CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Edge technologies in the automotive industry

An experimental latency evaluation of AWS Greengrass usage

Master's thesis in Computer science and engineering

Erik Gunnarsson
Alfred Kjeller

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Edge technologies in the automotive industry

An experimental latency evaluation of AWS Greengrass usage

Erik Gunnarsson & Alfred Kjeller

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Edge technologies in the automotive industry
An experimental latency evaluation of AWS Greengrass usage
Erik Gunnarsson & Alfred Kjeller

Supervisor: Philipp Leitner, Department of Computer Science and Engineering
Advisor: Tomas Carlfalk, WirelessCar
Examiner: Lucas Gren, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Edge technologies in the automotive industry
An experimental latency evaluation of AWS Greengrass usage
Erik Gunnarsson & Alfred Kjeller
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The digital transformation of the automotive industry is one of the most significant changes to the industry in its 140-year history. This transformation bring new functionalities to vehicles and changes how a vehicle is being used. In conjunction with the automotive industry's digital transformation there has been a trend to move all computational power to the cloud. The cloud offers an efficient way to deal with data, but the network has started to become a bottleneck. The purpose of the study is to evaluate how the usage of edge technology can reduce latency and bandwidth usage in the automotive industry.

The study is based on a controlled experiment where a prototype system was developed to evaluate different implementations of edge technologies. The system consists of an accelerometer connected to a car, and a machine learning algorithm that uses the data from the accelerometer to try to predict the type of road surface the car is driving on. Three concrete experimental setups was evaluated: One system where all processing is done in the cloud. One system where the pre-processing is done in an edge computational layer inside the vehicle instead of the cloud. And finally, one system where all processing is performed in an onboard edge computational layer. When testing the different implementations different bandwidth limitations were enforced as well.

The study showed that moving all computational power to an edge node directly in the car reduces latency compared to other implementations, no matter the bandwidth limitation. The study also showed that with high bandwidth available, a system only running in the cloud could be faster than a system that uses both edge and cloud computing. However, when having a limit on the available bandwidth, a system that uses both edge and cloud computing is faster than a system only running in the cloud.

Keywords: Edge computing, Cloud, Latency, Bandwidth, Automotive, Greengrass, AWS

Acknowledgements

We would like to thank our supervisor, Philipp Leitner for his contributions to this thesis. We would also like to thank Tomas Carlfalk, Jeffrey Spång, Jens Andersson and Anna Gunlycke at Wirelesscar for their help during the project. Finally we would like to thank Linn Alholt at New Minds for helping us get in touch with WirelessCar.

Erik Gunnarsson, Alfred Kjeller, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1

Introduction

The digital transformation of the automotive industry is one of the most significant changes to the industry in its 140-year history e.g., digital technologies stand for more than 50% of a modern vehicle's value [4]. The transformation brings new functionalities to vehicles and changes how a vehicle is being used. In conjunction with the automotive industry's digital transformation there has been a trend to move all computational power to the cloud. The cloud offers an efficient way to deal with data, but the network has started to become a bottleneck. This bottleneck is due to bandwidth limitations and the need for lower network latency. Thus, becoming an increasingly larger problem as modern vehicles generate a lot of data that gets processed in the cloud [5].

The data gathered by modern vehicles usually needs some form of processing, examples include creating range estimations based on battery discharge rates in an electric vehicle or using machine learning to analyze data from various kinds of sensors, such as video feeds. By doing the processing in the cloud, two problems occur. Firstly, processing the data in another location could suffer from latency and bandwidth limitations. Secondly, each network package sent between the cloud and the vehicle comes with a monetary cost.

Edge computing could be used to offload processing vehicles do in the cloud or move the processing directly to the vehicle. For latency-sensitive processing such as using machine learning to analyze various kinds of data, e.g., video feeds or sensor outputs. A deployment closer to the data source has the potential to improve performance by decreasing the time it takes for the vehicle to analyze the given data. Edge computing also has the potential to improve performance for systems running in vehicles with bad cellular receptions since the amount of data that needs to be transferred could be reduced with edge computing.

In this thesis, an experiment is conducted for the following edge computing use case. A system which consists of an accelerometer connected to a car, and a machine learning algorithm (in this case, a Random Forest classifier) that uses the data from the accelerometer to try to predict the type of road surface the car is driving on. As stated before, this is a type of system that could benefit from edge computing. The architecture of our system is as follows: Firstly, the accelerometer that is mounted in a car collects acceleration data. Secondly, a pre-processor that takes the data from

the accelerometer and calculates statistical metrics. And lastly, a machine learning classifier that uses the calculated statistical metrics to classify which type of road surface the car is driving on. Three concrete experimental setups are evaluated: One system where all processing is done in the cloud, like the current state of the art. One system where the pre-processing is done in an edge computational layer inside the vehicle instead of the cloud. And finally, one system where all processing is performed in an onboard edge computational layer.

This thesis is conducted in collaboration with WirelessCar as an industrial partner. WirelessCar, a subsidiary of Volkswagen AG and Volvo AB, are creating digital services for the automotive industry. The purpose of this study has been developed together with WirelessCar and the result of the study aims to give WirelessCar further insights regarding cloud and edge computing in the automotive industry. The study also aims to contribute to global research in this field.

1.1 Purpose

The purpose of the study is to evaluate how the usage of edge technology can reduce latency and bandwidth used for the automotive industry. As machine learning becomes more prevalent in the automotive industry our evaluation will be based on a comparison between a baseline and two implemented systems. All systems send data to a machine learning classifier trying to predict which kind of road the car is currently driving on. Figure 1.1 depicts a high-level overview of the three systems, the first one being completely connected to the cloud and not processing data on the vehicle at all. Our second system pre-processes the given data on a local edge computing layer to send less data to the cloud, this is to mitigate the bandwidth bottleneck in the industry today. Our final system uses an edge computing middle layer to both pre-process the data as well as classifying it using a machine learning model. These three systems are described in more detail in the methodology chapter of this report.

By providing an empirical comparison this thesis intends to benefit both researchers and practitioners within the automotive industry by giving insight into how the latency between a device and the cloud behaves in different systems.

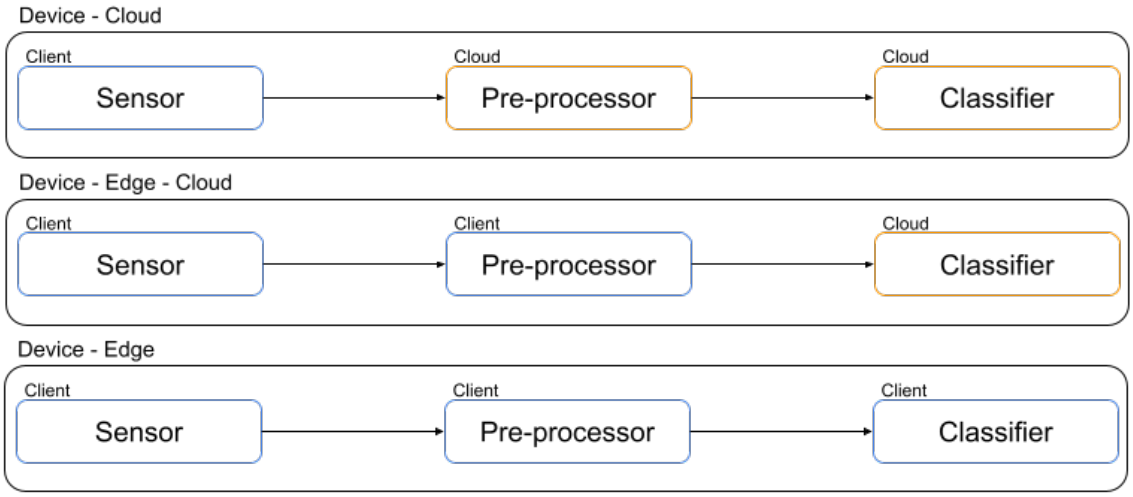

Figure 1.1: Overview of the baseline and the two systems using edge technologies.

1.2 Research Questions

Based on the introduction the following research questions have been chosen to fulfill the purpose of the study. That is, to evaluate how the use of edge computing can reduce latency and bandwidth within the automotive industry.

1. By using edge technologies to pre-process data before a sensor sends it to the cloud for classification, how much can latency be improved when operating at different network speeds?

2. By using edge technologies to both pre-process and classify data instead of using cloud technologies, how much can latency be improved when operating at different network speeds?

2

Background

This chapter aims to give the reader relevant background information regarding important topics for this thesis. The topics explained are the following: cloud computing, machine learning, edge computing, the MQTT protocol and function-as-a-service (FaaS).

2.1 Cloud computing

During the last decade the use of cloud computing has risen tremendously. Castro et al. [6] claims that cloud computing would account for 67% of enterprise IT infrastructure spending by the year 2020. At the end of 2021 Amazon Web Services (AWS) was the market leader in cloud infrastructure with a market share of 33% [7]. Cloud computing could be regarded as a distributed system, able to offer computing services via the internet. Dillon et al. [8] describes the paradigm as "Cloud computing is a model for enabling 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". Thus, cloud resources are transparent to users, who do not need to know the resources' physical location. Figure 2.1 shows an example of a cloud computing architecture.

Users can access applications and data hosted in this kind of system at any time whilst it is being shared between a large number of users. Sadiku et al. [9] describes the benefits with the paradigm, stating that if properly used the location-independent resource pooling and on-demand self-service may have great opportunities for businesses of all sizes.

There are also several challenges with the cloud, specifically in privacy and security. In some contexts, the latency from request to delivery can also be an issue [9]. Shi and Dustdar [5] also identifies a challenge with cloud computing, which is especially prevalent in the automotive sector, in that network bandwidth does not increase as fast as the number of requests sent to the cloud. With the increasing number of requests comes a larger strain on the network bandwidth, which in turn leads to higher latency for the entire system.

