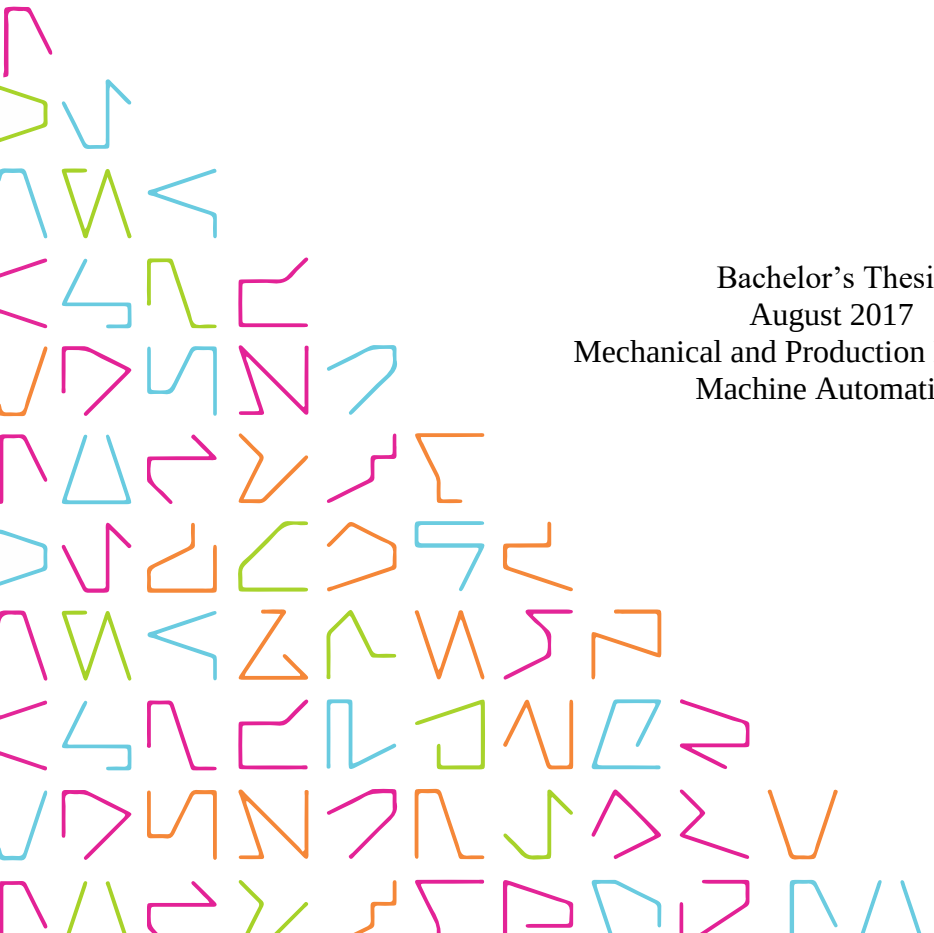
TAMPEREEN
AMMATTIKORKEAKOULU

PLC Virtualization and Software Defined Architectures in Industrial Control Systems

Mikael Peltonen



Bachelor's Thesis
August 2017
Mechanical and Production Engineering
Machine Automation

ACKNOWLEDGEMENTS

The completion of this thesis work could not have been possible without great support of colleagues and fellow researchers, who have instructed me and helped me through steps where my know-how has not been quite enough to proceed. That's why I'd like to express my thanks to people, who have assisted me with my thesis work.

The thesis work was accomplished at the Machine Solutions headquarters of Schneider Electric in Marktheidenfeld, Germany. I'm supremely grateful for Schneider Electric of the chance to work on my thesis in the company.

My special thanks to my supervisor Omid Givehchi, who's expertise on IT has been truly valuable. His case study [95] gave me a good overview on the topic of my research in the beginning of my thesis work.

I would also like to express my very great appreciation to Cedric Vandendriessche, who did a similar research on the topic in the Ghent University [68]. This thesis would have perhaps had a bit different approach without the substantial support of Vandendriessche, who provided me significant assistance on the topic.

Support provided by my colleagues Jyri Niinikoski, Alexander Kempf, Franz-Josef Krebbers and Henry Krüger was greatly appreciated as well.

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Mechanical and Production Engineering
Machine Automation

PELTONEN, MIKAEL
PLC Virtualization and Software Defined Architectures in Industrial Control Systems

Bachelor's thesis 53 pages, appendices 3 pages
August 2017

Today's automation systems are going through a transition called Industry 4.0, referring to the Fourth Industrial Revolution. New concepts, such as cyber-physical systems, microservices and Smart Factory are introduced. This brings up the question of how some of these new technologies can be utilized in Industrial Control Systems. Machines and production lines are nowadays controlled by hardware PLCs and this is considered as a state-of-the-art solution. However, the market demands are continuously increasing and pushing the industry e.g. to lower the operational costs and to develop more agile solutions. Industry 4.0 provides promising approaches to take a step forward and consider PLC virtualization.

The purpose of this thesis was to evaluate PLC virtualization possibilities using different Software Defined Architectures. Requirements and benefits of different solutions were evaluated. The major objective of the case study was to compare container- and hypervisor-based virtualization solutions using Docker and KVM.

The case study provides a modular and scalable IIoT solution in which a virtual PLC takes over the control instead of a hardware PLC. Node-RED was used as a runtime environment and an I/O-module was needed to set up a control loop test. Response time of the control loop was measured by capturing Modbus traffic with tcpdump. Multiple iterations were performed to show minimum, maximum, average, median and 90^{th} pctl. latencies.

The results indicate that the container-based solution has a smaller overhead than the hypervisor-based solution and it has a very little overhead in general. Peak latencies are a concern and even the average latencies show that this solution would not be suitable for any hard real-time or safety-related applications.

Further investigation on the topic would be needed to estimate the actual potential of PLC virtualization on hard real-time applications. First of all, a more powerful hardware PC would be needed to perform such tests. Secondly, a faster industrial protocol than Modbus TCP/IP would be required. Perhaps another kind of approach would be needed to overcome the issues that were experienced in this case study. It would be interesting to test a direct communication between virtual PLC and I/O and use Node-RED nodes for example to trigger inputs. Anyhow, it seems that container-based solution is holding much promise as a virtualization approach.

Key words: IIoT, Industry 4.0, PLC, virtualization, hypervisor, container

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ALC	App Logic Controller
CAPEX	Capital expense
CPS	Cyber-physical system
DCS	Distributed control system
DevOps	Software **dev**elopment & information technology **op**eration**s**
ERP	Enterprise Resource Planning
HMI	Human Machine Interface
ICS	Industrial Control System
IIoT	Industrial Internet of Things
IT	Information Technology
KVM	Kernel-based Virtual Machine
MES	Manufacturing execution system
NFV	Network Functions Virtualization
OPEX	Operational expense
OS	Operating System
OT	Operational Technology
(v) PLC	(virtual) Programmable Logic Controller
QEMU	Quick Emulator
RTOS	Real-Time Operating System
SCADA	Supervisory Control and Data Acquisition
SDA	Software Defined Architecture
SDN	Software Defined Networking
VM	Virtual Machine
VMM	Virtual Machine Monitor

1 Introduction

The First Industrial revolution took its place from the 18th to 19th centuries, when new manufacturing processes were invented [4]. Back then manufacturing was mainly dependent on items produced by hand, but innovations in Great Britain enabled manufacturing by machines and better usage of water and steam power. These innovations gave a burst to manufacturing, providing advantages like reducing production time significantly. Innovation is the key to improve one's business and innovation had naturally a big role in the second and third industrial revolutions as well. The ongoing industrial revolution is the Fourth Industrial Revolution, also called as Industry 4.0. The basic idea of Industry 4.0 is the same than in other industrial revolutions: to improve one's business e.g. by reducing production time, lower the costs of the production materials, reduce the amount of product and manufacturing defects and make the jobs easier for humans by creating machines which can do the jobs on our behalf.

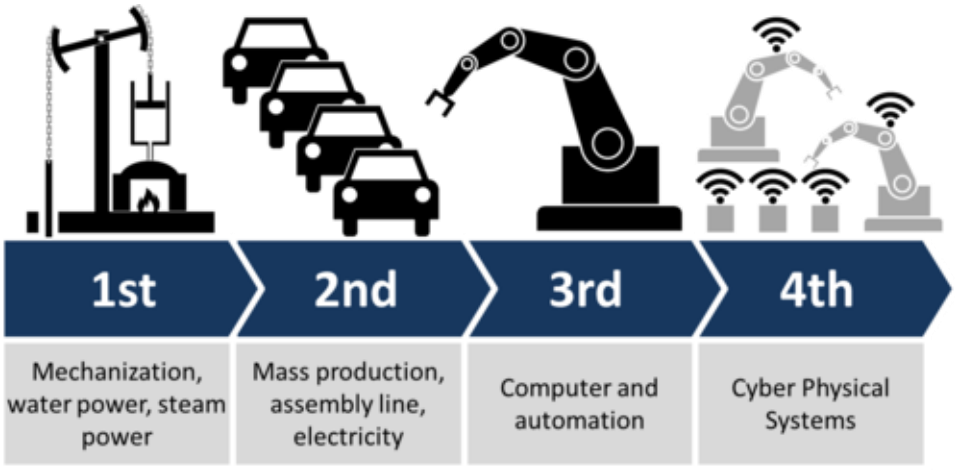


Figure 1. Industrial revolutions [5]

Industry 4.0 is the term for the ongoing industrial revolution. It originally refers to the digitization of manufacturing, but in fact it also refers to the digitization of other industries like healthcare, logistics and oil and gas [3]. Often you hear about concepts like smart factory, smart city or smart device. Industry 4.0 is about the Internet of Things (IoT), cyber-physical systems (CPSs), convergence of Information Technology (IT) and Operation Technology (OT), cloud computing, machine learning and many other technologies [5]. These technologies lead companies to new business models – others are willing to take risks by deploying new technologies in an early phase, whereas others might be more

careful doing this. To overcome the threshold the industry needs innovations, lots of research and several implementations.

To put it simply, Internet of Things is about connecting ”things” to the Internet. As an example, you could use sensors to gather data from your water bottle, transfer the gathered data to the cloud and access the data from a web browser. One might think, what would we do with this data gathered from the water bottle? Naturally we don’t need to connect everything to the Internet, but there are lots of cases where we can benefit from the use of Industry 4.0 technologies. Especially the industry can benefit by using these technologies and in this case, we would talk about Industrial Internet of Things (IIoT). IIoT refers to the use of IoT technologies in manufacturing [6]. Smart factory applications, predictive and remote maintenance, industrial security systems and asset tracking are example use cases of IIoT [7]. The biggest challenge of IIoT is cybersecurity, but because that topic is a very complex topic of its own, it was out of scope of this thesis.

Since the late 1960s Programmable Logic Controllers (PLCs) are used in Industrial Control Systems (ICSs) to perform control functions. PLCs were developed in response to the complex machine control with electromechanical relays. The aim was to develop control systems more flexible, reduce machine downtime and perform logic functions with this new device [8]. PLCs have indeed made their way to the factories and today they are seen as state-of-the-art control devices. Since PLCs have been there for few decades, they have achieved the reliability to control machinery even in safety-critical applications.

Thanks to the IT innovations, there are now different ways to lower operational costs by use of virtualization (chapter 5.1) and cloud computing for example. The latest IT innovations are already state-of-the-art solutions in the office and enterprise world, but it isn’t as easy to deploy these technologies and solutions in the industrial applications, as the requirements are typically high and the consequences of a system failure might be critical. It’s questionable whether Industry 4.0 technologies are already so developed and reliable that they could be of use in ICSs in which high requirements of determinism and real-time computing have to be met. Anyways, manufacturers already benefit from Industry 4.0 technologies and the next step could be to try to virtualize the control plane and use software rather than physical hardware in order to lower operational costs and have more agile control environment. PLC virtualization, in other words using virtual PLCs (vPLCs) could be the next step.

2 Schneider Electric

Schneider Electric is a French company with more than 180 years of history. At the time of the Industrial Revolution in 1836, two brothers named Adolphe and Eugène Schneider acquired the Creusot mines, forges and foundries and two years later the brothers founded Schneider & Cie, focusing mainly on steel & heavy industry, railroads and ship building [1]. In 1999 the company changed its name to Schneider Electric to emphasize company's expertise. Nowadays Schneider Electric is a global specialist in energy management and automation with around 144 000 employees around the world [2].

Since 2000, Schneider Electric has its headquarters in Rueil-Malmaison, France. Currently Jean-Pascal Tricoire is the Chairman and Chief Executive Officer (CEO) of the company. The company's revenue in 2016 was 24.7 billion euros and 43% of the revenue came from Building-segment (figure 2). 30 percent of all the employees are women and thus Schneider Electric got CEO Leadership Award in 2015 by United Nations Women and at the same time the company was awarded as a Global Compact as Champion for Gender Equality. Green values are truly important for the company.

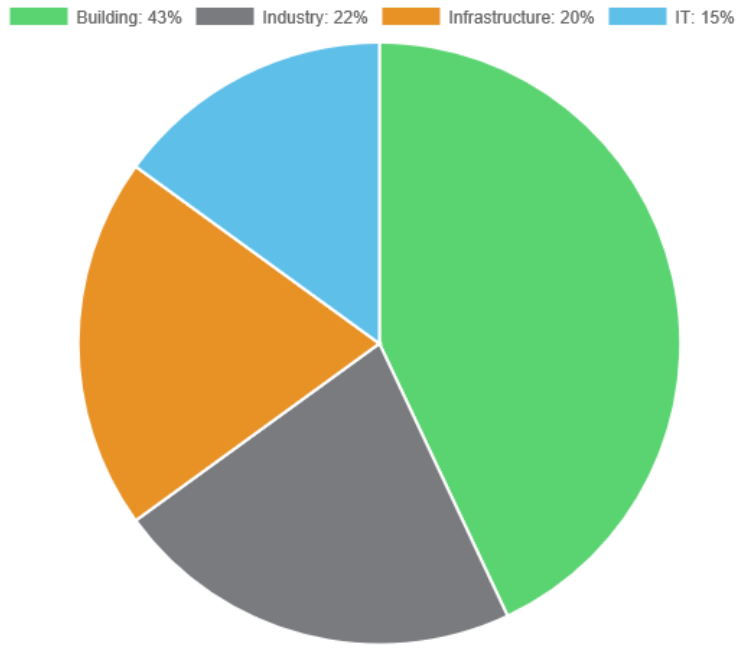


Figure 2. 2016 revenues by business [2]

2.1 Customers and market

Schneider Electric has a great variety of customers, starting from cities via all kind of industry companies till consumers. The company offers industry solutions and automation and electrical devices for discrete and process manufacturing. This thesis was accomplished in the Machine Solutions headquarters in Marktheidenfeld, Germany. Machine Solutions provides motion control and safety solutions and products for different customers globally, for example for hoisting, packaging, material handling or HVACR (heating, ventilating, air condition and refrigeration) applications.

In the future Schneider Electric is going to invest in cities. The company is trying to build urban cities with energy efficiency and infrastructure management. Smart grid, smart mobility, intelligent and green buildings and renewable energy sources are the keys. At industry side Schneider Electric is a pioneer and a leading technology developer for Industry 4.0.

Figure 3. Logo of Schneider Electric [2]

3 Software Defined Architectures

There's always room for enhancements and innovations in the industrial world. Before the birth of automation, systems and machines had to be operated manually. Automation has provided many advantages from reduced time-to-market to fewer failure products. Despite all this, development is required as market demands keep increasing. Today's issues could be overcome by the use of Industry 4.0 technologies. Flexibility is the key for tomorrow's industrial automation: more and more plant data should be available, codes should be easily movable and reusable, systems should be modular and scalable and industrial companies should have a possibility to select their preferable vendors [9]. Machines are turning into autonomous machines that will ease humans' tasks. Cyber-physical systems enable the communication and interaction between virtual and physical machines and systems [10].

