Fog-based IoT Framework for Large-Scale Data Classification

by

Marc Jayson Baucas

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Masters of Applied Science
in
Engineering

Guelph, Ontario, Canada

© Marc Jayson Baucas, December, 2019
ABSTRACT

FOG-BASED IOT FRAMEWORK FOR LARGE-SCALE DATA CLASSIFICATION

Marc Jayson Baucas
University of Guelph, 2019

Advisor:
Petros Spachos

Internet of Things (IoT) is a network that is used to develop large-scale data sensing applications. As networks grew, servers could no longer handle the processing constraints. As a result, processes were reallocated to the end devices, which led to high power consumption. Then, fog computing was incorporated to offload processes between the cloud and the end device into an intermediary device. However, an unoptimized fog-based IoT network can still run into the same issues. Therefore, this thesis aims to propose an architectural template or framework that can be used to balance the fog-based IoT network for large-scale data sensing. For feasibility testing, Urban sound classification was the selected application. Therefore, reallocation and active power states were included as techniques in configuring the framework. The results optimized the framework and created a reasonable balance between cloud and fog-based networks that demonstrated low power consumption and lesser server strain.
ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for giving me the strength, wisdom, knowledge, and opportunity to persevere through this research study with the right mindset. Only through God’s grace, would I have been able to obtain such an achievement. And so, I humbly return the glory to Him who made all of this possible.

As I embarked on this journey, I met my supervisor Dr. Petros Spachos. He became my teacher, role model, friend and guide. Dr. Spachos has provided me with the right amount of guidance and discipline that directed me towards gaining more knowledge about proper research conduct. He gave me the freedom to pursue my research without any restraints while making sure that I would not stray away from the reason why I started this journey in the first place. Without his guidance, I would not have been able to complete this degree in a timely and organized manner. For this, I am grateful for his supervision.

Along the way, I met like-minded colleagues that were on the same quest for knowledge. They have filled each semester with motivation and camaraderie that contributed to my drive in aiming to complete my thesis. This journey would have been empty without our constant fellowship and the memories that we shared. I would also like to thank the University of Guelph and its staff for their services and compassion that made my graduate experience extraordinary.

Let me dedicate all this work to my parents, Marcos and Zenaida Baucas. Your sacrifices were not in vain. I would not be here if not for the two of you. Your dreams of giving me and my siblings a better life has brought me here standing as a living testimony of all your hard work. May you take pride in
this achievement as the fruits of your labour. To my siblings: Marianne Joy Baucás and Mazenne Jane Baucás, your words of encouragement and gestures of compassion gave me the motivation to keep pushing on towards this achievement. To my family in the Philippines: my aunts and uncles, grandparents and cousins, I would like to thank you for all the love and support that you’ve given me.

To those whom I love and cherish the most, know that I would be nowhere near where I am today if not for your love and support. From all that defines me, I thank all of you.
## Contents

Abstract ii

Acknowledgements iii

List of Tables viii

List of Figures ix

List of Abbreviations xi

1 Introduction 1

1.1 Problem Definition 3

1.2 Contributions 4

1.3 Organization 5

2 Fog Computing in IoT 6

2.1 Cloud vs Fog 6

2.2 Types of Fog 10

2.3 Solutions to Fog related issues 13

2.4 Related Works 14

2.5 Chapter Summary 16

3 Design Overview 17
3.1 Experimental Setup ........................................ 20
   3.1.1 Design Specifications ................................ 20
   3.1.2 Sound Recorder ....................................... 21
   3.1.3 Sound Classifier ...................................... 22
      3.1.3.1 Dataset ........................................ 22
      3.1.3.2 Feature Extraction ............................. 23
      3.1.3.3 Neural Network ................................. 26
   3.1.4 Testbed Setup ........................................ 27
3.2 Chapter Summary ........................................... 28

4 Testing and Evaluation ...................................... 30
   4.1 Testing Overview ....................................... 30
      4.1.1 Preliminary Power Consumption Tests ............. 32
      4.1.2 Preliminary Latency Tests ........................ 36
      4.1.3 Introduction of the Third Configuration ........... 37
   4.2 Chapter Summary ........................................ 41

5 Relevant Applications ....................................... 42
   5.1 Urban Noise Categorization ............................. 42
      5.1.1 System Overview .................................. 43
      5.1.2 Architecture Design ............................... 44
      5.1.3 Evaluation ......................................... 47
   5.2 Remote Patient Monitoring ............................... 51
      5.2.1 IoT in RPM ......................................... 51
      5.2.1.1 Issues with current applications ............... 52
      5.2.2 System Overview ................................... 54
      5.2.3 Design .............................................. 55
      5.2.4 System components ................................. 56
5.2.5 Design flow ........................................ 58
5.2.6 Preliminary Results ............................... 61
5.3 Chapter Summary .................................... 63

6 Future Work and Extensions .......................... 64
  6.1 A more decentralized approach to Fog-IoT networks using permissioned blockchains and BLE ....................... 64
  6.2 Improving network security using private blockchains and cryptography ........................................ 66
  6.3 Chapter Summary .................................... 67

7 Conclusion ............................................. 69

Bibliography ............................................. 72
## List of Tables

4.1 Testing results for power consumption and runtime.  
4.2 Testing results for latency.  
4.3 Configuration scoring.  
5.1 Noise categorization of UrbanSound8K dataset.
List of Figures

1.1 server reliance in cloud-based IoT network when classifying data from different sources ................................. 2

2.1 Cloud-based IoT network ........................................... 7
2.2 Fog-based IoT network ............................................. 9
2.3 Cloud-Fog-Device example ......................................... 11
2.4 Fog-Device example ............................................... 12

3.1 Edge/Fog-based setup logic flow ................................. 18
3.2 Cloud-based setup logic flow ................................. 19

4.1 Power consumption test setup ................................. 32
4.2 Power consumption comparison between configuration A and B over 20 iterations ................................. 33
4.3 Difference in power consumption with and without sending the data to the server after recording ................................. 34
4.4 Runtime comparison between configuration A and B over 20 iterations ........................................... 35
4.5 Latency test setup ................................................. 36
4.6 Latency comparison between configurations A and B ........................................... 37
4.7 Hybrid setup logic flow ............................................... 39
5.1 Network setup for the benchmark. ........................................... 45
5.2 IoT setup of proposed design. ................................................. 46
5.3 Power consumption of local and server configurations at 20 iterations. 48
5.4 Latency of the local and server configurations at 20 iterations. .... 49
5.5 Proposed RPM room setup. .................................................... 54
5.6 General design flow. ......................................................... 58
5.7 Raspberry Pi operation flow. .................................................. 59
5.8 Jetson operation flow. ......................................................... 60
5.9 Training accuracy and validation. .......................................... 61
5.10 Training loss and validation. .............................................. 62
List of Abbreviations

ANN  Artificial Neural Network
BLE  Bluetooth Low Energy
CNN  Convolutional Neural Network
DCT  Discrete Cosine Transform
FFT  Fast Fourier Transform
IoT  Internet of Things
LTE  Long-Term Evolution
MEMS Micro-Electrical-Mechanical System
MFCC Mel-frequency Cepstral Coefficient
NN   Neural Network
RPM  Remote Patient Monitoring
STFT Short-time Fourier Transform
TCP  Transmission Control Protocol
USB  Universal Serial Bus
Chapter 1

Introduction

The Internet of Things (IoT) is a network of devices that are wirelessly connected through the Internet [1, 2]. Some known applications that uses IoT are mobile asset tracking, secure communication, and environment sensing [3]. At the centre of most of these IoT-based applications is a cloud server. It provides storage for these services, which led to cloud-based IoT networks. Integrating cloud-based computing to IoT allowed most applications to incorporate large-scale algorithms such as; image processing, natural language translation, sound classification and other forms of data classification in their services [4]. However, most cloud-based networks relies too much on the cloud for most of its processing and data handling [5]. Figure 1.1 depicts this reliance on data classification towards the cloud in a cloud-based IoT network.

Cloud service providers are accountable for maintaining a secure and accessible means of storing and sharing data, due to this reliance [6]. In addition, further exposure towards applications that conducted data sensing and classification revealed that the cloud struggles sensing data in real-time [7]. One solution is to move the processing from the cloud to the devices in hopes of relieving the processing load from servers. However, most large scale data
sensing applications that incorporate an IoT network makes use of low-end devices for scalability and cost efficiency [8]. In addition, most processors on low-end devices used for data sensing were not built to withstand these processing requirements due to the high demand in power. With both options being disadvantageous to either the server or the device. Another option is the incorporation of fog computing. Fog computing introduces an extra devices layer located in between the end device and the cloud. This layer provides the network the ability for better regulation of data transmissions and for easier load balancing [9]. As a result, a greater balance in processes within the architecture could optimize the fog-IoT network to minimize the effects of these issues on the application.
1.1 Problem Definition

The addition of fog computing can result in improvements in data sensing for wireless IoT networks. However, if processes and application loads are not well balanced within the network, the network will run into end device power consumption and server data management issues. To address these issues, the intention is to create a fog-based IoT architectural template or framework that can be used to manage complex applications on data classification that uses low-end processing devices. To be able to verify the feasibility of this framework, this thesis chose Urban sound classification as its application for testing. Urban sound classification is a type of environmental data sensing framework that focuses on categorizing the different sounds within a city area [10]. It is composed of multiple sensing devices for recording sound and a central server for storing data and classification results. To make the proposed framework feasible as a large scale network application, it must address the following issues as previously mentioned [9] [11];

1. **High Edge Device Power Consumption** - In urban sound classification, devices transmit data to the server at intervals [12]. Each device must transmit independently and is self-sustaining. Therefore, they should be optimized before the service deploys them. However, the proposed framework is intended for low-end devices. One of the main issues of these type of sensing devices is their battery life [13]. Since services deploy these devices in areas without a reliable power source, they need to account for battery capacities. Sound is being recorded continuously by this sensing framework. As a result, power becomes an issue if the software driving the device is not optimized. Power consumption must be minimized to get the most out of each deployment. It could define the lifespan of the framework.
2. **Difficult Data Management due to Decentralization** - The end devices within a fog-based network are decentralized [14]. If it is improperly optimized and programmed, it could lead to inaccurate data [13]. Data precision is essential in classifying the types of information that are being sensed by the devices. Without a proper data managing design for the end devices and the fog nodes, data traffic will be an issue for the server that will store all the results. Real-time applications such as environmental sensing, will have data that will continuously be streamed to the server by multiple nodes [7] [12]. If the receiving end of the server cannot handle the volume of data, then the proposed framework will less likely be feasible due to scaling issues.

