TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TEEMU MONONEN

CLOUD COMPUTING IN A MACHINE AUTOMATION APPLICATION

Master of Science Thesis

Examiners: prof. Jouni Mattila and prof. Matti Vilkko
Examiners and topic approved on 30th of August 2017

ABSTRACT

TEEMU MONONEN: Cloud computing in a machine automation application
Tampere University of Technology
Master of Science Thesis, 61 pages, 2 Appendix pages
October 2017
Master's Degree Programme in Automation Technology
Major: Process Automation
Examiners: Professor Jouni Mattila, professor Matti Vilkko

Keywords: Cloud computing, automation, Internet of things, sensors

Automation systems have evolved from local control systems to widely distributed, networked and complex beings. Distributing computing tasks to a number of individual computing units has changed the way automated systems function. Offloading demanding computing to nearly infinitely powerful cloud environments has introduced potential in reducing upfront hardware costs, system updating complexity and energy consumption. The use of cloud resources as a part of hard real-time machine control tasks has been researched in a number of studies. The use of the current cloud technologies has been found feasible in high-level supervisory control tasks, for example.

In automation systems, individual sensors and actuators can now have internet access (the Internet of Things, IoT), which enables data gathering to the cloud directly from the devices. In the cloud, vastly complex sensor data-based computing can be executed to gain insights of the automated system or to enhance its performance. This thesis is about creating an infrastructure for gathering sensory data to the cloud and enabling cloud computing in a machine automation application. The cloud resources are provisioned from the public cloud service provider Microsoft Azure and are studied from a functional viewpoint. As the focus is on the functionality of an end-to-end IoT system, intricate cyber security issues are out of the scope of this thesis. The designed solution components are selected and brought together in a case study involving a flexible hydraulic manipulator system and its local control unit. The communication with the cloud and the cloud computing performance were tested, providing information about the applicability of the cloud-based system.

In the tests conducted on the proposed system, the communication delays introduced by the wide area network between the local system and the cloud were between 40 and 60 milliseconds. Within this time period, a sensor data packet travelled over the network to the cloud, computations were performed on it and a confirmation message travelled back to the original sender. The obtained results also show that the designed system can support up to 500 Hz sensor data ingestion in the cloud. A cloud extension to an existing system could be made with a very low cost. For the system proposed in this thesis, the upfront hardware costs were about 30€. Additionally, about a 28€ invoice was paid monthly for the used cloud resources.

TIIVISTELMÄ

TEEMU MONONEN: Pilvilaskenta koneautomaatiosovelluksessa
Tampereen teknillinen yliopisto
Diplomityö, 61 sivua, 2 liitesivua
Lokakuu 2017
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Prosessien hallinta
Tarkastajat: professori Jouni Mattila, professori Matti Vilkko

Avainsanat: pilvilaskenta, automaatio, asioiden internet, anturit

Automaatiojärjestelmät ovat kehittyneet paikallisista säätöjärjestelmistä laajalti hajautetuiksi, monimutkaisiksi systeemeiksi. Laskennan jakaminen usealle laskentayksikölle on muuttanut tapaa, jolla automaatiojärjestelmät toimivat. Vaativien laskentatehtävien hajauttaminen pilviympäristöön, jossa laskentateho on lähes rajaton, on mahdollistanut etukäteen maksettavien laitteistokustannusten, järjestelmän päivittämisen monimutkaisuuden ja energiankulutuksen vähentämisen. Pilvilaskennan käyttöä paikallisen reaaliaikasäädön tukena on tutkittu ja sen käyttökelpoisuus mm. korkean tason valvovan säädön sovelluksissa on todettu monissa tutkimuksissa.

Nykyään automaatiosovelluksissa yksittäiset anturit ja toimilaitteet pystyvät yhdistymään internetiin (asioiden internet, Internet of Things, IoT), mikä mahdollistaa datan keruun suoraan laitteilta pilvipalveluihin. Pilvessä voidaan suorittaa hyvin monimutkaisia anturidatapohjaisia algoritmeja ja täten saada tietoa automaatiojärjestelmistä tai parantaa niiden toimintaa. Tässä diplomityössä luodaan infrastruktuuri koneautomaatiosovelluksen anturidatan siirtämiseksi julkiseen pilviympäristöön, jossa datalle suoritetaan laskentaa. Käytetyt pilvipalvelut valitaan Microsoft Azure –palveluntarjoajalta, ja niitä tutkitaan toiminnallisesta näkökulmasta. Työssä tutkitaan pilvipohjaisen IoT-järjestelmän toiminnallisuutta, mikä jättää datansiirron turvallisuusseikat osin aiheen ulkopuolelle. Luotavan järjestelmän osat valitaan käytettäväksi hydraulisesti toimivan taipuisan puomin ja sen paikalisen säätimen kanssa. Pilviresurssien kanssa käytävän datakommunikaation ja pilvilaskennan toimintaa mitataan, jolloin voidaan tehdä päätelmiä toteutetun järjestelmän käytettävyydestä.

Kokeissa havaittiin, että luodulla järjestelmällä internetin aiheuttamat kommunikaatioviiveet pysyvät pääosin 40 ja 60 millisekunnin välissä. Tässä aikamääreessä anturidatapaketti kulki verkon yli pilveen, datalle suoritettiin laskentaa ja vastausviesti kulki takaisin anturidatan alkuperäiselle lähettäjälle. Kokeissa saatujen tulosten perusteella voidaan myös sanoa, että luotu järjestelmä tukee 500 Hz datankeruutaajuutta pilveen. Pilvilaskennan hyödyntäminen olemassa olevassa automaatiojärjestelmässä voitiin aikaansaada pienin kustannuksin. Tässä työssä esitellyn järjestelmän laitteiston hankintakulut olivat noin 30€. Lisäksi käytetyistä pilvipalveluista koitui noin 28€ kuukausikulut.

PREFACE

This thesis project was a great chance to learn about the emerging technologies in automation systems. I would like to thank professor Jouni Mattila for this opportunity.

I would like to thank David Hästbacka for the advice on the subjects in this thesis. A special thanks to all my colleagues at the Laboratory of Automation and Hydraulic Engineering for creating a positive, supportive working environment.

I greatly appreciate the support I've received from my friends and family throughout my studies at the university.

Tampere, 23.10.2017

Teemu Mononen

CONTENTS

LIST OF ABBREVIATIONS

ACL	Access Control List
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
AWS	Amazon Web Services
BES	Batch Execution Service
CaaS	Control as a Service
CAN	Controller Area Network
CLI	Command Line Interface
CoAP	Constrained Application Protocol
CPS	Cyber-Physical System
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
FC	Functional Component
FG	Functional Group
GUI	Graphical User Interface
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IEEE	Institute of Electrical and Electronics Engineers
IHA	Intelligent Hydraulics and Automation
IMU	Inertial Measurement Unit
IoT	Internet of Things
IP	Internet Protocol
LAN	Local Area Network
MQTT	Message Queuing Telemetry Transport
MTU	Maximum Transmission Unit
NIST	National Institute of Standards and Technology
NSG	Network Security Group
PaaS	Platform as a Service
PC	Personal Computer
PLC	Programmable Logic Controller
QoS	Quality of Service
RaaS	Robot as a Service
RAM	Robotics, Automation and Mechatronics
RAPP	Robot Application
REST	Representational State Transfer
RRS	Request-Response Service
RTT	Round-Trip Time
SaaS	Software as a Service
SCADA	Supervisory Control And Data Acquisition
SDK	Software Development Kit
SLAM	Simultaneous Localization And Mapping
SPI	Serial Peripheral Interface
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
uint	Unsigned Integer

URI	Uniform Resource Identifier
VM	Virtual Machine
VPN	Virtual Private Network

1. INTRODUCTION

Throughout their history, automation systems have been local systems with sensors, actuators, controllers and communication between these components. In its early days, sensor data-based computing and actuating was performed on a single computing unit situated near the controlled process. Data communications took place using parallel cabling before dedicated fieldbus systems were developed. The research and development of automated systems led to distributed computing systems, where a number of local computers handled individual tasks to reach a common goal. As the systems grew more complex, the amount of data exchanged over the communication networks started increasing beyond the capacity of the fieldbuses. The geographical distribution of nodes communicating with each other grew wider as plants and systems increased in size. This development required adjustments to the communication systems and led to the modern fieldbus systems and Ethernet-based solutions. The introduction of Ethernet communication in the field devices brought up the idea of connecting those resources to a wide area network. [65][66]

A new paradigm in automation engineering is to take advantage of internet connectivity in simple devices, such as sensors and actuators, to acquire vast amounts of data that is used to orchestrate the way a site functions [6][21]. This connectivity of devices is called the Internet of Things (IoT). Using the IoT design, the distributed computing assets can be moved completely off the premises of the physical systems, offloading applications like monitoring, control and optimization away from the physical devices and the site itself [10]. This network-based data consumption opens the possibilities of virtualization, remote control rooms and Big Data analytics as well as plant distribution.

The emergence of IoT has been adopted in every field of automation, including machine automation and industrial robotics. Machinery used in various fields such as forestry and mining are mostly human operated, but carry an increasing amount of automated systems to assist the operator. Industrial robotics are almost completely autonomous, with a vast level of automation. Introducing remote intelligence based on such machines' operational data transported over a network connection enables the remote monitoring of massive amounts of assets. Thanks to IoT, the monitored aspects can be specified to concern each actuator on the machine, which will enable e.g. predictive maintenance. Such solutions have already been applied to mobile work machinery by John Deere, for instance [24].

To extend the distributed system architecture even further, computing is offloaded to a vastly scalable, ubiquitous, on-demand computing infrastructure available over an Internet connection. This kind of computing resource is called *cloud* – a buzzword in modern

technology design. Using cloud computing, a vast amount of expensive computing hardware can be replaced with cloud services running in a remote hardware infrastructure possibly provided by a third party. This development can improve system monitoring and actuating as well as business intelligence by allowing remote access to these resources.

Companies have been using cloud computing mostly for their business intelligence and data storing. These tasks don't require extreme timeliness or low data transport latency and can thus be offloaded to geographically distant computing resources. For time-critical tasks, some have constructed their own private clouds situated near their physical automation systems. The communication delays over such short distances are negligible and enable the cloud to handle more time-critical tasks. No solutions using cloud resources thousands of kilometers from the physical system for autonomous control tasks have been adopted to the industry. Some research on the subject has been conducted (e.g. [18][27]), but most systems in actual use are based on near-by cloud infrastructures.

Up to date, there has been a plethora of research in the fields of Internet of Things and cloud computing. The amount of papers and journals published regarding these topics is increasing, as more applications and improvements to current solutions are presented. These fields are being developed rapidly, resulting them in leaping towards their full potential.

1.1 Problem statement

In automated systems, control algorithms don't usually require vast amounts of computing power, but e.g. some modern system state reconstruction algorithms may be too demanding for local real-time controllers. Offloading such algorithms to a cloud environment, where the underlying physical computing infrastructure's power is close to unlimited, provides a platform for compute intense computations. The public cloud service providers offer exactly that, with prices significantly lower than those of powerful computing hardware. Utilizing the public cloud resources as a part of local control system enables the usage of less expensive local hardware that communicates with the cloud resources to reach a common goal. This way, a control system architecture can be designed to have time-critical tasks executed in a local controller and demanding, less time-critical tasks in the cloud. However, configuring such systems from scratch is a tedious task.

The problem with using a cloud-based solution in time-critical tasks are the delays and uncertainties in the communication between the local system and the remote cloud assets introduced not only by the data transport medium and topology, but the selected technologies and architectures as well. While a valid way of reducing upfront costs and energy consumption, the overall performance of a cloud-extended solution has to meet the requirements of a given application. In a cloud-extended sensor data based automation solution, at least the following criteria should be met:

- Low communication latency from the physical system to the cloud and back,
- Support for the amount of data produced by the sensors,
- Sufficient computing power in the cloud computing environment.

The vast amount of ways to deploy such solutions can be overwhelming. Communications, cloud services and local infrastructure can be designed in various ways. Using these design criteria, most of these options are eliminated through technology review and further criteria will help pick the best suited ones.

1.2 Thesis goals

This thesis was a part of a research project at the laboratory of automation and hydraulic engineering at Tampere University of Technology. More specifically, at the former department of Intelligent Hydraulics and Automation (IHA). The project topics were flexible structure modeling and control, and extending parts of such systems to a cloud environment. The researchers at the department of IHA are focused on local control design of fluid power machine automation, not cloud technologies. They wish to offload high-level system control and supervision to the cloud in their future research. This thesis provides a look at some of the viable technologies to be used in cloud-extended machine automation and a solution tailored for the requirements of an experimental machine automation system.

The main goal of this thesis is to design and evaluate a solution for sending sensor data to a public cloud environment, where a cloud computing platform is provisioned for data-based computations. The computing results could be then sent back to a local system. Microsoft Azure was selected as the cloud service provider because of the amount of services and tutorials they provide. The system was designed for an experimental flexible beam control scheme studied by Mäkinen [52]. This case system is a good example of a machine automation application with a hydraulically actuated manipulator. Inertial measurement sensors were mounted on the flexible beam manipulator to measure the beam acceleration and angular velocity at different lengths. This data was sent to the cloud, where an algorithm was executed on the data. In the context of this thesis, the algorithm only acted as a computational load, since the main focus is on the functionality of the cloud-extended part of the system. Near real-time cloud supervisory control is the main application the system created in this thesis can be used for. A similar premise has been researched e.g. by E. Goldin et al. [19] and D. Coupek et al. [14].

As the focus is on the functionality of the cloud-extended system, the data communication and computing are presented from their performance viewpoint more than their architectural perspective. A number of system criteria are defined to be used in the research and design process. To get to the thesis goal, the ways of establishing the sensor-to-cloud connectivity and a cloud computing platform are studied and compared to determine which ways can best satisfy the defined criteria. This comparison is presented in this work

to provide the reader with a sense of the strengths of different technologies. Intricate system and data security details are out of the scope of this thesis, but must be addressed when designing a system for production purposes.

Two main research questions about the designed system's performance and applicability arise. The first point of interest is how quickly the proposed public cloud-based system can react to sensor data, i.e. how long the data round-trip time over the public Internet is using the selected architecture. This knowledge will determine the feasible timing requirements of the tasks performed in the cloud. The next point of interest is the amount of data sent per second the designed system supports, i.e. how large the data send frequency or the amount of connected sensors can be. This quantity indicates the size of the sensor network this design is feasible to carry, as well as the supported amount of data produced by each sensor.

This thesis does *not* implement a system that can be used in mobile machinery in rough environments. The point of the study is to create a cloud computing system in laboratory conditions to study its performance without constrained networks.

1.3 Thesis outline

Chapter 2 introduces the concepts and technologies related to the goal of this thesis. References to previous studies are made to present the current state of the mentioned concepts.

In chapter 3 the use case of the cloud-based system is described. Based on the case, detailed design criteria for each distinct architectural part of the solution are defined and a number of potential technologies are compared. Technologies are selected based on the criteria. Then, the system components are designed in detail. The algorithms deployed in the cloud are not presented.

Chapter 4 is about setting up a testbed for the developed solution and presenting the main test results. The accomplishments and shortcomings of this thesis are summarized and future work suggested in chapter 5.

2. BACKGROUND

In this chapter, the background of the thesis topic is reviewed. With the goal of the thesis in mind, the relevant topics have to do with automation, Internet of Things and cloud computing. The benefits and drawbacks of using cloud technologies in automation applications are introduced along with some of the recent research on the topic.

2.1 Traditional automation architecture

Automation technology aims to design control algorithms and communications solutions for machines and plants to enable their autonomous operation. Current automation systems are monitored by human operators, but a vast amount of tasks have been successfully automated. The traditional architecture of an automation system can be illustrated in layers such as in figure 1. The different layers in the system architecture are field, control, plant management and enterprise level.

On the lowest level are the sensors and actuators that interact with the physical system. These devices gather data from the process and conduct actuating based on control signals from level L1, and are typically located on the plant floor level (field). The control level contains the real-time controllers that execute algorithms that aim to produce the desired behavior of the physical system by sending control signals to the field devices. These controllers are situated on level L1. On L2 are the human-machine interface (HMI) and supervisory control and data acquisition (SCADA) devices. These components are used to supervise the automated system and set the desired parameters. Level L3 holds the plant management applications and services. They are used to orchestrate the different subsystems in such a way that the entire controlled system produces the desired output. E.g. a manufacturing line is set to produce a certain amount of products in a timeframe by assigning the operation frequency for each part of the line and synchronizing all of the individual parts. Finally, L4 homes the enterprise applications that set the goals for the plant based on profit goals. These goals are met by the plant management orchestrating the physical system's performance accordingly. [21]

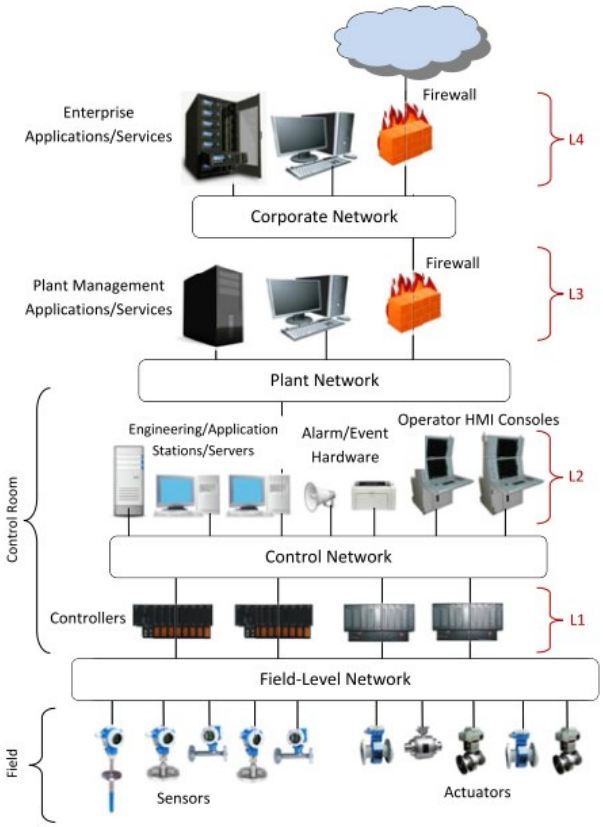


Figure 1. The architecture of automation systems [21, © 2014 IEEE].

The traditional approach contains communication between different levels. The communication takes place in a well-restricted intranet that is only used by the site of the system. This denies access from unauthorized users and provides security to the system. Data from each level is transported within the site and none of the low-level process values are shared via public connections. As can be seen from figure 1, the communication takes place over a number of networks designed for certain operations and applications. Dedicated, topographically unique networks are involved in the field level, control level, plant level and enterprise level [76]. Ethernet-based communications take place between PCs at higher levels of the architecture. The field devices and controllers use different mediums for data transfer, e.g. controller area network (CAN) cabling, allowing extremely low latency communication. However, industrial Ethernet [76] is a widely researched technology that allows sensors and actuators to communicate and get power over a single Ethernet cable.

