Design and Implementation of Google Cloud Framework for Monitoring Water Distribution Networks

by

Bhavana Hodasalu Sadananda

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Applied Science

in

Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2019

Abstract

With urbanization and growing human population, water demand is constantly on the rise. Due to limited water resources, providing access to fresh potable water to the rising needs is challenging. Water distribution systems are the main arteries that supply fresh water to all the house-holds, offices and industries. Various factors such as excessive water pressure, aging or environmental disturbances, e.g., from road traffic, can all contribute to damage of water distribution pipelines and can result in leaks in water distribution networks (WDN). This could in turn result in financial loss and could pose additional challenges in providing potable water to the entire community, sometimes even leading to contaminant intrusion. Traditional leak detection methods such as visual inspections can detect leaks; however, this method is reactive in nature and can result in potentially losing large amounts of water before intervention strategies can be employed. On the other hand, hardware-based inspection techniques can accurately detect leaks, but are labor intensive, time consuming, expensive and effective only for short distances. Some existing software techniques are less expensive; however, their effectiveness depends on the accuracy of data collected and operating conditions. Modern existing leak detection techniques based on Internet of Things (IoT)—consisting of data collection sensor sub system, internet connectivity and a decision making sub system—alleviate many issues associated with hardware and software methods, however they are considered to scale poorly and face security issues, fault tolerance issues, interoperability issues, insufficient storage and processing abilities to store and process large quantities of real time data captured by the sensor sub systems. As a potential solution to these issues, this thesis deals with the application of a cloud-based leak detection system within the overarching concept of IoT. A detailed design and implementation of Google Cloud Platform (GCP) which can provide scalable, secure data processing system to analyze both real-time and batch data collected from IoT devices monitoring a WDN is presented. To circumvent the issue of access to a live WDN, the proposed system uses emulators, python Hyper Text Transport Protocol (HTTP) client running on a computer and a python HTTP client running on an IoT device (Raspberry Pi 3) to simulate live streams of acoustic pressure data from hydrophone sensors. Since the data itself was collected from a live WDN, the decision-making subsystem mimics results expected from live WDN data. The data ingestion layer on GCP incorporates two types of authentication: OAuth2.0 authentication and Application Program Interface (API) key authentication along with other GCP components using service account features to ensure end-to-end secure data processing. Decision support sub-system includes simple, yet powerful algorithm, namely the one class support vector machine (OCSVM) with non-linear radial basis function (RBF) kernel. It is shown in this thesis that GCP provides a scalable and fault tolerant infrastructure at every stage of data life cycle such as data ingestion, storage, processing and results visualization. The implementation in this thesis demonstrates the applicability of the leak detection IoT framework and the concept of a cloud based IoT solution for leak detection in WDN, which is the first demonstration of its kind to the author's knowledge.

Acknowledgements

I would like to sincerely thank my supervisor Dr. Sriram Narasimhan for giving me an opportunity to be a research associate in his team and then accepting me as a Master's student. I really appreciate his guidance, motivation and help throughout my graduate studies. I thank him for providing feedback on my thesis.

I thank Professor Fue-Sang Lien and Professor William W. Melek for accepting to be my thesis readers and to be in the exam committee.

I also would like to appreciate Jinane Harmouche for providing me with her invaluable signal processing expertise and providing technical assistance in implementing data analysis algorithm. I thank her for the feedback that she provided on my thesis.

My sincere thanks goes to Professor Keshav Srinivasan and Costin Ograda-Bratu from the Computer Science Department for their help during the initial stages of this project.

I appreciate Dirk Friesen for building a resilient electronic device and developing the requisite firmware to collect field data. The field data that I used in this project was collected by the members of the SDIC Lab, Dirk Friesen and Roya Cody. I really appreciate their hard work.

Dedication

To my loving husband Naveen Chandrashekar who babysat our son whenever I was busy studying, to my dear parents and to my caring sister, niece and brother-in-law for supporting and encouraging me to pursue graduate studies. It is specially dedicated to my son Soorya who puts a smile on my face everyday.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Water is an integral part of human life. Only 3% of water on earth is fresh water and is not readily available for human use as most of it is trapped in glaciers and polar ice caps or under-ground and only a small portion of it is available above the ground or in the air (Wu, Z.Y. et al., 2011). Consequently, water needs to be accessed from underground or surface water sources and pumped to treatment plants, treated, stored and distributed based on demand. Due to rapid increase in human population and urbanization, water demand has increased tremendously, placing a lot of pressure on reducing or eliminating water wastage. Water distribution networks (WDNs) are the main arteries used to transport treated water from water sources to treatment plants, and from treatment plants to homes, industries and offices. In most WDNs, a large amount of water is lost in transit from treatment plant to the consumers; the amount of water lost is estimated to be 20-30 percent between treatment and delivery and in older WDNs this may be as much as 50 percent (Hunaidi, O. et al., 2000).

Water loss can be due to pipe flushing, metering errors, leakage or theft. Leakage is one of the major causes of water loss (Hunaidi, O. et al., 2000). Leakage can occur in different parts of the distribution system such as service connection pipes, joints, valves, transmission pipes, distribution pipes and fire hydrants. Several factors including wear and tear due to aging of pipeline infrastructure, excessive loads and vibration from road traffic, natural disasters, vandalism, excessive water pressure, material defects, corrosion, faulty installation and ground movement due to extreme weather condition can all contribute to leaks in WDNs. Leaks not only result in wastage and non-revenue water but could also pose health risks as proximal sewage water could intrude into water mains under vacuum conditions resulting from large main breaks. Small leaks could lead to damage in the pipe network, e.g. erosion of pipe bedding, leading to bursts, pipe breaks and eventually damage to the foundations of roads and buildings (e.g., Figure 1.1). All of these negative consequences are forcing municipalities and infrastructure owners to reconsider conventional leak detection methods, which are mainly visual in nature, and implement robust technology-driven leakage monitoring and control technologies.

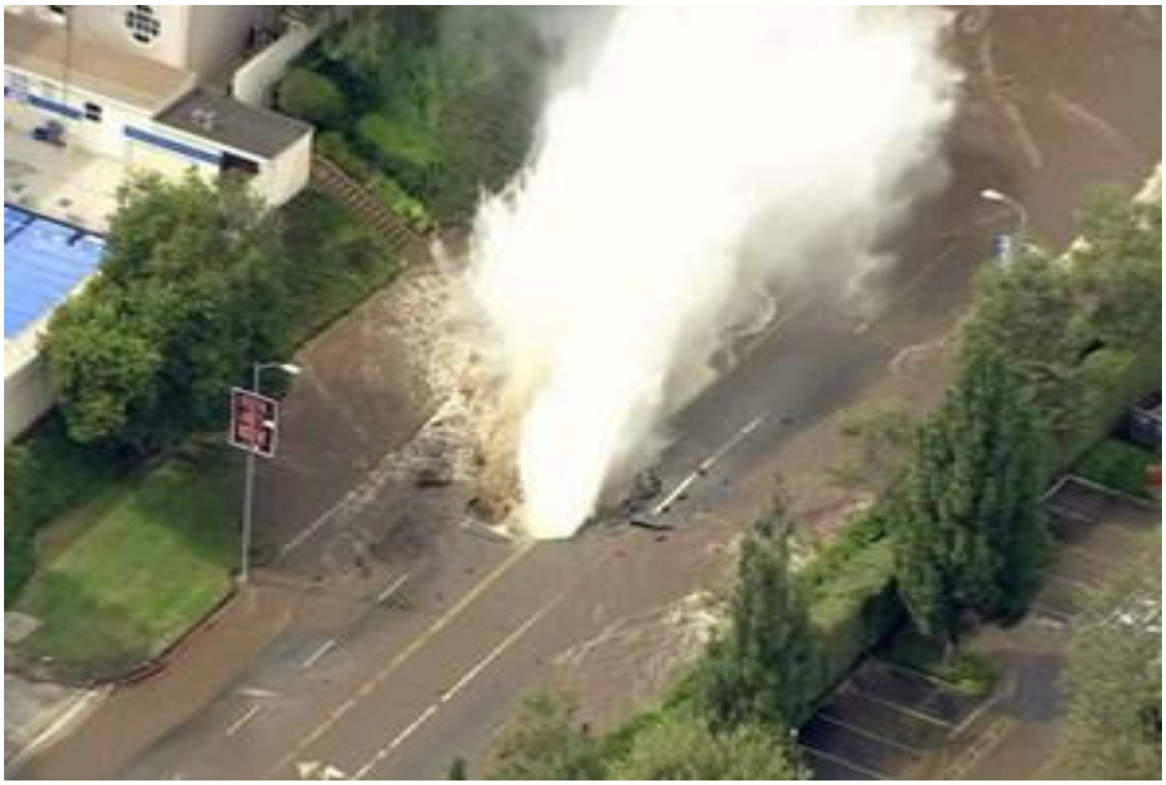


Figure 1.1 Water burst resulting in damage to a road [1]

1.2 Long-term monitoring for leaks using an IoT framework

Traditional leak detection methods are manual, visual, reactive, and allow failures in pipes to occur before intervention, which is not an optimal maintenance strategy. Furthermore, manual inspection of the entire WDN for signs of leak(s) is labour intensive. As well, many leaks do not surface at all, which results in a low detection rate. Hence, there is a pressing need for a monitoring system which can continuously monitor WDNs to detect leaks as soon as they occur so that intervention strategies can be put in place. Leaks can be detected by monitoring various parameters such as water pressure, acoustic pressure, soil properties of surrounding soil along pipelines, temperature of surrounding soil and amount of water that flows across a given point along the pipeline. In most cases, sensors need to be deployed within the pipeline which provide a direct means to monitor a WDN system in order to gain insights regarding their condition. Traditionally, deploying dense sensor networks have been associated with large sensor hardware costs and software complexity. However, with the recent advances in micromachinery and semiconductors, sensors, microcontrollers and single board computers have matured in terms of power consumption, storage capacity and processing ability, while becoming cheaper at the same time. This has opened up the possibility to deploy relatively inexpensive dense sensor networks within reach of many cities and municipalities. Such dense networks form the basis for what is commonly known as Internet of Things (IoT).

Although modern microcontrollers/single board computers have advanced significantly in terms of their capabilities, the high data acquisition rates from sensors (pressure sensors and hydrophones) deployed for leak detection introduce several challenges in terms of power management, storage, processing and transmission of large amounts of data. Some of the existing leak detection techniques use data in its raw form (Whittle, A. J. et al., 2013). Processing data in its raw form is not efficient and scales poorly. Some leak detection techniques (Karray, F. et al., 2016) run leak detection algorithm on sensor/edge nodes, however running complex algorithms on edge devices is not power efficient. To overcome this, lossy compression techniques (Feng, N. 2019) such as extracting features from raw data, (while retaining key leak related information), can be employed at the sensor nodes to transmit only the results to remote servers with large storage and processing capacities, while discarding raw data. This is in general the preferred alternative to transmitting all the raw data and storing and processing at a remote centralized location.

As cities grow, WDNs need to expand and so do sensor networks. Traditional sensor networks often have one or two central processing computer systems which process the data collected from all the nodes (Adedeji, K. et al., 2016). Such an architecture is not scalable and does not possess sufficient redundancy. To overcome these issues, distributed systems are needed which provide good fault tolerance with abundant processing and storage capabilities. More recently, commercial vendors, called cloud service providers, have started to provision and maintain multiple computers and software resources to provide data processing and storage services, which are subscription based. Such cloud services also provide ready access to user friendly data analyzing tools such as signal processing or machine learning tools, without the need to build such applications from scratch. Some studies have investigated cloud based leak detection solutions (Arangoa, I. M. et al, 2014), however, they lack the description of key design and implementation aspects and hence outside the reach of current academic community interested in this application.

1.3 Objective

The overarching objective of this thesis is to demonstrate the design and implementation of an auto-scalable cloud based system that can achieve reliable leak detection in WDNs in both real time and in batch. This thesis aims to demonstrate this technology using actual acoustic data captured from a live WDN in Southern Ontario and state-of-the-art classification tools available in a leading cloud services platform.

1.4 Organization

This thesis is organized as follows.

Chapter 2 reviews existing leak detection methods, their advantages, drawbacks, and provides basic information about the system proposed in this thesis which can overcome some of the main drawbacks identified in existing solutions.

Chapter 3 provides the details of the software architecture which is used in this thesis. It also introduces the basic concepts of the cloud platform which is used to develop the leak detection framework for WDNs.

Chapter 4 describes the details of data collection, feature extraction, data analysis and results of a powerful classification algorithm which is employed in this thesis to detect leaks. The implementation details of emulators and cloud platform components used in the architecture are provided in Appendix A.

Chapter 5 concludes the thesis by summarizing major findings in this research and proposes ideas for future research.

2 Background and Literature Review

In this chapter, traditional leak detection techniques in the literature are reviewed and their advantages and limitations are discussed within the context of the overarching objective of this thesis. State of the art tools based on IoT, their advantages, disadvantages, existing cloud-based solutions and their limitations are also discussed. This chapter will then set the stage for the motivation and the stated contributions of this thesis.

2.1 Traditional leak detection methods and limitations

Pipelines are the most economical way of transporting water from one place to another. Older distribution pipelines are typically made of asbestos cement or cast-iron material. More recently, they have been made of poly vinyl chloride (PVC), polyethylene or ductile iron material. Regardless of the material used, these pipelines are designed and constructed to last for several decades with many well past their intended service life. Additionally, many factors such as material defects, corrosion, excessive water pressure, ground movement due to freezing or drought, excessive vibration caused by traffic loads can all result in pre-mature failure of these pipes. Though pipelines made of ductile material (such as copper pipes) resist cracking, pinhole-sized leaks (shown in Figure 2.1) can occur in them which grow slowly over time. In pipes made of less ductile materials such as cast iron and asbestos cement, crack-like (cracks are shown in Figure 2.2) leaks occur in episodes with each episode separated by days, months or even years. Both pinhole leaks and cracking leaks over a period of time may progress to complete failure of the pipeline structure and could eventually lead to bursts. In order to detect leaks before they progress into larger bursts, numerous techniques have been developed and discussed in the literature [cive700, 2018]. A formal classification method is not available; hence they are discussed from the standpoint of the core technology used in these methods, which is summarized next.

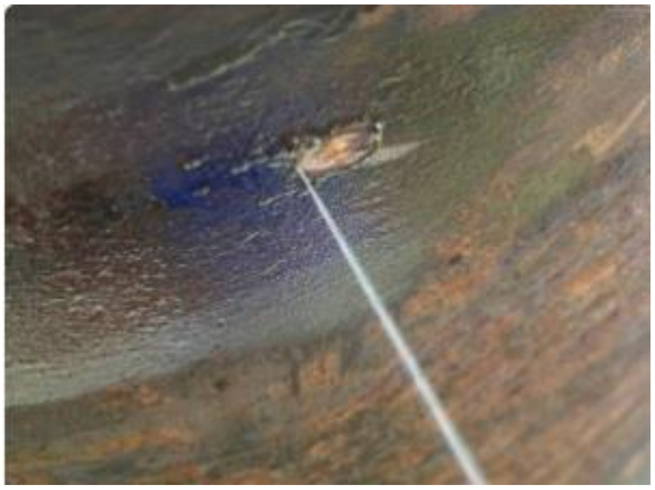


Figure 2.1 Example of a pinhole leak [2].

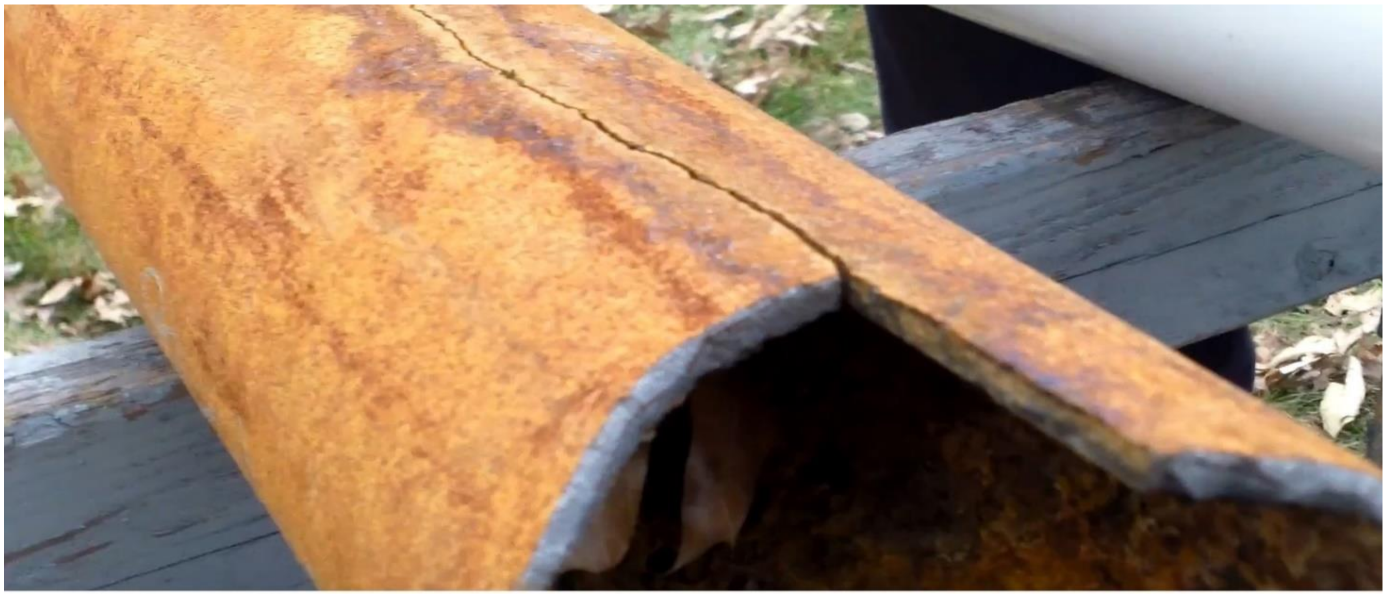


Figure 2.2 Example of a cracks in the pipes which can result in large leaks [3].

2.1.1 Reactive and proactive methods

At a very basic level of classification, leak detection techniques can be reactive or proactive. As the name implies, in the reactive approach leaks are detected either through eye-witness notifications or customer complaints. This is not a formal leak management strategy and can easily lead to water loss of up to 40% (Puust, R. et al., 2010, Ramos, H. et al., 2001) and it presents inevitable problems for customers. Proactive approach is meant to be preventive in nature and it involves frequent monitoring of pipelines for anomalies, which are indicative of leaks. Detecting anomalies and repairing small leaks early prevent progressive catastrophic damage. The literature review in this chapter focuses on proactive techniques and are grouped according to the core technology being utilized for detecting leaks and anomalies in each case.