This thesis proposes an architectural template or framework that explores the different configurations of Fog-based IoT networks to find an iteration that minimizes these issues in power consumption and data management. Also, it investigates the network optimizing capabilities of process reallocation and powering states.

### 1.2 Contributions

The works presented in this thesis focuses on an analysis of fog-based IoT networks and using it for applications on large-scale data sensing. Using this framework takes advantage of wireless technology to obtain and classify data from low-end devices that are connected through a fog-based IoT network. Included in this thesis are the following components and contributions:

- A fog-based IoT framework that makes use of urban sound classification as its application to justify its scalability and efficiency presented in [15].
Two applications that made use of some aspects of the fog-based IoT framework. First is an implementation of a modified version of the UrbanSound8K dataset to classify a sound as either noise or not in an urban setting [16]. Second is an assistive framework that uses the Google speech dataset for remote patient monitoring [17].

- A testbed to test the fog-based IoT framework[18].

1.3 Organization

The rest of this thesis is as follows: An in-depth discussion of fog computing in IoT network is found in Chapter 2. Next, Chapter 3 presents the proposed design. Then, the conducted tests and evaluations are presented in Chapter 4. Chapter 5 enumerates the relevant applications presented by this thesis on fog-based IoT networks. Then, Chapter 6 cites potential future extensions and improvements to the proposed framework. Finally, Chapter 7 concludes the thesis with an overview of all the observations and results.
Chapter 2

Fog Computing in IoT

The very first iterations of IoT networks lacked scalability as networks grew [9]. Integrating cloud computing enabled IoT to handle the incoming wave of large-scale applications [4]. However, the introduction of innovative applications and services using more complex algorithms in data sensing resulted in issues with cloud-based IoT networks [5]. Most services now demand applications with features such as real-time sensing, larger capacities, which cloud servers are no longer able to sustain by itself [19]. As a result, networks incorporated fog computing to extend the cloud. Also known as clouds at the edge, it uses nodes within the boundaries of the network to carry out computations [20]. Fog computing creates a computing facility for IoT services or other latency-sensitive applications through a distributed architecture [13].

2.1 Cloud vs Fog

Cloud computing is a model that enables convenient, on-demand network access to a shared pool of computing resources [21]. This shared pool is made possible through cloud storage technology. Cloud storage technology provides
users with a means to store, retrieve, or back up their data through an online storage [22]. It makes this possible with the use of cluster applications, network technology, distributed file systems, and many more online resources. Partnered with IoT, cloud computing becomes a revolutionary tool for devices that rely on online servers for data storage.

As technologies progressed, network applications have started demanding more resources and required more processing power. As a result, traditional servers began to have difficulties with the increase in the cost of operation and maintenance. Cloud computing enters in as a solution by introducing parallel processing. Parallel processing allows a remote server to offload tasks into subroutines systematically. Figure 2.1 shows a diagram that visualizes

![Cloud-based IoT network](image-url)
the concept of a cloud-based IoT network and some of its known features and applications [4, 22].

However, even though cloud computing has notable strengths over traditional systems. It also comes with some bottlenecks and risks [21, 23]:

1. **Server overloading** - In cloud computing, most low-end sensing devices have their uses reduced to only sending their data to the cloud servers while waiting for the results of the computation. However, as the number of users connected to a cloud increases, data starts being received by servers at higher volumes and velocities. This sudden increase contributes to the data traffic, which results in a bottleneck. Due to data congestion, large-scale servers are significantly slowed down by high latency. High latencies will render a server unresponsive. As a result, the potential of integrating any form of real-time application is reduced by this bottleneck.

2. **Scalability issues due to centralization** - In a centralized cloud infrastructure, capacity defines the number of users a service can handle. Managing the data traffic within a server will become more tedious if the network funnels all the resources into one centralized unit. A server that can only handle a limited number of devices cannot be scaled up. As a result, when scaling up a cloud-based architecture, its capacity will always be its limiting factor.

Fog computing is an alternative to cloud computing that can be used to move storage and computation components of applications from the server to the IoT devices [24]. The difference is that cloud computing uses a server while fog computing uses a network edge or an edge device. It is an end-user device, located close to the IoT network [25]. It not only provides data but also processes data. In fog computing, the end-user device requests the services
and also handles the computing task. Cloud computing offloads the data management to the server while edge computing distributes the management load towards its edges.

On the edge of the network are the fog which are devices located between the server and the data source [25]. It effectively handles computing tasks that include the processing, storage, caching, and load balancing of the data transported to and from the cloud. With the existence of this intermediary data processing area, the computation is offloaded and distributed to its edges. Figure 2.2 shows a diagram that visualizes the concept of a fog-based IoT network and some of its notable features and applications [9]. This computing

Figure 2.2: Fog-based IoT network.
design points out benefits and improvements to the standard cloud computing infrastructure[26, 27]:

1. **Better offloading and reduced server strain** - Fog computing allows the network to offload the data processing from the cloud to the fog nodes. Offloading reduces any resulting data traffic and server strain. Therefore, more users can now be managed more efficiently with edge computing.

2. **Scalability through parallelism** - Fog computing allows scalability through parallelism and decentralization. It focuses on adding edge devices to the network. On the other hand, cloud computing scales a cloud server by increasing its size in terms of data capacity.

### 2.2 Types of Fog

The internal architecture of a fog-based IoT network is composed of multiple device layers [11]; the cloud, the fog, and the end devices. These layers can be arranged to create two different types of fog-based IoT network configurations [9]: Cloud-Fog-Device and Fog-Device. In an IoT platform, each device layer is connected to one another using a wireless medium (e.g. Bluetooth, LTE, ZigBee). Each layer can be rearranged, while some can be removed to put focus towards certain aspects of the network. The following is a discussion of these layer arrangements:

1. **Cloud-Fog-Device** - A representation of this arrangement is shown in Fig. 2.3. It consists of three layers arranged in increasing order based on storage and computing capabilities.

   a) **Device** - An end device in an IoT network can either be mobile or fixed depending on its application [9]. Most mobile IoT devices can either be worn or carried. Some examples of heterogeneous user-oriented devices are
fitness trackers, smartphones, and smartwatches. Fixed IoT devices have specific areas where they are deployed depending on their intended tasks. These types of end devices have limited energy and computing resources. They are only there to collect data [28]. Not many computations are carried out on these devices. The edges send the data to the higher layers for analysis and long-term storage.

b) Fog - A fog is any device that is capable of computing, rerouting, and storing data [13]. Some examples of these devices are switches, routers, proxy servers, bridges, and any other computing device [9]. As a result, time-sensitive applications can run in fog nodes.

c) Cloud - This layer is a computing and storage platform that provides various IoT applications. Cloud servers accommodate on-demand data

Figure 2.3: Cloud-Fog-Device example
storage and other server resources that are accessible to any device connected to the Internet [9, 29].

2. Fog-Device - A representation of this arrangement is shown by fig. 2.4. It is composed of two layers with a similar arrangement in terms of storage and computing capacities.

a) Fog - In this layer, the fog nodes cooperate without the use of a cloud server to provide service to the devices [9]. Each node decides on actions within a network using a distributive structure where they work as a collective unit [14]. The fog is the highest layer in this category. As a result, its nodes will have to handle the storing of device data without a dedicated cloud server for storage.
b) Device - Similar to Cloud-Fog-Device, the lowest layer is the end device. However, more of the computations are allocated by the network to the end device [14]. With the absence of a cloud server, more processes are required to be placed on the device to relieve the strain on the fog servers. The edges will still send the data to the fog layer for analysis and storage.

This thesis focuses on using the Cloud-Fog-Device structure for the proposed framework. This decision was attributed to the simplicity of its potential implementation compared to the Fog-Device structure. A Fog-Device network would require an algorithm that would allow better collaboration between multiple servers to achieve proper decentralization. As a result, a more complex design would be warranted. On the other hand, Cloud-Fog-Device will be easier to implement, since the layers and their purposes are better defined. Also, the data flow is more straightforward for Cloud-Fog-Device as it travels from end device to fog to cloud then back. However, Fog-Device is more complex because it warrants a protocol that can manage the data received from the end devices before they are presented.