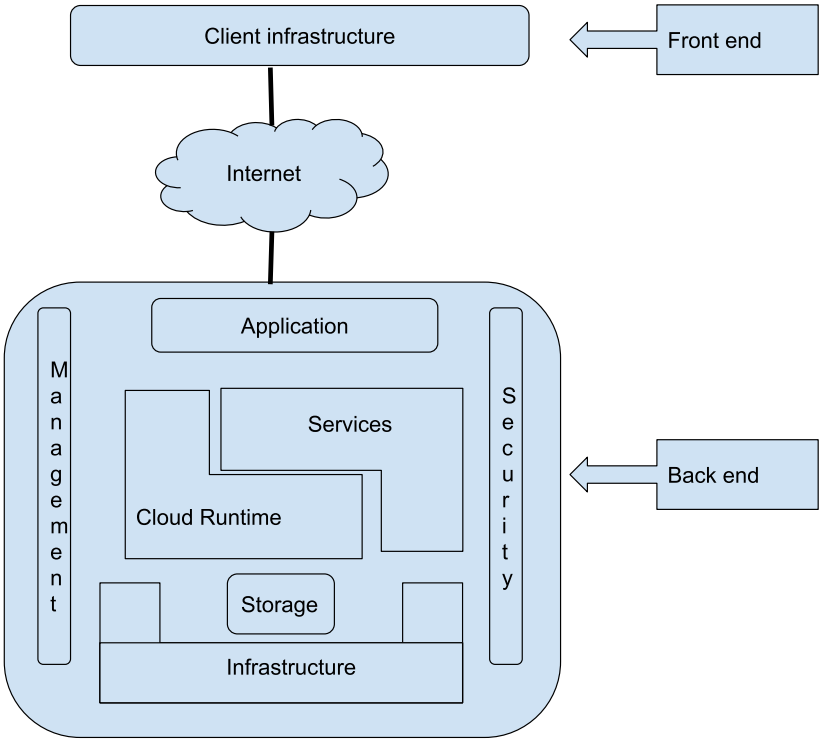

Figure 2.1: Overview of cloud computing architecture. Inspired by: Bhale et al. [1]

2.1.1 Serverless computing

Castro et al. [6] defines serverless computing as "Serverless computing is a platform that hides server usage from developers and runs code on-demand automatically scaled and billed only for the time the code is running". The use of this kind of system further improves upon the scalability of cloud computing, which is, according to Marston et al. [10], one of the major reasons that cloud computing has become so successful.

Running your code on a pay-as-you-go model removes the cost of having idle servers for systems that are prone to sudden bursts of extra workload. Developers using serverless computing also do not have to write their own scaling policies, as well as no longer needing to care for maintenance, security updates, and availability of servers. This makes serverless computing beneficial for applications in which an extra-large workload can come in bursts [6]. Alongside this, developers can often get the cost savings and scalability that traditional cloud computing offers, without the high level of cloud expertise usually required for getting the most out of traditional cloud computing.

While only paying for a server when it is actually used is great, it also has some inherent challenges, such as cold starts [11]. These cold starts appear as the virtual

servers get shut down after not being used for a period of time, thus slowing down execution time when the servers themselves need to start up before executing any code. Developers have tried to circumvent this issue by repeatedly pinging the cloud to ensure the virtual servers are not shut down, this however goes against the principle of on-demand scaling.

2.2 Function-as-a-Service

One of the more important services within serverless computing comes in the form of Function as a Service (FaaS). FaaS is an approach to building event-driven software [12] by hosting a function on the cloud and triggering it from a specific event. What kind of event is used to trigger the function can vary by a lot, for example it could be a file uploaded to a database or an API request.

FaaS-functions are usually ran in a deeply virtualized manner, often in containers which could depend on other serverless systems themselves. This makes them extra prone to cold starts of half a second or more as each serverless sub-system in their architecture could require its own cold start [13].

2.2.1 AWS Lambda

Our study uses AWS lambda as the FaaS platform of choice for performing our experiment. Bertilsson and Grönqvist [14] found that AWS had a latency of only two thirds that of their largest competitor, Microsoft Azure's FaaS platform. Alongside this, AWS Lambda also comes with a significantly lower standard deviation when looking at function trigger times compared to Microsoft Azure [14]. Our industrial partner, WirelessCar, also uses AWS as one of their cloud providers.

AWS Lambda is Amazon Web Services FaaS platform, and cited FaaS trigger events include image uploads, in-app activities, website clicks, outputs from connected devices, or other custom requests [15]. AWS Lambda currently supports functions written in Java, Go, PowerShell, Node.js, C#, Python, and Ruby and can scale from one to tens of thousands of concurrently running function instances. When integrating FaaS triggers with AWS Lambda you don't need to write any code, allowing developers to focus on their applications logic [12]. AWS Lambda is also designed to be integrated into Amazon Web Services Edge computing and IoT platform, AWS Greengrass.

2.3 Edge computing

Edge computing refers to a distributed computing paradigm in which processing, and data storage are brought closer to the source of the data [16]. To enable edge computing, software is run on devices called edge devices. Shi and Dustdar [5] describes an edge device as a computing resource that operates between the cloud and at least one data source. This could be a smartphone handling data from a body sensor, or a server that is run on the mobile network that operates closer to the user than a data center.

Multiple research projects have demonstrated that programmers must carefully tailor their edge solutions to work properly between the client and the cloud. This has mostly been done manually thus requiring much work from the developers. Therefore, easy-to-use frameworks are needed for greater adoption of edge computing [5]. AWS Greengrass is a framework that might solve this issue and is explained in the next section.

2.3.1 AWS Greengrass

AWS Greengrass is Amazon Web Services edge computing platform. When using Amazons AWS Greengrass, a device can be configured to run the AWS Greengrass Core software, this turns the device into what is called a Greengrass core device. Multiple components can then be deployed to the Greengrass core device. These components can offer a wide array of functionality, everything from simple logging tools to configurations for SSL certification, but they can also contain AWS Lambda functions that can be run locally on the device. This allows for the same code configured in the same way to be run both in the cloud and on a local Greengrass device, essentially making the Greengrass core device into a local implementation of an AWS cloud server. Alongside just running the functions, the Greengrass core software also allows for easy deployment of modifications to a large group of Greengrass devices simultaneously. Greengrass also offers native support for AWS MQTT broker, which lets any Greengrass core device seamlessly connect its surroundings with the AWS IoT hub via MQTT messages. Finally, Greengrass devices can run long-lived functions, which can remove the risk of a "cold start" [17]. An example of how the AWS Greengrass software communicates with AWS central servers can be seen in Figure 2.2.

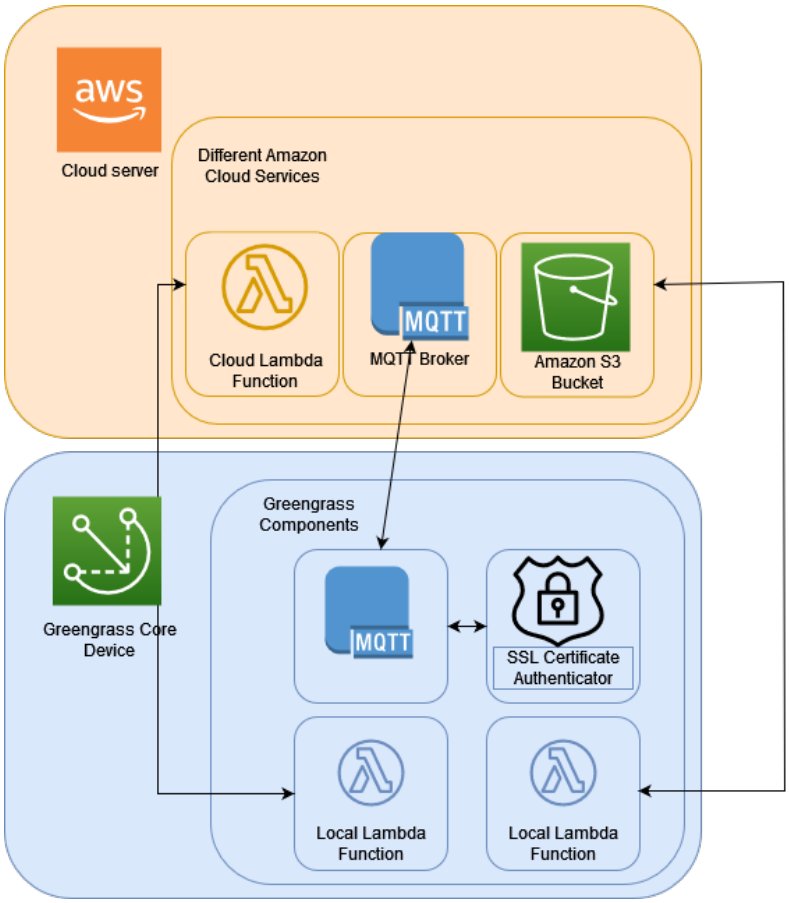

Figure 2.2: Overview of AWS Greengrass connection with Amazon Web Services.

2.4 MQTT

The Message Queue Telemetry Transport protocol (MQTT) is a light-weight messaging protocol designed with Internet of Things (IoT) and Machine-to-Machine (M2M) communication in mind. The protocol makes use of a client-server architecture using topics to allow for communication between actors. The protocol is standardized by the Organization for the Advancement of Structured Information Standards (OASIS). Clients can subscribe to a topic after connecting to a MQTT broker, once subscribed the broker forwards any messages on that same topic to the clients who have subscribed. Similarly, a client can publish a message to a topic using the same broker. This pipeline is shown in Figure 2.3. The protocol is most commonly run over TCP/IP and is frequently used within the automotive industry. MQTT offers three varying degrees of quality of service (QoS), allowing developers to define if messages should be delivered "At most once", "At least once" or "Exactly once" depending on their context. As a higher quality of service requires the broker to publish more messages back to the publisher, a higher quality of service can lead to a higher cost in terms of data transfer.

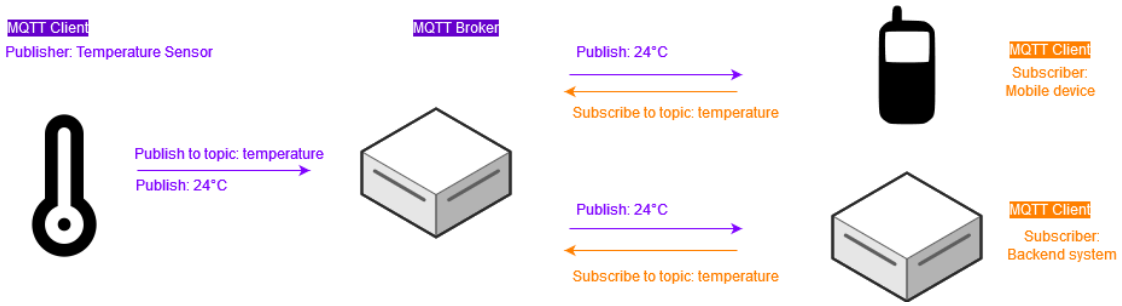

Figure 2.3: Overview of the MQTT protocol. Inspired by: MQT [2]

3

Related Work

This chapter aims to relate our work with already conducted research. The chapter is divided into two parts, one handling latency related papers, and one for bandwidth related ones.

