FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Azure IoT Edge Proof of Concept

Francisco Tomé Neto Queirós

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: António Miguel Pontes Pimenta Monteiro

July 19, 2019

Azure IoT Edge Proof of Concept

Francisco Tomé Neto Queirós

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Jorge Alves da Silva
External Examiner: Doctor Helena Cristina Coutinho Duarte Rodrigues
Supervisor: Doctor António Miguel Pontes Pimenta Monteiro

July 19, 2019

Abstract

A rise in technologies is allowing mass communication to open ways for the use of these new technologies to improve services delivered by management software in enterprises and other groups, specifically Cloud Computing and Edge Computing. Edge Computing has been seen as an alternative to Cloud Computing due to limitations it brings, like strong dependency on centralization, which can cause excessive stress in computer networks and high amounts of network latency. However, Edge Computing needs more devices spread out geographically, therefore it's necessary to develop technologies to manage these devices. To this end, Microsoft has developed Azure IoT Edge, a tool that tries to make this task easier and also bring other features. This research looks to expose the qualities and limitations of this tool and develop prototypes to use it. The prototypes were developed in the context of a corporation, and results were then measured and evaluated with this context in mind. The results obtained are then shown and discussed, and lastly, the conclusion and what future work can be done to improve upon the work done and the context surrounding it, along with establishing some good practices.

Resumo

Um aumento em tecnologias que permitem comunicação em massa têm aberto portas para o uso destas tecnologias para melhor o serviço fornecido por software de gestão em empresas e outros grupos, especificamente computação na "nuvem" (Cloud Computing) e computação na "fronteira" (Edge Computing). Edge Computing tem vindo a ser uma alternativa ao Cloud Computing devido a algumas limitações provenientes do uso dessa tecnologia e arquitetura, como uma alta dependência em centralização, que pode causar um esforço exagerado em redes de computadores e causa grandes quantidade de latência nessas mesmas redes. No entanto, Edge Computing depende de mais dispositivos espalhados geograficamente, logo é necessário desenvolver tecnologias para a gestão destes dispositivos. Para este fim, a Microsoft desenvolveu Azure IoT Edge, uma ferramente que tenta facilitar esta tarefa e também trazer novas funcionalidades. Esta investigação visa procurar as qualidades e limitações desta ferramenta e desenvolver protótipos que a usem. Os protótipos foram desenvolvidos no contexto de uma empresa, e os resultados foram medidos e avaliados com esse contexto em mente. Os resultados obtidos são depois mostrados e discutidos, e no fim, a conclusão e que trabalho futuro pode ser feito para melhorar o trabalho feito e o contexto que o rodeia, e juntamente definir algumas boas práticas.

Acknowledgements

First I would like to thank my supervisors, Miguel Monteiro from the Faculty of Engineering of the University of Porto and João Santos from Critical Manufacturing for aiding me in doing this research. I would like to thank my teaching institution, the Faculty of Engineering of the University of Porto, for five years of notable learning experiences which I'm sure will be a strong foundation for my upcoming years as a professional. Also Critical Manufacturing for having accepted me in for the opportunity to develop this dissertation with them. I've learned a fair share of things in my short time there and were a good help to get the work done for this project. Now, my colleagues and friends, especially those who were with me during my time at Critical Manufacturing. And last but not least, my family, my parents and siblings, which have also been very helpful in every step of the way.

Francisco Tomé Neto Queirós

Contents

CONTENTS

List of Figures

List of Tables

LIST OF TABLES

Abbreviations

AML	Azure Machine Learning
AMQP	Advanced Message Queuing Protocol
DCS	Distributed Control Systems
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IPv6	Internet Protocol version 6
IaaS	Infrastructure as a Service
IoT	Internet of Things
MES	Manufacturing Execution Systems
MQTT	Message Queuing Telemetry Transport
OS	Operating System
PaaS	Platform as a Service
RFID	Radio-frequency identification
RTT	Round-trip time
SAS	Shared Access Signature
SCADA	Supervisory Control and Data Acquisition
SDK	Software Development Kits
SaaS	Software as a Service
TCP	Transmission Control Protocol

Chapter 1

Introduction

Industrial automation is the main context for this dissertation, inserted inside the growing world of Industry 4.0 and Internet of Things (IoT), which are promising paradigms with significant foreseen economical impact [LBK15].

Industrial production enterprises have a need for increased efficiency and productivity. This need gave rise to automated industry and the software used for the management of the machinery. To this end, custom software solutions were created, usually incorporated in a class of IT solutions designated as Manufacturing Execution Systems (MES).

MES focus on optimizing the efficiency of production on the factory floor, where as an Enterprise Resource Planning (ERP) system has value in more managerial roles such as human resources or accounting. For this reason, MES need to be responsive to changes in the system to keep the production line running efficiently.

Additionally, there have to be mechanisms for MES to receive information about the status of machinery. For this purpose, different types of architectures and systems were and are developed like Supervisory Control and Data Acquisition (SCADA) systems and Distributed Control Systems (DCS). To acquire the computational power for these tasks, enterprises often resort to architectures like Cloud Computing that provide high scalability, redundancy and availability. One downside of Cloud Computing that is particularly important for manufacturing environments is latency. To mitigate this issue, an extension to Cloud Computing was developed.

Instead of having a centralized processing unit, one can distribute many processing units closer to the nodes of the network that provide the data. These nodes then process and filter the data and send the relevant information to higher levels, like the MES. This architecture is called Edge Computing.

1.1 Motivation

In edge computing, due to the large number of devices, it's necessary to manage all of them. To aid in this task, Microsoft developed a tool in its cloud platform, Azure, for creating and managing edge devices in a network. This tool is named IoT Edge. Critical Manufacturing, interested in learning more about this tool, proposed this dissertation theme. Critical Manufacturing develops software that is used for production lines. They have an internal solution that is used to distribute data and events from and to edge devices, but it requires a heavier software payload and a custom infrastructure.

Additionally, customers might already use Microsoft Azure solutions in their network and infrastructure. This is an untapped resource if unused. Using Microsoft Azure IoT Edge provides the opportunity to offer a solution that is more lightweight, and that also adapts more easily to the already existing infrastructure of the client.

1.2 Goals

As for goals, it's necessary to further investigate the working conditions for the tool, its functionalities and how to deploy software solutions developed on top of this tool. After that, develop feature parity with the solution provided by Critical Manufacturing. And along with these experiments, measure metrics that define usefulness of the tool, like performance, and some good practices.

1.3 Expected benefits

This research has the expected benefit of creating an infrastructure for messages through Microsoft Azure IoT Edge, which allows the use of already deployed Microsoft Azure technologies for easier deployment and integration, while testing for metrics in performance and scalability and learn techniques to use with this tool.

1.4 Structure

This document contains four other chapters:

- Bibliographic Review: This chapter describes the technical aspects surrounding several themes that affect the environment in which the research focuses on, detailing how each impact the problem and why they are important.
- Problem description: Here, the problem presented is described and the approach taken to solve it, while presenting some details on decisions made.
- Implementation: Further details on the solution are presented in this section, attempting to explain every step taken to reproduce the solution and how it was integrated in the target environment.

- Conclusion: Summary of the work done, conclusions taken from the work, value created from research and possible future work in the area.

Chapter 2

Bibliographic Review

2.1 Industry 4.0

Industry is the activity surrounding the manufacturing of materials from the extraction and processing of raw materials. In the pursuit for better methods of manufacturing, there have been technological adoptions that, in some cases, revolutionized how humanity thought about manufacturing.

Some of these moments, when key technology adoption was achieved and generalized, have been recognized and called "industrial revolutions". The possibly most known example is the first industrial revolution, relating to technologies like the steam engine and allowing a better utilization of mechanization processes. The other past industrial revolutions are the use of electrical energy, and digitization.

The evolution in this digitization era has spawned a conversation about a new industrial revolution, where digitization is more present in the shop floor of a factory. This stage is most commonly called "Industry 4.0", referencing it as being the fourth industrial revolution. It will be supported by aspects such as cyber-physical systems, heterogeneous data, knowledge integration [LFK+14], modular and efficient manufacturing systems and products, controlling their own manufacturing process [Lu17].

The result of this new paradigm is intelligent, "smart", machines and materials, resulting in a new way of doing things in industrial production [LFK+14] that promises "substantially increased operational effectiveness" and "entirely new business models, services and products" [HPO16].

Right now there is a centralized approach of manufacturing, where this central component is where all management is done, deciding what steps to take and when to take them and how they mesh with the steps of other resources being processed at the same time. By enabling connections between several actors, such as people, machines and resources, it's possible to move away from that, and create a model where the products themselves know their production state, know

what processes they went and have to go through, and manipulate the machinery to achieve the goal. [HPO16]

There are several factors that are inducing this change as mentioned by [LFK+14]:

- Shorter development periods
- Individualization on demand
- Flexibility
- Decentralization
- Resource efficiency

Industry 4.0 is also closely related to other emerging ideas such as Internet of Things. [HPO16]

2.2 Manufacturing execution system

For three decades, companies have spent resources in investments in favor of information system to achieve productivity gains. This resulted in a steady growth for the enterprise information system market. Some types of information systems, namely enterprise resource planning systems, have become a very widespread phenomenon in the information system market. However, these systems have historically neglected the shop floor perspective.

Due to the complicated and intricate requirements for each enterprise's manufacturing processes, the solutions created for industrial operations support have normally been custom software. Data is monitored and controlled for each production stage. It can be hard to aggregate and maintain this data.

However, due to this difficulty, there was a push to integrate all these systems into singular solutions. These packages are called "Manufacturing Execution Systems" (MES).

This type of software arose from demand to fulfill several requirements, similar to the ones mentioned for Industry 4.0: reactivity; quality; respect of standards; reduction in cost; and deadlines. These requirements have a focus on manufacturing activities, which are then reflected on the functions of manufacturing execution systems. [SdUAP09]

According to Manufacturing Enterprise Solutions Association International White Papers on manufacturing execution systems, there are eleven key features [Int97]:

- Resource Allocation and Status
- Operations/Detail Scheduling
- Dispatching Production Units
- Document Control
- Data Collection/Acquisition

- Labor Management
- Quality Management
- Process Management
- Maintenance Management
- Product Tracking and Genealogy
- Performance Analysis

2.3 Internet of things

In an increasingly connected world, with easier and better access to networking services such as the Internet, there's a new paradigm being developed where nearly every device will eventually be connected to a network. This concept is called "Internet of Things" (IoT). However, given how recent this term is, compounded by the broad and abstract nature, its definition is still in discussion. Other topics still in discussion are: what it is, how it works and why it exists. Essentially it boils down to a large amount of devices identifying and connecting themselves using a big set of communication technologies. From this comes "an inter-connected world-wide network based on sensory, communication, networking, and information processing technologies" [LXZ14], as seen in figure 2.1. Technologies used for this purpose are plentiful, notably:

- IEEE 802.11 (Wi-Fi)
- IEEE 802.15.1 (Bluetooth)
- Internet Protocol version 6 (IPv6)
- Wireless broadband technologies such as 4G
- Radio-frequency identification (RFID)

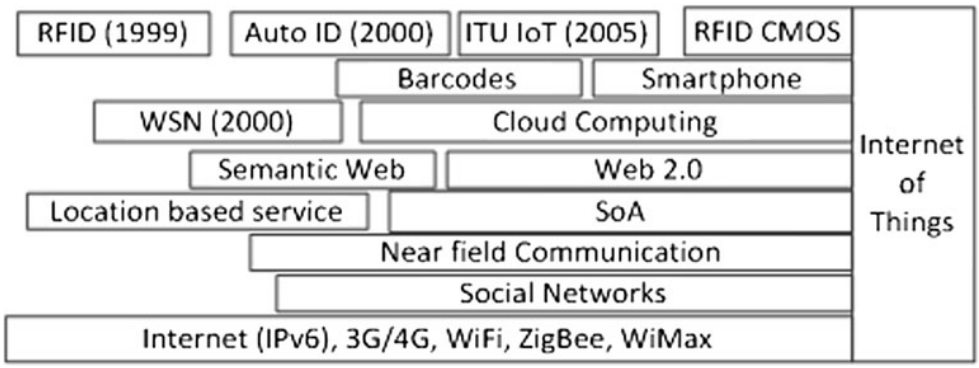

Figure 2.1: Internet of Things enabling technologies. [LXZ14]

Due to the high degree of interoperability required to make such an idea more ideally implemented, there has been a significant push to generate standards around the technologies utilized by internet of things, mainly radio-frequency identification and communication protocols [LXZ14]. Besides interoperability, standardizing these aspects as much as possible might yield increased compatibility, reliability and effectiveness when dealing with communications at a global scale [LXZ14]. Governments and private institutions alike have recognized the added value of a well invested internet of things infrastructure and therefore, there have been sizeable investments in this area [LXZ14].

2.4 Cloud computing

Due to the high availability of connectivity and the constant rise of desire for computation power, the space for a new market focused on computation as a service is showing up.

Much like a utility resource like water and power, we're reaching a point where individuals want to frequently use computation resources as to being relevant to consider offering computation charged like a utility, based on how much is used, during how long. There have been methods to try and solve this problem, namely Cluster Computing and Grid Computing. However, these are very restrictive in terms of access and complexity.

A new paradigm focused on what is served rather than how is served, abstracts the underlying hardware from the end user. Unlike the methods mentioned, for which access is more restricted, limited, and controlled, Cloud Computing has a bigger focus on being accessible for any individual; the service is always ready to serve the end user, with on-demand scaling, with the requirement of being accessible from the Internet [BYV+09, Xu12].

