connection will be opened and the network traffic will be observed to determine whether the emulation responds correctly to client requests. This will help to simulate the behaviour of the emulation in real world applications.

7.1. Validating Generated Responses

The aim in this validation is to evaluate if the emulation handles request packets and generates response packets properly. Five scenarios will be performed - one for each of the different requests that the emulation would handle in a real life setting. Five requests will be manually sent to the interface that the emulation is listening to. The requests are captures from real communications between the I/O device and PC agent. The capture also contains the responses to these requests and their contents will be compared to the contents of the packets generated subsequently by the emulation. It is also important to note that these captures are not the same captures that the solution uses when generating responses. The comparison will be done through Wireshark but, for the purposes of the validation, a console output will be generated indicating what type of request was received and the contents of the generated response will be written.

Scenario 1 is validating the connection establishment response. The results are shown in Figure 23 which presents the capture response (on the left) and the emulated response (on the right), and Figure 24 that shows the Console output.

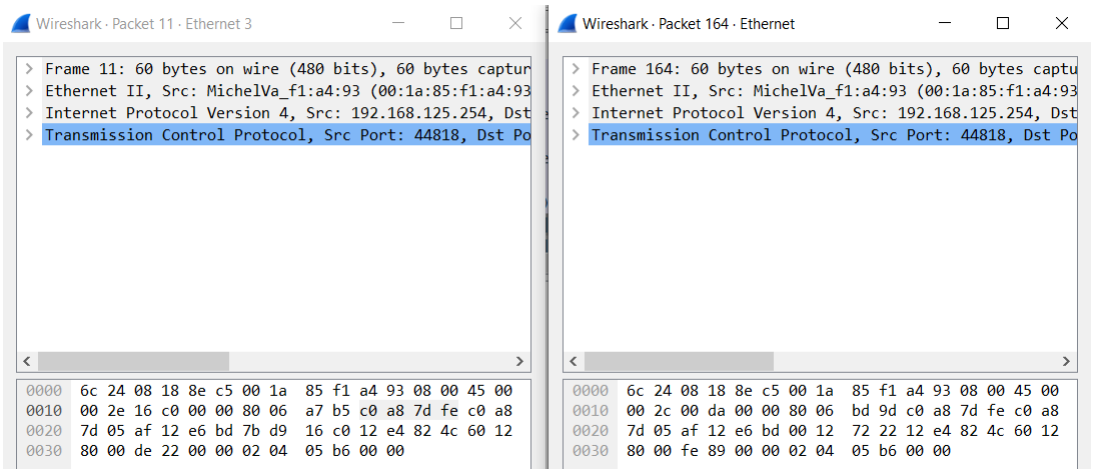

Figure 23: Comparison of packet contents for Scenario 1. Captured response (on the left) and emulated response (on the right)

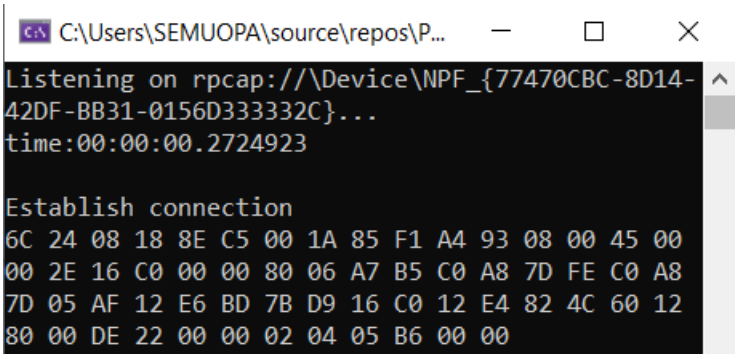

Figure 24: Console output of Scenario 1

The emulated response matches the captured response in all places except where it shouldn't. Identification field, Header Checksum and TCP Checksum are unique to each packet and so therefore the emulated response differs in those fields, which is expected behaviour.

Scenario 2 is validating registering session response. The results are shown in Figure 25 and 26. Again, the emulated response matches in all fields except where it doesn't need to.