2.3 Solutions to Fog related issues

It was previously mentioned that fog-based IoT networks run into end device high power consumption and server data management issues. The following are proposed solutions that are aimed to address one or both issues to improve the proposed fog-based IoT framework. To test the framework, Urban sound classification was selected as the application to be implemented. Urban sound classification determines the type of sound in a city using environmental sensing [10]. It is an IoT application that monitors an area for informatic purposes [30]. For urban sound classification, its focus is the different sounds of the city. Further discussion of the solutions on the previously cited issues are as follows;
1. **Active Low and High Power States** - Like any sensing application, continuous data transmissions are a concern to end devices, especially to low-end devices that have a limited power supply [30] [31]. In IoT-based fog computing that make use of these devices, power management is important to keep a standard quality of service [13]. A design can optimize each device by setting each peripheral within the device to only be powered if needed. Implementing power states on the end devices could increase the efficiency of the proposed framework.

2. **Process reallocation** - In data sensing, information will be in and out of the fog and cloud servers continuously. Similar to all sensing frameworks, it will cause each server to fail once it can no longer manage the data traffic efficiently [30]. However, if a design can be implemented to keep the incoming data at a stable and workable size, then it will be easier to manage. Transmitted data should not vary and remain small. The proposed solution aims to find the best process configuration within each layer of the proposed framework. This solution can reallocate processes to achieve a balance between computation load and times [11]. With reallocation, the network can anticipate the incoming data. As a result, implementing proper server scheduling methods becomes possible. Also, with processing loads balanced, latency will be less of an issue, which reduces data traffic to a minimum [13].

### 2.4 Related Works

In [32], the authors created a general scope of the potential of fog computing in terms of large scale data analysis. They proposed a multitier architecture for smart cities with the use of fog computing. Their paper opens a possibility for fog computing to drive the next wave of smart city applications. Urban sound
classification is one of the intended sensing applications for smart cities. By being able to create a sensory architecture to gather data around a large area, smart cities are made to be a more plausible implementation. However, their design provides an important detail that this work aims to reinforce. Cloud computing still carries a critical role in enabling the fog.

A similar design was discussed in [33]. They proposed a hierarchy of layers that combined fog and cloud technology to handle data from a larger scale. The network collects the data through the edge devices. Each device is grouped locally using fog nodes. The data then gets pre-processed from the multiple fog layers until it can be analyzed and stored globally within the cloud. This work takes a similar stance by aiming to utilize the cloud as more than just a storage device but also a medium for more general computing.

Both papers focused on a feature to benchmark the performance of their designs. [32] decided to use the number of jobs completed by each fog layer at given time intervals. On the other hand, [33] chose to test their design according to the accuracy of their classifier. This thesis decided on using power consumption, runtime, and latency, which differs from the previously discussed works. These metrics were selected to achieve a more in-depth investigation of the behaviour of the server and the end devices at different processing load configurations. Although the previously mentioned works were able to create a framework for a design that incorporates both cloud and fog computing, there are drawbacks in offloading processes within the network hierarchy.

Moving computing and processing loads from the server to the edges require devices to have more processing power. As a result, the power consumption and application runtime among edges increases. Too much of this offloading might render a system undeployable. However, focusing on minimizing the power and runtime requirements from the devices will lead to server strains.
Therefore, a proper load balance is needed within the network to make this system scalable. By focusing on these factors, this metric was used to base the feasibility of the proposed sensing platform. To test this framework, an application that demanded a large amount of attention to power consumption, runtime, and server strain as desired. This demand leads to the decision of using urban sound classification.

2.5 Chapter Summary

Presented in this chapter were information on fog-based IoT networks. Fog computing in IoT networks presents an advantage of scalable designs through better process management and allocation. These concepts were considered, and solutions were discussed to address the issues cited in the problem statements in the previous chapter. These solutions were further motivated by the discussed related works and how this thesis approaches each issue differently. The next chapter takes these solutions and proposes a design for the framework.
Chapter 3

Design Overview

This chapter presents the proposed fog-based IoT framework. It is composed of a cloud layer connected to a fog layer that is connected to multiple end devices. Each device layer is programmed to communicate with one another wirelessly. The aim of this thesis is to create the best configuration of the proposed fog-based IoT framework. Initially, to create a better configuration, a benchmark was established by configuring the framework to be either cloud focused or edge focused. Depending on the configuration, the sound data is either classified within the end device or the server. To test each configuration, the power consumption of the end device and the latency of the server was measured. Power consumption is one of the mentioned bottlenecks in terms of scaling a system. The sound classification was the selected application due to its high processing requirements that result in high demand in power from its sensing nodes or end devices. Latency is another limiter for servers that attempt to scale up by adding more connected users. Servers tend to run into issues in managing incoming data once it reaches a certain threshold of connected devices or data traffic. To achieve a scalable design, the aim is to
investigate the feasibility of each framework configuration by simulating them under the proposed sound classification framework.

The first setup was based on edge computing. In this setup, the device does both the recording process as well as the classification. It was labelled as configuration A. The general flow starts similarly with configuration A where the end devices collect the sound data from the environment. However, instead
of sending the sound data to the server to be classified, its classification is carried out within the end device. The results are then wirelessly sent to the server storage. Then, the server notifies the device via a message once the results have been received and stored. Figure 3.1 shows the logical flow of the edge/fog setup or configuration A as implemented in the proposed design.

On the other hand, the second setup was based on cloud computing. It was labelled as configuration B for the conducted experiments. The general flow of this setup focuses all of the processing to the cloud. It starts with the edge devices, which is in charge of collecting the sound from the environment.
Ideally, each device will be placed strategically around an area to get as much sound data as possible. Then, the end devices will send the sound data to the cloud server wirelessly. After receiving the data, the server converts these sound files into feature maps. Each feature map is used to classify the incoming sound. Then, the source device is notified by the server that it has successfully stored the classification results. Figure 3.2 shows the logical flow of the cloud-based setup or configuration B, as implemented in the proposed design.

The purpose of starting with these two setups was to distinguish the differences in processing time and power consumption of the two represented computing practices in a controlled environment.

3.1 Experimental Setup

The experimental setup is composed of 4 major sections. These sections are as follows; Design Specifications, Sound Recorder, Sound Classifier, and Testbed Setup.

3.1.1 Design Specifications

In terms of hardware used, configurations A and B are the same. This decision was made to eliminate any extraneous variables on the experimental setup. To represent the end devices of the network, using a Raspberry Pi 3 Model B. Pis were used because they were easier to program and were capable of rapid prototyping. Each Pi 3 model B uses a Quad-core ARM Cortex A53, with a processing benchmark of 1.2 GHz. During experiments, multiple Pis communicate with a remote server through a router. The router serves as the fog layer that bridges the end device and the server. Each Pi was pre-loaded with the default Raspbian-Jesse operating system for the end device implementations. A wifi router was used to be the fog layer. As for the cloud server, a remote
A server that runs on an Ubuntu 16.04 operating system was used. For hardware specifications, the server used an Intel Xeon Processor E5-2640 that had 6 cores running at 2.50 GHz each. The Pis and the server contain Python with version 3.6 to cater to the software needed by the libraries in the design. An STM32 NUCLEO-64 board with an attached X-NUCLEO-CCA02M1 expansion board to serve as a digital MEMS (Micro-Electrical-Mechanical System) microphone to record the sound. Using a digital microphone made it easier for the data that was read in by the Pi to be processed. In addition, any wireless transmission is conducted using a Socket Transmission Control Protocol (TCP).

Raspberry Pi 3s devices were selected due to their modularity and affiliation towards rapid prototyping. The proposed framework requires multiple sensing devices that are identical in functionality. Pis can also be easily reprogrammed and adapted in situations when immediate changes to any of the scripts are needed. It is a strong development tool that allows the incorporation other technologies such as the STM32 NUCLEO-64 board without any difficulties due to its flexibility. Also, Pis are very low powered, which is an advantage in testing when it comes to requiring multiple sensing devices that are running at the same time.

3.1.2 Sound Recorder

To implement the urban sound application within the framework, sound needs to be recorded for the classifier. The sound is recorded through a digital microphone connected to the Pi that is controlled by a programmed Python script. This script uses the microphone wired through a USB (Universal Serial Bus) on the Pi and a Python library called PySound. There was no need for any initial data processing because the microphone was digital. The microphone setup used for the STM32 NUCLEO-64 board and the X-NUCLEO-CCA02M1
were their default configurations (i.e. 16 kHz sampling rate, single-channel, and 16-bit resolution). The script specifies a variable recording length to help in the classifier later on. It calls the PyAudio library to initialize the recording by creating an audio stream that uses the specifications of the digital microphone.

During recording, the Pi reads in frames of data from the microphone. The selected frame size was 4096 bytes per frame. Each frame is appended to a list by the script. Upon reaching the recording length, the library closes the stream and a file handler stores the data into a WAV file. Afterwards, depending on the configuration, the sound file is either sent to the server for classification or be classified within Pi and have the results sent to the server.

3.1.3 Sound Classifier

The sound classifier implementation consists of 3 sections: the dataset, the feature extraction process, and the learning architecture.

3.1.3.1 Dataset

The classifier used in the application implemented in the proposed framework makes use of an UrbanSound8K dataset [10]. This dataset is a group of sound files compiled from different sound types in an urban setting. The original UrbanSound dataset is a collection of 1302 full-length recordings. Each sound, labelled according to the sound occurrence and salience annotations, varied from a duration of 1-2 s to over 30 s. The 8K version of the dataset was a modified version of the original. This version splits the 1302 recordings into 4-second clips resulting in 8732 sound excerpts. Listening tests were conducted by Salamon [10] to find out the most optimal sound format to achieve the best identification accuracy. The results indicated that 4 seconds was the best clip duration, which yielded an accuracy of 82%. The UrbanSound8K dataset
consists of 10 low-level classes: air conditioner, car horn, children playing, dog barking, drilling, engine idling, gunshot, jackhammer, siren and street music. The classifier was based on the design advised by Salamon [10]. Also, in terms of the dataset, he arranged the sound clips in 10 folders. Each folder contained around 850 clips upon average. These folders were used to train the model outlined in this work.