Cloud Computing is powered by data centers with typically powerful physical machines running virtualization software where different virtual machines are running for potentially different users, on the same physical machines. This allows for flexibility in the system image used and creates another layer of security between the different computing environments.

Cloud Computing can comes in several types of services:

- Software as a Service (SaaS), which provide a set of functionalities like email clients, chat services, among others.
- Platform as a Service (PaaS), that give tools to develop higher level software.
- Infrastructure as a Service (IaaS), simply offer low-level surfaces, very close to simply offering hardware.

among other types. This ubiquitousness of Cloud Computing servicing also coins a new term "everything as a service" or XaaS [Xu12].

Some world wide providers of cloud computing services include:

- Microsoft Azure [Mic19g]
- Amazon Web Services [AWS19]

- Google Cloud [Goo19]
- IBM Cloud [IBM19]

among others.

2.5 Edge computing

Cloud Computing has become a very important and powerful tool in the daily lives of people and enterprises as it has evolved from its genesis, giving humanity services like email, social networks, file hosting, and many other services. In order to keep Cloud Computing sustainable, there has been a lot of work done in developing software solutions as infrastructure for the Cloud Computing world [SCZ+16].

With the advent of Internet of Things, there has been a big surge in data produced from physically spread devices. Some Internet of Things scenarios can require low response times, dealing with private data or deal with amount of information too large for wide area networks to handle. For these cases, it's believed that cloud computing is not sufficient [SCZ+16].

As data is produced by spread out devices, it is logical that it would be more efficient and perhaps effective to process this data closer to these devices instead of distant data centers. To try and solve this problem, the idea of Edge Computing evolved, where any device in the network pathway to the cloud can instead process and respond to the requester in order to answer faster and avoid pressure on inward, towards the cloud, network. In some cases this might even be necessary like the mentioned cases of reduced latency and private data, where it's against the user's will to send data to data centers.

Some literature use Fog Computing and Edge Computing to mean similar things or at least interchangeably [YLL15, CZ16, LYZ+17], while other takes them as having different meanings [DB16].

2.6 Microsoft Azure

Among the many solutions for cloud computing that emerged world-wide, Microsoft has its own solution called Azure. Microsoft mentions that Azure is "an ever-expanding set of cloud computing services" [Mic19d]. Azure has services in several areas:

- Internet of Things
- Artificial Intelligence
- Web Services
- Data Warehousing

and more [Mic19a].

Azure's ability for computation and other factors has already been discussed in several pieces of literature with [SGH15, Han11, YZGH12, RDCN12] as some examples.

The main interest in Azure for this project are its internet of things capabilities, notably Azure IoT Hub and Azure IoT Edge.

2.7 Microsoft Azure IoT Hub

Microsoft Azure IoT Hub is a Microsoft Azure "managed service, hosted in the cloud, that acts as a central message hub for bi-directional communication between an IoT application and the devices it manages".

Azure IoT Hub primarily uses three different communication protocols:

- Advanced Message Queuing Protocol (AMQP)
- Message Queuing Telemetry Transport (MQTT)
- Hypertext Transfer Protocol Secure (HTTPS)

AMQP and MQTT are commonly used protocols in the Internet of Things context, based on publish/subscribe models of communication [AFGM+15, BRSF18], which makes them good choices in order for Azure IoT Hub to align with the rest in terms of communication protocols. These can also be used to communicate over WebSocket, if the ports for AMQP or MQTT are not available, as WebSocket works through HTTP or HTTPS ports which are commonly available ports.

To connect to this platform, a device can use several methods [Mic19e]:

1. develop software written using their software development kits (SDK), available in several languages including:
   - C
   - .NET languages (C#, among others)
   - Java
   - Python
   - JavaScript using Node.js
2. in the cases where using these software development kits are impossible or not preferable, Azure IoT Hub can be communicated with using native implementations of the aforementioned protocols.
3. when neither of the previous two solutions are viable, it's also possible to use other protocols by extending Azure IoT Hub to support each new needed protocol.

For security, Azure IoT Hub supports a few authentication methods for devices:

1. presenting a shared access key related to the device to authenticate, although this method is not recommended.

2. presenting a shared access signature (SAS) related to the device to authenticate. This is more recommended than a shared access key.

3. using X.509 certificates. This is the most secure method, but has disadvantages such as being more complicated to implement and need to take precautions to make it possible to replace expired certificates on running devices.

It's also possible to connect directly to the Azure IoT Hub instead of a device identity. The methods available are the same, however for the shared access key, a hub actually have the ability to have several keys and assign different permissions for each of the keys:

- Registry read: The registry keeps track of information about the devices registered in the Azure IoT Hub. This permission allows for the connection, authenticated using the access key, to read this information from the Azure IoT Hub.

- Registry write: Much like the registry read, but instead of allowing information to be read, it also allows to be written. This allows to create and delete devices.

- Service connect: This allows the connection to be used to write and listen to endpoints at the cloud level, for example to listen to all devices.

- Device connect: Allows the connection to write and listen to endpoints associated to specific device identities. So this is necessary if one were to use an Azure IoT Hub connection to send a message in place of a device connection.

When sending a message, it is encapsulated in a data structure that contains other data and the original message is encoded into a binary format. Some data is generated by the Azure IoT Hub, for example, a message identifier, destination of the message and timestamp of reception. Additional data can be added by the emitter of the message. This user data can be read without deserializing the original message.

Messages received by Azure IoT Hub can be routed using rules defined by the user based on message contents and other data, allowing messages to reach different exit points. Different applications can listen to different exit points of Azure IoT Hub, depending on the goal of each application and the significance of a message coming out of a certain exit point.

A potentially valuable aspect of Azure IoT Hub is the possibility to link incoming messages to other services in the Azure ecosystem related to areas such as machine learning, from Azure Machine Learning (AML) [Mic19l], predictive maintenance [Mic19h], actuators [Mic19b], among other potentially valuable services, depending on the user's needs. To use these services, custom endpoints can be specified that represent each of these services, to which then messages can be routed by rules, as mentioned before.

To prevent abuse, increased stability and additional layer to their business model, Microsoft employs usage limits, such as quotas and throttles. Quotas are usage limits on a long period, in this case daily limits on messages sent by the user. Throttles are usage limits on short periods to avoid overloading the service. These limits affect messages differently from user messages, for example how many devices can connect per second.

The quotas depend on the subscribed service and are described by Microsoft as in table 2.1 shown by [Mic19i].

Table 2.1: Azure IoT Hub pricing

Edition type	**Price per IoT Hub unit per month**	**Total number of messages per day per IoT Hub unit**	**Message meter size**
B1	€8.433	400,000	4 KB
B2	€42.165	6,000,000	4 KB
B3	€421.650	300,000,000	4 KB
Free	Free	8,000	0.5 KB
S1	€21.083	400,000	4 KB
S2	€210.825	6,000,000	4 KB
S3	€2,108.25	300,000,000	4 KB

2.8 Microsoft Azure IoT Edge

Microsoft Azure IoT Hub runs its logic exclusively on Azure data centers, thus having a somewhat centralized architecture. Essentially, Azure IoT Hub is a case of cloud computing, mentioned in the previous section 2.4. However, Microsoft also has another tool named Azure IoT Edge which serves as a solution that can be used to instead implement edge computing, also mentioned in the previous section 2.5.

This tool is powered by a piece of software called the Azure IoT Edge runtime, which has to be installed and run in whatever machine intended to be used as an IoT Edge device. Azure IoT Edge can be used on most operating systems (OS) that support containers, however, Microsoft has defined two tiers of operating systems, according to their level of support [Mic19k]:

- Tier 1 operating systems "can be thought of as officially supported". Microsoft includes these operating systems in automated tests and provide installation packages. This tier's operating systems according to Microsoft as of writing this document are shown on table 2.2.

- Tier 2 operating systems "can be thought of as compatible with Azure IoT Edge and can be used relatively easily". Microsoft claims to have done some testing on these operating systems or knows of a partner that was successful at running Azure IoT Edge on that operating system and installation packages for other operating systems may work on these operating

Table 2.2: IoT Edge Tier 1 Operating Systems

Operating System	**Platform**
Raspbian-stretch	ARM32v7
Ubuntu Server 16.04	AMD64
Ubuntu Server 18.04	AMD64
Windows 10 IoT Enterprise, build 17763	AMD64
Windows Server 2019, build 17763	AMD64
Windows Server IoT 2019, build 17763	AMD64
Windows 10 IoT Core, build 17763 (Public preview)	AMD64

systems. This tier's operating systems according to Microsoft as of writing this document are shown on table 2.3.

Table 2.3: IoT Edge Tier 2 Operating Systems

Operating System	**Platform**
CentOS 7.5	AMD64 and ARM32v7
Debian 8	AMD64 and ARM32v7
Debian 9	AMD64 and ARM32v7
RHEL 7.5	AMD64 and ARM32v7
Ubuntu 18.04	AMD64 and ARM32v7
Ubuntu 16.04	AMD64 and ARM32v7
Wind River 8	ARM32v7
Yocto	ARM32v7

The containers' guest operating system can differ from the host operating system of the physical machine. For example, the installation package for Windows has a flag that installs Linux-based containers instead. A machine running a particular operating system can run containers of a different operating system but Microsoft states that this setup should only be used for development purposes whose target is the guest operating system and only gives support for configurations where the guest operating system is the same as the host operating system.

This tool is a simpler and locally run Azure IoT Hub. Some tasks, like authentication, still need to be forwarded to the cloud-based Azure IoT Hub, which is a source of truth for the IoT Edge runtime [Mic19f].

Besides having the Azure IoT Edge device sending messages to Azure IoT Hub using its own authentication, there can be downstream devices that also have an identity in Azure IoT Hub but use the Azure IoT Edge device as a gateway to Azure IoT Hub. Microsoft has predicted three patterns for this behavior [Mic19c]:

- Transparent gateway: the downstream devices have all the prerequisites to connect to the Azure IoT Hub directly, meaning they support one of the main protocols, MQTT, AMQP or HTTP and possess an identity on the hub, but instead use the Azure IoT Edge gateway

device. In this mode, the devices establish a connection that masks the presence of the gateway, making both the devices and users interacting with the device unaware of the gateway. This unawareness is what gives the name "transparent".

- Protocol translation gateway: the downstream devices do not support one of the main protocols, and therefore use some other protocol. The edge device will use an IoT Edge module that communicates with the downstream devices and translates their protocol into a usable protocol to then communicate with the Azure IoT Hub. This approach has a couple of downsides:
  - all data will go to the Azure IoT Hub as belonging to the edge device since the downstream devices are not being authenticated and sent with their identification. To mitigate this issue, some identifying value can be added in the message or as a message annotation. This value can then be interpreted at some other point in the process, for example, when the message is consumed. This identifying value only exists from the programmer's point of view, so Azure IoT Hub still sees the messages as coming from the Azure IoT Hub. Users from outside the gateway only see the gateway as an entity and must interact with the downstream devices by sending messages to the edge gateway, assuming the gateway will be able to understand these messages are meant for the downstream devices.
  - since the Azure IoT Hub only sees one device, even if the edge device has several downstream devices, the throttle limits and quotas will be measure for just one device, which can be a significant limitation if the devices in the group are supposed to send a large amount of messages and thus require that the quotas and throttles are applied by device, which does happen in the other two patterns, transparent gateway and identity translation gateway.
- Identity translation gateway: similar to protocol translation gateway but also takes care of associating an identity from Azure IoT Hub to the device and translating other mechanisms from Azure IoT Hub such as device twins and methods. In this case, the Azure IoT Hub will receive messages as coming from the downstream devices, so it's not necessary to add values to identify the source of the messages like the protocol translation gateway pattern. Users from outside the gateway can see the downstream devices and interact with them, by having the gateway abstracted during the communication. It will not be evident that communication is happening through a gateway.

There have been performance tests made in the past that showed Azure IoT Edge can keep one of the promises of edge computing, reduced bandwidth that leads to decreased network stress. According to those tests, latency was similar to its cloud counterpart [DPW18].

A unique aspect of Azure IoT Edge is the presence of modules. Modules are units of code that can be run within the Azure IoT Edge runtime. Modules have several capabilities to communicate

with other modules in the same Azure IoT Edge runtime, with the runtime itself and the Azure IoT Hub the edge device is connected to:

- Message routing: Modules within the Azure IoT Edge runtime can use APIs to send and receive message through named exit and entry points.

To make most of these configurations, the IoT Edge runtime relies on receiving a document called deployment manifest that specifies a set of parameters, among them:

- Custom variables that modules can then read off the manifest, mostly setup variables.
- List of modules to be loaded, including their Uniform Resource Identifiers (URI) to be extracted from.
- Uniform Resource Locator (URL) of container registries used, alongside authentication data required to access them.
- Routes for messages coming off modules to follow. Messages mostly come from modules and are meant to either be routed into other modules or sent to the "upstream" which is the Azure IoT Hub. Modules can have different exit and entry points and can have names. The routes use these names to specify the path to be taken. Messages can go into different entry points of a module for a different behavior from the module.

The deployment manifest can be sent remotely to the edge device. The new deployment manifest is compared to the old deployment manifest. New modules are automatically downloaded and initialized and modules that are no longer used are terminated and removed.

2.9 Conclusion

These fields of research and development work on architectural aspects that are in constant evolution but that is also believed to be the foundation for a new technological system. Computers are getting increasingly more connected and powerful, with a set of sensors to relay information about the physical world around us.