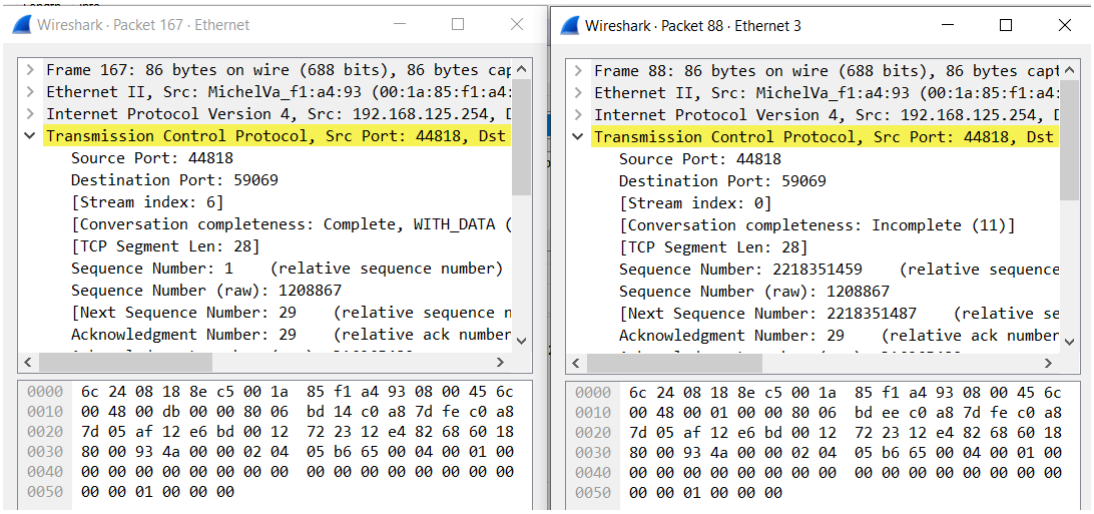

Figure 25: Comparison of packet contents for Scenario 2. Captured response (on the left) and emulated response (on the right)

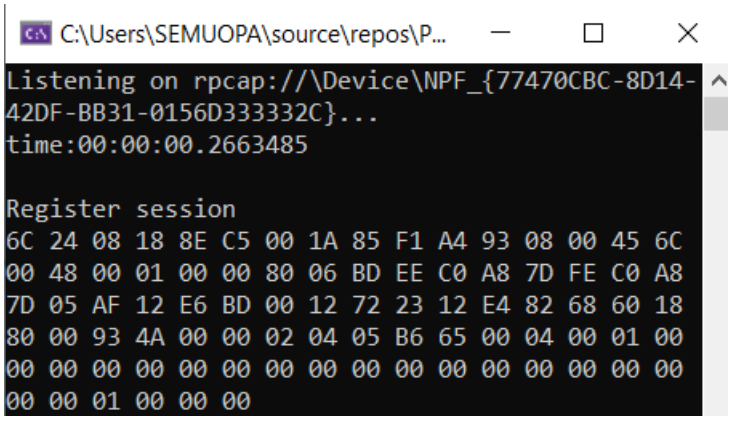

Figure 26: Console output of Scenario 2

To avoid repetition, the following scenarios will show comparison of Wireshark capture response with output console. Scenarios 3 and 4 are validating responses for setting a single attribute and getting a single attribute respectively. The results shown in Figure 27 and 28 demonstrate passed validation.