3.1.3.2 Feature Extraction

Feature extraction is a part of pre-processing data before it gets used to train and test the classifier model [34]. This process obtains these features as vectors that represent specific aspects of the sound clip [35]. However, data has become too large for conventional processing. Fortunately, even though data gets too big, it tends to be redundant. In that case, the data can be transformed into a representative set of features. Converting this data from its original form into a smaller set is called feature extraction. The thesis uses a programming library called Librosa to extract the features from the obtained urban sounds. This library loads in the dataset and converts them into feature maps. The conversion is carried out using feature extraction methods provided by Librosa. Also, this work consists of 5 feature extraction sources: MFCC, Chromagram, Mel-scaled Spectrogram, Spectral Contrast, and Tonnetz. The following is a brief and general discussion of the methods of extracting the features that were executed by this library in the framework.

1. **MFCC and Mel-scaled Spectrogram.** - It is a technique that extracts features in the cepstral domain [34]. The cepstral domain is a mathematical algorithm that obtains the envelope of the spectrum in the logarithm domain. First, the acoustic signal is converted by the function into a digital signal. This signal represents each level of the signal at every discrete
time step. Next, the data is filtered, which maximizes the magnitude of the highest frequencies in comparison to its lower counterparts. This procedure is called pre-emphasis, which is performed to improve the overall signal-noise ratio of the signal.

After pre-emphasis, the sound samples undergo a short-time Fourier Transform (STFT). In this procedure, the signal is framed and windowed by the STFT function. After removing any discontinuities from the resulting output of STFT, it undergoes Fast Fourier Transform (FFT). The result is a spectrogram, which is a visualization of the signal in the frequency domain. Then, a Mel-Scale filter is applied to the signal, which yields a Mel-scale spectrogram. The features from this signal were obtained by relating the perceived frequency to actual frequency [36].

The ability to distinguish pitches and relate them to real-world sounds can help the neural network to even classify digital sounds from recordings or radio. The MFCC variation of this signal was still obtained to widen the feature scope of the classifier. Before obtaining the cepstral coefficients, the spectrum undergoes a logarithmic process. It is a process where the logarithm is taken from the powers of each Mel frequency. Then, the resulting log powers undergo discrete cosine transform (DCT). DCT is a type of fast algorithm that transforms the domain of a set of signals from times to frequency [37]. The resulting frequency spectrum is the cepstral coefficients.

2. **Chromagram.** - It is a technique that visualizes pitches as energy distributions. Also known as Harmonic Pitch Class Profile, chromagrams are spectrums that distinguish different pitch classes. There are multiple ways of obtaining the chroma spectrum. In this work, Librosa uses the STFT approach. This approach starts with the sound input visualized in the frequency domain using STFT. Then the signal power is calculated
using a logarithmic scale, which results in frequency bins. These bins are eventually added together mathematically. Now expressed as energies, the values are warped onto one octave to fit the salience of 12 pitch classes [38].

3. **Spectral Contrast.** - It is used to extract musical features by considering the difference between the spectral peaks and valleys in a spectrum. These peaks correspond to harmonic components, while the valleys are non-harmonic components such as noises. The computation process begins by expressing the sound input into the frequency domain via FFT. Then, the Octave Scale Filters are applied to the output, which will divide it into sub-bands. The strength of spectral peaks, valleys and their differences in each sub-band are then estimated. Next, the resulting spectrum is translated in the Log domain. Finally, with the use of the Karhunen-Loeve transform, the features are mapped in orthogonal space [39].

4. **Tonnetz.** - It is a harmonic network representation of pitch intervals [40]. The first stage is the Constant-Q spectral analysis, which is a derived logarithmic frequency. These spectrum vectors are then framed and windowed into 12-bits. Then, the resulting stream of bits is transformed into Tonal Centroids in the 6-D space based on harmonic associations between pitches. The sequence of tonal centroid vectors is modulated using the overall rate of change of the smoothed tone and the distance between the vectors. Finally, the features from the resulting peaks in the signal are obtained.
3.1.3.3 Neural Network

After extracting the features, the scripts compile them into a sound map and then fed into an artificial neural network. An artificial neural network is a computing system that is widely used by programs for classification [41]. It is a biological neural behaviour that has been observed and applied through computational algorithms. Neural networks come in different shapes and forms based on their intended learning flow. Each network models a general system that decides on an output based on its given input. A neural network can consist of three types of layers: input layers, output layers, and hidden layers. First, the input layer contains all the possible features or parameter values. Neural networks use these values for decision-making. Next, the output layer contains all the possible outcomes of the model given the possible inputs. Lastly, the hidden layer consists of a variable number of layers. These layers decide the different ways an input can take to reach an output. The network is trained programmatically using a given dataset.

This thesis uses a neural network that is programmed and built with Tensorflow. By having the model observe the behaviour of the data during different instances, it can deduce patterns and relationships between inputs that eventually create the neural network [42]. The classifier in this work uses a basic neural network that has 2 hidden layers. Also, each one contains 280 and 300 nodes, respectively. Tensorflow was used along with three folders from the Urbansound8K dataset to train the neural network. Each folder consists of 870 clips that were 4 seconds long as described by Salamon [10]. Also, all the clips were resampled as 22050 Hz sound files for feature extraction. Next, the training scripts were reconfigured to have variable training epochs and learning rate to obtain the highest accuracy. Then, the dataset was split 70-30, where 70% of the sound files were used to train the classifier. The remaining 30%
was used to test and verify the classifier afterwards. After training and testing the classifier under multiple iterations, the best accuracy was 85%. This result came from a training epoch of 5000 and a learning rate of 0.1.

Urban sound classifiers are sensing applications that analyzes the different sounds within a city area [10]. The collected data is categorized based on a defined set of sound types [30]. It was chosen because it was an application that requires fog computing as an architecture. It requires multiple sensing nodes resulting in large volumes of incoming and outgoing data. To integrate this application to the proposed framework, real-time data processing and big data management was needed. Ideally, using the fog should be capable of these due to its load balancing and node management features [19] [43]. However, as mentioned in the previous section, fog-based IoT network can still run into issues in high power consumption in end devices and data management in servers. These could prevent the framework from being a viable option for this application. This work proposes active low and high power states and process reallocation to address these concerns. The next section discusses how the proposed framework is tested.

3.1.4 Testbed Setup

Three tests were conducted to obtain the three metrics that are the focus of this work. The tests were to measure the end device power consumption, application runtime and the server latency. The testbed takes these three tests and breaks them into two sections: the power consumption and the runtime test, and the test for latency. For the first test, a script was executed to sample a 10 second sound clip through the microphone. After finishing the recording, the program saves the sound clip as a WAV file. Next is another script that reads in the WAV file and extracts its features. An urban sound classifier model
defined in the neural network section then uses these features. Once the model returns the results, the program then exits.

Firstly, the testbed measures the power consumption of the sensing devices by using the Monsoon power monitor. The sensing devices in this case are the Raspberry Pi connected to the microphone. This test allows the observation of any significant differences between the two practices in terms of their effective execution. This power monitor will be used to directly power the Pis to measure any changes in power consumption while running the program. By taking note of when these changes occur and aligning them with the recorded timestamps, the testbed can be used to map out the overall power usage of the device during testing. Next, the testbed measures the runtime by adding timestamps during every major procedure within the program. In measuring the total runtime of the application, each end device will wait for a confirmation from the server that the results have been successfully obtained and stored by the classifier.

The next test takes each implementation and measures its latency by measuring the times it takes for the Pi to transmit all of the data to the server. Each Pi will be sending packets to the server at the same time. A round-robin type of scheduling is used to manage each Pi to obtain each of the data they are transmitting sequentially. Each transmission is timestamped to check when a Pi starts a sending and when it ends. This design made it possible to measure the latency of each Pi and pool them together to obtain an average value representative of each configuration.

3.2 Chapter Summary

This chapter presented an overview of the proposed fog-based IoT framework. It was then followed by the introduction of the experimental setup. Each of its aspects was presented by enumerating each component and explaining its
significance to the thesis proposal. The last section of the chapter takes the
design and creates a testing set up for the incoming experiments. The following
chapter takes this setup to investigate the feasibility of the framework.
Chapter 4

Testing and Evaluation

This chapter presented an overview of the proposed fog-based IoT framework. It was then followed by the introduction of the experimental setup. Each of its aspects was presented by enumerating each component and explaining its significance to the thesis proposal. The last section of the chapter takes the design and creates testing setups for the incoming experiments. The following chapter takes this setup to investigate the feasibility of the framework.

4.1 Testing Overview

Initially, the tests conducted in this thesis takes two configurations. Configuration A has the classifier programmed within the Pi while B has the classifier inside the server. These configurations were labelled configurations A and B, respectively. Configuration A was set up by programming the classifier into the Raspberry pi. First, the operation flow starts with the microphone recording the sound for 10 seconds. Next, the program will save the sound data into a file that is read by the classifier script. The classifier code will then extract a feature map from it. Then, the trained model will classify the results of the
Lastly, the results of the classifier are sent by the devices to be stored by the server.

Configuration B was set up by programming the classifier into the server to be executed in the cloud. The recording is the same with configuration A. However, instead of classifying the sound file in the Pi, it will be sent wirelessly via packets to the server. Using wireless socket communication, the Raspberry Pi will establish a connection to the server. Once a connection was established, the Pi will send the sound file to the server as packets. The server will then reconstruct this information into a workable file. Finally, the program will classify the sound and store the results into the database.