2.1.2 Hardware and software methods

Leak detection methods can be hardware based or software based. Hardware based methods employ appropriate equipment such as listening rods, leak correlators and leak noise loggers to detect leaks. Some sub categories of hardware based techniques are acoustic leak detection (Kim, M. et al., 2009), fibre optic sensors (Tapanes, E., 2001), pigging (Furness, RA. et al., 2009), vapor or liquid sensing tubes (Geiger, G. et al., 2006), liquid sensing cables (Geiger, G. et al., 2006) and soil monitoring (Lowry, W.E. et al., 2000). Hardware based techniques have shown to exhibit good sensitivity to leaks and are quite accurate in finding leak locations. However, they are expensive and their installation can be very complex and time consuming. Leak detection using such techniques is limited to relatively short distances. Hence, they are used more for inspections as required (when a leak is known to have occurred or discovered during leak surveys) rather than for long-term monitoring when the presence or absence of leaks are unknown to begin with. Software based techniques involve monitoring internal pipeline parameters such as pressure, flow and temperature with or without the aid of sensors. Various sub-categories of software based techniques to detect leaks are: digital signal processing (US Department of Transportation, 2007), real time transient modelling (Hauge, E. et al., 2007, Carbó-Bech, A. et al., 2017), statistical approaches (Zhang, J. et al., 1998), negative pressure waves (Mpesha, W. et al., 2001), mass-volume balancing (Liu, J. et al., 2008), to name a few. Based on a multitude of implementations used in the literature using the above subcategories of software based techniques, it is safe to conclude that software techniques are less expensive compared to their hardware counterparts. However, their effectiveness depends on the accuracy of collected data, operating conditions and system characteristics (Golmohamadi, M., 2015, Gamboa-Medina, M.M. et al., 2014). Despite an enormous activity in this area, there has not been a consensus regarding the best method or class of methods to be used for leak monitoring or detection.

2.1.3 Visual, hydraulic and acoustic methods

Yet another way to categorize leak detection techniques is, for example, based on whether they rely on visual, hydraulic or acoustic characterization of the WDNs.

Visual Inspection

Some of the visual inspection methods available are laser scans, satellite based methods, ground penetrating radar (Eyuboglu, S. et al., 2003, Ayala–Cabrera, A. et al. 2013, Abouhamad, M. et al., 2016) and infrared thermal camera based methods. The effectiveness of visual inspection methods is relatively less established compared to hydraulic and acoustic methods and hence is not discussed in detail here. Infrared thermal camera-based inspection can detect and locate leaks in pipelines buried as deep as 30 meters below ground surface (Weil, G.J. 1993, Jackson, C.N., et al. 1998). Modern satellite-based methods use images taken from satellites to perform complex analysis for possible signs of leaks. Such techniques are convenient, cost and time effective. However, the limitation of this method is spatial resolution; high image resolution is necessary to identify the leaks as leaks generally occur in small regions along the pipe (Hadjimitsis, D.G. et al., 2010; A. Agapiou et al., 2014). Ground penetrating radars locate leaks in water pipes by detecting underground voids created around the pipes near the leak locations by leaking water. These voids are easily formed in sandy soils whereas in the soft clay soil such voids are not easily formed and is difficult to detect using radar (Hunaidi, O. et al., 2000). In laser technology, fibre-optic cables are installed along the length of the pipeline. When a leak occurs, water comes in contact with the fibre-optic cable changing its temperature and thus changing the optical response of the laser beam pulse passing through the cable, indicating a leak (Geiger, G. et al., 2003, Großwig, S. et al., 2001). This technique works only if the water is at a temperature different from that of fibre-optic cable, not otherwise.

Hydraulic approach

In the hydraulic approach, pressure, flow, and transient pressure are used to detect leaks as leak information is known to manifest in these parameters which can be measured relatively easily within the WDN, either locally at the pipe level or globally at the network level. Subcategories of this type are district metered areas (DMAs), pressure and flow based burst detection, and transient pressure based techniques.

(a) DMAs: DMAs are discrete areas of WDN which are created by closing boundary valves which can be opened if more water needs to supplied to the area and hence remain flexible for changing demands. DMAs can also be created by permanently disconnecting two areas within the WDN. In DMAs, water flowing in and out of DMAs are monitored by using meters (Lambert, A., 1994, MacDonald, G. 2005). Due to the incompressible nature of water, flow meter reading at the DMA level can indicate loss of water and is effective in detecting the rate of rise of leakage. Limitation of this method is that there can be flow meter error which is usually around +/-5%, so only those flows larger than this error can be detected using this technology.

(b) Burst detection: A sudden, high volume outflow of water from a single point in the network is called a burst which can be detected directly through pressure and flow monitoring, both of which are hydraulic parameters. Such events however, are usually reported by the public. Zan, T. et al., 2014 demonstrated that large leaks are associated with pressure transients and can be detected using pressure sensors. Wu, Y. et al., 2017, Mounce, S.R. et al., 2011 and Kim, Y. et al., 2016 discussed techniques to detect bursts using a data driven framework. Since bursts result in heavy loss of water, its immediate detection and shut off of water can be highly valuable (Lambert, A., 1994).

(c) Transient based leak detection: In transient based leak detection approach, small but sudden changes in the pipeline pressure caused due to changes in the pipe wall such as a crack or a hole is captured by monitoring pressure (Pudar, R.S. et al., 1992, Kapelan, Z.S. et al., 2003, Kapelan, Z.S. et al., 2002, Colombo, A.F. et al., 2009). There are pressure (hydraulic) transients and acoustic transients, both of which could be caused by cracking events, but tend to be small and difficult to detect. So, sophisticated signal processing and artificial intelligence (AI) techniques may be needed to detect them (Vitkovsky, J., et al., 2000). Since small leaks cause negligible transients, detection of pinhole leaks is difficult using conventional transient detection techniques.

(d) **Induced transient reflection:** Unlike passive transients monitored due to sudden changes in the pipe conditions, induced transient reflection requires creation of pressure transient, e.g., by a sudden closure of a valve (or, using a solenoid valve). This is an active method, where reflections of the active transient waves captured by sensors are used to identify the presence of a leak (Wang, X.J. et al., 2002, Karney, B. et al., 2009, Pothof, I. et al., 2012, Feng, N. 2019). Any change in the physical structure of the pipe such as a crack or a hole causes wave reflection, thus altering system's flow and pressure response. The transient signal from a leaking pipeline exhibits a distinct singularity not present in intact pipes. The transient signal in the leaking system decays (Nixon, W. et al., 2006) more rapidly compared to that in the intact system. The properties of reflected signal and the measure of decay of transient signal at a measuring location are used to identify the presence of leak in the pipelines (Colombo, A.F. et al., 2009). Transients can introduce disturbances into the pipelines and cause the pipelines to rupture. So, measures need to be taken to ensure that the transients generated are safe transient and do not by themselves result in compromising the pipe structure integrity. Some devices that provide protection from water surges are surge tank (open), surge vessel (air chamber, closed surge tank, bladder tank and hybrid tank), feed tank, surge anticipation valve, pressure relief valve, air release/vacuum valve and pump bypass line. The general principle behind the functioning of such devices is to store water or otherwise delay the change in flow rate or to discharge water from the pipeline (Boulos, P. F. et al., 2005). Though induced transient reflection method helps to identify leaks, it is not very popular as it requires active generation of transients.

Acoustic approaches

Acoustic approaches are by far the most prevalent leak detection methods in use today. The subcategories of this method are: inline acoustics, ground sounding, sounding at fittings and correlation methods.

(a) **Inline acoustics:** In the inline acoustic method, acoustic sensors such as hydrophones or encased accelerometers are inserted into pipes. In Chang, Y.C. et al., 2009 (PipeProbe project), the authors proposed a tiny capsule of a waterproof mobile sensor system consisting of EcoMote with a built-in accelerometer and a pressure sensor (MS5541C from Intersema). This capsule is dropped into the pipeline at one end, which then traverses the length of the pipe and stores pressure measurements on an on-board flash memory, which is then subsequently retrieved manually and analysed in the laboratory. This method is intrusive and needs a lot of supervision. SmartBall, a commercially available system (Pure technologies) consists of a range of acoustic sensors, accelerometer, magnetometer, ultrasonic transmitter and temperature sensor. It travels along the pipe when the water flows, collects acoustic data, detects and locates the leak. Similarly, Seto, L. et al., 2013 proposed the idea of a small ball embedded with an acoustic sensor, temperature, and a pressure sensor inserted into a pipe and the data is collected and analyzed for any anomalies inside the pipe. While this technique is promising for inspections where a leak is known to occur, it can be quite intrusive and expensive to deploy on a large scale when the approximate leak location is unknown.

(b) **Sounding at fittings:** Depending on the size and type of the pipe, leak sound/vibrations are transmitted either through the pipe wall or through the water column over long distances. This is especially true for the case of metal pipes and not as much for plastics. Such mechanical vibrations produced by water leaking from pressurized pipes can be detected by the sensors and acoustic devices at various convenient-to-access locations in the network. Sounding at fittings involves placing sensors at fittings such as service connections, fire hydrants or isolation valves and listening for leaks (Hunaidi, O. et al., 2004, Pal et al., 2010). Loggers equipped with remote communication capabilities then record and transmit the sound file or features extracted from such sound files to monitoring centres for processing. This method is popular as it generally utilizes existing access points in the distribution network for monitoring and does not require major construction or modification to the existing pipe network.

(c) **Ground sounding**: This method involves placing a ground microphone at the suspected leak location. Initially, the presence of leak is identified, by listening to leak sounds from fire hydrants/valves. It is then localized, by listening directly above the pipe, on the ground surface, at close intervals of about 1 meter. This process is time consuming and useful only to localize the leak, if the leak is already detected (Hunaidi, O. et al., 2004).

(d) **Leak correlators:** In the correlation method, acoustic sensors are placed at fittings separated by a fixed and known distance. Time synchronized signals (from two hydrophones or accelerometers) are transmitted from the sensors to a central device. The signals are then cross-correlated and the leak is localized using the known acoustic speed in a pipe and the lag corresponding to the peak in the cross-correlation. Correlation methods have been found to perform well and are more accurate compared to listening devices (Choi, J. et al., 2017, Zhang, L. et al., 2013, Hunaidi, O. et al., 2004, M. Pal et al., 2010). Compared to accelerometers, hydrophones have been shown to be less susceptible to environmental noise and better tuned to low frequency sounds compared to accelerometers. However, in the correlation-based methods, isolating leak induced noise, which tends to be small compared to other environmental noise factors such as pumps, still remains a major challenge.

2.1.4 Non-acoustic methods

Other non acoustic leak detection methods are also available, which do not fit into the general classifications described previously. They are: tracer gas (Lowry, W.E. et al., 2000), electromagnetic methods (Liu, Z. et al., 2013) to name a few. Their use is limited and their effectiveness is relatively less established compared to acoustic methods and hence are not discussed in detail here.

2.1.5 SCADA approaches

Traditional SCADA (supervisory control and data acquisition) systems have also been considered to mine for leak related data at a system level. In this schema, permanent flow meters from various points in the WDN telemetrically send data to the SCADA system for analysis, e.g., flow rate data, which can then be analysed for anomalies by mining this SCADA information (Mounce, S.R. et al., 2007). Although such an approach can provide near real time monitoring information, most SCADA systems in use today generally collect information from pumping stations, water tanks or reservoirs, which do not allow for local monitoring and can only be used for large bursts and are associated with large initial deployment cost (Cheung, P. et al., 2014). Interoperability is another issue within this method; devices manufactured from different vendors, or different versions of devices from same vendor often present challenges in being able to work together. Scalability is another issue; due to its traditional architecture, SCADA performance degrades with an increase in the number of users. Furthermore, SCADA systems are designed mainly to run the day-to-day operations and so it can ingest and store only a finite amount of data. SCADA systems tie together decentralized subsystems which makes security a major threat (Gao, J. et al., 2013).

With the introduction of IoT technology, many of the issues described previously are addressed by augmenting SCADA systems with IoT. SCADA can still act as one of the data sources (Romano, M. et al., 2013), while IoT focuses on analyzing the granular machine data. Details about IoT technology and other IoT based leak detection systems present in literature are discussed next.

2.2 State-of-the-art monitoring technologies

With the exception of very few, a majority of the technologies discussed previously can be regarded as inspection methods and cannot be easily scaled to monitor large WDNs. With the advent of IoT and the back-end cloud technology, recent trends have demonstrated the use of such tools for leak detection in long-term monitoring situations. This section discusses some modern leak monitoring systems in WDNs, their advantages and their limitations and then concludes with a summary of major gaps and a road map to address key limitations in current technology.

2.2.1 IoT

IoT, which is a hardware and software based pro-active technique, has been explored as a framework in modern leak detection systems to a limited extent. In simple terms, IoT can be viewed as a network of physical devices or “things” embedded with sensors, software and connectivity which enable those things to connect, collect, and exchange data (Anzelmo, E. et al., 2011, Barnaghi, P. et al., 2012). IoT extends internet connectivity beyond standard devices such as laptops, desktops, smartphones and tablets to everyday objects which were traditionally considered “dumb”. IoT enables things or systems to be observed and understood using measured data. As these things are embedded with sensors, they can collect data and by analyzing such data an organization or an individual can make informed decisions and benefit from timely decisions or interventions. This concept also supports the overarching theme of Smart Infrastructure, where IoT devices and decision support systems can help manage deteriorating infrastructure such as WDNs. The basic IoT concept is described in Figure 2.3 and consists of three components: sensor and data acquisition system, network connectivity and data processing and decision making layer. WDN when integrated with these three components can be envisioned as a smart water distribution system, or an IoT system.

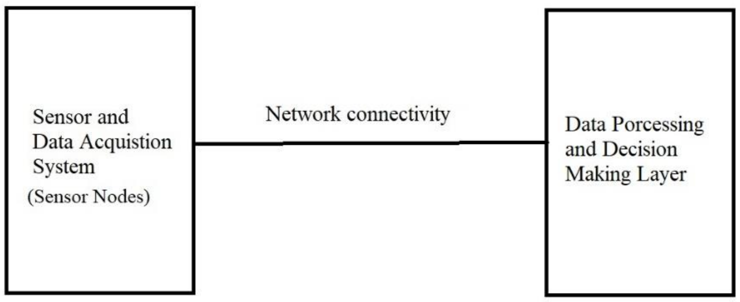


Figure 2.3 Three components comprising the IoT

More details about the three components namely, sensor and data acquisition system, network connectivity and data processing and decision making layer are discussed in the following sub-sections. The discussion is limited to IoT applications involving water pipes and distribution systems in order to keep the literature review both pertinent to this thesis and manageable.

Sensor nodes

A network of sensor nodes is at the heart of any IoT based leak detection solution. A sensor node typically consists of sensor(s), a microcontroller/microcomputer and a power source. There is no accepted standard for the type of sensors to be used in leak detection application and a variety of sensors and micro-controllers have been employed for this purpose. For example, Christodoulou, S. et al. 2010 used sensor nodes comprising of a Mica 433MHz mote data acquisition system placed in a waterproof package and a Decagon Ech$_2$o dielectric sensor for soil volumetric water content (soil moisture) measurements. Water flow was measured using KENT V100 meters and Zonescan-800 leak noise logger was used for leak noise detection. Mohamed, N. et al., 2011 proposed acoustic wireless sensor nodes with an acoustic transceiver, a processor, a battery, a memory, and small storage at each sensor node. AL-Kadi, T. et al., 2013 used two layers of sensors: a hub layer and an in-soil layer sensor. The hub layer was deployed inside the pipeline and consisted of pressure sensors and acoustic sensors. The soil layer was deployed outside the pipeline and consisted of sensors that could measure soil properties such as moisture and temperature. These measurements were sent to a head cluster located at the pump station where the data was processed and sent to a remote administration centre. Whittle et al. 2013 used a wireless sensor node with hydrophone and pressure sensors, a 72MHz ARM Cortex M3 CPU with 64KB of RAM, a 2GB SD card for storage, a GPS with pulse-per-second functionality for time synchronization and a 33Ah 12 V battery. Adedeji, K. et al., 2016 employed a pressure sensor, an Arduino microcontroller, a battery and a flash memory in the sensor node. Kartakis, S. et al., 2016 used four nodes with an Intel Galileo board and an ethernet module, two nodes with an Intel Edison board with WiFi connectivity. Along with these, a smart water sensor board consisting of a flow meter, a pressure sensor and a motorized valve were used. Karray, F. et al., 2016, proposed a wireless IoT solution (EARNPIPE), where each sensor node consists of an ARM processor, a Kalman filter accelerator, a sensor interface, ADC, memory, a radio transceiver and a power management unit. Stoianov, I. et al., 2007 used a sensor node consisting of an Intel Mote with ARM7 core, 64kB RAM, 512kB flash and a Bluetooth enabled communication connected to a pressure sensor and an ultrasonic sensor. The sensor node in Sadeghioon, A.M. et al., 2018 consists of a relative pressure sensor, temperature sensor, one attached to the pipeline wall and the other connected to soil in close proximity to the pipe. Abdelhafidh, M. et al. 2017 used a TelosB sensor node with a flow sensor and a pressure sensor. The above examples show the heterogeneity in the sensor selection used in the literature and the fact that there is no consensus on the sensor nodes to be used for monitoring WDN.

Network connectivity

Sensors in the context of IoT have to work collaboratively (rather, the information obtained from them) in order for it to be useful. For example, localization of leaks is only made possible by processing multiple time-synchronized data sets. Hence, it is important to study how individual sensors have been connected or networked. Sensor networks can be wired or wireless. Mohamed, N. et al., 2011 proposed multiple sensor node architectures for underwater pipelines, one of which is a wired sensor network. Needless to say, wired connections suffer from reliability issues resulting from damage to the wires in the network and installation cost issues. As a result, a vast majority of the IoT based leak detection systems are configured as wireless sensor networks (WSN). The following examples show that a variety of wireless technologies were used in literature.

Whittle, A.J. et al., 2013 used a cellular modem with 3G connectivity to transmit data from sensor nodes to a remote server. Adedeji, K. et al., 2016 used ZigBee communication module to send data from several Arduino micro-controllers to a central node. Data collected at the central node was then transferred to a remote computer using RS232 or LAN interface. Kartakis, S. et al., 2016 used six sensor nodes to collect data and four of them communicated data with a central server using an ethernet module and two of them communicated data with a central server using WiFi. Karray, F. et al., 2016, used a radio transceiver for data communication between sensor nodes and a central server. Stoianov, I. et al., 2007 used Bluetooth for communication between the data acquisition system and a single board computer, with GPRS for communication between a single board computer and a backend server. In Christodoulou, S. et al., 2010, data acquisition boards transmitted data to a *Stargate* gateway which in turn communicated data with a remote computer through GPRS link. Sadeghioon, A.M. et al., 2018 used a transmitter to send data from a sensor node to a laptop using radio frequency signals, following which a 3G cellular network was used to transmit this data to the cloud. Abdelhafidh, M. et al., 2017 used radio signals to send data from a group of sensor nodes called a cluster to a central location called the base station through a cluster head. AL-Kadi, T. et al., 2013 proposed a magnetic induction-based communication between sensor nodes and a cluster head, which in turn communicated this data to a remote computer through electromagnetic waves.