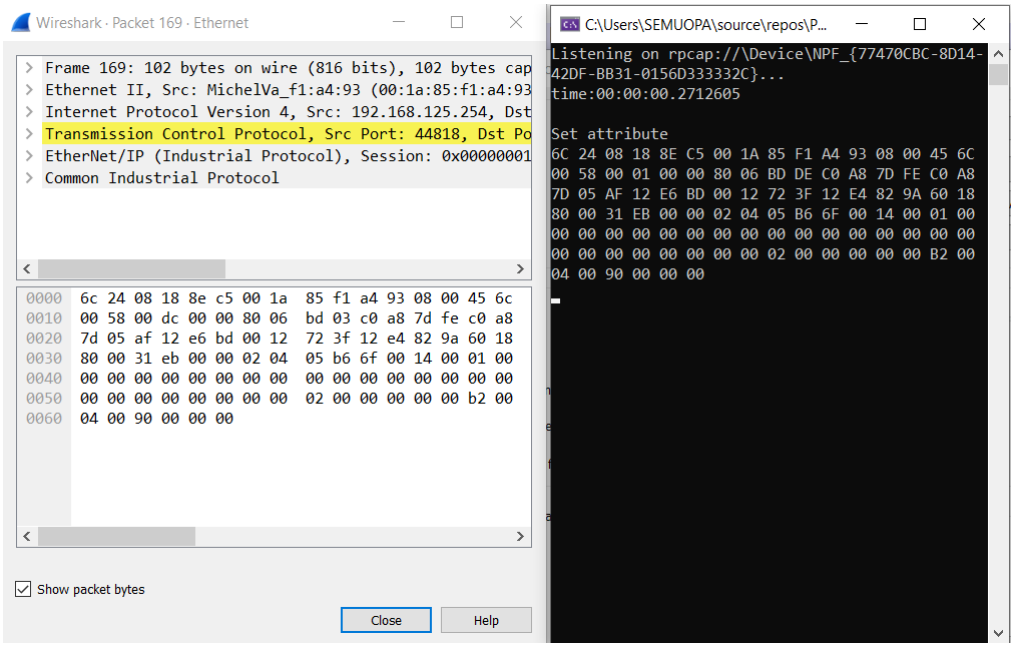

Figure 27: Scenario 3 - Comparison of Wireshark capture response with console output of emulation

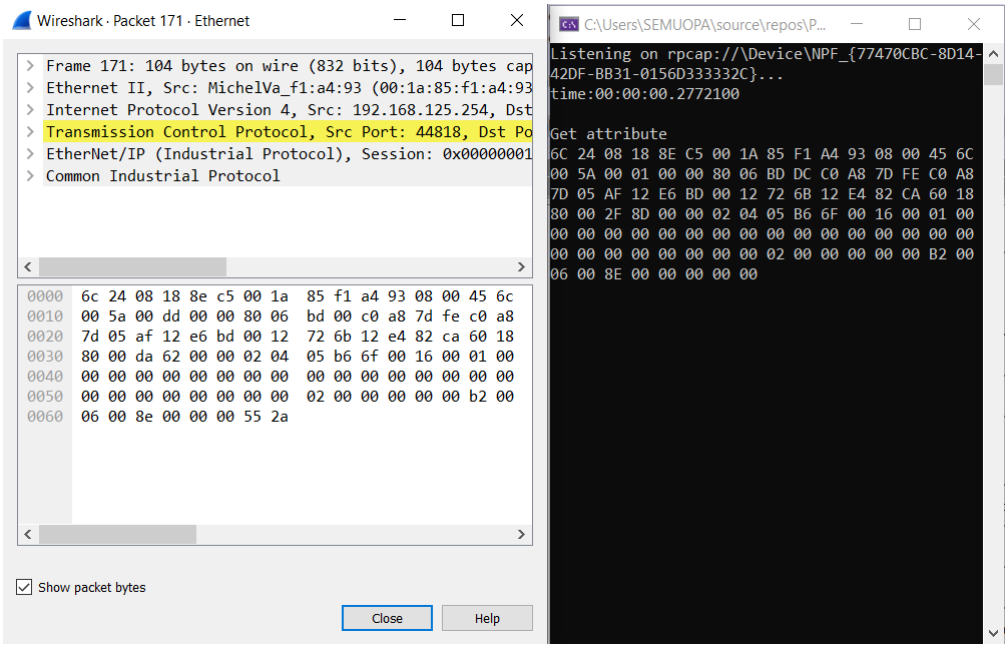

Figure 28: Scenario 4 - Comparison of Wireshark capture response with console output of emulation

Finally, in scenario 5, validation of terminate connection response is performed. Results are shown in Figure 29.