3.1 Edge computing experiments relating to latency reduction

Baresi et al. [18] performed a study where they compared latency and throughput of a cloud solution with an edge solution. They used AWS for the cloud solution and developed a Mobile Edge Computing (MEC) system for the edge solution. MEC is a system where computational resources are moved to base stations of the cellular network. The idea is that latency can be improved by moving the servers closer to the clients, however the processing power is usually more limited and therefore the throughput tends to decrease. The result of their study was that the cloud solution had 80% more latency than the MEC system. The throughput was similar between the systems, but the authors argue that for more requests per time unit the limited computational power of the MEC would be a bottleneck for the system, whereas the cloud solution provides almost unlimited resources. Baresi et al. [18] also test their system where the edge solution is deployed to a local client. This solution performs poorly for larger amount of request due to overload. The system that uses a local client is built similar to the systems tested in this study. The key difference is the number of requests, they send more requests than a single client could produce. The systems tested in this study are tailored for a single client and therefore more accurate data for a local edge solution is provided.

Pelle et al. [19] developed a prototype system that utilized both AWS Lambda and AWS Greengrass. The purpose of the prototype was to showcase that latency could be controlled by offloading processing to AWS Lambda if the edge node was congested. The edge node was primarily used to pre-process the data to reduce bandwidth used. However, Pelle et al. [19] do not present results of extensive testing, but rather just show a system as a proof of concept. Our study aims to provide data gathered from benchmarking an edge solution.

Skirelis and Navakauskas [20] compared the network delay, service time and processing time between a device-cloud solution and a solution using an edge middle-layer in the video streaming domain. Their simulation found a network delay reduction of more than 70% when using the edge-layer, however they also found that the hardware of the edge-layer bottlenecks the system for a large number of users. Ultimately, they argue that cloud-based solutions without edge devices result in lower overall service times for domains requiring a large number of concurrent users. Placing resource limited nodes close to the edge can also drastically reduce network delay and thus also service time.

According to experiments conducted by Liu et al. [3] the Round-Trip-Time (RTT) for simply sending data to the cloud and back is barely affected by a wireless edge layer at message payloads of less than 10kb. By using a device - edge solution and disconnecting the system from the cloud the system experiences significant latency reduction for all payload sizes. Depending on what kind of artificial intelligence model is being utilized for making predictions with the given data the time difference by doing the prediction in the cloud or on a small raspberry pie is negligible. They found that a regression model takes less than 1ms on both systems, although there is as much as a 300ms when comparing predictions done by a long short-term memory-encoder decoder. These findings suggest that if the prediction can be done accurately by a simpler model there can be significant latency improvements by making the prediction on an edge device attached to the car rather than sending the data to the cloud for prediction.

3.2 Edge computing experiments relating to bandwidth strain

Liu et al. [3] also experimented with edge computing relating to different network speeds. They found that the added edge layer did not suffer from significantly increased latency for internet speeds above 32kbps. They also found that their edge system using the MQTT messaging protocol did provide optimal performance for payloads of 10kb or less. When comparing different message payloads for their device - cloud system with a system containing an edge layer they had similar results. I.e., the systems get strained and suffer from significantly increased at payloads of 10kb or more. At their maximum payload of 250kb the edge middle layer performed the worst of the systems, experiencing approximately twice the performance loss as their other systems, as seen in Table 3.1.

Shi et al. [21] discusses how edge computing affects bandwidth usage. They conclude that bandwidth usage is lower when using edge computation because pre-processing could lower the amount of data that gets sent to the cloud. This is specifically useful in a scenario where the client device might suffer from bad connection. For example, a smartphone connected to a private WiFi has access to a good connection and is not bandwidth constrained. However, the same smartphone connected to a cellular network could suffer connection problems. Shi et al. [21] also discuss how bandwidth

Table 3.1: Perfomance of systems at high payload compared to 10kb. Source: Liu et al. [3]

System	Performance at 250kb payload
Device - Cloud	59.88%
Device -Edge -Cloud	23.47%
Device - Edge (TCP/IP)	51.02%
Device -Edge (MQTT)	44.64%

used relates to overall latency. Our study aims to test how bandwidth is affected by introducing edge computation for a cloud system, where pre-processing is included. Our study also provide results regarding how latency is affected by the amount of bandwidth that is used.

4

Methodology

In order to evaluate how edge technologies and bandwidth may affect latency a controlled experiment is conducted where three separate systems using various degrees of edge technologies are benchmarked to be compared to each other. This section describes the experiment setup as well as choices made during experiment design.

4.1 Scientific method

Our study intends to learn more about the relationship between the round-trip time (RTT) latency from our sensor and the cloud and message payload. To achieve this goal a controlled experiment was conducted. A controlled experiment allows us to measure the variables more accurately since there is less risk for random events to impact the result [22]. The study began with implementing the different systems and designing the experiment. The implementation of the systems is described in Section 4.2 and the experiment design in Section 4.3. This resulted in three implemented systems which was later benchmarked to answer the research questions. Section 4.4 explains the execution of the controlled experiment. Firstly, sensor data was collected which was used to train our machine learning classifier. Then the benchmarks were run to gather results data. Lastly, the results data were statistically analyzed.

A flowchart detailing the experiment design and execution procedure is shown in Figure 4.1.

4.2 Execution environment

The three developed systems are described in this section. Alongside this, software configurations are detailed to allow our experiment to be reproduced.

4.2.1 System architecture

In this section the architecture of our three developed systems is described.

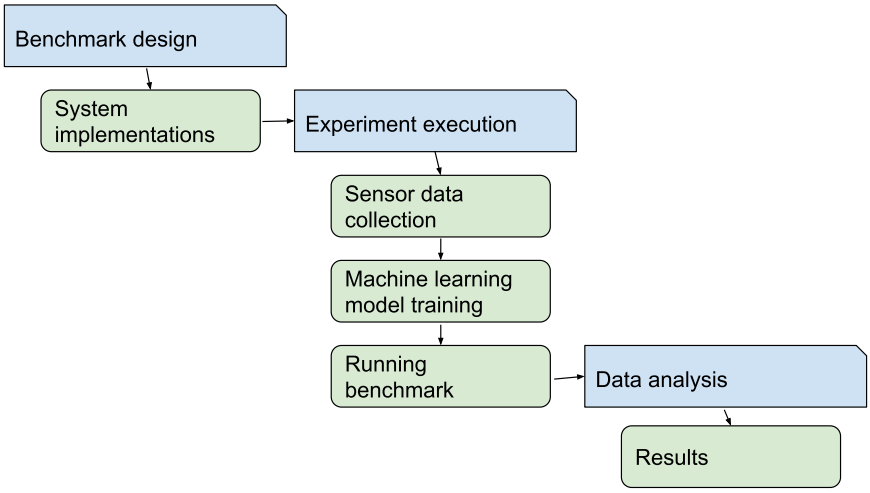

Figure 4.1: Flowchart illustrating the high-level phases of the empirical study

4.2.1.1 Device - Cloud

As seen in Figure 4.2 our baseline system is built using an accelerometer connected to a Raspberry Pi which collects data from the accelerometer, and sends batches of consecutive data points via a REST API which invokes a lambda function in AWS Lambda for pre-processing the data. The Lambda function then invokes a second Lambda function. Upon start-up of the second lambda function a machine learning model is fetched from an AWS S3 bucket. As the model is fetched on start-up it is not necessary to fetch the model again between executions, unless the lambda instance is terminated. The second lambda function then uses the pre-processed data as the input for our machine learning model. The second Lambda function then returns the prediction that the random forest classifier made to the pre-processor function which sends the prediction as a response to the REST API-request sent by the Raspberry Pi.

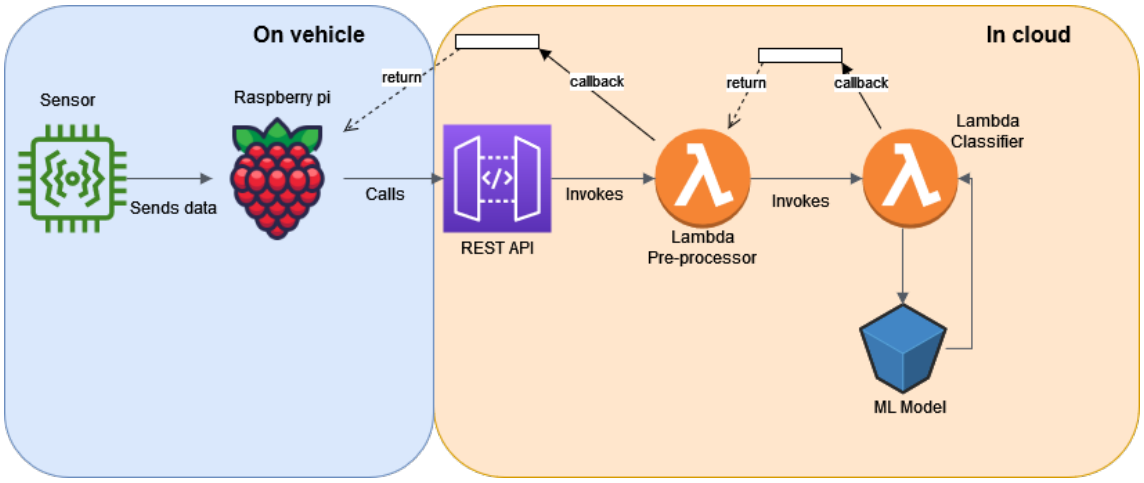

Figure 4.2: Overview of Device - Cloud.

4.2.1.2 Device - Edge - Cloud

The second system can be seen in Figure 4.3 and is still using the same setup of a sensor connected to a Raspberry Pi. In this system the Raspberry Pi acts as an AWS Greengrass core device which lets it run lambda functions locally rather than in the cloud. The Raspberry Pi uses the paho MQTT broker to publish data to be pre-processed to a topic that the devices Greengrass MQTT broker is subscribed to. The pre-processing lambda function is thus invoked by the message being published to the topic. After pre-processing the data the first lambda function invokes the classifier in the cloud just as in Device - Cloud. The classifier loads the model on start-up and predicts the road-type in the same way as the Device - Cloud system does. After the prediction has been returned to the pre-processor the result is published to a different MQTT topic which the Raspberry Pi is subscribed to in order to get the result.