Data processing and decision making layer (DPDML)

The downstream of measurement and communications in an IoT based leak detection system consists of processing and analyzing the collected data, detecting and localizing leaks, all of which constitute the data processing and decision making layer (DPDML). The data analysis methods in the DPDML can be classified as model-based approaches, signal processing methods, and knowledge based approaches. Model based approaches use concepts such as conservation of energy, conservation of mass or conservation of momentum to build a model to predict leaks. One such method that is widely used in many IoT based leak detection solutions is the Kalman filter. Karray, F. et al., 2016 used a linear predictor Kalman filter to compress data for the identification of leaks. Signal processing based leak detection techniques involve signal analysis in time domain or frequency domain. Support vector machines, pattern recognition, expert systems are some of the examples that have been used based on signal processing techniques. Stoianov, I. et al., 2007 mainly used pattern recognition to detect leaks, where pressure data and flow sensor data were used to find large leaks, whereas to find small leaks acoustic data was used. A simple Haar wavelet transform was used to detect pressure pulses. Leaks were detected based on the wavelets associated with coefficients higher than predefined values and a cross-correlation algorithm was used to find leak locations. Romano, M. et al., 2013, employed several self-learning AI, statistical and geo-statistical algorithms for data analysis. These techniques included wavelets to de-noise pressure/flow signals captured from sensor nodes, artificial neural networks (ANNs) for short-term forecasting of the pressure/flow signal values, evolutionary algorithms to find the optimal ANN input structure, statistical process control techniques to study burst induced pressure/flow variations and geo-statistical techniques to find leak locations. Whittle, A.J. et al., 2013 employed wavelet decomposition method and time domain statistical analysis to detect transients that are the indicatives of leaks. Adedeji, K. et al., 2016 employed improved negative pressure wave algorithm to detect leaks. Abdelhafidh, M. et al., 2017 used real time transient and wave propagation methods to detect leaks, and a genetic algorithm optimization technique to find leak locations. Sadeghioon, A. M. et al., 2018 used a seasonal hybrid extreme studentized deviate (S-H-ESD) algorithm developed by Twitter to find leaks. Christodoulou, S. et al., 2010 used ANNs and fuzzy logic for detecting and localizing leaks.

2.2.2 Advantages of an IoT based solution

One of the key advantages of employing IoT is the ability to scale sensors across large geographical areas and interoperability features in protocols such as HTTP, MQTT (Message Queuing Telemetry Transport), which is central to platforms being able to communicate across devices from different vendors. Most IoT devices in the market today support HTTP, MQTT and other common protocols such as WiFi, Bluetooth and ZigBee which have been shown to be scalable. IoT enables remote monitoring of the WDN along with near real time data processing capabilities, which is crucial to detecting leaks and proactively addressing the issues that could result in saving valuable resources and money.

2.2.3 Challenges in existing IoT leak detection solutions

The main challenges in WSNs is the issue of power management as sensor nodes are typically powered using batteries. This problem can be overcome by using low power sensors (Mois, G. et al., 2016), low power data acquisition boards (Kartakis, S. et al., 2015), light weight operating systems (Christodoulou, S. et al., 2010, used Tiny OS), low power network modules (Adedeji, K. et al., 2016 used ZigBee network module) to send and receive data, sending compressed data (Karray, F. et al. 2016) or features extracted from raw data (Feng, N. 2019) from edge nodes to the central node and/or using alternative energy sources such as solar (Christodoulou, S. et al., 2010). Rather than continuously collecting and sending data, intermittent capturing and sending of data from the data acquisition board to a central module can help reduce power consumption (Whittle, A. J. et al., 2013).

The main use with the deployment of real-world IoT leak detection systems is in their scalability, which means that the IoT leak detection solution can be scaled to work well with potentially hundreds or thousands of sensor nodes. This means that the data collected by those sensor nodes grows very quickly over a very short amount of time. Though modern single board computers or micro-controllers (which collect data from sensors) are equipped with relatively large storage and processing abilities, modern sensors capture data at very high rates, which can outgrow the storage and processing capabilities of even modern single computers and micro-controllers. The large deployment of sensor nodes combined with high data rates of the sensors can forfeit the benefit of the of IoT if proper measures are not taken to store and process the growing volume of data. Most of the existing IoT leak detection systems have not specifically addressed this issue of exponential growth of data within their architectures and have mainly focused either on the power management issue (Kartakis, S. et al., 2015, Sadeghioon, A.M. et al., 2014, Stoianov, I. et al., 2007, AL-Kadi et al., 2013) or on developing a good leak detection algorithm (Adedeji, K. et al. 2016, Sadeghioon, A.M. et al. 2018). Clearly, a holistic approach where power management, algorithms and scalability work together is necessary for a practical solution to leak detection in large scale.

There are many applications or frameworks available that can handle large varying volumes of heterogeneous data. For example, real time data analysis can be done using *Apache Storm* or *Apache Spark* streaming. A list of such frameworks is given in the Figure 2.4, but such applications only provide the framework to process varying volume of data and they still need supporting infrastructure such as computing resources, hard disks, CPUs, operating systems, regular back ups of data, upgrading the hardware and software and providing fault tolerant infrastructure by providing a distributed computing system. Such infrastructure maintenance along with the required applications (listed in Figure 2.4) that can process the Big Data, are provided by a modern computing paradigm, called cloud computing.

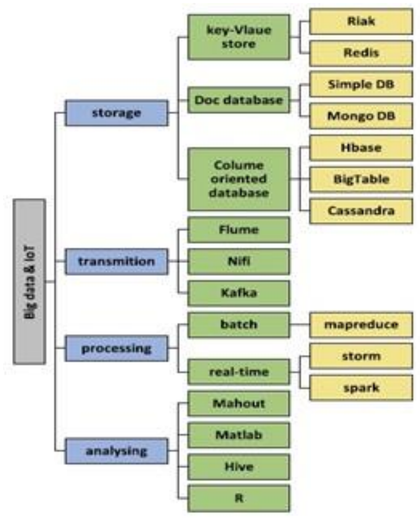


Figure 2.4 List of tools available to handle Big Data (Shadroo, S. et al., 2018)

Ahmed, K.E.U. et al., 2011 proposed the integration of WSN with cloud computing. This approach solves a major problem of WSN in general, namely storing and processing large amounts of data. This approach of using cloud computing in WSN was applied to the leak detection in WDN as well. However only a handful of IoT leak detection solutions in literature have applied cloud computing to address the *Big Data* problem. Koo, D. et al., 2015 presented a conceptual development of an IoT application for *Big Data* collection through a large number of water clients. However, this is only a conceptual design without implementation. Rohit, A. et al., 2018, used IoT system for forecasting and monitoring water consumption in WDN, where only the design of the system is presented and no details were presented with regards top system evaluation. Cheung et al. 2014 proposed a cloud computing solution called INFOSAN SaaS provided *by Optimale LTDA* [4] to address water loss in urban water distribution systems. Due to the proprietary nature of this solution, technical details regarding accuracy and an objective evaluation of the results are not publicly available. Mounce, S.R. et al., 2015 proposed a solution for water leak detection using a cloud-based portal, *YouShare*

[5], available to academics and research institutions. However, *YouShare* is not a self service cloud portal where users can access cloud services directly, rather access requests have to be first approved by the *YouShare* administrator. Arangoa, I. M. et al., 2014 used a cloud-based system, *Azure Cloud* [6], for data analysis and decision making in WDNs. This platform provides scalability in terms of running applications, where users can provision virtual machines (VMs) as needed. However, the process of provisioning VMs is manual and user input is needed for which user should have the knowledge regarding the number of VMs needed for data processing and often such information is not available *a priori*.

Though many WSN based solutions for leak detection have been proposed for monitoring leaks in WDNs, none of these studies present an end-to-end solution which can address all the issues such as ease of use, scalability, heterogeneity in sources, storage at scale, security, real time leak detection and visualization of results. To the knowledge of the author, this study is the first attempt towards providing a detailed design and implementation for a scalable, cost effective cloud based IoT online leak detection system. The Google Cloud Platform (GCP) is selected for demonstration purposes, as GCP can securely process and analyse varying volume of real time data as well as batch data while providing significant flexibility and range of available tools for analytics. The architecture, GCP concepts and GCP components used in the implementation of the cloud solution for leak detection in this thesis are discussed next in Chapter 3.

3 Architecture

3.1 Introduction

This chapter provides an overview of the architecture used in this thesis to implement a leak detection solution for WDNs, followed by a detailed review of each component used in the architecture. As the architecture in this thesis mainly follows an IoT framework, this has a DAL (data acquisition layer), a NL (network layer) and DPDML (data processing and decision making layer). Broadly speaking, DAL is connected to the DPDML using the NL. DAL collects time series data from the WDN at regular intervals, preprocesses it, calculates features and sends them over the NL to the DPDML for further analysis. NL provides wireless connectivity between the DAL and DPDML to enable communications between these two components. DPDML validates the data sources, validates the data (ingestion), stores the data, analyses it, checks for the presence of leak(s) in the WDN, and finally displays the results. The overall IoT architecture discussed above is described in the Figure 3.1. Figure 3.1 displays the key DAL stages of acquiring, preprocessing data and feature extraction, along with other components, tools and devices used to accomplish those stages, e.g., MATLAB®, emulators, sensors and microprocessors. These details are described next.

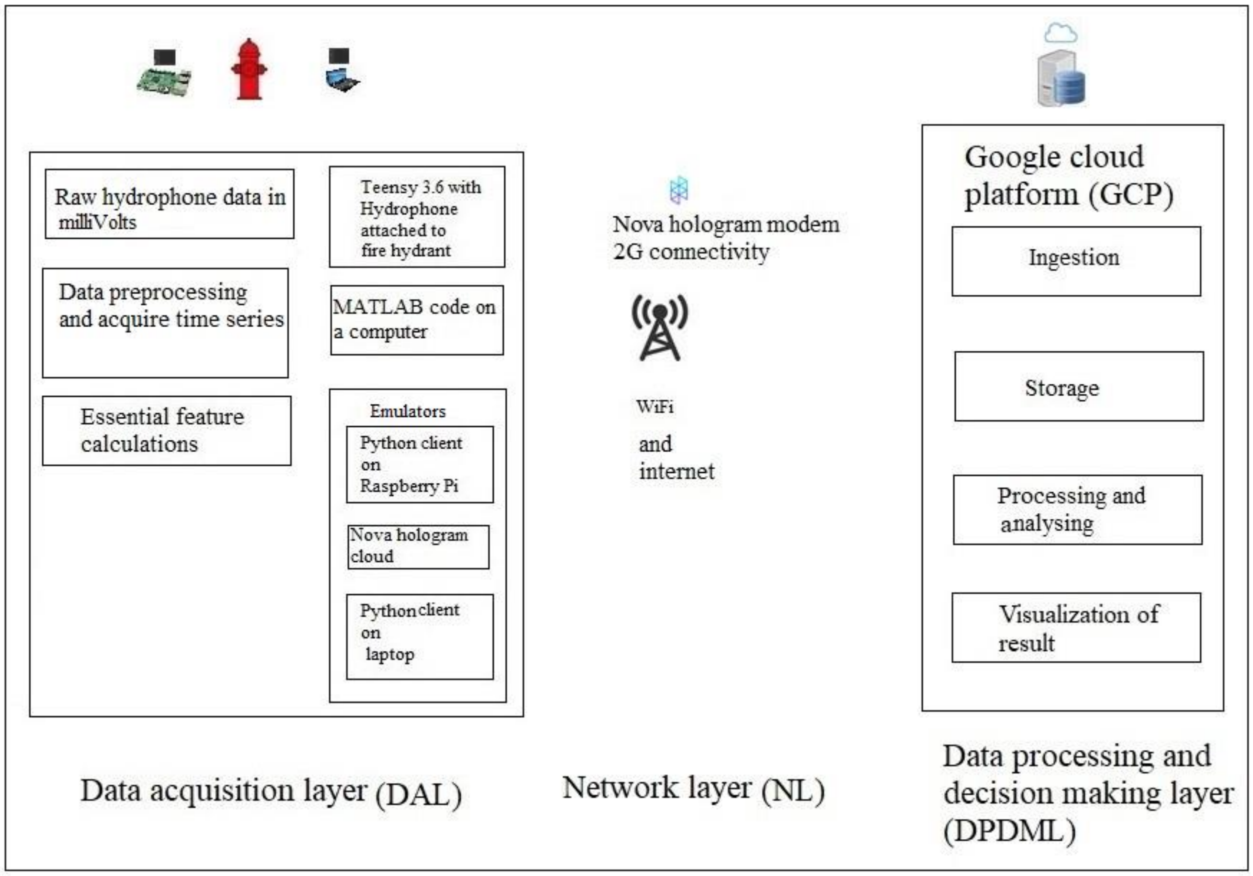

Raw hydrophone data in milliVolts
Data preprocessing and acquire time series
Essential feature calculations
Teensy 3.6 with Hydrophone attached to fire hydrant
MATLAB code on a computer
Emulators
Python client on Raspberry Pi
Nova hologram cloud
Python client on laptop
Nova hologram modem 2G connectivity
WiFi and internet
Google cloud platform (GCP)
Ingestion
Storage
Processing and analysing
Visualization of result
Data acquisition layer (DAL)
Network layer (NL)
Data processing and decision making layer (DPDML)

Figure 3.1 IoT Architecture

For the leak detection application, the DAL consists of a hydrophone sensor connected to a Teensy 3.6 microcontroller. Because of limited access to a live WDN, the data collected from the DAL is stored on a computer, preprocessed and transferred to two types of emulators (Python client on a Windows 10 laptop and a combination of Python client on Raspberry Pi 3 and Hologram Cloud), where relevant features are calculated and then sent to the DPDML. The NL consists of WiFi, Internet and a Nova Hologram modem which provides 2G connectivity. DPDML solely consists of GCP.

3.2 Data acquisition layer

3.2.1 Field hydrants, sensors and microcontrollers

The data used in this thesis was collected from a live municipal WDN, which is the DMA11 in ClairFields, Guelph, Ontario (Figure 3.2), which is predominantly a residential district which uses a mix of 6-inch and 12-inch PVC water pipes. The hydrant locations where sensors were installed are clearly marked in red in Figure 3.2 and are numbered from 1 through 5. These addresses correspond to 70 clairfields, 105 clairfields, 30 Paulstown, 10 Murphy, and 38 Keys, respectively.

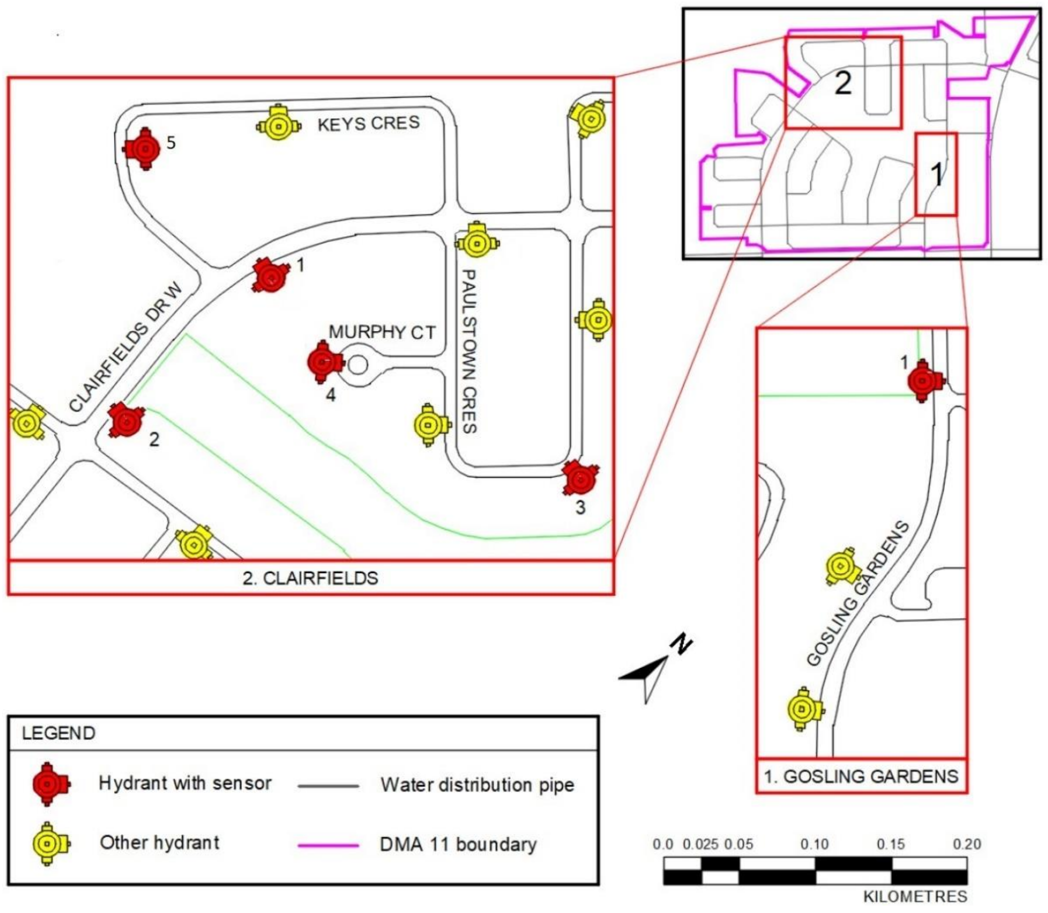


Figure 3.2 Map of the test location and test hydrants in Guelph, Ontario

The data-acquisition hardware consisted of a microcontroller (Teensy 3.6) onboard a custom printed circuit board (PCB) with a 2G internet and GPS chip. Firmware on this microcontroller was written in C++ programming language with open source libraries such as digital signal processing (DSP) libraries, Adafruit FONA library and SD card libraries. DSP library was used for vector statistical calculations and feature calculations such as fast Fourier transformation (FFT), Adafruit FONA library was used for cellular GSM connection, and secure digital (SD) card library for storage using the SD card. This main circuit board was placed outside the hydrant above the ground level. A second board, called the sensor board, was located below the ground level located at the bottom of a modified fire hydrant stem, shown in the Figure 3.3 (a), (b) and (c). Communication between the two boards occurred using the serial peripheral interface (SPI) protocol. The sensor board consists of an on-board 24-bit analog-to-digital converter (ADC) chip to digitize the hydrophone, a pressure transducer, and a temperature sensor inputs. It is important to note that only the hydrophone data is used in this thesis for analysis purposes.