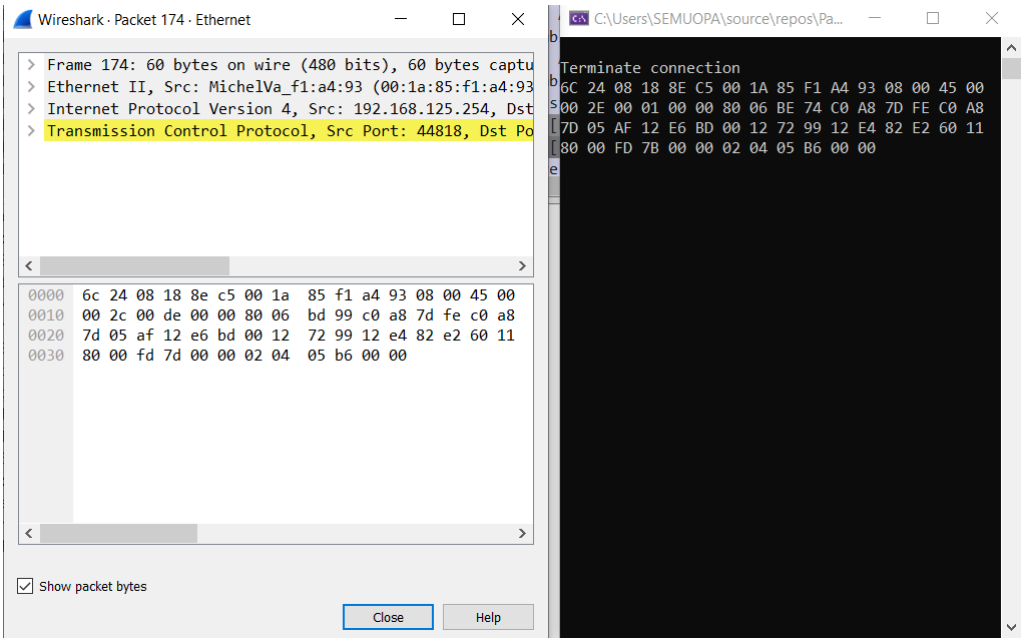

Figure 29: Scenario 5 - Comparison of Wireshark capture response with console output of emulation

7.2. Validating Performance of Emulation

To answer RQ2, a validation of emulation performance is done. The approach is quite simple: measure the time it takes for the emulation to generate a response and compare it to the current response time. The outcome of this validation helps to present possible behaviour of the emulation during real-life application (running in the background of test cases). To get the response time, each request that the emulation is expected to handle was sent over the interface 100 times and the time was observed. The results are shown in Table 2 and 3, the first showing emulated response times and the latter real response times.

	Connection establishment	Registering session	Setting Attribute	Getting Attribute	Connection termination
Minimum	0.0259	0.0259	0.0248	0.0204	0.0201
Mean	0.0296	0.0275	0.0266	0.0245	0.0246
Median	0.0297	0.0275	0.0265	0.0245	0.0249
Maximum	0.0335	0.0291	0.0286	0.0288	0.0284
Standard deviation	0.0022	0.0009	0.0011	0.0026	0.0025

Table 2: Emulated response time metrics for different requests (in seconds)

	Connection establishment	Registering session	Setting Attribute	Getting Attribute	Connection termination
Minimum	0.000121	0.000143	0.000133	0.000176	0.000101
Mean	0.000261	0.000237	0.000260	0.000255	0.000162
Median	0.000268	0.000247	0.000268	0.000259	0.000166
Maximum	0.000391	0.000323	0.000390	0.000326	0.000211
Standard deviation	0.0000752	0.0000565	0.0000789	0.0000460	0.0000323

Table 3: Real response time metrics for different requests (in seconds)

The results indicate that the response time across all metrics increased in the emulated setting. While it is justifiable for generation and building of packets to take more time, this is still a big

increase in response time. This could lead to latency in responses and connections could timeout in cases where the emulation is flooded with requests, which is to be expected in real life application where the robot controller and PC are connected to hundreds of I/O devices.