While there is already a significant amount of research on the base ideas of this project such as internet of things and edge computing, not much research was found on Microsoft's tools, especially Azure IoT Edge, making additional research valuable.

Chapter 3

Problem description

3.1 Problem

Edge computing is a promising computational paradigm. To take advantage of this, Microsoft developed Azure IoT Edge, a tool that is used to manage edge devices. Microsoft Azure has the ability to register different physical computers that are running a runtime software, IoT Edge. This runtime software will be used by Microsoft Azure to manage the device.

With the IoT Edge runtime, it's also possible to add custom software bundled in units called modules. These modules can either be accessed from public module repositories, private module repositories or a development project and deployed into a device.

Critical Manufacturing, a company whose main product is a manufacturing execution system, and is the proponent of this research task, was interested in the possible use of IoT Edge to improve aspects related to their product.

The goals set for this research were:

1. describe the system (hardware and software) and Azure IoT Edge licensing/subscription requirements as well as the Azure IoT Edge pricing model.

2. capture and describe the Azure IoT Edge capabilities.

3. perform a proof-of-concept of converting one existing Connect IoT driver and Controller as an IoT Edge module.

4. Extend the proof-of-concept by adding a simple example for telemetry and machine learning module.

The first item, "describe the system (hardware and software) and Azure IoT Edge licensing/-subscription requirements as well as the Azure IoT Edge pricing model" will be based on researching documentation that Microsoft supplies on these tools. Since Azure IoT Edge depends on Azure IoT Hub, the requirements for Azure IoT Edge will extend from Azure IoT Hub.

For the proof-of-concept, the goals ask to convert a Connect IoT Driver and Connect IoT Controller into IoT Edge Modules. However, these elements of the Critical Manufacturing MES, Connect IoT Driver and Connect IoT Controller are heavily integrated into their software, making extracting these components out into Azure IoT Edge modules complicated and ultimately not valuable to Critical Manufacturing.

Instead, an API was developed to interface with the Azure IoT Hub and Azure IoT Edge and then a Connect IoT Driver was developed that calls upon this API to interact with the Azure IoT Hub and Azure IoT Edge. This driver can be executed directly from the MES, making Azure IoT Hub available to be used within the software.

To develop this API, exploring the capabilities of the Azure platform and the pieces of interest, Azure IoT Hub and Azure IoT Edge, was important. After this research, the proof-of-concept prototype was developed in several parts:

- Software to receive messages: This would be integrated into some custom software in a terminal for some end user or managing software and used to capture and consume messages sent to Azure IoT Hub directly or through Azure IoT Edge. The custom software can then do whatever it needs to do with the incoming messages.

- Message sender: The role of the message sender is to abstract the SDKs of Azure IoT Hub and Azure IoT Edge into a simpler API for the purpose of the prototype. The SDKs expose complex APIs for a large set of scenarios that aren't always necessary.

3.2 Methodology

The prototype will be based on two main components, a piece of software to send messages and another to receive messages, to and from Azure IoT Hub, respectively. With these components in place, another step was to allow the message sender to use an IoT Edge runtime as a proxy for messages. And finally, integrate the prototype within Critical Manufacturing's Connect IoT module for use within the manufacturing execution system.

Azure IoT Hub and Azure IoT Edge have official software development kits in several languages, and Node.js with JavaScript was selected because this was more compatible with Critical Manufacturing's code base, because it is written in TypeScript for the components used for this research. However, it was told that C# code was also possible to be used but would require some kind of translation or interface layer and would result in extra work to get the code running successfully, so it was not selected.

For results, messages were sent to the Azure IoT Hub and then read on the receiver software. The time between sending and receiving is measured, giving a round-trip time of the message while changing several variables for each run: messages per second, the protocol used, if the messages are relayed through the IoT Edge runtime or not.

The protocols used were MQTT and AMQP as these are among the most used in internet of things scenarios and are easily compatible with Azure IoT Hub and IoT Edge, supplying their own

implementations through their SDKs. Relaying through the IoT Edge runtime was important to determine if the extra advantage of having an edge gateway would significantly increase issues, like latency and lack of scalability, which can reduce its added value or viability.

Just one computer was used for all scenarios, both as a message sender, receiver and IoT Edge device. The computer has the following specifications:

- CPU: Intel Core i7-3610QM
- RAM: 8GB
- OS: Windows 10 Enterprise build 17763

In figures 3.1 and 3.2 it's possible to see the normal architecture for the two tools, Azure IoT Hub and IoT Edge.

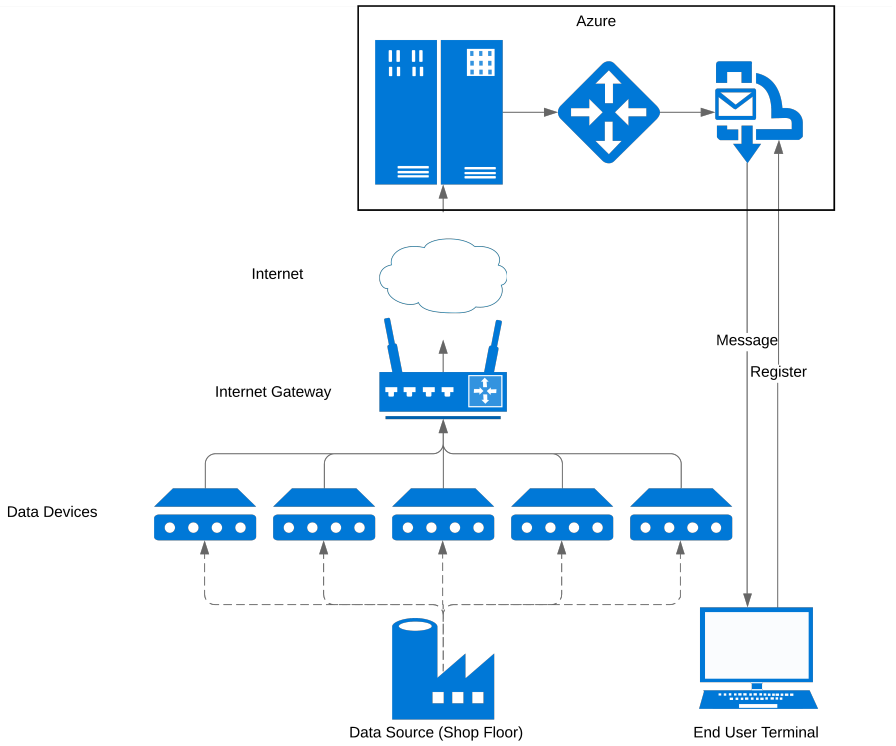

Figure 3.1: Diagram of Azure IoT Hub prototype Architecture

In figure 3.1 we can see a "Data Source" which represents any entity or place that contains the "Data Devices". The "Data Devices" will be any machine that has the purpose of sending data to the Azure IoT Hub. They generate data, be it their internal temperature, or some other data that can be measured and it's of interest to store or process.

The "Internet Gateway" is whatever device the local network's devices use to connect themselves to the Internet, like a router or modem. This creates a connection to the Internet that can be

then be used to access the Azure IoT Hub servers. Here the message is received by the service and stored temporarily.

Simultaneously, there can be a device, a "End User Terminal", which is a machine that its purpose is to retrieve the messages sent by the "Data Devices" and act upon the messages. The "End User Terminal" is capable of running custom code and thus do whatever needed to the incoming messages. To do this, the "End User Terminal" must first connect to the Azure IoT Hub and register its intent to receive messages that are received by the service. While the program that registered the machine is active, whenever a message is received, a function is called with the message's contents. From this point, the "End User Terminal" has received the message and can do whatever it needs to do, from storing into a file, database or directly consume the message.

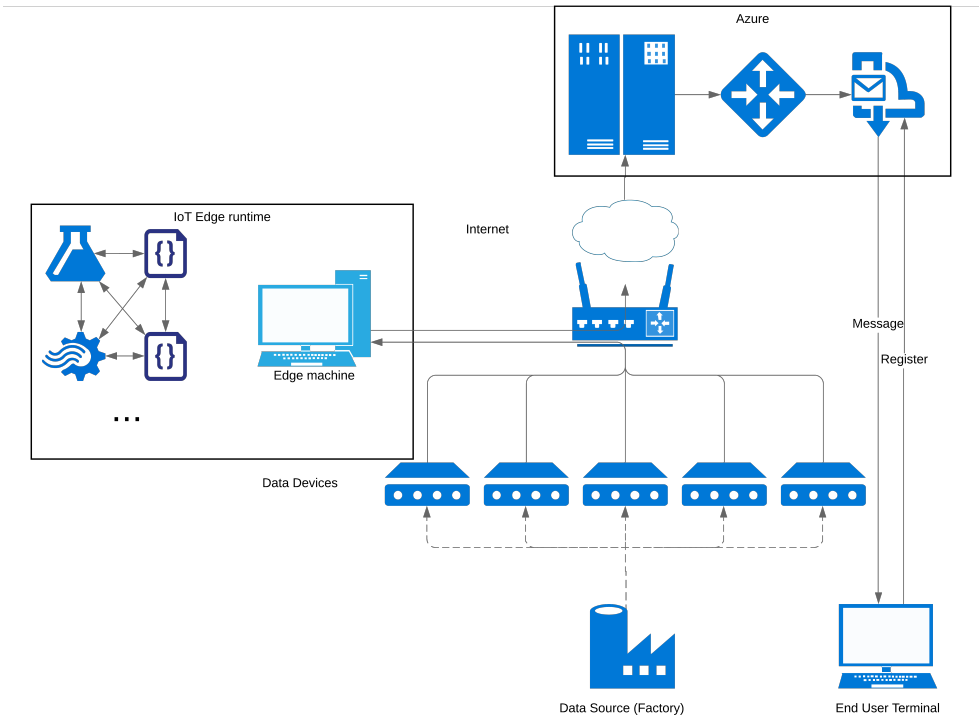

Figure 3.2: Diagram of Azure IoT Edge prototype Architecture

In figure 3.2 it's possible to see that the architecture is very similar, except that instead of the message being directly sent to the Internet once it reaches the Internet gateway, the local network first redirects the message to the Azure IoT Edge device. The device can then use different pieces of software that act upon this received message. One of these pieces can then send the message to the Azure IoT Hub, although this is not obligatory, and only done if it's the intention of the programmer. The Azure IoT Edge device can be used for filtering, for example. The Azure IoT Edge device can also send a message to the Azure IoT Hub that differs from the original message. Some Azure products can be exported into pieces of software that the Azure IoT Edge device can run locally, such as Azure Machine Learning and Azure Stream Analytics, among others. The rest

of the model is similar to the Azure IoT Hub architecture. The "End User Terminal" still connects to the Azure IoT Hub in the same way.

Chapter 4

Implementation

4.1 Back-end application

Azure IoT Hub and IoT Edge are tools where one of the goals is to ultimately send received messages that must then be relayed, even if altered or processed, to some consumer of those messages.

A back-end application must then be made to receive those messages and show an applicable message to the user. In this specific prototype the messages are some string that would simulate a common use for these tools, like telemetry.

This back-end application would then be the same entity as the "End User Terminal" shown in pictures 3.1 and 3.2, which will register itself in a messaging system on Azure IoT Hub and then receive messages that Azure IoT Hub receives and that are sent to the back-end application.

The back-end application was developed using TypeScript with Node.js by following a guide published by the Azure IoT Hub team, geared towards developing an application for this purpose, to read messages coming into the Azure IoT Hub. This guide uses a piece of example code presented in a public source code repository of examples to use the Node.js SDK for Azure IoT Hub [Git19]. The example code is originally written in JavaScript as of writing this document, so it was first converted into TypeScript, which is a programming language that is a syntactical superset of JavaScript and has some features that JavaScript lacks, with the most notable example being optional static typing, to align with the rest of the developed code. It was also necessary to fill in a variable with a connection string that includes the hostname, shared access key and shared access key name for the target Azure IoT Hub.

In the developed solution and test condition, the Azure IoT Hub, upon receiving the message from the message sender, routes the message to an endpoint that is the default and built-in endpoint for Azure IoT Hubs and is connected to a service called Event Hubs. Therefore, to connect to this endpoint to then receive the message, the back-end application uses a set of APIs developed to connect to Event Hubs, which is contained in a npm [npm19] package.

The back-end application creates a connection using one of the APIs to connect to the Event Hub associated to the endpoint in the Azure IoT Hub and registers a callback that will be called whenever the Event Hub receives a message. The callback is custom code, allowing for any manipulation or decision to be made based on the message.

4.2 Message sender

Devices in the shop floor will produce data relevant to their function, such as telemetry. Telemetry is the automatic measurement and sending of data, primarily some kind of measurement from a sensor, from remote equipment to other equipment that will process and monitor that data. This data should be processed or stored to extract value from it, to assist in maintenance, identifying problems in the manufacturing process, or other aspects that can be relevant to the entities managing the factory.

In order for data to be processed for this project, it must be sent to the Azure IoT Hub. To do this, the data must be sent to some computer that is present locally and that can receive the data through some channel, such as serial ports or the local network.

The computer will then be running software that is capable of sending the data to the Azure IoT Hub for eventual processing or storage. The source machines are represented as the "Data Devices" in figures 3.1 and 3.2, and the computer could either be the machines themselves, if they have internet connectivity and it's in the interest of the owners to do so, or some other machine that serves as a bridge between them and the "Internet Gateway" shown in the same above figures. During research, both cases were developed, but ultimately the important part is the code that takes care of the communication with the Azure IoT Hub, as the behavior before that step depends heavily on the intentions of the user.