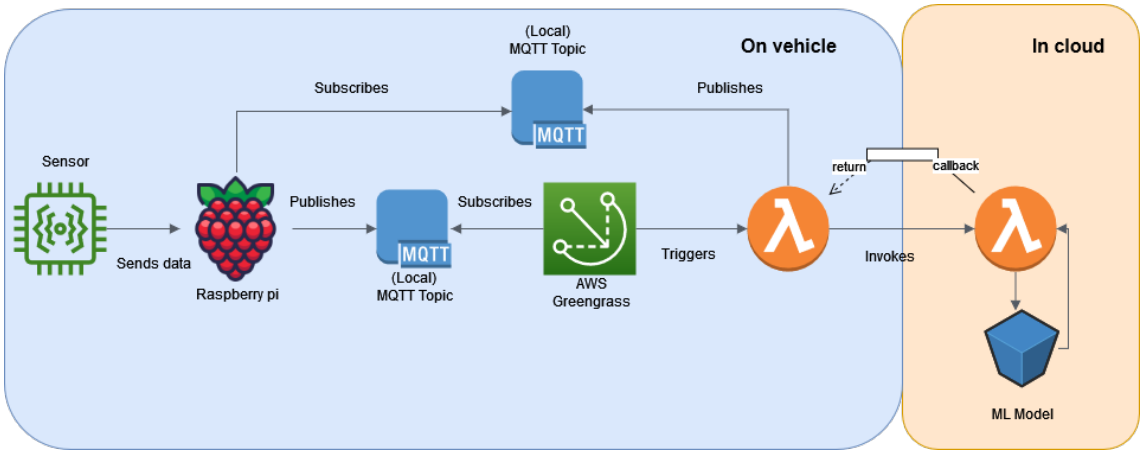

Figure 4.3: Overview of Device - Edge - Cloud.

4.2.1.3 Device - Edge

The last system is shown in Figure 4.4. This system does not only pre-process the data on the Raspberry Pi but also hosts the classifier on the device, in doing so the system is disconnected from the cloud entirely in its prediction. Similarly to Device - Edge - Cloud this system act as an AWS Greengrass core device and uses paho MQTT as a broker to connect to the Greengrass MQTT broker. After pre-processing the data in the first lambda function, the data is being published to a topic that our local classifier lambda function is subscribed to. In this system the machine learning model is also downloaded on startup from our S3 bucket so that the system does not need to connect to the cloud for each prediction. After the prediction has been done the classifier lambda function publishes the prediction directly to the topic that our Raspberry Pi is subscribed to, thus skipping a layer of callbacks in the process.

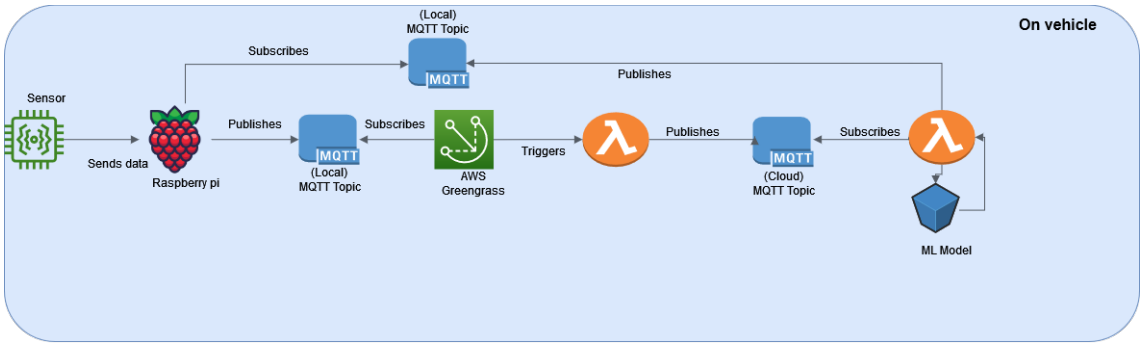

Figure 4.4: Overview of Device - Edge.

4.2.2 Software configurations

Greengrass IoT
For the greengrass IoT configuration a Raspberry Pi acts as both a Greengrass core device as well as a MQTT broker endpoint to allow the Greengrass MQTT broker to communicate with the devices local MQTT broker. There are ten Greengrass components deployed onto the core device, three of which are responsible for the communication with AWS lambda services, two of which are required to authorize the use of AWS SSL certificates for use over MQTT, another two for communication between the Greengrass local MQTT broker and the MQTT broker endpoint configured in AWS, one for simplistic logging using the AWS greengrass CLI and finally one for the pre-processing lambda function and one for the machine learning classifier.

Python
As AWS greengrass v2 is not compatible with a python version after 3.8 all code has been developed for python 3.8.

Paho MQTT
The client application uses Paho MQTT in order to subscribe and publish messages to a local MQTT broker. The client is configured with an SSL certificate provided by AWS and the clients QoS is set to 1 as we need to guarantee a message is delivered at least once.

AWS
All AWS services are hosted on AWS Frankfurt servers as it is the closest AWS servers to our location in Gothenburg where greengrass is available.

4.2.3 Client application

The client application is written in python and consists of a serial_reader and a connection class. The serial_reader reads values from the connected accelerometer and saves them as a .csv file. This allows the client to send information to the lambda functions in real time as well as saving a specific dataset to use later on to ensure retestability. The connection class handles connection to the AWS lambda functions and allows for using both real time data and a pre-saved .csv file. The data can both be sent to the cloud using a POST call to a REST API gateway as well as being transmitted via a paho MQTT broker. This enables the client to communicate with the devices' AWS greengrass IoT core independent of which of the three systems are being used. Upon a callback from the cloud or a message being published to the correct MQTT topic the connection writes the measured response time(s) to a .csv file for later analysis.

4.2.4 AWS Lambda functions

Function 1
Pre-processes the data by extracting mean, median, standard deviation and the variance from the given data batch. This information then gets forwarded to the machine learning model in **Function 2** in order to be used in the prediction.

Function 2
The model is a pre-built RandomForestClassifier and trained on 488 batches of data. The model is stored in an AWS S3 bucket and is pre-loaded on startup. Pre-loading the model allows a greengrass device to use the model without connecting to the internet (except for when the device is turned on).

4.2.5 Hardware

In this section we discuss the hardware used to conduct our controlled experiment.

Raspberry Pi
We use a Raspberry Pi 3 which uses 4× ARM Cortex-A53 1.2GHz and runs on the Raspbian operating system.

Accelerometer
Our accelerometer uses the ADXL345 integrated circuit which streams data via CSV. In our experiment we set the frequency of the accelerometer to 100Hz and its range to ± 2g. The accelerometer is connected to our Raspberry Pi using a standard USB 2.0 cable.

4.2.6 Bandwidth limitation

To answer the research questions, the network bandwidth must be controlled for the benchmarks. The Greengrass Core device uses Ethernet to connect to internet and the internet connection is a 100Mbps fiber connection. This setting is not suitable for a system that represents a vehicle driving around. Vehicles rely on a cellular connection which in many cases are far slower than a 100 Mbps fiber connection. 2G is a cellular network with good coverage and since cars might drive in rural areas, 2G is a suitable network connection for the benchmarks. 3G is a faster connection but might have less coverage.

The bandwidth limitation for 2G is 64Kbps [23] and one setting of the benchmarks was therefore 64 Kbps. 2G might seem like a worst-case scenario but an important note is that a typical vehicle has more than one system that needs an internet connection and therefore needs to share the bandwidth. 3G has a far higher theoretical bandwidth but according to Qureshi et al. [24], the peak upload network speed for a vehicle using 3G is less than 140 Kbps. Therefore, 128 Kbps were also chosen as one of the bandwidth limitations. The third setting for our benchmark was by running the system without any bandwidth limitation, which is the 100 Mbps fiber connection. This is the best-case scenario for the systems. A tool called Wondershaper [25] was used to limit the bandwidth of the systems during the benchmarks.

4.3 Experiment design

In the experiment we benchmark the RTT of three edge-cloud systems (Device-Cloud, Device-Edge-Cloud and Device-Edge). The client application deployed on the edge device is written in python and every second it asks our AWS lambda function to classify which kind of road the car our sensor is connected to is driving on. A visual representation of the three systems can be seen in Figure 4.5.

In the baseline system (Device - Cloud) the data sent is simply all collected values from our sensor. This data is processed by our lambda function before being passed onto a machine learning classifier which returns its prediction to our device.

In the Device - Edge - Cloud (**RQ 1**) system the collected data is sent to a local lambda function implemented with AWS Greengrass in order to pre-process the data in the same way that the first lambda function in Device - Cloud does. After being processed, the data is sent to the same, cloud based, machine learning classifier as in the Device - Cloud system to get our prediction.

In the Device - Edge (**RQ 2**) system the collected data is pre-processed on the edge device in the same manner as in Device - Edge - Cloud, but the edge device also carries out the machine learning prediction (also implemented with the use of AWS Greengrass).

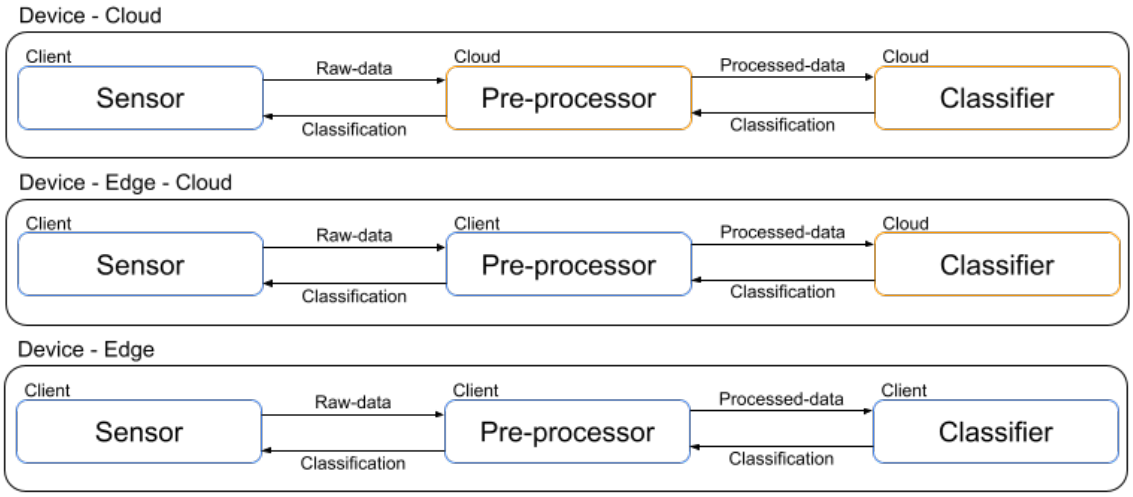

Figure 4.5: Overview of the benchmarked systems.

4.3.1 Experiment variables

Dependent variables
In our experiment the dependent variable is the RTT latency for classifying what kind of road the car is driving on. The RTT is measured by the time it takes for the system to send a batch of 100 sensor datapoints to be pre-processed and later classified by our machine learning model before returning the result to our device again.