It was discussed in this thesis that high power consumption among end devices and data management constraints in the cloud server were potential issues in a fog-based IoT framework. As a result, active low and high power states and process reallocation were proposed as solutions. The framework can be made to only run parts of the architecture by defining low and high power states. So, the Pis were programmed to use only the necessary peripherals for each process.

There are two main sections in the design; sound recorder and sound classifier. It was determined which of these sections can be moved between the end device and server to know which parts can conserve energy. The recording process needs to stay within the Raspberry Pi because it is what drives the digital microphone. However, the sound classifier can either be placed on the server or remain within the Pi. A program was created that can be divided into these sections so that the processes could be reallocated effectively.
4.1.1 Preliminary Power Consumption Tests

Both initial configurations were executed for 20 iterations. A sample setup of the test is in fig. 4.1. The first metric measured was the power consumption of each iteration. Figure 4.2 shows a comparison between configurations A and B in terms of their respective iterations. Through this graph, it could be observed that most of the points from configuration A is higher than that of configuration B. The average power consumption of configurations A and B were 1852.00 mW and 1830.54 mW, respectively. Configuration B has a lower power consumption than A by a value of 21.46 mW. However, this value is not large enough to be considered a significant difference. Therefore in terms of power consumption, configuration A and configuration B are relatively the same.

Another test was added to check the distribution of the measured power consumption within each configuration. In this test, the effects of wireless transmission to the power consumption of the framework were measured by investigating the sound recording section. The outcome of the test within 20 iterations was plotted in fig. 4.3. Averaging the results showed a power difference of 127.54 mW. For every iteration, the first configuration transmits around 130 kilobytes (KB) of data while the other configuration only transmits 4 bytes (B) being the size of a single integer. As a result, data size becomes a significant variable for wireless transmission. The first configuration sends
a whole sound file to the server for classification. The second configuration only sends the results of the classifier. These two configurations may have yielded the same power consumption, but it was because of different reasons; processing power and data size. The first configuration takes all the processing load by executing both recording and classifying portions of the framework. However, the second configuration demands more power within the wireless transmission to move the sound data from the end device to the server. Having smaller transmitted data sizes saves energy, but it is cancelled out by processing demands of classifying at the edge of the network.

Also, the runtime of the program was measured for 20 iterations. To make sure that the observations between the runtime of the two configurations are
Figure 4.3: Difference in power consumption with and without sending the data to the server after recording.

more distinct, the test was broken down into two runtime measurements: the recording phase and the classification phase. As mentioned in the setup, the end device will only record the runtime once the server confirms that all the server-side processes (i.e. Classification for configuration B or storage for configuration A) are complete.

Figure 4.4 shows a graphical representation of the runtime measurements for each phase. As observed that the recording phase for both configurations was the same. However, the time it takes to complete the classification phase has configuration B taking 6.02 seconds while configuration A takes 47.36 seconds. This significant difference shows that configuration B is better than configuration A in terms of runtime by being 41.31 seconds faster overall. The
calculated average runtimes were 57.77 and 6.42 seconds for configurations A and B, respectively. By taking the results of the power measurements, this time difference points out that the disparity in power consumption could be due to the placement of the classifying process. This difference points out that the measured power consumption is due to the placement of the classifying process. Also, the runtime results point out the disparity in the execution times of the first two configurations. There is no significant difference in power consumption between the configurations. The power drawn by running the classification in the end device is offset by the power required to transmit the sound files to the cloud for classification. However, the time differences points out at advantage of having the classifier in the server. It improves the overall time it takes to carry out the whole sound classification process.
4.1.2 Preliminary Latency Tests

The next concern is the server data management. Once the number of nodes within the framework increase, will it be able to handle the incoming data? This test took the three configurations used in the power consumption test and verifies their viability in data management. Latency was used to measure each of their effectiveness in managing data. Latency is the time it takes a data packet to finish transmitting from the device to the server. The experimental setup implements a star topology with multiple nodes loaded with the same configuration. Each node sends data to the server. The server then takes each packet and measures the time between the beginning and the end of the transmission. A sample setup of the latency test is shown by fig. 4.5.

![Latency Test Setup](image)

**Figure 4.5:** Latency test setup.

The test setup uses a round-robin type of schedule. All nodes are listed based on their given IP address. The implemented scheduler goes through this list and allows the selected device connection one at a time. The latency of each configuration was tested by creating networks that had a varying number of nodes. Each device was programmed to transmit data 10 times to the
Figure 4.6: Latency comparison between configurations A and B

server. Figure 4.6 shows the results of the latency test. It can be observed that configuration B significantly slows down as the number of Pis increases. This observation exposes the inability of the configuration to manage growing networks. On the other hand, configuration A shows that it can manage larger systems through its steady graph. However, based on the previous test, configuration A is significantly slower in terms of runtime than B.

4.1.3 Introduction of the Third Configuration

The previous tests results were taken into consideration, which resulted in a third configuration. The tests in power revealed that wireless transmissions affected the power consumption of the framework equally as the classifier being in the end device. In addition, configuration A was significantly slower than B due to the classifier placement. As a result, the slowest section of configuration A was taken and some of its parts were offloaded to the server. The classifier is
composed of two subsections; feature extraction and the actual classification. Before a sound file is classified, it’s feature vectors are first extracted. This section was divided into two and having the feature extraction process moved to the end device. In theory, this could improve some aspects of the design by requiring less power and reducing latency by transmitting fewer data. This reallocation of processes reduced the size of the transmitted data from around 130 KB to a constant 1.6 KB. This consistency is due to the feature vectors predefined to have a dimension for every sound clip. With this change, the data transmission has become more constant and stable in case the design required longer recording times. Also, due to the effects of transmitting data wirelessly to the power consumption of the end device, active low and high states were implemented towards this component. Instead of having it on all the time, the Pis were programmed to keep it off unless there was a need to either transmit sensed data wirelessly or communicate with the server. Therefore, a new configuration was created to have the feature extraction conducted in the Pi and the actual classification in the server. Instead of sending a whole sound file to the server, the Pi sends a feature map. Also, the Pi will only use its wireless component when it needs to send data or communicate with the server, otherwise, it is kept off. Figure 4.7 presents the logical flow of the resulting hybrid setup.

The new configuration, labelled as C, was tested similar to A and B. Table 4.1 shows a summary of the power and runtime test comparing the results. Configuration C yielded an average power consumption of 1786.86 mW being lower than both A and B. Also, the average runtime was 53.02 seconds. This measurement shows a potential drawback to this configuration by being only slightly faster than A, but still significantly slower than B. Lastly, latency was measured. Table 4.2 shows a summary of the latency test comparing the
Figure 4.7: Hybrid setup logic flow.

results. Configuration C was observed to be steadier than B in terms of latency increase over network growth. Also, its values and trend appear to be the same as A.

As observed from the yielded values, C is the best in terms of power consumption. In terms of runtime, B is the best leaving the other two significantly slower. However, in the last test, C and A show their ability to handle growing networks by maintaining relatively low latencies at higher network traffic while
### Table 4.1: Testing results for power consumption and runtime.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Average Power Consumption (mW)</th>
<th>Average Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1852.00</td>
<td>57.77</td>
</tr>
<tr>
<td>B</td>
<td>1830.54</td>
<td>16.42</td>
</tr>
<tr>
<td>C</td>
<td>1786.86</td>
<td>53.02</td>
</tr>
</tbody>
</table>

### Table 4.2: Testing results for latency.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Average Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>A</td>
<td>0.6</td>
</tr>
<tr>
<td>B</td>
<td>9.7</td>
</tr>
<tr>
<td>C</td>
<td>1.7</td>
</tr>
</tbody>
</table>

B does not. Though B can be faster, it is not scalable since it cannot handle an increase in sensing nodes. Therefore, run time does not indicate any advantages of B over the other configurations. This observation puts more weight on power consumption and latency than runtime. Each test result was rated for A, B, and C by using a point system: 1 point for third, 2 for the second, and 3 for first. The configuration that had the highest number of points is arguably the best. For the runtime and power consumption, the scoring was based on the obtained values. Latency, on the other hand, was on the effectiveness of each configuration as the Pi’s increased in number. Table 4.3 shows the ranking of each metric for the data collected. By consulting the data and the results, the best configuration is C, which is a balanced combination of both the end device and the cloud server.
### Table 4.3: Configuration scoring.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Power consumption</th>
<th>Runtime</th>
<th>Latency</th>
<th>Tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

4.2 Chapter Summary

This chapter carried out tests that investigated the most optimal configuration of the proposed framework. It initially presented two configurations as benchmarks of the designs. After conducting preliminary tests, a third iteration was created to reflect improvements towards the previous two. Upon testing, it presents its results and a comparison of this iteration to the previous benchmarks. The results show that the third configuration is an overall improvement of the other two. However, to widen the scope of the framework, the following chapter presents derivations of the framework to prove its ability to be used in other contributions that focus on other types of data classification.
Chapter 5

Relevant Applications

This chapter enumerates related contributions that are derivations of the classification framework presented by this thesis. Each takes the main concepts of fog-based IoT networks and implements it under different applications. There are two applications discussed. The first application presents an alternate use for the urban sound classifier by converting it into a noise classifier while using the same framework. The second application makes use of a different dataset. The dataset is a Google speech dataset that will drive a remote health monitoring system with the use of speech recognition. The two contributions are discussed below; A full documentation on the noise classifier is presented in [16], while the remote patient monitor is presented in [17].