The data analytics and storing takes place in the enterprises physical resources such as servers, computers and storage archives. The hardware requirements can be vast, including server rooms, computing units and extensive wiring. Monitoring and control of plant-wide systems takes place in control rooms situated near the process machinery. HMI and SCADA are operated on-premises and require controller personnel to work at the plant.

In automation systems, certain tasks have requirements for the time they're allowed to take. The timing requirements can be categorized in hard real-time and soft real-time based on the time it takes before a task outcome becomes useless. Figure 2 shows the difference of the two. An example of hard real-time tasks is system control in the control level, where the controller has to react to changes in the system at subsecond rate or the system can't be controlled properly. Other tasks, such as historic data acquisition and storing, are not as time critical, since delays in such tasks don't affect the system performance and history data will be relevant after a considerable period of time.

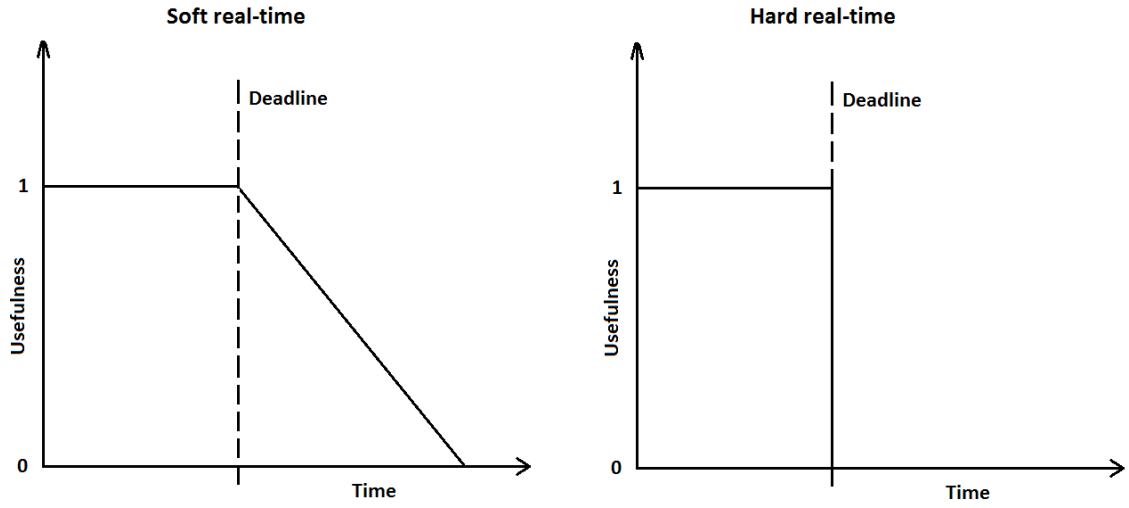


Figure 2. The difference between soft real-time and hard real-time requirements.

In figure 2, a task's outcome usefulness reduces after a predefined deadline. With soft real-time requirements, there is some use for the outcome after a certain amount of time after the deadline. For hard real-time, the outcome loses all usefulness after the specified deadline.

In this thesis, time-criticality means a task is executed timely and quickly. A timely execution of a task is performed within a certain time window. When this window is small and starts at subsecond range, the execution is quick. For example, a soft real-time task can be assigned a time window such that the task is to be executed between 100 and 200 milliseconds. When the task is always finished within this window, the time-criticality requirement is met. A very time-critical task has a small window in millisecond range. For example, a hard-real time task can be assigned a window between 0 and 2 milliseconds, making it very time-critical.

2.1.1 Machine automation

Machine automation is a branch focused on automated functionalities present in mobile and work machines, such as forest harvesters, load handling equipment and mining machinery. These machines are mostly controlled by human operators, but are becoming

more autonomous and robot-like [62, p. 1066]. In the future, the work machines can be completely autonomous. In most cases, such systems are hydraulically actuated using pumps, valves and cylinders. Machine automation is based on the same principles as robotics, where the kinematics of joints and links between them govern the system design. In mobile machinery, the low-level automation components are onboard along with their controllers and a SCADA system. The enterprise level is not integrated on board but instead is handled at a remote headquarters. The data gathered from these machines is vital in optimizing the business aspects, detecting faults and developing the machinery and control [54].

Machine automation low-level control is indeed a hard real-time task. As most of these systems are actuated with control valves, whose dynamics are fast, changes in the controlled actuator can happen in a range of milliseconds [62, p. 188]. This sets the sample time for a control algorithm smaller than the response time of the actuator. This means the controller is required to execute the control algorithm and apply a control signal to the actuators within the sample period. When it comes to supervisory control, that handles higher level online calculus, the real-time requirements can be less strict. A small delay in delivering a command to the real-time control system doesn't jeopardize the control performance, although it might not result in optimal control.

In machine automation, sensors are to acquire measurements used to determine the state of the hydraulic circuit components and the position and speed of the hydraulically actuated manipulators. Oil flow rate, pressure and temperature are some of the measures that indicate the hydraulic circuit state. The manipulator position can be determined using measurements from hydraulic cylinder positions, manipulator acceleration and angular velocity. One of the most popular real-time communication systems in machine automation is CAN, that is used to transport measurement data from the sensors and control signals from the on-board controllers to the hydraulic actuators.

2.2 Internet of Things

The Internet of things has been receiving an increasing amount of attention from the engineering community [6]. L. Atzori et al. explain the meaning of the IoT, proposing that IoT is a worldwide network of things with unique resource identifiers [2]. Essentially, IoT describes a network of devices with Internet connectivity enabling their communication between other such devices, services and users. Internet of things promotes the connectivity of very simple devices, such as sensors.

The industrial scene is changing towards Internet connectivity and Big Data analytics [6]. The amount of interconnected things in industrial plants is steadily increasing as the IoT paradigm is gaining more ground. Analyzing the massive amounts of data produced by all the components in a given system will unveil intelligence that was never before accessible. This promotes plant optimization, predictive actions, remote monitoring and loads

of other applications that can make a process more productive. With the concept of IoT, data-based intelligence can be moved from the application premises to remote locations as data transmitting is no longer bound to the premises, but can be done over the internet. Controlling the connected simple devices from distant locations over the internet connection is now becoming simpler.

Connecting industrial devices such as sensors, controllers and actuators directly to the Internet poses a serious threat of cyberattacks and connectivity failures [51, p. 11]. An attack on devices that are in charge of any physical machinery could result in life threatening situations. Also, losing connection to such devices could also lead into a dangerous outcome. Therefore, security and redundancy are imperative issues when adopting IoT paradigm to production applications. Providing long lasting energy to portable devices can also prove problematic in rough environments [51, p. 11].

2.2.1 IoT reference architecture

The internet of things reference architecture is a model describing the typical parts of an IoT solution. There are a number of views of the IoT reference architecture, including information view that specifies entities involved in information flows and the information structure. IoT solutions are usually very complex and require all of these components to be configured. In this thesis, the focus is in the functionality of such system. Thus, the more interesting architectural view is the functional view, even though the information processes are considered when designing the system.

The functional model of IoT reference architecture is presented in figure 3 as depicted by Bauer et al. [3, p.168]. This model is an abstract proposition divided into functional groups that describe the several functions in an IoT system. The model can alleviate the design process of an IoT solution by presenting a guideline and a description of functional elements in such process. The model doesn't introduce the interactions between the functional groups, but can be used to visualize the data and action flow through the groups from an IoT device to the application using the IoT service. Much like reference architectures usually do, this one often doesn't describe one's solution's structure one-to-one. The parts of the reference architecture to be used in one's solution depend on their requirements. [3] Seven elements between the user's devices and the application where the IoT system is used can be derived from the model. These elements are called functional groups (FGs) and are: the IoT devices, management, service organization, IoT process management, virtual entity, IoT service, security and communication. Each functional group consists of a number of functional components (FCs), that further describe the tasks involved in each group. In this thesis, we're not focusing on all the functional groups presented in this model. The interest is in communication, the IoT service and security on an elementary level. The groups of interest are highlighted in green in figure 3. The groups outside the highlighted area are not discussed further.

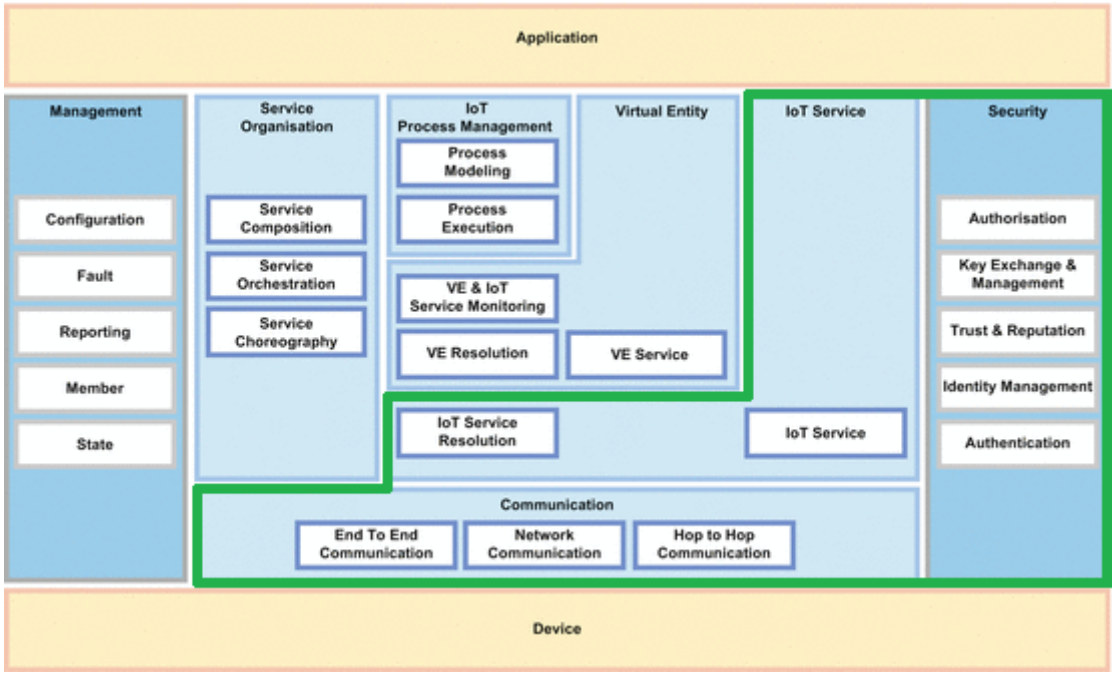


Figure 3. The functional IoT reference architecture and its parts included in this thesis [adapted from 3, p. 168].

IoT service takes care of collecting information from connected IoT devices and sending control messages to actuate or configure such resources. IoT service is discoverable and contactable by a user, whether they're human or devices. This functionality is realized using resource identifiers provided by the IoT service. [3, pp. 172-174] The IoT service resolution FC handles resource discovery and naming. In small-scale scenarios, such as in this thesis, the services are configured in advance to know the available IoT devices. In large scale, the necessary devices must be found by the IoT Service. [3]

The communication functional group describes the interfaces used to connect to the IoT service. The connected devices send their data frames over the Internet through hop to hop communication. The communication between participants is established by the network communication functional component using resource locators and the gateways. Transportation and translation is handled in end to end communication functional component. [3, pp. 174-176]

The security FG handles the IoT solution's security and privacy. The key parts in providing these attributes are authorization, key exchange, trust, identity management and authentication. Using access control policies, authenticates and known user identities the system can avoid cyberattacks and data loss. [3, pp. 176-178]

2.2.2 Communication patterns

In network communication, there are different ways of configuring the data exchange between participants. The data is communicated using a certain messaging pattern that

specifies the types of messages sent between participants. The main communication patterns in IoT systems are request/response, publish/subscribe and push.

Communicating resources are named in certain ways so that they're discoverable by other participants. In IoT solutions, the common resource naming methods are internet protocol (IP) addressing and uniform resource identifiers (URI). IP addressing is based on labeling resources with numerical identifiers. It provides the host interface identification and location addressing, i.e. where the resource is in the web. [74] Where IP address is numerical, URI is based on letters and words. The developer creates the URIs in their application programming interface (API) in such a way that they're intuitive to use in the code where the service calls are made. The URI formulation is as defined by Microsoft [33]:

```
{URI-scheme} :// {URI-host} / {resource-path} ? {query-string}
```

URI scheme defines the used protocol, such as hypertext transfer protocol (HTTP) [73]. URI host is the domain name or IP address of the service hosting server. Resource path describes the called resource or a collection of resources at the service. Query string holds the additional parameters to specify e.g. the resource selection criteria or the API version. [33]

In the request/response pattern, a participant sends a request to the other(s) and awaits for their response [3, p. 188-189]. The request contains the action which the sending participant wants the receiver to perform. This communication pattern is mostly used in HTTP messaging, and is very common. The communicating participants are given URIs that are used to direct the messaging. The first participant sends a HTTP request message to the other, that services the request by sending a response. The basic HTTP requests are PUT, GET, POST and DELETE. Using these, data can be exchanged between participants in various ways. A popular HTTP-based communication architecture is called REST (representational state transfer). RESTful communication has been mostly used in computer-to-computer communication and web service implementations, but has gained popularity in IoT systems too.

The publish/subscribe pattern is based on topics of the message contents. The communication consists of data subscribers and publishers. A data subscriber only receives data from certain topics they're interested in, i.e. subscribed to. The data they receive is originated from publishers that produce data under certain topics. Such topics in IoT systems can be e.g. different sensor readings, such as temperature or acceleration. This pattern contains a message broker that forwards the data from publishers to the subscribers based on the topics. The advantage in publish/subscribe pattern is its scalability: the communicating participants don't need to know the identifiers or the existence of each other, which means they need no additional information if the amount of communicating participants increases. This communication pattern is well-suited in communication between a vast, changing number of participants. [49]

In the push pattern, a communication participant continuously listens for data originating from a well-known, predefined set of data producers. This pattern is used in situations, where the communicating participants remain mostly the same. An example of push pattern is a computing server listening to streams of data generated by a sensor, whose resource identifier is well known by the server. The sensor pushes data to the receiver without other types of message exchange required. [3, p. 188] Unlike the publish/subscribe pattern, push communication isn't easily scalable as communicating participants need to know the resource identifiers of each other. However, it is very suitable for high frequency data acquisition.

2.2.3 Communication protocols

Communication protocols define the structure of data packets, the communication pattern and possible error handling mechanisms. In IoT, there are specially developed protocols providing simplicity and light-weight communications for constrained devices. The more common network communication protocols are also used, when they're appropriate.

TCP

Based on the internet protocol, Transmission control protocol (TCP) is an end-to-end communication protocol based on IP addresses of the end nodes. The protocol includes initialization of the connection between the participants, data transmission and connection termination. The pattern of communication is push. To establish a connection, the first participant sends an initial message to the listening participant, who confirms the connection via another message. The first participant sends an acknowledgment to the other, and data exchange can begin. [8][75]

During the data transfer stage, participants send acknowledgment messages to each other for every data sequence they've received. If the sequence isn't followed by an acknowledgment, the data hasn't reached its destination. In that case, the lost sequence is retransmitted to the receiver. This procedure makes TCP a reliable protocol, when it comes to making sure the data reaches the intended receiver. [8][75]

TCP can also limit the frequency at which a data receiver receives packets from a sender. This is called flow control, and it is a mechanism to ensure the receiver can handle the data incoming at a high frequency. [75] This can reduce the allowed sensor send frequency in an IoT solution. Combined with the message retransmission, flow control has a negative impact on the data transmission time-criticality.

TCP takes a stream of data and divides it into packets. Each TCP data packet consists of a header and a payload. The header contains information about the sequence source port, destination port, data integrity and the sequence number. The size of a typical TCP packet header is 20 bytes. The payload contains the data to be transferred, e.g. sensor readings.

UDP

Like TCP, user datagram protocol (UDP) is based on the internet protocol. UDP is described as an unreliable protocol because of its connectionless nature. Where TCP initiates a connection between two participants, UDP sends datagrams to the specified IP address and port of a receiver without establishing a connection. There is no acknowledgment for any packages, retransmission or flow control, which makes UDP a low-latency protocol suitable for sending a high frequency stream of data. [23] This can create performance requirements for the data receivers and the network performance. The communication pattern using UDP is push.

The header of a UDP packet is 8 bytes long. It only contains the destination IP address and port as well as a checksum for data integrity and the packet length [23]. This usually means that most of the transferred data contains the payload, meaning UDP uses the network bandwidth more efficiently than TCP.

Being a simple protocol, UDP is widely supported by simple devices. The only requirements for UDP communication is an IP address, network connectivity and network socket support.

MQTT

Message queue telemetry transport (MQTT) is a lightweight machine-to-machine protocol. It works using the publish/subscribe pattern. As mentioned in chapter 2.2.2., this results in high scalability, as connected devices don't need to know the identifiers of all the other clients to whom data is to be sent [64].

MQTT is based on TCP, which means the connection initialization and data transport acknowledgment is included. The protocol is designed for devices with limited resources, such as battery life or computing power. However, there is a need for configuring a message broker, which introduces an extra entity to the communication scheme.

The MQTT data packet header is only 2 bytes long. It contains the description of the packet type and flags of the type, e.g. connect, publish or subscribe. [55]

COAP

Much like MQTT, the constrained application protocol (CoAP) is developed for simple, resource constrained devices. It is based on UDP and has a similar URI-based, HTTP request compliant communication scheme as REST. CoAP is connectionless and unreliable similar to UDP, which may lead into messages not reaching the receiver or messages arriving in wrong order. A lightweight reliability mechanism is implemented in CoAP to reduce such behavior. [60] The communication pattern is request/response.

CoAP supports time-critical data transfer and high frequency data streams. In addition to functionalities of UDP, CoAP provides confirmable and non-confirmable messages to provide transport success for certain messages, while enabling high frequency data transmission.

2.3 Cloud Computing

A term heard more and more frequently in today's engineering community is cloud computing. It is a general term for describing virtualized web-based computational resources maintained and offered by a cloud service provider. These resources are accessed through a cloud access device with an internet connection, such as a PC or a smartphone. So instead of owning computing hardware, a cloud service client can access computing power using a far less powerful device [57]. The computing takes place in one or more servers situated in one of the service provider's data center locations. The National Institute of Standards and Technology (NIST) defines cloud computing in [28] as follows:

> *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [28]*

The term includes data storage, routing, computing, web-based applications, gateways and many other services that process the workloads that traditionally are handled on-premises. For businesses, the motivation for using cloud computing is to distribute the on-premises infrastructure to a virtual environment creating savings in hardware costs and enabling data access, storing, sharing and analysis within and between sites. The computing resources in the cloud can be accessed by applications that run in the cloud or on local systems [78].