Independent variables
The independent variable for this experiment is which system is being used (Device - Cloud, Device - Cloud - Edge or Device - Edge), this is used to answer **RQ 1** and **RQ 2**. The network bandwidth for the systems connected to the cloud is being varied between 100Mbps (fiber connection), 128kbps (3G) and 64kbps (2G). This is also used to answer **RQ 1**.

4.4 Experiment execution

The execution of the experiment consists of three steps. Firstly, data from the accelerometer needs to be collected, secondly the random forest classifier is trained on parts of the collected data, and lastly the collected data needs to be used to run the benchmark of the systems. The reason why accelerometer data needs to be collected is that the systems is not benchmarked with live data. The collected data is used to simulate a car driving to ensure we can reproduce the experiment and to ensure that each system run on the same conditions. The collecting of sensor data is explained in Section 4.4.1, the training of the random forest classifier is explained in Section 4.4.2 and how the benchmarks were run in Section 4.4.3.

4.4.1 Collecting sensor data

The built systems require collected accelerometer data to be tested. By mounting the accelerometer in a car and driving, data could be gathered. The accelerometer was fastened to the car's dashboard as seen in Figure 4.6. The mounting point provided a flat surface as well as connecting the accelerometer to the car's vibrations. A script recorded data from the accelerometer and saved it in csv files on a connected computer. Since the system's purpose is to predict the road surface which the car is driving on, data collection from different roads was required. Two long stretches of both gravel and tarmac were selected in southern Sweden. Each stretch was suitable for 5 minutes of driving. The data collection started and stopped while the car was in motion and cruise-control was used to maintain a stable velocity of 50 km/h. Although the accelerometer sends data for three dimensions, only the acceleration in the z-axis was stored because the classifier only relies on vibrations up and down for the classification. Thus, each single data input is a floating-point number representing acceleration in the up and down dimension. The accelerometer records 100 data points each second and therefore did the 10 minutes of driving resulted in 61 000 data points.

These data points are saved and stored in csv files and later used to simulate a driving car when benchmarking our systems. This is done so that network conditions can stay constant during the experiment, as conditions may vary vastly while actually driving.

Figure 4.6: How the sensor was attached to a car for data collection.

4.4.2 Training the classifier

The collected data was first split into two datasets, a training set, and a test set. The training set contained 80% of the collected data (48 800 data points) and the test

Classifier	Accuracy
Dummy Classifier	56.36%
Decision Tree Classifier	90.91%
Random Forest Classifier	92.73%
Perceptron	50.91%
Logistic Regression	89.09%
Linear SVC	81.82%
MLP Classifier	50.91%

Table 4.1: Cross-validation accuracy of different classifiers on the training set.

set contained the other 20% (12 200 data points). Secondly were multiple machine learning classifiers evaluated with cross-validation on the training set. As seen in table 4.1, Random forest classifier were the best performing classifier on the training set, and therefor was it chosen as the classifier. Thirdly were the parameters for the random forest classifier configured. Cross-validation was used to train and get an accuracy score for the classifier on the training data. Different parameters were evaluated and the default parameters of Scikit-learns [26] implementation of the random forest classifier managed to get the highest accuracy rating (92.73%), and therefore the default parameters were chosen for the model. The third step was to train a new model that utilized all training data available, this is the final model. This model was evaluated on the test set and had an accuracy of 92.86%. Further tuning might have improved the result but since this thesis is not about building the best classifier this model was chosen for the benchmarks.

4.4.3 Benchmarking the systems

After the systems were built and the sensor data had been collected the benchmark was run. The collected data consisted of 61000 data points, that were sent in batches of a hundred to the systems. The order which the systems were tested in can be seen in Table 4.2.

Order	System	Bandwidth
1	Device - Cloud	100 Mbps
2	Device - Edge - Cloud	100 Mbps
3	Device - Edge	-
4	Device - Cloud	128 Kbps
5	Device - Edge - Cloud	128 Kbps
6	Device - Cloud	64 Kbps
7	Device - Edge - Cloud	64 Kbps

Table 4.2: The order the systems were tested

The systems were rebooted after each run to ensure that all systems had the same starting conditions. Device - Edge was only benchmarked once since it only requires an internet connection on startup, and therefore is not affected by the bandwidth

limitations during the benchmark. When extracting data, we have only used values from executions that finished. This is likely to lead to a difference in sample size between the systems due to a slight difference in errored/unfinished executions.

4.5 Analysis procedure

In order to test if the differences between the systems are statistically significant, different statistical tests will be performed. R Studio will be used for the tests, by using the car and effzise library. A normality test will be performed for each population, both by performing a Shapiro-Wilk test but also by looking at QQ-plots. If the populations turn out to be normally distributed, pair-wise-t-testing will be conducted to test for statistically significant differences between each system. Should the data turn out to not be normally distributed, a Kruskal-Wallis test will be performed limitation followed by a Wilcoxon signed rank test in order to ensure statistically different distributions between the systems. These tests will compare all systems with the same bandwidth limitation with each other.

If the pair-wise-t-test show a statistically significant difference between two systems, the effect size for this difference will be calculated using Cohen's D. If the Kruskal-Wallis and Wilcoxon signed rank test show a statistically significant difference between systems Cliff's Delta will be used to calculate the effect size instead.

5

Results

In this section we provide results from our controlled experiment. Figures 5.1, 5.4 and 5.7 show boxplots for our results. In our boxplots we have removed any outliers larger than $Q_3 + 1.5 * IQR$ and smaller than $Q_1 - 1.5 * IQR$ from the plot. While Figure 5.1 shows the mean, median and standard deviation of the systems. Figures 5.2, 5.5 and 5.8 shows the cumulative distribution functions for all systems at different network connections and Figures 5.3, 5.6 and 5.9 shows histograms of all finished executions. The results from the experiment is being provided grouped by the network conditions. Finally the statistical significance of the results is provided at the end of the section.

5.1 Experiment results

In Table 5.1 we can see a slight difference between the sample sizes used to test our systems as a handful of calls did not return but instead errored during their execution of one of our functions. The difference in successful executions is small enough to most likely be due to chance. For all systems their mean latency is slightly larger than their median latency, this is to be expected when measuring delay.

Device - Cloud performed well when connected to a good network connection, performing at a mean latency of 308.3ms at 100Mbps. From our results in Table 5.1, we can see that when the network bandwidth drops down to 128Kbps there is a notable increase in mean as well as median latency by about 12%. The standard deviation is significantly increased indicating an increase of outliers, this can also be seen when looking at the 99th percentile of executions for the system. The same trend continues when network connectivity is throttled to 64Kbps.

Device - Edge - Cloud performs as well at 100Mbps as it does at 128Kbps having a near identical mean, median and 75th percentile latency. It can however be seen that the slowest executions at 128Kbps seem to have been slightly impacted by the throttled network connection. Interestingly enough, the standard deviation for 128Kbps is more than 10% lower than for the same system running at 100Mbps, this indicates fewer outliers. When dropping the network bandwidth down to 64Kbps the mean and median increase by 10.6% while the standard deviation is increased by more than 350%.

Our proposed Device - Edge system was only tested once as it does not rely on any network connection in order to classify which road a car is driving on. The system performs at a mean latency of 173.8ms while having a low standard deviation of 14.8 showing a low spread of execution times. This is further proved by the 75th percentile being just more than 5ms slower than the median RTT latency.

	D-C 100Mbps	D-C 128Kbps	D-C 64Kbps	D-E-C 100Mbps	D-E-C 128Kbps	D-E-C 64Kbps	D-E
Sample size	609 (-1)	608 (-2)	604 (-6)	607 (-3)	607 (-3)	607 (-3)	605 (-5)
Mean (ms)	308.4	349.2	375.6	325.4	325.7	360.4	173.8
Median (ms)	307.2	342.4	364.3	324.2	322.7	350.2	172.1
Standard Deviation	19.7	51.6	89.2	17.1	21.3	96.9	14.8
75th percentile (ms)	318.2	352.9	372.3	331.8	330.4	357.5	177.7
99th percentile (ms)	383.9	467.5	629.9	395.1	413.5	577.0	230.6

Table 5.1: Summary of results from our controlled experiment.

5.1.1 Comparison of results with a bandwidth limitation of 100 Mbps

In Figure 5.1 we can see that the Device - Cloud system and Device - Edge - Cloud systems perform similarly to one another, but that the Device - Cloud system seems to perform slightly better when running at a 100Mbps fiber connection. The Device - Edge - Cloud system does have a narrower boxplot, meaning that more of its executions finished around the mean RTT. Our proposed Device - Edge system performs much better than both other systems, both in terms of mean value and when looking at how closely most executions finish around the mean. After removing outliers we can see that the Device - Cloud and Device - Edge - Cloud systems have approximately the same highest execution time.

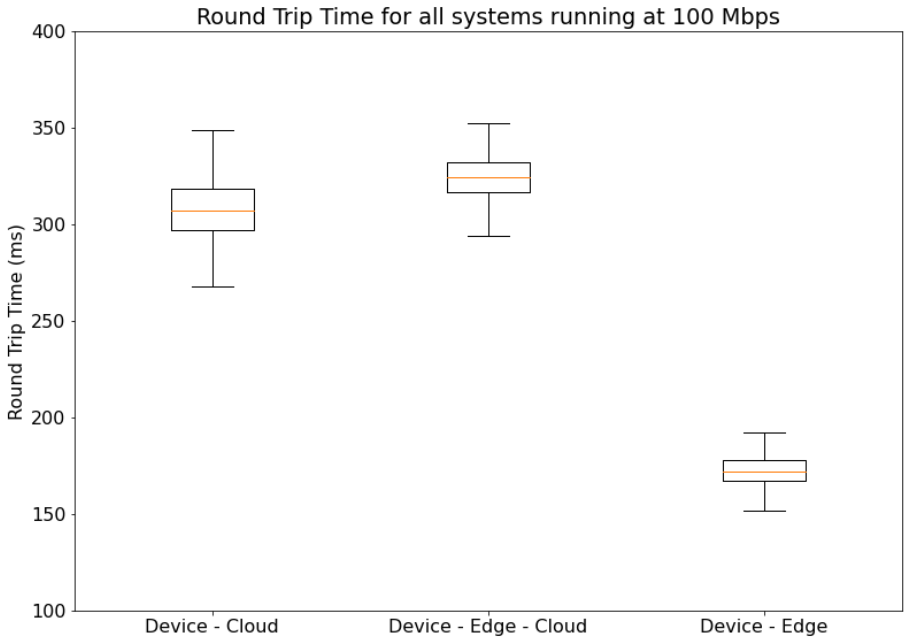

Figure 5.1: Boxplot of experiment results when running at 100 Mbps

If we look at the cumulative distribution functions for these systems at 100Mbps (Figure 5.2) we can see that the Device - Cloud actually have some executions that have a longer RTT than the Device - Edge - Cloud system. This can be reflected in the D-E-C systems slightly lower standard deviation seen in Table 5.1. This also explains the steeper slope of the distribution for the Device - Edge - Cloud system than its Device - Cloud counterpart.