5.1 Urban Noise Categorization

The unavoidable increase of population in metropolitan areas on a world-wide scale is continuously contributing towards noise levels with a direct impact on noise pollution. Consequently, an increase in noise pollution affects negatively
the living quality standards for residents within metropolitan areas. Therefore, it is of critical importance for metropolitan authorities to pinpoint areas in which noise levels are high and frequent such as to empirically establish policies to improve the general well-being of residents. Nonetheless, dispatching personnel for data collection over particular periods invokes high costs and it has proven to be impractical. Hence, it is vital to propose solutions that utilize the envisioned smart city paradigm through effective automation and smart sensing. One of the services provided by the smart city platform is Urban sound classification. An implementation of this scope is made possible due to IoT networks and their ability to transmit data wirelessly within large areas. This work takes the classification and network aspects of smart cities and combines it to create a proposed design as a solution to the lack of a platform for noise classification. The proposed design deploys recording devices that are made up of Raspberry Pi 3s and a digital microphone. Programmed inside each Pi is the customized neural network for classifying the noise. Each Pi records the sound within its area and classifies its sound levels. The results from each classification are then sent to the server for further analysis. Through the herein reported findings it could be argued in the fare of a cheap, time and energy-efficient platform that can adequately confront the pragmatic requirements for noise classification within an urban environment.

5.1.1 System Overview

In this work, the goal is to classify the noise within a city or urban area. To be able to build a model, the different categories must first be defined. Which of these types of sounds are considered noises? For testing purposes, the design uses a predefined dataset. This work uses the UrbanSound8K database. This dataset makes use of different urban-related sounds taken from field recordings
Table 5.1: Noise categorization of UrbanSound8K dataset.

<table>
<thead>
<tr>
<th>Sound type</th>
<th>Category</th>
<th>Number of soundclips</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air conditioner</td>
<td>Regular sound</td>
<td>1000</td>
</tr>
<tr>
<td>Car horn</td>
<td>Noise</td>
<td>429</td>
</tr>
<tr>
<td>Children playing</td>
<td>Regular sound</td>
<td>1000</td>
</tr>
<tr>
<td>Dog barking</td>
<td>Regular sound</td>
<td>1000</td>
</tr>
<tr>
<td>Drilling</td>
<td>Noise</td>
<td>1000</td>
</tr>
<tr>
<td>Engine idling</td>
<td>Regular sound</td>
<td>1000</td>
</tr>
<tr>
<td>Gun shot</td>
<td>Noise</td>
<td>374</td>
</tr>
<tr>
<td>Jackhammer</td>
<td>Noise</td>
<td>1000</td>
</tr>
<tr>
<td>Police siren</td>
<td>Noise</td>
<td>929</td>
</tr>
<tr>
<td>Street music</td>
<td>Regular sound</td>
<td>1000</td>
</tr>
</tbody>
</table>

uploaded to www.freesound.com. The category of these sound files ranges from mechanical (i.e. Drilling, Air condition, Gunshot) to natural (Dog barking, Children playing, People singing) [10].

Table 5.1 presents the noise categorization of the different types of sounds from this dataset. This table will be used in this work to simplify the classification process to minimize the number of variables to focus on building the system as a whole. The number of sound clips under each sound type within the dataset was included in the table to show the size of the training model used. It was previously mentioned that data needs to be formatted before it is used to train or test the classifier. This procedure is called feature extraction. It is the process of mathematically or algorithmically acquiring related information from data [34]. Each extraction method focuses on a specific aspect of the sound. In this application, a combination of Mel Frequency Cepstral Coefficients and Mel-Scale Spectogram was used[36].

5.1.2 Architecture Design

This application aims to bring out the strengths of IoT networks in the proposed design. To understand the strengths of this design, a benchmark needs to be
established for comparison. If the IoT aspect was taken away, a design is created that makes use of local computers to collect sound data. Without the wireless capabilities of IoT networks, the only alternative for the medium of transmission are wired connections. After collecting the sound data, it is processed and sent to a server. The resulting design is named Design A. Figure 5.1 shows the network setup for this design.

Design B, which is the proposed design, starts with each end device. The end device, in this case, is a Raspberry Pi 3. Raspberry Pis were used because they are easy to program and reconfigure for prototyping. Also, images of the same system can be replicated across all Pis that are being used. This allows fast prototyping and modularity as well as easier testing. Attached to each Raspberry Pi 3 is a digital microphone for sound collection. Another advantage of the Pi is its ability to cater to a diverse selection of add-ons and peripherals. Digital microphones are easier to integrate into a prototype. Most digital mics are plug and play which makes them easier to manage for designs.
that require multiple instances of the recording setup. Also, sound quality is less of a concern during the earlier phases of the platform and the focus is more on having a working overall system. Therefore, the quality of the sound from the digital mic being used is less significant as of now. The Pi is programmed to enable each microphone to record sound for a specified time. The sound is then saved in a file for easier data transfer. The classifier which is also within the Pi will then take this sound file classifies it. A python script is used to run the classifier by taking the sound file and feeding it into the model. During the classification process, the file will undergo feature extraction and its categorization of noise. Using IoT networks as the medium, the results of the classification is sent to the server for storage and further analysis. Figure 5.2 shows the ideal network setup of the platform. By comparing the two designs, the following advantages IoT networks give to the proposed design are listed as;
1. **Cheaper** - Design B provides a more scalable design by using Pis and transmitting data wirelessly. Due to the number of sound collecting devices, deploying Design A will be costly. It will also not be cost-effective due to the intended placement of the device. The proposed platform requires the devices to be placed around the city where natural forces can weather and damage it. A CPU costs more to replace than a Pi. Overall, A has cheaper parts that are easier to replace and redeploy.

2. **Faster** - Without the IoT network, it will be harder for the data to be sent to the server effectively. Using a wired system creates interference in data transmission. Also, deploying the system is faster overall. Wiring design A requires less effort when setting up the platform. As a result, there is less downtime during maintenances.

3. **Low powered** - In terms of power consumption, Pis are significantly lower while still being able to do its function. Compared to design B that uses CPUs to collect data, the amount of power required to run that platform on a citywide scope might be too much.

Looking at the two designs, design B is cheaper, faster and more efficient overall. Design A is built on the other options that take away the main features in design B. This analysis shows how much IoT networks contribute to the overall scalability of any large scale system.

**5.1.3 Evaluation**

Experiments were conducted to investigate the difference between conducting the classification locally within the Pi or sending the sound data and have the server classify the data. Each configuration was tested by measuring the power consumption of the Pi as it executes the system.
Figure 5.3: Power consumption of local and server configurations at 20 iterations.

Figure 5.3 shows a plot of the power consumption of each configuration for 20 iterations of the system. The first configuration had the classification executed within the Pi. The results yielded an average power consumption of 1853.00 mW. The second configuration moves the classifier to the server. This configuration yielded an average power consumption of 1830.84 mW. Based on the results, the classifying the sound data through the server is better than doing it locally by 22.16 mW. This value shows no difference between the two configurations when looking at the scope of the design. Therefore a second test was conducted.

The next test was to check if which configuration performs better under the intended application of the design. The design is being proposed to collect sound data within a city. Therefore, multiple instances of the Pi will be simultaneously collecting and eventually sending data to the server.
Latency is the metric to check the performance of each configuration as the number of Pis connected to the server increases. Each configuration is tested using 4, 8 and 12 Pi attempting to send data to the server. Each Pi connects to the server wirelessly through socket transmission. Data will be sent from the Pis in the form of packets that will be received by the server. To obtain the latency, the time it takes from the first to the last packet of the transmission to reach the server is measured. Figure 5.4 shows the result of the latency test. Based on the results, classifying the sound within the server will not be able to handle the platform. The resulting graph shows that as the number of Pis connecting to the server becomes 12, its latency shoots up to 300ms. On the other hand, classifying locally maintains a reasonably low latency during all three test iterations.

Both configurations have relatively similar power consumptions. However, in terms of latency, having the classification done in the server is not ideal.
Therefore, to be able to implement the design successfully, the classification needs to be done within each Pi.

With the increase in population density, noise levels and consequently noise pollution affecting the well-being of residents is increasing. Moreover, the collection of noise level data by authorities such as to identify hot spots of high noise pollution is a cumbersome task. It is therefore of paramount importance to facilitate mechanisms that collect and further process data to intelligently identify noise pollution hubs in an efficient, effective and lightweight fashion. The herein reported work demonstrates the efficacy of edge-based processing for classifying urban noise levels under an IoT-enabled platform. With the use of Raspberry Pi’s lightweight data collection approach, a basis for distributed and edge-based classification of noise pollution levels can be illustrated. The representative datasets illustrated that the instrumentation of the proposed design promotes low power requirements on distributed fashions whereas latency is increased in the case where classification is undertaken centrally. Thus, through a distributed, edge-based platform a means of facilitating the platform design of future classification schemes in urban settings that require close-to-real-time analytics is created.

The classifier used in this work can help create a representative description of each area or section of the city. This feature can allow cities to analyze each area and act accordingly to the data collected. The proposed platform makes use of a combination of an urban sound classifier and an IoT network. This design provides a cheap, fast, and low powered means of collecting and classifying data. Based on the experiments, there is no significant difference between the power consumption of executing the classification within the Pi or using the server. However, in terms of latency, the server will not be able to
handle the proposed design if it also carries out the classification. Therefore, the classifier needs to be run within the Pis.