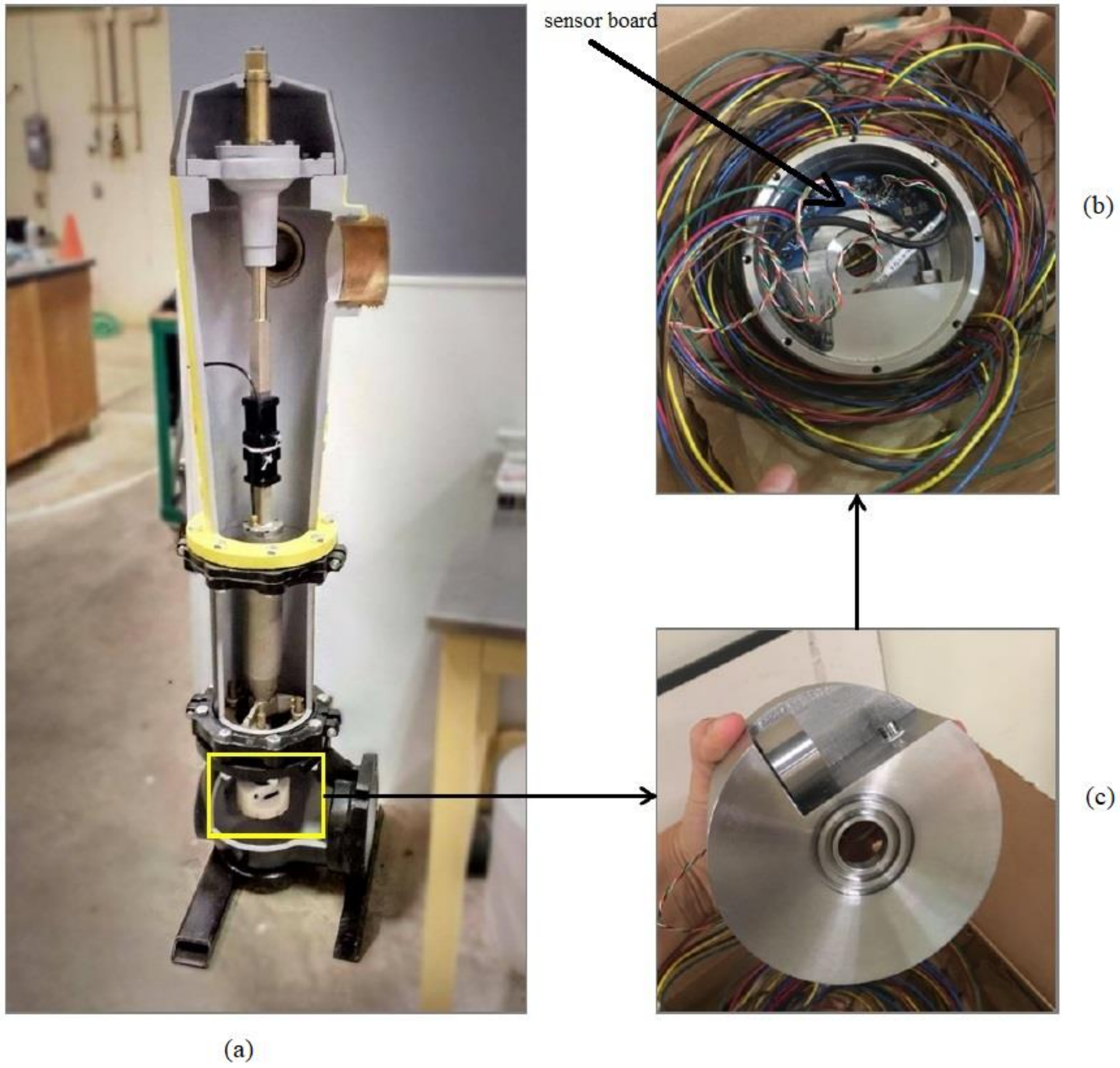


Figure 3.3 (a) Retrofitted fire hydrant showing the placement of a metal case, (b) sensor board inside a metal case and (c) A metal case

The main board and the sensor board along with the sensors are shown in the Figure 3.4 (a), (b) and (c) respectively. The sensors were mounted on a retrofitted fire hydrant, which is a Century® fire hydrant make.

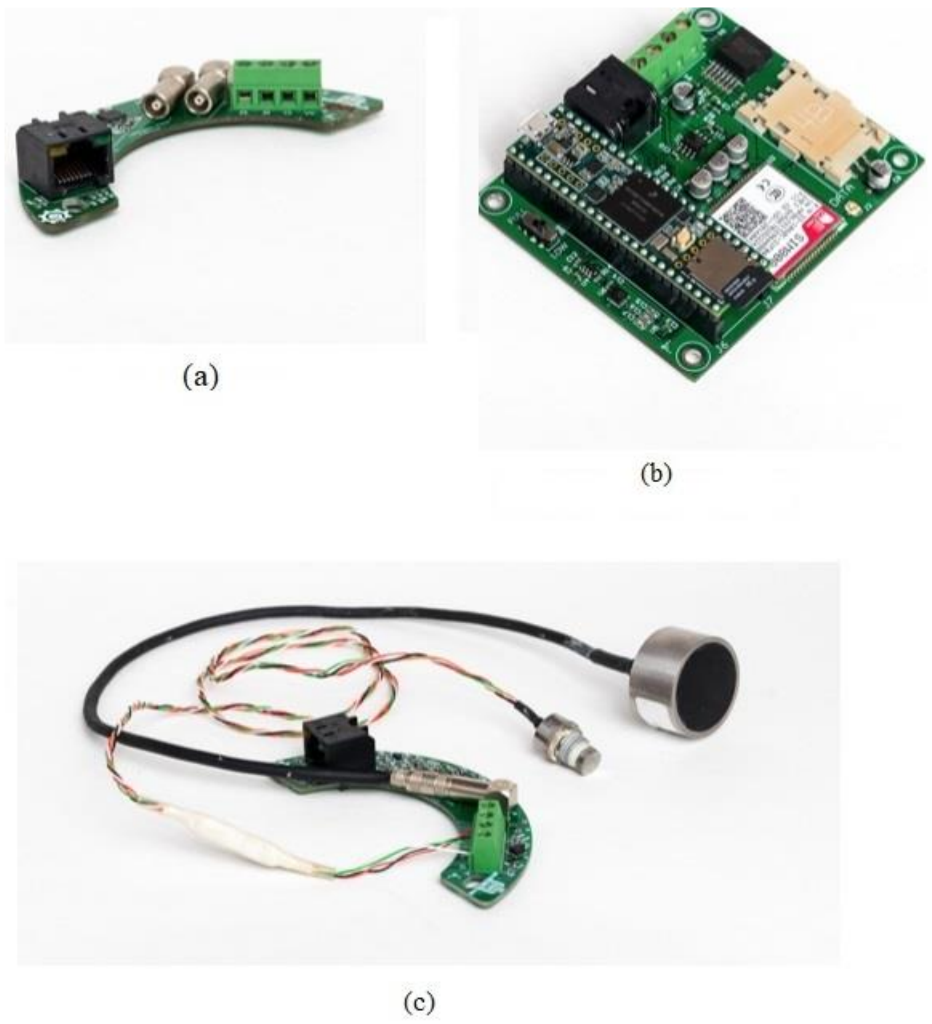


Figure 3.4 (a) Sensor board, (b) Main circuit board and (c) sensors.

There are two settings under which the data was collected in the field: ambient (normal operating condition) and leak (simulated flows from the hydrant) condition. Ambient case does not necessarily mean that no leaks exist; it is merely meant to represent a baseline from which anomalies are detected. Simulated leaks were created from instrumented and un-instrumented hydrant locations by opening an external valve with an attached flow meter to measure outflow. The aforementioned leaks were simulated using two hydrants which were located in the same municipal WDN, DMA11 in

ClairFields, Guleph, where the data was collected. Figure 3.5 shows the leak locations and the fire hydrants where the sensors were installed to collect the hydrophone data. The leak locations are located in 54 Paulstown and 22 Keys, which correspond to leak location 1 and 2, respectively. The leaks simulated on these hydrants correspond to various sizes, ranging from 200, 100, 50 or 25 litres/minute and each leak data was captured for 10 minutes on October 13th and October 21st 2018 on an SD card between 12:40 a.m. and 3 a.m. The ambient case data was captured on 13th, 19th, 20th and 21st of October, 2018, for 3 hours between 3 a.m. and 6 a.m. The sensor locations in the Figure 3.5 are the same fire hydrants with sensors shown in Figure 3.2.

The data on the SD card consisted of saved .csv files. Small amounts of data collected from live WDN was sent directly to the Nova Hologram cloud, which was then communicated to GCP, mainly to test and demonstrate the communication functionality. However, a majority of the data used for analysis was collected offline in order to reduce cellular modem costs as the intention was to stay within the free data usage limit provided by Hologram. Hence, emulators are used in this thesis that partly act as the data acquisition layer.

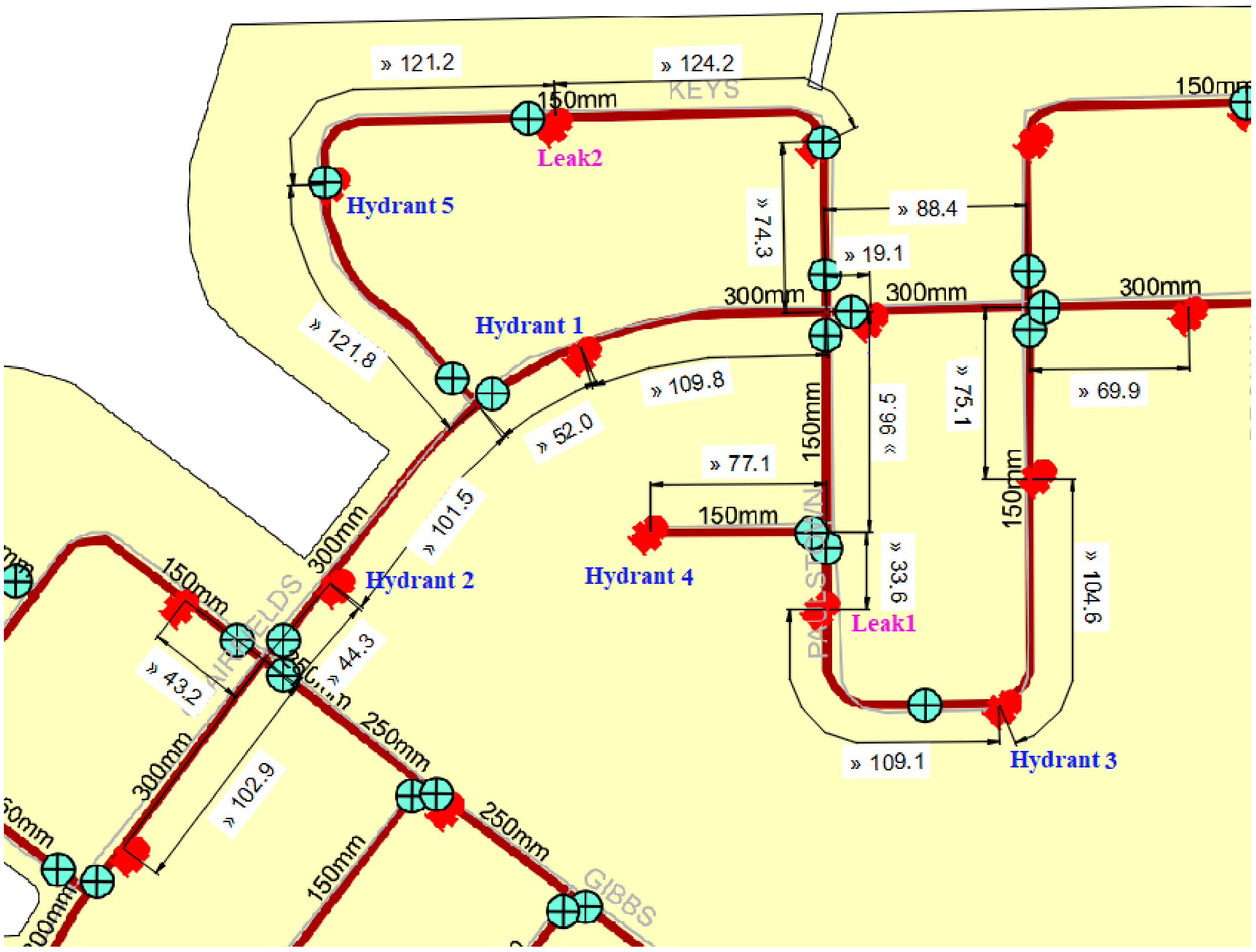


Figure 3.5 Hydrant locations 1 through 6 and Leak locations 1 and 2

After the data is acquired from the WDN by the sensors, the next step is to perform analysis, which includes: (i) data preprocessing, (ii) extraction of features and (iii) data analysis.

Data preprocessing is performed on a desktop computer, feature extraction is performed on emulators and machine learning algorithms (data analysis) are executed on GCP. The details of the each of the three subtasks of data analysis are explained in the following sections.

3.2.2 Data preprocessing

For this thesis, the sensor data are retrieved from the storage devices from hydrant locations and processed offline using a desktop computer. As described previously, this is mainly to reduce the cost of wireless transmission as a separate paid plan was not purchased for this research. However, it is important to note that this does not compromise the representativeness to actual field implementation as many of the intermediate steps are either emulated (only the end results are of concern) or simulated (results in intermediate steps are also of interest) in a software environment.

Data pre-processing includes the following two key operations:

1. Data conversion: Raw data which is obtained from the hydrophone is not in physical units and is in electrical units, milli-voltage (mV). This is converted to physical units of acoustic pressure Pascals (Pa), using the acoustic receiving sensitivity of the hydrophone and accounting for the pre-gain (25dB) of the in-line amplification circuit which lies between the hydrophone and the ADC module.
2. Data cleansing and sorting: Ideally, streaming data is captured without missing entries and in sequence. However, due to the internal hardware and software architecture on board the ADC and the main processing modules, on occasion there are missing entries and some data are stored out of sequence. Hence, both data cleansing and sorting operations need to be conducted prior to feature calculations.

Fully simulating both the processes is outside the scope of this work as this depends on the exact hardware and firmware configuration to be used in final deployment. Instead, an emulator (laptop computer) is used for data cleansing and sorting and all the programs were written and executed in MATLAB® environment prior to being sent for feature extraction.

3.2.3 Feature extraction

The next step in the data pre-processing step is feature extraction. Raw acoustic time series data are typically acquired at a high acquisition rate. In this application, the data rate used was approximately 4000 samples/sec (approximate because the clock and the actual hardware and software configuration determines the exact value at which data is acquired and typically there is a small variation from the prescribed target rate). Assuming 4 bytes of storage for a float this translates to 16kB/sec and for typical data acquisition time of 10 minutes at one hydrant location this results in10 mega-bytes of data per acquisition. This data size does not account for other parameters such as time and headers, which also have to be stored and transmitted in addition to the sensor data. Hence, it is easy to envision a scenario where the data transmission size could be of the order of hundreds of Mega bytes or even Giga bytes for a small city with several sensors simultaneously collecting data for 10 minutes.

Clearly, transmitting the entire raw time series data over a cellular wireless network is expensive. Considering that the final leak detection algorithms will employ statistical features and not the raw time series themselves (may be necessary for correlation-based techniques, but this is not pursued here), it is expedient to calculate such features at the sensor node. Such features are smaller in size and hence involve significantly reduced transmission overheads. Hence, those features which are non-redundant and statistically independent of one another, yet representative of data under consideration, need to be extracted from time series data and transmitted for processing. According to Cody, R. et al., 2018, although it is easy to detect large changes in the hydraulic conditions within pipes, it is challenging to detect events in the presence of background noise such as flow caused by water usage. To address this issue, the authors of the aforementioned study used spectral peak and root mean square (RMS) features derived from time series data for leak detection. Consistent with their findings, the same features are calculated on emulators (laptop and Raspberry Pi 3) and then readied for transmission. The features are calculated using a Python program executed on Raspberry Pi 3 (a single board computer) and the calculated features are then sent using the Python Nova Hologram software development kit (SDK) to the Hologram Cloud using Hologram 2G GSM connectivity. The features from Hologram Cloud are then transmitted to GCP using a HTTP POST request over the internet. The features (RMS and spectral peak) are also calculated on a laptop computer running Windows 10 and are then sent to GCP using a Python script via HTTP request over a WiFi connection.

The features are calculated on two types of emulators, Raspberry Pi and a computer laptop, to simulate the scenario where the sensor nodes are from two different platforms. Raspberry Pi runs on the OS Raspbien which is the Debien flavor of Linux OS, while the laptop computer runs windows 10 OS. Proper authentication is required prior to Hologram or the laptop emulator communicating the calculated features to GCP. Upon successful authentication by GCP, the features are then transmitted for downstream processing by GCP. There are two types of authentication methods: API key and OAuth 2.0. Nova Hologram Cloud supports only the API key authentication type, whereas the Python emulator supports both types of authentication. For this thesis, while API key is deemed sufficient, OAuth 2.0 is also implemented to test the overall architecture capabilities. The two authentication methods are described next.

(i) API key: API keys are simple encrypted strings such as 'AIzaSyAxLhEDXnG3h8ES6He76aW8wkFyOm6jiqE' which is provided by GCP. For API keys authentication, the emulator must include the API key in each HTTP POST request. Upon the receipt of this HTTP POST request, GCP checks for the correctness of API key and if it is correct, the HTTP request is processed further; otherwise an error is sent back to the emulator.

(ii) OAuth 2.0: For this type of authentication, the process is more involved. The first request from the emulator to GCP obtains credentials such as client ID and client secret which are known to both GCP and the emulator, which are then used to request an access token from the authorization server (Google). Such requests depend on the type of application. Javascript applications could request an access token using a browser redirect to the Google server, while a Python application like the emulator that is being used in this thesis requires GCP user (owner/admin of the GCP pipeline for data processing) to grant permission to the application that is requesting the access token by logging on to the computer where the emulator is running. An HTTP server is run as a part of emulator, at a predefined port 9004, to listen to the Google authorization server. When the GCP user grants permission, Google server sends the authorization code to the port 9004. Emulator uses this authorization code, exchanges it for access token with Google authorization server and sends the access token in an authorization header which is a secure way of sending the token to the Cloud function, which is GCP ingestion component. In OAuth 2.0 authentication process, only the first request goes through the authentication process and obtains the access token from GCP and the rest of the requests to GCP use the access token obtained by the first request. If the GCP user denies permission, an error message is sent by Google authorization server to the HTTP server running at port 9004. This authorization process is shown in the Figure 3.6.

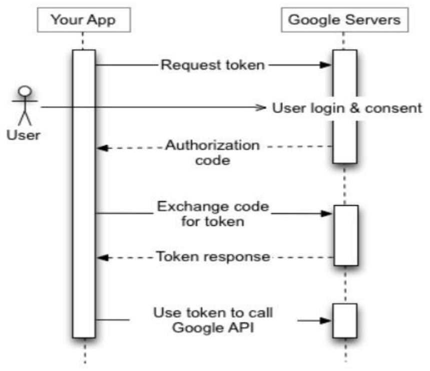


Figure 3.6 OAuth 2.0 process [7]

The main advantage of the API key authentication is that it is simple to implement whereas OAuth is relatively more involved. API key authentication however provides a static key (i.e., the API key does not change once it is generated by GCP), whereas OAuth provides a dynamic access token (i.e., a new key is generated by GCP for a new authorization process) in response to an application request. In general, a dynamic key is more secure compared to static key, especially over time. However, OAuth requires GCP user's (owner of the GCP resources that form a processing pipeline) presence to grant access to the requesting application (emulator) to access GCP. API key on the other hand, does not involve GCP user interaction with the requesting application. The static nature of API key can be changed by periodically changing the API key on GCP and distributing the newly generated key manually to authorized applications.