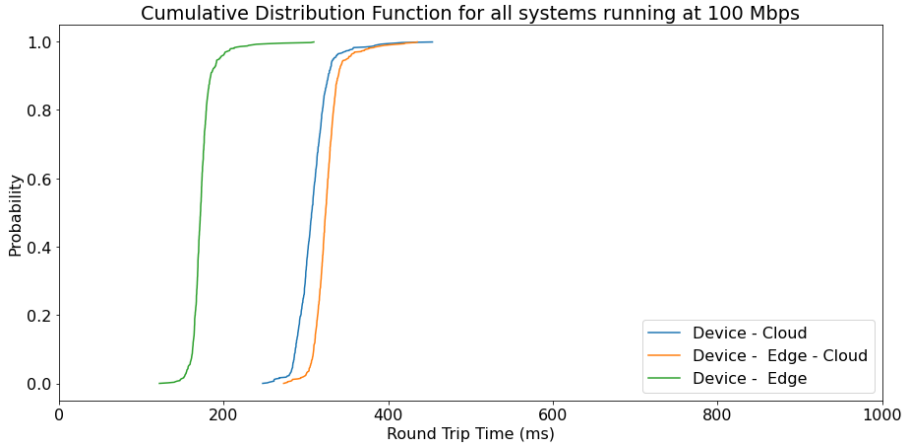

Figure 5.2: Cumulative Distribution Function for our three systems running at 100Mbps.

By looking at Figure 5.3, some similar characteristics can be seen between the systems running at 100 Mbps. All systems seem to have a similar distribution. They all look rather normally distributed but are slightly right skewed since all systems have outliers on the right side of their peak. This is also reflected in the longer tail at the top of the distributions cumulative distribution functions than at their bottom. None of the systems are multi-modal since they all have a distinct single peak. The histogram (Figure 5.3) does show that Device - Cloud and Device - Edge - Cloud have many overlapping values, but Device - Cloud have more of its mass to the left of Device - Edge - Cloud.

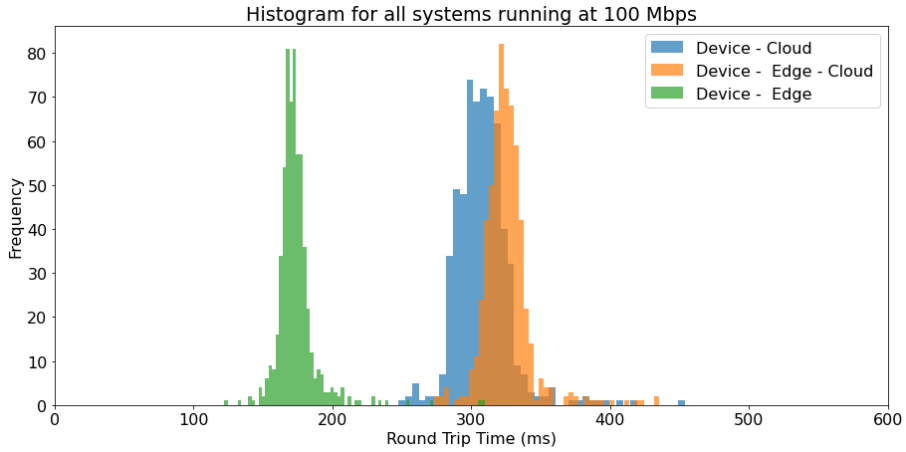

Figure 5.3: Histogram for our three systems running at 100Mbps.

5.1.2 Comparison of results with a bandwidth limitation of 128 Kbps

The difference between the Device - Edge system and the Device - Cloud system is more noticeable in these network conditions as the Device - Edge is not dependant on network speed. 128kbps might also be a more realistic setting than 100Mbps as a car is unlikely to have access to an upload speed of 100Mbps when driving anywhere else than a large city centre. In contrast to our results when running at 100Mbps we can see in Figure 5.4 that the Device - Edge - Cloud system performs better than the state-of-the-art Device - Cloud system when running at slower network connections. For 128 Kbps specifically we can see that our implementation of a Device - Edge - Cloud system runs at almost exactly the same RTT as when it is running on a full 100Mbps connection.

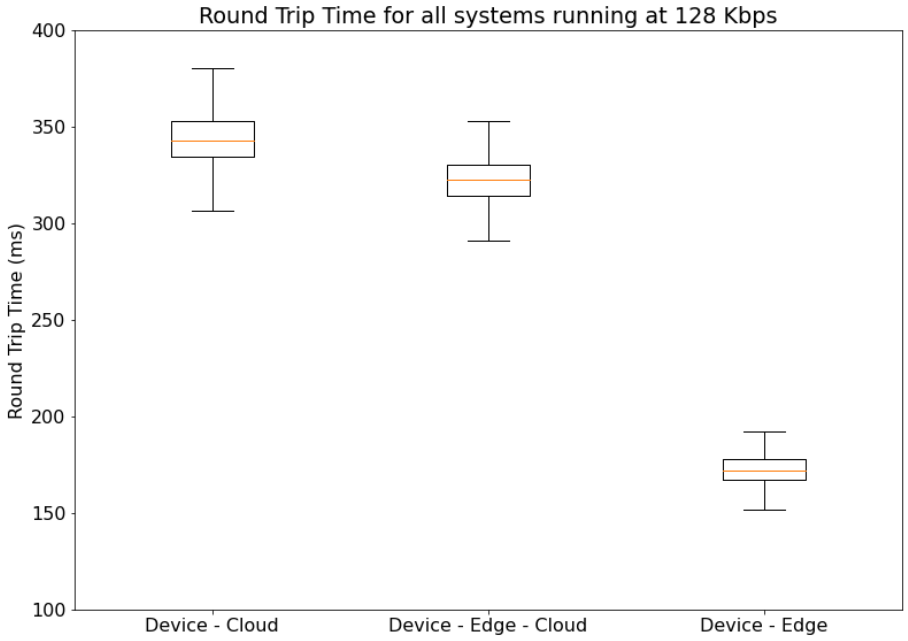

Figure 5.4: Boxplot of experiment results when running at 128 Kbps

In Figure 5.5 we can also see that the Device - Edge - Cloud system performs very similarly to when it is running at 100Mbps, it does however have a few more outliers at the top of the distribution that cannot be seen in the boxplot. The cumulative distribution function also shows that the Device - Cloud system seems to perform much worse than both other systems at these network conditions. The tail for this system now goes past the 1000ms mark at which we have decided to cut off the graph. Similarly to the results at 100Mbps, the slope for our Device - Edge - Cloud is steeper than the traditional Device - Cloud system.

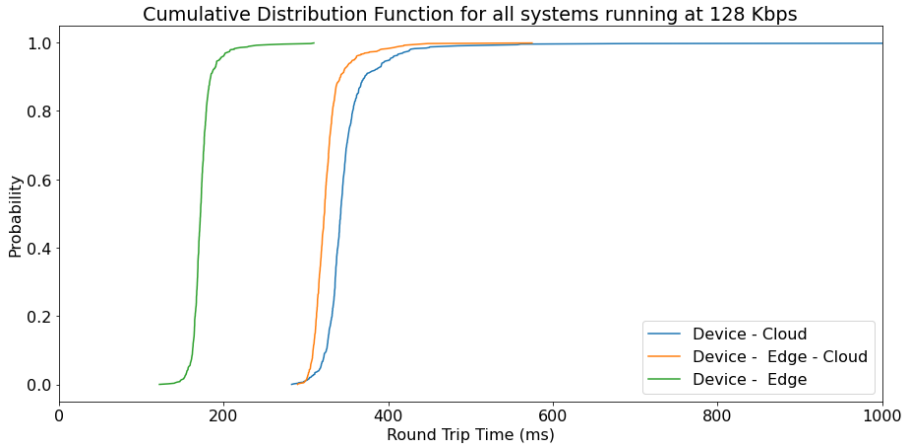

Figure 5.5: Cumulative Distribution Function for our three systems running at 128Kbps.

Even though the bandwidth has been limited to 128 Kbps, the histogram (Figure 5.6) shows the same characteristics as the histogram (Figure 5.3) for when the systems are limited to 100 Mbps. The systems are still not multi-modal since they each have a clear peak. One difference that is notable when comparing Figure 5.3 and Figure 5.6 is the distance between the peak for Device - Cloud and Device - Edge - Cloud. When those systems were limited to 100 Mbps, their peaks were much closer to each other. When changing the bandwidth limitation to 128 Kbps, the peaks are further apart and have switch place. Device - Edge - Cloud is clearly faster than Device - Cloud when the bandwidth limit is 128 Kbps. The number of overlapping values between Device - Cloud and Device - Edge - Cloud has decreased as well.

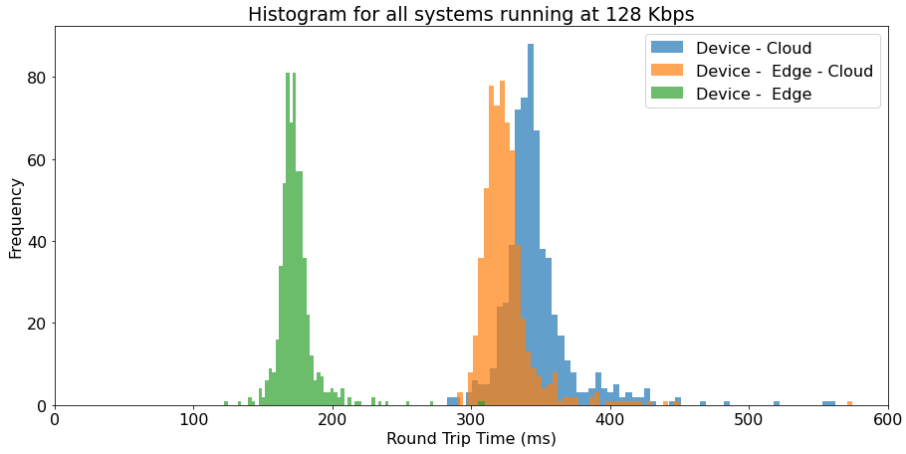

Figure 5.6: Histogram for our three systems running at 128Kbps.