Cloud computing can be seen as *virtualization* of a physical computing infrastructure. By definition, in virtualization a single hardware stack is used to run multiple operating systems or virtual machines that share the processors and memory of the physical device. The benefit of this technique is the ability to run multiple tasks in multiple operating systems to take full advantage of the physical hardware. The drawback of doing so is that you introduce a single point of failure and set a high demand for computing power. The overall performance may also get reduced as multiple processes run on a single hardware stack. [9, p. 164-186] With cloud computing, the underlying infrastructure is very vast and powerful, which rids these drawbacks. Another key difference between the two technologies is the elasticity of cloud computing. This means that the used computing hardware can be scaled up and down fast and easily depending on the requirements at a given time. Also, the way cloud computing services can be provisioned is different from that of virtualized assets. Cloud computing supports automated and self-service provisioning,

while in virtualization, providing clients with services requires manual work from the providers [9, p. 180].

There exists a number of technologies with similarities to cloud computing, such as cluster computing, parallel computing and distributed computing [9, p. 1-8]. All of these technologies include a group of connected computers or central processing units (CPUs) working to achieve a common goal.

2.3.1 Cloud types

The four cloud environment deployment models are public cloud, private cloud, hybrid cloud and community cloud. The different models describe the purpose of the cloud and have significant differences that will affect the applications that are feasible to run in each model.

Public cloud is a cloud environment open for public usage. Anyone can create an account in it to provision the available services, as they are available over a network connection. [28] The large public cloud providers host their services on datacenters spread around the world in different locations for increased availability and reduced latencies. The main advantages of public cloud are the ease and speed of provisioning and configuring the offered IT services [56]. Due to the size of the public cloud, computing resources can be scaled up and down on demand with very little time spent [56]. The resources are vast and the amount of available computing power close to unlimited. Public clouds use a pay-as-you-go pricing model where the customer only pays for the resources they use. Some resources are only billed per use, which makes running small and non-frequently used applications in a public cloud an affordable solution. In chapter 2.3.2, a number of public cloud service providers are introduced.

A private cloud is completely dedicated to a single business. Unlike public cloud, private cloud is only accessible by the users that it is provisioned to. The services can be hosted on service provider's datacenters or locally, which would reduce network latencies drastically compared to those of a public cloud. The level of security is greater in private cloud as well. The private cloud can be isolated from the public internet, which provides security but means that the computation resources must be located on the site. The resources can be scaled as well and control over the specific underlying components like CPUs and storage is greater. [56] The private cloud approach is suitable for businesses with strict security rules concerning their data and control over the infrastructure components.

Hybrid cloud connects the on-premises resources to those on the cloud creating a network where applications can share data and computational power. This enables the user to run certain parts of their applications on their physical hardware and others using the service provider's resources. Combining a local private cloud with public cloud resources is one

of the ways to create a hybrid cloud [28]. Some businesses have moved their computationally or memory intense applications to the remote cloud, where the resources can be scaled, while running their other applications locally. Data is transferred between the two domains in a secure manner and it is available for the users authenticated to view or modify it.

Community cloud is available for a number of organizations that have same goals and concerns. It is a similar concept to private cloud, but the amount of clients allowed to use the resources is greater and spans over several organizations. The infrastructure can also be hosted by a third party or the community using it. [7]

2.3.2 Service providers

A cloud service provider hosts computing services on their databases. Their customers can access the services via a network connection. Nowadays, there are over 200 cloud service providers, but three companies rise above the competition when it comes to public cloud service hosting: Amazon Web Services, Microsoft Azure and Google Cloud Platform. While other providers exist and advance their businesses, they fall short in regional availability, service scalability and thus popularity when compared to the three big players.

When using any of the three big providers, their cloud services are provisioned and maintained either using their web portal or a command line interface (CLI). A portal provides a graphical user interface (GUI) for service configuring. The GUI is useful in testing and monitoring, but for production the CLI is far more efficient. The user can use an application programming interface to interact with their cloud resources using only command line requests. Using the same API, resource provisioning and configuring can be automated by running them in a program code.

In many contexts, *Amazon Web Services* (AWS) is described as the sole market leader in cloud services hosting [26][29][59]. It was the first company to provide cloud services, in 2006 and has had time to mature their technology. AWS has 44 data centers around the world in all continents except for Africa and Antarctica.

Microsoft Azure, formerly known as Windows Azure until 2014, was released in 2010 [11]. It provides a variety of services that include similar functionalities as the ones AWS provides, thus making Azure their greatest competitor. The service categories include compute, mobile services, storage services, messaging, media services, data management and machine learning among others. Currently, Azure has 34 data center locations around the world in five continents: North America, South America, Europe, Asia and Australia.

Released in 2011, *Google Cloud Platform* is the youngest of the three main players in public cloud services providing [71]. The most popular services Google provides are

Google Compute Engine and Google App Engine. Google has 33 data centers, one of which is located in Finland.

An interesting, smaller service provider *Ortelio* offers cloud services specifically designed for robotics. Their Robot Application Platform (RAPP) provides e.g. image and voice recognition services. In the future, RAPP could host simultaneous localization and mapping (SLAM) services for robots with internet connectivity. [58] Some of the other cloud service providers include IBM, Rackspace, Joyent, Aruba and Digital Ocean.

In this thesis, the cloud services are provisioned from Microsoft Azure. Due to the lack of documents and tutorials on using the small public cloud service providers, the provider selection was made between the three big companies. These three have little difference in the types of services they provide. The same basic components can be found in each of them and in some cases the choice between the providers comes down to user preference. Google is the only service provider with a data center in Finland, which could enable low-latency data transfer. However, Google Cloud Platform doesn't offer the same number of in-depth tutorials and documentary as AWS and Azure do, so the learning curve is too steep for the duration of this thesis work. The feedback from colleagues at Tampere University of Technology also suggests to pick AWS or Azure over Google.

Between Amazon Web Services and Microsoft Azure, the better service provider is difficult to differentiate. One could argue that AWS has been around longer and that it has had more time to mature their technology. However, Microsoft has done a good job responding to the competition by launching their own services and perfecting them. When developing solutions in the environment Azure provides, one can see updates occurring quite frequently. To differentiate between the two providers, we refer to B. S. Đorđević et al. [16], who conducted a performance comparison between AWS and MS Azure. The results suggest that Azure has a slight edge over AWS when it comes to CPU and disk intensive operations.

2.3.3 Service models

The cloud service providers offer services that can be divided into three main models: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Figure 4 presents a layered architecture of cloud services with some examples of each service model [78]. The managed resources are visualized on each layer. With each service model and example, the resources below the dashed lines are managed by the service provider.

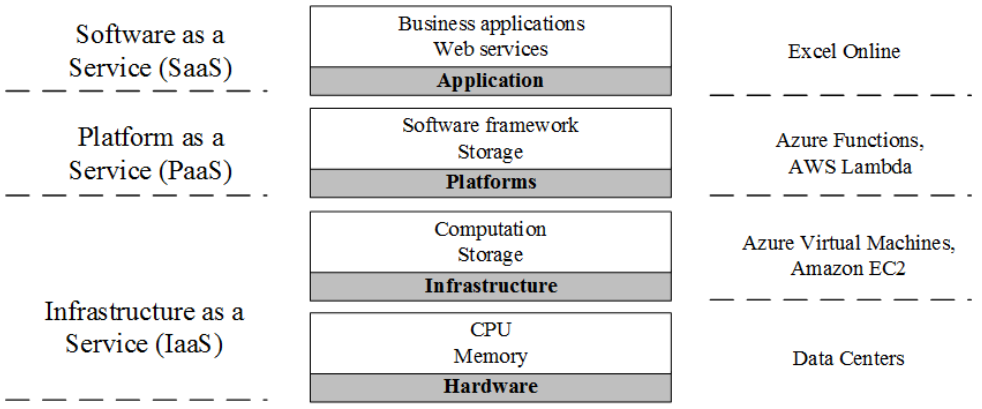


Figure 4. The layered architecture of cloud service models [adapted from 78].

SaaS provides the customer with fully maintained software services running in web environment. The customer manages only the applications they use, with the rest of the layers handled by the service provider. An example of SaaS would be Excel Online, where the user uses Excel's functionalities on a web browser without installing any software components on their device. None of the computing in Excel Online uses the user's resources, but the service providers' instead.

Platform as a Service provides the customer with the freedom to build their own applications running on the provider's infrastructure [9, p. 69]. PaaS enables high-level programming with reduced complexity, as the customer doesn't need to configure networks, servers or other tools needed to develop and maintain an application. For example, cloud service providers offer PaaS with cloud environment tools that are used for launching code or web applications in the cloud. The customer has control over the software framework (e.g. Java or Python) and the storage they use. For example, Microsoft Azure provides a service called Azure Functions that allows the customer to develop, test and run program code in a web browser code editor using a variety of programming languages they can select from [31]. The code can be used in numerous applications, like saving ingested data to a database for example. More on Azure Functions in chapter 2.6.3. Amazon offers a similar service called AWS Lambda.

Compared to PaaS, Infrastructure as a Service provides the customer with more freedom over the frameworks, storage and computing aspects they want to use in their solution. IaaS is an instant computing infrastructure [46] where the customer can select between servers, storage and networking while keeping control over the configuration and management of the tools they want to use. The underlying physical cloud infrastructure remains out of the reach of the customer, but the applications, operating systems and storage are controlled by them [28]. IaaS usually means virtual machines that run on the service provider's hardware. More on virtual machines in chapter 2.6.3.

As cloud technologies become subjects of more and more research, different kinds of service models have been proposed. For example, control as a service (CaaS) is discussed

in [18] with a case study using virtualized programmable logic controllers (PLCs). This paper experiments controlling an automation system using a controller deployed in a cloud environment. Robot as a service (RaaS) was proposed e.g. by Y. Chen et al. [10]. Computations required by an operating robot are provisioned as cloud services the robot can access over web requests.

2.3.4 Challenges

Cloud computing is a real trend technology at the moment, but there are problems that limit its feasibility in many applications. With public cloud, the network latency can't be predicted, which will have a negative effect on applications that rely on some real-time requirements [20]. Since the cloud services are only accessible over a network connection, latency and bandwidth bottlenecks can't be decisively tackled.

Another significant problem with cloud technology is security [15]. Certain applications need the transferred and stored data as well as the deployed applications to be out of reach for outsiders and cloud technology should answer to that requirement. For businesses that can afford to use private cloud, the issues are partially taken care of. But for those leveraging public cloud, some special measures may be needed to guarantee safe and secure data transfer and storing. Such measures include data encrypting, access authentication, identification and authorization [15]. To safely store sensitive information in the public cloud storage requires a trustworthy cloud provider with state-of-the-art security efforts.

The planned or unplanned service unavailability due to updates or repairing operations of the underlying cloud infrastructure is something to consider when using cloud technologies. Another reason for not being able to connect to one's cloud resources is network downtime, which is usually not caused by the actions of the cloud provider but rather the network provider. Because of these reasons, the cloud architecture for production solutions should be designed with redundancy in mind. Distributing the data center locations of different services is one way of reducing the risk of losing all your resources for an extended period of time.

The Quality of Service (QoS) a provider can offer will change according to the loads experienced by the servers. Other things such as load balancers and the amount of money one is willing to pay for the services also affects the QoS. Therefore, it is hard to place deciding values on a cloud service performance at a given time, since the time of the day and other factors will affect it.

2.3.5 Fog Computing

Another emerging concept in cloud technologies is called fog computing or edge computing. The main idea is to erase certain problems introduced by cloud computing by bringing time critical actions to the edge of the network. This means that some of the processing

is done in fog nodes: local devices that handle on-premises tasks and communications with cloud services. The network latency associated with cloud computing is no longer present in the applications deployed at the edge of the network. This enables a solution to provide real-time capability whilst taking advantage of cloud technology. Less time-critical actions can be deployed in a cloud environment, reducing the upfront hardware costs and required physical space. Fog computing is situated between the end devices and cloud computing data centers that are accessible over a network connection. [20][68]

The devices handling edge computing are often referred to as fog nodes. In some contexts, the terms edge and fog are defined as synonyms. Sometimes the fog is defined to consist of a separate infrastructure while edge computing refers to any single or multiple devices that perform operations at the edge of the network while also communicating with the cloud. From this point of view, fog computing is similar to cloud computing, but the underlying infrastructure would be situated near premises. Fog would then be available for service provisioning much like the cloud. [61]

The programming, infrastructure and workflows associated with fog computing haven't been mapped or standardized thoroughly yet. Thus, the concept is still quite flexible in its definition. However, offloading some of the computing tasks from the cloud to the edge offers an improvement in real-time capabilities and security, while the immense power of cloud computing can be harnessed simultaneously. Fog computing solves some of the problems certain time-critical applications experience when using cloud technologies, which makes it a useful paradigm in, for example, automation applications.

2.4 Cloud-based IoT

Cloud-based IoT takes advantage of the internet connectivity of simple devices and ubiquitous cloud resources. Cloud-based IoT solutions consist of two main aspects: device connectivity and cloud-based data processing. Additionally, data presentation and connectivity to business aspects can be considered. [48, pp. 3-4]

The IoT devices can connect to the cloud resources directly or via a field gateway. Direct connection to the cloud is for IP capable devices that can establish secure connections over the Internet [48, p. 8]. When the devices aren't IP capable or they use short distance communications (e.g. Bluetooth) or they can't be connected to the internet securely, they connect to the cloud using a field gateway. This gateway acts as a communication enabler performing e.g. device data preprocessing, filtering and batching. It can also send control messages to the connected devices. The field gateway is commonly situated in the premises of the connected devices and can be either a piece of software or a physical device. [48, pp. 8-10] Using a field gateway, a local network of e.g. sensors communicating using Bluetooth can send data to a nearby gateway that further sends the data to the cloud using long range communications.

In the cloud, data originating from IoT devices is collected by a cloud gateway. This component enables bidirectional communications with the devices either directly or through the field gateway. It is a service or a piece of software running in the cloud environment, providing the communications, directing the data to computing services and performing authentication of the connected devices. The cloud gateway has its own resource identifier, and is reachable over the public internet, a virtual private network (VPN) or a private connection. [48, pp. 10-11] Figure 5 depicts the simplified architecture of a cloud-based IoT solution.

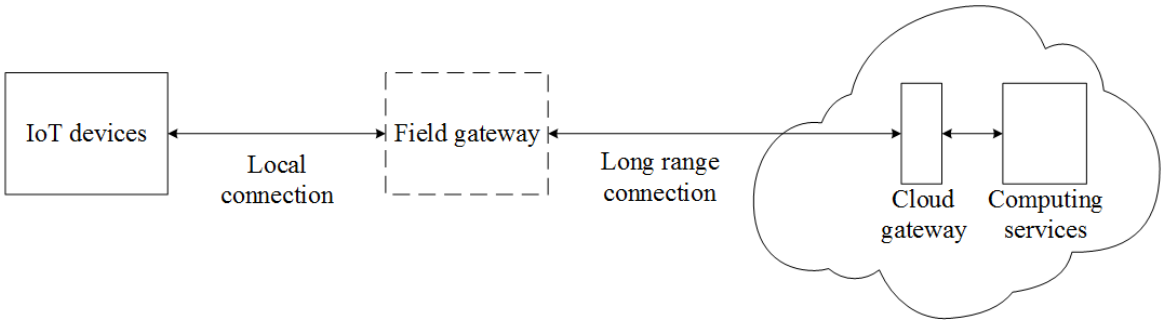


Figure 5. The simplified architecture of a cloud-based IoT solution.

When communicating over the public Internet, data is routed through communication hops such as servers and switches. Securing the connection is handled using authenticates and data encryption by the communicating participants. Problems in transmission time can occur due to other traffic in the network, as the network bandwidth is finite.

A private network is an enclosed network using a private IP address space that is not publically routable. A virtual private network is a virtualized private network that extends over a public network [30]. A VPN connection between a restricted network and a remote user creates a secure tunnel between them encrypting all the exchanged data. In this sense, VPN mimics a private network as data streams can take place over the public network securely like they would in a private network. The resources connected using VPN can communicate like they would in a local area network (LAN).

A VPN connection is established using VPN gateways. They act as virtual network gateways that send encrypted traffic across a public connection passing, blocking or routing VPN traffic. Usually, such a gateway is a physical router device, but it can also be a firewall or a server.

VPN comes with constraints related to the encryption of data. Devices at the both ends of the network must use CPU resources to encrypt and decrypt outgoing and incoming data, which sets certain requirements for the devices. As the size of the data increases, the encryption process becomes more demanding. Thus, for applications sending large amounts of data, VPN can decrease the performance.

A private connection to the cloud resources is a secure way of connecting one's IoT devices. The connection is completely separated from the public Internet, which provides a more reliable connection and reduces the latencies over the connection [37]. This connection type is used in private clouds, where the resources are at close proximity. In public cloud solutions, a private direct connection can be provisioned in cooperation with an Ethernet provider. Such a connectivity requires a dedicated cable, and establishing such a connection is very expensive compared to public Internet and VPN connections. For example, Microsoft Azure's direct connection services cost from 250€ to 43,261€ per month [38], whereas Azure's VPN costs only from 30€ to 720€ [44]. Public Internet connection costs include only the Internet service provider's costs.

2.5 Automation and Cloud Technologies

Combining the connectivity of IoT devices and cloud computing results in widely distributed systems, where data analytics can be performed off-site. In automation systems, this approach reduces upfront hardware costs and increases energy efficiency. Compared to a traditional, on-premises system, scaling and manipulating a cloud-based solution is much more agile and requires far less changes in the physical hardware. The trend of Internet-based, smart automation systems is widely referred to as the fourth industrial revolution [76]. Combining cloud technologies with automation systems requires extending the communications over a wide area network. While this can be done reliably using modern communication technologies, the physical distance between the cloud infrastructure and the local system can introduce communication problems.

Previous studies show that levels L2 to L4 (figure 1) can be successfully migrated to a public cloud environment [21]. Level L1, where the real-time control is located is difficult to deploy to a public cloud in a feasible way, because of the hard real-time requirements and no tolerance for major faults [21]. Level L1 also contains alarms and other safety measures that must work with extreme reliability. Exposing these systems to network latency and communication uncertainties could cost lives at the site.

On level L1 response times are measured in subseconds, whereas on level L2 they are measured in seconds [21]. On levels L3 and L4 the response times vary from minutes to hours, which makes deploying them as cloud services viable [21]. For now, when migrating level L1 to a cloud, the safety measures should be handled on-premises and some combination between edge computing and cloud computing is advised.

In [26] K. Lee et al., used Amazon's public cloud resources to extend a sensor network into the cloud. The study focused on gathering sensor data from remote devices to use in environmental monitoring and modeling applications. The study shows the agility and scalability of a cloud infrastructure in a sensor network application.

In control applications extended to a cloud environment, the network latency – if measurable – can be taken into account using certain control designs. Such is the case in [21], where the unpredictable varying delays caused by a network between the system to be controlled and the controller itself are compensated using an adaptive Smith Predictor. The paper shows that without such delay compensation, the system output experiences severe oscillation and eventually becomes unstable as the network delay increases. With the proposed compensator, the system response remains stable and saturates to its set point with network delays dozens of times longer than the sampling period.