3.2.4 Communication protocol

GCP supports two communication protocols, HTTP and MQTT, to communicate with IoT devices (emulators in this case). A study conducted by Yokotani, T. et al., 2016 shows that MQTT is more advantageous over HTTP when used in the implementation of an IoT solution. HTTP however is simple to use and in this thesis, when the Python emulator on Raspberry Pi sends features to Nova

Hologram cloud, these are forwarded to GCP using HTTP only (Nova Hologram does not support MQTT at the time of writing this thesis). Hence, HTTP is used as the communication protocol between emulators and GCP in this thesis. The details of HTTP, MQTT and the study conducted by Yokotani, T. et al. 2016 are discussed below.

HTTP is an application layer protocol in the internet protocol suite, also known as transport control protocol/internet protocol (TCP/IP) stack, which is a standard model provided for communication in internet and computer networks. HTTP is the foundation of data communication for the world wide web where resources such as documents, files, pictures and videos, can be easily accessed on a web browser or other HTTP clients such as the Python HTTP client. HTTP provides communication based on standard request response model (Fielding, R. T. et al. 1999). In the request response model, clients such as a web application or a program requests the server (for example, to access a resource on the server) and the server responds to such requests by providing the requested resource. HTTP provides various methods to carry out different operations on the aforementioned resources. Some of the popular methods are GET (request to retrieve a resource), POST (request to accept a resource provided), PUT (request to create or modify a resource) and DELETE (request to delete a resource). In the leak detection application, emulators provide data to be ingested by GCP. Typically, these are called POST requests. Like HTTP, MQTT is also an application layer protocol in the TCP/IP stack which provides communication based on the publish-subscribe model. In MQTT, the publisher of a message does not send the message directly to the receiver, instead categorizes messages into classes called topics without the knowledge of the subscriber, while storing such messages on shared messaging queues on the MQTT server or the MQTT broker. Similarly, subscribers express interest in receiving messages of a particular class (topic) and they receive messages belonging to that topic from the MQTT server, or the MQTT broker.

Since HTTP operates over TCP/IP, such communications are deemed reliable. However, connections established between the client and the server for communication are not retained across subsequent requests. This causes overhead and consumption of network resources during communication (Yokotani, T. et al. 2016) as every HTTP request needs to establish connection between the client and the server and such connections include a client authentication process which consumes resources to carry out authentication. MQTT on the other hand requires a similar bandwidth as that of HTTP for establishing the initial connection between the MQTT broker and the MQTT client (either a subscriber or a publisher) as the communication requires authentication credentials to be sent from the MQTT client to the MQTT broker. Connection is retained across multiple requests and hence authentication credentials are not required to be sent in each request. This results in relatively less overhead and network bandwidth compared to HTTP.

Figure 3.7 shows the bandwidth used by HTTP requests and MQTT requests when 10 devices, 100 devices and 1000 devices are used (Yokotani, T. et al. 2016) for different payload (data) sizes. This clearly shows that for a given payload size, for fewer devices (say 10), the bandwidth for both HTTP and MQTT protocols are comparable. However, as the number of devices increase (say to 100 or 1000), MQTT uses far less bandwidth compared to HTTP. Since an IoT solution typically involves a large number of data sources, MQTT is a better choice for communication compared to HTTP.

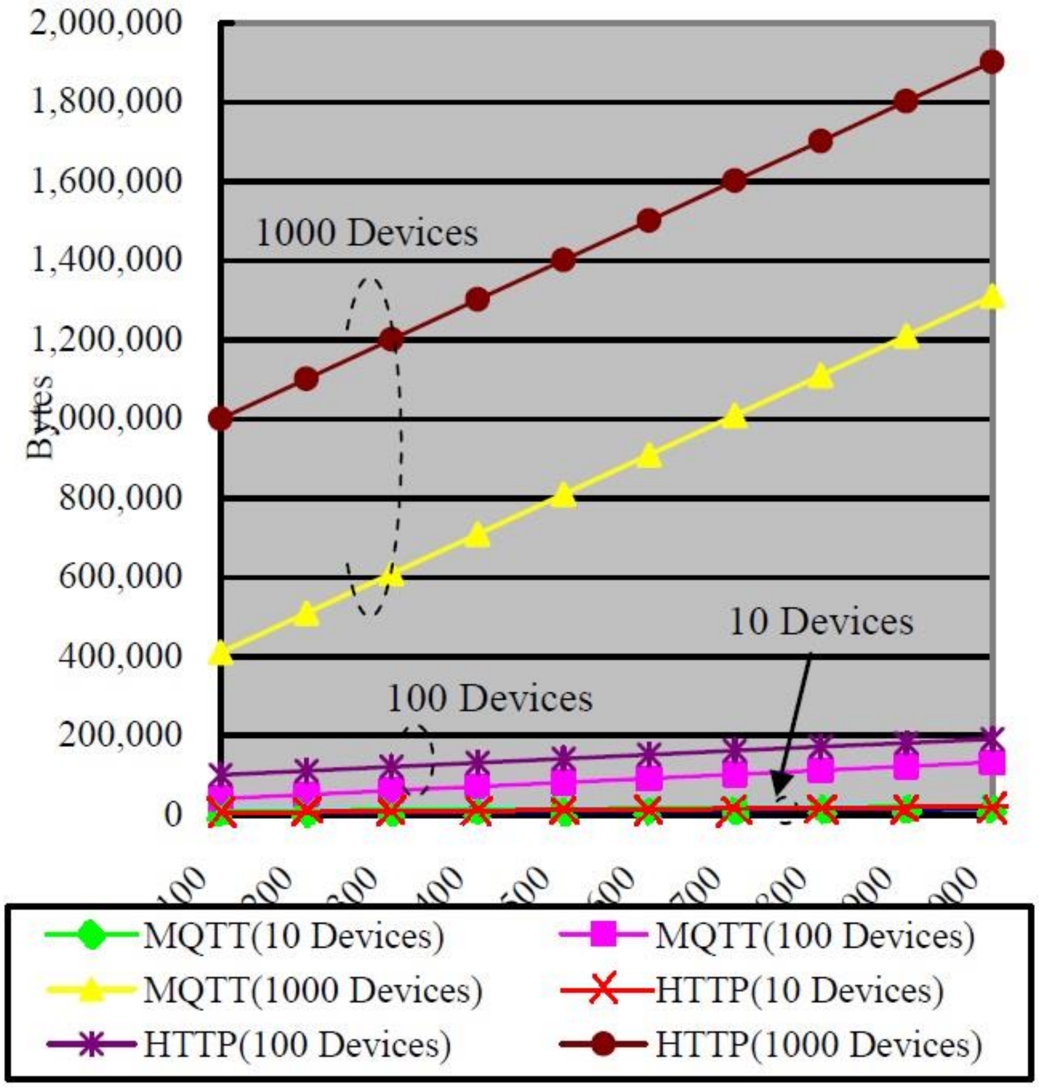


Figure 3.7 Required bandwidth according to variable devices and payload sizes. (Source: Yokotani, T. et al. 2016)

For the same aforementioned reason (that HTTP does not retain connection across requests), HTTP takes longer time for communication compared to MQTT. Table 3.1 shows the required duration for each protocol between the start and completion of communication. It is clear that the first MQTT request (#1) takes nearly the same duration as HTTP, while the second MQTT request takes significantly less time compared to HTTP, both in terms of average and maximum times between the start and completion of the request.

		MQTT #1	MQTT #2	HTTP
No od Round Trips		4	1	5
Required duration between start and completion	MAX	800ms	200ms	1s
	AVE	400ms	100ms	500ms

Table 3.1 Comparison between MQTT and HTTP with respect to communication duration (Source: Yokotani, T. et al. 2016)

For the purposes of this thesis, the number of devices is small and the tests are conducted in controlled environments. HTTP is widely supported by many applications including the Nova Hologram Cloud and GCP used in this thesis. Hence, for demonstration purposes HTTP is used, although in practical full-scale implementations of this technology involving a large number of devices MQTT would be the preferred protocol option.

3.3 GCP

GCP [8] is offered as “Software as a Service” (SaaS) which depends on internet to provide its service. This means GCP provides its users access to its infrastructure such as hardware resources including storage hard disks, servers, networking, computing and software resources such as operating systems, software applications and databases via the internet. Google locates its infrastructure at various physical locations called data centres around the world and manages those resources, which includes all the physical connectivity of the computers, upgrading the computers, backups, upgrading the operating systems and resolving networking issues. Users can rent services offered by GCP and pay only for the resources used and for the duration those resources are used. As well, GCP provides easy to use APIs which can interface with all the infrastructure offered and provides the above mentioned infrastructure at a scale required for IoT implementation, including data collection, processing and analysis of large amounts of data. The overall process pipeline and various components of GCP used in this thesis are illustrated in Figure 3.8. To provide readers with a complete picture of the overall architecture used other components namely DAL and NL are also shown in the Figure. The four different stages namely, ingestion, storage, analysis and visualization of results are discussed next.

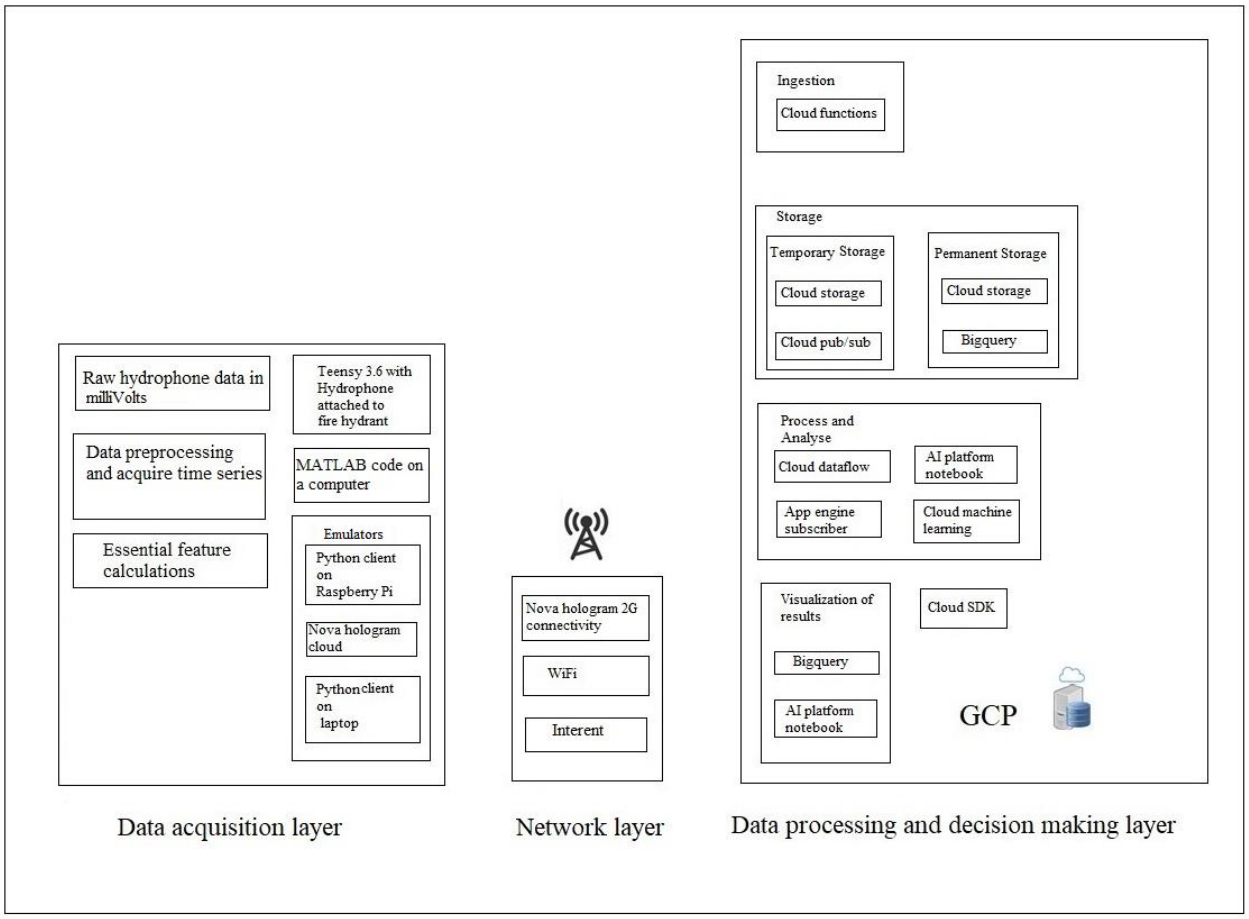


Figure 3.8 The process pipeline used in this thesis with various GCP sub-components.

3.3.1 Ingestion

Cloud Functions

Data from the source, i.e., from the Python client or from the Hologram Cloud (using HTTP POST requests as described previously), is first verified by the GCP to ascertain the legitimacy of the source, for the validity of the data type, and for the validity of the request. These checks are carried out at the ingestion stage using the *Cloud Function* subsystem of the GCP. *Cloud Function* is central to ingestion tasks in the process pipeline and acts as an event listener which responds to events such as HTTP trigger, an incoming message on *Cloud Pub/Sub* for a given topic, a file upload on *Cloud Storage* or a log change in *Google Analytics* API. Since HTTP is used as a trigger from the data source in this thesis, *Cloud Function* responds to such HTTP requests and carries out the following tasks:

1. Respond to the HTTP event.
    a. Check for HTTP POST requests.
    b. Check for authentication of the data source.
    c. Perform data validation.
    d. Form an output JSON data consisting of the data provided by the emulators and encode it into a byte string format.
2. Invoke other services.
    a. Send the encoded output data to a temporary storage component or an API of GCP such as *Cloud Pub/Sub* using HTTP POST request.
3. Write back to the event generator.
    a. Send a response to data source (emulator) that triggered *Cloud Function*.

The above tasks are illustrated in Figure 3.9 as well.

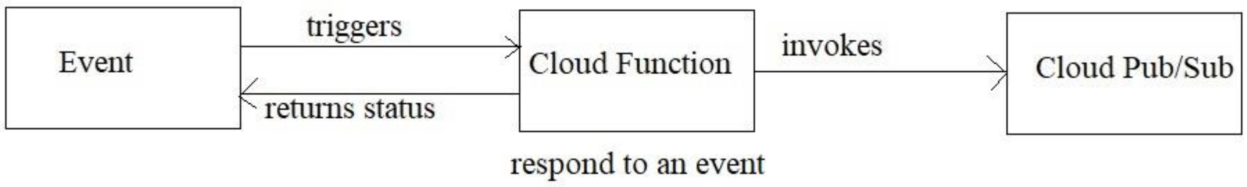


Figure 3.9 Pipeline showing ingestion of data by *Cloud Function*

Since *Cloud Function* is managed by GCP, it can scale from a few invocations to millions of invocations without the need for users to manage the underlying infrastructure. Instead, users write a simple event handler function on *Cloud Function* API using a programming language such as Python to respond to an event such as the HTTP trigger from an emulator. In this thesis, Python 3.7.1 is used on *Flask* web framework to write an event handler which performs all the ingestion tasks mentioned earlier. *Flask* is a small and powerful web framework which is provided as a default framework by GCP and is used with Python to build web applications quickly.

3.3.2 Storage

Once the data source and the data are validated, the data is stored for further analysis. There are two types of storage used, temporary and permanent, depending on the type of processing which is carried out on the data. Both the types of storage are discussed below.

Temporary storage: Temporary storage components used for this thesis are *Cloud Pub/Sub* and *Cloud Storage*. Temporary storage is utilized when programs (e.g., data processing code on *Cloud Dataflow*) need a location to stage the binary executable files during execution. Temporary storage used for this application is the *Cloud Storage*. When the data is streamed, which is the kind of data generated continuously by the data sources, such data is incrementally processed using stream processing techniques without the need to access all the data at any given time. In the current application, emulators transmit this incrementally, one set of features at a time at a pre-determined frequency, say every second. Temporary storage used in this case is *Cloud Pub/Sub*, which temporarily stores the streaming data and forwards it to the data processing unit *Cloud Dataflow*.

Note that *Cloud Storage* is also used as permanent storage and hence this is discussed in the permanent storage section. *Cloud Pub/Sub* details are provided next.

Cloud Pub/Sub

Cloud pub/sub acts as a temporary storage while providing real-time messaging service which allows applications to send and receive messages. *Cloud pub/sub* is an auto-scalable GCP-managed event ingestion and delivery system, which is suitable for stream analytics pipelines. It provides many-to-many asynchronous messaging, i.e., many senders can send messages to many receivers without senders and receivers being present at the same time. This provides a decoupled system which is flexible for both senders and receivers to post and receive messages at their convenience. *Cloud Pub/Sub* servers are distributed at various Google's data centers around the world. Different instances of a cluster which is a logical grouping of machines that share the same local network and power are located in data centers. *Cloud pub/sub* is a global service that is divided into mainly two parts: data plane and control plane. Data plane servers are responsible for moving messages from the Publisher to the Subscriber. Control plane servers are responsible for assigning Publishers and Subscribers to a data plane. Publishers and Subscribers are unaware of the physical locations of the *Pub/Sub* servers and they can publish and subscribe messages across large geographical distances. *Cloud Pub/Sub* follows the following rules to deliver messages:

1. Publisher or a sender of a message can send messages to *Cloud Pub/Sub* only using a named resource called "topic". For example, an HTTP client can send telemetry to *Cloud Pub/Sub* using a topic called "sensor".
2. The receiver of a message or a Subscriber can receive messages from *Cloud Pub/Sub* topic only if a subscription is created by that subscriber to that topic. Subscription is a named entity

that subscriber creates for a topic that associates Subscriber to that topic. Only those messages that are published after the subscription to a given topic is created are available to the Subscriber of that topic.

3. Topics can have multiple Subscribers as well as multiple Publishers. However, subscription belongs to a single topic. *Cloud Pub/Sub* delivers every message to every subscription at least once.
4. Each message is by default kept for the maximum of 7 days. If there is no subscription available before the expiration time, that message will no longer be accessible. The message retention time can be configured between 10 minutes to 7 days. If there is no subscription available to a topic and if Publishers publish messages to that topic, then those messages will not be delivered even after a new Subscriber creates a subscription to that topic. Subscriptions by default expire after 31 days of inactivity. The deletion clock is reset every time there is Subscriber activity. The expiration time can be configured and subscriptions can be made persistent regardless of activity.
5. When the *Cloud Pub/Sub* is waiting for an acknowledgement from a Subscriber for a sent message before the deadline, the message is called an “outstanding message”. Once the message is sent to a Subscriber, that Subscriber sends an acknowledgement back to *Cloud Pub/Sub* within a limited time, called acknowledgement deadline which is defined by each Subscriber. Once the deadline passes, *Cloud Pub/Sub* resends the message to that Subscriber to ensure at least one-time delivery. *Pub/Sub* delivers messages in the order in which they were published. However, sometimes the messages may be delivered out of order or delivered more than once. So, Subscribers should make sure that such messages are handled properly.