Finally, table 4 presents the comparison of response time metrics across all responses (not divided by type of response). The results here, again, show significant increase which could have major implications in real world application.

	Emulated response	Real response
Minimum	0.0201	0.0002
Mean	0.0266	0.0002
Median	0.0268	0.0001
Maximum	0.0335	0.0003
Standard deviation	0.0027	0.00007

Table 4: Comparison of overall response time in the real and emulated setting

7.3. Experimental Validation of an Industrial Use Case

In this section, the emulation solution will be run through a simple use case scenario. The scenario depicts the use of the communication between the I/O devices and PC agent, as it would in real test cases. The abstraction layer software will send a request to set an input in the I/O device to 1. A real client-server connection will be opened even though no physical connection to the I/O device exists.

Network traffic was observed through Wireshark and it is shown on Figure 30.

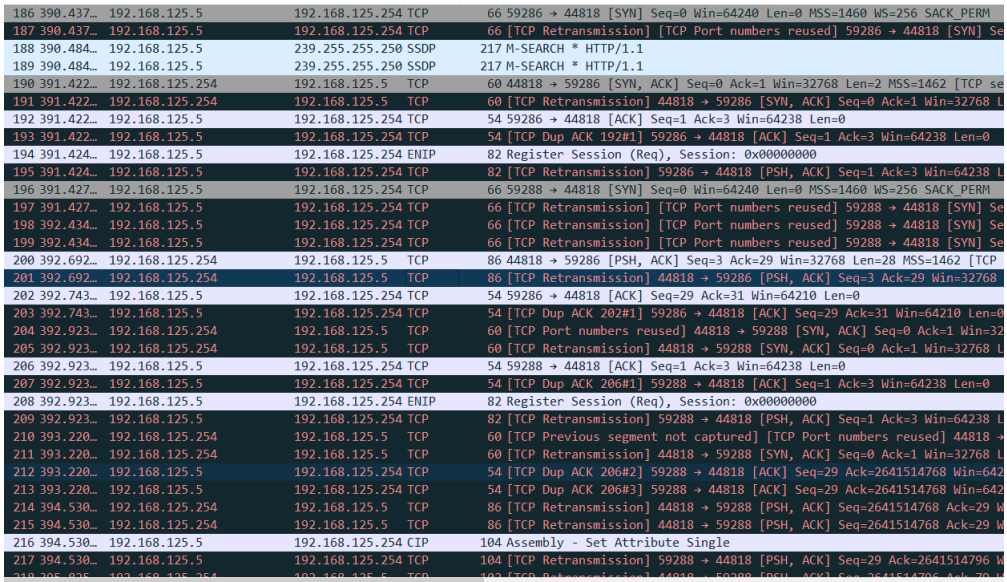

Figure 30: Network capture during experimental validation

As seen in the Figure, the network communication did not flow as expected. Multiple TCP retransmissions occurred and the emulator didn't respond to many requests as quickly as needed. However, timeout for connection establishment and session registration are not as strict and so the emulator managed to establish a connection with the client and register the session. However, the program crashed once the inputs were needed to be set on the device. The emulation generated the response but not before the timeout was reached.

Therefore, the solution does not pass this validation and it can't be implemented in a real-world setting yet. Causes to this communication flow can be attributed to the latency in sending responses, as demonstrated in the previous section. However, more research needs to be performed.

8. Discussion

This section focuses on the overview of summarizing the outcomes achieved in the work through implementation and evaluation results, in addition to its limitations encountered with the process.

8.1. Research Questions

1. **RQ1**. *What existing approaches, if any, can be used to emulate network communication and behaviour of I/O devices inside robot cells in a virtual environment?*

   The concept of I/O device emulation in the context of robotic systems has not been explored yet in literature. However, network virtualization and emulation of networks have been researched extensively in recent years. Network traffic capture, modification and replay are becoming more popular especially in the context of network testing. Putting these findings into the industrial context, it became clear that the solution had to combine all of them to get the necessary outcome. Therefore, these approaches (network capture, modification and replay) can be utilized for the emulation of the network communication between I/O devices and other parties in the robot cell. An approach that combines real-time network capture, its analysis and filtering, further generation of new network traffic based on earlier capture and subsequent replay of modified traffic can be used to emulate network traffic and network communication of I/O devices inside robot cells.