The usage of cloud resources as a supervisory part of a local real-time control scheme has been studied in a number of papers. D. Coupek et al. [14] introduce a high-level cloud controller interacting with a local real-time control system to improve control results. In their work, the cloud controller provides the local assembly control system with optimal actuating policies. In their application the measurement data based cloud optimization algorithm is to be executed within 60 seconds. The system is not designed for subsecond reaction times from the cloud. A similar premise is studied by E. Goldin et al. [19]. They present an architecture for cloud-based online process model and control tuning with big data analytics using Amazon's cloud services. They implemented the architecture to a furnace system, where the system dynamics are extremely slow.

The study in [67] proposes an architecture to offload human-machine interaction (HMI) industrial automation tasks to the cloud through mobile devices. Since HMI systems handle tasks with different real-time requirements, the paper suggests the task processing platform to be selected based on those requirements. Hard real-time tasks would be processed on-premises, soft real-time tasks through mobile devices and tasks with more flexible timing requirements in the cloud. Furthermore, Nguyen et al. [53] propose a hybrid cloud based SCADA system. They conclude that using such design, the physical process data can be accessed from anywhere and the costs for maintaining or expanding the system are lower than in traditional, on-premises SCADA systems.

In machine automation, the use of intelligent devices can improve the data acquisition from mobile devices. Instead of collecting sensor data to a local storage on the machine itself, the data can be sent to a processing server location at the company's headquarters instantly. This reduces the amount of hardware to be carried on the machine itself. Using IoT, the data can be delivered to where intelligence and analytics take place. This way, the business can be optimized and faults can be detected while the machine operators work at their worksite. Of course, there are some major issues in network connectivity in, for example, forestry machines working where there are hardly any connections. The emergence of 5G networks may alleviate these problems [76], but work is still to be done to enable reliable connectivity to such machines.

2.5.1 Cyber-physical systems

In current automation solutions, the physical and computational resources are tightly interconnected creating an orchestrated, controlled system capable of autonomous operation and adaptation. These kinds of systems are often referred to as *cyber-physical systems (CPS)*. [77] CPSs contain embedded computing units, communications components, sensors and actuators. CPS applications range from autonomous vehicles to industrial automation [77].

The concept of IoT is closely present with CPS, as sensors can send their data over a network to a location nearby or far away. The integration of CPS, IoT and cloud computing presents a concept harnessing both the autonomous nature of CPS and the ubiquitous cloud computing paradigm [4]. This kind of solution could e.g. migrate SCADA to a cloud environment making it consume no physical space within the solution itself.

2.6 Azure IoT services

Microsoft Azure offers a number of services suitable for cloud-based IoT solutions. They include cloud gateways, connectivity services and computing services. Linking together their SaaS and PaaS components, an IoT solution can be configured as a combination of services each dedicated to certain tasks. Another way of configuring such solutions is to provision IaaS services and develop the system functionalities from scratch. While this requires more work, it provides more freedom over the used frameworks and protocols.

2.6.1 Cloud gateways

Microsoft Azure offers three cloud services that can be used as cloud gateways: Azure IoT Hub, Azure Event Hub and Azure Virtual Machines. The service models are SaaS and IaaS.

Azure IoT Hub

Microsoft Azure provides a variety of SaaS to be used in Internet of Things solutions. Their primary preconfigured cloud gateway service called IoT Hub is specifically designed for device data ingestion and bidirectional communication. This SaaS offers an application programming interface for creating device identities in an identity registry, controlling access to the service and bidirectional communications between the client and the service. IoT Hub acts as a gateway between the IoT device and the cloud services situated in Microsoft Azure cloud. It can direct the incoming data to a number of PaaS and SaaS that can be provisioned from the Azure marketplace. [41]

To establish a connection between the IoT device and IoT Hub, Azure offers software development kits (SDKs) for various programming languages. Using these SDKs, a programmer can use predefined tools to create a device identity and communicate with IoT Hub [41], that has its own resource identifier. The data is transferred over a network using selected data transfer protocol. At this point, IoT Hub only supports MQTT, AMQP (Advanced Message Queuing Protocol) [1] and HTTPS for device-to-cloud messaging.

IoT Hub provides all the functionalities required for device-to-cloud connectivity and access control. Using IoT Hub is simple in the device's point of view, but can add some complexity in the cloud infrastructure with its data structure and frequency specifications. The key benefits when using IoT Hub are security and reliability, but not time-critical data ingestion. It supports push communication in data ingestion, but also request/response methods. Either of these communication patterns can be implemented depending on the user's requirements.

IoT Hub is priced by the unit and per month. This means that each IoT Hub unit has a static monthly price that doesn't depend on the usage. There are four tiers of IoT Hubs in Microsoft Azure: F1, S1, S2 and S3. [42] The tiers differ in the amount of messages they can ingest per day and, of course, their price. F1 and S1 tier hubs support 12 messages per second, S2 tier hub supports 120 and S3 6000. The monthly prices for these units are, respectively, 0€, 42€, 420€ and 4200€.

Azure Event Hub

Azure Event Hub is a gateway service dedicated for data ingestion. It doesn't establish a bidirectional communication or a device identity storage, but acts as an endpoint for pushing data to one's cloud computing services. Event Hub processes events published by the connected devices into so called partitions, where events are stored as an ordered sequence [34]. Clients can access the event data from one of these partitions at their own pace. The usage of partitions differs from message queueing, where data is queued based on their arrival, in that there are multiple queue-like partitions in parallel. This way, the communication can occur with scalability as more events can be sent to a number of partitions [34].

Devices are programmed to communicate with Event Hubs using Microsoft Azure SDKs. Event Hubs support MQTT, HTTP and AMQP protocols for event publishing. Event hubs are not recommended as the primary cloud gateway, but rather for ingesting data within the cloud, e.g. forwarding data from an IoT Hub to backend computing services. [48, p. 11][34]

Event Hubs are priced per unit by the amount of hours they're active and the amount of events they process. There are basic and standard tier hubs, but the price of data ingestion is 0.024€ per million events for both of them. For the basic hub, the active time price is 0.013€ per hour and for the standard hub 0.025€ per hour [35].

Azure Virtual Machines

If the quotas or pricing of the preconfigured cloud gateways aren't suitable for one's solution, a cloud gateway can be configured from scratch using a virtual machine (VM). Any programming framework can be installed on an IaaS virtual machine to enable code generation in a desired programming language. While this approach requires work and knowledge of communication systems in IoT, it can erase some of the limitations present when using IoT Hubs or Event Hubs. Such limitations are e.g. data ingestion frequency and protocol support. A more detailed look at virtual machines is taken in chapter 2.6.3.

2.6.2 Communication routes

In addition to supporting communication over the public internet, Azure offers solutions for VPN and dedicated connections. In these options, SaaS are provisioned and from Azure Marketplace and configured with network address settings and performance tiers.

A VPN connection to Azure resources is established using Azure VPN Gateway. It handles the data encryption and connection authentication. VPN Gateway runs on dedicated virtual machines that are automatically configured upon VPN Gateway provisioning. The configuration of these VMs are based on the selected gateway settings and can't be manually configured further. Azure offers four tiers of VPN Gateways: basic, VpnGw1, VpnGw2 and VpnGw3. The basic tier is advised to be used in development or proof of concept scenarios, whereas the other tiers are suitable for production. The VPN connection can be configured as point-to-site and site-to-site. Point-to-site provides connectivity to single client devices, whereas site-to-site connects an on-premises network to an Azure virtual network. [30] The price for VPN gateways is from 30€ to 720€ per month [44].

A dedicated connection separate from the public internet is established using Azure ExpressRoute. Configuring this setup requires working with an Ethernet provider company that handles ExpressRoute connectivity. They configure a separate communication route based on the settings provided by the client. An ExpressRoute SaaS is provisioned from Azure to monitor and control the connection costing 250€ to 43,261€ per month [37][38].

2.6.3 Computing

The three main ways of executing computations in the Microsoft Azure cloud are Azure Functions, Azure Machine Learning Studio and Azure Virtual Machines.

Azure Functions

Azure Functions is a so called serverless computing resource. This means that the underlying computing infrastructure is automatically configured and scaled based on the com-

puting load [13]. An actual computing server is still required, but its allocation and configuration is offloaded from the user. Functions is a PaaS and can be used in numerous ways (e.g. cloud storage maintenance or data manipulation) and is accessed by the user through Azure Portal graphical user interface.

Each time an Azure Function runs, it executes the code developed by the user. Functions defined by the user can run multiple executions in parallel. For data-based computations, this means that data won't be lost if the function code is in the middle of execution when a new data point arrives. Instead, the new data point will be ingested by another execution instance of the same function. [32] The code can be written using a web user interface and a number of programming languages, for example C#, F#, Node.js, Python or PHP. The way an Azure Function works is by function calls or triggers. The functions are triggered using HTTP triggers from a source of the user's choice: Azure storage, Azure event hub (e.g. IoT Hub), HTTP request etc. In IoT solutions, the Function is triggered when new data is collected by a cloud gateway.

The pricing is per execution and currently a million executions cost €0.169, with one million free executions per month.

Azure Machine Learning Studio

Azure Machine Learning Studio is a platform service containing models for predictive analytics and ways to perform data-based computing. This PaaS offers a graphical user interface for injecting data, training different machine learning models, writing code and launching solutions as web services. [39][40] The user compiles and trains a model of chosen algorithm, writes any data science code they need, then deploys the solution as a web service to be used as an on-demand computing resource.

The machine learning models are trained using a set of training data that the user must provide themselves. Representative input output pairs are used to train the models offline to determine which algorithm works the best and if they could be made better by tuning their parameters. To train the model, Machine Learning Studio has a block called "train model", making the process very simple. Once the model is trained, it can be used to produce intelligence on selected input data. The machine learning models Azure offers can be chosen from the following algorithm families [40]:

- Regression,
- Anomaly detection,
- Clustering,
- Two-class classification,
- Multiclass classification.

There are different algorithms within the families. The largest selection of algorithms is in the two-class classification family with nine different choices. Clustering only contains

one algorithm: K-means. The algorithm parameters are freely tunable. Regression models are used for predicting values, anomaly detection aims to detect unusual data points, clustering discovers structure of data, two-class classification and multiclass classification predict categories in which data points belong to.

To enable complex data analytics and computations, Azure Machine Learning contains modules for running Python and R scripts [36]. In addition, Anaconda – a Python environment containing scientific Python packages, such as NumPy and SciPy [12] – is installed and ready to be used when writing code. Any algorithms can be written using the online code creation interface or imported as .zip files [36]. In Machine Learning Studio, the code is then presented as a block that has inputs and outputs. Any inputs for the code are inserted by placing an input block and connecting it to the code block. Outputs are presented similarly.

Once the model has been trained or otherwise configured, it can be launched as a web service. It can then be consumed as request-response service (RRS) or batch execution service (BES) by authenticated remote applications. RRS provides a low-latency service called using REST API. The connection between the request making application and the service is secured and authenticated using secure sockets layer (SSL) and HTTP. The requesting application asks the web service for an output from the machine learning model with the data in the request as the input. If authentication of the requesting application is successful, the output is given in JSON-format and will be parsed by the receiver. [39] RRS is used when the response for given data needs to be acquired immediately after the request. Batch execution service takes a batch of data as an input and runs the users choice of algorithms on it. The results will be saved to a storage account and their location is returned to the requester. This kind of execution is suitable when the results are not needed right away. A large set of data points can be analyzed using BES and the outputs can be used for intelligence later.

Machine learning studio can be used for free with certain limitations considering the amount of modules in an experiment, storage space and web APIs. A standard tier subscription has no such limitations, but has a monthly and hourly fee. The free tier is suitable for research projects, while the standard tier provides a better platform for production purposes.

Azure Virtual Machines

Azure Virtual Machines are a virtualization of their physical computing resources. A cloud virtual machine is IaaS, providing the user with the freedom of selecting some of the underlying computing hardware. It's the user's responsibility to install programming frameworks and other applications they need in their solution. The user can use the VM as a regular PC using remote desktop connection to access the operating system.

In VMs, a so called hypervisor, a software component that provides the virtual infrastructure for VMs, is situated between the physical infrastructure and the virtual machine. [9, p. 175] The hypervisor adds another layer between the physical hardware and the operating system, which can be called a virtualization layer. This means the operating system used by the user doesn't communicate with the physical hardware directly but through this additional layer. This changes the virtual machine behavior compared to that of a desktop PC so that there is an additional overhead and thus delay in operations.

Virtual machines support a wide range of communication methods and protocols. They can be configured as web servers that use any communication pattern selected by the user. The virtual machines have their own public IP addresses and can be assigned an URI for RESTful communications. This makes virtual machines easy to access by authorized devices and users.

Any algorithm computing programs and frameworks can be installed on a virtual machine. This, and the fact that VMs support basically every communication protocol there is, makes them well suited for IoT data-based computation.

In Microsoft Azure, virtual machines come in large variety of sizes and operating systems. The VMs can be categorized in general purpose, compute optimized, memory optimized, storage optimized and high performance computing machines. The compute optimized instances provide CPU performance over memory and are ideal for batch processes and application servers. [43] High performance compute instances take the CPU performance even further by offering the fastest processors available among Azure Virtual Machines. For CPU intensive computation applications, these two categories are the most suited. For Big Data analytics, memory optimized VMs will do the best. [43]

The VM pricing model is per usage time. This means that the specified costs will accumulate only when the VM is running, no matter the amount of CPU used or data ingested. The prices are defined per hour, but will be applied based on the minute: running a VM for less than an hour will charge you less than for an hour's usage. The more powerful the VM, the higher the hourly fee. [47]

Users can create their own custom VM images of appropriate size and containing the required software and applications. This is done by first provisioning a VM off a base image found in the marketplace, then installing the features needed on the instance and lastly saving the machine image to an Azure storage account as a .vhd file. After these steps, a custom VM can be provisioned using the saved image. The VM will contain all of the installed software components and features, which means they don't have to be manually installed after provisioning. This is ideal for automating the provisioning of virtual machines in certain applications, where the need for computing power changes.

3. SYSTEM DESIGN

The use case for the cloud-based system designed in this thesis is presented in the following chapter. Then, the functional criteria for each distinct part of the solution are presented. These criteria are determined based on the main purpose of the solution and reflect the requirements set by the case system. The choice between potential technologies is made based on the criteria. These design steps are taken starting with cloud communication architecture, followed by communication protocol and cloud computing service. After the technology selections, the detailed design is described.

3.1 Case description

In this case study, a sensor network consisting of 5 microcontroller-based nodes with inertial measurement units (IMUs) is setup on a hydraulically actuated flexible beam manipulator. This setup is illustrated in figure 6. The flexible beam modeling and control has been studied by Mäkinen in his master's thesis [52]. The sensors are a product of other projects in the department of IHA, and they're described in detail in chapter 3.6.1. In addition to the IMU sensors, an angle encoder is situated at the base of the beam. The local system also contains a local real-time control unit. This controller gathers the measurement data from the sensors and the encoder and uses it to compute control signals to a hydraulic cylinder. The goal is to control the position of the beam end point in two dimensions in Cartesian space by actuating the cylinder. The local controller is configured to run at a 0.002 second sampling time, i.e. 500 Hz. The task for the cloud-based IoT system is to gather the measurement data from the sensors and send it to the cloud to run algorithms that are not feasible to execute in the local controller.

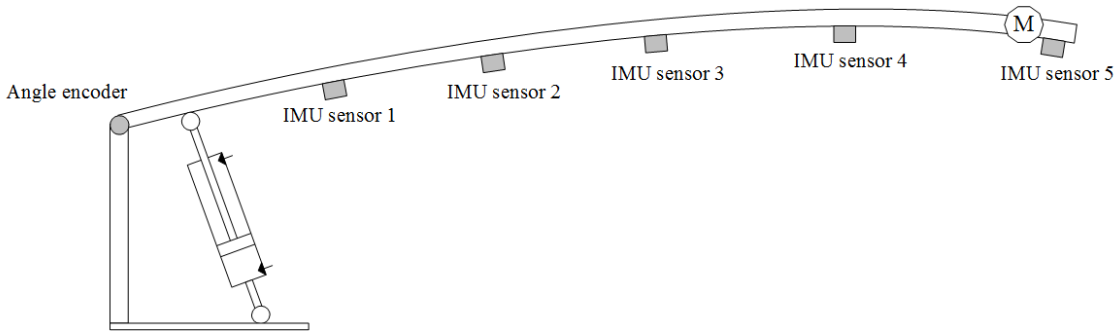


Figure 6. The flexible beam, five IMU sensors and the angle encoder setup. Weights are attached to the beam tip, which causes the beam to flex.

The main focus of the case is on the IoT and cloud computing, which means the hydraulic flexible beam system and the load algorithm are given less attention in the following

sections. The goal of the case study is to design and demonstrate the cloud-extended sensor network and it working with a local system. An end-to-end IoT solution capable of bidirectional communication is configured.

The system to be designed in this thesis is to enable sensor data acquisition in the public cloud and cloud services configured for sensor data based computations. These computations include advanced mathematics and physics. The cloud-based IoT system architecture is similar to that described in figure 5 in chapter 2.4. The communication architecture, protocol and cloud computing service are selected among potential technologies. The system is designed to receive sensor data, perform calculations and send data to a local system. It doesn't save history data, making it cost-efficient for soft real-time computing applications. The focus in the study is the *functionality* of the components and the complete system. The underlying high-detail architectural aspects included in each technology are left out of the scope.

The criteria for the complete system include the following. The deployed system's cloud services will be accessible only by the sensors we use and grant access to, i.e. it isn't shared as a public service. The system is built to support quick response times and for usage in soft real-time interactions with a local automation system, e.g. for local system parameter updating. When the system is capable of such fast reactions, the number of applications it can be used for is larger. The system is also designed to use Ethernet for communications throughout the system.

3.2 Communication architecture

The design of the communication between IoT devices and the cloud includes the selection of a possible field gateway, communication protocols and the cloud gateway. Also, the resource identifier type is selected.

3.2.1 Criteria

The cloud communication architecture has to support the amount of data generated per second. In this thesis, the requirement is 500 Hz data ingestion, which is set by the local controller's sampling time. The communication system has to be able to collect all the sensor data points from the network to perform computations on all the data. The data production frequency should be defined on premises in the sensor nodes and devices connected to the cloud resources – not in the cloud. This way, there are no strict limitations to how frequently data can be sent to the cloud and the designed system can be applied where each data point should invoke a reaction from the cloud.

Another key aspect when choosing cloud communication methods is pricing. Microsoft Azure has different pricing models for different services. Some of them are based on

hourly usage, others on the amount of data. The best pricing model in this study can't be based on data amounts, since they are allowed to be vast.

The cloud gateway should support the typical IoT communication protocols. The more protocols supported, the more freely the communication protocol can be selected. Also, the chosen architecture shouldn't be overly complex or require a noticeable number of additional components to be configured. To summarize, the criteria for cloud connectivity architecture are:

- Support for the amount of data points generated by the sensors each second (500 Hz),
- Support for a number of communication protocols,
- Simplicity,
- Low price.