So, the transition from Industry 3.0 to Industry 4.0 requires naturally some time, it doesn't happen overnight. Different technologies are developed, tested and implemented and at the same time something fancier is already brought to the market. In any case, the industrial architectures are going through a change towards software-based approach and we can therefore talk about Software Defined Architectures (SDA) due to developed IT technologies. Software Oriented Architectures (SOAs) came already into use in the beginning of the 21st century, but Software Defined SOA (=SDA) is a new style of architecture (figure 4). SOAs gave better architecture flexibility compared to the previous architecture models by providing services that run a small function [11]. SDA brings even more flexibility and scalability to the architectures by moving systems to the cloud. SDA means basically that server hardware and software are decoupled and the data processing is executed in a software.

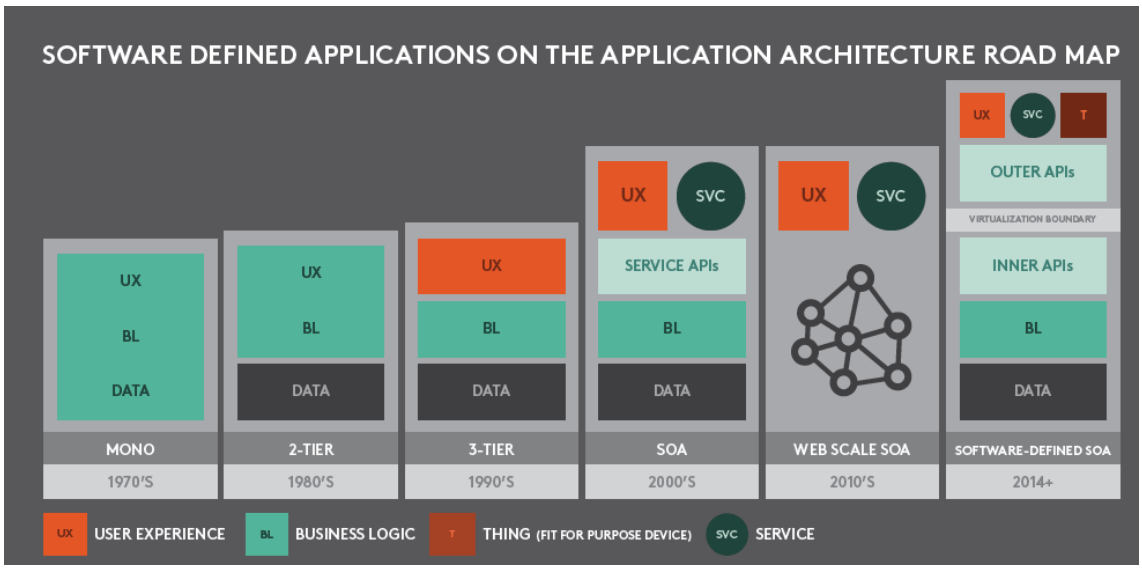


Figure 4. Software Defined Applications on the Application Architecture Road Map [14]

3.1 SDA requirements

SDAs have to meet the requirements of growing needs of the industrial world. One of the most important factors is reliability: if you can't rely on your system, you are risking your business. To acquire high level of reliability, the industry needs time to test and run new technologies. Real-time capabilities and deterministic behavior are on the table when talking about Industrial Control Systems (ICSs). SDAs also need to have a high availability, meaning that the system should be continuously available and can recover quickly from a system failure [13]. Cybersecurity (=IT security) is a big concern in general in Industry 4.0 and that is in fact a huge topic of its own, that's why it is out of scope of this thesis. Anyways cybersecurity is something that must be taken care of in a serious manner to ensure that no outsiders can access your sensitive data. SDAs should be easily scalable and modular to manage lifecycle operations of the components: hardware updates & replacement, software updates and network changes should be able to be done without any loss of service [9]. SDAs should also target to a more simplified architecture, enable isolation between components/processes when needed, enable communication between these components and processes and make software codes reusable to e.g. reduce time-to-market [13].

3.2 SDA benefits

Today's industry demands an architecture that is flexible for modifications. SDA enables easy scalability and system modularity by allowing you to replace or add components without affecting the rest of the system. SDAs are designed with open platforms that allow users to select preferred components and solutions which means that the users have the flexibility to choose between different vendors (=no vendor lock-in). In Industry 3.0 systems, it is not so easy – or maybe not possible at all – to use multiple vendors' components in the same architecture. There's typically no hardware dependence in SDA, so it's easy to migrate and reuse software [9]. SDAs use virtualization (chapter 5.1) and more software instead of hardware, which leads to cost reduction and smaller footprint because there's a smaller amount of required hardware. The aim is to provide COTS (COTS = commercial off-the-shelf) software / hardware products that are ready-made and available for sale [12].

Software Defined Architecture means Software-centric model, which gives an advantage in management, network processing and security when the system is centralized. Remote monitoring reduces OPEX while the maintenance engineer or the operator doesn't always need to be on site to check the status of machines. Centralized management eases the remote monitoring, because then it's enough to access only one software platform to manage your assets. By cloud computing and by use of smart sensors (sensors that include communication capability and on-board diagnostics [15]) machine data is pushed to the cloud and the data is then accessible on the user interface (HMI). The machine data can be used for predictive maintenance (figure 5), which means that the machine data can be used to estimate the time to a forthcoming failure of the machine. SDAs also support DevOps (Software **dev**elopment & information technology **op**eration**s)**, which is converging IT and OT. DevOps (figure 6) is a set of practices that improve collaboration between software developers and IT team and enables you to build, test and deploy software faster [17] [18] [19] [20]. Organizations can significantly reduce their software's time-to-market by use of DevOps practices, since it aims to continuous delivery and continuous deployment. This enables more frequent software releases.

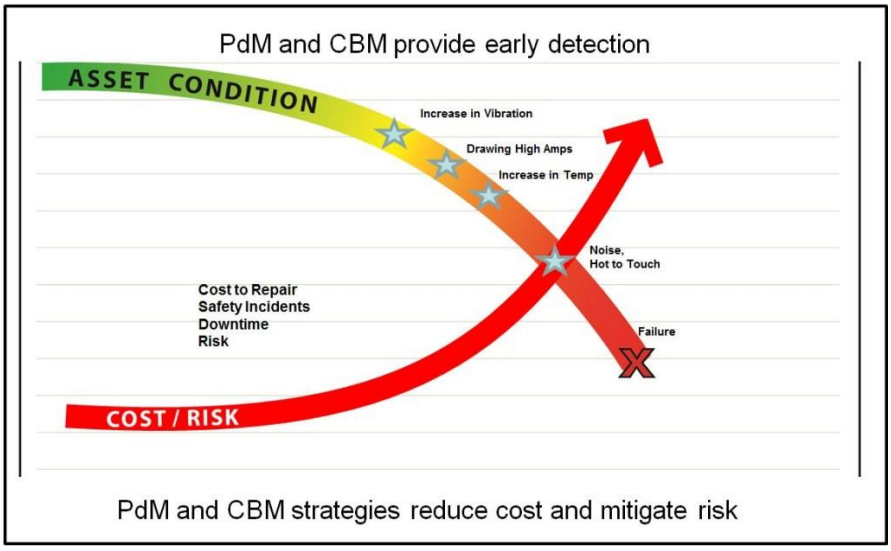


Figure 5. Predictive maintenance [16]

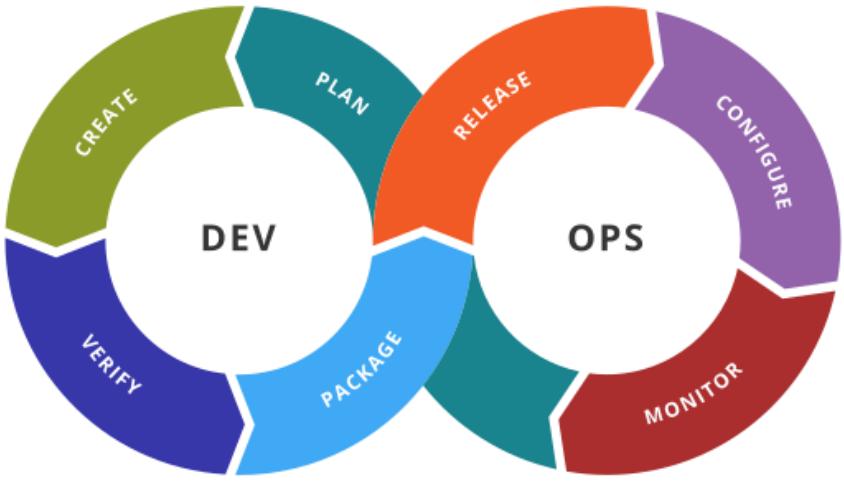


Figure 6. DevOps [20]

3.3 SDA technologies

So, Industry 4.0 is bringing Software Defined Architectures to organizations. These SDAs should meet at least the same reliability, safety and security requirements than the traditional architectures, but also enhance one's business by e.g. reducing time-to-market, hardware and maintenance costs and increase systems' flexibility, scalability and modularity. How does an organization implement SDAs? There are several different approaches available (introduced in the next chapter): hypervisor-based virtualization, container-based virtualization, Software Defined Networking (SDN), Network Functions Virtualization (NFV) and so on. This thesis is focusing on PLC virtualization, so we'll also have a look on an approach called App Logic Controller (ALC) (chapter 8).

4 SDA approaches

4.1 Virtualization

To jump on PLC virtualization, we need to understand what the term virtualization means. Virtualization is already widely used technology in IT, but it's gradually making its way to Industrial Automation. Hypervisor-based virtualization refers to a concept in which underlying, single hardware resources are shared by multiple guest operating systems [21]. This enables better server utilization, since CPU's are otherwise generally underutilized. Basically it means that you don't always need to purchase new hardware for every function, but you can make better use of already existing hardware by virtualizing operating systems and/or applications. By using virtualization one can benefit at least in energy savings (less required hardware) and system flexibility.

There are different types of virtualization methods, like e.g. storage virtualization, network virtualization, software virtualization and desktop virtualization. In this chapter following techniques of x86 computer virtualization are introduced:

-Full virtualization

-Paravirtualization

-Hardware assisted virtualization

In addition, following virtualization methods are explained:

-Operating System-level virtualization

-Embedded virtualization

Full virtualization means virtualization in which guest OS is completely abstracted from the hardware, therefore there's no need to modify the guest OS [22] [23]. This abstraction is executed by a virtualization layer called hypervisor (also called Virtual Machine Monitor (VMM). The guest OS is not aware that it is being virtualized. Compared to other virtualization methods, full virtualization provides best isolation of guest operating systems. The downside of full virtualization though is the performance overhead caused by decoupling the hardware and the guest OS. VMware provides full virtualization technique using Binary Translation and Direct Execution (figure 7). There are four privilege levels: Ring 0, 1, 2 and 3. User applications run usually in Ring 3, whereas the ones that need to

have a direct access to the hardware resources run in more privileged Rings (depending on the application).

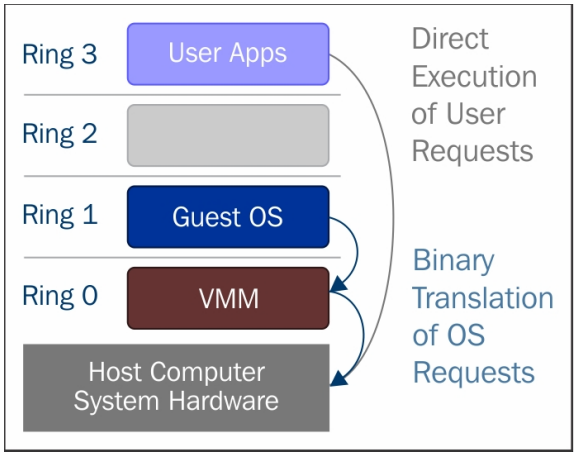


Figure 7. VMware's full virtualization method using Binary Translation and Direct Execution [22]

Paravirtualization improves the overall performance having lower overhead than full virtualization. Software products on the guest OS are able to call the hardware resources directly instead of guest OS making the calls on software's behalf, thus improving the overall efficiency [24]. The drawback is though that the OS kernel needs to be modified, which leads to a poor portability [22]. The reason for necessary OS kernel modification is that the guest OS has to be aware of the fact that it's being virtualized. The virtualization layer between modified guest OS and system hardware is a software interface that receives hypercalls from the guest OS (figure 8). Xen Project (=a hypervisor) [25] is an example of paravirtualization.

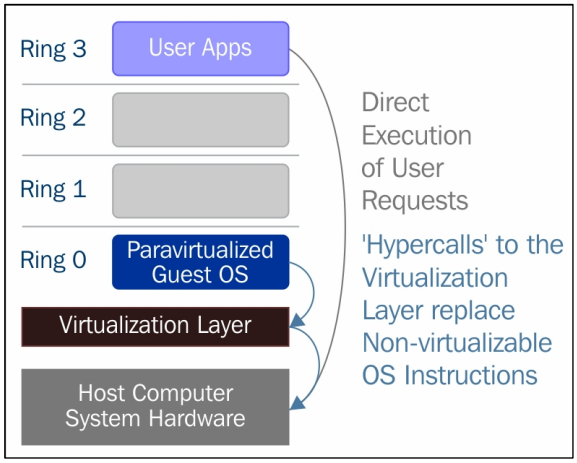


Figure 8. Paravirtualization [22]

Hardware assisted virtualization (figure 9) method provides virtualization performance increase by having a processor extension, currently either Intel's Virtualization Technology VT-x or AMD's AMD-V. Both extensions provide CPU execution mode feature that allows the hypervisor run in Ring -1. Hardware support is possible also for memory virtualization (Intel EPT / AMD NPT) and device and I/O virtualization (Intel VT-d / AMD IOMMU {I/O Memory Management Unit}). In other words, memory & I/O virtualization is performed by the chipset. There's no need to modify the guest OS as in paravirtualization. When it comes to PLC virtualization, hardware assisted virtualization is the virtualization method with best performance and could be therefore considered as a virtualization technique for PLCs.

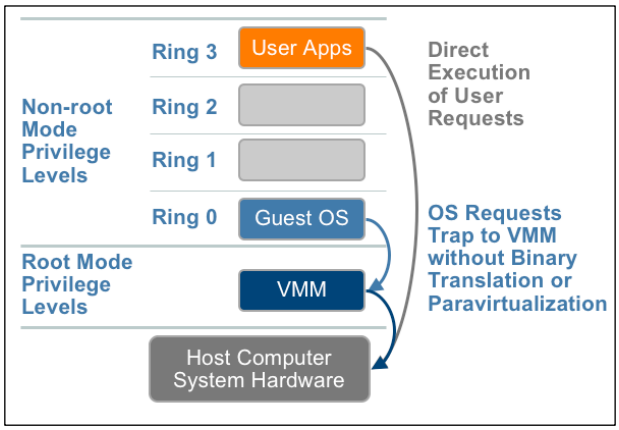


Figure 9. Hardware Assisted Virtualization [22]

Operating system-level virtualization refers to a virtualization method in which isolated instances (called containers) share the same host kernel and they run on a shared OS [23] [27] [28]. OS-level virtualization is also called containerization or container-based virtualization. Containerization also makes better use of hardware resources as "traditional virtualization"-methods do and it is even more lightweight solution. This means that it is possible to create several isolated instances with a single piece of hardware, more than it is possible to create Virtual Machines (VMs) in hypervisor-based virtualization.

The benefit of using containers over VMs is that you don't need to run an entire OS to run an application. Containers include an application, libraries, binaries, dependencies and configuration files and they are abstractions of the host OS rather than the underlying hardware like in hypervisor-based virtualization [29]. As an example, Docker (figure 10) is a container technology that consists of Docker Engine and Docker Hub [30]. Docker engine is installed on the host OS and it creates and runs containers, whereas Docker Hub

is for storing and managing Docker images. The negative aspect of containerization is that all the containers need to run on a same host OS, whereas VMs can use different guest OSs (one VM with Linux OS, another one with Windows etc.).