2. **RQ2**. *Does the approach to emulating I/O device behaviour impact performance of test cases that are run on robot cells?*

   A model based on the approaches found in RQ1 was proposed and a prototype was developed. The solution to the emulation was validated in Section 7.. Experimental results found that the solution does well in replicating responses and their contents match how they would be constructed in real life. However, the results also showed that the client-server communication became up to 100 times slower due to a delay in the emulated responses. This is mostly caused by the generation and building of new packets as that takes up the majority of the solutions execution time. Therefore, it is safe to assume that this approach would lead to, at the very least, slower execution of test cases if it was ever implemented in a real world setting.

8.2. Limitations

As demonstrated in Section 7., the proposed solution has limitations. The biggest limitation is the big delay in sending response packets to the interface due to the expensive process of building new packets. As a consequence, this can lead to unexpected client behaviour, continuous sending of new requests (each of which gets the response late) which can in turn overwhelm the emulation and cause bigger delays, creating a domino effect. This was also proven to be true during the experimental validation of an industrial use case in section 7.3. In fact, the network traffic didn't resemble the flow of traffic before emulation, precisely due to response times.

Setting response times aside, another limitation of the solution is that it currently only emulates the communication of the I/O device with just the digital base. In real settings, add-ons are frequently used when more inputs are needed. Network traffic, requests and responses might be completely different for the devices with add-ons. Apart from that, this approach only analyzed communication of one I/O device and didn't consider the real extent of all the network traffic in a robot cell. Taking these into account, the solution would have to be optimized to handle heavy traffic, which might not be possible due to current technological limitations.

Another limitation of the current solution is that it might not handle unexpected or erroneous situations, such as incorrect inputs, invalid data, or exceptional conditions. The emulation will not respond to packets that don't conform to the the packets outlined in Section 6.. However, currently it is not known whether slight changes to the packets could lead to problems, whether that is simply taking up unnecessary execution time or breaking a communication flow between

devices. Further testing and validation need to be done to ensure adequate error handling and enhance the overall quality and resilience of the software.

8.3. Contributions

With all the limitations taken into account, this thesis work provides a significant academic contribution. During the literature review it was established that not much research into I/O device and network emulation was made, and almost none in the field of robotics. Publications that were found had its own limitations and problems, making them inapplicable in the context of this thesis work, indicating the thematic is still in early stages of research. However, network traffic virtualization and seamless replay is gaining traction due to increased interest in developing intrusion detection systems.

Furthermore, no literature was found on models that combine network capture, packet recognition, modification and generation of new packets, and replay. This thesis work can provide insight into how such models and solutions based on them can be created, allowing for the improvement and elimination of current limitations.

On the other hand, in a practical sense, the work done in the thesis has the potential to solve many problems in the industry. Robotic systems are increasing in complexity and their dependence on hardware systems make it harder to effectively test the software. The thesis demonstrates this in Section 5., where research into the current test process in the company ABB Robotics showed that testing relies heavily on the robot cell setup which cannot be replicated anywhere and everywhere. Therefore, the research and validation done in this project is a step in the right direction towards solving these limitations in the industry.

9. Conclusions and Future Work

Testing robot controller software is becoming increasingly harder due to the rise in robotic system complexity. With the rise in complexity, the number of hardware systems that the robot depends on also increases. This can present a barrier in testing of robot controllers which is vital to ensure that robots behave safely and effectively in their intended applications. The use of simulation as a platform for software testing of robots is now possible, but in test cases that depend on input and output signals sent to robot controllers, simulating everything in a virtual environment isn't achievable due to required physical connections to hardware. Therefore, this thesis work aimed to emulate I/O device network communication to enable further advancements in virtual testing processes. The approach used for this was a method that combines real-time network traffic capture, its modification and rebuild, as well as subsequent replay.