In order to send the data to the Azure IoT Hub, it's necessary to either have authentication data for the Azure IoT Hub or devices registered within the Azure IoT Hub. The difference between these two methods lies in how static the device association is. If for a specific need, it's only necessary to send messages with a reduced set of devices identities, these identities can be created beforehand, have their shared access keys extracted and loaded into the application. Then for each of the identities, a different connection can be created and then used when needed. There are some cases where there's a need for higher flexibility in the device identification. Either the names of all the devices are not known beforehand or it's too many to create a connection for each device. For these cases, a shared access key, that provides access to the Azure IoT Hub as a while, can be obtained and given to the application. The application will now be able to access the devices by name, or even create new devices if necessary.

In the Azure IoT Hub SDKs there are functions that can be used to aid in these tasks. It might vary slightly between SDKs for different languages, as they're not completely the same, although being similar in structure and behavior. Some languages lack features that others languages have

that prevents following particular structures for the SDK. During the research, the language chosen, JavaScript, didn't have its Azure IoT Hub SDK lacking any critical features. The C# language SDK is among the most feature versions of the SDK.

4.3 Integration with IoT Edge

In order to integrate with IoT Edge as the goals required originally, there was some effort into developing modules and studying how to work with them.

In the end, modules didn't seem practical for the goal of Critical Manufacturing. However, IoT Edge was still integrated by using the edge device as a gateway through which messages are sent. Transparent gateway paradigm was used to receive messages from devices and then send them to Azure IoT Hub through Azure IoT Edge.

To send messages through IoT Edge with the transparent gateway paradigm, it is necessary to do three things:

1. Generate X.509 certificates.
2. Configure the sender to trust these certificates.
3. Install the certificates on the IoT Edge runtime.

To generate the certificates for testing purposes, Microsoft provides a guide and tools to create temporary certificates. The details of how these certificates are created will vary depending on the user's need, but the material provided was sufficient to get the connection with IoT Edge runtime working, to perform the necessary measurements.

4.4 Integration with Critical Manufacturing MES

The original goals intended to create modules to replace two systems in the Critical Manufacturing MES, the Connect IoT Driver and Connect IoT Controller.

These two systems are parts on a component in the MES related to interact with physical devices and belong to a larger family of systems.

The Connect IoT Driver is responsible for having logic that is able to translate messages from the device to the MES and from the MES to the device, so that the MES can receive information about the status of the machine, for example, and also the MES can send messages to the device, which the device will interpret as if it was programmed to perform its task, from the factory controller or personnel.

The Connect IoT Controller is another layer that interacts with the Connect IoT Driver to do higher level logic. The Connect IoT Controller has one or more Connect IoT Drivers associated. The Controller is programmed using visual scripting and blocks called tasks. Tasks have inputs and outputs, the output values are typed and they can only be assigned to other inputs if the types

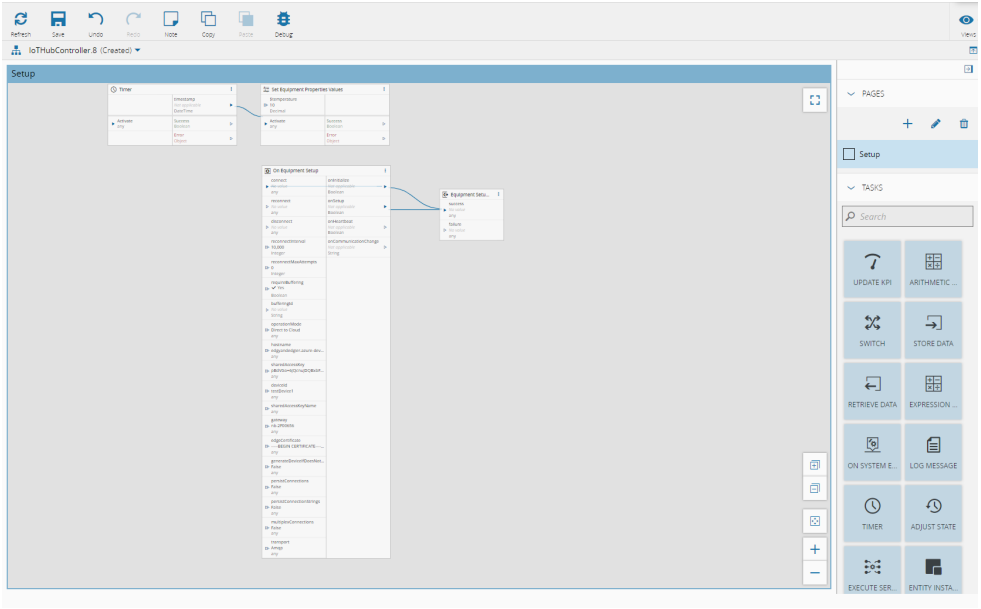

Figure 4.1: Connect IoT Controller visual scripting editor.

match. However, the visual scripting environment also allows the controller to use type converters to fix these incompatibilities.

The visual scripting environment has some tasks that are used to interact with the driver such as receive new values from the device, send new values to the device and other functionalities.

To integrate Azure IoT Hub and Azure IoT Edge with the MES, there was two possible work scenarios. Either develop a task or a Connect IoT Driver to take care of the communication with the Azure IoT Hub, where the Azure IoT Hub would be like a device to the MES. From these options, a Connect IoT Driver was developed. It's still possible to send messages from a device to the Azure IoT Hub using this option because a Connect IoT Controller can have several Connect IoT Drivers, so a driver's incoming messages' task's output can be connected to the developed driver's outgoing messages' task's input.

Future controllers can use this driver to send messages to Azure IoT Hub.

4.5 Measurement software

It's interesting to make measurements of objective metrics about the object of study. The same applies to Azure IoT Hub and Azure IoT Edge, leading to the effort to measure latency between these two solutions to understand if Azure IoT Edge shows the same level of scalability as Azure IoT Hub.

A program was made that combines the back-end application with the message sender to receive and send messages. However, to measure time, a data structure is created where time is stored. Each message will have an identifier, obtained from an incremental integer, sent along with it and when the message gets sent, the data structure receives a new entry that associates

the message identifier to a new timestamp obtained from the `Date.now` function in JavaScript. This function returns a numerical value that represents the amount of milliseconds since January 1, 1970.

When the message is received on the receiving portion of the program, the program takes another timestamp using the same function and obtains the timestamp taken when sending the message from the aforementioned data structure and then subtracts the older timestamp from the latest timestamp to create a measurement for the data.

As each measurement is generated, they are written onto a file which was later used for analysis.

Some variables were changed to see how the time would change in reaction to these differences:

- Rate of messages: To check if and when messages would start getting queued. This would represent a lack of capacity of message consumption from some point in the chain.
- Transport protocol: Used both AMQP and MQTT to see if there would be a difference in latency with these protocols and if they would reach different rate of messages before problems started to appear.
- Either directly connected to Azure IoT Hub or going through edge device: To test if the Azure IoT Edge can have the same level of scalability as the cloud solution.

4.6 Results

For both cases of running the measurement through a direct connect to the Azure IoT Hub or through the Azure IoT Edge device, the protocol was varied between AMQP and MQTT and the message rate was varied between 1, 2, 4, 8, 16, 32, 64 and 128 messages per second. This generated 32 different test scenarios.

4.6.1 AMQP protocol directly to Azure IoT Hub

In figures 4.2 and 4.3, it's possible to see that at most message rates, the message latency stayed around the same amounts of 60 to 120 milliseconds, with some outliers, with the exception of the 16 messages per second series which generated significantly higher values of 50 to 300 milliseconds.

Table 4.1 also shows a correlation between the message rate and the average latency.

Figure 4.4 shows how for four different series, 1, 2, 4 and 8 messages per second, messages are sent with high probability within the 66 to 68 millisecond range. For values over this range, these series start to reduce in probability, while the other series of 16, 32, 64 and 128 messages per second increase in probability.

In figure 4.9 it's possible to see a trend where the higher the message rate, the tighter the cumulative distribution function curve will be, with some exceptions, especially the 16 messages per second series, which has a much tighter curve than the other series.

4.6.2 MQTT protocol directly to Azure IoT Hub

Table 4.2 shows a correlation between higher message rate and higher minimum and average message latency.

Figure 4.6 shows that the series for 64 and 128 messages per second have very high latency in its messages, some messages taking almost 2 seconds in the 64 messages per second series and over 4 seconds in the 128 messages per second series to get a response. The 1, 2, 4 and 8 messages per second series show similar latency to each other of around 63 to 80 milliseconds, with some outliers. Series for 16 and 32 messages per second show higher latency of 65 to 160 milliseconds, with some outliers.

In figure 4.7, it's possible to see that in series for 64 messages per second and 128 messages per second, the latency is increasing as more messages are being sent. And the higher message rate of the two, 128 messages per second, has a higher rate of increase than the lower message rate. This behavior is not present in the other series. It's also possible to see some spikes in the chart that represent outliers.

In figure 4.8, the series for 1, 2, 4 and 8 messages per second are shown to have a higher probability of having between 66 and 68 milliseconds of latency. Higher message rates are less likely to have these lower amounts of latency.

In figure 4.9, the 64 and 128 messages per second series curves have a shape similar to a tilted line, which means that the probability is roughly the same throughout all the occurring latency values. In general, the higher the message rate, the tighter and lower the curve is on the chart.

4.6.3 AMQP protocol through Azure IoT Edge device

When running the tests running the AMQP protocol and having the messages go through the Azure IoT Edge device, it was not possible to run the test at higher than 2 messages per second.

According to table 4.3, running 1 message per second has higher minimum, maximum and average latency.

The box plot in figure 4.11 shows the same conclusion and also several outliers in the data points. However, most messages stay within the 65 to 90 millisecond range.

The graph for the figure 4.12 shows higher and more frequent spikes in latency for the 1 message per second test than the 2 messages per second test. However, both test keep themselves relatively stable in latency.

Most of the messages for both runs are within the 66 to 68 milliseconds range, as the figure 4.13 shows.

The graph in figure 4.14 shows that the 2 messages per second run had a bigger portion of its messages in the 65-80 milliseconds range than the 1 message per second run.

4.6.4 MQTT protocol through Azure IoT Edge device

In the table 4.4 it's possible to see a correlation between message rate and maximum and average latency.

In the box plot 4.15, the higher message rate tests are shown with very high latency, namely the 128, 64, 32 and to some extent the 16 messages per second tests. The lower message rates have very similar distribution between them, with most of their messages in the 65 to 80 millisecond range.

In the graph in the figure 4.16, the lower message rate tests stay relatively stable in message latency throughout the test, while tests with message rates starting at 16 messages per second have their latency rising as the test runs, showing a shape similar to a tilted line, indicating a linear rise in latency. The larger the message rate, the more pronounced the tilt is, indicating a faster rise in latency.

In figure 4.17 the lower message rate tests show a high amount of probability of latency between the 67 to 69 milliseconds. The higher message rate tests do not have many messages in the latency range of the graph.

In figure 4.18 it's possible to see that the 16, 32, 64 and 128 messages per second runs form a curve that approximates a straight and tilted line. In this graph, that shows that throughout the test, the latency distribution is roughly uniform for every latency in the range.

Table 4.1: Average, minimum and maximum latency for different message rates with AMQP protocol connecting directly to the Azure IoT Hub

Message rate	Minimum latency	Maximum latency	Average latency
1 message/s	65 ms	373 ms	73.052 ms
2 messages/s	65 ms	685 ms	76.610 ms
4 messages/s	65 ms	373 ms	80.980 ms
8 messages/s	65 ms	489 ms	82.434 ms
16 messages/s	61 ms	437 ms	148.360 ms
32 messages/s	67 ms	251 ms	94.769 ms
64 messages/s	62 ms	223 ms	91.430 ms
128 messages/s	60 ms	212 ms	90.837 ms

4.7 Discussion

In terms of implementation, a prototype was developed that attempts to connect a potential machine to the Azure IoT Hub in order for that machine to send messages. That prototype was then extended to support having the messages first going through a Azure IoT Edge device.

For the results, a few patterns appeared:

- In general, the higher the message rate, the higher the values of latency for its latency distribution. This is believed to be the case due to stress caused on the network, Azure IoT Hub or Azure IoT Edge system, where applicable. If one of these systems cannot handle so many

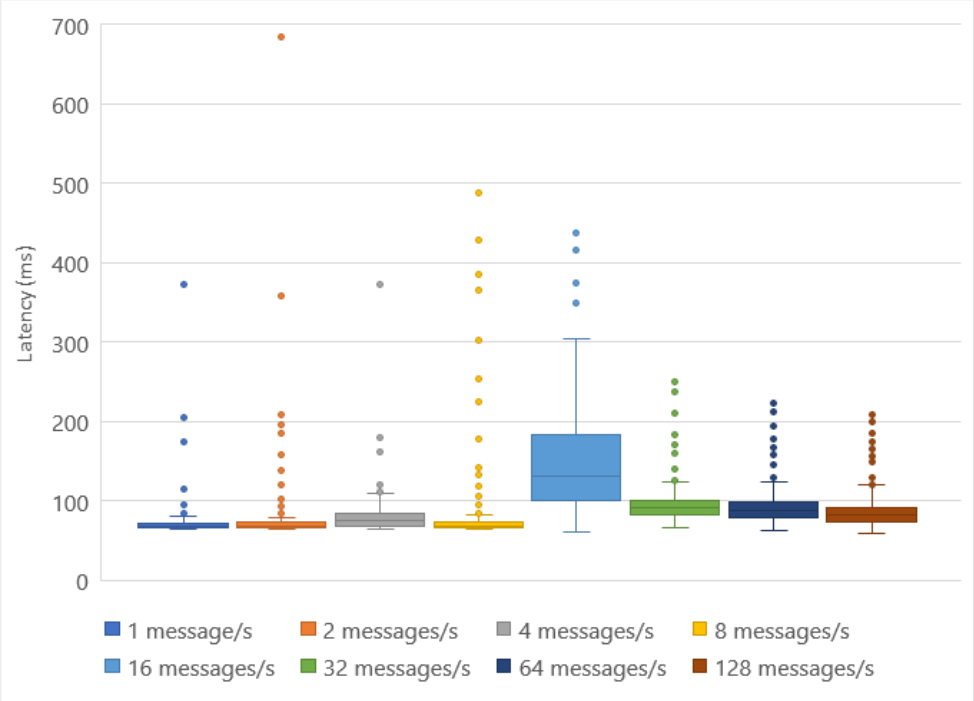

Figure 4.2: Box plot of message latency for different message rates - AMQP directly to Azure IoT Hub.