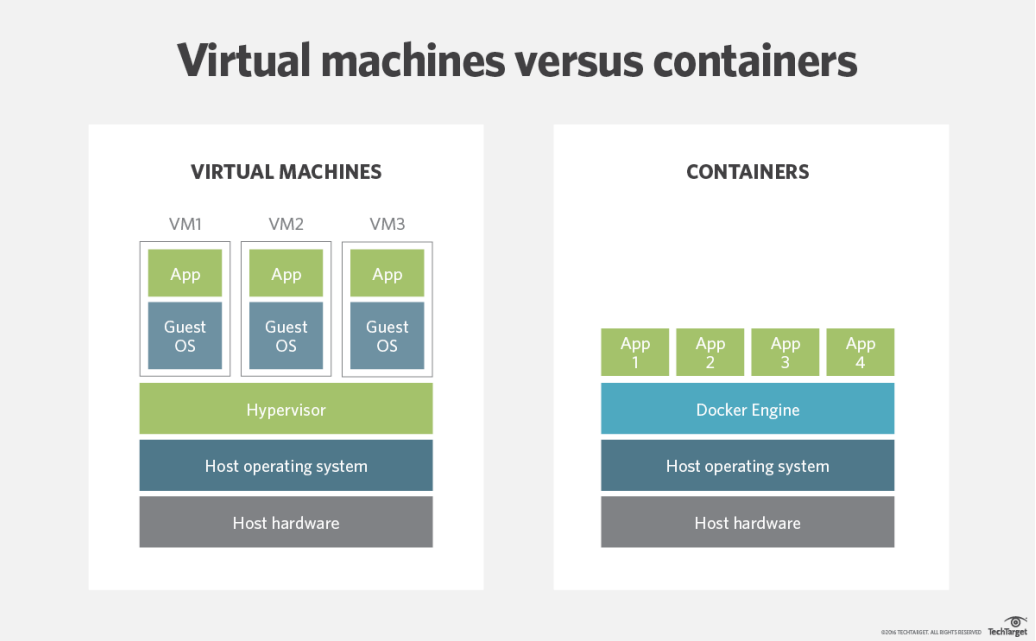


Figure 10. Hypervisor-based virtualization versus containerization architecture [27]

Embedded Virtualization refers to a virtualization implemented in embedded systems [31]. Performance, security, isolation and communication requirements for embedded systems are typically much higher than for enterprise systems [32] [42]. Thus, an 'embedded hypervisor' with real-time computing capabilities is required. The embedded system virtualization is implemented by partitioning a General Purpose Operating System (GPOS) and a Real Time Operating System (RTOS). The non-real-time component might have processing real-time information, whereas the real-time component performs tasks with critical deadlines [33]. It is really important that these two components communicate with each other. For example PikeOS, OKL4, NOVA and Codezero have an embedded virtualization solution. PikeOS embedded virtualization solution (figure 11) is used in the avionics industry.

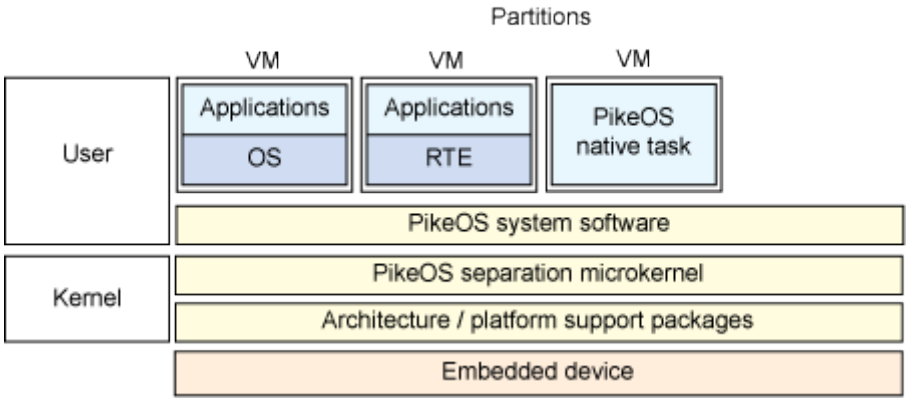


Figure 11. Embedded virtualization solution by PikeOS [32]

4.2 Hypervisor-based virtualization

Hypervisor (also called a Virtual Machine Monitor (VMM)) creates and runs Virtual Machines (VMs) that are emulations from physical computers running operating systems and applications [47] [48]. Hypervisor-based virtualization refers to the hardware virtualization method, in which hypervisor allocates the resources of the single underlying hardware for guest operating systems (=VMs). Traditional infrastructure before virtualization consisted of several servers that were underutilized [49]. Virtualization enables server consolidation and a better use of hardware resources. Physical hardware is expensive and it occupies a lot of space, so virtualization provides significant cost savings and smaller footprint [50].

There are two different types of hypervisors: type 1 and type 2 hypervisor [51]. Type 1 hypervisors are often called bare metal-hypervisors or native-hypervisors. Type 1 hypervisors run on top of system hardware, whereas type 2 hypervisors run on top of a host operating system (figure 12). Type 1 hypervisors have a better performance than type 2 hypervisors, because they have a full control over the underlying hardware resources. However, type 1 hypervisors might require particular hardware features.

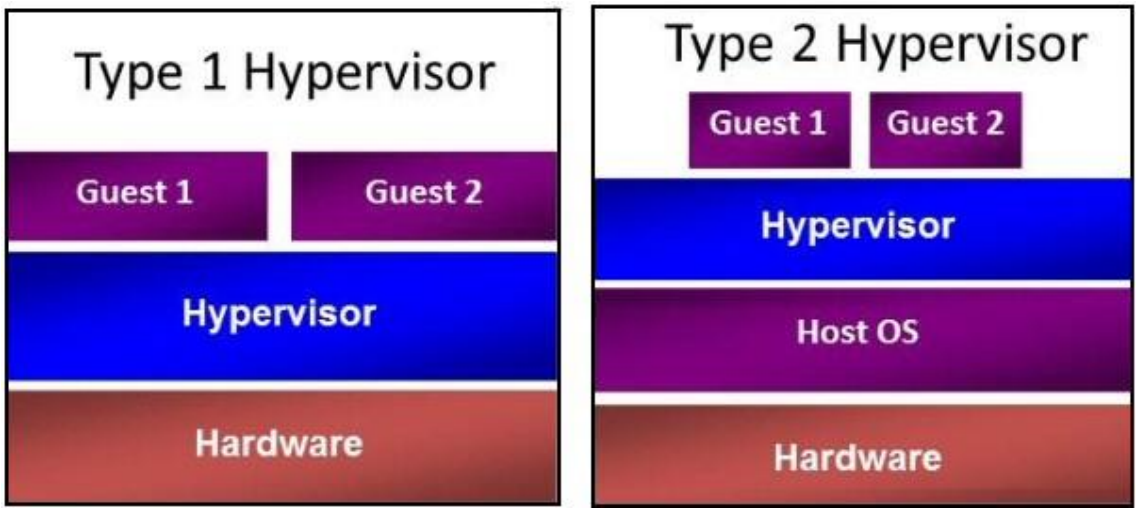


Figure 12. Type 1 and type 2 hypervisor [52]

VMs are isolated instances, which means that each VM is isolated from other VMs. This provides redundancy: if one VM fails, it doesn't affect the other VMs. VMs can also be saved e.g. by using snapshots (VMware). Snapshot saves the current state of a VM and in case the system fails, it can be easily redeployed by using the snapshot image [53]. What is also good about VMs, is that they can be migrated to a new hardware as long as the used hypervisor is compatible with the hardware [54]. VM migration from one host to another is helpful for example when you want to be sure that a VM won't suffer from changes or upgrades you want to do to your host operating system. Each VM has its own operating system that allows users to use several different operating systems on a single hardware if required. User can for example run one VM with a Windows OS and another one with a Linux OS. The big advantage of consolidating servers by using virtualization is centralization. It is significantly easier to manage all your assets from one or two servers compared to couple dozens of them. It is also much easier to expand your system by adding a VM rather than purchasing an expensive new server.

The major downside of hypervisor-based virtualization and virtualization in general is the upfront cost. This means that businesses need to invest a lot of money upfront to deploy virtualization, especially if a business is running a huge number of servers. In spite of that, virtualization provides cost savings in a long run. Businesses also need to consider the security risks they are taking when deploying virtualization. Usually for a host operating system there's already a software detecting and removing viruses, but that's not enough to secure your data within VMs as well. While centralization is a big advantage of virtualization considering asset management, it is also a risk to cause substantial down-

time. A server running a single application is causing downtime just for this one application, whereas a server running multiple applications causes downtime for all the assets managed by this server.

4.3 Container-based virtualization

Containers are sandboxed instances in a single operating system [55] [28]. Container-based virtualization, also called containerization, refers to the operating system-level virtualization method in which the containers use the resources of a single kernel. While hypervisors emulate an entire computer including hardware abstraction, container-based virtualization occurs at the operating system level and that's why it's also called operating system-level virtualization [56]. Hypervisors manages hardware access for VMs, whereas containers use the same kernel of the host operating system [57]. VMs always run an operating system while containers don't, but they need an operating system to run top of it (figure 13). Containers are created from images (binary representation) and they typically include dependencies, libraries, binaries, configurations files and application.

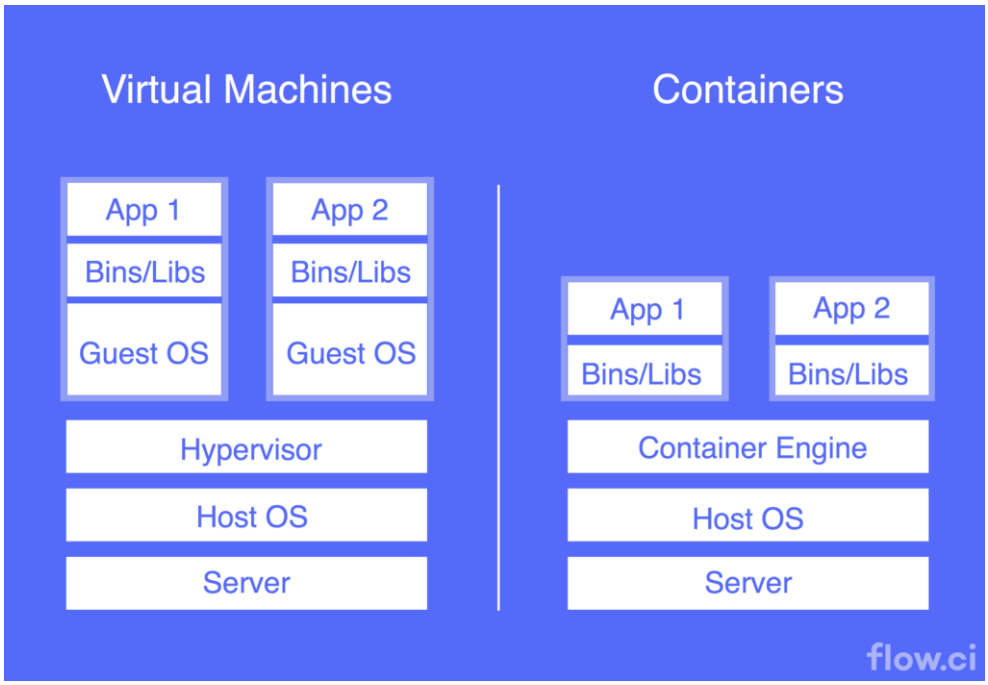


Figure 13. Type 2 hypervisor-architecture vs. Container-based architecture [57]

Since containers don't need to run a whole operating system containing device drivers, kernel etc., the overhead is smaller and they are therefore more lightweight than VMs.

Basically it means that there's only one operating system taking care of hardware access and this leads to greater overall performance. Provisioning of these lightweight instances is really quick as they can be started even in just a few milliseconds, while VMs need much more time to boot [58]. Container images are much smaller (megabytes) than VM images (gigabytes). Hence it is possible to run more containers than VMs on a single hardware. Containerized systems are easily scalable and portable and redundancy can be built to provide high availability. When containers are copied to a new location among the same operating system, the image can be reused as it is [59]. Containers are isolated from each other by using namespaces and cgroups that are certain kernel features [60]. In addition, containerization provides isolation between different processes that are running inside a single container.

Containerization supports DevOps, continuous delivery and deployment and microservices. Microservices is an architecture approach in which an application is decomposed into smaller services [63]. It is a development from monolithic applications (figure 14) [64]. Microservices is a way to a more modular architecture and containerization is one way to implement microservices by separating and isolating processes. Having a common containerized environment enables smooth cooperation between different teams, when building, testing and deploying your application occurs inside containers [18]. It is possible to run different applications written in different programming languages within containers, so containerization improves architecture agility. This gives developers also the flexibility to choose among their preferred technologies. In case an application is divided into several microservices, a part of this application may be updated without having any impact on other parts of the application and thus enabling continuous delivery. Additionally easy sharing and reusing of containers is also a big benefit. Overall the required time from building to deploying your application can be decreased by using containers.

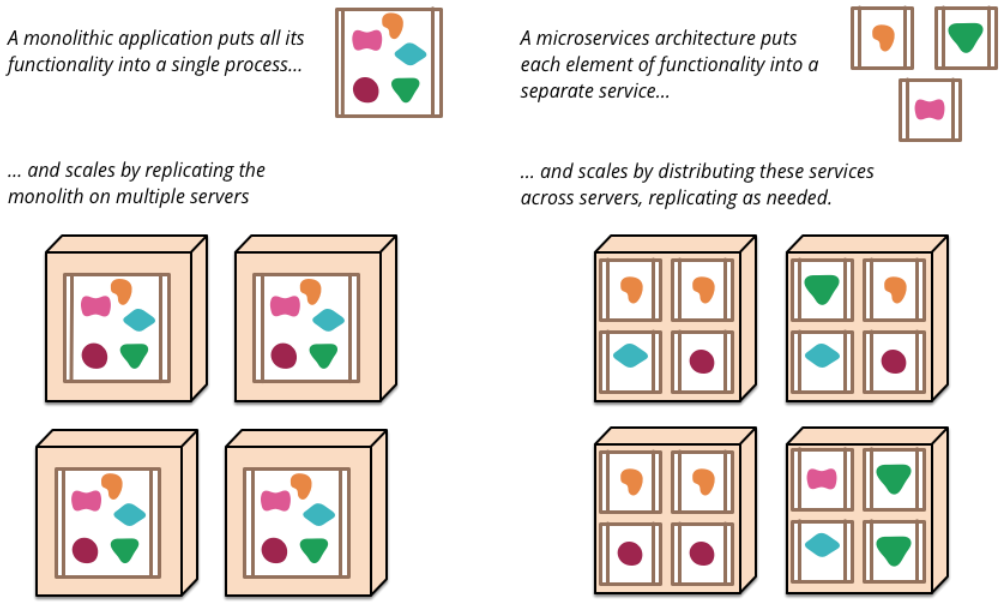


Figure 14. Monolithic application vs. Microservices architecture [64]

Containerization seems to have many advantages over hypervisor-based virtualization, so why would one still use virtual machines instead of containers? The downside about sharing the same kernel between containers is that there's no flexibility to use different operating systems or different kernel than the host kernel. If a system requires two or more different operating systems, a hypervisor is needed to run a guest operating system, which is different than the host operating system. The other option is to buy a new server, which is expensive of course. The second major disadvantage of sharing the same kernel among containers is poor security. In case someone violates the host kernel, it will have an impact on all the containers, so the so called "sandboxed" containers are not so well isolated after all.

4.4 SDN

SDN stands for Software Defined Networking (figure 15) that is an approach to make computer networking more flexible [65]. In SDN the network data plane is decoupled from the network control plane and the network control is programmable and centralized [66]. SDN was created in response to large data center demands. Big benefit about SDN is that it is based on open standards and there's no vendor lock-in, so it supports the SDA approach and Industry 4.0 vision. Compared to traditional networking where the network

switch doesn’t have programmability, SDN enables users to program and control the network via software. As an example, when having a skype call using traditional networking and the used path is down, the skype call gets cut [67]. SDN enables to use alternative paths in case one path is down, so that the call doesn’t get cut. SDN is managed by SDN controller that takes care of the actions of the network switch.