The work presented current academic research into these approaches and technologies and the current challenges in the testing process were further elaborated by performing research in a company leading in industrial robot development. A model concept was proposed and a prototype was developed. The solution showed potential to solving the current challenges in robot controller testing because it is capable of network capture, modification and replay on network layers 4. However, experimental results showed many limitations, i.e. many delays in response generation and further research and development is needed.

The work can be expanded upon in many areas. Firstly, the work can be optimized by looking into approaches to reduce or remove the current limitations. Research into reducing the execution time of packet building or approaches to utilizing pre-built packets can be made. Furthermore, packet recognition and response generation doesn't need to rely on patterns observed during capture, but rather artificial intelligence and machine learning approaches can be utilized for this purpose.

References

[1] "OSHA Technical Manual (OTM) - Section IV: Chapter 4 | Occupational Safety and Health Administration." [Online]. Available: https://www.osha.gov/otm/section-4-safety-hazards/chapter-4

[2] [Online]. Available: https://notes.shichao.io/tcpv1/ch13/

[3] J. Thornton, "The definitive guide to traffic replay," Apr 2023. [Online]. Available: https://speedscale.com/blog/definitive-guide-to-traffic-replay

[4] J. F. Nunamaker, M. Chen, and T. D. M. Purdin, "Systems development in information systems research," *Journal of Management Information Systems*, vol. 7, no. 3, pp. 89–106, 1990. [Online]. Available: http://www.jstor.org/stable/40397957

[5] Wikipedia, "Osi model — wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=OSI%20model&oldid=1146735354, 2023, [Online; accessed 30-March-2023].

[6] A. K. Gupta, S. K. Arora, and J. R. Westcott, *Industrial Automation and Robotics: An Introduction*, Jan 2017.

[7] J. Johannessen, *Robot Ethics and the Innovation Economy*, ser. Routledge Studies in the Economics of Innovation. Taylor & Francis, 2021. [Online]. Available: https://books.google.se/books?id=G0crEAAAQBAJ

[8] D. Brugali, *Software Product Line Engineering for Robotics*. Cham: Springer International Publishing, 2021, pp. 1–28. [Online]. Available: https://doi.org/10.1007/978-3-030-66494-7_1

[9] A. Afzal, C. L. Goues, M. Hilton, and C. S. Timperley, "A study on challenges of testing robotic systems," *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pp. 96–107, 2020.

[10] "Input/output module – 21071," Mar 2013. [Online]. Available: https://www.robotpark.com/academy/inputoutput-module-21071/

[11] S. Andersson and G. Carlstedt, "Automated testing of robotic systems in simulated environments," Master's thesis, Mälardalen University, School of Innovation, Design and Engineering, 2019.

[12] Wikipedia, "Industrial robot — wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Industrial%20robot&oldid=1135030203, 2023, [Online; accessed 12-February-2023].

[13] ——, "Emulator — wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Emulator&oldid=1146638885, 2023, [Online; accessed 07-May-2023].

[14] W. b. V. Fulber-Garcia, "Differences between simulation and emulation," Dec 2022. [Online]. Available: https://www.baeldung.com/cs/simulation-vs-emulation

[15] "Iso/iec 7498-1:1994 — information technology — open systems interconnection — basic reference model: The basic model," https://www.iso.org/standard/20269.html, 1999, [Online; accessed 30-March-2023].

[16] Mar 2014. [Online]. Available: https://en.wikipedia.org/wiki/Internet_protocol_suite

[17] M. E. Shacklett, A. Novotny, and K. Gerwig, "Tcp/ip: What is tcp/ip and how does it work?" Jul 2021, [Online; accessed 30-March-2023]. [Online]. Available: https://www.techtarget.com/searchnetworking/definition/TCP-IP

[18] P. Loshin, *TCP/IP Clearly Explained*. Morgan Kaufmann, Jan 2003.

[19] Wikipedia, "Transmission control protocol — wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Transmission%20Control%20Protocol&oldid=1152643298, 2023, [Online; accessed 01-May-2023].