When the Publisher publishes a message to *Cloud Pub/Sub*, this is written to storage. Subscribers can retrieve messages either by a *Pull* or a *Push* mechanism. This type can be configured by the Subscriber anytime. In a *Pull* subscription, the Subscriber initiates the request to obtain messages from the *Pub/Sub* server. The sequence of operations that occur in a *Pull* request is described in Figure 3.10. The Subscriber initiates a request to the *Pub/Sub* server for messages for which *Pub/Sub* responds with a message or error and an acknowledgement ID. The Subscriber then acknowledges the delivery to *Pub/Sub* using this acknowledgement ID.

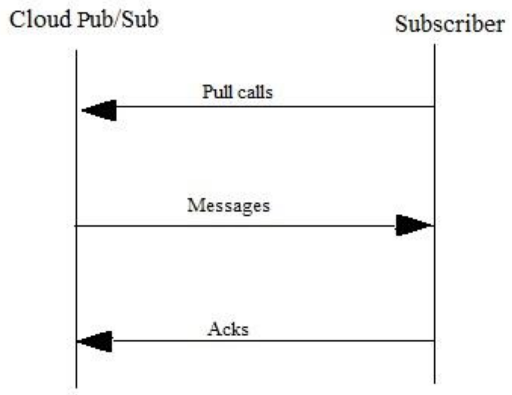


Figure 3.10 Pub/Sub pull subscription

In a Push subscription, *Pub/Sub* initiates an HTTP request to send messages to the Subscriber application, which then sends acknowledgement back to *Pub/Sub* for both success and failure. This is shown in the Figure 3.11. In the case of failure, *Pub/Sub* resends the message.

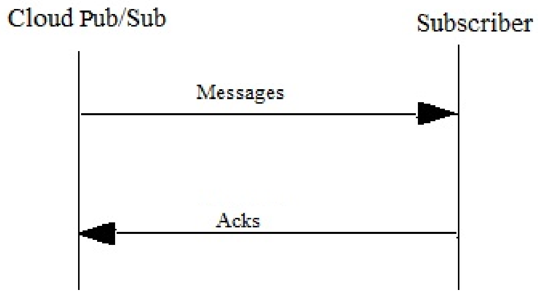


Figure 3.11 Pub/sub push subscription

In this work only the Pull subscription is used by *Cloud Dataflow* and the topic used is called "hydrophone". Default settings are used for subscription inactivity duration and message retention time. The publisher is the *Cloud Function* and the Subscriber is *Cloud Dataflow* (described later).

Permanent storage: Permanent storage is used to permanently store large amounts of data. The data stored in permanent storage can also be used for batch processing, which involves waiting for the entire data to arrive completely before the actual processing of the data begins. The type of permanent storage used depends on the specific application and two scenarios are possible within permanent storage, as follows:

1. *Cloud Storage* is used to store a large amount of data in the file format. For example, in this thesis, training data sets and the training model are stored on *Cloud Storage* permanently for further use by the *Cloud Machine Learning* engine (*Cloud ML* engine) for downstream analysis and decision making.
2. *BigQuery* is used to store a large amount of data in the in a table format. Such storage facilitates SQL-like queries to work with that data. This type of permanent storage is used to store the processed data and the results of processing (in case of stream processing) and data to be processed (in case of batch processing).

Cloud Storage is explained in this section whereas *BigQuery* is explained under visualization of results, as this is used in that stage as well.

Cloud Storage

Cloud Storage could be both temporary or permanent in nature and provides storage and retrieval of any amount of data at any time for a range of scenarios such as temporary storage for program execution, serving website, storing data for archival and disaster recovery. All the data on *Cloud Storage* belongs to a project. Buckets are the basic storage containers that hold data. When a Bucket is created in the Storage Class, three properties need to be specified: unique global name, a location where the Bucket and its contents are stored and a Storage Class. There are four Storage Classes available: multi-regional, regional, nearline and coldline storage.

(i) Multi-regional storage: In multi-regional storage, data is stored in multiple geographical locations. This is suitable for storing data that is accessed frequently such as those serving website content.

(ii) Regional storage: The data is stored only in a specific region instead of having redundancy. Since the data is stored only in a specific region, this storage cost is low.

(iii) Nearline storage: This storage is highly durable, low cost and suitable for storing infrequently accessed data. It requires minimum of 30-day storage. This is suitable for data back-up and disaster recovery storage.

(iv) Coldline storage: It is similar to nearline storage except that this data is available within seconds not hours or days. Again, this is suitable for data back-ups and disaster recovery.

By default, if the Storage Class is not specified, multi-regional storage class is used to create a Bucket. Data stored in the *Cloud Storage* is redundant and the data is stored in at least two geographic locations to ensure redundancy. Data inside each resource in the *Cloud Storage* is opaque to *Cloud Storage*. Each object stored in *Cloud Storage* is immutable which means that it cannot change between its creation and deletion. This means that once a file is created, users or applications cannot undertake “append” or “truncate” operations. The object can only be overwritten, i.e., a new version of the same object can be created.

Data is encrypted prior to it being stored on *Cloud Storage*. There are two types of encryptions, client-side encryption (while the user uploads data) and server-side encryption (encrypted before the data written to a persistent disk). No additional configuration is required and when read by an authorized user such data is transparently and automatically decrypted. In this thesis, *Cloud Storage* is used as a temporary location for holding intermediate files and directories for *Cloud Dataflow*. It is also used by *Cloud ML* engine to store machine learning models and input data. Regional storage class was specified when a storage Bucket was created.

Cloud SDK

GCP services can be accessed in three different ways: (i) using programming languages such as Python and Java; (ii) using GCP website console.cloud.google.com; and (iii) using *Cloud SDK*.

Cloud SDK provides a set of command line tools which include `gcloud`, `gsutil`, and `bq` and core libraries. These commands and their use are given below.

`gcloud`: This manages interaction with GCP APIs and authentication.

`bq`: This is used for working with data in *BigQuery*.

`gsutil`: This is used for working with *Cloud Storage*.

`Core`: Libraries used internally by *Cloud SDK*.

In this work `gsutil` is used to upload data files and machine learning models from the local machine to *Cloud Storage*.

3.3.3 Data processing and analysis

GCP enables data to be processed in batch as well as in streaming mode. In batch processing, processing of data is performed upon the arrival of all data, whereas in stream processing or real time processing, data is processed as and when the data arrives. Figure 3.12 illustrates the sequence of GCP subcomponents invoked during data processing in stream (the black arrows) and in batch processing (red arrows) modes.

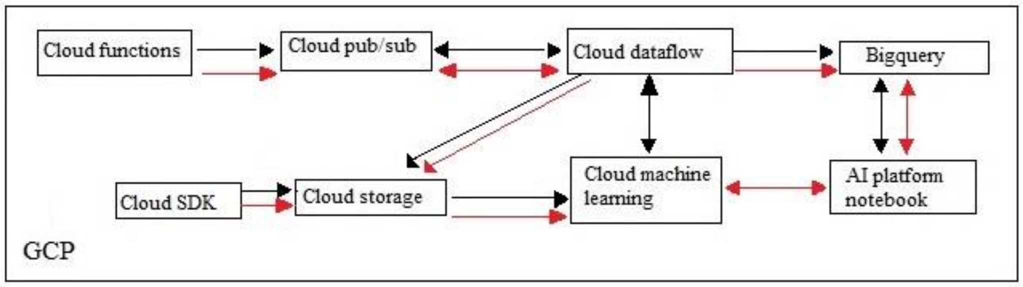


Figure 3.12 The sequence of GCP sub-components invoked for stream and batch processing.

In the context of the current leak detection application, acoustic data stored in *Cloud Pub/Sub* in the case of stream processing and the data stored in *BigQuery* in the case of batch processing are processed to detect patterns and anomalies in the data. *Cloud Dataflow* is used for stream and batch processing and the AI *Platform Notebook* is used for fetching data from *BigQuery* and calling *Cloud ML* for batch prediction and visualization of result (from both batch and stream processing). For both batch and stream processing, training of data, creating the training model, and model prediction are performed using the *Cloud ML* engine, which is a component provided by GCP. The same GCP components are used in both stream and batch processing. In stream processing, *Cloud ML* is invoked by *Cloud Dataflow* whereas in batch processing, *Cloud ML* is invoked by *AI Platform Notebook. Cloud Dataflow* and *Cloud ML* engine are explained next and the *AI Platform Notebook* is explained in the visualization of results section.

Cloud Dataflow

Cloud Dataflow is a managed data processing pipeline which works on the *Apache Beam* framework. *Apache Beam* is a unified open source programming model which enables developers to develop both batch and streaming pipelines, with relatively minor modifications to the settings of the pipeline definition. The default mode of processing is batch and *Apache Beam* allows parallel processing of data as well and supports Java and Python software development kit (SDK). *Cloud Dataflow* manages all the low-level tasks of distributed processing such as sharding (partitioning) data sets on the disks and coordinating individual worker computers which perform data processing tasks, while providing useful abstractions in the form of *Apache Beam* which insulates all such details from users. The abstractions provided by *Apache Beam* to work on a large-scale distributed data processing are described as follows.

(i) Pipeline: The tasks of reading the input data from the source, transforming the data and writing the output to a sink are all encapsulated in a programming object called, Pipeline. Dataflow programming begins with creating this `Pipeline` object, which is then used for creating data sets called `PCollection`.

(ii) `PCollection:` This represents datasets in *Dataflow* and can hold datasets of a fixed size or varying size.

(iii) Transforms: The input data is processed in *Dataflow* and the processing operation is called Transformation. Transformation takes one or more `PCollection` as inputs and performs operations specified by the user on each element in that collection and produces one or more `PCollection` as outputs. There are various Transforms which can be applied on data and those used in this thesis are discussed towards the end of this section.

(iv) Runners: Runners execute the software `Pipeline` objects. The two Runners available are `DirectRunner` and `DataFlowRunner`. `DirectRunner` executes the pipeline code on a user's local machine (often used for testing the *Dataflow* code before it is deployed on cloud) whereas `DataFlowRunner` executes the pipeline code on the cloud. By default, Virtual Machines (VMs) that are allotted for running *Dataflow* jobs are automatically deleted when the job is completed. This means that any persistent disks attached to a particular VM are deleted automatically when the job is completed and the files stored on those disks are deleted as well. So, in order to avoid data loss, users need to provide (in the command prompt while executing the *Dataflow* job) `--staging_location` and/or `--temp_location` which specify the *Cloud Storage* paths where the intermediate files from pipeline execution should be stored.

Cloud Dataflow subscriber on its own requests *Cloud Pub/Sub* for any new messages and processes such messages if there is any new message. This asynchronous *Pull* provides higher throughput as the application does not have to block a new message. Synchronous *Pull* is also available, where new messages are blocked until the Subscriber finishes processing the old message. Unless specified, asynchronous Pull is used and this setting is not changed in this thesis. Also, in this thesis, the data source used is *Cloud Pub/Sub* and the sink used is *BigQuery*. Apache Beam with Python SDK is used with a streaming pipeline for the data, with `DataFlowRunner`. For stream processing, the following data Transformations are applied:

- Read the data from *Cloud Pub/Sub.*
- Decode the data from input format (byte string format) to the format readable by *Cloud Dataflow* (utf-8).
- Extract data values from JSON data embedded in utf-8 string.
- Predict the class of data using one-class support vector machine (OCSVM).
- Create a record containing the results as well as the input data in the format (JSON encoded in utf-8 string) which is compatible with *BigQuery*.
- Write the record to *BigQuery.*

For batch processing, the data is streamed through *Cloud Dataflow*, stored in *BigQuery,* followed by the execution of the one-class support vector machine (OCSVM) algorithm (written in Python programming language) on the entire data set using *AI Platform Notebook* and *Cloud ML engine*.

Cloud ML engine

This is a GCP managed service which provides training and prediction tools for developers to build and run machine learning models. *Cloud ML* engine supports the Python programming language used on popular frameworks such as Tensorflow, Scikit-learn and XGBoost to perform machine learning on data. The *Cloud ML* work-flow is shown in Figure 3.13, where the filled boxes correspond to GCP managed services.

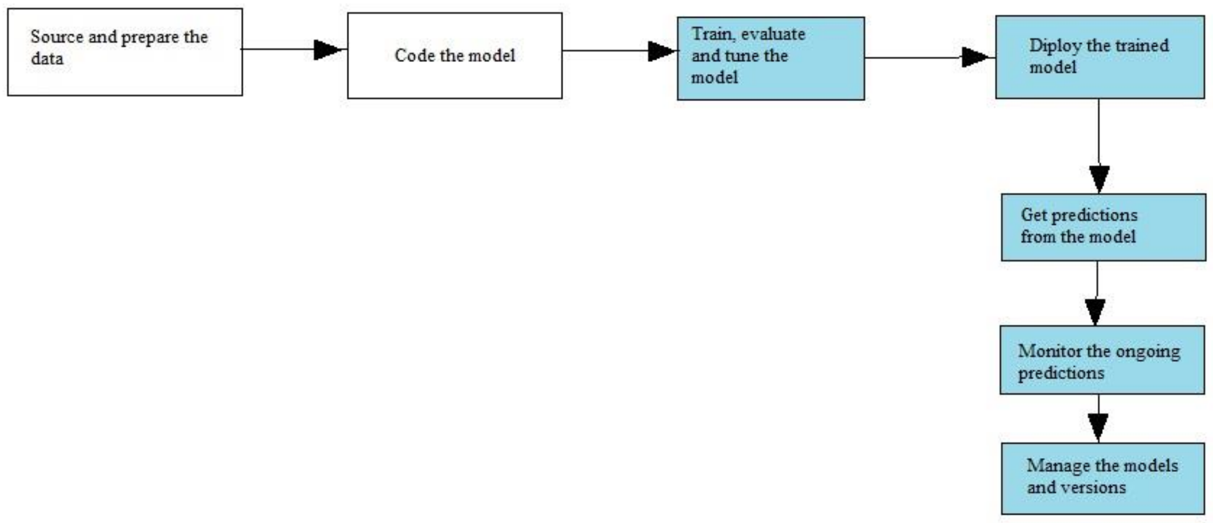

Source and prepare the data
Code the model
Train, evaluate and tune the model
Diploy the trained model
Get predictions from the model
Monitor the ongoing predictions
Manage the models and versions

Figure 3.13 Cloud ML work-flow, where the filled boxes represent GCP managed services.

The steps involved in developing a production ready machine learning model which can be used for prediction on stream as well as batch data are:

1) Collect and prepare data.
2) Develop a user model, train it, evaluate it and tune the hyper-parameters.
3) Deploy the above trained model on *Cloud ML* engine.
4) Send prediction requests to this trained model; this can be either batch or online (for streaming data) requests.
5) Monitor the predictions.
6) Manage the model and model versions.

As discussed previously, in this thesis, both batch and online predictions are used. *Cloud Dataflow* sends a stream (online) prediction request to *Cloud ML* whereas, the user defined Python code on *AI Platform Notebook* sends a batch prediction request on the data stored in *BigQuery*. For prediction OCSVM with Gaussian radial basis functions (RBF) kernel is used by Cloud ML.

3.3.4 Visualization of results

Once the data processing stage is completed, the results of processing are visualized using *AI Platform Notebook,* which in turn uses *BigQuery*. The details of *BigQuery* and *AI Platform Notebook* are provided next.

BigQuery

BigQuery belongs to both storage and visualization stages of DPDML in the overall architecture. Storing and querying a large amount of data produced by a large number of IoT devices can be quite challenging, time consuming, and expensive without the right hardware and infrastructure. To address this, GCP provides *BigQuery* which is a Petabyte scale low cost data warehouse that is fully managed by GCP. *BigQuery* provides super-fast SQL style querying interface to define and manipulate data records. This API can be accessed using multiple tools such as GCP web UI, command line tool or through REST API calls using client libraries provided in various languages such as Java, .net, or Python. *BigQuery* manages storing the data, including compression, encryption, replication, performance tuning and scaling. It supports several input formats, comma separated values (CSV), JSON, AVRO source format, amongst others. Once imported, data is converted to internal columnar format. Every column in addition to its value, stores two numbers, definition and repetition level which helps in keeping track of the nested structure. Data can be loaded on to *BigQuery* either through the batch load or through the stream load.

In batch load, the data is loaded from files. The default source format for batch loading is CSV and these files can be loaded from *Cloud Storage* or from other Google services. *BigQuery* supports both UTF-8 and ISO-8859-1 data formats. By default, UTF-8 encoding is enabled. In the case of stream load, data is added to *BigQuery* one record at a time. Client libraries of supported languages such as

Java, Go, Python, php, C#, Ruby can be used to stream data into *BigQuery*. In this thesis, *Dataflow* creates tables and stream loads the data along with the results of inference using the Python client library. The data is sent as JSON in utf-8 format. *BigQuery* is also used by *AI-Platform Notebook* to query the batch data, perform batch prediction on the queried data, query the stream processing data and to display the stream processing ML results.

AI Platform Notebook

AI Platform Notebook is used for both data analysis as well as for result visualization activities and enables users to use VM instances equipped with latest data science and machine learning libraries such as the suite of Python and Deep Learning packages, Tensorflow and Pytorch. It provides Jupyterlab instances through a protected, publicly available notebook instance URL. Jupyterlab is an open-source web application that allows users to create, edit, execute code. In this thesis, *AI Platform Notebook* is used for both batch processing of the data and visualization of the results from both batch and stream processing. For batch processing, *AI Platform Notebook* uses *Cloud ML* engine which in turn uses Scikit-learn Python libraries which are equipped with the OCSVM module to enable prediction. The results of prediction and data analysis are described in the next chapter.

3.4 Data flow diagram

The overall data flow between the various components of IoT architecture used in this thesis for leak detection (as discussed in previous sections in this chapter), is shown in the Figure 3.14.