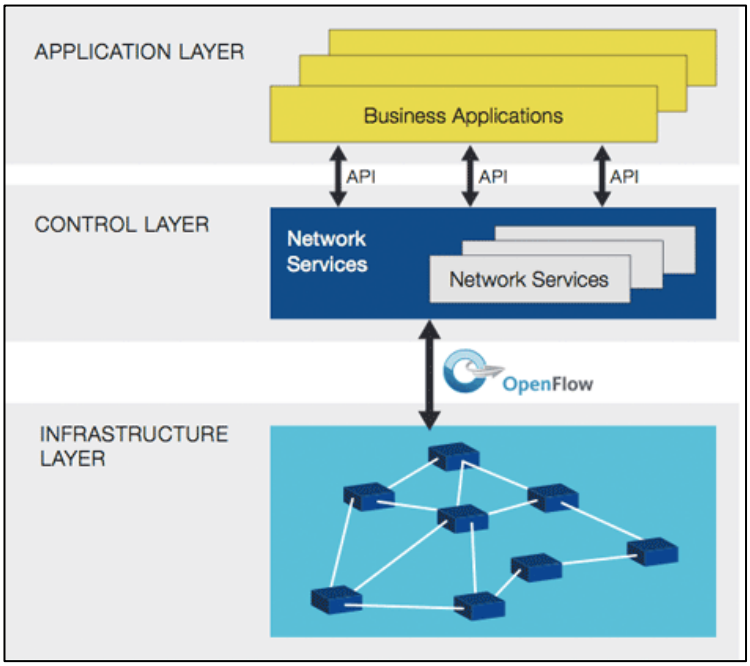


Figure 15. SDN [65]

5 PLC virtualization

5.1 From classic automation pyramid to a more flexible model

The classic automation pyramid (figure 16) represents a model of today's ICSs [34]. All the physical devices from sensors to actuators are in the field level from where the data flows to the second level that controls the physical hardware by using e.g. PLC. The next level is a supervisory level that allows users to monitor and control their processes by SCADA systems [35] [36]. SCADA is abbreviation of Supervisory Control and Data Acquisition and typical SCADA architecture includes the first three levels of the classic automation pyramid. MES and ERP systems are then above the SCADA architecture. MES stands for Manufacturing Execution System and it refers to systems which monitor and control manufacturing data in real-time [37]. MES systems enable tracking of goods for the complete production process. Enterprise Resource Planning (ERP) systems accommodate the highest level of the automation pyramid. ERP systems manage real-time monitoring and controlling of core business processes, such as production or product planning, materials management and finance [38].

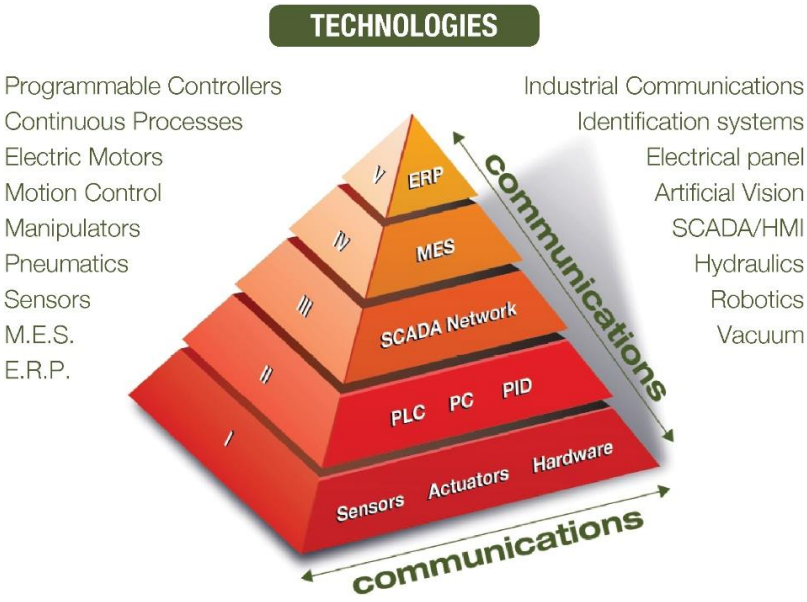


Figure 16. The classic automation pyramid [34]

ICS architectures are going through a change since the arrival of Industry 4.0 and cyber-physical systems. As an example, a MES supplier SYMESTIC [39] realizes a model scenario (figure 17) in which the old automation pyramid model is replaced by a machine-to-machine and machine-to-human communication [40]. This vision supports the increased needs of flexibility and scalability of future ICSs. The field devices, machines and factories have already become "smarter" so that we can talk about Smart Devices, Smart Machines and Smart Factories. However, the "brains" of ICSs, PLCs, are yet to follow the vision. PLC virtualization is one approach, in which virtual PLCs (vPLCs) would replace legacy hardware PLCs.

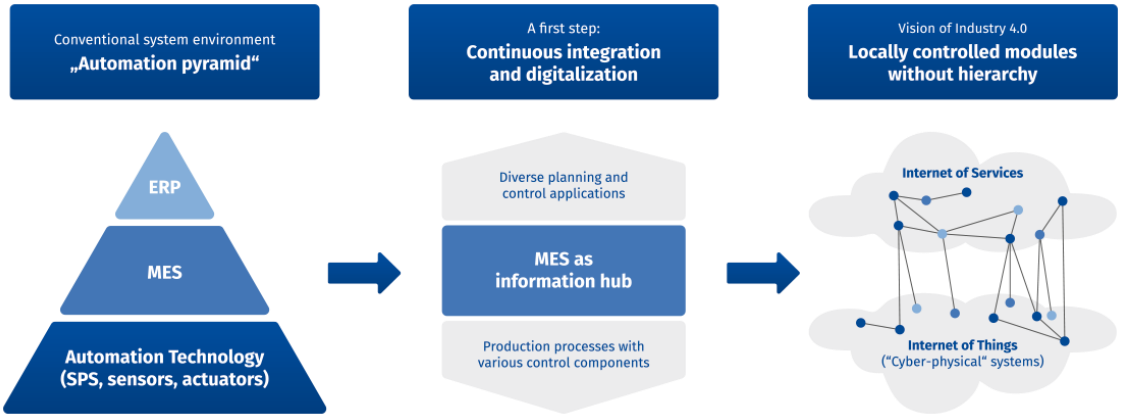


Figure 17. Transition from the automation pyramid to a new Industry 4.0-based model (SYMESTIC) [39]

5.2 PLC virtualization advantages and question marks

Distributed Control System (DCS) requires lots of hardware from wires and cables to several I/Os and control devices. If the all the hardware control devices were replaced by virtual PLCs, it would basically mean software replacing hardware. This supports the SDA approach. Operational costs, such as power and cooling costs, can be reduced by reducing the amount of necessary hardware. This leads also to a smaller footprint, since virtual PLCs don't occupy space inside control cabinets. Usually embedded systems require multiple processors, but by the use of virtualization servers can be consolidated. PLC virtualization is a promising approach in response to the increasing needs of flexibility of ICSs.

Virtualized systems can be easily modified by adding or replacing components without any impact on the rest of the system. The aim in SDAs is to have no vendor lock-in,

meaning that third party components could be easily used. That's not the case with hardware PLCs, because normally you are tied to the PLC-vendor. In addition to modularity, virtualized systems are also easily scalable. It is more affordable, faster and simpler to add virtual PLCs to a DCS than physical hardware PLCs. It is also possible to move a virtual PLC to a completely different machine [41]. Redundancy can be guaranteed by running multiple virtual PLCs: in case one fails, other one will take over the control. This way uptime can be increased and downtime minimized. Redundancy can be achieved by cloning a virtual PLC. There's then no need to test the cloned virtual PLC, because unchanged objects will surely function the same way. Virtualization secures safe environment for tests and deployment: in case of a system failure, it is easier and faster to recover from that failure when running your system in a virtualized environment.

One approach to compare virtual PLCs with hardware PLCs is to compare PC- and PLC-based control [43] [44]. PLCs are developed for automation world providing high performance and robustness, whereas PCs might not meet the same requirements, at least not yet. There are also industrial PCs on the market that are robust and powerful enough for automation, but there's still factors that makes manufacturers choose a hardware PLC over an industrial PC. PC-based control is pushed by *Beckhoff Automation* already since the 1980s [45]. PLC has an embedded real-time operating system (RTOS) that guarantees managing tasks with critical deadlines, but PC should be able to achieve the same reliability by using a real-time kernel or RTOS. PC's built-in user interface is an advantage of PC-based control whilst PLC would need some additional components, such as operator panel(s), switches or even an industrial PC for the user interface. PCs number of interfaces also beat the amount of interfaces in PLC that would need additional modules to provide the same variety of interfaces. It is also easy to add extra memory and more computing power to PCs, whereas PLCs always have their constraints. Multi-core technology [46] enables multitasking, so by using a PC or an industrial PC with a multi-core processor reduces essentially the amount of necessary hardware, because PLC-based control would need more hardware to perform several tasks. Splitting multiple tasks on multiple cores improves the overall performance by balancing loads and providing more CPU time. When it comes to safety-critical applications though, PLC is still much more reliable thanks to its long-term service in ICSs. PC-based control has still a long way to go to achieve the same reliability in terms of safety and determinism.

In a PC-based control, developers have more to choose from when it comes to programming. PLC programming is typically based on the standard IEC 61131-3, while widely used programming languages like C or C++ can be used in PC-based control. Code execution differs between the two different control options. PLC's program execution is either scan- or event-driven or a mixture of these, whereas PC runs a code as event-driven. The scan-driven program execution is priority-based, so tasks with high priority are run first. The drawback is that this might take longer sometimes than the event-driven execution.

What about when it comes to money? Upfront cost is higher in PC-based systems, but in a long run it might pay off because of better system flexibility and scalability. It always depends on the application, whether a PC-based solution or a PLC-based solution would be a better choice.

The major question marks of PLC virtualization are cybersecurity, determinism and migration to legacy systems. Security is a big concern, especially now, when the Industry 4.0, IoT, SDAs etc. are taking over. PLC's dedicated OS is less susceptible for virus attacks than PCs, though for PLCs there is no software to detect and remove viruses. It is still uncertain, whether a PC-based control can meet hard real-time requirements even though there are RTOSs and real-time kernels available. There are simply no implementations yet where a virtual PLC is able to control the system meeting hard real-time requirements. Appropriate communication protocols are important along proper Internet connections. Businesses already lose significant amount of money nowadays because of dropped Internet connections, so if the future's businesses are even more dependent on Internet, how big losses are we then talking about? It is still to be seen whether virtual PLCs can be easily migrated to legacy systems. When manufacturers intend to deploy new technologies, they want to be sure that the new elements and the old elements are compatible. A virtual PLC has its constraints anyways: it can't for example support all the possible communication protocols, which means it won't be just 'plug-in and play'.

6 Case Study

6.1 Software and components

6.1.1 Node-RED

Node-RED is a flow-based programming tool and developed by JS Foundation [71]. It has been especially developed for IoT and it is based on Node.js. Node-RED enables wiring together hardware devices, online services (like Twitter) and application programming interfaces. The programming interface is accessible via browser (figure 18). Node-RED flows are created using nodes that can either be input-, output- or function-nodes, for example. As an example, flow can consist of an inject-node (input-node) and a debug-node (output-node). Inject-node produces a message either manually or repeatedly at regular intervals, depending how the user wants to set the properties of the node. Debug-node provides messages which are then displayed in the debug sidebar tab.

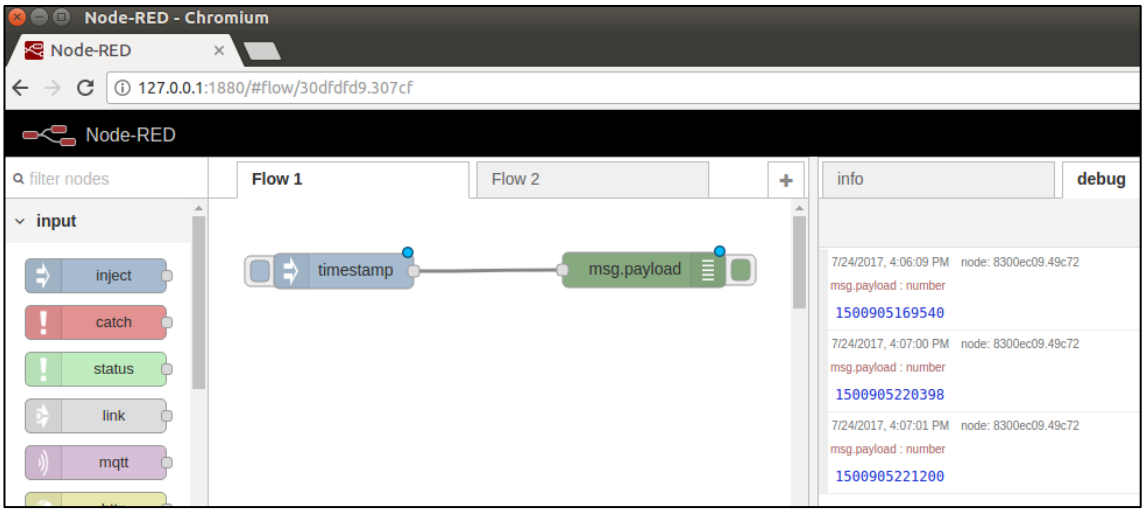


Figure 18. Example flow in Node-RED

6.1.2 OpenPLC

OpenPLC is an open source PLC project created by Thiago Rodrigues Alves [72]. It is a standard industrial controller that can be programmed with all the standard PLC languages (IEC-61131-3) from Structured Text (ST), Ladder Diagram (LD), Function Block

Diagram (FBD) and Instruction List (IL) to Sequential Function Chart (SFC). The OpenPLC software is portable and it supports multiple platforms, such as Windows, Linux, Raspberry Pi and Arduino. The OpenPLC uses Modbus/TCP for communication. The project also includes a programming software called PLCopen Editor. PLC programs can be created with any of the standard programming languages, but when the user wants to download the program to the OpenPLC, the program needs to be generated to a ST-program, because OpenPLC is only capable to run ST programs. ST programs can be downloaded to the OpenPLC via webserver (figure 19), where the OpenPLC can be also set to run or stop and PLC logs can be viewed.

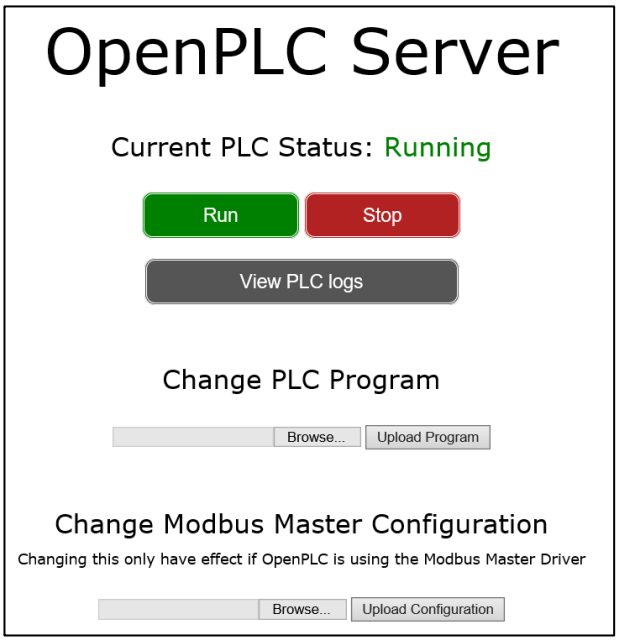


Figure 19. OpenPLC webserver

6.1.3 Docker

Nowadays almost all the containers are Linux containers that are known as LXC. Docker is probably the most well-known container platform based on LXC [61]. Docker Community Edition is a free platform, but Docker also provides Enterprise Edition-solutions with software, support and certification. Docker consists of two components: Docker Engine and Docker Hub. Docker Engine is for building and containerizing applications, while Docker Hub is managing and sharing these applications as a SaaS service [62].