[20] ——, "Internet protocol version 4 — wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Internet%20Protocol%20version%204&oldid=1148691723, 2023, [Online; accessed 01-May-2023].

[21] "Ethernet/ip protocol overview - real time automation," Jan 2023, [Online; accessed 30-March-2023]. [Online]. Available: https://www.rtautomation.com/technologies/ethernetip/

[22] Y. García Concejero and M. A. Salazar del Río, "Emulation of industrial fieldbus modules for virtual commissioning," Ph.D. dissertation, 2019. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-17486

[23] [Online]. Available: https://www.wireshark.org/docs/wsug_html_chunked/

[24] J. Forshaw, *Attacking Network Protocols: A Hacker's Guide to Capture, Analysis, and Exploitation*, Dec 2017.

[25] F. Aghili and J.-C. Piedboeaeuf, "Emulation of robots interacting with environment," *IEEE/ASME Transactions on Mechatronics*, vol. 11, no. 1, pp. 35–46, 2006.

[26] F. Aghili and J.-C. Piedboeuf, "Contact dynamics emulation for hardware-in-loop simulation of robots interacting with environment," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 1, 2002, pp. 523–529 vol.1.

[27] A. M. Ghalamzan E. and M. Ragaglia, "Robot learning from demonstrations: Emulation learning in environments with moving obstacles," *Robotics and Autonomous Systems*, vol. 101, pp. 45–56, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889017302981

[28] H. B. Huang, "Communication between virtual emulation system and plc by modbus/tcp protocol," Ph.D. dissertation, 2015, copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2023-03-04. [Online]. Available: http://ep.bib.mdh.se/login?url=https://www.proquest.com/dissertations-theses/communication-between-virtual-emulation-system/docview/1896119014/se-2

[29] K. Maeda, M. Ishino, E. Kohno, and T. Kishida, "A userland network emulator with packet capture and replay," in *2007 International Symposium on Applications and the Internet*, 2007, pp. 13–13.

[30] A. Cheng, "Pac-gan: Packet generation of network traffic using generative adversarial networks," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2019, pp. 0728–0734.

[31] J. Matoušek and P. Korček, "Precise ipv4/ipv6 packet generator based on netcope platform," in *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2011, pp. 319–324.

[32] A. Nicolae, L. Gheorghe, M. Carabas, N. Tapus, and C.-L. Duta, "Lldp packet generator," in *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, 2015, pp. 7–11.

[33] D. Baba, N. Suematsu, S. Nabeshima, and T. Miyazaki, "Resne: Reconfigurable sensor network emulator virtualizing integrated large-scale sensor network," in *2016 International Conference on Computing, Networking and Communications (ICNC)*, 2016, pp. 1–5.

[34] O. Niculaescu, M. Carabas, L. Gheorghe, R. Rughinis, and N. Tapus, "Graphical packet generator - a solution for automatic discovery of the network topology and packet generation," in *2014 RoEduNet Conference 13th Edition: Networking in Education and Research Joint Event RENAM 8th Conference*, 2014, pp. 1–6.

[35] H. Wu, H. Liu, B. Wang, and G. Xin, "Accurate traffic replay based on interactive sequence and timestamp," in *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, 2017, pp. 1107–1110.

[36] D. Pham Van, M. Zhanikeev, and Y. Tanaka, "Effective high speed traffic replay based on ip space," in *2009 11th International Conference on Advanced Communication Technology*, vol. 01, 2009, pp. 151–156.

[37] S. Qiao, C. Xu, L. Xie, J. Yang, C. Hu, X. Guan, and J. Zou, "Network recorder and player: Fpga-based network traffic capture and replay," in *2014 International Conference on Field-Programmable Technology (FPT)*, 2014, pp. 342–345.

[38] M. Toll, I. Behnke, and O. Kao, "Iotreeplay: Synchronous distributed traffic replay in iot environments," in *2022 IEEE International Conference on Cloud Engineering (IC2E)*, 2022, pp. 8–14.