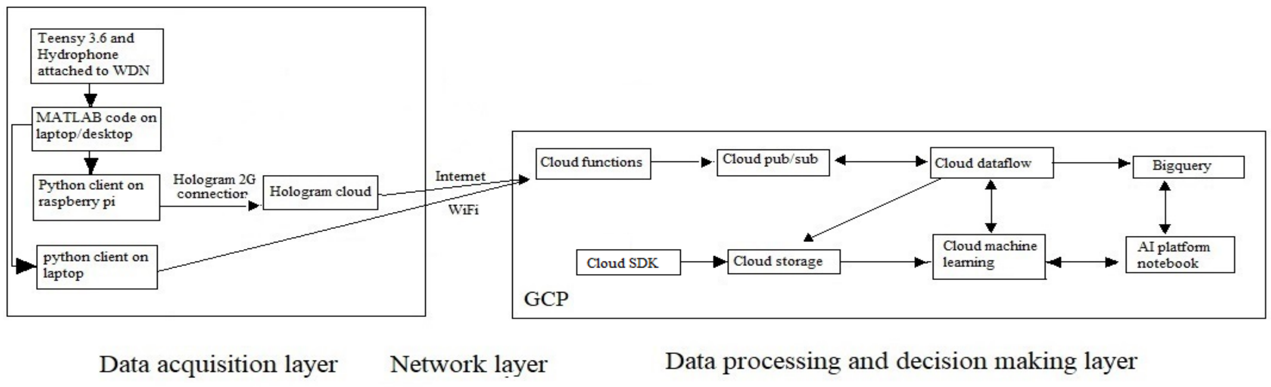


Figure 3.14 Data flow diagram between various IoT components used in this thesis.

The raw hydrophone data was acquired using a Teensey 3.6 microcontroller and transferred to a laptop. Following this, a custom MATLAB code was used to extract the time series. Then, using emulators, features were extracted and sent to GCP over the network layer. On GCP, the features were ingested by *Cloud Function* after verifying the data source and forwarded to *Cloud Pub/Sub* for temporary storage and then sent to *Cloud Dataflow*. In stream processing, each feature set was sent to *Cloud ML* for prediction. OCSVM model was stored *on Cloud Storage* and linked to *Cloud ML*. *Cloud ML* used the trained model from *Cloud Storage* to perform prediction. *Dataflow* also used *Cloud Storage* to store the executable code for running the *Apache Beam* framework. For stream processing, the results of prediction along with the original data from *Cloud Pub/Sub* are stored in *BigQuery*, following which the results are displayed on *AI Platform Notebook*. In batch processing, the data received by *Cloud Dataflow* is forwarded to *BigQuery* and processed by *Cloud ML* and the results were displayed on *AI Platform Notebook*. In batch processing too, *CloudML* used the model stored on *Cloud Storage*.

4 GCP Implementation and Leak Detection Results

4.1 Introduction

This chapter provides the implementation details of data acquisition, data pre-processing and feature extraction along with data analysis and the results of ML engine prediction (leak detection). The implementation details of the remaining IoT components in the pipeline discussed in Chapter 3 such as emulators (Raspberry Pi3, Nova Hologram cloud and laptop emulator) and GCP (*Cloud Function, Cloud Pub/Sub, Cloud Dataflow, BigQuery, Cloud ML, AI Platform NoteBook, Cloud Storage*) are provided in Appendix A.

4.2 Data acquisition layer

Seven hydrants were used for the data acquisition activity, where five instrumented hydrants (one through five from Figure 4.1) were used for collecting acoustic pressure data and two hydrants were used to simulate leaks. Data collected from the instrumented hydrants were categorized into ambient and leak cases, where "ambient" represents the baseline condition prior to simulating leaks by opening a valve on a hydrant and "leak" corresponds to the case where a valve was opened to atmospheric pressure. Leak sizes achieved during this study corresponded to: 200 (size A), 100 (size B), 50 (size C) and 25 (size D) litres/minute. These flow amounts were verified by attaching an external flow measurement device to the hydrant used for simulation. Ambient data collected on the

13th, 19th, 20th and 21st of October 2018, Leak 1 (location 54 Paulstown) data collected on the 13th and 21st and Leak 2 (location 22 Keys) data collected on the 21st of October are used for further analysis in this thesis.

4.3 Data pre-processing

The data collected from the WDN was converted to CSV format, which was then transferred to a personal computer (laptop, 64 bit, windows 10 operating system and Intel Core, 2.8GHz, 8GB RAM) for pre-processing. After pre-processing, the file containing the hydrophone data—converted to physical units of pressure (in Pascals)—is sorted in an ascending order and stored (using a portable storage device) in ASCII format and readied for ingestion by the emulators. This manual offline process could be easily automated either by transmitting the raw data wirelessly or by using a conventional computer at the sensor location. Both options were deemed too expensive for pilot demonstration purposes, while the latter is associated with an additional problem of securing the computer on site and providing remote power. In practical scaled implementation, this process needs to be automated and warrants additional investigation; however, this is not a serious impediment in the context of the central objectives in this thesis.

Typical pre-processed time series for ambient, size A, size B, size C and size D leak1 (Figure 4.1) data collected from the hydrant3 (Figure 4.1), are shown in the Figure 4.2.

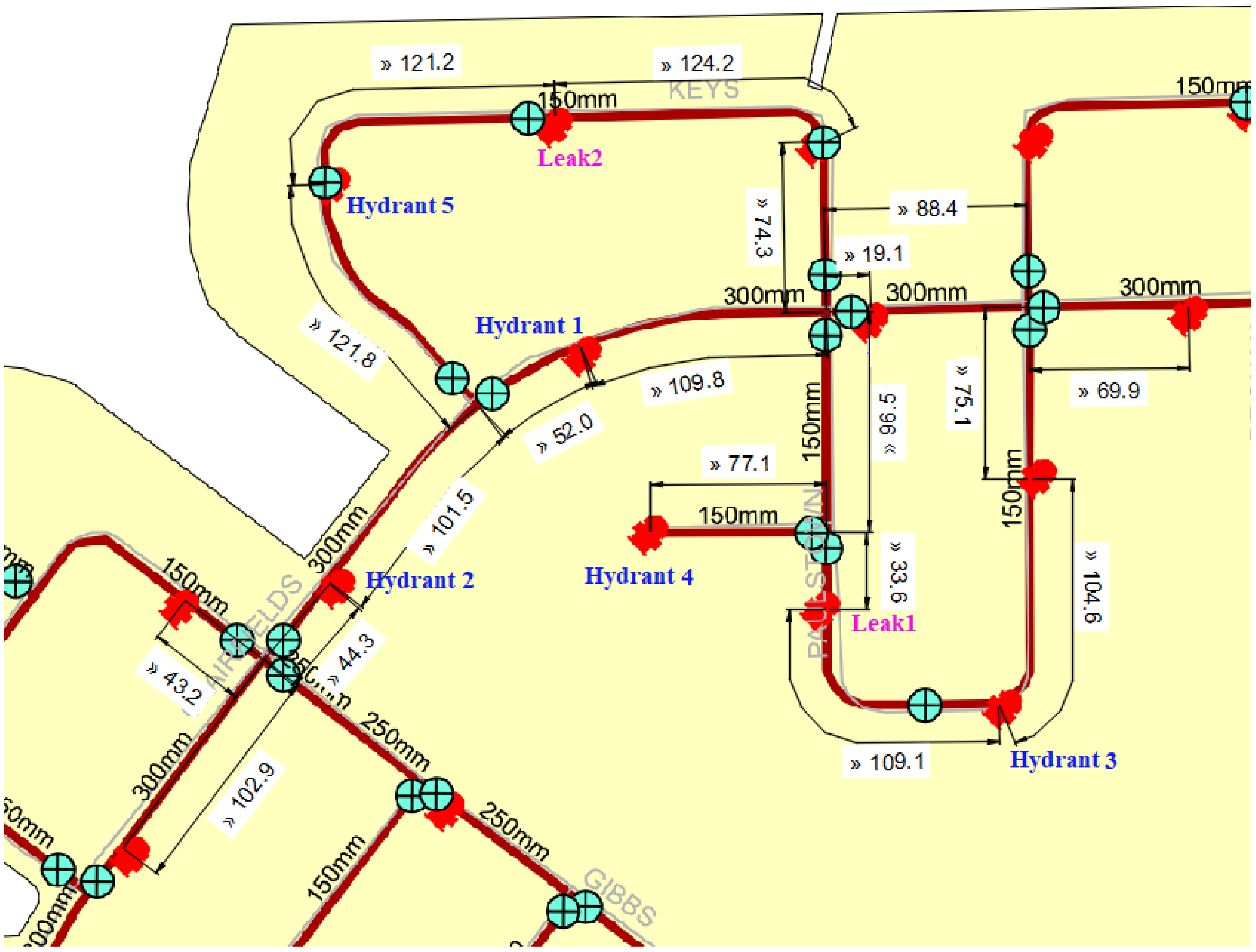


Figure 4.1 Hydrant locations 1 through 6 and Leak locations 1 and 2

From Figure 4.2, it is clear that larger leaks (size A) are associated with higher amplitude compared to smaller leaks (size B, size C and size D) and ambient data and in general the amplitudes are in the descending order starting with the largest leak size. It is important to note here that the amplitudes are not stationary over time and hence traditional techniques such as control charts in tandem with thresholds cannot be used in this application. In other words, the range of operational variability typically masks variations due to leaks if the correct baselines are not employed. Hence, more sophisticated anomaly detection techniques have to be employed for leak predictions. Furthermore, since transmitting raw time series is associated with significant bandwidth requirements, instead, only the features extracted from the time series data are sent over the network to GCP.

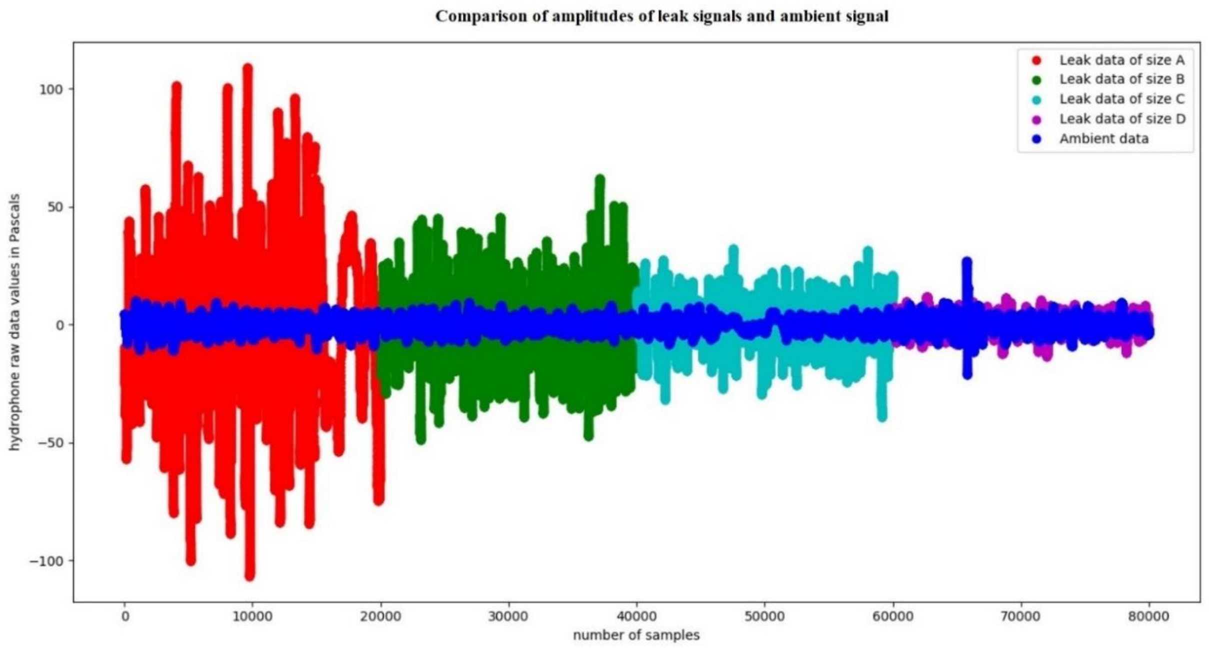


Figure 4.2 Typical time series for ambient and simulated leaks of size A, size B, size C and size D leak1 data collected from Hydrant 3.

4.4 Feature extraction

One of the main pre-processing steps is the extraction of features and the associated calculations (algorithmic steps to calculate the features). Such feature calculations could be undertaken either on the micro-controller board (Teensy 3.6) or on an external computer by ingesting the hydrophone data from the microcontroller unit. Direct implementation on the micro-controller was not undertaken for this thesis and the feature calculations were emulated on a computer laptop and on a Raspberry Pi 3 as though they could be processed after ingesting the data from a storage device on-board the data acquisition system.

Two features, RMS and spectral peak are deemed the most useful for anomaly detection, as reported in a previous study (Cody, R. et al. 2017). The programming language used in this thesis for feature calculations was Python. RMS provides a measure of the energy contained within a fixed time segment of time series data and is calculated using the equation:

$$x_{rms} = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right)^{1/2} \tag{1}$$

where, x_i represents the time series data value and n represents the length of the samples used to calculate the RMS, which is 4000 (sampling frequency is 4 kHz, which corresponds to a 1 sec RMS). While RMS provides a notion of the time-averaged energy content, spectral peak is the maximum value in the array of elements obtained by carrying out discrete Fourier transform (DFT) on the time series data. DFT is the discrete version of the FT which transforms a time signal (or a discrete sequence) from the time domain representation to its frequency domain representation. This transformation (discrete case) is mathematically defined by (Cooley, J. W. et al., 1965):

$$y_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \qquad k = 0, \ldots, N-1 \tag{2}$$

where, N is the number of elements in the sequence, x_n represents the time domain values (hydrophone data), $e^{i2\pi/N}$ is a primitive Nth root of unity. DFT converts equally spaced samples from the time domain to the frequency domain, where the number of transformed points in the frequency domain remains the same as in the time domain. Time complexity of DFT calculation of n data points by using the definition given in the mathematical equation (2) is $O(n^2)$. This means that for large transforms (n is large), time taken by DFT is very large (quadratic time). To reduce the computing time of large transforms, more complex but less time-consuming algorithm, the fast Fourier transform (FFT) is used. FFT is an efficient algorithm which computes DFT (Cooley, J. W. et al., 1965 and Press, W. et al., 2007) on a pre-specified number of fixed frequency points (2^N, where N is an integer) and the time complexity of FFT is $O(n \log n)$ (quasilinear time) which is far less than $O(n^2)$ for $n>1$. Since FFT is faster and efficient, this is used as a feature in this thesis.

RMS and spectral peak features are calculated on the data collected for every second. For example, for a time series containing 160,000 rows of hydrophone data, the number of feature rows is 40 for each feature (40 x 2 matrix). The features calculated on the time series data were sent at an interval of 1 second from the emulator to GCP using HTTP POST request. The implementation (coding) details of emulators and GCP cloud components are discussed in Appendix A.

4.5 Sensitivity analysis

Sensitivity analysis was performed on various acoustic time-series data sets. The first data set consisted of training (ambient only) and testing data (ambient and leak data) collected on the 13th of October 2018. The second data set consisted of training (ambient only) and testing data (ambient and leak data) collected on the 21st of October 2018. The third data set contains training data (only ambient) from the 19th, 20th and 21st of October and testing data (both ambient and leak) from the 21st of October 2018. The main motivation for this analysis is to investigate the sensor locations which were sensitive to Leak 1 and locations which were sensitive to Leak 2. The details of the number of features used for this sensitivity analysis towards Leak 1 are provided in the Table 4.1 and those corresponding to Leak2 are provided in Table 4.2.

From the results in Table 4.1 and Table 4.2, it is clear that significant compression of data is achieved by calculating the features rather than using the raw time series on their own. For example, the total number of feature sets from Table 4.1 and Table 4.2 taken together is nearly 85,000, which is a significant reduction from the original 340x10^6 discrete time values in the original data. This reduction results in a reasonably small number of data values which need to be processed for the ensuing sensitivity analysis.

1st dataset	number of features sets					
Hydrant	ambient training (13th of October, 2018)	ambient testing (13th of October, 2018)	size A	size B	size C	size D
1	4762	180	179	179	179	179
2	4762	180	180	180	180	180
4	1596	180	181	181	182	181
5	1596	180	180	180	180	180

Table 4.1 The number of features sets used for sensitivity analysis for Leak 1.

2nd and 3rd datasets	Number of feature sets						
Hydrant	ambient training (19th and 20th and 21st of October, 2018)	ambient training (21st of October, 2018)	ambient testing data (21st of October, 2018)	size A (21st of October, 2018)	size B (21st of October, 2018)	size C (21st of October, 2018)	size D (21st of October, 2018)
1	11505	547	180	180	60	60	180
2	12208	547	180	179	178	60	180
4	12120	502	180	193	194	194	193
5	12258	502	180	180	178	180	180
6	12661	1540	180	179	179	180	179

Table 4.2 The number of features sets used for sensitivity analysis for Leak 2.

To test the sensitivity of the instrumented hydrant locations to Leak1, the first data set was considered which consisted of October 13th ambient data and Leak 1 data for all sizes. The feature graphs for sizes A, B, C and D for all the hydrants for which the data was available are shown in Figures 4.3, 4.4, 4.5 and 4.6. From a simple observation of the feature clusters, it readily apparent that the instrumented Hydrant 3 is senstive (the ambient and leak classes are separable) to Leak 1 for sizes A, B and C. The instrumented Hydrant 4 is also sensitive to Leak 1, but only for large leak sizes such as A and B. Hydrant 3 is sensitive to Leak 1 (sizes A, B and C) because Hydrant 3 is the closest hydrant to Leak 1 (at a distance of 109.1m) on the map (Figure 4.1). The next hydrant that is closer to Leak 1 is Hydrant 4 (at a distance 110.7m from Leak1 (figure 4.1)). So Hydrant 4 is also sensitive to Leak 1(but only for sizes A and B). So it can be concluded that the data collected at an instrumented hydrant that is the closest to a leak shows the highest sensitivity towards that leak and sensitivity towards a leak is directly proportional to distance of the hydrant location where the data is collected.

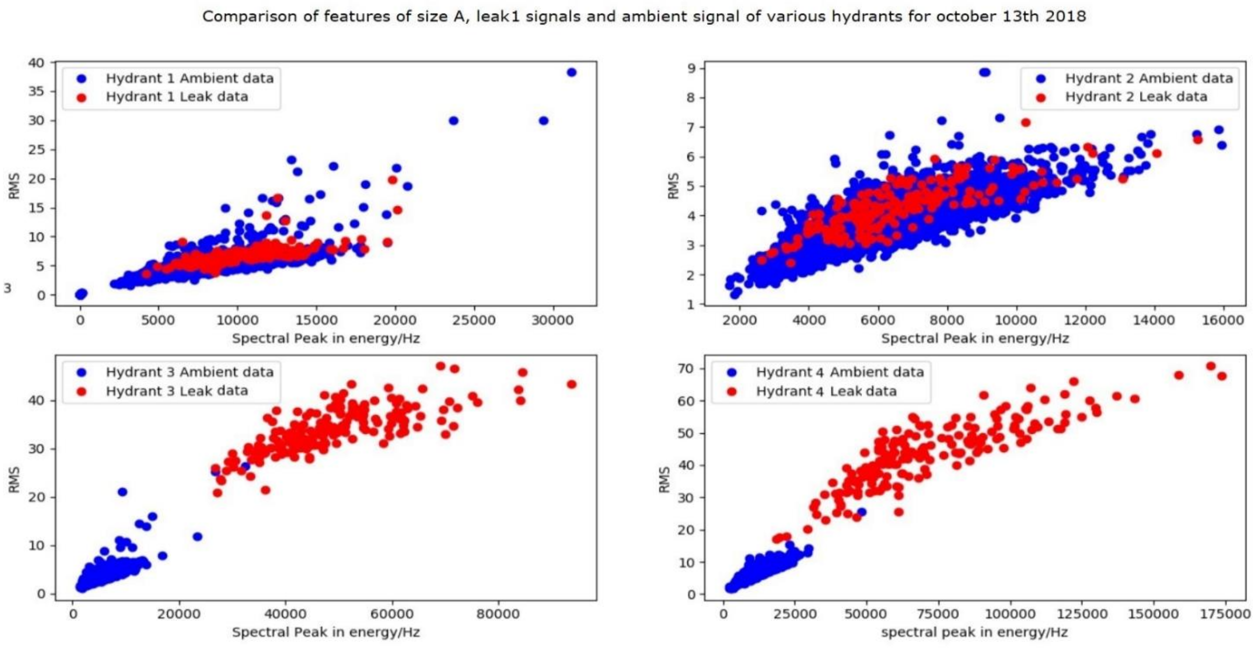


Figure 4.3 Feature graphs of ambient and leak (size A) data from October 13th, for various hydrants.