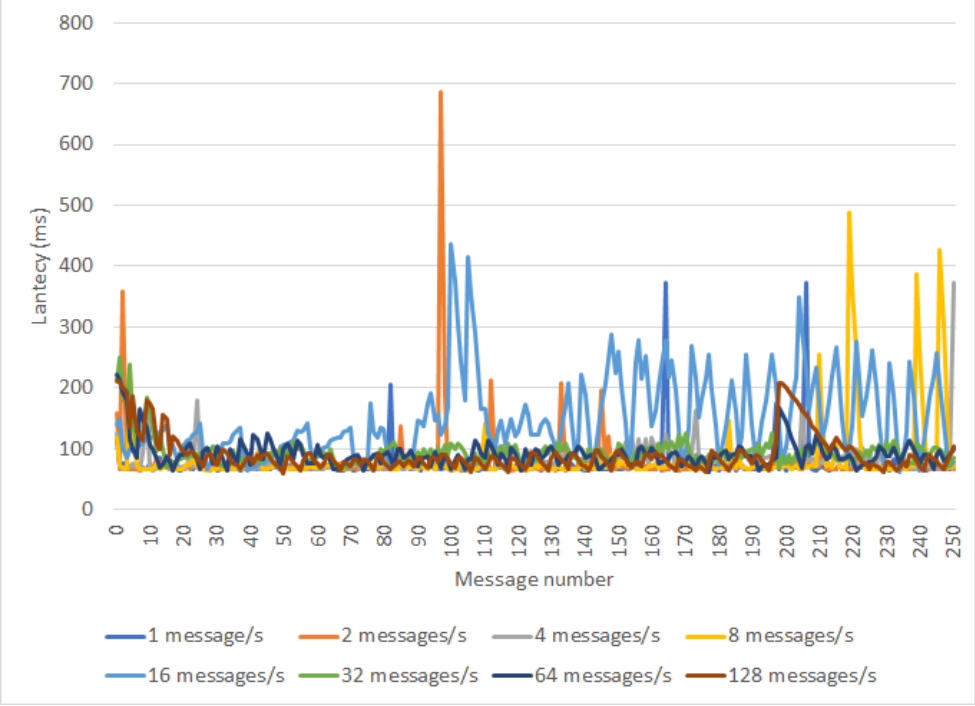

Figure 4.3: Latency of messages sent - AMQP directly to Azure IoT Hub. Message number represents the order of the message.

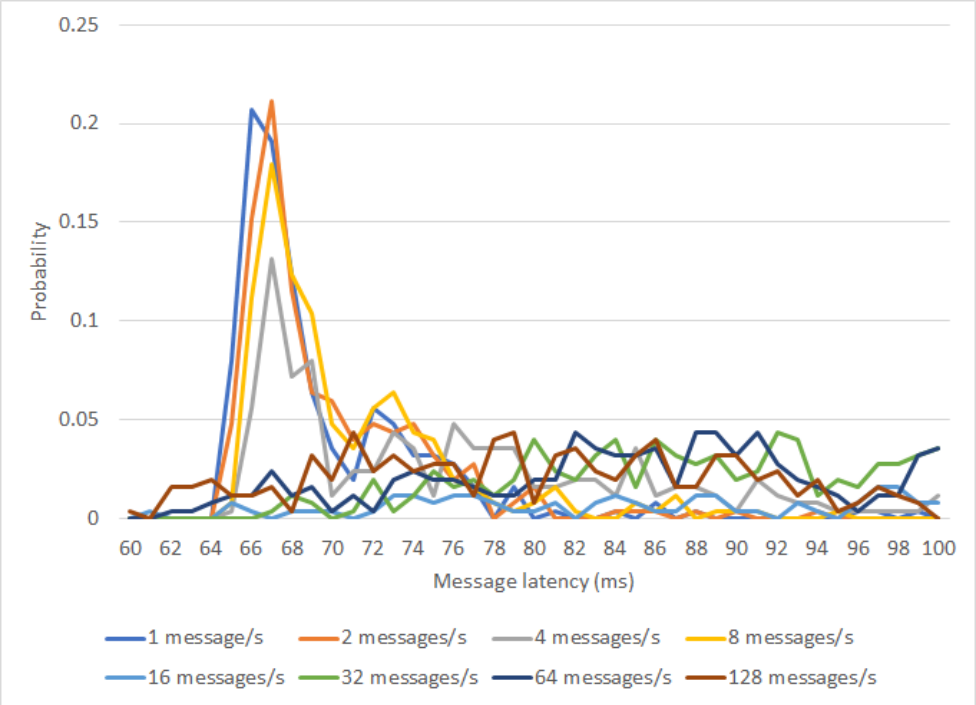

Figure 4.4: Probability of message latency for different message rates - AMQP directly to Azure IoT Hub - cropped for up to 100 milliseconds for better visualization.

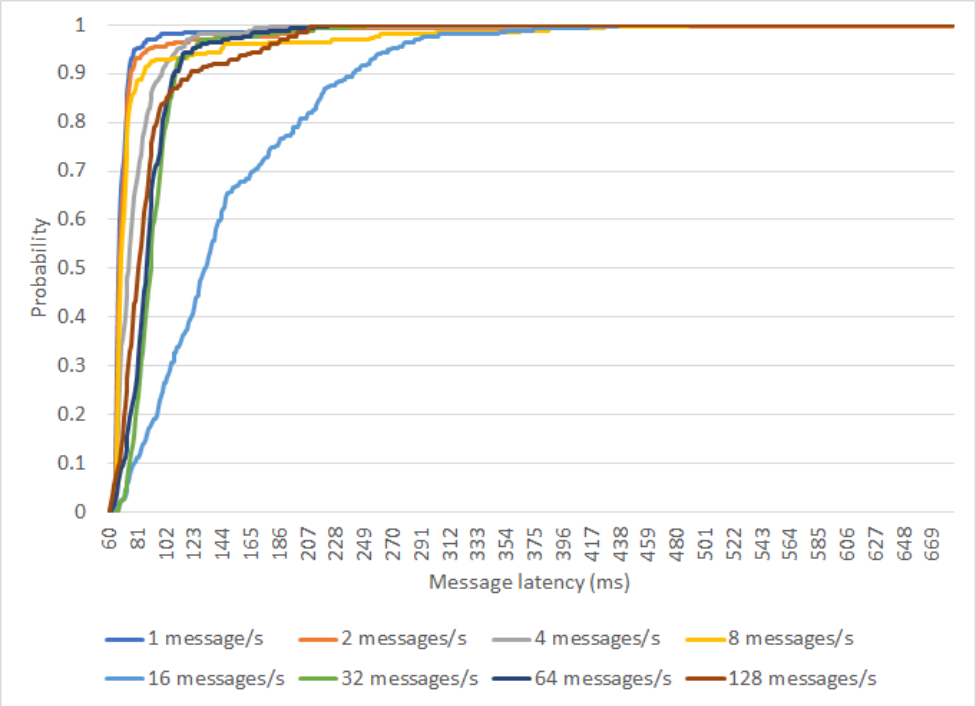

Figure 4.5: Cumulative distribution function of message latency for different message rates - AMQP directly to Azure IoT Hub.

Table 4.2: Average, minimum and maximum latency for different message rates with MQTT protocol connecting directly to the Azure IoT Hub

Message rate	Minimum latency	Maximum latency	Average latency
1 message/s	65 ms	385 ms	81.171 ms
2 messages/s	64 ms	678 ms	75.769 ms
4 messages/s	63 ms	378 ms	74.295 ms
8 messages/s	64 ms	131 ms	70.948 ms
16 messages/s	67 ms	328 ms	103.371 ms
32 messages/s	65 ms	324 ms	102.598 ms
64 messages/s	109 ms	1880 ms	973.502 ms
128 messages/s	120 ms	4070 ms	2086.920 ms

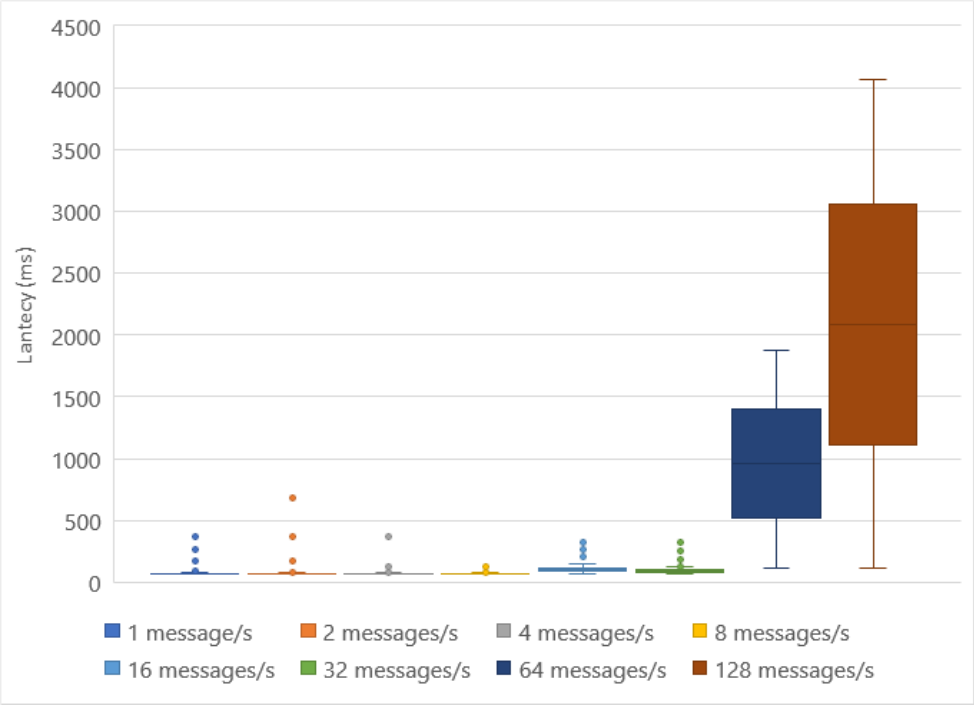

Figure 4.6: Box plot of message latency for different message rates - MQTT directly to Azure IoT Hub.

Table 4.3: Average, minimum and maximum latency for different message rates with AMQP protocol connecting through the Azure IoT Edge device

Message rate	Minimum latency	Maximum latency	Average latency
1 message/s	66 ms	819 ms	83.657 ms
2 messages/s	65 ms	705 ms	78.622 ms

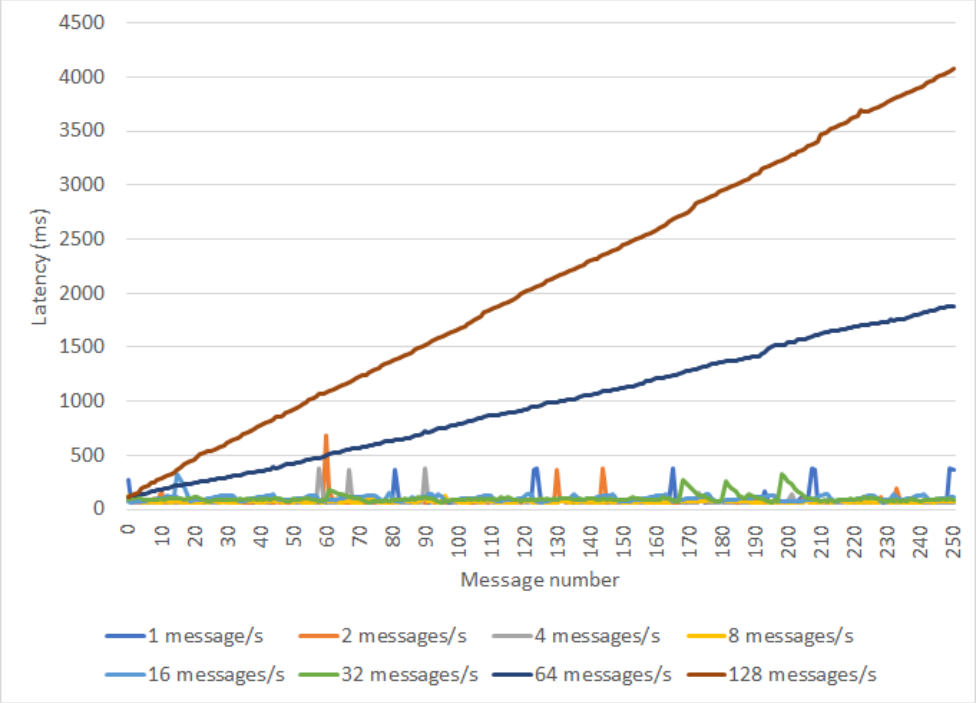

Figure 4.7: Latency of messages sent - MQTT directly to Azure IoT Hub. Message number represents the order of the message.