3.2.2 Technology selections

For the simplicity and price of the solution, the communication is selected to take place over the public Internet. There is additional complexity when using VPN, since a VPN gateway is required and the clients have to initiate the connection using VPN certificates. This is not supported when using simple devices, e.g. microprocessors. The usage of VPN may prove to be impractical, because the connection is to be established each time the local system starts up. While the dedicated direct connection to the cloud resources provides the best performance, it is too expensive in the context of this thesis.

We decide to use a field gateway device to collect data from the sensors and forward it to the cloud resources. To make sure each data point is sent to the cloud at the same time as it is collected by the local controller, the field gateway collects the sensory data from the controller. This helps making sure that the time stamps in data points from each sensor match. As intricate security aspects aren't a part of this thesis, the local controller is not communicating over the Internet, making the field gateway necessary. The field gateway collects data points from all our sensors and sends the data to the cloud as a single message. Using data batching, the data send frequency to the cloud can be reduced by the selection of the batch size, i.e. the number of consecutive data points per sensor. This will allow the data processing in the cloud to take more time before a new batch arrives and to reduce the ratio of data packet overhead and payload.

C. Horn and J. Krüger [22] make a comparison between latencies using devices that can be used for such edge-to-cloud connections in industrial automation. The comparison was made between a standard industrial PLC, microprocessors and a direct connection. While connecting the devices directly to the cloud resources using a dedicated connection proved to be the best choice, microprocessor-based connection provided quite similar performance. Thus, we select to use a microprocessor for data collection from the local

sensors and communications with the cloud resources. The communication from the cloud to the local system, e.g. a real-time controller, is also handled by the chosen field gateway device.

In the system, we configure a virtual machine to act as a cloud gateway. The restrictions in data ingestion frequency of reasonably priced IoT Hub tiers are too strict for usage in our application, where data is generated at 500 Hz. The problem with an Event Hub as a cloud gateway is the ingestion-only connection: messaging from the cloud to the local system may prove difficult. The per-hour pricing model of the virtual machine is suitable for the system development, as the system can be shut off frequently. Virtual machines support all network communication protocols and the data transfer costs using virtual machines are almost negligible.

The selection of resource identifiers defines the way data transfer is implemented. IP communication is not easily scalable, as the participants must know the IP addresses of each other participant. If more clients are added, their IP addresses must be known and introduced to the other participants. While URI-based communication provides better scalability, the resource naming and parsing the structured data messages introduces a need to configure an API. To reduce the complexity of the communications, IP addressing of the resources is selected.

3.3 Communication protocol

In this chapter, the selection of a communication protocol for sending data between the field gateway microprocessor and Microsoft Azure cloud environment using IP-based communication over the public internet is made. The best protocol for sending messages from cloud to the on-premises devices depends on the type of the messages and the application in which the cloud is used. This chapter focuses on the selection of the protocol used in sending data to the cloud. The selection of the communication pattern is inherited from the selection of the protocol. Again, the criteria for the used communication protocol are defined and the technology selection is made based on the criteria.

3.3.1 Criteria

In this thesis, the focus is on the connectivity protocols that perform sufficiently in high frequency sensor data acquisition. As mentioned in chapter 2.1.1, machine automation applications tend to be quite time-critical because of the fast dynamics of such systems. This means that data used in algorithms should be available within a predefined period of time. Along with the chosen communication architecture, the chosen communication protocol has an impact on the data transfer speed.

The size of each packet sent over a network can’t have a negative impact on the network performance. This would compromise the usefulness of a system, where quick reaction

times are desirable. The size of each data packet depends on what additional information about the data and communication is transferred. The communication protocols have different amounts and lengths of fields included in each data packet. If the packet payload is smaller or even similar size as the packet header, most of the network performance is used to carry information other than that produced by the sensors. Thus, the packet header should be small.

Since the frequency of data produced by sensors used in our machine automation application is 500 Hz, there's no real need for acknowledging the receiving of each data point. Thus, the selected protocol doesn't require such acknowledgment mechanisms in our solution. This way, high-frequency data streams are supported.

As sensors are usually very simple devices, the protocol used for data transfer shouldn't require any excess computation resources such as encryption. The criteria that the selected communication protocol must satisfy are:

- IP-based resource identifiers,
- Support for time-critical, low-latency applications,
- Support for high frequency data transfer (500 Hz),
- Small packet header,
- Support for simple devices.

3.3.2 Technology selection

The way potential communication protocols fulfill the design criteria are shown in table 1. The indexes are scaled from 1 to 3, with 1 meaning poor performance, 2 acceptable performance and 3 good performance. Time-critical index indicates the protocol's ability to handle time-critical data transport. Message frequency index depicts the feasible amount of messages that can be sent each second using the protocol. Support for simple devices describes the feasibility of using the protocol in a simple device.

Table 1. *The potential communication protocols versus the design criteria.*

Protocol	Time-critical index	Message frequency index	Support for simple devices	Header size (bytes)
TCP	1	3	3	20
UDP	3	3	3	8
MQTT	2	3	3	2
CoAP	3	3	3	4

As table 1 depicts, UDP, MQTT and CoAP have very similar indexes and TCP is clearly worse in most categories. This goes to show that while a reliable and widely used protocol, TCP can't answer to the time-critical requirements of our solution as it includes retransmissions and flow control. MQTT falls partially to the same category, as it runs on TCP and requires a message broker application.

Based on table 1, the protocol is selected between UDP and CoAP. Both of them can answer to some time-criticality requirements, but packets arriving out of order can reduce the timeliness of the data transport. UDP has a larger header in data packets and lacks built-in confirmable messaging. However, CoAP uses a RESTful interface for data sending and receiving, which causes additional work when configuring the data receivers: an API must be configured to enable communication. This makes UDP the simpler of the two from the developer's perspective. As this thesis is about deploying high-frequency sensor data transport to the cloud, creating an API to save 4 bytes in each packet header is not worth the time and effort.

Previous study conducted by D. M. Lofaro et al. [27], show the feasibility of UDP in sending sensory data over the internet to enable geographically adjacent real-time control of a robot system. They compared a number of protocols including TCP and UDP to determine the latencies involved. UDP provided the shortest latency and using it in sensor-to-cloud data transport, they are able to control the robot motion with cloud in the control loop. This result is promising in our application as well, where low-latency performance is of interest.

Based on these results, the selected communication protocol for sensor-to-cloud messaging is UDP. With this selection, the used communication pattern is push. In this thesis, UDP is used for communication from cloud to the local system as well. This way, the implementation for communication is uniform and simple. This selection means the reliability of such messaging should be configured manually, e.g. sending acknowledgment UDP messages from the local devices in response to the messages from the cloud.

3.4 Computing service

The computing service is responsible for executing user-defined algorithms on the sensory data. In this thesis, the algorithms can be developed by the user and are not the main focus.

3.4.1 Criteria

As we've already selected the cloud communication architecture and the used protocol, the cloud computing service must support them. This means the service supports IP and port based communication over the Internet and UDP message receiving and sending.

The key requirement for the cloud computing service is the support for data-based computations. This includes at least matrix operations, advanced mathematical operations, data type conversions and advanced trigonometry. These computations are widely used in algorithms executed in automation systems.

As the selected communication architecture and protocol both support high frequency data transport, the computing service should do so as well. This means, the service has to be powerful enough to perform algorithms in subsecond range. Scalability of the underlying processing infrastructure is an important aspect in succeeding in this. This way, all the sent data will be received and can be acted on by the service.

The different PaaS and IaaS have different pricing models. Price is a design criteria in this thesis, as we're trying to achieve a cost-effective system as well. To sum up, the selected computing service should:

- Support the selected cloud communication method and protocol,
- Support data-based algorithm computations,
- Support high frequency (500 Hz) data ingestion,
- Have an affordable price.

3.4.2 Technology selection

While the platforms and software services provided by Azure require no maintenance of the underlying infrastructure from the user, there is a difference in the supported data ingestion methods and protocols compared to an IaaS virtual machine. The algorithm deployment is similar in all the potential technologies, but only virtual machines support any programming language, as such frameworks can be installed by the user. There are differences in the support for mathematical computations in the different programming languages as well. The best support is found in C# and Python, and e.g. Node.js lacks a complete support.

Azure Functions has no support for extra mathematical libraries, and thus doesn't meet our criteria for computation support as well as virtual machines and Azure Machine Learning Studio. As viable as Azure Machine Learning Studio is in its other properties, it doesn't support IP-based communication and data transport using UDP. The underlying computing power is not configurable by the user, which means the algorithm execution times can't be influenced. Virtual machines support the chosen communication protocol, a vast choice of frameworks for data-based computing and the pricing model is per-hour. The price can be selected by the user by choosing a virtual machine of certain size. The underlying computing power can be increased by the VM size selection as well. Thus, Azure virtual machines are chosen as the cloud service to provide data-based computations.

3.5 The proposed system architecture

The complexity of the cloud end of the system is reduced by using a single virtual machine as a cloud gateway and a data processing resource. Using the chosen technologies, a sensor-to-cloud connection can be established by assigning IP addresses to the sensors, the field gateway and the cloud gateway. The diagram of the proposed system functionalities is presented in figure 7.

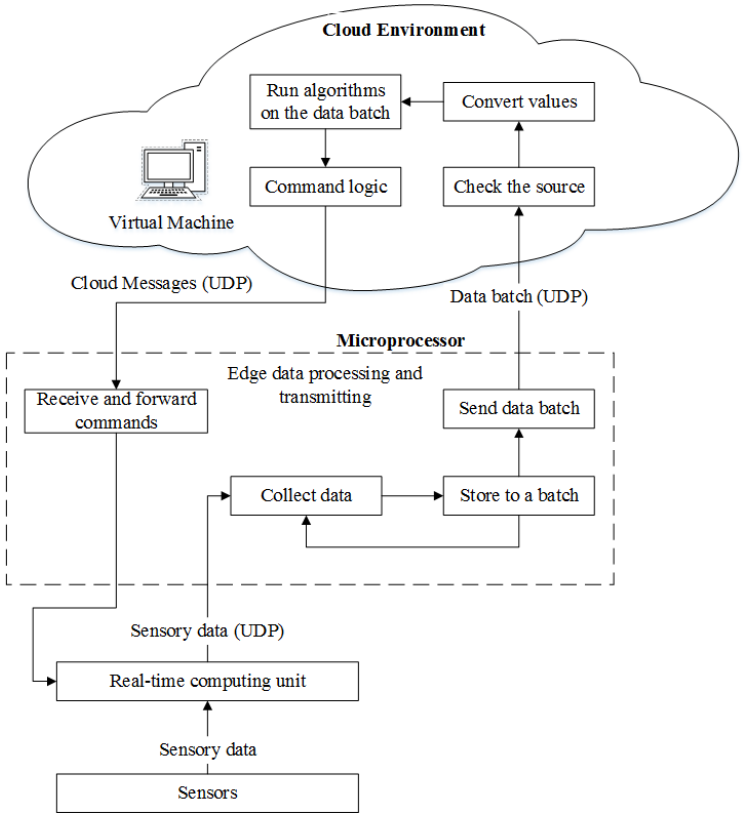


Figure 7. The architecture and functionality of the proposed system.

The sensor network sends the measurement data to a real-time computing unit via UDP. In this real-time unit, hard real-time control algorithms can be deployed to actuate the flexible beam. The sensory data is forwarded to the field gateway microprocessor located at the edge of the Internet, handling bidirectional edge-to-cloud communication. The field gateway collects the sensor data to a batch containing a user-defined number of consecutive measurements to regulate the data send frequency. The batch is then forwarded to the cloud gateway virtual machine using UDP and IP-based communication over the public Internet. The virtual machine checks the source and an elementary authentication key of each data batch, converts bit-valued data to double precision and runs an algorithm on the data. After the algorithm execution is finished, the previous data is discarded and the ports are being listened for the next data. The system is designed in such a way, that the VM can perform command logic and send command signals to the field gateway. The gateway can then deliver the cloud commands to the local controller. This way, the cloud system can update or otherwise interact with the local algorithms. In this case study, we're not

implementing such functionalities, but use the bidirectional communication to study the communication performance. Command functionalities were implemented by the author of this thesis in a study, where local control parameters were updated based on cloud commands [50].

The functional groups and components used in the proposed solution are depicted in figure 8 in respect to the IoT architectural reference model described in chapter 2.2.1. The functional groups used are communication, IoT Service and security. Communication method and protocol are included in the communication FG. The IoT service FG consists of the virtual machine provisioned in the cloud environment. IoT service resolution provides the IP address of the VM. The security FG consists of authentication and authorization. Access to the IoT service is limited to the clients that pass the access control policies set by the system developer. The local control system is the application that uses the cloud computing service to gain results from the algorithms executed in the cloud.

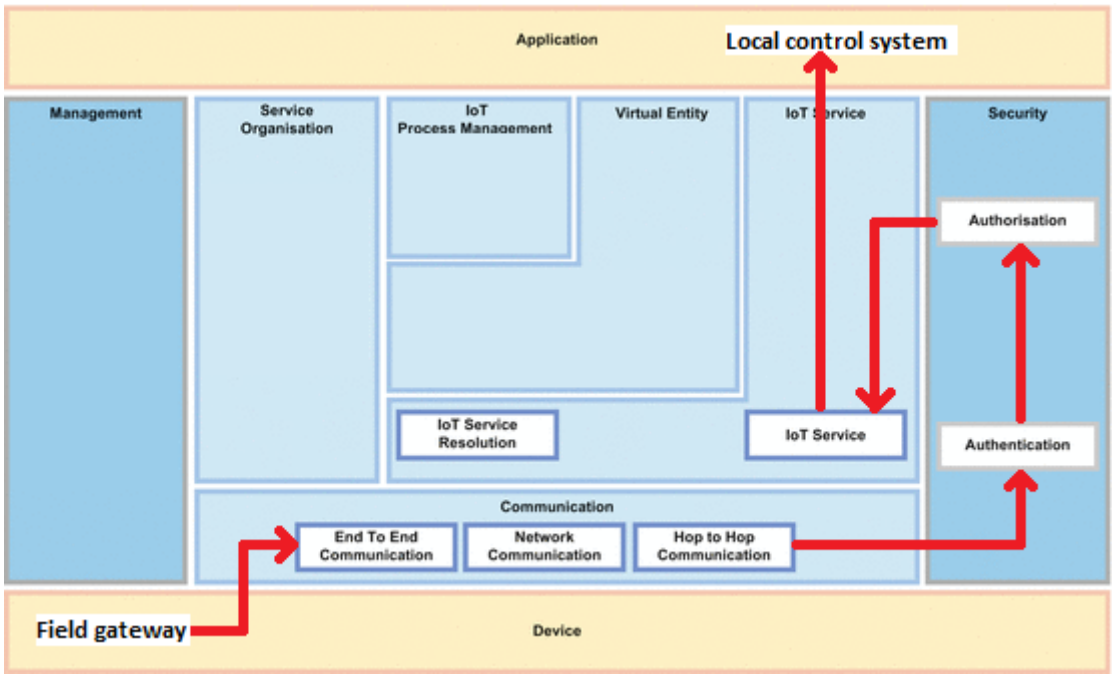


Figure 8. The functional groups and components used in the proposed system [adapted from 3, p. 168].

It can be seen that the proposed system architecture is very simple. This is due to the fact that the sensor network data ingestion and algorithm can be managed using a single cloud resource: a virtual machine.

3.6 Solution components and their functions

The used hardware and the functionalities they have are presented in the following sections. The distinct components of the solution are the sensor nodes, real-time computing

unit, field gateway and cloud virtual machine. The used hardware is introduced followed by algorithms and functions configured in them.

3.6.1 Sensor nodes

The inertial measurement unit used in this case study is Bosch's BMI160 [5]. It provides accelerometer and gyroscope readings for axes x, y and z, meaning the sensor produces six separate values: three acceleration measurements and three angular velocity measurements. The sensor unit itself is 2.5 by 3.0 millimeters and costs less than $4. The range of gyroscope readings is up to 2,000 degrees per second and can be selected via serial digital interface [5]. For this case, the range is set to ±125 degrees per second. As for the acceleration, the range can be up to ±16 g, and it is set to ±2 g. The sensor output data rate is 1600 Hz.

The measurement data is collected from the IMU sensor by a microcontroller. In this case study, we use STM32F407 [63] microcontrollers. It is a low-cost ($40) component that comes from a line of microcontrollers by ST, a global semiconductor company. The STM32F4 series is one of their high performance models and it has a Cortex-M4 core running at 168 MHz [63]. The microcontroller comes with up to 1 Mbyte of Flash memory.

Connecting an ST microcontroller to a network requires additional components. In this thesis, we're using an experimental, low-cost sensor node design by Eelis Peltola (e.g. [79]) in another project at IHA. The sensor node consists of the IMU, microcontroller and a custom baseboard including Ethernet physical layer PHY chip. The chip enables the sensor node to connect to an Ethernet network. The sensor node both takes power and sends data using a single category 5 [69] based customized Ethernet cable. These sensor nodes are completely encased and attached to the flexible beam firmly, such that there is as little vibration due to loose fitting as possible. One of the sensor nodes is presented in figure 9. The blue custom base board can be seen in the black casing. The Bosch IMU is attached to the pins of the board. For clarity, the ST microprocessor board has been taken out of the casing and is seen on the right.

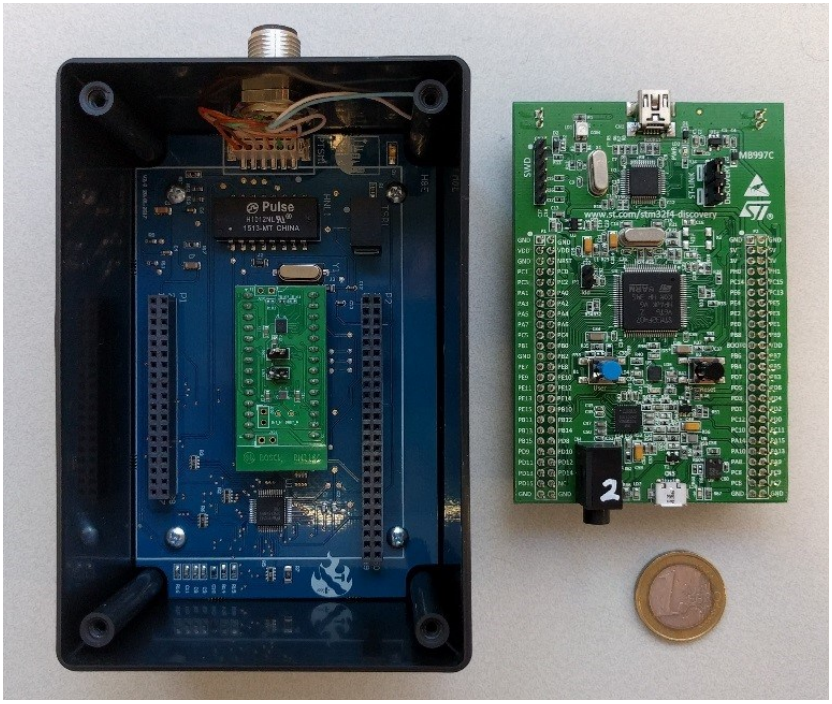


Figure 9. The sensor node components with the STM32F407 microprocessor (on the right) taken out of the casing [79]. The one euro coin provides scale.