Docker containers can be created by using images that are each based on a Dockerfile. Dockerfile is a script of commands that allows one to build images. Multiple containers can be created using the same image. As an example, if you have a Dockerfile inside a *"test"*-folder in your Desktop, you can create an image of it by running a single command (figure 20):

```
ipc@ipc-S-MPC61SV022415:~/Desktop/test$ sudo docker build -t testimage .
```

Figure 20. Building an image called "testimage" from a Dockerfile inside "test"-folder

After creating an image called *"testimage"*, docker images can be listed by running the following command (figure 21):

```
ipc@ipc-S-MPC61SV022415:~/Desktop/test$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
testimage           latest              2dcece824614        8 seconds ago       769 MB
```

Figure 21. List of Docker images

A container called *"testcontainer"* can then be created from the *"testimage"*-image by running a command *"docker run"* (figure 22):

```
ipc@ipc-S-MPC61SV022415:~/Desktop/test$ sudo docker run --name testcontainer testimage
```

Figure 22. Docker container creation

Running containers can be seen by running the following command (figure 23):

```
ipc@ipc-S-MPC61SV022415:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
d16f7739ec84        testimage           "npm start -- --us..."   About a minute ago   Up About a minute   1880/tcp            testcontainer
```

Figure 23. List of running Docker containers

6.1.4 KVM and QEMU

KVM is a hardware assisted virtualization solution for Linux hardware [75]. KVM is often called a Linux kernel module. Hardware must have a virtualization extension, either AMD-V or Intel VT depending on the processor. KVM stands for Kernel-based Virtual Machine and it is an open source software. KVM's task is to accelerate hardware and enhance virtual machine's performance [76]. KVM can't create and run virtual machines

just by itself, but it needs QEMU (=Quick Emulator) that in fact emulates the hardware. However, QEMU needs KVM to minimize the overhead and thus boost performance of a system. QEMU itself is a type 2 hypervisor, but in combination with KVM it becomes a type 1 hypervisor.

6.1.5 I/Os, SoMachine

I/Os were needed to implement the control loop setup. Schneider Electric provided I/Os for this case study: TM251MESE logic controller and TM3DM8R I/O module were used (figure 24). As this thesis has its focus on PLC virtualization, the logic controller was not used as a PLC, but the logic controller was necessary to establish a Modbus TCP/IP connection between the I/Os and Node-RED. It's not possible to establish a connection between SoMachine software (see next paragraph) and the I/O card alone. The logic controller was connected to the software to set the device address. This device address could then be used in Node-RED to read the inputs and write the outputs. The TM3DM8R consists of four digital inputs and four relay outputs.

Figure 24. TM251MESE logic controller and TM3DM8R I/O-module

SoMachine [69] is a programming software by Schneider Electric. The software is based on Codesys [70]. Software can be used to program the entire machine including a PAC or PLC, HMI, motor control and related network automation functions. In this case study SoMachine was used to set a device address for the logic controller (figure 25). In addition the used devices (PLC+I/O-module) had to be added to the software configuration.

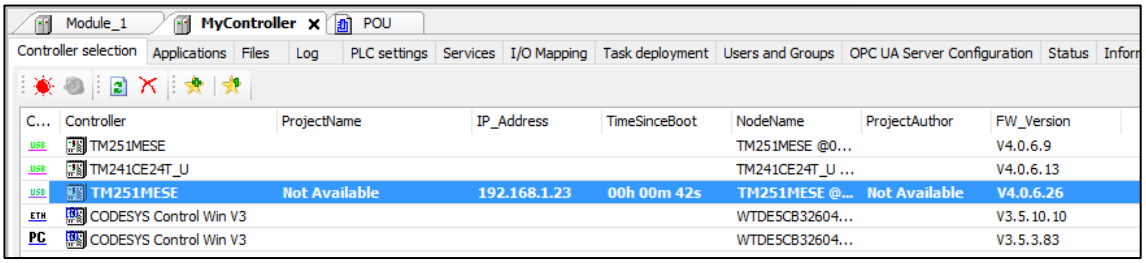


Figure 25. SoMachine

6.1.6 Modbus TCP/IP

Modbus TCP/IP is a Modbus communication protocol developed by Modicon (now Schneider Electric) [73]. Modbus TCP/IP runs on Ethernet over a TCP interface [74]. In other words, Modbus TCP/IP uses TCP/IP and physical network (=Ethernet) to carry messages. TCP stands for Transmission Control Protocol and IP refers to Internet Protocol. TCP makes sure that all data packets are received correctly and IP ensures correct addressing and routing. The protocol uses the *Client-Server model*. A client might be also called as a master that sends data requests to servers. This means that a client always establishes the communication as servers just wait for data requests. Server's task is simply to respond to data requests.

Modbus data model consists of four different data types:

-Discrete inputs

-Coils

-Input registers

-Holding registers

Discrete inputs are only readable, whilst coils are readable and writable. Discrete inputs and coils (also called as discrete outputs) are 1-bit registers, whereas input and holding

registers are 16-bit registers. Input registers may only be read, holding register may be both read and written.

6.1.7 Industrial PC

Node-RED and OpenPLC were running on an industrial PC (iPC) that had Ubuntu 16.04 LTS as an operating system. Industrial PC was chosen to this implementation as a PC hardware to test the suitability of an iPC in such a solution. The solution was performed on a Magelis iPC (HMIBMPSI74D4801) manufactured by Schneider Electric. This Magelis iPC has an Intel QM87 chipset with hardware assisted virtualization support (Intel VT-x) and Intel Core i7-4650U processor with four 1.7 GHz CPU cores. It has 8 GB RAM memory and 80 GB SSD disk.

6.2 Setup

For the case study, cooperation with Cedric Vandendriessche was done [68]. Vandendriessche earns a big praise for his support. Without his support this thesis would have most likely had a different approach. The aim was to continue his work by making similar tests but also compare the results with a hypervisor-based approach. Therefore, the main idea of this case study was to compare container- and hypervisor-based approaches and analyze whether these approaches would already be suitable for some industrial applications.

This case study focuses on a test setup that represents a control loop (figure 26). In ICSs, a control loop consists of “input-logic-output”-chain. In other words, a control loop needs firstly an input, such as a sensor or a switch. Input is read by a processing device (e.g. PLC) that executes the logic. After the logic execution, the processing device needs to write the output that can be for instance a contactor or a variable speed drive. In this case study, Schneider Electric’s TM3 I/O was used as input and output and OpenPLC was used as a processing device (vPLC). Node-RED was used to upload a program on OpenPLC and enable communication between OpenPLC and the I/O. I/O and iPC were physically connected to a network switch, so that Modbus messages could be sent over Ethernet.

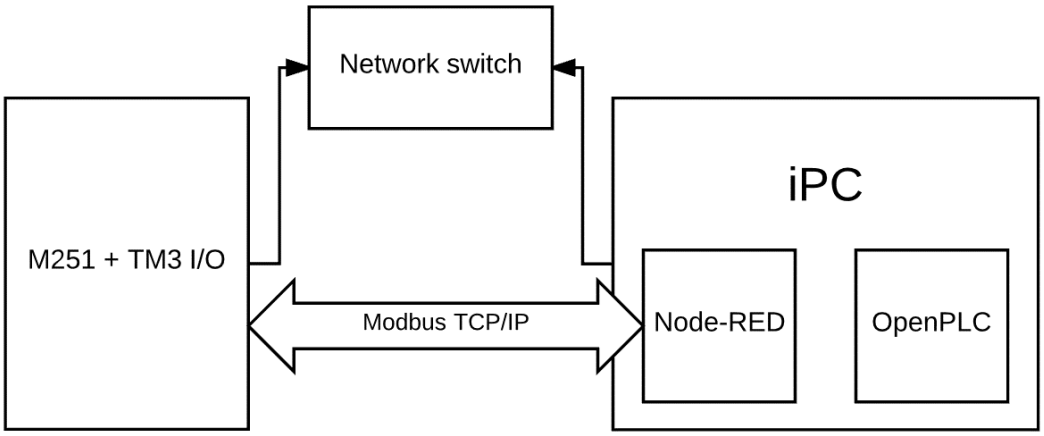


Figure 26. Control loop setup

Control loop test was implemented with three different approaches: container-based virtualization, hypervisor-based virtualization and without virtualization. Container-based virtualization was implemented by using Docker and hypervisor-based virtualization was implemented with KVM. To capture the Modbus TCP/IP traffic, tcpdump [77] was used. Wireshark [78] was then used to analyze data packets and measure latencies. Control loop flow was created in Node-RED (figure 27).

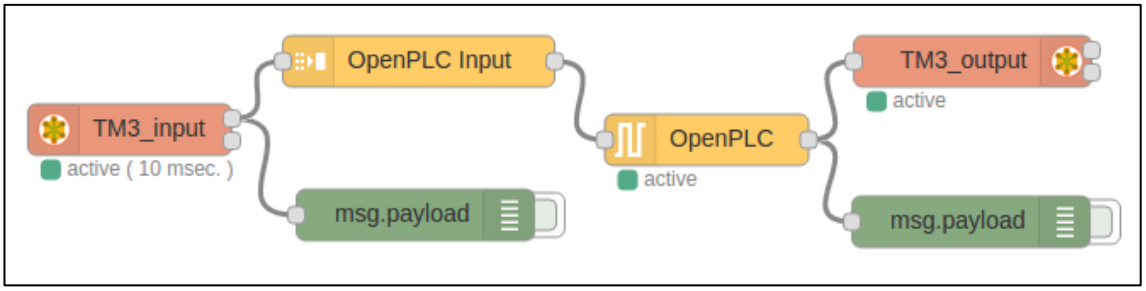


Figure 27. Control loop flow in Node-RED

To read and write Modbus messages, Modbus contribution package for Node-RED was installed [85]. In addition there's a contribution package for OpenPLC nodes [86]. The great thing about the OpenPLC node is that there's no need to access the web server to upload a program to the OpenPLC, but the program can be written in the node configuration (figure 28). The program execution port needs to be specified in the node configuration (port 8080 in the figure below).

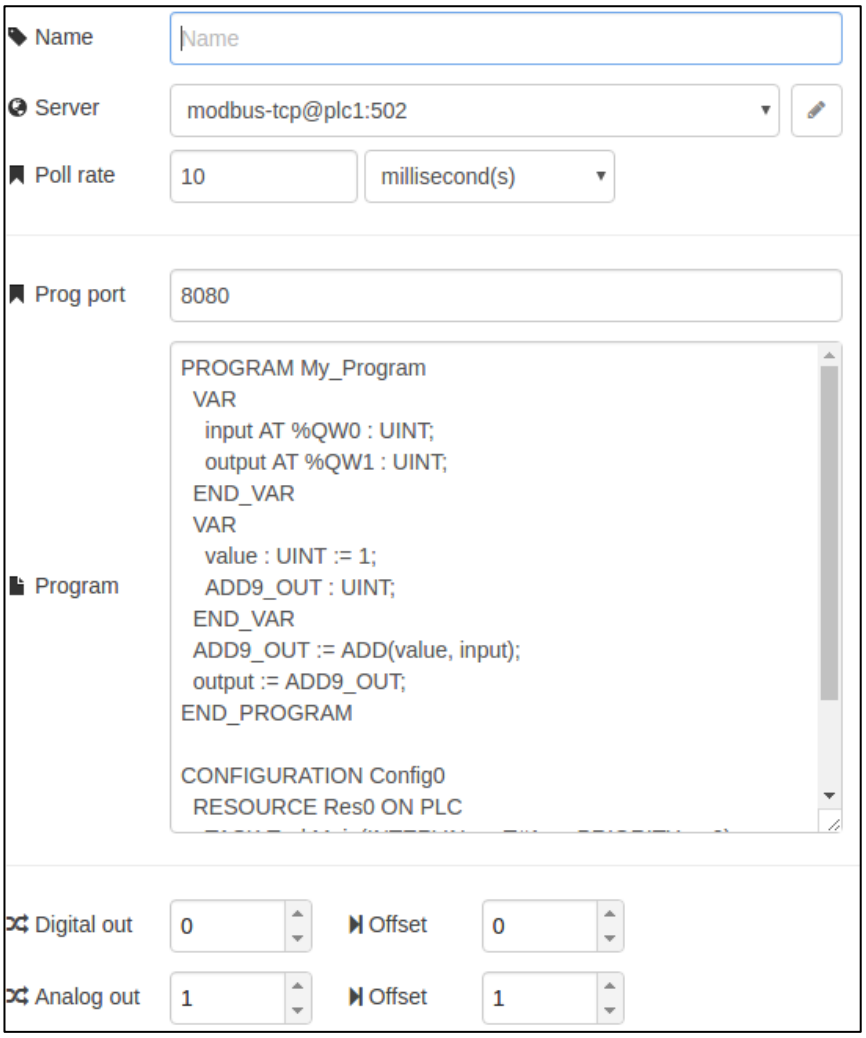


Figure 28. OpenPLC-node configuration

To enable communication between the nodes, host addresses needed to be specified. The device address of TM251MESE logic controller was given to *TM3_modbus-client*-server, which was used for *TM3_input* and *TM3_output* nodes (figure 29). In container-based virtualization, OpenPLC and Node-RED containers communicated over a common network that was created in Docker (see next chapter). A localhost address was used for OpenPLC-server in hypervisor-based virtualization.

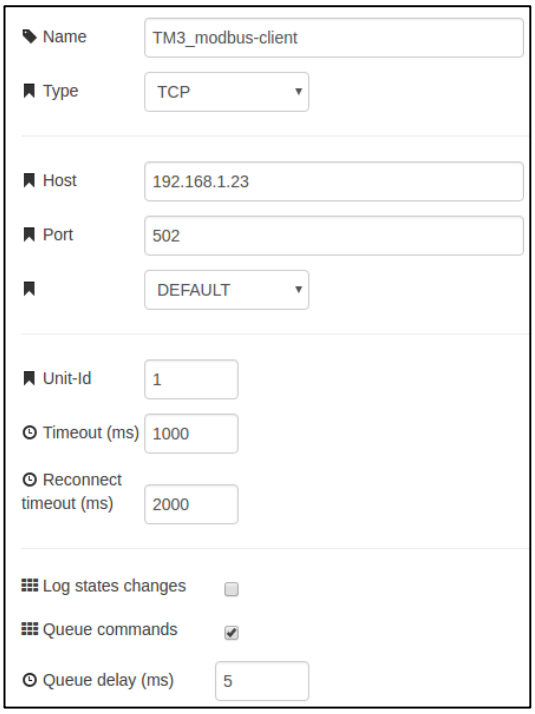


Figure 29. Configuration of the TM3_modbus-client-server

6.2.1 Docker = Container-based virtualization

Firstly, Docker was installed on Ubuntu OS. Afterwards images of Node-RED and OpenPLC were created based on their Dockerfiles. To enable communication between Node-RED and OpenPLC, a new network had to be created using Docker. A new network can be simply created by running a simple command (figure 30):

```
ipc@ubuntu:~$ docker network create fog-layer
```

Figure 30. Creating a new docker network called fog-layer

After creation of the new network, Node-RED and OpenPLC instances can be connected to this network either when creating the container instances by using *'docker run'*-command with a *net*-flag or after creating the instances by using *'docker network'*-command [79].

Container instances of Node-RED and OpenPLC were finally created by using *'docker run'*-command. When starting Node-RED inside a container, port 1880 at the localhost was exposed to access Node-RED via browser.

6.2.2 KVM = hypervisor-based virtualization

Before the implementation of this case study, Jailhouse [80] and Xen [81] were considered as hypervisor-based virtualization solutions, but in the end there was not enough time to do research on these solutions. Thus KVM was chosen, as the KVM virtualization solution was really easy to implement and the performance should be good enough for the case study.

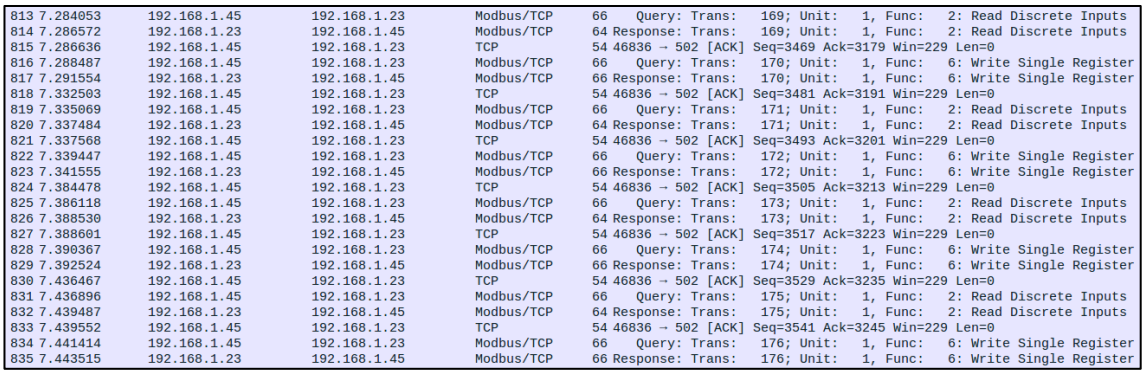
Firstly, a new VM was created with Ubuntu 16.04 LTS OS. 5000MB memory, 20 GB RAM and 4 CPU cores were allocated. After successful installation of the OS, the control loop environment had to be set up. Node-RED and OpenPLC were installed on Ubuntu [82] [83]. Then nodes were configured to enable communication between I/O, Node-RED and OpenPLC. To install OpenPLC nodes, a different version of Node.js had to be installed. With the use of nvm, a node.js version 6.9.2 was installed successfully [84] and then OpenPLC nodes could be used in Node-RED.