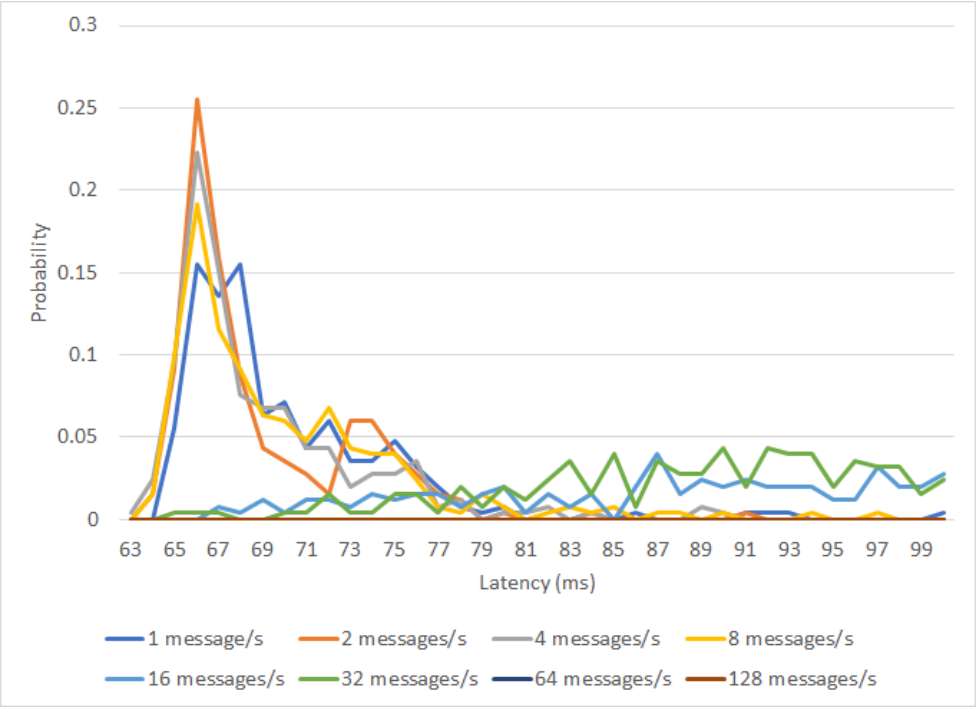

Figure 4.8: Probability of message latency for different message rates - MQTT directly to Azure IoT Hub - cropped for up to 100 milliseconds for better visualization.

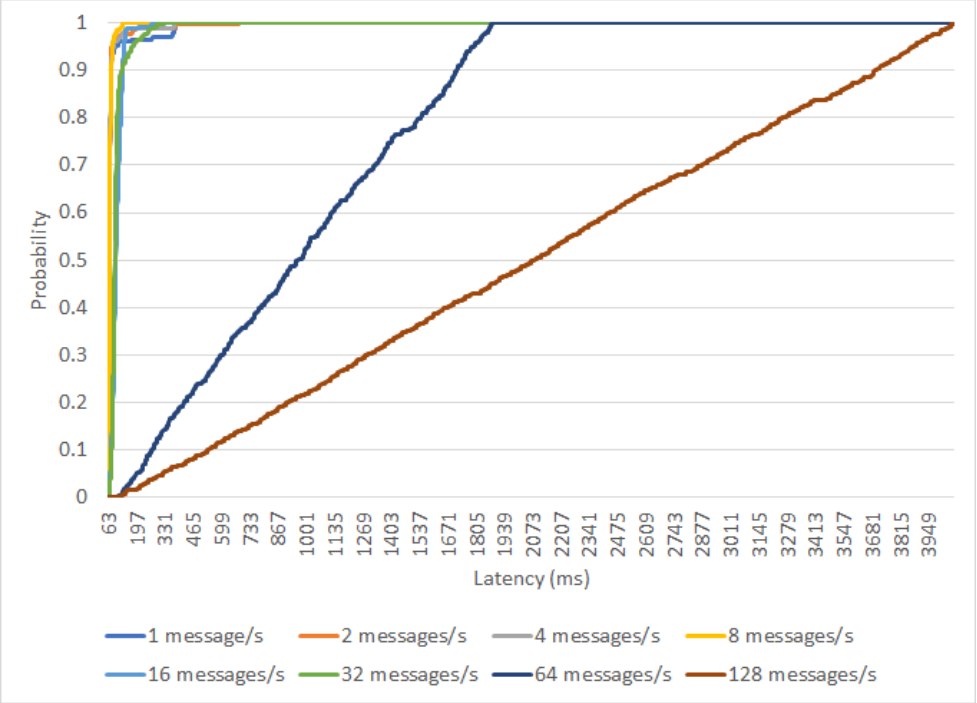

Figure 4.9: Cumulative distribution function of message latency for different message rates - MQTT directly to Azure IoT Hub.

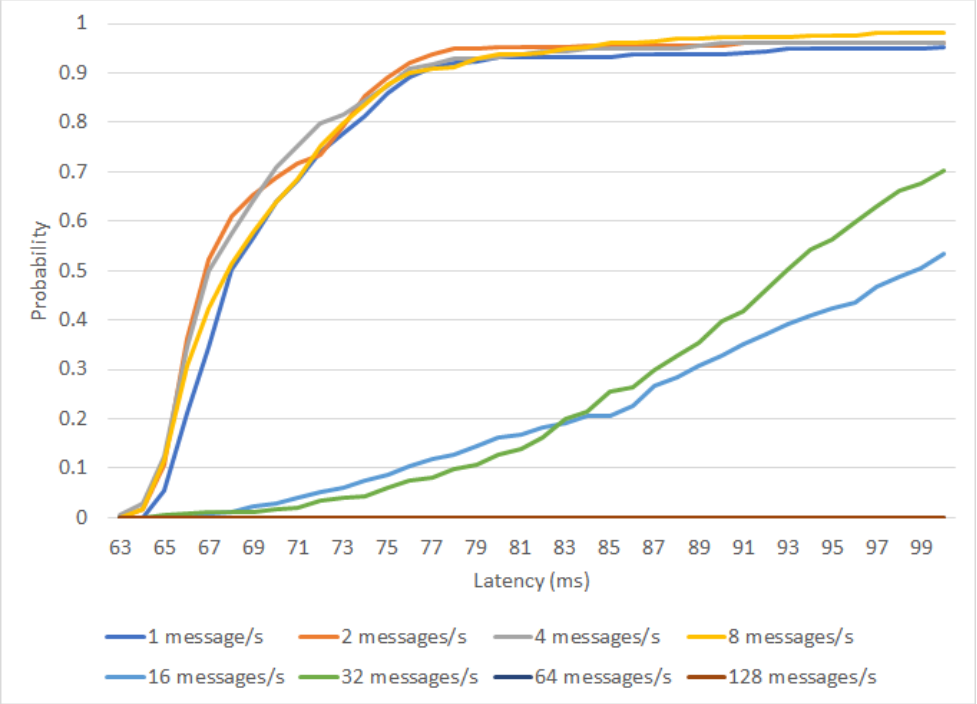

Figure 4.10: Cumulative distribution function of message latency for different message rates - MQTT directly to Azure IoT Hub - cropped for up to 100 milliseconds for better visualization.

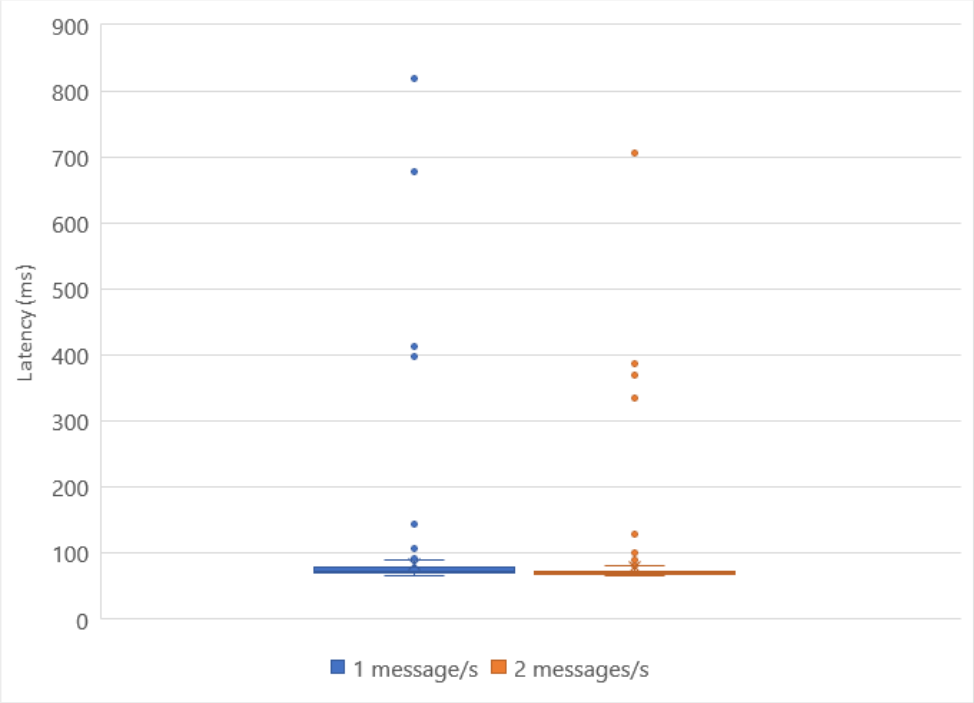

Figure 4.11: Box plot of message latency for different message rates - AMQP through Azure IoT Edge device.

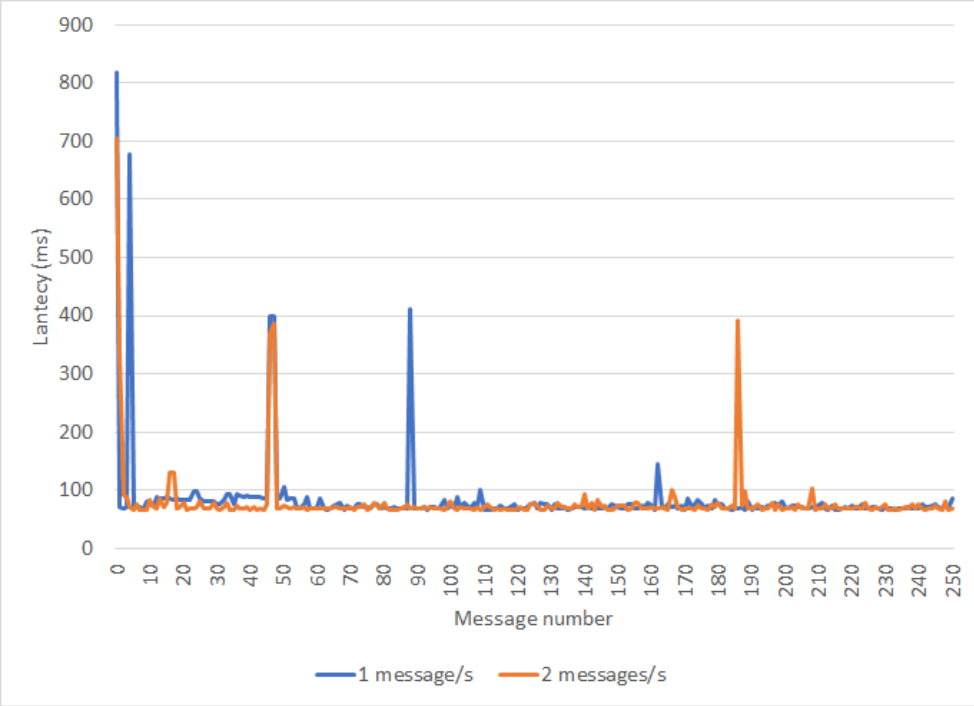

Figure 4.12: Latency of messages sent - AMQP through Azure IoT Edge device. Message number represents the order of the message.

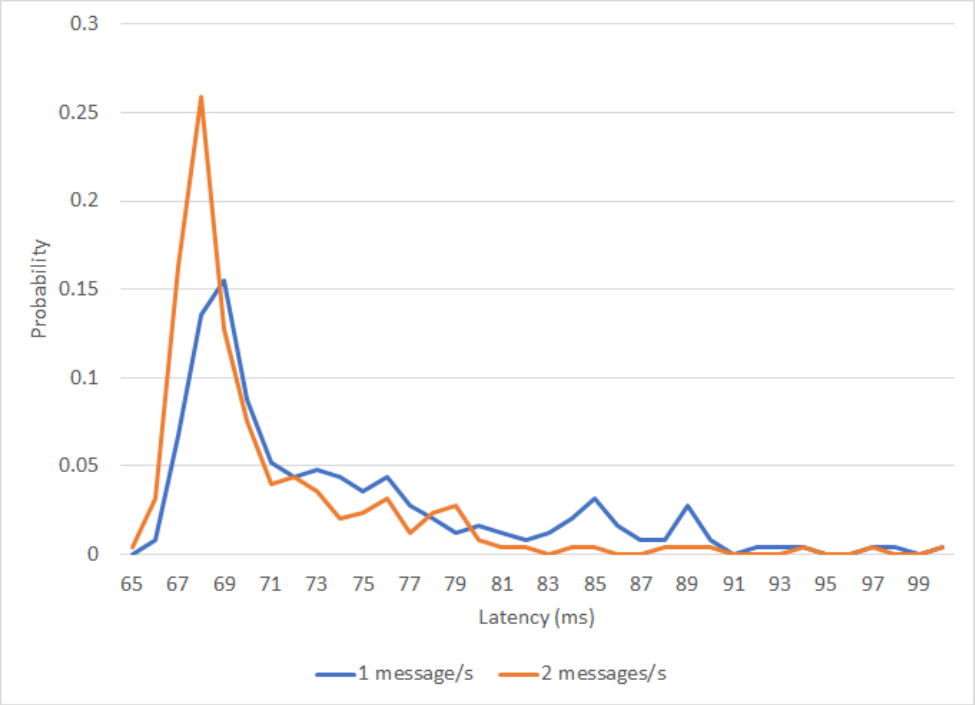

Figure 4.13: Probability of message latency for different message rates - AMQP through Azure IoT Edge device - cropped for up to 100 milliseconds for better visualization.