The application running in the STM microprocessors that read data from the Bosch IMUs has also been configured by Eelis Peltola. The application was developed using Matlab's Simulink Waijung block set designed for ST microprocessors and Simulink Stateflow for application programming. The advantage of doing so is the simplicity of creating the application, and the ease at which its functionalities can be seen from the model. Using Simulink's built-in "build" operation, the created Simulink model was then converted to C language and built to run on the STM.

Figure 10 describes the complete model running in the sensor nodes. The model consists of initialization blocks and a state chart. The target on which the model is built along with a number of execution properties are defined in a target setup block. Using this block, the sample time at which the model is run in the microprocessor is defined. For this case, we select to use the fastest sample time supported by the board: 0.001 milliseconds. This equates to 1 MHz sampling frequency.

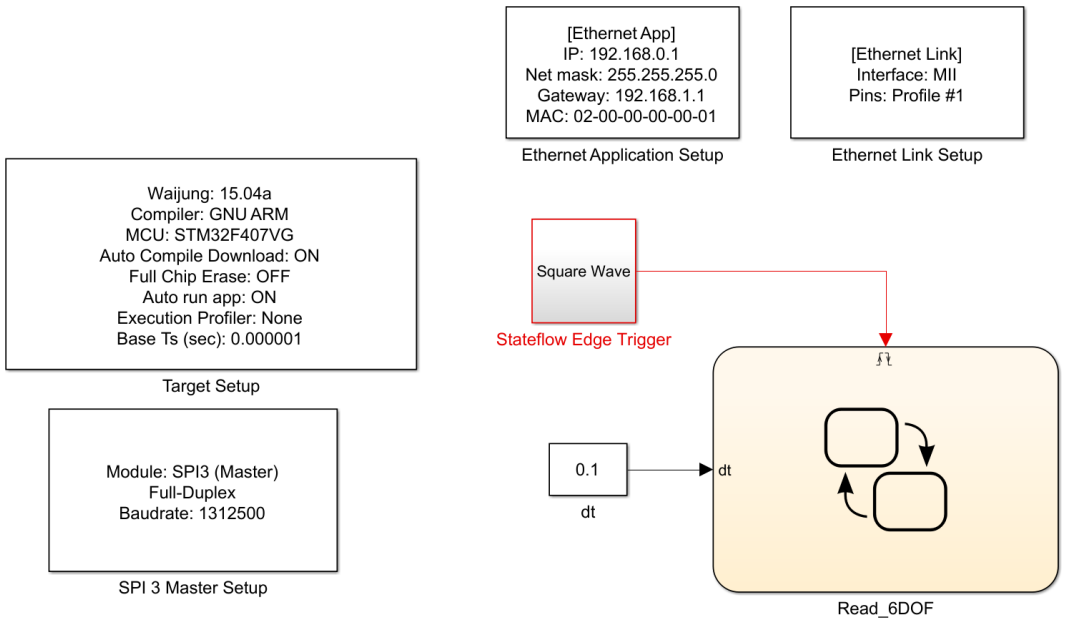


Figure 10. The complete model executed by a sensor node.

The initialization blocks are configured for each STM node. The pins to which the Ethernet port component is connected to are defined in the Ethernet Link Setup. This way, the microprocessor will use the correct pin configuration for data communications. For each STM node, the static IP and MAC addresses are determined manually using the Ethernet Application Setup block. At the time of writing this thesis, there was no support for dynamic IP addressing in the Waijung block set, meaning the resource identifiers can't be set automatically by an external source, e.g. dynamic host configuration protocol (DHCP) server [45]. The serial peripheral interface (SPI) block configures the pins where the Bosch IMU is attached to, enabling data gathering from the sensor.

The basic functionalities in the state chart (called "Read_6DOF" in figure 10) are preparing the measurement unit, conducting fast offset compensation, reading data from the sensor and sending the sensor data using UDP. These functionalities are implemented to the Simulink Stateflow chart that is presented in appendix A. The Stateflow chart is executed each sample time by configuring a trigger to the chart. When the trigger port input value changes, the chart is executed. Using a square wave that has a different value each time instance enables the chart to be updated at the set sample time rate.

The IMU measurement is acquired in 8-bit unsigned integer form. The length of each measurement is 12 bytes, with two bytes for each value. The three axes gyroscope data is located in the first six bytes and the acceleration data in the last six. With two bytes for each measurement, the data can be converted to double precision by combining them as depicted later in section 3.6.4. The sensor units are configured to send the measurement data to the real-time unit as 8-bit unsigned integers (uint8) based on the configured IP address of the unit.

The encoder used to measure the beam base angle is a commercial product. Its technical details are not important in this thesis. The encoder produces the angle measurements in radians and is connected to the local controller using CAN. The measurement data is used in the local controller where it is also converted to an 8-bit long uint8. This data is forwarded to the field gateway using Ethernet cabling and UDP.

3.6.2 Real-time computing unit

The real-time computing unit used in this case setup is dSPACE [17] model DS1005. Local control algorithms are implemented on it to control the beam motion. The dSPACE runs at a sample rate of 0.002 seconds, collecting data from the sensor nodes at a constant interval. Using dSPACE, the measurement system is synchronized, i.e. the timestamps of the sensory data match.

In the dSPACE unit, all the data from the sensor network is collected at 500 Hz. The data is then compiled into a vector containing a measurement from each sensor. Then, the vector is sent forward to the field gateway microcontroller via UDP. The dSPACE unit supports UDP messaging using 32-bit unsigned integer (uint32) format. As the uint8 sensor data reaches the dSPACE, it is automatically encoded to a uint32 vector.

3.6.3 Field gateway

For the STM that handles gathering the data from the measurement nodes and sending it to the cloud, a Simulink Stateflow model was created. The model was built on a separate STM board that has been configured with an Ethernet port. This STM gathers a user-defined amount of consecutive data points from each data reading STMs and the encoder. It then sends those measurements to the cloud over the Internet as UDP packets. The Simulink model consists of the initialization of the STM board, the state chart and Simulink functions used for data communication. The initialization steps are the same as depicted in chapter 3.6.1. The Simulink state chart for sensor data reading and cloud message receiving is shown in figure 11.

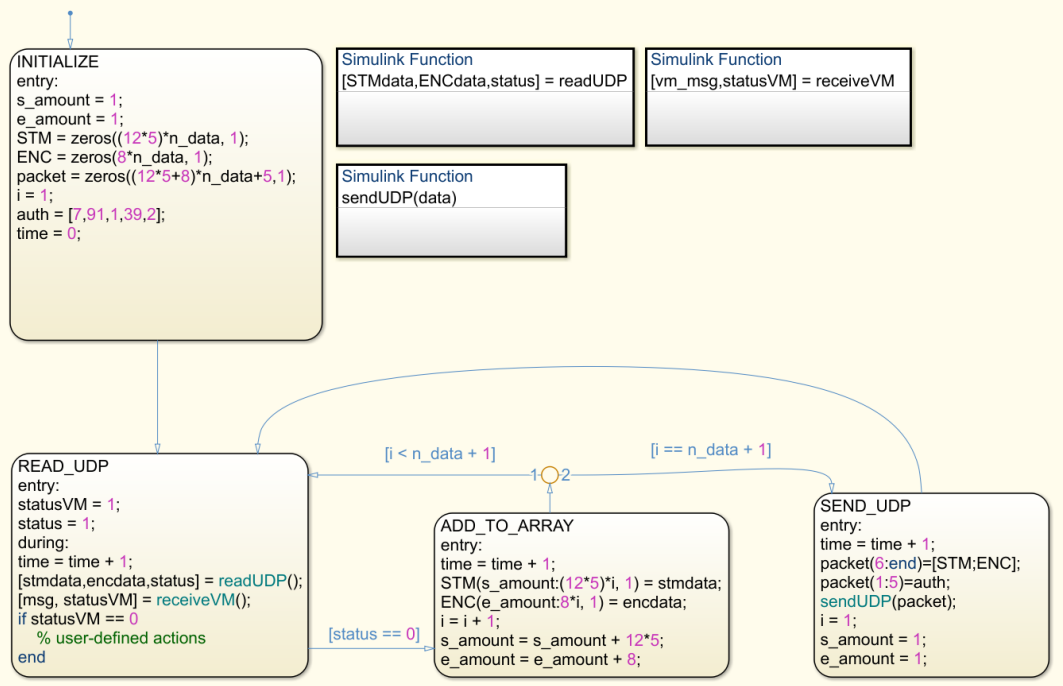


Figure 11. The Simulink Stateflow chart of the field gateway functions.

The chart contains states for variable initialization, sensor data reception, packet creation and packet sending. After the variables are initialized, the field gateway starts to listen to the port where the real-time dSPACE unit is forwarding the sensor data. Also included in the model is a timer, which is implemented in case packet timestamp information is needed. When a data packet reaches the listened port, the "readUDP" Simulink function status changes to zero. The received data is then converted from uint32 to uint8 and added to an array. If the array is shorter than the defined data amount, more measurements are added to it. Otherwise, a small authentication vector is added to the start of the packet that is then sent to the cloud virtual machine using UDP. This vector represents the usage of, e.g., a digital signature or a hash key [70][72]. In this study, this authentication isn't fully functional, but indicates the authentication code usage in the data packets. To implement more proper security, the unique authentication key should be generated by an algorithm.

The field gateway is configured to receive messages from the VM via UDP. In the model, messages from the cloud are listened for at the same rate as the sensor data packets. If a new message arrives from the cloud, user-defined actions can be configured to take place. In this case study, the bidirectional communication is used for data transfer performance monitoring.

3.6.4 Virtual machine

In the cloud, the main tasks are data ingestion, data conversion, data-based computations and sending data. To make the cloud tools independent of any software licenses, these

tasks are performed using Python programming language available as a free online download. To gain all the tools needed in algorithm computations, Anaconda, a Python distributable containing a vast library of mathematical tools, is used.

For the case study, a virtual machine with Windows Datacenter 2016 operating system was selected. The virtual machine size is chosen as D2_v2, that contains two cores and 7 Gb of RAM-memory to provide sufficient all-around performance. The price for running this VM is €0.176 per hour. The location of the remote data center was selected to be west Europe, which is geographically closest to Finland, where the case study was conducted. To allow data flow into such an Azure virtual machine, some configurations in the cloud environment have to be made. The virtual machine is situated within a virtual network subnet that has its own network security group (NSG) with a configurable access control list (ACL). The rules for allowing incoming data sent using certain protocol is configured in the Azure Portal. For this case study, a port for listening to incoming UDP traffic from a specific IP address is opened. In addition to allowing the data into the subnet, it has to be allowed to reach the virtual machine. To achieve this, two steps must be taken: allow UDP traffic in at the virtual machine's NSG and allow it in through the VMs Windows firewall. Figure 12 demonstrates the security configurations. After these configurations, a UDP server can be written in code using Python.

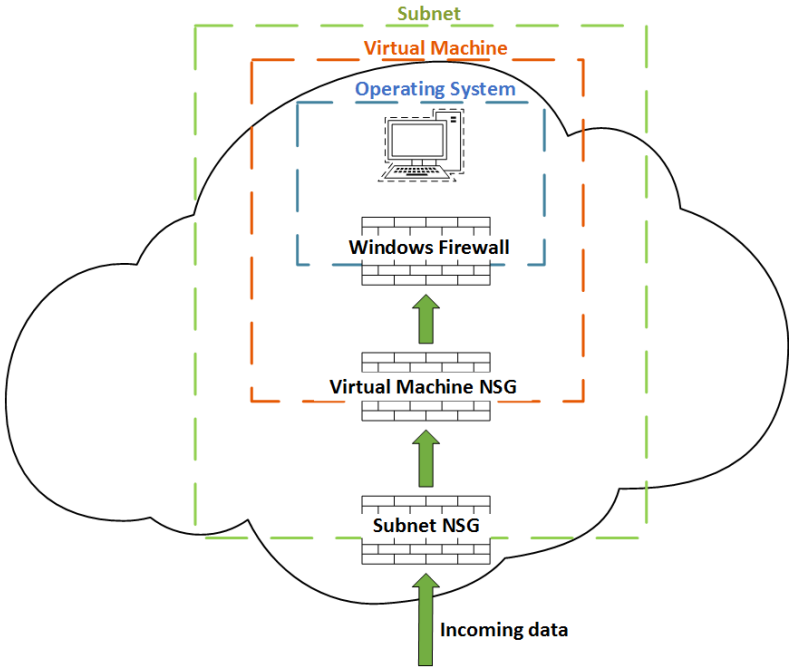


Figure 12. The security measures in the cloud.

In the virtual machine, a Python script containing UDP server functionalities, data conversion and algorithm computing is created. The UDP server is configured to listen to certain ports for incoming datagrams from predefined IP addresses. In this case study, the edge-to-cloud communication takes place using a network router with its own static IP

address. All of the data coming from the field gateway is originated from a single UDP port, which makes access control simpler. In Python, UDP server can be coded using UDP sockets. A socket is defined with a specific buffer for data ingestion. Packets smaller than or equal to the buffer size will be ingested, larger ones dropped. The incoming data is checked for the source IP address, source IP port, data packet length and the authentication key at the beginning of the packet.

The data is transported over the network in bits. When a packet reaches the virtual machine UDP server, it converts the data into its original form, 8-bit unsigned integers. The authentication key at the beginning of the datagram is then recognized and the data computations commence. If the authenticate is wrong or not there, the data packet is discarded immediately. The encoder data is converted to double precision by using Numpy's built-in function called "view". The STM IMU sensor data is converted to double precision by manipulating the corresponding uint8 bytes as formulated as a block diagram in figure 13.

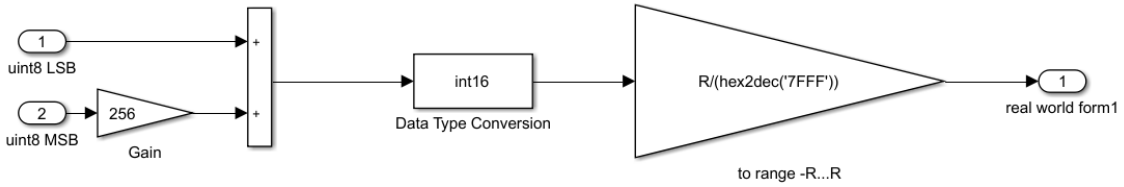


Figure 13. STM IMU sensor data conversion computation as a block diagram.

The most significant bytes are every other bytes in the data packet and using only them in the conversion would give values close to the actual measured values. The least significant byte provides more accuracy in the converted value. The more significant byte is multiplied by 256 to make it more significant in the int16 created from the two bytes. The 16-bit integer is multiplied by the specified measurement range and lastly divided by the maximum int16 value, 65535 or 7FFF in hexadecimal.

Once the received data batch is converted to double precision, it can be used in an algorithm of the user's choice. In this case study, the algorithm acts as a computing load to represent the performance of the cloud computing instance. The mathematical concepts of the algorithm are out of the scope of this study. The algorithm contains large matrix inversions and multiplications, which can be computationally demanding. A high-level description of the algorithm is given in appendix B.

3.6.5 Communications

The physical communication medium in this case study is standard Ethernet. It doesn't provide as reliable real-time performance as the traditional fieldbus systems [76], but for soft real-time tasks, it should provide low latency. Also, the use of Ethernet in all parts of

the solution provides homogeneity of communications and eases the system development. IP-based communications can thus be used throughout the system.

The local components are configured to be in the same local area subnet by assigning their IP addresses accordingly. This way, the communication between them can take place through local area network switches. The STM IMU sensors, real-time computing unit and the field gateway are within the same subnet and can communicate with each other. Using CAN, the data from the encoder is transmitted to the real-time controller where it can be accessed from using the controller's IP address. The virtual machine is situated in its own virtual network subnet, and it can communicate with the on-premises network through a network router. The router has two static IP addresses: one in the local area network and the other in the public Internet. The local IP is not routable unlike the public IP that is used in the VM to authenticate the received data source and locally to receive data from the cloud. The Ethernet communication scheme is depicted in figure 14. In the figure, the network hops in the public Internet are represented by the lightning symbol. Here, the number of hops affects the time it takes for a data packet to reach its destination. The network connection used is a 100 Mbps hardline.

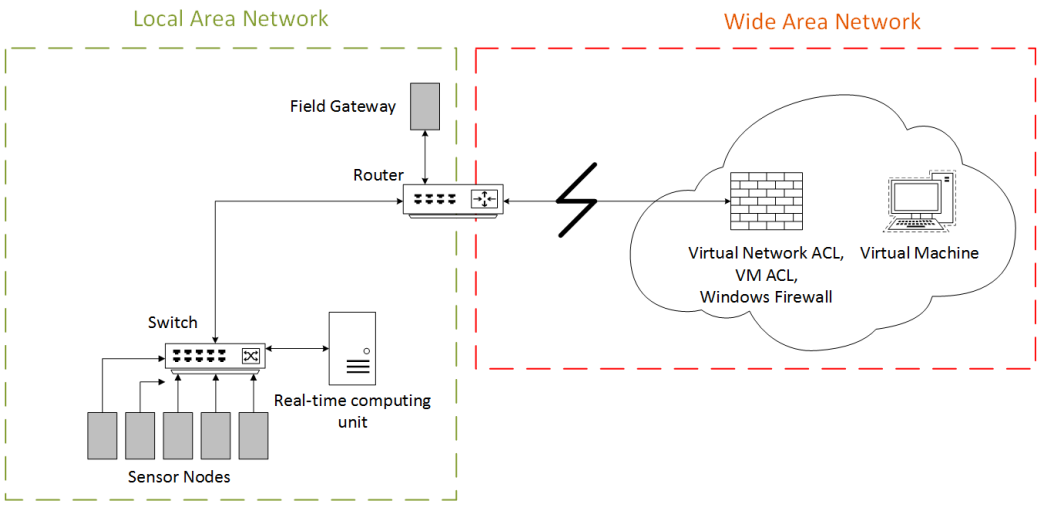


Figure 14. The Ethernet communication scheme of the case system.

4. TEST SETUP AND RESULTS

To test the performance of the deployed solution, a series of tests are run to find answers to the main research questions: the response time of the cloud computing server and the supported amount of data per second. With the answers to these questions, the time-criticality of the tasks performed in the cloud can be evaluated. We'll also pay attention to the system data transport reliability, i.e. does the data reach the intended receiver.