5.1.3 Comparison of results with a bandwidth limitation of 64 Kbps

Finally, when looking at the systems running at a severely limited bandwidth of 64 Kbps we start to see that the performance of the Device - Edge - Cloud system decreases. It still outperforms the Device - Cloud system at these network conditions, but not as much as when running at 128 Kbps. In Figure 5.7 we can see that the boxplot for both the Device - Cloud and the Device - Edge - Cloud systems start to get compressed, which shows that the executions perform closer to the mean RTT. This seems counter intuitive as the standard deviation for these systems increase when running at a slower network connection, but this is due to the fact that the outliers now start to be even further away from the mean which influences the standard deviation.

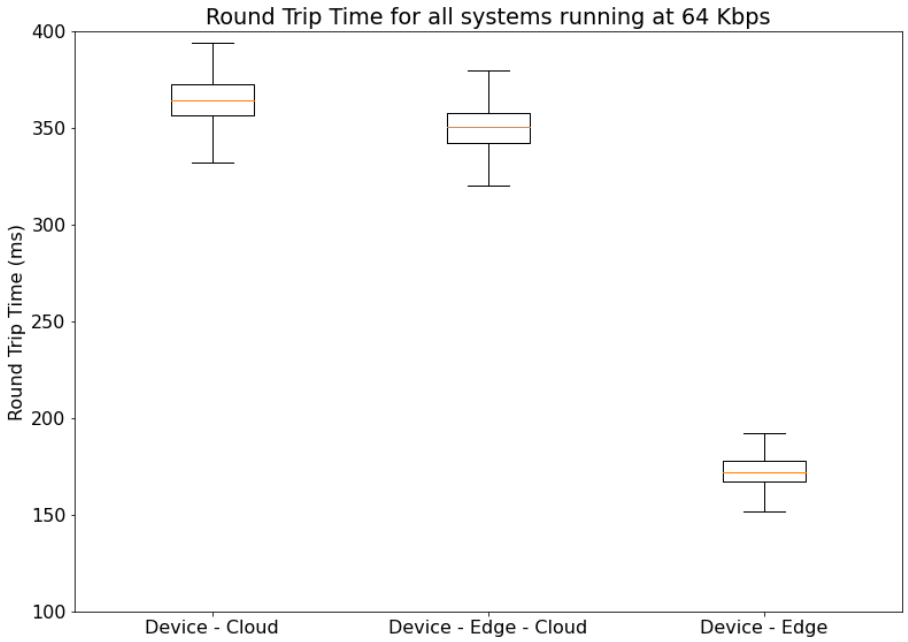

Figure 5.7: Boxplot of experiment results when running at 64 Kbps

The cumulative distribution function (Figure 5.8) also indicates that the systems start to behave more similar to one another when running at these network conditions. The Device - Cloud and Device - Edge - Cloud systems slopes start to have more similar inclines at 64 Kbps and although it might be hard to see as they lie so close to each other, both systems now have datapoints that go past to 1000ms mark when our graph gets cut off.

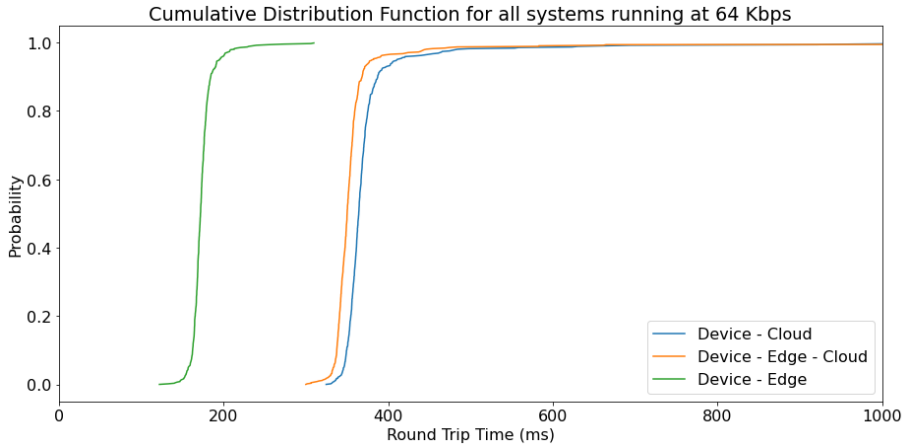

Figure 5.8: Cumulative Distribution Function for our three systems running at 64Kbps.

The histogram (Figure 5.9) does still show similar characteristics as both previous histograms (Figure 5.3 & 5.6). The systems do still have a single peak and are slightly right skewed. Device - Cloud and Device - Edge - Cloud are closer to each other then they were when the bandwidth limit was 128 Kbps. Device - Cloud and Device - Edge - Cloud also have a lot more overlapping values when limiting the bandwidth to 64 Kbps.

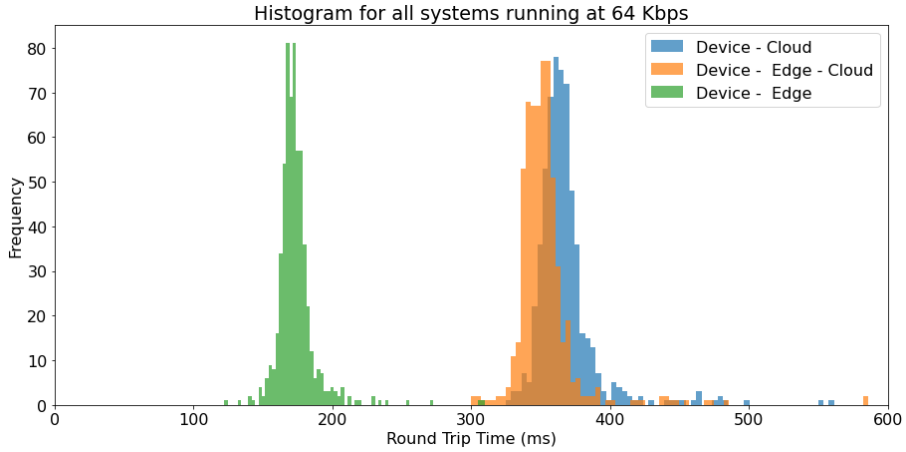

Figure 5.9: Histogram for our three systems running at 64Kbps.

5.2 Statistical Significance

Neither of the populations were normally distributed according to the performed Shapiro-Wilk test and the QQ-plots. Therefore a Kruskal-Wallis test was used for determining wether the different systems produce results with different distributions. As seen in the results of the Kruskal-Wallis test in Table 5.2 there is a difference in distributions at all bandwidths when tested for 95% confidence. After performing a Wilcoxon signed rank test to determine which distributions differ we can see in Table 5.3 that there is a significant difference between all systems.

Bandwidth	P-value
100 Mbps	$P < 0.05$
128 Kbps	$P < 0.05$
64 Kbps	$P < 0.05$

Table 5.2: Results of Kruskal-Wallis Test for all systems.

In order to determine how large of an impact the system difference has we have calculated the Cliff's Delta for all comparisons within the same bandwidth limitation. As seen in Table 5.4 there is a very large positive effect on latency from using the proposed Device - Edge system as compared to any other system. There is also a largely negative effect of using the Device - Edge - Cloud system when operating at unthrottled bandwidths. When bandwidth is limited to 128Kbps or 64Kbps there is however a largely positive effect of using the Device - Edge - Cloud system instead.

System A	System B	Bandwidth	P-Value
D-C	D-E-C	100 Mbps	$P < 0.05$
D-C	D-E	100 Mbps	$P < 0.05$
D-E-C	D-E	100 Mbps	$P < 0.05$
D-C	D-E-C	128 Kbps	$P < 0.05$
D-C	D-E	128 Kbps	$P < 0.05$
D-E-C	D-E	128 Kbps	$P < 0.05$
D-C	D-E-C	64 Kbps	$P < 0.05$
D-C	D-E	64 Kbps	$P < 0.05$
D-E-C	D-E	64 Kbps	$P < 0.05$

Table 5.3: Pairwise comparisons with Wilcoxon test.

System A	System B	Bandwidth	Cliff's Delta
D-C	D-E-C	100 Mbps	-0.604
D-C	D-E	100 Mbps	0.996
D-E-C	D-E	100 Mbps	0.999
D-C	D-E-C	128 Kbps	0.656
D-C	D-E	128 Kbps	1
D-E-C	D-E	128 Kbps	0.999
D-C	D-E-C	64 Kbps	0.579
D-C	D-E	64 Kbps	1
D-E-C	D-E	64 Kbps	1

Table 5.4: Cliff's Delta for all systems.

6

Discussion

This chapter reflects upon our findings from our controlled experiment in order to draw a conclusion to our research questions. Alongside this, any potential concerns about threats to the validity of our findings are discussed at the end of the chapter.

6.1 Research questions

This section aims to discuss how the results of our experiment answer our previously discussed research questions.

6.1.1 Research question 1

By using edge technologies to pre-process data before a sensor sends it to the cloud for classification, how much can latency be improved when operating at different network speeds?

Based on the results of our controlled experiment, using edge technologies to pre-process data before sending it to the cloud can reduce latency by upwards of 24ms when running on a slower network connection. If the systems have access to a large amount of bandwidth, however, one can expect that using an edge node to pre-process data is likely to slow the system down compared to sending it directly to the cloud. Alongside a lower mean at some network speeds, the edge layer also has the added benefit of reducing the spread of latency which can make the system more predictable.

When working on a 100Mbps internet connection it is clear that pre-processing data at the network edge does not lead to lower latency for the settings in this thesis. When our message payload to the cloud is so small the network can easily handle one node sending a payload of 100 datapoints every second.

Our Device - Edge - Cloud system likely was able to surpass the performance of the regular Device - Cloud system as the data was not pre-processed in a way that required a large amount of computational power. Thus, as both the AWS lambda function running on Amazons cloud servers and on our local hardware were able to pre-process the data at comparable speeds, the reduction in data payload from pre-processing our accelerometer data allowed for faster transmission to the cloud.

After doing a statistical analysis of our controlled experiment we found that the usage of an edge layer in this kind of setting had a largely negative effect on latency when running at a full 100Mbps. When running the experiment on a 128Kbps connections the edge layer had a largely positive effect instead and when throttled to 64Kbps a small positive effect was found.

This effect is likely due to the fact that the cloud has an advantage in terms of computational power over our local hardware. This advantage becomes decreasingly important as the network becomes more of a bottleneck as less bandwidth is available to the network. If the message payload required more computational power to process it is likely that the Device - Cloud implementation would outperform a Device - Edge -Cloud solution regardless of network connectivity. The same would apply if the difference between pre-processed and unprocessed payloads would have been smaller.

6.1.2 Research Question 2

By using edge technologies to both pre-process and classify data instead of using cloud technologies, how much can latency be improved when operating at different network speeds?