6.2.3 No virtualization

The same tests were also implemented without virtualization to have a reference point and evaluate the overhead of the different virtualization solutions. Control loop environment was basically set up the same way as it was done for the hypervisor-based solution. The only difference was that the setup was not done in a VM but in the host OS. Same host OS (Ubuntu 16.04 LTS) was used as in the other approaches.

7 Results

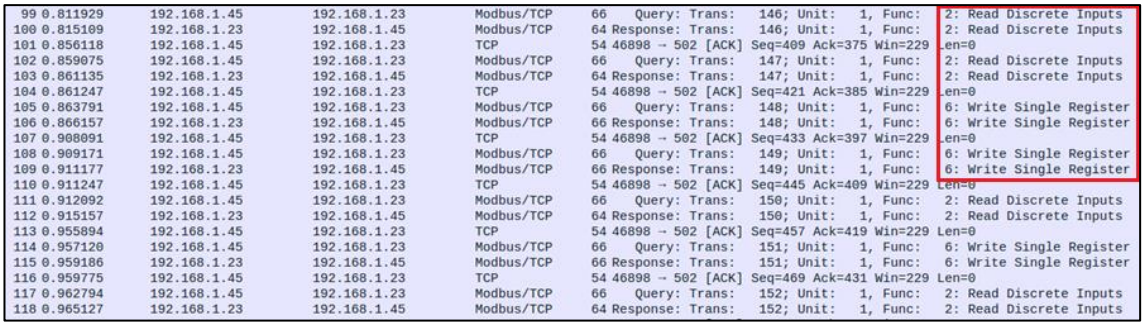
To measure the control loop performance, Modbus traffic was captured and analyzed. Control loop starts from reading the inputs and ends up to writing the output. Response time was measured by analyzing the Modbus traffic on Wireshark (figure 31). Modbus traffic from and to the I/O was captured.



813	7.284053	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 169; Unit: 1, Func: 2: Read Discrete Inputs
814	7.286572	192.168.1.23	192.168.1.45	Modbus/TCP	64 Response: Trans: 169; Unit: 1, Func: 2: Read Discrete Inputs
815	7.286636	192.168.1.45	192.168.1.23	TCP	54 46836 → 502 [ACK] Seq=3469 Ack=3179 Win=229 Len=0
816	7.288487	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 170; Unit: 1, Func: 6: Write Single Register
817	7.291554	192.168.1.23	192.168.1.45	Modbus/TCP	66 Response: Trans: 170; Unit: 1, Func: 6: Write Single Register
818	7.332503	192.168.1.45	192.168.1.23	TCP	54 46836 → 502 [ACK] Seq=3481 Ack=3191 Win=229 Len=0
819	7.335069	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 171; Unit: 1, Func: 2: Read Discrete Inputs
820	7.337484	192.168.1.23	192.168.1.45	Modbus/TCP	64 Response: Trans: 171; Unit: 1, Func: 2: Read Discrete Inputs
821	7.337568	192.168.1.45	192.168.1.23	TCP	54 46836 → 502 [ACK] Seq=3493 Ack=3201 Win=229 Len=0
822	7.339447	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 172; Unit: 1, Func: 6: Write Single Register
823	7.341555	192.168.1.23	192.168.1.45	Modbus/TCP	66 Response: Trans: 172; Unit: 1, Func: 6: Write Single Register
824	7.384478	192.168.1.45	192.168.1.23	TCP	54 46836 → 502 [ACK] Seq=3505 Ack=3213 Win=229 Len=0
825	7.386118	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 173; Unit: 1, Func: 2: Read Discrete Inputs
826	7.388530	192.168.1.23	192.168.1.45	Modbus/TCP	64 Response: Trans: 173; Unit: 1, Func: 2: Read Discrete Inputs
827	7.388601	192.168.1.45	192.168.1.23	TCP	54 46836 → 502 [ACK] Seq=3517 Ack=3223 Win=229 Len=0
828	7.390367	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 174; Unit: 1, Func: 6: Write Single Register
829	7.392524	192.168.1.23	192.168.1.45	Modbus/TCP	66 Response: Trans: 174; Unit: 1, Func: 6: Write Single Register
830	7.436467	192.168.1.45	192.168.1.23	TCP	54 46836 → 502 [ACK] Seq=3529 Ack=3235 Win=229 Len=0
831	7.436896	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 175; Unit: 1, Func: 2: Read Discrete Inputs
832	7.439487	192.168.1.23	192.168.1.45	Modbus/TCP	64 Response: Trans: 175; Unit: 1, Func: 2: Read Discrete Inputs
833	7.439552	192.168.1.45	192.168.1.23	TCP	54 46836 → 502 [ACK] Seq=3541 Ack=3245 Win=229 Len=0
834	7.441414	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 176; Unit: 1, Func: 6: Write Single Register
835	7.443515	192.168.1.23	192.168.1.45	Modbus/TCP	66 Response: Trans: 176; Unit: 1, Func: 6: Write Single Register

Figure 31. Modbus traffic on Wireshark

Sometimes Modbus messages arrived late and that caused latency issues. After reading the inputs, output should be updated, but sometimes messages were delayed and output was updated late (figure 32). The amount of delayed messages varied depending on the approach and the polling rate.



99	0.811929	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 146; Unit: 1, Func: 2: Read Discrete Inputs
100	0.815109	192.168.1.23	192.168.1.45	Modbus/TCP	64 Response: Trans: 146; Unit: 1, Func: 2: Read Discrete Inputs
101	0.856118	192.168.1.45	192.168.1.23	TCP	54 46898 → 502 [ACK] Seq=409 Ack=375 Win=229 Len=0
102	0.859075	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 147; Unit: 1, Func: 2: Read Discrete Inputs
103	0.861135	192.168.1.23	192.168.1.45	Modbus/TCP	64 Response: Trans: 147; Unit: 1, Func: 2: Read Discrete Inputs
104	0.861247	192.168.1.45	192.168.1.23	TCP	54 46898 → 502 [ACK] Seq=421 Ack=385 Win=229 Len=0
105	0.863791	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 148; Unit: 1, Func: 6: Write Single Register
106	0.866157	192.168.1.23	192.168.1.45	Modbus/TCP	66 Response: Trans: 148; Unit: 1, Func: 6: Write Single Register
107	0.908091	192.168.1.45	192.168.1.23	TCP	54 46898 → 502 [ACK] Seq=433 Ack=397 Win=229 Len=0
108	0.909171	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 149; Unit: 1, Func: 6: Write Single Register
109	0.911177	192.168.1.23	192.168.1.45	Modbus/TCP	66 Response: Trans: 149; Unit: 1, Func: 6: Write Single Register
110	0.911247	192.168.1.45	192.168.1.23	TCP	54 46898 → 502 [ACK] Seq=445 Ack=409 Win=229 Len=0
111	0.912092	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 150; Unit: 1, Func: 2: Read Discrete Inputs
112	0.915157	192.168.1.23	192.168.1.45	Modbus/TCP	64 Response: Trans: 150; Unit: 1, Func: 2: Read Discrete Inputs
113	0.955894	192.168.1.45	192.168.1.23	TCP	54 46898 → 502 [ACK] Seq=457 Ack=419 Win=229 Len=0
114	0.957120	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 151; Unit: 1, Func: 6: Write Single Register
115	0.959186	192.168.1.23	192.168.1.45	Modbus/TCP	66 Response: Trans: 151; Unit: 1, Func: 6: Write Single Register
116	0.959775	192.168.1.45	192.168.1.23	TCP	54 46898 → 502 [ACK] Seq=469 Ack=431 Win=229 Len=0
117	0.962794	192.168.1.45	192.168.1.23	Modbus/TCP	66 Query: Trans: 152; Unit: 1, Func: 2: Read Discrete Inputs
118	0.965127	192.168.1.23	192.168.1.45	Modbus/TCP	64 Response: Trans: 152; Unit: 1, Func: 2: Read Discrete Inputs

Figure 32. Latency issue

Tests were done using 10ms, 50ms and 500ms polling rates. 12 measurements were always done for one deployment and then the same configuration was deployed again. This was repeated 20 times, so 240 iterations were performed in each case for each approach (see attachments 1,2 and 3). Minimum, maximum, median, average and 90th percentile latencies are shown in graphs (figures 33, 34 and 35).

With a 10ms polling rate the system was not able to poll every 10ms, but every 15-18ms. This applied to all the different approaches. Messages arrived late occasionally, but the response time was 13-15ms on average. Maximum values are a concern, because such peak latencies result to a fact that this solution can't be used in applications in which less than a 35-40ms response time is required, depending on the approach. As expected, containerization seems to have a smaller overhead than the hypervisor-based solution. Minimum values were more or less the same, but median, average and maximum values indicate that the hypervisor-based solution is less performant. As the maximum latency value without virtualization is higher than the equivalent with Docker, we can expect that such latencies can also occur with Docker but more iterations would have been needed to realize that. In this 10ms polling rate test, the hypervisor-based solution was a little bit unstable and during the tests it felt like the system is going to crash. Most likely the iPC was just not powerful enough to process everything and in addition there's the overhead of hypervisor. The nodes were configured the same way in all the different approaches, so the node configurations can't be blamed.

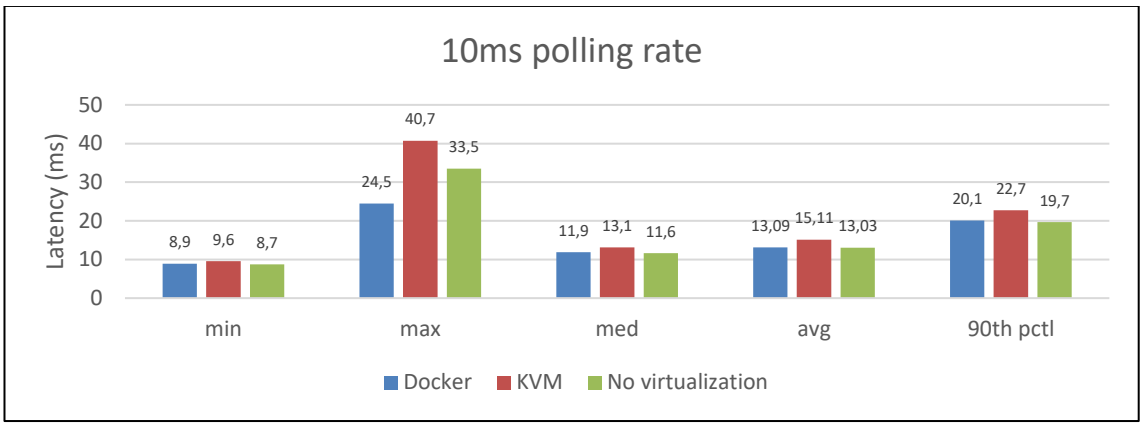


Figure 33. Response time of a control loop with a 10ms polling rate

With a 50ms polling rate the system was actually able to poll every 50ms. Peak latency values were above 50ms in every approach, which is more than expected. Before the response time tests it was tested, how redeploying the configuration has an impact if it has any. By redeploying the same configuration over and over again, inconsistent latencies were seen when comparing the latencies after each redeployment. That's the reason why the configuration was always redeployed 20 times and the measurements were not taken in just one shot. For instance, if the first deployment showed latencies between 5 and 30ms, the next deployment might have shown latencies between 30 and 50ms. The cause

of this remained unsolved. Otherwise the results show the same than the results with the 10ms polling rate: Docker has a smaller overhead than KVM and a very little overhead in general.

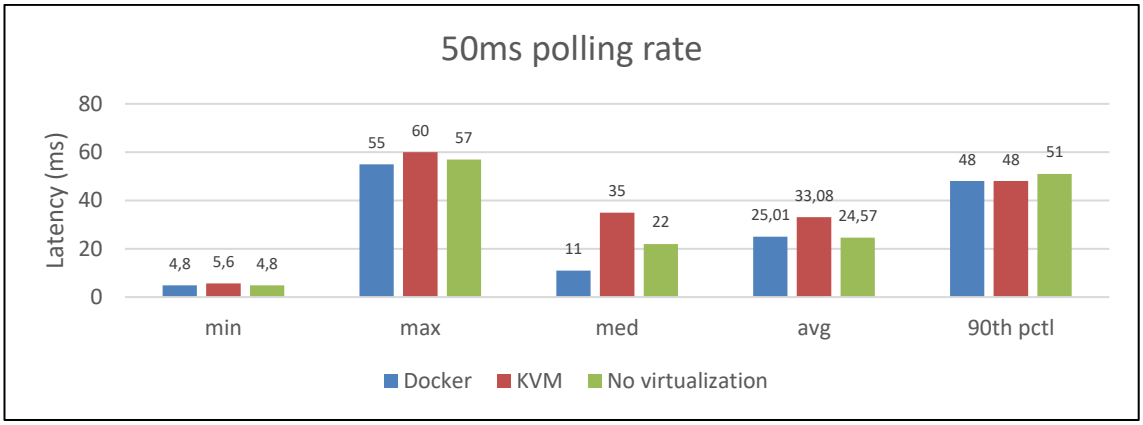


Figure 34. Response time of a control loop with a 50ms polling rate

The same issue occurred with a 500ms polling rate: latencies above the polling rate were measured both in hypervisor-based virtualization and in container-based virtualization. The minimum and median values were low, but occasional high latencies of 450-500ms are a huge concern. 500ms polling rate is naturally not suitable for general PLC applications, but the idea here was to test whether a higher polling rate would stabilize the system and latencies above the polling rate would not be seen anymore. Also tests with an even higher polling rate (1s, 2s, 4s) were performed, but the same issue seemed to remain.

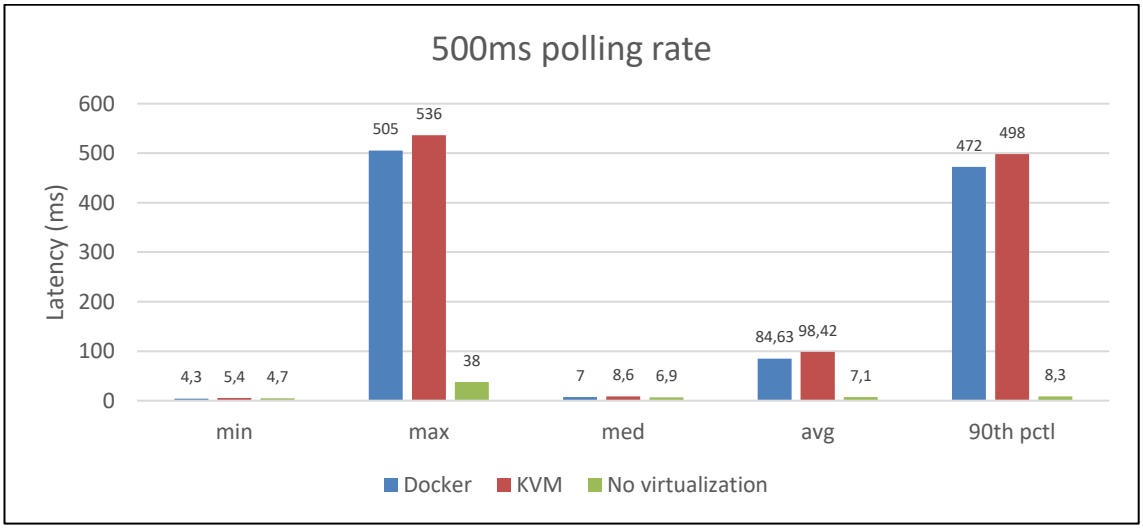


Figure 35. Response time of a control loop with a 500ms polling rate

8 Conclusion

The ongoing industrial revolution is pushing the automation world to use new IT technologies, such as virtualization. However, there are still several issues that force us to develop these technologies and come up with new ones to overcome the issues and that way respond to the market demands. One of the big concerns is network uptime. Even with nowadays' systems the businesses encounter situations in which network downtime causes major problems in terms of money. The question is, how much downtime are we then going to face when the systems increase the use of IT technologies and network? When talking about ICSs, hardware PLCs are yet so much more reliable than any of the so far proposed virtual solutions that it's going to take some time before the ICSs are going to say goodbye to legacy hardware PLCs, if ever.