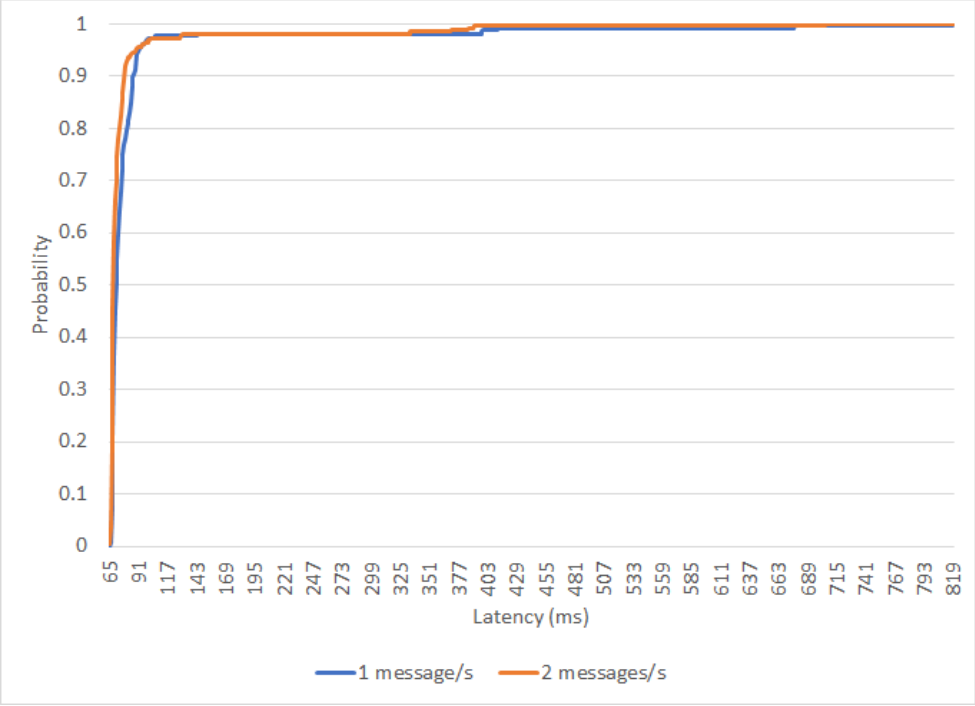

Figure 4.14: Cumulative distribution function of message latency for different message rates - AMQP through Azure IoT Edge device.

Table 4.4: Average, minimum and maximum latency for different message rates with MQTT protocol connecting through the Azure IoT Edge device

Message rate	Minimum latency	Maximum latency	Average latency
1 message/s	66 ms	687 ms	78.048 ms
2 messages/s	66 ms	384 ms	71.865 ms
4 messages/s	65 ms	693 ms	81.542 ms
8 messages/s	65 ms	608 ms	84.817 ms
16 messages/s	75 ms	2309 ms	826.82 ms
32 messages/s	71 ms	9567 ms	4843.4 ms
64 messages/s	69 ms	14274 ms	7407.5 ms
128 messages/s	70 ms	16144 ms	7807.9 ms

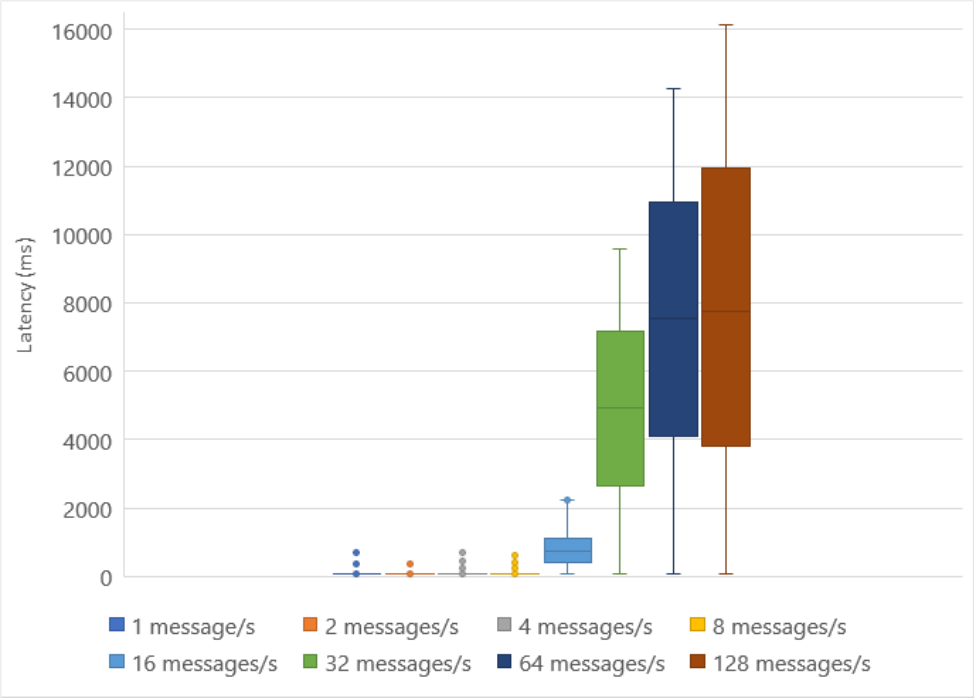

Figure 4.15: Box plot of message latency for different message rates - MQTT through Azure IoT Edge device.

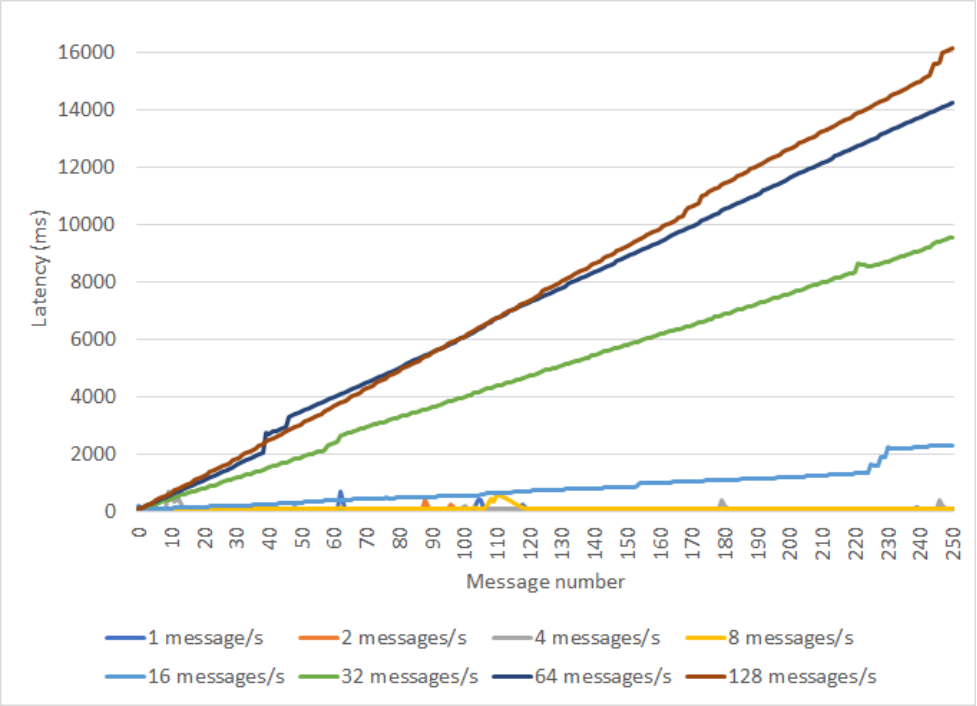

Figure 4.16: Latency of messages sent - MQTT through Azure IoT Edge device. Message number represents the order of the message.

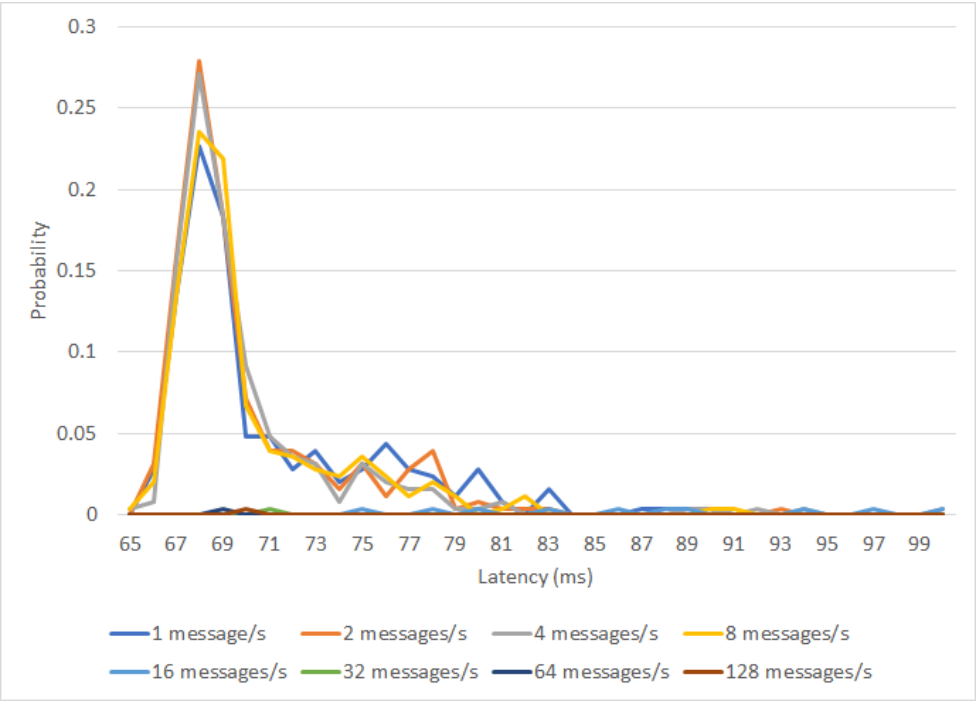

Figure 4.17: Probability of message latency for different message rates - MQTT through Azure IoT Edge device - cropped for up to 100 milliseconds for better visualization.

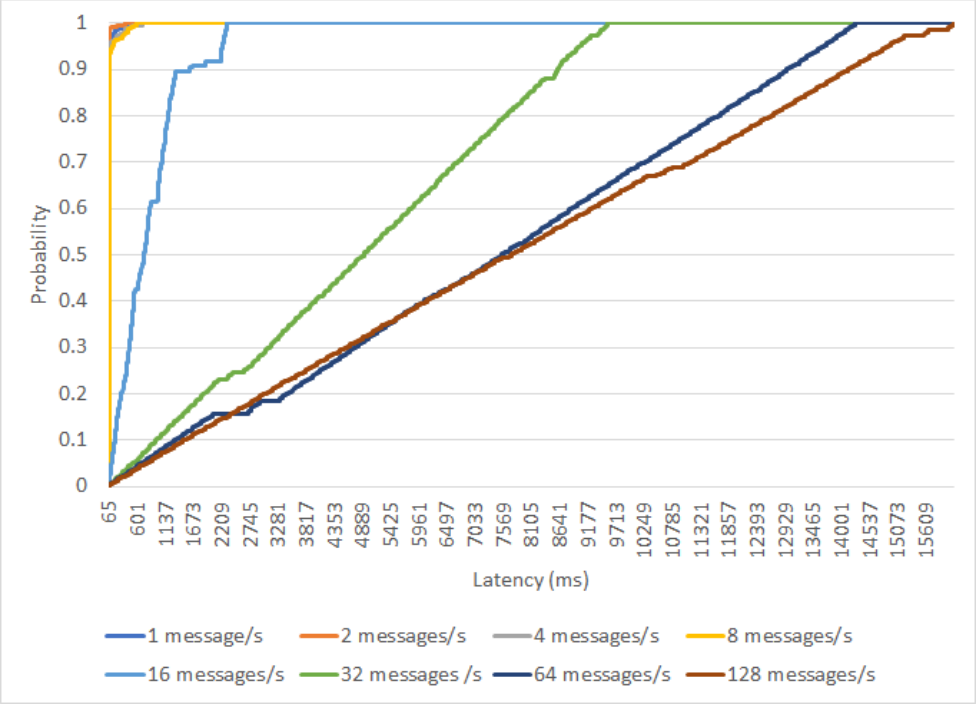

Figure 4.18: Cumulative distribution function of message latency for different message rates - MQTT through Azure IoT Edge device.