5.2 Remote Patient Monitoring

Health care resources have started to become scarce due to their increase in demand. Hospitals have begun to run out of space, forcing them to deny admission of patients. Remote Patient Monitoring (RPM) has the potential to help citizens who suffer from chronic diseases and provide environments where easy to access healthcare is available. RPM allows people to receive the same amount of care without having difficulties in finding a spot at a hospital ward. However, some roadblocks end up preventing RPM from being implemented by more healthcare providers. Data integrity, user privacy and high power consumption are some of these concerns. With data transmission and transaction, privacy and confidentiality have always been an issue. High power consumption is a concern due to RPM’s demand for continuous data collection. This work proposes a framework that reinforces the RPM system to address these concerns. The design not only allows better data filtering for privacy but also a more responsive system with the use of controlled surveillance and speech recognition. Overall, this framework provides an opportunity for RPMs to be a viable implementation for healthcare providers all over the world.

5.2.1 IoT in RPM

IoT-based RPM systems are wireless network services that allow a patient to be observed by a medical centre without having to be physically present in a clinic [44]. Through remote patient monitoring, health care providers can now check on their patient’s conditions without requiring them to make appointments. This feature is made possible with the use of IoT. It is used
to connect the devices needed for monitoring the patient within a wireless network [45]. Distance is no longer a constraint for health care providers since IoT has made it possible for data to be obtained over a wireless network. The monitoring technology has even evolved to incorporate common devices such as smartphones [46] in their services. IoT has enabled RPM to be available to a larger demographic by making it more convenient not only for the patient but also for the health care providers. Patients are no longer required to travel to reach their family doctor, while medical centres no longer need to worry about resources and costs when checking on their patients [47].

5.2.1.1 Issues with current applications

IoT-based RPMs are very beneficial in expanding the ability of medical centres of ensuring the health of their patients. However, this system comes with a few caveats.

1. Data Integrity - The integrity of the data is significant to RPMs. RPMs need to give healthcare providers accurate information to properly detect potential anomalies in a patient’s health. However, there is currently no way of maintaining high integrity and accuracy of the data that clinics receive from their devices in the current state of RPMs [48]. Data needs to be secure. Its integrity is lost if its reported values are different from its original value. The wireless devices that are used to collect and transmit data are often deployed in hostile environments with low bandwidth [49]. As a result, insecure channels are created, which may lead to maliciously modified data. For instance, an attack results in modifying data to change from reporting an anomaly to a normal data return. This loss in data integrity leads to the medical centre not being able to react accordingly, which would endanger the patient’s life. Adding a component that can further verify the credibility of received data allows the
medical centres to confirm and react accordingly towards an emergency or false alarm.

2. Patient Privacy - One of the biggest issues for patients who are presented with monitoring services is their privacy [50]. Patients find discomfort in disclosing their information due to this issue. As a result, the likelihood of medical centres of implementing RPM systems is low [51]. Another aspect that contributes to dissuading patients from investing in the service is due to its wireless component. Wirelessly transmitting data over long distances could result in various security attacks[52]. Healthcare services are responsible for keeping the data that is obtained from a patient’s private and confidential. A system that is left insecure can easily result in stolen patient data. RPM or wireless health monitoring systems need to maintain an adequate level of security towards their patient’s data [53]. Proper data filtering would be able to prevent such attacks from being able to steal important data.

3. Device Power Consumption - Medical centres use wireless devices to monitor their patient’s health long term [54]. These devices are programmed to transmit data continuously over the network. However, continuous transmission of data can result in higher power consumption in devices. Most devices are usually handheld or worn, which makes them battery dependent. The biggest contributor to power lost in batteries is from its continuous transmission of data [46]. These wearable devices must stay on to make sure that a patient is fully monitored by their medical service providers. Sensors also contribute to the power loss as well. However, it is a component that should be kept on to continuously obtain data from the patient. On the other hand, the wireless component of the wearable device does not need to be on all of the time. An effective optimization technique could lead to reducing its significance in the power consumption of a device.
5.2.2 System Overview

The proposed framework aims to address the concerns mentioned in the previous section. Mainly, it addresses issues with data integrity and privacy as well as improving its overall power consumption. The framework is a system that takes the sensing abilities of devices partnered with neural networks to allow a more interactive response system for patients in times of emergencies. Most RPMs have patient data being streamed all the time to the doctors and physicians. Such a design resulted in an issue with privacy and patient confidentiality. RPMs need to monitor their patients’ vitals 24/7 to catch any emergency. However, this results in high power consumption and exposes patient data. The proposed framework is shown in Fig. 5.5

The proposed framework reduces these concerns by integrating a system that records and streams data only when given the authorization by the system during emergencies. The idea is to have a system that will interact with the
patient during an emergency. If the patient permits or does not respond for a given time frame, then a video feed revealing the patient’s current status will be sent to the clinic. It is then up to the clinic to decide the best action during the presented situation.

This system provides proper controlled surveillance for the patient. Also, it maintains privacy for its users while saving power by only recording and transmitting data if prompted. The system starts by staying in low powered status or sleep mode. Most of its functionality is turned off and will only activate once triggered. The system serves as a filter and will only send the data to the clinic if any anomaly or significant change is detected. This design creates a more confidential system that only cares about the well-being of the patient without getting too much data or using up too much power. Upon detection of an anomaly, the system decides on its severity. If the system determines an emergency, a listening device placed within the household of the patient interacts with the user. It then asks the user if medical attention is needed or if the trigger is just an alarm.

Overall, this creates a framework that improves RPM systems in terms of maintaining the data integrity and privacy for each patient while keeping the system energy efficient.

5.2.3 Design

This framework is composed of three major sections:

1. Sound recording,
2. Surveillance capture and,
With the presence of an already existing health monitor, this framework serves as a filter between patient and healthcare service. It only reports data that is needed by the patient for treatment and maintenance.

5.2.4 System components

A combination of hardware and software components was used to build the proposed framework. In terms of hardware, the system uses 3 Raspberry Pis and an NVIDIA Jetson TX1 developer kit. The Raspberry Pis are the sensing devices that will be placed strategically around the room. They record the patient’s response based on the health monitor data. They can also filter out the data being sent by the monitor to the clinic. The Raspberry Pis were loaded with a Raspbian operating system. The Jetson board already came with a preloaded with an Ubuntu 16.04 operating system, which was used in this prototype.

The sound is recorded with the use of an STM32 NUCLEO-64 board with an attached X-NUCLEO-CCA02M1 expansion board to serve as a digital MEMS (Micro-Electrical-Mechanical System) microphone. This microphone is connected to the Pi and controlled using a python script. The code used in recording the sound uses the PyAudio library. This library was used to configure recording stream to match the default configurations of the MEMS microphone that was used. By default, the microphone records with a sampling rate of 16 kHz and a 16 bit sound resolution. A patient can be monitored effectively by having the 3 Raspberry Pis placed on each corner. Each Pi will wirelessly transmit the sound file to the Jetson board for processing.

The Jetson serves as the central processing hub for the system due to its computing capabilities via its built-in GPU. Before the sound gets classified by the model, a script will merge the sound clips into one waveform. This classifier
is a Convolutional Neural Network (CNN). It was the selected network due to its ability in creating acoustic models [55]. CNN is also known as a powerful toolkit for speech recognition [56].

The CNN used in this framework is coded using Python. The code makes use of a combination of neural network training libraries; Tensorflow and Keras. It also uses Scipy and Numpy for its sound processing and feature extraction. The CNN makes use of 4 types of layers to build the architecture: Convolutional Layer, Pooling layer, Normalization layer, and Full-Connected or Dense layer.

The neural network was built using blocks. Each block contained; the convolutional layer, pooling layer, and the normalization layer. The input starts by going through 3 of the previously mentioned blocks. Then, the network flattens the output before the dense layer uses it for predictions based on the trained model.

CNN was used due to the intention to train the model to detect more complex terms for future iterations. It uses the Speech Commands Dataset provided by Google. This dataset contains 65000 one-second utterances of 30 short-form words. Each word was recorded from thousands of different people, which completes a comprehensive dataset for speech recognition. However, during the system's preliminary stages, the classifier was only prepared to respond to two keywords out of the 30; “yes” or “no”, while ignoring everything else. As a result, the amount of data being used in training this model is cut down to 4752 clips of the selected keywords. This design provides a simpler model for proof of concept. This application aims to not only cover simple answers but also to create a fully programmable interface that can react to more situations.
5.2.5 Design flow

The general design flow of the system is shown in Fig. 5.6. The system starts with the Pi detecting an anomaly from the data sent by the worn health monitor. Based on the issue, the Pi reports it and does either one of two things; returns to data monitoring or interacts with the patient. If the anomaly is severe, the system asks if the patient needs immediate assistance. The Pi then switches to active mode by turning on its microphone peripheral to listen to the patient’s response. If the patient does not need help, then the system returns to low power mode. However, if the patient does need assistance or does not respond within 15 seconds of the prompt, then the Jetson is prompted to turn on its built-in camera. This camera is used to survey the patient in the...
Figure 5.7: Raspberry Pi operation flow.

room. The video feed is sent to the clinic to show the patient’s current state. The clinic is then expected to respond accordingly to how they evaluate the situation.
In terms of the system prototype, a recording mechanism was created by using the Raspberry Pi and the digital microphone. The module can record sound data and send it to the NVIDIA Jetson board via wireless socket transmission. The system transmits the sound data as packets and is rebuilt as a file once it reaches the Jetson board. A detailed representation of the operation flow of the Raspberry Pi in the design is shown in Fig. 5.7. Also, the trained classifier was placed within the Jetson board. Its job is to process the sound
file once the Raspberry Pis have completed their transmission. Based on the results of the classifier, the board with either do one of two things; turn on the camera or return to low-power mode while waiting for the next transmission. A detailed representation of the operation flow of the Jetson in the design is shown in Fig. 5.8.