Container-based virtualization, hypervisor-based virtualization and SDN were introduced in this thesis. According to the marketing research and the case study, container-based solution seems to be a more promising approach than hypervisor-based solution when talking about PLC virtualization. Still we have to keep in mind that there are also downsides like security on container-based virtualization. In addition, the hypervisor used in this case study is not the best available one, but some more performant real-time hypervisors could provide better results. The future will show us, how we could also benefit from SDN-enabled communications fabric [96]. Anyhow, the solution introduced in this thesis would not be suitable to any hard real-time, high speed motion- or safety-related applications. Further investigation on the topic would be needed to estimate the actual potential on hard real-time applications. First of all, a more performant hardware PC would be needed to perform such tests. Secondly, a faster industrial protocol than Modbus TCP/IP would be required. Perhaps another kind of approach would be needed to overcome the issues what were experienced in this case study. It would be interesting to test a direct communication between virtual PLC and I/O and use Node-RED nodes for example to trigger inputs.

As the IT technologies keep developing, we can later on surely take PLC virtualization more seriously. Anyways, it always depends on the application what kind of solution would be suitable. For instance, it doesn't make sense to virtualize systems if we are talking about a really small industrial company with a machine or two to control. In any

case, it'll be exciting to see how SDAs will take over the control of ICSs and whether some hard real-time or safety-related applications will be controlled by virtual PLCs in the future.

8.1 ALC

In this chapter a concept called App Logic Controller (ALC) is introduced. ALC is a fairly new concept and at the moment it is just in a Proof of Concept-level, like PLC virtualization in general. Today's automation systems require lots of different field devices such as relays and sensors to control machines or production lines. For instance, a cutting machine controlled by a PLC most likely needs couple sensors to collect important information from the machine. This requires the machine user to purchase at least an input module. Input modules consist of certain amount of inputs from 4 to 16 or even more, depending on the vendor. So, if the machine user needs 2 inputs, but the vendors only offer input modules with 4, 8 or 16 inputs, the machine user needs to pay for more inputs than he needs. This is just one example, but especially in huge factories, where many machines and production lines need to be controlled, the customers pay for more than what they actually need. The current demands force the product vendors to develop their products to be more agile and fit better to the needs of the customers. A good definition for an app was found from *WhatIs.com*: *"An application is a software program that's designed to perform a specific function directly for the user or, in some cases, for another application program"* [87]. So if a PLC could be replaced by an ALC that would provide the user only the functions that the user needs, wouldn't that be a perfect solution to respond to the market demands?

ALC is a promising approach especially for ICSs, both for customers and for vendors. Concept of Field Device App introduced by Syed Shiraz Gilani, Tim Tack, Holger Flatt and Jürgen Jasperneite [88] demonstrates one ALC approach. This concept introduces two scenarios in which the Field Device App could be used: *"Providing field device functionality through an app"* and *"Enhancing a device with additional functionality through an app"*. The first scenario's procedure starts from finding and installing the app to the execution of the app. The second scenario begins with reconfiguring the field device, which is followed by finding, installing and executing the app. To explain the functionality of the Field Device App in other words, the field device functionality is provided by

a software product installed from an *"App Distribution Platform"*. The app is (or apps are) then executed in reconfigurable field devices by the machine user.

There are also others who are interested in the concept. Maarten Ectors introduced the ALC concept in October 2016 [89], listed its benefits and actually developed a prototype [90]. ALC introduced by Ectors would consist of two processors: a micro-controller that would take care of the logic execution and a typical mobile phone processor that would manage everything else from updating the logic on the micro-controller to system monitoring. Ectors states three advantages that ALC has over a standard PLC: it is a cheaper solution, it will be easier to program and it will be revenue generating for customers. Bosch Rexroth has introduced an ALC at Embedded World Nuremberg at the Ubuntu stand in March in 2017 [91]. Kunbus has worked on the concept as well by developing Revolution Pi which is based on apps called snaps [92]. The Kunbus has already a quick start guide and video tutorials online [93]. Revolution Pi is based on Raspberry Pi and it is open source. There's also Controllino [94], which is a step towards ALC. Controllino is based on Arduino Open Source Software Technology and it is freely programmable controller. There are few different models of Controllino already available and the same programming platform (Arduino IDE) than for Arduino can be used.

So, to sum up the ALC, it is a controller that can be reconfigured any time by downloading apps from a platform such as Google Play Store. It is an approach that would improve the modularity and scalability of ICSs. The customer would be able to use just the functions that the customer needs and there would be no extra costs for some functionality that customer doesn't need. The approach is really innovative: there could be a specific app store for Industry, where programmers could upload their apps. If ALC would be a physical hardware, then it would have a processor or two that have their limits of course. If a reliable and performant PLC virtualization solution can be developed, perhaps ALC could be a software product. In that case the virtualization benefits could be also utilized.

REFERENCES

[1] Schneider Electric. 2005. Schneider Electric, 170 years of history. Accessed 02.06.2017. http://www.schneider-electric.co.cr/documents/empresa/se_history_brands_march2005.pdf

[2] Schneider Electric. 2016. Schneider Electric. Accessed 02.06.2017. http://www.schneider-electric.com

[3] i-SCOOP. Copyright 2016-2020. Industry 4.0: the fourth industrial revolution – guide to Industrie 4.0. Accessed 09.06.2017. https://www.i-scoop.eu/industry-4-0/

[4] Wikipedia. 2017. Industrial Revolution. Accessed 09.06.2017. https://en.wikipedia.org/wiki/Industrial_Revolution

[5] Wikipedia. 2017. Industry 4.0. Accessed 09.06.2017. https://en.wikipedia.org/wiki/Industry_4.0

[6] TechTarget. 2015. Industrial Internet of Things (IIoT). Accessed 09.06.2017. http://internetofthingsagenda.techtarget.com/definition/Industrial-Internet-of-Things-IIoT

[7] i-SCOOP. Copyright 2016-2020. Industrial Internet of Things (IIoT): definition, benefits, standards and evolutions. Accessed 09.06.2017. https://www.i-scoop.eu/internet-of-things-guide/industrial-internet-things-iiot-saving-costs-innovation/

[8] Vanessa Romero Segovia and Alfred Theorin. 15.06.2012. History of Control, History of PLC and DCS.

[9] Wind River Systems, Inc. 2017. Open, Secure Industrial Automation Systems.

[10] Germany Trade and Invest. 2017. Cyber-physical systems. Accessed 13.06.2017. http://industrie4.0.gtai.de/INDUSTRIE40/Navigation/EN/Topics/The-internet-of-things/cyber-physical-systems,t=cyberphysical-systems,did=1201724.html

[11] TechTarget. 2014. Service-oriented architecture (SOA). Accessed 13.06.2017. http://searchmicroservices.techtarget.com/definition/service-oriented-architecture-SOA

[12] QuinStreet Inc. 2017. COTS - commercial off-the-shelf. Accessed 13.06.2017. http://www.webopedia.com/TERM/C/COTS.html

[13] Michael Wahler, Thomas Gamer, Atul Kumar, Manuel Oriol. 2015. FASA: A Software Architecture and Runtime Framework for Flexible Distributed Automation Systems.

[14] Modulus. 2015. Software-defined architecture taking web scale to the next level in the cloud.

[15] CFE Media. 2006. What is a smart sensor? Accessed 14.06.2017. http://www.controleng.com/single-article/what-is-a-smart-sensor/014bc5b589f4b6a1bfc6f874fc3ecfe4.html

[16] AssetPoint. 2015. TabWare Predictive Maintenance Management (PdM). Accessed 14.06.2017. http://www.assetpoint.com/capabilities/eam-predictive-maintenance-cmms/

[17] Puppet. 2017. DevOps. Accessed 14.06.2017. https://puppet.com/solutions/devops

[18] MediaOps, LLC. 2015. What Do Containers Have to Do with DevOps, Anyway? Accessed 14.06.2017. http://containerjournal.com/2017/01/11/containers-devops-any-way/

[19] Dell EMC World. 2015. DevOps, Containers and Microservices: Buzzwords or Fundamental to Survival? Accessed 14.06.2017. https://www.youtube.com/watch?v=ivHVUTWRmxs

[20] Wikipedia. 2017. DevOps. Accessed 14.06.2017. https://en.wikipedia.org/wiki/DevOps

[21] Golden B. 2007. Virtualization for dummies. John Wiley & Sons.

[22] VMware, Inc. 2007. Understanding Full Virtualization, Paravirtualization and Hardware Assisted Virtualization.

[23] Hyungro Lee. 2015. Virtualization Basics: Understanding Techniques and Fundamentals.

[24] Virtualization Softwares. 2011. What is…? Accessed 14.06.2017. http://www.virtualizationsoftwares.com/virtualization-article/

[25] Wikipedia. 2017. Xen. Accessed 14.06.2017. https://en.wikipedia.org/wiki/Xen

[26] Docker Inc. 2017. What is a Container. Accessed 14.06.2017. https://www.docker.com/what-container

[27] TechTarget. 2017. Container vs. VM: What's the difference? Accessed 14.06.2017. http://searchservervirtualization.techtarget.com/answer/Containers-vs-VMs-Whats-the-difference

[28] TechTarget. 2016. Containerization (container-based virtualization). Accessed 14.06.2017. http://searchservervirtualization.techtarget.com/definition/container-based-virtualization-operating-system-level-virtualization

[29] IDG Communications, Inc. 2017. What are containers and why do you need them? Accessed 14.06.2017. http://www.cio.com/article/2924995/software/what-are-containers-and-why-do-you-need-them.html

[30] Docker Inc. 2017. About Docker Engine. Accessed 14.06.2017. https://docs.docker.com/engine/

[31] Extension Media, Kim Hartman, TenAsys Corporation. 2017. Embedded Virtualization — the key to Real-time Determinism in Multi-OS Systems. Accessed 16.06.2017. http://www.embeddedintel.com/technology_applications.php?article=1830

[32] IBM, M. Jones. 2011. Virtualization for embedded systems. Accessed 16.06.2017. https://www.ibm.com/developerworks/library/l-embedded-virtualization/index.html

[33] Curt Schwaderer. 2014. Embedded virtualization: Latest trends and techniques. Accessed 16.06.2017. http://embedded-computing.com/articles/embedded-virtualization-latest-trends-techniques/

[34] SMC International Training. 2017. Automation pyramid. Accessed 19.06.2017. http://www.smctraining.com/en/webpage/indexpage/312

[35] QuinStreet Inc. 2017. SCADA - supervisory control and data acquisition. Accessed 19.06.2017. http://www.webopedia.com/TERM/S/SCADA.html

[36] Inductive Automation. 2017. What is SCADA? Accessed 19.06.2017. https://inductiveautomation.com/what-is-scada

[37] TechTarget. 2010. Manufacturing execution system (MES). Accessed 19.06.2017. http://searchmanufacturingerp.techtarget.com/definition/manufacturing-execution-system-MES

[38] Wikipedia. 2017. Enterprise resource planning. Accessed 19.06.2017. https://en.wikipedia.org/wiki/Enterprise_resource_planning

[39] symestic GmbH. 2017. Industry 4.0 and MES. Accessed 19.06.2017. http://www.symestic.com/en/industry-4-0.html

[40] Zumbach Electronic AG. 2017. OPC UA - Overview. Accessed 19.06.2017. http://www.zumbach.com/products/product-finder/opc-ua/opc-ua-overwiew.html

[41] Contact and Coil, Scott Whitlock. 2015. TwinCAT 3 Tutorial: Multiple Virtual PLCs. Accessed 21.06.2017. http://www.contactandcoil.com/twincat-3-tutorial/multiple-virtual-plcs/

[42] TenAsys Corporation. 2010. Embedded Virtualization - The Key to Real-time Determinism. Accessed 22.06.2017. http://www.tenasys.com/index.php/embedded-virtualization-technology

[43] Innovasic, Inc. 2013. PLCs vs. industrial PCs for automation today, part 1. Accessed 23.06.2017. http://www.innovasic.com/news/industrial-ethernet/plcs-vs-industrial-pcs-for-automation-today-part-1/

[44] Bosch Rexroth Corporation. 2011. PC vs. PLC: key factors in comparing control options. Accessed 23.06.2017. https://www.automation.com/pdf_articles/Rexroth_PLCvsPC_L.pdf

[45] PMMI Media Group. 2017. Industry 4.0 Meets the Internet of Things. Accessed 23.06.2017. https://www.automationworld.com/article/topics/industrial-internet-things/industry-40-meets-internet-things

[46] Control Design, Paul Commisso. 2015. Industrial Ethernet and Multi-Core Technology. Accessed 23.06.2017. http://www.controldesign.com/articles/2015/industrial-ethernet-and-multi-core-technology/?show=all

[47] Wikipedia. 2017. Hypervisor. Accessed 01.07.2017. https://en.wikipedia.org/wiki/Hypervisor

[48] Wikipedia. 2017. Virtual Machine. Accessed 01.07.2017. https://en.wikipedia.org/wiki/Virtual_machine

[49] Purch, Sara Angeles. 2014. The Pros and Cons of Virtualization. Accessed 01.07.2017. http://www.businessnewsdaily.com/6014-pros-cons-virtualization.html

[50] Cross Company, David Ward. 2015. The Pros and Cons of Virtualizing a Control System. Accessed 01.07.2017. https://innovativecontrols.com/blog/pros-and-cons-virtualizing-control-system

[51] TechTarget. 2010. What's the difference between Type 1 and Type 2 hypervisors? Accessed 01.07.2017. http://searchservervirtualization.techtarget.com/feature/Whats-the-difference-between-Type-1-and-Type-2-hypervisors

[52] Computer Performance LTD, Guy Thomas. Copyright 1999-2017. Windows 8 Hyper-Visor. Accessed 01.07.2017. http://www.computerperformance.co.uk/win8/windows8-hyper-v.htm

[53] TechTarget. 2014. Backup and disaster recovery in the age of virtualisation. Accessed 01.07.2017. http://www.computerweekly.com/feature/Backup-and-disaster-recovery-in-the-age-of-virtualisation

[54] Red Hat, Inc. 2017. What is migration? Accessed 01.07.2017. https://docs.fedoraproject.org/en-US/Fedora/19/html/Virtualization_Getting_Started_Guide/sec-migration.html

[55] VMware Cloud-Native. 2017. What is a Container? Accessed 03.07.2017. https://www.youtube.com/watch?v=EnJ7qX9fkcU

[56] Wikipedia. 2017. Operating-system-level virtualization. Accessed 03.07.2017. https://en.wikipedia.org/wiki/Operating-system-level_virtualization

[57] flow.ci. 2016. Introduction to Containers: Concept, Pros and Cons, Orchestration, Docker, and Other Alternatives. Accessed 03.07.2017. https://medium.com/flow-ci/introduction-to-containers-concept-pros-and-cons-orchestration-docker-and-other-alternatives-9a2f1b61132c

[58] Michael Eder. 2016. Hypervisor- vs. Container-based Virtualization.

[59] Alexandru Moga, Thanikesavan Sivanthi, Carsten Franke. 2016. OS-Level Virtualization for Industrial Automation Systems: Are We There Yet?

[60] Asad Javed. 2015. Linux Containers: An Emerging Cloud Technology.