By running all computations at the network edge our experiments showed a significantly lowered latency compared to both other tested implementations. At a network connectivity of 100Mbps our Device - Edge implementation managed to outperform the state-of-the-art Device - Cloud implementation by 134ms. By dropping the connection to 64Kbps the difference between implementations were even greater, with a difference of more than 200ms between the mean latencies.

The usage of our Device - Edge system led to a very large positive effect on the latency when compared to all other systems at any given network connection. This effect seems to increase as available bandwidth decreases while showing no signs of stopping.

Similarly, to our findings in RQ1 this effect is likely to be less noticeable when working with data that needs computation-heavy pre-processing. Unlike our previous findings this system does not suffer from the bottleneck that is introduced when throttling the bandwidth below 128Kbps. This is unsurprising as our implementation is not reliant on the internet at all. Thus, no matter the payload size and computational power required there should exist some limit of network speed where our proposed system would outperform the state-of-the-art device - cloud solution.

6.2 Threats to validity

In this section potential threats to validity of our findings are discussed. The discussion is composed of two popular categories of validity, internal and external validity. Internal validity refers to the extent that the tested relationship is trustworthy and not influenced by other factors. External validity refers to the extent that our findings can be generalized to other settings within the automotive industry.

6.2.1 Threats to internal validity

The dependent variable for this study is round-trip time. It is measured as the time between the client sends a request and it gets a response. The measured time could be broken down into smaller segments. Each system consists of multiple steps which could be measured individually. Each system tested has two computational steps. These steps do not run on the same hardware for the different systems, the cloud-based computation is done in AWS Lambda and the edge computation on a Raspberry Pi 3. The Raspberry Pi is considered to have more computing power than most vehicles on the market. Thus, results might differ when running on weaker hardware.

All three systems run a pre-processor and a classifier. The pre-processor calculates statistical variables such as mean, median and variance. The classifier uses a random forest classifier with the calculated statistical variables as input. The computational time for the pre-processor and the classifier is included in the round-trip time. If different workloads had been used instead the results might have differed since the computational time would not have been the same.

When benchmarking the systems, ethernet was used and connected to internet with fiber. A tool called Wondershaper was used to throttle the bandwidth of the connection. However, this throttling does not affect the latency of the connection. A car would use a cellular connection and would therefore have longer round-trip times. The benefit of an ethernet connection for the benchmarks is that it is a stable connection and would therefore presumably give similar results whenever it is tested. A cellular connection could be affected by many external factors, such as weather and location, and therefore generate more variance to the testing.

When running our experiment, we only ran it once on one specific day. There is a possibility that the Amazon server we were communicating with was experiencing issues that caused it to slow down the latency. Such an issue is likely to impact the Device - Cloud system the most and would likely not impact our proposed Device - Edge system at all since it only communicates with AWS at startup. This could however be argued to be an advantage of the proposed new system since these kinds of disturbances lose their impact on latency.

6.2.2 Threats to external validity

The thesis intends to investigate the effect of using edge computing by testing a single cloud service provider, in our case AWS. Different cloud services might behave differently, especially since their edge computing platforms are also different. There will always be inherent differences between platforms, but by using the current market leader our experiment is tailored to a large part of the cloud development community.

The sensor data for our experiment was collected using a prototype rather than deploying our system directly to a real car. Although running the experiment in a live setting would produce more realistic results the network connectivity on a road can be impacted by many outside factors. This would make it hard to draw any real conclusions from such an experiment. Alongside this, when running our experiment, we had only a single device sending requests to AWS. Cloud servers connected to thousands, or even millions of nodes could suffer from performance issues not present when requests come from a single node at 1 second intervals.

In our experiment we send a specific message payload and perform relatively simple computations on that payload. This might make the results hard to generalize to other contexts within the automotive industry and testing different settings might give deeper insight in how these kinds of systems behave in general.

7

Conclusion

The purpose of this study was to evaluate how the usage of edge technology can reduce latency and bandwidth used for the automotive industry. This section concludes the findings of this report.

The latency between a device and the cloud is greatly affected by the usage of edge computing, whether you use it as a middle layer or move all computations to the edge of the network. These results are of interest to actors within the automotive industry, and our industry partner WirelessCar in particular. Some computations done within the automotive industry are very latency-sensitive and thus by knowing how to reduce latency, companies could greatly improve parts of their products.

To answer our research questions from Section 1.2, three systems were implemented using AWS Lambda and AWS Greengrass. To test the latency between these systems and the cloud over 4200 requests were made and the latency for each one was recorded. From our research the following conclusions can be drawn:

When working on an unthrottled fiber connection at 100Mbps you can afford to send a lot of data to the cloud and process it there. Thus, the use of an edge layer in this kind of setting is likely to lead to a decrease in performance as data gets processed by weaker hardware. Running the same experiment on a slower network connection, i.e., 128Kbps allows the edge layer to outperform the state-of-the-art Device - Cloud solution by an average of 24ms. When working on an even slower network connection of 64Kbps the edge layer still outperforms the state-of-the-art, but only by a mere 15ms.

By moving computational resources to the edge of the network we can disconnect the system from the network conditions entirely. The Device - Edge system managed to outperform the state-of-the-art by 134ms at the unthrottled fiber connection of 100Mbps, 175ms on a reasonable 3G connection (128Kbps), and 201ms on a perfect 2G connection (64Kbps). Regardless of network speed the proposed Device - Edge system manages to outperform the state-of-the-art by a significant amount. At the worst reasonable network conditions, the proposed system outperforms a Device - Cloud equivalent by reducing RTT latency by upwards of 50%.

7.1 Future work

Our research has shown that adding an edge computation layer between a device and the cloud can reduce latency when working at specific network conditions with specific message payload and specific computational requirements. Future work in this field could preferably study the implications of varying message payloads and computational requirements within the automotive industry. This could include looking at sensors collecting different kinds of sensor data, such as video feeds which can collect a larger amount of data. Such a large amount of data could need more computational power to pre-process which could be in favor of the state-of-the-art Device - Cloud system.

Future work could also encompass the use of different trigger types within AWS Lambda. Our research has been using a REST API, other ways to communicate with AWS servers include uploading data to an S3 bucket or streaming it through Amazons DynamoDB.

There also exist several different edge computation platforms other than AWS Greengrass, and future work could be done in benchmarking different platforms. Other platforms could include Microsoft Azure IoT or Google IoT. It should be noted that it is usually hard to compare different platforms to each other as configuring them to be similar to each other is not an easy task.

Studying the effects of running a similar experiment on other AWS servers could also be beneficial as it is plausible that server location has an impact on RTT latency, especially if the invoker and the lambda functions are in different geographical areas.

Finally, running the experiment on an actual car is likely the best way to get results from realistic network conditions. By continuously running our systems from a driving vehicle in different areas and comparing their latency, researchers can more realistically simulate real-world conditions. This would give a better insight for developers in the automotive industry of the usefulness from the Device - Edge system proposed in this report.

Bibliography

[1] P. Bhale, V. Singh, and M. Sujeet, "Achieving cloud security using third party auditor, md5 and identity-based encryption," in *2016 International Conference on Computing, Communication and Automation (ICCCA 2016)*, 04 2016, pp. 1304–1309.

[2] [Online]. Available: mqtt.org

[3] Y. Liu, D. Lan, Z. Pang, M. Karlsson, and S. Gong, "Performance evaluation of containerization in edge-cloud computing stacks for industrial applications: A client perspective," *IEEE Open Journal of the Industrial Electronics Society*, vol. 2, pp. 153–168, 2021.

[4] C. Llopis-Albert, F. Rubio, and F. Valero, "Impact of digital transformation on the automotive industry," *Technological Forecasting and Social Change*, vol. 162, p. 120343, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0040162520311690

[5] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[6] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol. 62, no. 12, p. 44–54, nov 2019. [Online]. Available: https://doi.org/10.1145/3368454

[7] F. Richter, "Amazon leads $180-billion cloud market," Statista.com, Feb. 8, 2022. [Online]. Available: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/

[8] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 27–33.

[9] M. N. Sadiku, S. M. Musa, and O. D. Momoh, "Cloud computing: Opportunities and challenges," *IEEE Potentials*, vol. 33, no. 1, pp. 34–36, 2014.

[10] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing — the business perspective," *Decision Support Systems*, vol. 51, no. 1, pp. 176–189, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167923610002393

[11] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, "Cold start influencing factors in function as a service," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018, pp. 181–188.

[12] S. Dunn, "Serverless architectures with aws lambda," Amazon, Tech. Rep., 2017. [Online]. Available: https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf

[13] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of function-as-a-service computing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 1063–1075. [Online]. Available: https://doi.org/10.1145/3352460.3358296

[14] M. Bertilsson and O. Grönqvist, "Performance comparison of function-as-a-service triggers," M.S. thesis, Chalmers University of Technology, Gothenburg, Sweden, 2021. [Online]. Available: https://hdl.handle.net/20.500.12380/302822

[15] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 162–169.

[16] E. Hamilton. (2018) What is edge computing: The network edge explained. [Online]. Available: https://www.cloudwards.net/what-is-edge-computing/

[17] Amazon Web Services. (2022) Aws iot greengrass documentation. [Online]. Available: https://docs.aws.amazon.com/greengrass/index.html

[18] L. Baresi, D. Filgueira Mendonça, and M. Garriga, "Empowering low-latency applications through a serverless edge computing architecture," in *Service-Oriented and Cloud Computing*, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds. Cham: Springer International Publishing, 2017, pp. 196–210.

[19] I. Pelle, J. Czentye, J. Dóka, and B. Sonkoly, "Dynamic latency control of serverless applications operated on aws lambda and greengrass," in *Proceedings of the SIGCOMM '20 Poster and Demo Sessions*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 33–34. [Online]. Available: https://doi.org/10.1145/3405837.3411381

[20] J. Skirelis and D. Navakauskas, "Classifier based gateway for edge computing," in *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, 2018, pp. 1–4.

[21] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[22] K.-J. Stol and B. Fitzgerald, "The abc of software engineering research," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 3, sep 2018. [Online]. Available: https://doi.org/10.1145/3241743

[23] S. Shukla, V. Khare, S. Garg, and P. Sharma, "Comparative study of 1g, 2g, 3g and 4g," *J. Eng. Comput. Appl. Sci*, vol. 2, no. 4, pp. 55–63, 2013.

[24] A. Qureshi, J. Carlisle, and J. Guttag, "Tavarua: Video streaming with wwan striping," in *Proceedings of the 14th ACM International Conference on Multimedia*, ser. MM '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 327–336. [Online]. Available: https://doi.org/10.1145/1180639.1180714

[25] B. Hubert, J. Geul, and S. Séhier, "The wonder shaper," https://github.com/magnific0/wondershaper, 2021.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.