In this case study, the key communication performance indicators are the network round-trip time (RTT) and the amount of packets dropped in a given period of time. The round-trip time includes the time it takes for a packet to reach its destination, data manipulation time at the destination and the time it takes for the packet to reach its original sender again. The RTT depends mainly on the geographical distance between communication participants and the network speed, but also the network topology and the maximum transmission unit (MTU) of the network nodes. [25] In this setup, the cloud data center is located approximately 3000 kilometers away from the on-premises sensor network. The network connection speed is 100 Mbps. According to the study on RTT dependency on geographical distance by O. Krajsa and L. Fojtova [25], the RTT expected with the 3000 km distance is somewhere near 38 milliseconds. The tests conducted in this chapter can't give a conclusive value to the RTT, because we're using public Internet. The network traffic from all the other Internet users can affect the measured RTT that can thus get different values at different time of day. This test setup aims to show an average guideline of the measured values that can be expected.

The computing performance of the selected virtual machine can be determined by monitoring the algorithm execution times. The tests only give information about the certain sized VM running the selected algorithm. It is noteworthy that the computation load will determine the required VM performance and thus the VM size. The maximum data ingestion frequency can be determined using the processing time measurements, as further discussed in section 4.2.

The test setup is as follows. First, a reference value for the data RTT is determined by sending small packets containing only timestamp data from the field gateway to the cloud environment. Then, the measurement data from the sensors is sent to the cloud in batches. The data batch size is increased to determine the point at which the system's operation is severely degraded. For each data batch size, the data RTT is measured to determine how it depends on the packet size. In these tests, only the STM IMU sensor data is sent to the cloud, as we're evaluating the packet size's effects on the system rather than the packet contents'. This is done to evaluate the number of such sensors the system supports. As the encoder data points are of smaller size, the batch size can be increased in constant intervals using only 12-byte data points from the STM IMUs. Each batch will contain an

index telling the order of the batches. The VM will monitor this index to determine if packets are dropped while the data batch size is increased. The VM's computing performance with an increasing data batch size is measured simultaneously.

4.1 Communication performance

The initial RTT measurement results are depicted in figure 15. The field gateway microcontroller sends a packet containing the current time and the latest RTT to the virtual machine. The sent packet size is 24 bytes. The virtual machine receives the timing information and sends the received timestamp back to the microcontroller, while saving the received RTT. The packet sent back is 20 bytes long. The microcontroller receives the timestamp and calculates the difference between the current time and the timestamp to get the RTT.

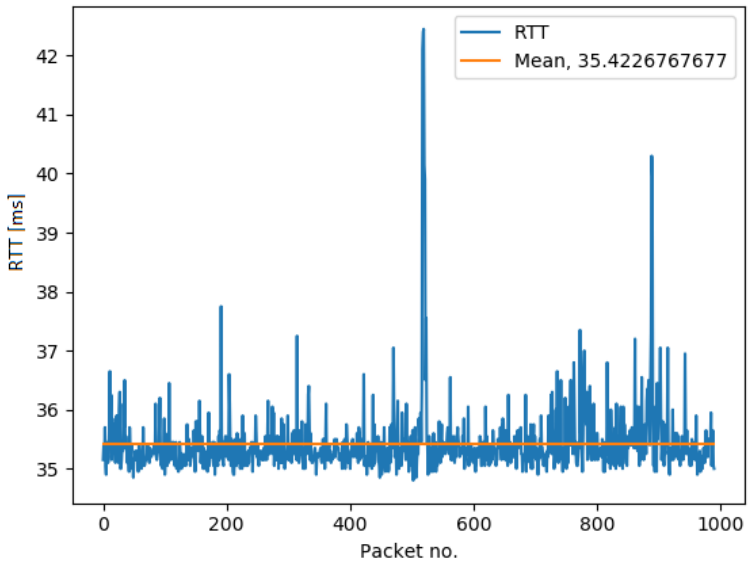


Figure 15. The initial round-trip time measurement results.

As can be seen from figure 15, the average round-trip time using the designed infrastructure and packets containing only timing information is 35.4 milliseconds. It is noteworthy, that these values are influenced by the network traffic at the time of the measurement, as the data is sent over the public Internet. Also, there is no notable computation load in the cloud virtual machine. However, this measurement gives a rough estimate of the smallest RTT achievable using this communication architecture.

As the packet size increases, the round-trip time will increase as well. To get a realistic evaluation of the RTT, the sensor network data is sent to the cloud along with the timestamps to gain the RTT data. In the virtual machine, the data-based algorithm is run, which will also affect the RTT. The processing times for different sized data packets are studied in chapter 4.2.

The packet payload consists of the sensor data (12 bytes), round-trip time, timestamp and the packet number. The last three are 8 bytes long each. Also included is a 5-byte elementary authentication key. The packet size in bytes can thus be calculated as:

$$size = n_{sensors} * 12 * n_{measurements} + 3 * 8 + 5, \quad (1)$$

where the number of sensors ($n_{sensors}$) is 5. The number of measurements per sensor ($n_{measurements}$) is increased until the system's functionality is compromised.

For each packet size, the data transfer is monitored for one minute. The measurement results for 1500 packets per packet size are shown in figure 16, where the boxplot boxes represent the range where 50% of the measurements are. The orange line inside each box is the median of the measurements. The whiskers represent the 5% and 95% quartiles, i.e. the values between which 90% of the measurements are.

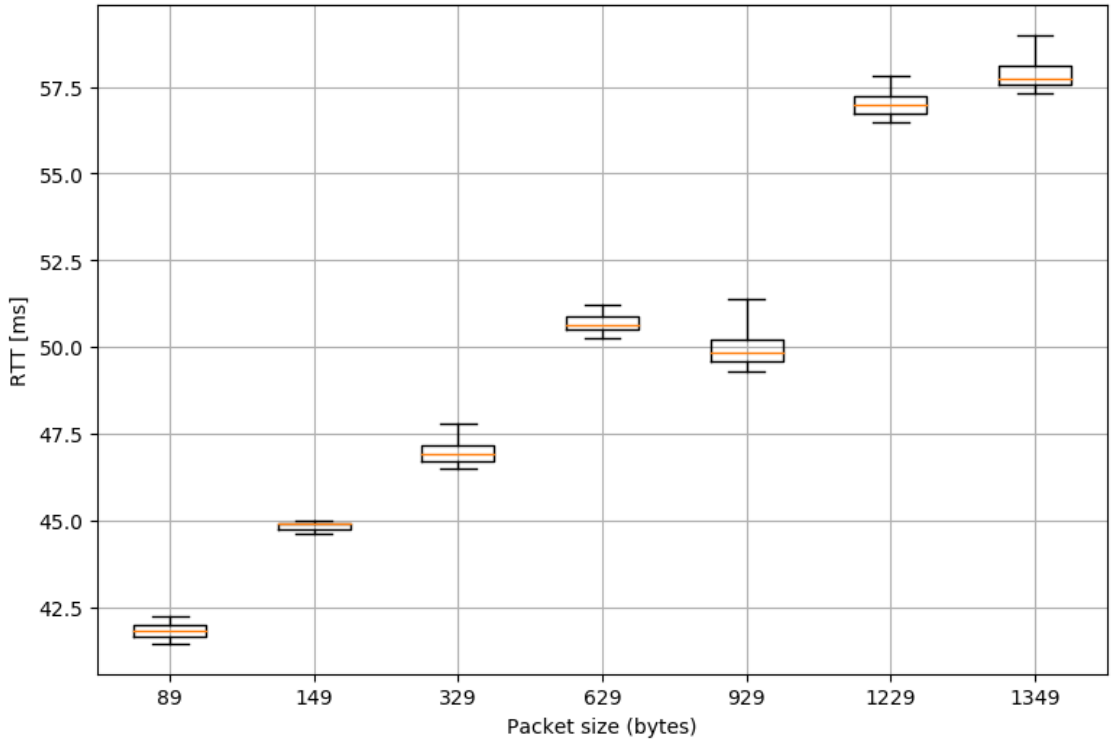


Figure 16. The round-trip time with different data packet sizes.

As expected, the RTT increases as the packet size gets larger. However, when the packet size is 929 bytes, the RTT is shorter on average than when the packet is 629 bytes, but there's a larger distribution spanning over two milliseconds for the larger packet size. We'll conclude that this is a coincidence that happened during the testing. When the packet size exceeds 1500 bytes, the transmission fails. This is due to the limits of the network nodes: somewhere along the route, a data transferring component has a maximum transmission unit of size 1500 bytes. This means, that the system as it is can support

data packet sizes lower than 1500 bytes. For five sensors, this equates up to 24 measurements per sensor in each data batch. Furthermore, in theory, the system can support 122 of the described STM sensors if the sent data packets only contain a single measurement from each sensor.

Based on these tests, the system can be used to complete tasks that should take under 60 milliseconds. However, in the measurements, there were a small number of RTT values that were clearly higher than 60 milliseconds. For figure clarity, these values weren't plotted in figure 16. This shows the timing unreliability of the communication over public internet, since the RTT can fluctuate considerably. With the designed system, the outcome of tasks performed in the cloud can sometimes take longer to reach the local system. Thus, the feasible tasks performed in the cloud should allow an occasional additional delay.

While the round-trip times were measured, the number of dropped packets per minute was also monitored. The results show, that for each packet size, not a single packet was dropped during the one minute timespan.

4.2 Computing performance

In this case, the processing time depicts the time it takes for the algorithm to execute on a single packet of data. Processing time is key in determining the frequency at which the system can function. With more measurements from each sensor per packet, the frequency at which data is sent to the cloud decreases, leaving the VM with more time to process each packet before the next one arrives. In figure 17, the processing time is measured for 1500 packets per packet size. The boxplot boxes and whiskers indicate the same statistical measures as in figure 16.

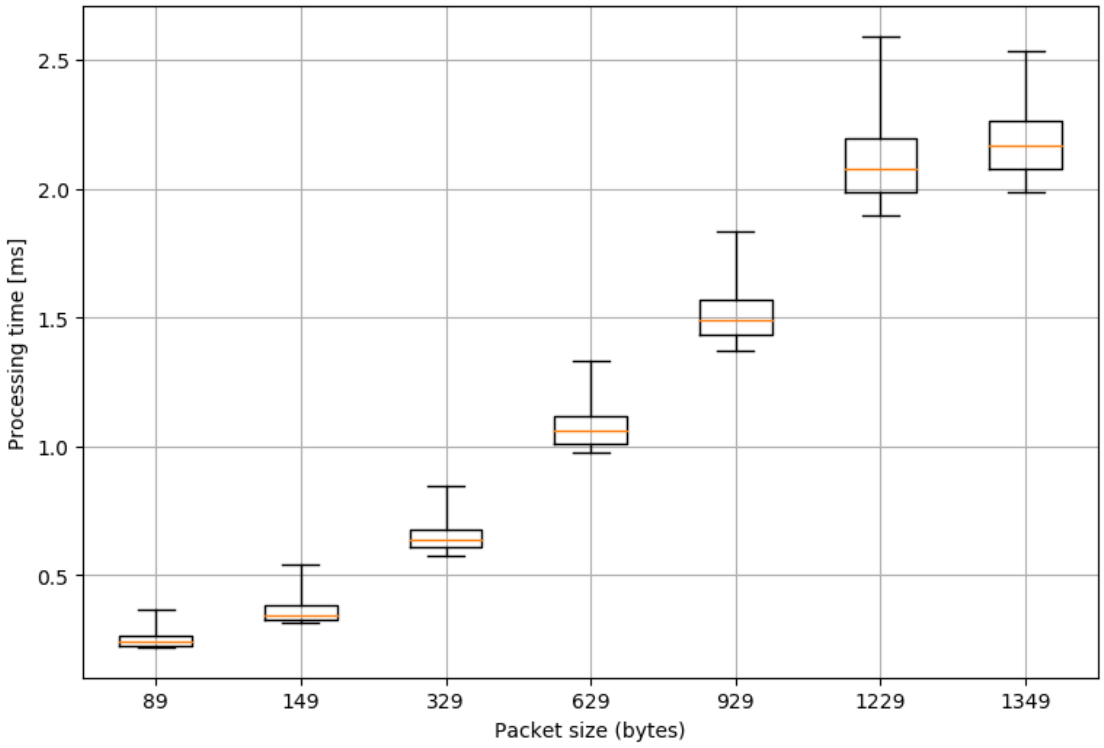


Figure 17. The selected virtual machine's processing times on different sized data packets.

As can be seen in figure 17, the processing time increases with the packet size, since more data points are used as an input to the algorithm. As we're operating with packets smaller than 1500 bytes, the processing time with the selected algorithm remains mostly below 3 milliseconds.

The supported data ingestion frequency is dependent on the processing time of each data packet. Successful data ingestion at a given frequency means the packets aren't dropped. If a packet arrives before the VM has processed the previous data, the packet will be dropped. The data ingestion frequency can be determined by studying the processing time measurements (e.g. figure 17). For certain sized packets, the data ingestion frequency is determined by the processing time. For instance, with our algorithm running in the selected virtual machine, with 929 byte packets, the system can ingest data every 2 milliseconds (figure 17), i.e. 500 Hz, receiving at least 90% of the packets. With packets smaller than 1500 bytes, the end-to-end system supports up to 333 Hz data send frequency.

5. CONCLUSIONS

In this thesis, a connection between a sensor network and a cloud computing platform for algorithm execution was designed and deployed. An architecture for using public cloud resources as a part of a local machine automation control scheme was proposed. The main design criteria included low-latency end-to-end communication, low pricing and support for data-based computations in the cloud. Tested in a machine automation application, the system performed with reliability and quick response times in accordance to the end goals set in the early stages of the project.

The performance tests revealed that the cloud-extended part of the system can support 500 Hz data ingestion, when the size of the data packets sent to the cloud is below 1000 bytes. The average round-trip time with this packet size is 50 milliseconds, with a computational load running in the cloud. In the testing environment, the network MTU was 1500 bytes, restricting the maximum packet size. The tests revealed that on average the system RTT remains below 60 milliseconds.

The designed solution has potential in supporting systems, where low-cost local hardware is used for real-time tasks. The cloud extension is feasible in scenarios, where a computing result from the cloud needs to be available in subseconds. Such applications are e.g. control sequence computing, local parameter updating and even real-time control. However, the RTT to the cloud and back fluctuates considerably, which may cause missed timing deadlines. Thus, tasks performed in the cloud should allow an occasional missed deadline. In such applications, the response time from the cloud would still remain below 60 milliseconds most of the time. The proposed system's performance in a supervisory control application has been tested and evaluated in a conference paper by the author of this thesis [50]. The flexible beam system's local control parameters were updated from the cloud with low latency and reliability in the testing conditions.

The overall costs of the proposed system are the combined costs of the used physical components and cloud resources. To apply the cloud-extended sensor network to the flexible beam control scenario, only a network router and the cloud virtual machine were purchased. The Laboratory of Automation and Hydraulic Engineering already had the sensor nodes, the field gateway microprocessor, the local control system of the beam and network cabling. The virtual machine pricing model is per-hour. If we assume the selected VM runs eight hours every weekday, the monthly cost is only 28€. The router cost about 30€, making the total upfront costs of extending the existing sensor network to the cloud 30€ plus a 28€ monthly fee. Compared to using a local computer for algorithm execution, the price is considerably lower.

The system further supports low-cost cloud services by utilizing data batching. When data is sent in batches, the data ingestion frequency required from the virtual machine decreases. This gives the VM more time to process the received data, which allows for more demanding algorithms to execute before the next batch arrives. Using a combination of the batch size and VM size, the system can be configured to run algorithms more computationally demanding than the one used in this case study, while maintaining a reasonable data ingestion frequency. This means, the system can maintain fast response times. This is definitely an advantage and can reduce the costs of implementing more demanding algorithms to the system, as a less expensive VM can be used. Further studying and testing on this aspect should be conducted.

The case study was conducted in a laboratory. The network speed in the case study was 100 Mbps and the sensors were connected to the network with Ethernet cables. In forestry machinery, for example, there won't be hardline connections available and a wireless connection has to be used. The proposed system should be tested with wireless sensors and field gateway to determine its performance with a constrained network. For machinery to find the cloud resources and to make sure of the quality of the connection requires extra development of the proposed system. Furthermore, security aspects need to be considered should the system be developed for usage in production applications. The proposed system was developed to measure the functional aspects of long-range communications with the cloud. The system should only be used for testing, since the lack of proper security measures is a major issue. Security can be implemented using a VPN connection to the cloud resources, which would change the system architecture by including a VPN capable device in the communication scheme. Performance tests for such solutions should be conducted. Another way to implement data authentication is to include a data dependent security code in each data packet and use public internet connectivity.

The proposed system wasn't designed for scalability. Changes in the computing load in the cloud may require virtual machine size or amount adjustments, which have to be done manually. As the system is to be used with a certain set of algorithms in the cloud, the CPU usage won't fluctuate by a great amount. Hence, it doesn't require scaling based on the CPU usage, because the VMs can be sized in advance to provide the desired performance. The lack of scalability is present with the amount of sensors as well. Adding more sensors to the system requires changes in the cloud algorithm, and in the algorithms running in the local controller and the field gateway. While these changes aren't large, they need to be configured by hand to each of the mentioned components separately. This requires care and attention to avoid human error in configuration parameters, such as IP addressing.

The amount of work required when translating an algorithm originally developed in Matlab and Simulink to Python was considerable. With little prior experience with Python, the task was demanding. However, the Python distributable Anaconda provided the same tools found in Matlab, which helped the cause. To translate the Simulink models

into code required a large amount of loop programming and complex tables and matrices. This shows the extra work required when using open source programming resources in the cloud.

REFERENCES

[1] AMQP, website. Available at (cited on 7.9.2017): https://www.amqp.org/

[2] L. Atzori, A. Iera, and G. Morabito, The Internet of Things: A Survey, in: Computer Networks, Elsevier, Vol. 54, No. 15, 2010, pp. 2787-2805.

[3] M. Bauer et al., IoT Reference Architecture, in: A. Bassi et al. (eds), Enabling Things to Talk, Springer, Berlin, Heidelberg, 2013, 349 p.

[4] A. Bereş, Semantic cyber-physical cloud systems, in: IEEE 5th International Symposium on Digital Forensic and Security (ISDFS), © 2017 IEEE, pp. 1-6.

[5] Bosch BMI160 datasheet, Bosch, document number BST-BMI160-DS000-07, 2015, 110 p. Available for download at (cited on 22.8.2017): https://www.bosch-sensortec.com/bst/products/all_products/bmi160

[6] A. Botta et al., On the Integration of Cloud Computing and Internet of Things, in: IEEE 2014 International Conference on Future Internet of Things and Cloud (FiCloud), © 2014 IEEE, pp. 23-30.

[7] S. Bruque-Cámara, J. Moyano-Fuentes, and J. M. Maqueira-Marín, Supply chain integration through community cloud: Effects on operational performance, in: Journal of Purchasing and Supply Management, Elsevier, Vol. 22, No. 2, 2016, pp. 141-153.

[8] V. G. Cerf, and R. E. Kahn, A Protocol for Packet Network Intercommunication, in: IEEE Transactions on Communications, © 1974 IEEE, Vol. 22, No. 5.