[61] Docker Inc. 2017. Docker. Accessed 03.07.2017. https://www.docker.com/

[62] Wikipedia. 2017. Software as a service. Accessed 03.07.2017. https://en.wikipedia.org/wiki/Software_as_a_service

[63] Wikipedia. 2017. Microservices. Accessed 03.07.2017. https://en.wikipedia.org/wiki/Microservices

[64] Martin Fowler. 2014. Microservices. Accessed 03.07.2017. https://martinfowler.com/articles/microservices.html

[65] Open Networking Foundation. 2017. Software-Defined Networking (SDN) Definition. Accessed 04.07.2017. https://www.opennetworking.org/sdn-resources/sdn-definition

[66] Ingram Micro. 2017. 5 Differences Between SDN and Network Functions Virtualization. Accessed 04.07.2017. http://www.ingrammicroadvisor.com/data-center/5-differences-between-sdn-and-network-functions-virtualization

[67] Sanjeev Palamand. 2014. Traditional Networking v/s SDN. Accessed 04.07.2017. https://www.youtube.com/watch?v=TPmvEnLr08g

[68] Cedric Vandendriessche. 2017. PLC virtualization for an agile Industry 4.0 compliant distributed system.

[69] Schneider Electric. 2016. SoMachine. Accessed 24.07.2017. http://www.schneider-electric.com/en/product-range-presentation/2226-somachine/

[70] 3S-Smart Software Solutions GmbH. 2017. CODESYS. https://www.codesys.com/

[71] JS Foundation. 2014. Node-RED. Accessed 24.07.2017. https://nodered.org/

[72] Thiago Rodrigues Alves. 2015. OpenPLC. Accessed 24.07.2017. http://www.openplcproject.com/

[73] Wikipedia. 2017. Modbus. Accessed 27.07.2017. https://en.wikipedia.org/wiki/Modbus

[74] Acromag, Inc. 2005. Introduction to Modbus TCP/IP. https://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf

[75] KVM. 2016. Main Page. Accessed 03.08.2017. https://www.linux-kvm.org/page/Main_Page

[76] WordPress; Sriram Subramanian. 2017. Blogs about Cloud, Virtualization and everything else under the sun. Accessed 03.08.2017. http://www.innervoice.in/blogs/2014/03/10/kvm-and-qemu/

[77] Tcpdump/Libcap. 2017. Tcpdump & Libcap. Accessed 04.08.2017. http://www.tcpdump.org/

[78] Wireshark. 2017. Wireshark. Accessed 04.08.2017. https://www.wireshark.org/

[79] Docker. 2017. Docker network create. Accessed 04.08.2017. https://docs.docker.com/engine/reference/commandline/network_create/#extended-description

[80] GitHub, Inc. 2017. Linux-based partitioning hypervisor. Accessed 04.08.2017. https://github.com/siemens/jailhouse

[81] Xen Project, A Linux Foundation Collaborative Project. 2013. Xen Project. Accessed 04.08.2017. https://www.xenproject.org/

[82] DigitalOcean Inc; Brian Boucheron. 2016. How to Connect Your Internet of Things with Node-RED on Ubuntu 16.04. Accessed 04.08.2017. https://www.digitalocean.com/community/tutorials/how-to-connect-your-internet-of-things-with-node-red-on-ubuntu-16-04

[83] Thiago Rodrigues Alves. 2015. Linux. Accessed 04.08.2017. http://www.openplcproject.com/getting-started-linux

[84] LinuxConfig.org; Lubos Rendek. 2016. How to install Node.js on Ubuntu 16.04 Xenial Xerus Linux server. Accessed 04.08.2017. https://linuxconfig.org/how-to-install-node-js-on-ubuntu-16-04-xenial-xerus-linux-server

[85] GitHub, Inc. 2016. Node-red-contrib-modbus. Accessed 04.08.2017. https://github.com/biancode/node-red-contrib-modbus

[86] GitHub, Inc. 2017. Node-red-contrib-openplc. Accessed 05.08.2017. https://github.com/SkyTrix/node-red-contrib-openplc

[87] TechTarget. 2011. App. Accessed 08.08.2017. http://searchmobilecomputing.techtarget.com/definition/app

[88] Syed Shiraz Gilani, Tim Tack, Holger Flatt, Jürgen Jasperneite. 2014. An App-Based Approach for Reconfigurable Field Devices.

[89] LinkedIn Corporation; Maarten Ectors. 2016. How ALCs will disrupt the PLC market... Accessed 08.08.2017. https://www.linkedin.com/pulse/how-alcs-disrupt-plc-market-maarten-ectors?lipi=urn%3Ali%3Apage%3Ad_flagship3_search_srp_content%3BHeWXon%2FdTDKK7%2BEecsQUyw%3D%3D&licu=urn%3Ali%3Acontrol%3Ad_flagship3_search_srp_content-object

[90] LinkedIn Corporation; Maarten Ectors. 2016. The first App Logic Controller (ALC) Prototype is here... Accessed 08.08.2017. https://www.linkedin.com/pulse/first-app-logic-controller-alc-prototype-here-maarten-ectors?lipi=urn%3Ali%3Apage%3Ad_flagship3_search_srp_content%3BHeWXon%2FdTDKK7%2BEecsQUyw%3D%3D&licu=urn%3Ali%3Acontrol%3Ad_flagship3_search_srp_content-object

[91] LinkedIn Corporation; Maarten Ectors. 2017. How Bosch Rexroth is out-innovating the PLC market... Accessed 08.08.2017. https://www.linkedin.com/pulse/how-bosch-rexroth-out-innovating-plc-market-maarten-ectors?lipi=urn%3Ali%3Apage%3Ad_flagship3_search_srp_content%3BHeWXon%2FdTDKK7%2BEecsQUyw%3D%3D&licu=urn%3Ali%3Acontrol%3Ad_flagship3_search_srp_content-object

[92] LinkedIn Corporation; Maarten Ectors. 2017. App Store for PLC – ALC Part 2 – industrial-ready Revolution Pi. Accessed 08.08.2017. https://www.linkedin.com/pulse/app-store-plc-alc-part-2-industrial-ready-pi-maarten-ectors

[93] KUNBUS GmbH. 2016. Revolution Pi. Accessed 08.08.2017. https://revolution.kunbus.com/

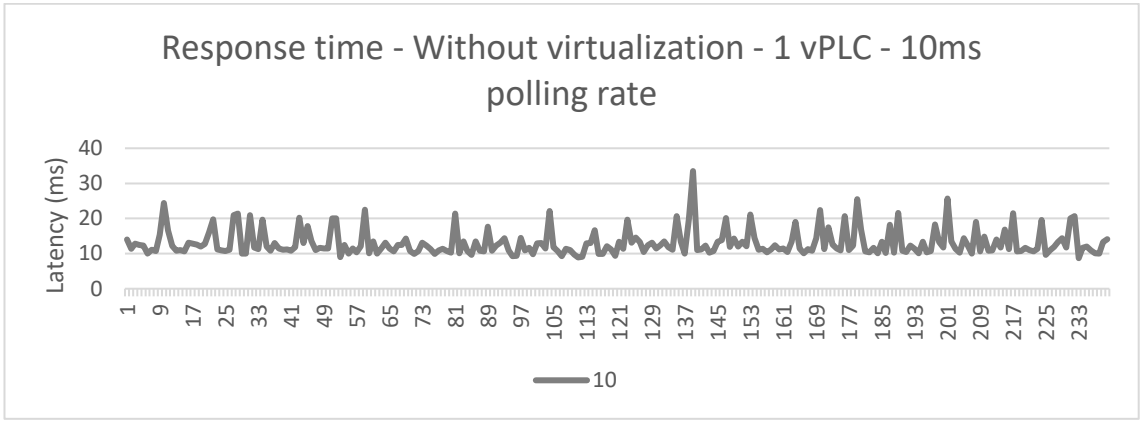
[94] CONELCOM GmbH. 2017. Controllino. Accessed 08.08.2017. http://controllino.biz/

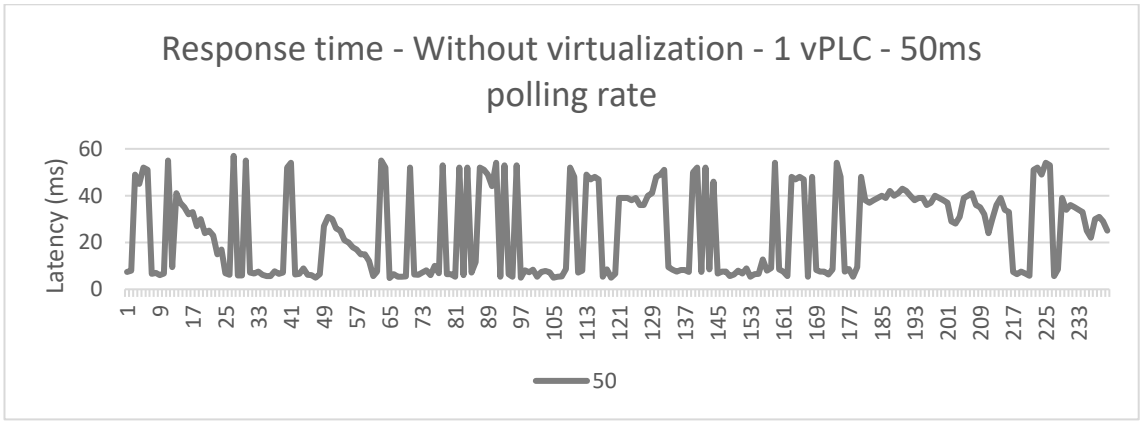
[95] Omid Givehchi, Jahanzaib Imtiaz, Henning Trsek and Juergen Jasperneite. 2014. Control-as-a-Service from the Cloud: A Case Study for using Virtualized PLCs.

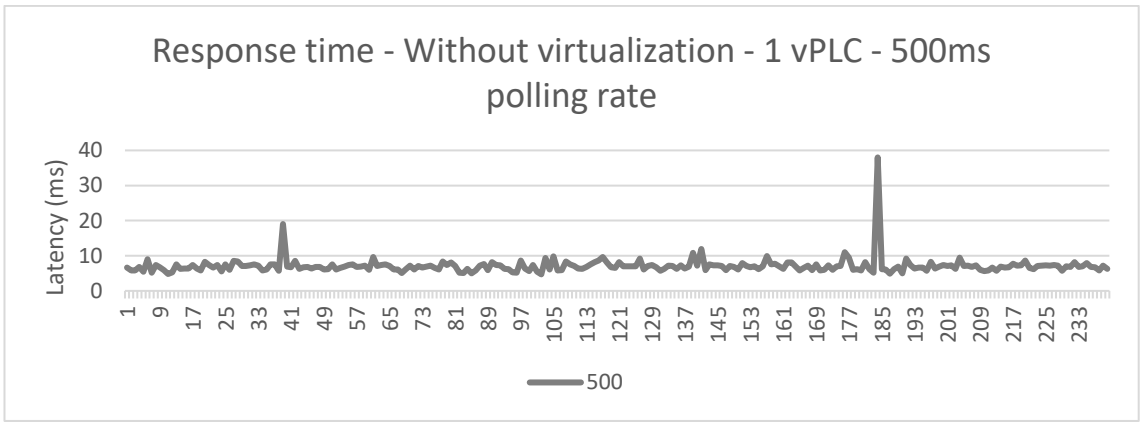
[96] Tiago Cruz, Paulo Simoes and Edmundo Monteiro. 2016. Virtualizing Programmable Logic Controllers: Toward a Convergent Approach.

APPENDICES

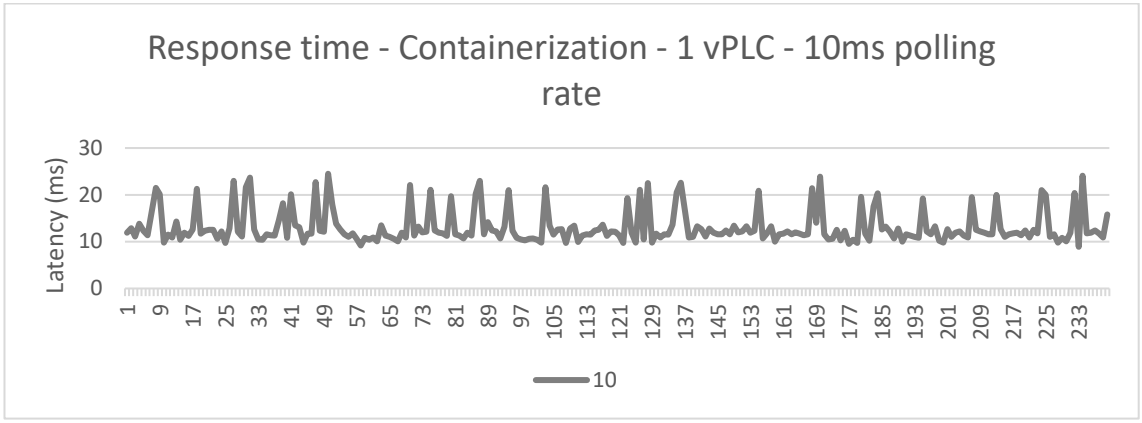
Attachment 1. Response time – Without virtualization

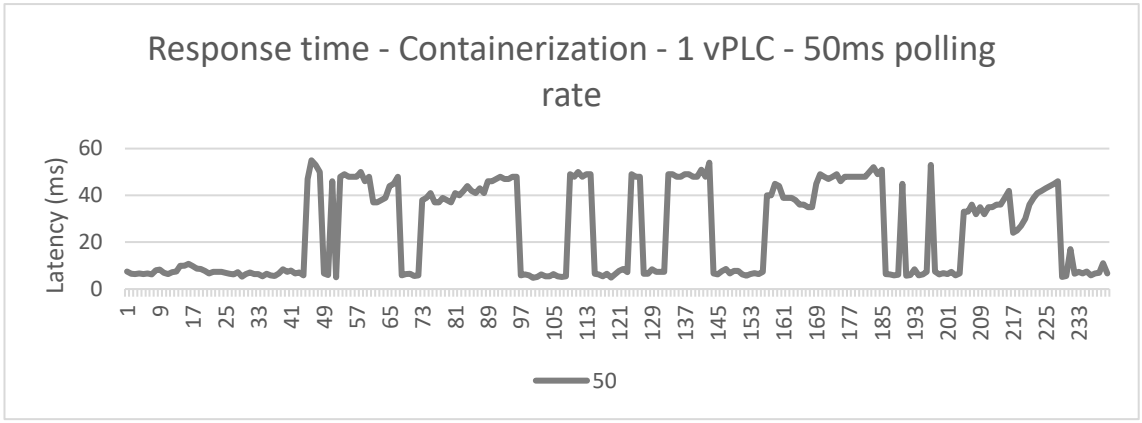


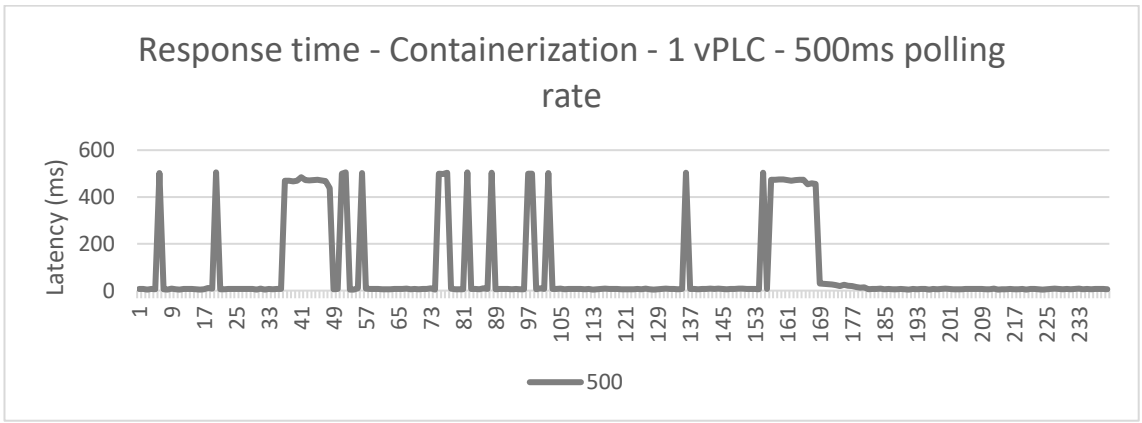




Attachment 2. Response time - Docker







Attachment 3. Response time - KVM