5.2.6 Preliminary Results

This section provides any results from the initial prototyping of the system. In terms of the classifier, the model is trained to detect either a keyword of “yes” or “no”. The training of the model was conducted with the Speech Commands Dataset provided by Google.

During training, the number of utterances for the specific words within the dataset were split 70-30; for training and testing respectively. It was reported that the current dataset subset was composed of 4752 clips. Based on the
setup, the training portion used 3327 randomly selected clips out of that subset for training while the remaining 1425 clips were used for the testing. The results where the verification accuracy reaches the intended accuracy are shown in Fig. 5.9. The results where the verification loss approaches the accounted loss are shown in Fig. 5.10. These figures showed that as the number of steps approached an epoch value of 12, the accuracy and loss started to fit within their verification curves. This behaviour indicated the increase in the model’s precision in classifying the intended keywords. The resulting classification accuracy of the trained model is 97% and further supported by a subsequent loss of 3%, which is promising. To examine the feasibility of the system further, it was identified that there is a need to conduct more experiments in different rooms and under different conditions.

Data integrity, user privacy, and high power consumption is a concern for RPM systems. Any healthcare provider who wants to implement an RPM
system should first address these issues. A framework was proposed that can be integrated into an RPM system to build towards minimizing these concerns. This framework is a sensing system that makes use of speech recognition to create an interactive design that caters to patients during emergencies. Also, this system creates a filtering mechanism that could potentially improve the privacy and confidentiality of any patient’s data. The classifier used in this framework is a CNN that yielded an accuracy of 97% and supported by a training loss of 3%.

5.3 Chapter Summary

To prove the feasibility of the framework for other data classification applications, this chapter presents two implementations that use the same concept of the framework proposed by this thesis and creates different derivations. Each derivation was discussed from an overview of the design to some experiments that were conducted. After citing the contributions that were already implemented, the next chapter will highlight ideas and future extensions towards improving certain aspects of the framework.
Chapter 6

Future Work and Extensions

This chapter discusses aspects of the fog-based IoT framework that could be improved through the integration of certain technology. These technologies will be presented along with their motivation as to why and how they could be considered significant in the future.

6.1 A more decentralized approach to Fog-IoT networks using permissioned blockchains and BLE

An IoT network is composed of a server, the gateway, and its end devices [57]. The server is the main data storage of an IoT network. The network caters to its clients by providing services and information. These clients are the end devices that are connected wirelessly to the server. However, these devices can only access the server and its services through a network router or gateway. Being the only medium for a client to reach the server, the gateway becomes a potential bottleneck to any IoT service or application. As a result, one of the main sources of performance constraints in an IoT network to be found within the limitations of its gateway.
An issue that limits a network through the gateway is its data throughput/capacity. The download and upload speeds of a network’s service is often due to the capabilities of its modem. A network needs a gateway design that can manage multiple devices demanding services at the same time. The next issue is security. Some setups that use multiple modems to cover an area runs into interference due to the centralized nature of the network. Due to the distance, a wireless signal needs to cover, a network is more prone to attacks.

The delegation of work is a feasible method that can expand the capacity of a network. By having multiple gateway nodes that can provide the same service, a network is provided with multiple entry points. As a result, a gateway’s overall performance can be increased. Blockchain technology is well known for its decentralized structure [58] [59]. By using this decentralized structure, the network can achieve a level of parallelism that can alleviate the throughput constraints of a standard centralized gateway. Permissioned blockchains can make sure that the information that is shared within the gateway nodes remain exclusive [60]. At the same time, it can provide the same decentralized protocol from a public blockchain that can synchronize each node while making sure that all services are still fully functional and secure [61]. As a result, the network will be able to cater to more devices by having more secure entry points via the multiple gateway nodes.

However, to make sure that these nodes are secure while being able to effectively work together in unison, there is a need for a means of a secure, low-powered and efficient means of communication. Bluetooth Low Energy (BLE) is a protocol that focuses on conserving power when transmitting data over a wireless medium [62]. It can be used to minimize the power drawn by each node while making the communication between the gateway nodes more secure. Continuous communication is key for a decentralized architecture. The
pairing mechanism used by BLE can make it easier to manage. By pairing all of the nodes to each other, a decentralized topology is established [63]. One of the advantages of BLE is that communication cannot be established unless pairing is initiated and accepted [64]. This creates a secure point-to-point medium that is unique to each node pair. Therefore, a Blockchain and BLE driven decentralized gateway architecture could improve the ability of a fog-IoT framework to securely expand its sensing network.

### 6.2 Improving network security using private blockchains and cryptography

With entry points being opened by the network every time a new device requests access, vulnerabilities are introduced to potential attackers [65]. This not only endangers the network, but also any new device that attempts to connect to it. A loosely secure wireless network is vulnerable to any attacker. With everyone’s personal information being stored digitally in a network, an entry point through these vulnerable devices and services poses a threat to the majority of the public. Having a system that regulates what devices can connect to the network can reduce the vulnerabilities and potential attacks that they attract.

A security layer can be placed so that all devices requesting to connect to the network are required to be registered before it is granted access [66]. It can function as a check on the initial handshake of the device to the network to filter through each device before revealing the network to the user. Devices that want to connect to a specific network should be required to register to the server with a unique identification before given access. If the device’s unique identification is registered within the network, then it is granted access. However, without this identification, the device will remain to have denied access to the network.
Every device has a unique serial number. Due to its uniqueness, this number can be used as a key to register the device to the network. These keys can be stored within a secure network instead of using a common database that could be open to tampering exploits. Blockchains provide a secure and immutable means of storing data [58]. However, most blockchain structures have their keys exposed to the public. A more exclusive option is a private or permissioned blockchain. These types of blockchains are often used as secure ledgers to store important information [67]. In this case, the identification of specific devices connected to the network can be stored within the blockchain. However, the storage and transmitting of these keys still need to be done systematically and securely. Cryptography is a way of creating schemes and protocols that ensure secure communication and management of information that is shared wirelessly [68]. Public-key storage and regulation can be introduced as is not new to blockchains as most cryptocurrency implementations such as Bitcoin and Ethereum use this design[60]. By reinforcing the ledger with cryptographic tools, the network can assign unique identification and public key pairs to connecting or registered devices that can be stored within a secure ledger. This ledger can then be used as a reference for future authentication and account verification for any end device that requests access to the network.

6.3 Chapter Summary

This chapter presents potential extensions to the framework as well as how they could improve its quality on certain aspects. It talks about the use of blockchain technology partnered with BLE and cryptography to create a more decentralized approach. Other features of the presented options were discussed to have the ability to improve the security and data management of the framework. These improvements could enhance the design in terms of overall quality
of service. After presenting all the necessary information, the next chapter will conclude the thesis based on the results of the experiment and the supporting details presented by the other chapters.
Chapter 7

Conclusion

Fog computing is a potential candidate in improving the quality of sensing applications in IoT networks. However, it runs into issues in device power consumption and server data management. As a result, applications run into scaling constraints. Urban sound classification is a sensing application that can benefit from fog-based IoT architecture. However, before being a possible implementation option, the cited issues need to be addressed. This work takes a Cloud-Fog-Device structure to create a network architecture template or framework that makes use of Urban sound classification as its application. Within this framework are implementations of the solutions proposed in this thesis. The solutions were process reallocation and active-low and high states. They aim to create a more scalable framework.

Tests were conducted to examine the feasibility of the solutions and ultimately the proposed framework. Two configurations were implemented initially; A and B. Configuration A was a more device heavy iteration based on fog computing. Both the recording and classification processes were included in the device for this design. On the other hand, configuration B relied more on
the processing capabilities of the server, which was based on cloud computing. It is different from the first by moving the classifier to the server.

The first test focused on the power consumption of devices. Power consumption relies on data size and process loads as significant variables. The configuration B resulted in better average power consumption of 1830.54 mW and runtime of 16.42 s compared to the 1852.00 mW and 57.77 s of A. The test results can be attributed to having the server having higher processing capabilities compared to the device. The next test is for data management in terms of latency, which is crucial for a growing network. Data speeds dictate how good a server is at managing data. Configuration B proved to be incapable of handling multiple devices in a fog network with a latency of 300 ms, while configuration A demonstrated its scalability with 5.4 ms. These two configurations showed their strengths and weaknesses in the tests conducted.

As a result, a third configuration was created as a hybrid of the two previous architectures. Configuration C balances the two by adapting based on the test results. Proper processing load balancing was implemented to reduce the power consumption of the devices. Also, the data sizes that are being transmitted between the device and the server was modified to remain constant to allow better data management for the server. The same tests were conducted with this configuration which resulted in average power consumption of 1786.86 mW and latency of 5.5 ms. These results present configuration C as a better overall implementation compared to the previous two.

Therefore, a framework is created that displays scalable capabilities that make it a feasible approach to implementing a Fog-based IoT network that uses an urban sound classification application for testing. With this configuration, proper processor load balancing is a step in the direction to minimize the power consumption of an end device and preserving its battery life. Also, focusing on
transmitted data size management can reduce the strain of incoming data traffic and improve a server’s data management. Overall, the concept of reallocating processes and managing resources such as power can lead to a more optimal configuration of a network. Also, other presented contributions in this thesis highlighted the ability of the framework to be used for other applications related to data classification. The future of IoT-applications is within the fog by finding optimal configurations and distributions of processes within a network. Not focusing on a single archetype can prove to be beneficial in creating a Fog-based IoT network that is the most power-efficient, and load-balanced.
Bibliography