[9] K. Chandrasekaran, Essentials of Cloud Computing, CRC Press, Boca Raton Florida, USA, 2015, 385 p.

[10] Y. Chen, Z. Du, and M. García-Acosta, Robot as a Service in Cloud Computing, in: Fifth IEEE International Symposium on Service Oriented System Engineering, © 2010 IEEE, pp. 151-158.

[11] G. Clarke, A Brief History of Microsoft Azure, website. Available at (cited on 11.9.2017): http://garyclarke.us/technology/a-brief-history-of-microsoft-azure/

[12] Continuum, Anaconda package list, website. Available at (cited on 7.9.2017): https://docs.continuum.io/anaconda/packages/pkg-docs.html

[13] B. Cotton, Serving up serverless science, website. Available at (cited on 11.9.2017): https://www.nextplatform.com/2017/03/17/serving-serverless-science/

[14] D. Coupek, A. Lechler, and A. Verl, Cloud-based control for downstream defect reduction in the production of electric motors, in: IEEE International Conference on Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles & International Transportation Electrification Conference (ESARS-ITEC), © 2016 IEEE, pp. 1-6.

[15] A. Dobrin et al., Cloud challenges for networked embedded systems: A review, in: IEEE 20th International Conference on System Theory, Control and Computing (ICSTCC), © 2016 IEEE, pp. 866-871.

[16] B. S. Đorđević et al., Cloud Computing in Amazon and Microsoft Azure platforms: Performance and service comparison, in: 22nd Telecommunications Forum Telfor (TELFOR), © 2014 IEEE, pp. 931-934.

[17] dSPACE homepage, website. Available at (cited on 22.8.2017): https://www.dspace.com/en/inc/home.cfm

[18] O. Givechi et al., Control-as-a-Service from the Cloud: A Case Study for using Virtualized PLCs, in: 10th IEEE Workshop on Factory Communication Systems (WFCS), © 2014 IEEE, pp. 1-4.

[19] E. Goldin et al., Cloud computing for big data analytics in the Process Control Industry, in: 25th Mediterranean Conference on Control and Automation (MED), © 2017 IEEE, pp. 1373-1378.

[20] Z. Hao et al., Challenges and Software Architecture for Fog Computing, in: IEEE Internet Computing, © 2017 IEEE, Vol. 21, No. 2, pp. 44-53.

[21] T. Hegazy, and M. Hefeeda, Industrial Automation as a Cloud Service, in: IEEE Transactions on Parallel and Distributed Systems, © 2015 IEEE, Vol. 26, No. 10, pp. 2750-2763.

[22] C. Horn, and J. Krüger, Feasibility of Connecting Machinery and Robots to Industrial Control Services in the Cloud, in: IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), © 2016 IEEE, pp. 1-4.

[23] IPv6.com, UDP – User Datagram Protocol, website. Available at (cited on 15.8.2017): https://www.ipv6.com/general/udp-user-datagram-protocol/

[24] John Deere, PowerSight™ Technology Solutions, website. Available at (cited on 16.8.2017): https://www.deere.com/en_US/products/engines_and_drivetrain/powersight/powersight.page

[25] O. Krajsa, and L. Fojtova, RTT measurement and its dependence on the real geographical distance, in: 34th International Conference on Telecommunications and Signal Processing (TSP), © 2011 IEEE, pp. 231-234.

[26] K. Lee et al., Extending Sensor Networks into the Cloud using Amazon Web Services, in: 2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications, © 2010 IEEE, pp. 1-7.

[27] D. M. Lofaro, A. Asokan, and E. M. Roderick, Feasibility of Cloud Enabled Humanoid Robots: Development of Low Latency Geographically Adjacent Real-Time Cloud Control, in: IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), © 2015 IEEE, pp. 519-526.

[28] P. Mell, and T. Grance, The NIST Definition of Cloud Computing, US National Institute of Standards and Technology, 2011.

[29] G. McGrath et al., Cloud Event Programming Paradigms: Applications and Analysis, in: 2016 IEEE 9th International Conference on Cloud Computing, © 2017 IEEE, pp. 400-406.

[30] Microsoft, About VPN Gateway, website. Available at (cited on 4.9.2017): https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-about-vpn-gateways

[31] Microsoft, An introduction to Azure Functions, website. Available at (cited on 14.3.2017): https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview

[32] Microsoft, Azure Functions hosting plans comparison, website. Available at (cited on 1.9.2017): https://docs.microsoft.com/en-us/azure/azure-functions/functions-scale

[33] Microsoft, Azure REST API Reference, website. Available at (cited on 10.7.2017): https://docs.microsoft.com/en-us/rest/api/

[34] Microsoft, Event Hubs features overview, website. Available at (cited on 6.9.2017): https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-features

[35] Microsoft, Event Hubs pricing, website. Available at (cited on 6.9.2017): https://azure.microsoft.com/en-us/pricing/details/event-hubs/

[36] Microsoft, Execute Python machine learning scripts in Azure Machine Learning Studio, website. Available at (cited on 6.7.2017): https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-execute-python-scripts

[37] Microsoft, ExpressRoute overview, website. Available at (cited on 6.9.2017): https://docs.microsoft.com/en-us/azure/expressroute/expressroute-introduction

[38] Microsoft, ExpressRoute pricing, website. Available at (cited on 6.9.2017): https://azure.microsoft.com/en-us/pricing/details/expressroute/

[39] Microsoft, How to consume an Azure Machine Learning Web service, website. Available at (cited on 6.7.2017): https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-consume-web-services

[40] Microsoft, Introduction to Machine Learning in the Azure cloud, website. Available at (cited on 6.7.2017): https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-what-is-machine-learning

[41] Microsoft, Overview of the Azure IoT Hub service, website. Available at (cited on 5.7.2017): https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-what-is-iot-hub

[42] Microsoft, Reference – IoT Hub quotas and throttling, website. Available at (cited on 11.9.2017): https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-quotas-throttling

[43] Microsoft, Sizes for Windows virtual machines in Azure, website. Available at (cited on 30.6.2017): https://docs.microsoft.com/en-us/azure/virtual-machines/virtual-machines-windows-sizes

[44] Microsoft, VPN Gateway pricing, website. Available at (cited on 6.9.2017): https://azure.microsoft.com/en-us/pricing/details/vpn-gateway/

[45] Microsoft, What is DHCP?, website. Available at (cited on 23.8.2017): https://technet.microsoft.com/en-us/library/dd145320(v=ws.10).aspx

[46] Microsoft, What is IaaS?, website. Available at (cited on 14.3.2017): https://azure.microsoft.com/en-us/overview/what-is-iaas/

[47] Microsoft, Windows Virtual Machines Pricing, website. Available at (cited on 30.6.2017): https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/

[48] Microsoft Corporation, Microsoft Azure IoT Reference Architecture, 2016. Available at (cited on 2.10.2017): https://azure.microsoft.com/en-us/updates/microsoft-azure-iot-reference-architecture-available/

[49] Microsoft Developer Network, Publish/Subscribe, website. Available at (cited on 23.8.2017): https://msdn.microsoft.com/en-us/library/ff649664.aspx

[50] T. Mononen, and J. Mattila, A Low-Cost Cloud-Extended Sensor Network for Supervisory Control, in: 8th IEEE International Conference on Robotics, Automation and Mechatronics (RAM), IEEE, accepted, 2017.

[51] S. C. Mukhopadhyay, N. K. Suryadevara, Internet of Things: Challenges and Opportunities, in: S. Mukhopadhyay (eds), Internet of Things, Smart Sensors, Measurement and Instrumentation, Springer, Cham, Vol. 9, 2014, 261 p.

[52] P. Mäkinen, Modeling and Control of a Single-link Flexible Hydraulic Robot, master's thesis, Tampere University of Technology, 2016, 80 p. Available at: http://URN.fi/URN:NBN:fi:tty-201610244632

[53] V. H. Nguyen, Y. Besanger, and Q. T. Tran, Novel hybrid cloud based SCADA approach for interoperability of micro-grid platforms, in: Innovative Smart Grid Technologies Conference (ISGT), 2016 IEEE Power & Energy Society, © 2016 IEEE, pp. 1-5.

[54] J. Nurmi, On Increasing the Automation Level of Heavy-Duty Hydraulic Manipulators with Condition Monitoring of the Hydraulic System and Energy-Optimised Redundancy Resolution, dissertation, Tampere University of Technology, Publication 1481, 2017, 59 p. Available at: https://tutcris.tut.fi/portal/files/11161374/nurmi_1481.pdf

[55] Oasis, MQTT Version 3.1.1 Plus Errata 01, website. Available at (cited on 10.7.2017): http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[56] Rackspace, The Difference Between Private and Public Cloud, website. Available at (cited on 24.5.2017): https://www.rackspace.com/cloud/cloud-computing/difference

[57] V. Rajaraman, Cloud Computing, in: Resonance, Springer India, Vol. 19, No. 3, 2014, pp. 242-258.

[58] RAPP Cloud, website. Available at (cited on 14.8.2017): https://rapp.cloud/

[59] N. Serrano, G. Gallardo, and J. Hernantes, Infrastructure as a Service and Cloud Technologies, in: IEEE Software, © 2015 IEEE, Vol. 32, No. 2, pp. 30-35.

[60] Z. Shelby et al., The Constrained Application Protocol (CoAP), Internet Engineering Task Force (IETF), 2014, 112 p.

[61] W. Shi et al., Edge Computing: Vision and Challenges, in: IEEE Internet of Things Journal, © 2016 IEEE, Vol. 3, No. 5, pp. 637-646.

[62] B. Siciliano, O. Khatib, Springer Handbook of Robotics, Springer-Verlag Berlin Heidelberg, 2008, 1591 p.

[63] ST, STM32F407/417, website. Available at (cited on 22.8.2017): http://www.st.com/en/microcontrollers/stm32f407-417.html?querycriteria=productId=LN11

[64] D. Thangavel et al., Performance evaluation of MQTT and CoAP via a common middleware, in: IEEE Ninth International Confernece on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), © 2014 IEEE, pp. 1-6.

[65] J.-P. Thomesse, Fieldbus Technology in Industrial Automation, in: Proceedings of the IEEE, © 2005 IEEE, Vol. 93, No. 6, pp. 1073-1101.

[66] S. Vitturi et al., Guest Editorial Special Section on Communication in Automation, in: IEEE Transactions on Industrial Informatics, © 2016 IEEE, Vol. 12, No. 5, pp. 1817-1821.

[67] L. Wang, and A. Canedo, Offloading industrial human-machine interaction tasks to mobile devices and the cloud, in: Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), © 2014 IEEE, pp. 1-4.

[68] Z. Wen et al., Fog Orchestration for Internet of Things Services, in: IEEE Internet Computing, © 2017 IEEE, Vol. 21, No. 2, pp. 16-24.

[69] Wikipedia, Category 5 cable, website. Available at (cited on 22.8.2017): https://en.wikipedia.org/wiki/Category_5_cable

[70] Wikipedia, Digital signature, website. Available at (cited on 2.10.2017): https://en.wikipedia.org/wiki/Digital_signature

[71] Wikipedia, Google Cloud Platform, website. Available at (cited on 11.9.2017): https://en.wikipedia.org/wiki/Google_Cloud_Platform

[72] Wikipedia, Hash function, website. Available at (cited on 2.10.2017): https://en.wikipedia.org/wiki/Hash_function

[73] Wikipedia, Hypertext Transfer Protocol, website. Available at (cited on 7.9.2017): https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[74] Wikipedia, IP address, website. Available at (cited on 6.9.2017): https://en.wikipedia.org/wiki/IP_address

[75] Wikipedia, Transmission Control Protocol, website. Available at (cited on 1.9.2017): https://en.wikipedia.org/wiki/Transmission_Control_Protocol

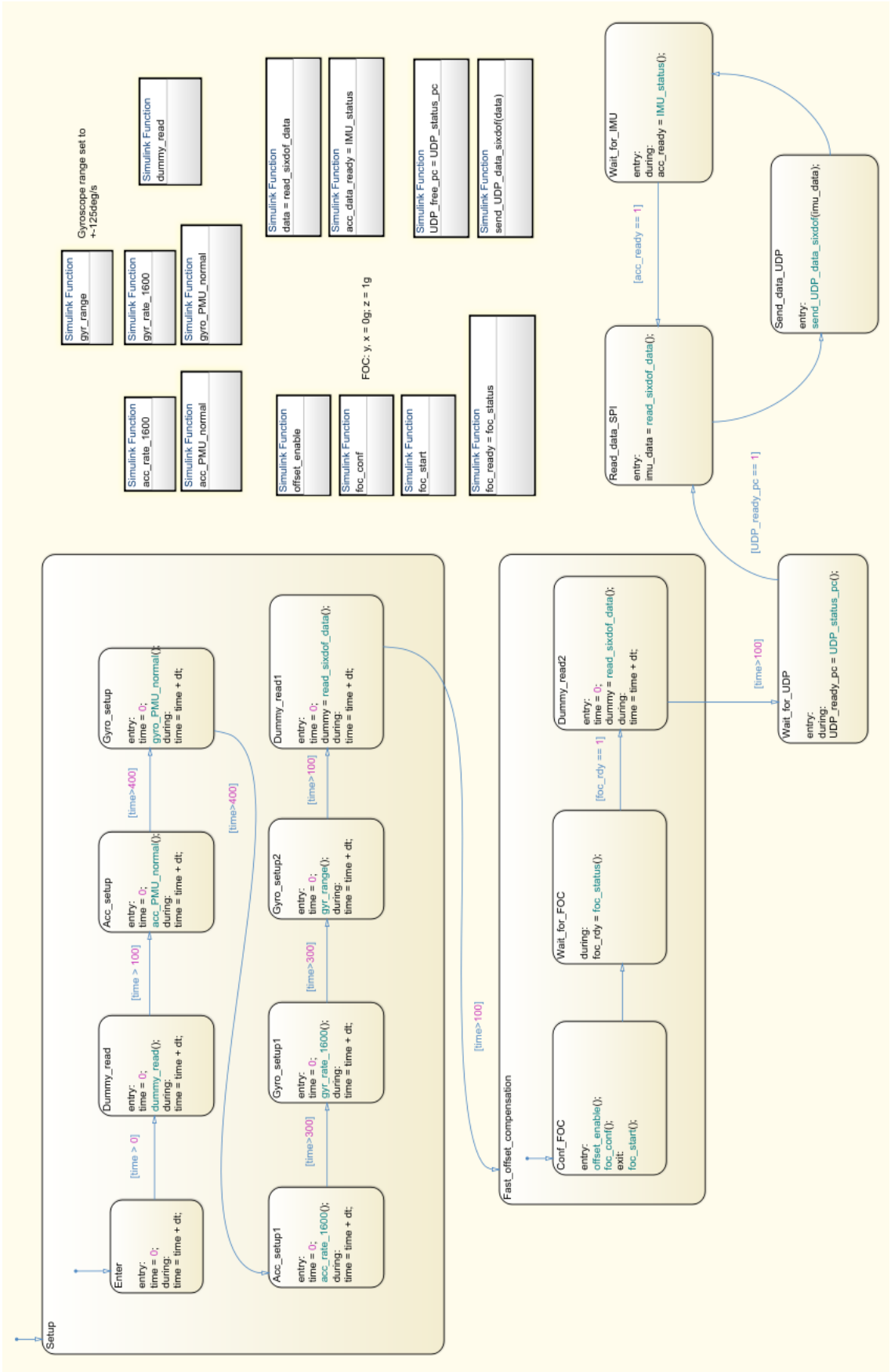
[76] M. Wollschlaeger, T. Sauter, and J. Jasperneite, The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0, in: IEEE Industrial Electronics Magazine, © 2017 IEEE, Vol. 11, No. 1, pp. 17-27.

[77] S. Zanero, Cyber-Physical Systems, in: Computer, © 2017 IEEE, Vol. 50, No. 4, pp. 14-16.

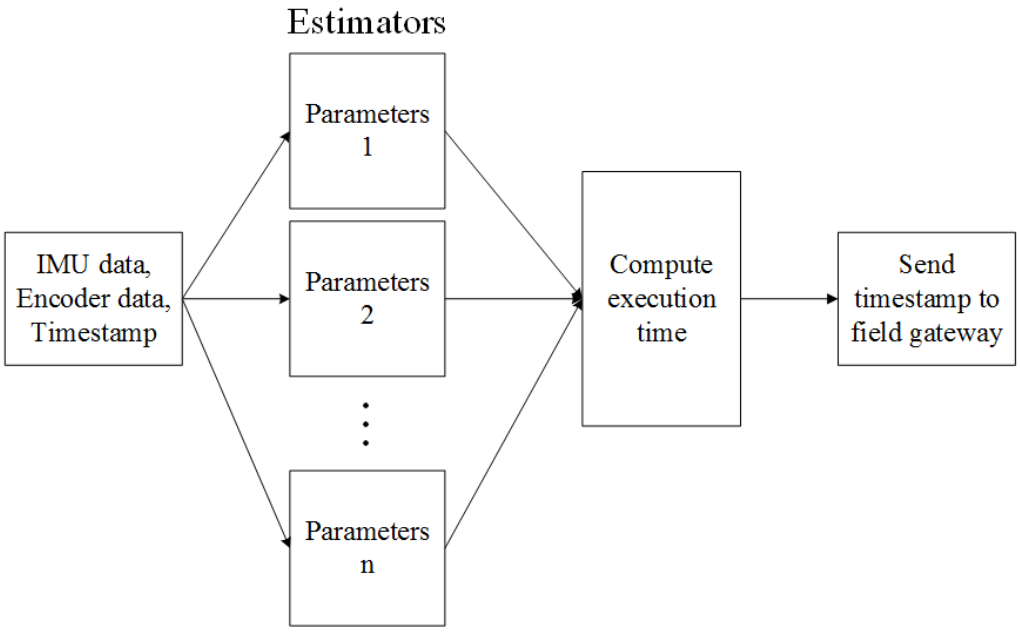
[78] Q. Zhang, L. Cheng, and R. Boutaba, Cloud computing: state-of-the-art and research challenges, in: Journal of Internet Services and Applications, Springer London, Vol. 1, No. 1, 2010, pp. 7-18.

[79] X. Zhang, E. Peltola, and J. Mattila, Joint angle estimation for floating base robots utilizing MEMS IMUs, in: 8th IEEE International Conference on Robotics, Automation and Mechatronics (RAM), IEEE, accepted, 2017.

APPENDIX A: STM SENSOR NODE STATEFLOW CHART



APPENDIX B: THE ALGORITHM EXECUTED IN THE CLOUD



In the cloud, the sensory data was received and used as input to a number of parallel state estimation algorithms with different sets of parameters. The state estimation was used to find out the deformation of the flexible beam compared to a rigid one. A finite element method based model and a dynamic state observer were used to compute the beam's deformation at the IMU positions [52]. A timestamp was also received in the sensory data packet. The time to execute the parallel estimations was computed and the timestamp was sent back to the field gateway, where the RTT could be computed. In the tests, the number of parallel estimators was 3.