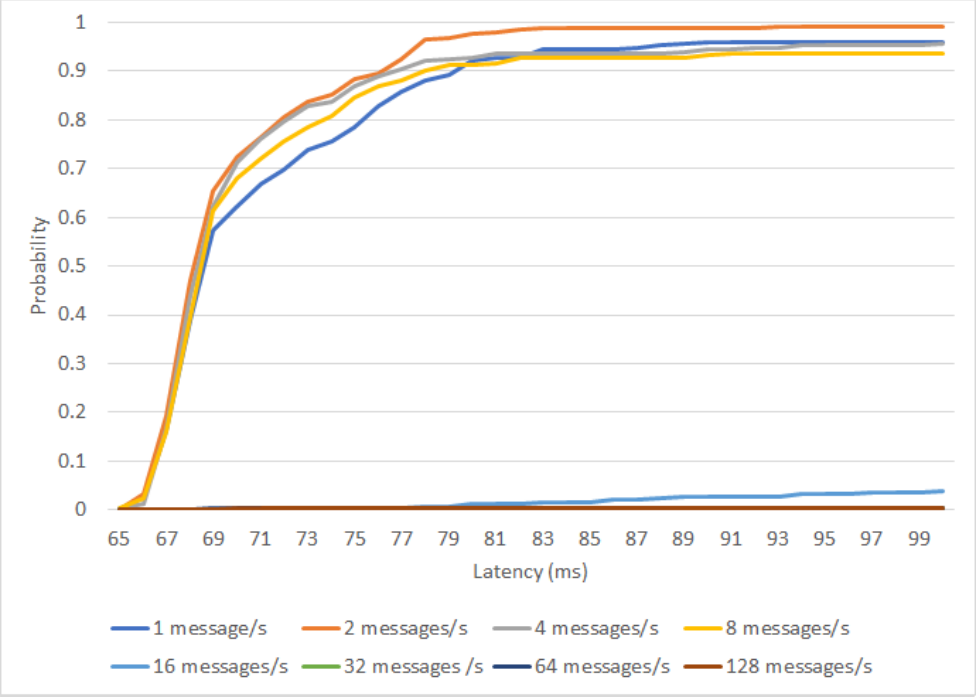

Figure 4.19: Cumulative distribution function of message latency for different message rates - MQTT through Azure IoT Edge device - cropped for up to 100 milliseconds for better visualization.

messages per second, these messages are queued and take additional time to be received, processed and for a response to be returned. There were some exception, especially the 16 messages per second test in the scenario of section 4.6.1, covering "AMQP protocol directly to Azure IoT Hub". It is believed that these exception are due to fluctuations in either the network or the Azure IoT Hub service or the Azure IoT Edge service. To prevent this issue, more tests should have been run to mitigate these variances.

- A large part of the messages were having latency of approximately 66 to 68 milliseconds, especially for low message rates. This is believed to be due to the round trip time (RTT) of the communication with the Azure IoT Hub plus some processing time factor. The round trip time is the time it takes for a network packet to reach a destination and return to the sender. The time the messages in the tests will take to be sent to and received from the Azure IoT Hub cannot be smaller than the round trip time to the server. As mentioned in [FJ17], the Azure IoT Hub server has protection against `ping` mechanisms, which are the most commonly methods to attempt to measure round-trip time to a server. The protection is typically around either the port used by the protocol that `ping` uses which is the Internet Control Message Protocol (ICMP) or the ICMP itself. However, to measure the round-trip time, it's possible to use a program that will instead use the Transmission Control Protocol (TCP) or some other protocol that bypasses these protections, and run it against a port that is known to be open on the target server. Azure IoT Hub relies on using either AMQP, MQTT and HTTPS, so the ports associated to these protocols, which are 5671, 8883 and 433, respectively, will be normally open. In this case, a Windows machine was used, so a program called `psping`[Mic19j] was used. The round trip time to the server that hosted the Azure IoT Hub used in the tests was measure at an average of 45.52 milliseconds. So there is likely a 21 to 23 millisecond overhead in the Azure IoT Hub that causes this time to increase to a 66 to 68 millisecond range for latency.

- The MQTT test when connecting directly to the Azure IoT Hub had the problem of higher message rates creating an ever increasing latency as more messages are sent. This issue makes it so that it's not a viable option to use this scenario for this message rate, as messages will take longer and longer to get a response and mostly likely eventually result in dropped messages, messages that aren't stored or processed. The problem is then amplified when the messages go through the Azure IoT Edge device. This could either be because of a weakness in the MQTT protocol or because the Azure IoT Edge device doesn't have powerful enough hardware to dispatch messages fast enough. Also, the inability to run the test that used the AMQP protocol through the Azure IoT Edge device at higher message rates made it so it was not clear if the blame could be more towards the MQTT protocol. If the AMQP through Azure IoT Edge device test could have been run at higher message rates, and showed good stability, it could show AMQP would be a protocol that would give good results.

Comparing to results from [FJ17], the latency values seem to be overall better in the case of the work done, but the cited work used OPC-UA and dealed with much higher throughput values

and could have significantly different network conditions. For [DPW18], their results for using Azure Edge seem to be better, but their test is different and the network conditions are most likely also different. However, they seem to have used a lower power device as the edge device.

Chapter 5

Conclusion

The main focus of this research was meant to be on Azure IoT Edge. However, this tool depends on Azure IoT Hub, and IoT Hub ended up being a core component of this investigation and much more interesting and not enough time was invested in IoT Edge itself, despite some work still being done on it.

5.1 Objectives reached

The goals set for this research were:

1. describe the system (hardware and software) and Azure IoT Edge licensing/subscription requirements as well as the Azure IoT Edge pricing model.

2. capture and describe the Azure IoT Edge capabilities.

3. perform a proof-of-concept of converting one existing Connect IoT driver and Controller as an IoT Edge module.

4. Extend the proof-of-concept by adding a simple example for telemetry and machine learning module.

The goal of describing system requirements, licensing requirements and pricing model of Azure IoT Edge was completed successfully. This is well documented by Microsoft and carried over to this document.

Capturing and describing the Azure IoT Edge capabilities was partly done. The core functionalities of IoT Edge were successfully captured and described, but there are a lot of different other features with a lot of different combinations and nuance. All the different potential functionalities a system running IoT Edge can have can only be captured by either having uses cases for all the combinations or by running a much more thorough and long-running investigation.

A proof-of-concept prototype was successfully created for Critical Manufacturing, using their Connect IoT driver and controller models. However these were not developed as IoT Edge modules as this was deemed not practical and instead developed as standalone software.

Telemetry examples were successfully developed, being able to intake data from the MES and outputting to other applications listening on the IoT Hub that was receiving the telemetry messages. Machine learning was touched upon and able to work with it but not successfully integrated as an example for Critical Manufacturing's MES work.

Some conclusions were taken from the data. Azure IoT Hub generating a approximately 21 to 23 millisecond overhead inside their data centers to then send the message out the receiving computers. The MQTT protocol implementation does not handle high message rates well, causing the message latency to increase rapidly as more messages are sent. However, even though the AMQP protocol implementation did not suffer this issue, it was not possible to get it working with the Azure IoT Edge device, even at 4 messages per second. And overall, latency increases as the message rate increases.

5.2 Future work

When doing the testing, only simple messages were sent with mock data about machines reporting status data. These messages were basic strings that are small in size and simpler to encode, representing small JSON objects. It should be also useful to use more complicated data sets like binary formats or much larger strings describing more complex objects.

Future research should collect more data to mitigate the fluctuation shown in the results for this project.

Although there was some work on using other Azure features such as Azure Machines Learning on Azure IoT Edge, there can be more investment in this area, even in the manufacturing execution system and industrial internet of things scenario in attempts to gather value from those tools, such as useful predictive maintenance models.

Azure IoT Edge and IoT Hub are very extensive tools with a lot of details to investigate. Attempting to refine what use cases most benefit from these tools under what conditions is an important step to take to also better evaluate these tools in the future.

Not enough time was dedicated to Azure IoT Edge as Azure IoT Hub was a requirement to understand before dealing with IoT Edge in depth, so some future work could dedicate more time for IoT Edge, having in consideration the time to understand Azure IoT Hub.

Besides all of these points, the impression was left that IoT Edge is still evolving and not quite ready for the most demanding tasks. Despite the appeal of the potential that tool has, as of right now, it has shown some odd results, which could diminish its appeal. Therefore, this tool should be revised for research at a later release version to determine if there were improvements.

References

[AFGM+15] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *Ieee Communications Surveys and Tutorials*, 17(4):2347–2376, 2015.

[AWS19] Inc. or its affiliates Amazon Web Services. Amazon web services (aws). https://aws.amazon.com, June 2019. Accessed: 2019-06-19.

[BRSF18] D. Bezerra, R. Roque Aschoff, G. Szabo, and D. Fawzi Hadj Sadok. An iot protocol evaluation in a smart factory environment. In *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, pages 118–123, Nov 2018.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[CZ16] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[DB16] A. V. Dastjerdi and R. Buyya. Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116, Aug 2016.

[DPW18] A. Das, S. Patterson, and M. Wittie. Edgebench: Benchmarking edge computing platforms. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 175–180, Dec 2018.

[FJ17] S. Forsström and U. Jennehag. A performance and cost evaluation of combining opc-ua and microsoft azure iot hub into an industrial internet-of-things system. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, June 2017.

[Git19] Inc. GitHub. Github - azure-samples/azure-iot-samples-node: azure-iot-node-samples provides a set of easy-to-understand, continuously-tested samples for using azure iot hub and azure iot hub device provisioning service using node.js sdk. https://github.com/Azure-Samples/azure-iot-samples-node, June 2019. Accessed: 2019-06-21.

[Goo19] Google. Cloud computing services | google cloud. https://cloud.google.com, June 2019. Accessed: 2019-06-19.

[Han11] Yan Han. Cloud computing: Case studies and total cost of ownership. *Information Technology and Libraries*, 30(4), 2011.

[HPO16] M. Hermann, T. Pentek, and B. Otto. Design principles for industrie 4.0 scenarios. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 3928–3937, 2016.

[IBM19] IBM. Ibm cloud. https://www.ibm.com/cloud, June 2019. Accessed: 2019-06-19.

[Int97] MESA International. Mesa white paper #02: Mes functionalities and mrp to mes data flow possibilities. https://services.mesa.org/resourcelibrary/showresource/2cedfe75-daed-4b9c-b187-f421cf90fdd2, Mar 1997. Accessed: 2019-06-04.

[LBK15] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015.

[LFK+14] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industrie 4.0. *Wirtschaftsinformatik*, 56(4):261–264, 2014.

[Lu17] Yang Lu. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6:1–10, 2017.

[LXZ14] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259, 2014.

[LYZ+17] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 4(5):1125–1142, 2017.

[Mic19a] Microsoft. Directory of azure cloud services. https://azure.microsoft.com/en-us/services/, June 2019. Accessed: 2019-06-04.

[Mic19b] Microsoft. Enterprise integration with azure logic apps. https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview, June 2019. Accessed: 2019-06-07.

[Mic19c] Microsoft. Gateways for downstream devices. https://docs.microsoft.com/en-us/azure/iot-edge/iot-edge-as-gateway, June 2019. Accessed: 2019-06-17.

[Mic19d] Microsoft. Get to know azure. https://azure.microsoft.com/en-us/overview/, June 2019. Accessed: 2019-06-04.

[Mic19e] Microsoft. Introduction to azure iot hub. https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub, June 2019. Accessed: 2019-06-06.

[Mic19f] Microsoft. Learn how the runtime manages devices - azure iot edge. https://docs.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime, June 2019. Accessed: 2019-06-11.

[Mic19g] Microsoft. Microsoft azure cloud computing platform & services. https://azure.microsoft.com, June 2019. Accessed: 2019-06-19.

[Mic19h] Microsoft. Overview of azure stream analytics. https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction, June 2019. Accessed: 2019-06-07.

[Mic19i] Microsoft. Pricing—iot hub | microsoft azure. https://azure.microsoft.com/en-us/pricing/details/iot-hub/, June 2019. Accessed: 2019-06-22.

[Mic19j] Microsoft. Psping - windows sysinternals | microsoft docs. https://docs.microsoft.com/en-us/sysinternals/downloads/psping, June 2019. Accessed: 2019-06-22.

[Mic19k] Microsoft. Supported operating systems, container engines - azure iot edge. https://docs.microsoft.com/en-us/azure/iot-edge/support, June 2019. Accessed: 2019-06-17.

[Mic19l] Microsoft. What is - azure machine learning service. https://docs.microsoft.com/en-us/azure/machine-learning/service/overview-what-is-azure-ml, June 2019. Accessed: 2019-06-07.

[npm19] Inc. npm. npm | build amazing things. https://www.npmjs.com/, June 2019. Accessed: 2019-06-22.

[RDCN12] E. Roloff, M. Diener, A. Carissimi, and P. O. A. Navaux. High performance computing in the cloud: Deployment, performance and cost efficiency. In *CloudCom 2012 - Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, pages 371–378, 2012.

[SCZ+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[SdUAP09] B. Saenz de Ugarte, A. Artiba, and R. Pellerin. Manufacturing execution system – a literature review. *Production Planning & Control*, 20(6):525–539, 2009.

[SGH15] Nicolas Serrano, Gorka Gallardo, and Josune Hernantes. Infrastructure as a service and cloud technologies. *IEEE Software*, 32(2):30–36, 2015.

[Xu12] Xun Xu. From cloud computing to cloud manufacturing. *Robotics and Computer-Integrated Manufacturing*, 28(1):75–86, 2012.

[YLL15] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, Mobidata '15, pages 37–42, New York, NY, USA, 2015. ACM.

[YZGH12] Peng Yue, Hongxiu Zhou, Jianya Gong, and Lei Hu. Geoprocessing in cloud computing platforms – a comparative analysis. *International Journal of Digital Earth*, 6(4):404–425, 2012.