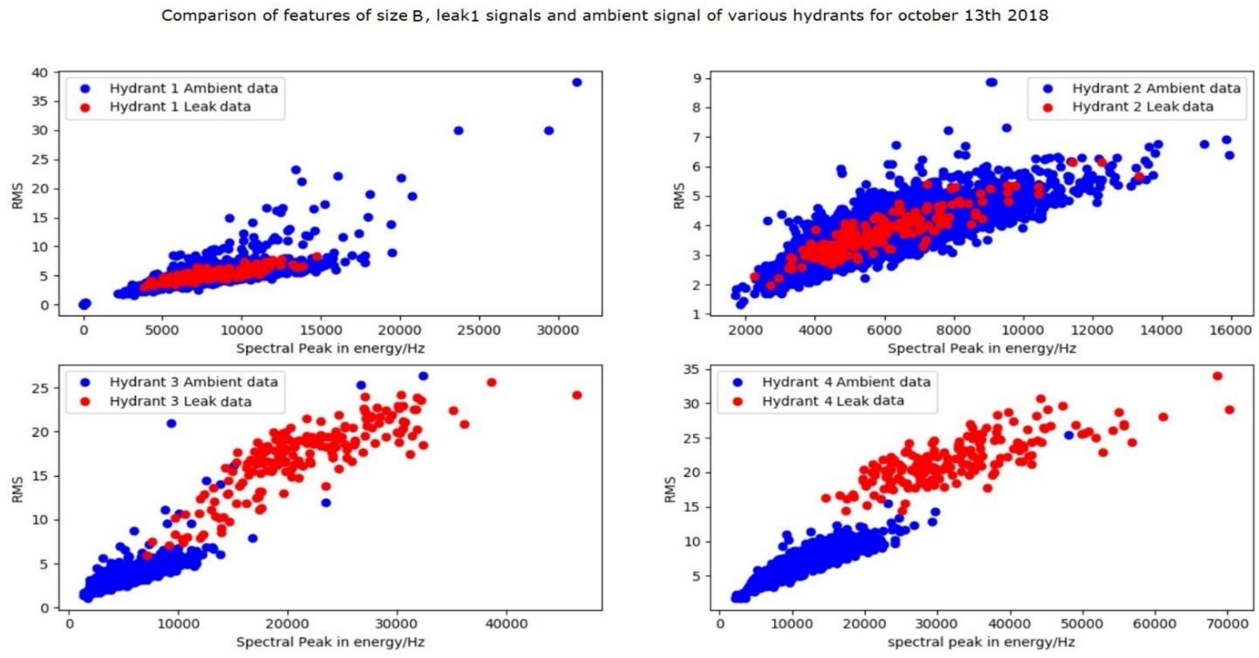


Figure 4.4 Feature graphs of ambient and leak (size B) data from October 13th, for various hydrants.

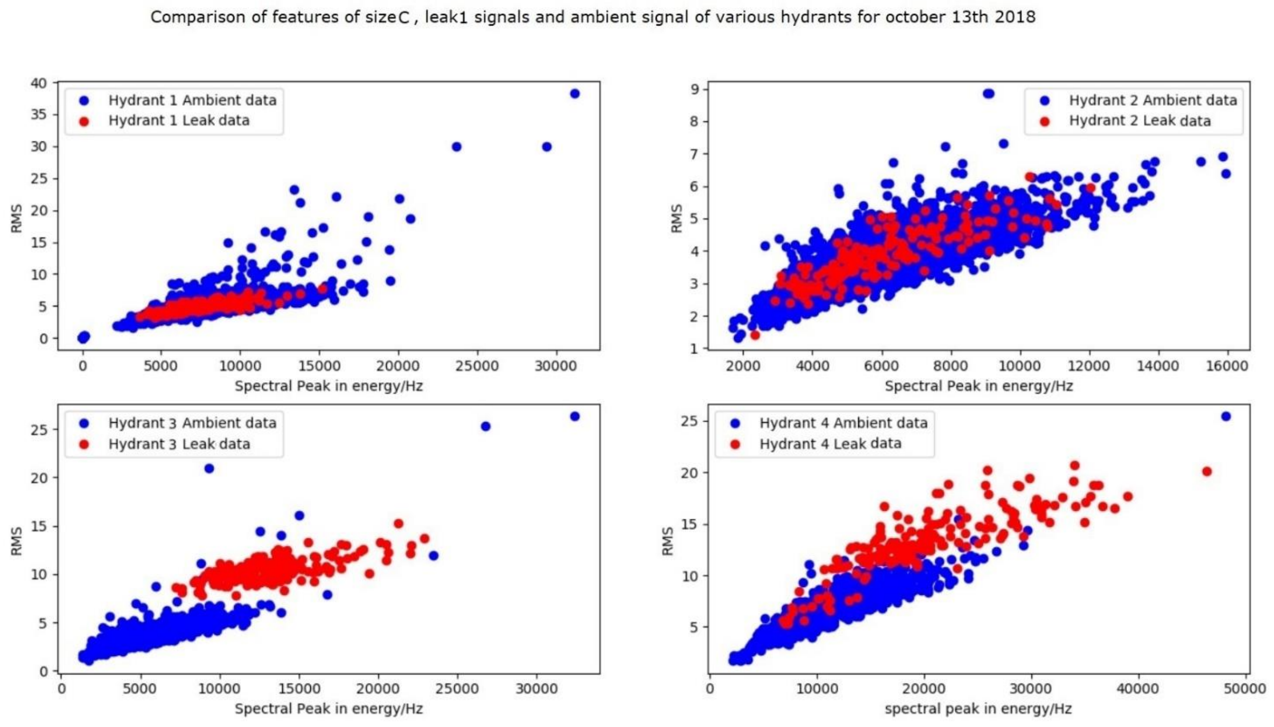


Figure 4.5 Feature graphs of ambient and leak (size C) data from October 13th, for various hydrants.

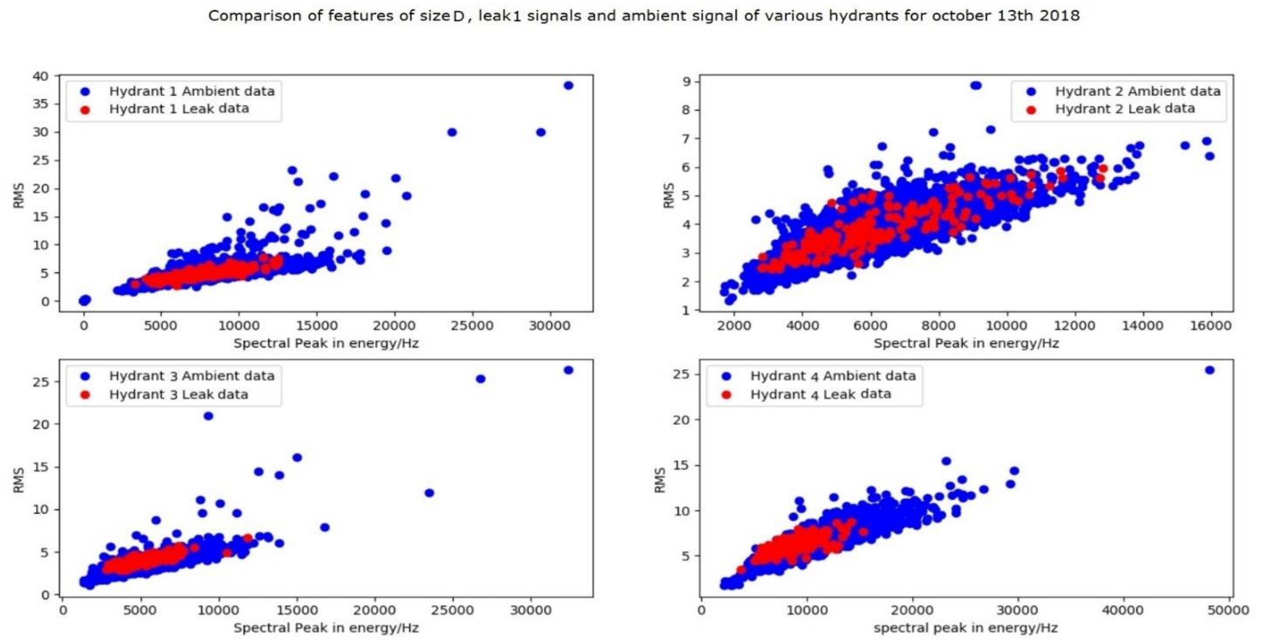


Figure 4.6 Feature graphs of ambient and leak (size D) data from October 13th, for various hydrants.

The sensitivity analysis was carried out on Leak 2 as well to confirm that the distances of the instrumented hydrants from a leak play a major role in determining whether the features extracted (from the data collected) from those hydrants show sensitivity towards that leak.

Figures 4.7, 4.8, 4.9 and 4.10 show the two-dimentional feature plots of ambient data and Leak 2 data conducted on the October 21st 2018, for sizes A, B, C and D at various instrumented hydrant locations. It can be seen from the feature maps that the instrumented Hydrant 5 is senstive to leak 2 (as the ambient and leak cases are clearly seperable into two distinct classes), sizes A, B and C. However, none of the instrumented hydrants are sensitive to size D Leak 2. From the above discussion, it is clear that the features RMS and spectral peak can be used to observe the sensitivity of instrumented hydrants towards various leaks and that the data collected from instrumented hydrants that are closer (upto a distance of around 120 meters) to the leaks show more sensitivity towards that leak.

Given that the instrumented Hydrant 3 is more sensitive to Leak 1 and the instrumented Hydrant 5 is more sensitive to Leak 2 compared to other hydrants, the results of prediction using OCSVM algorithm (using Sklearn Python code library) are presented for the instrumented Hydrants 3 and 5 in Table 4.3 and Table 4.4, respectively. These results are discussed later in this Chapter after evaluating the prediction model. The process of training the OCSVM algorithm, evaluating the performance of each model (through grid search) and obtaining the prediction results are provided next.

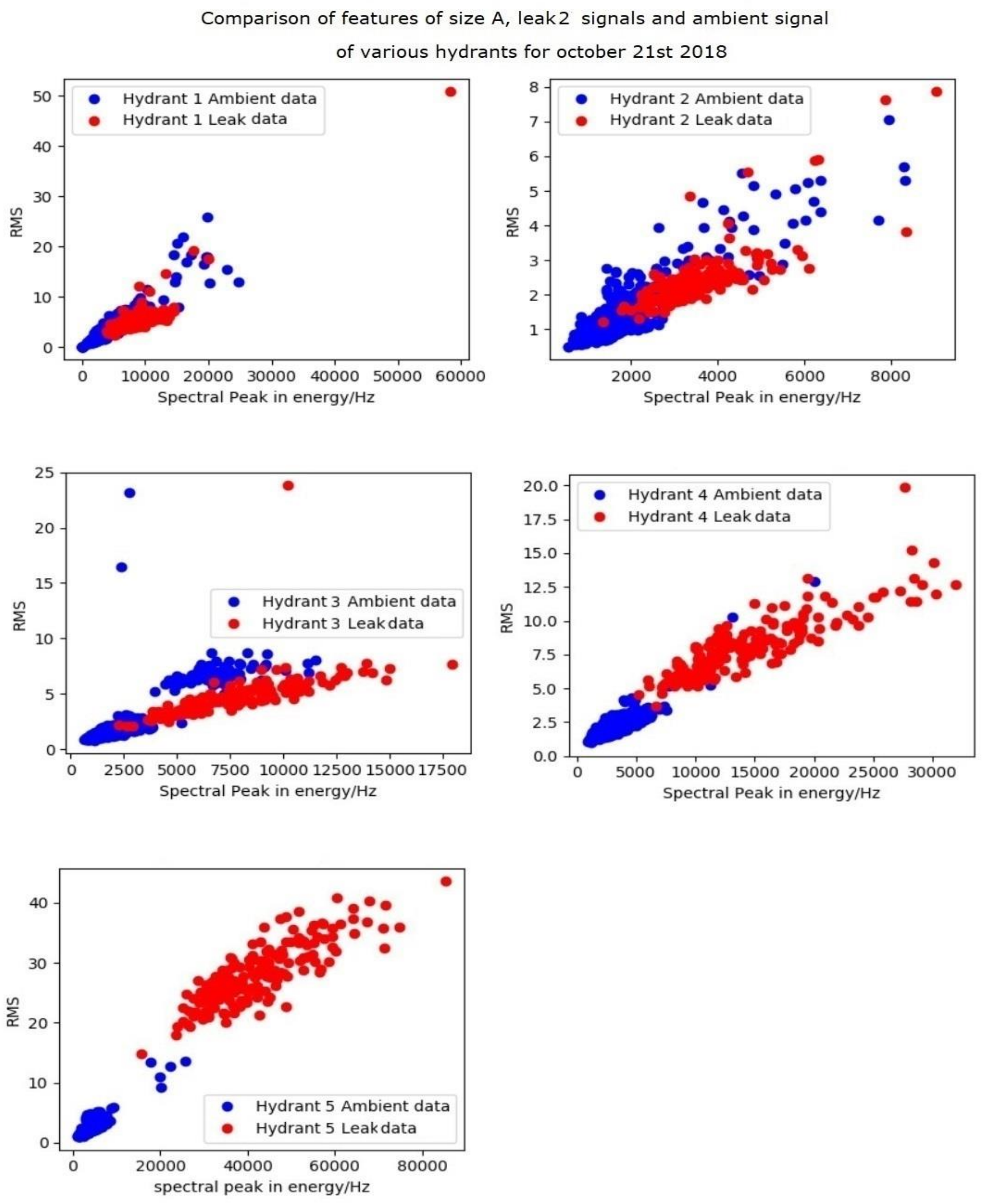


Figure 4.7 Feature graphs of ambient and leak (size A) data from October 21st, for various hydrants

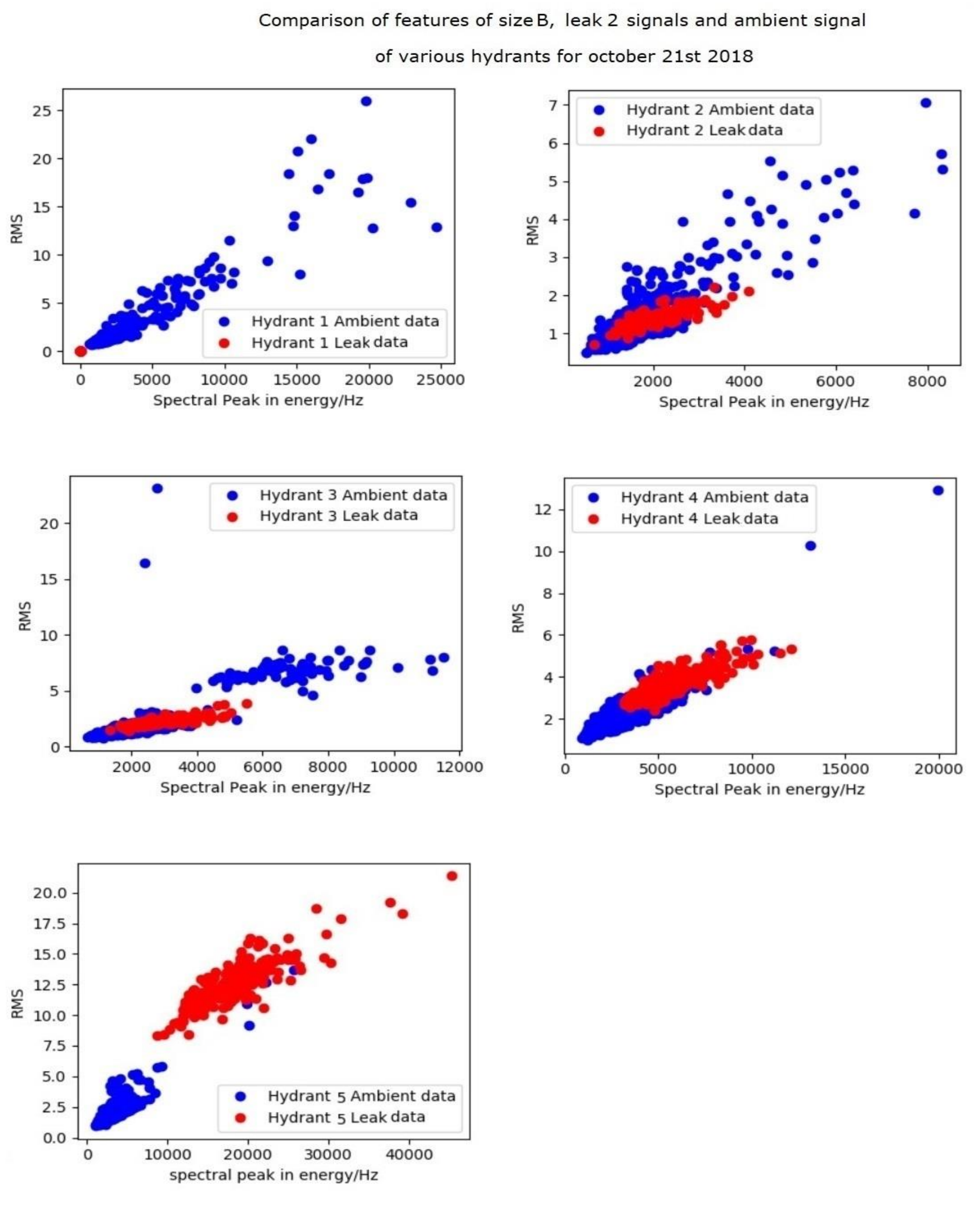


Figure 4.8 Feature graphs of ambient and leak (size B) data from October 21st, for various hydrants

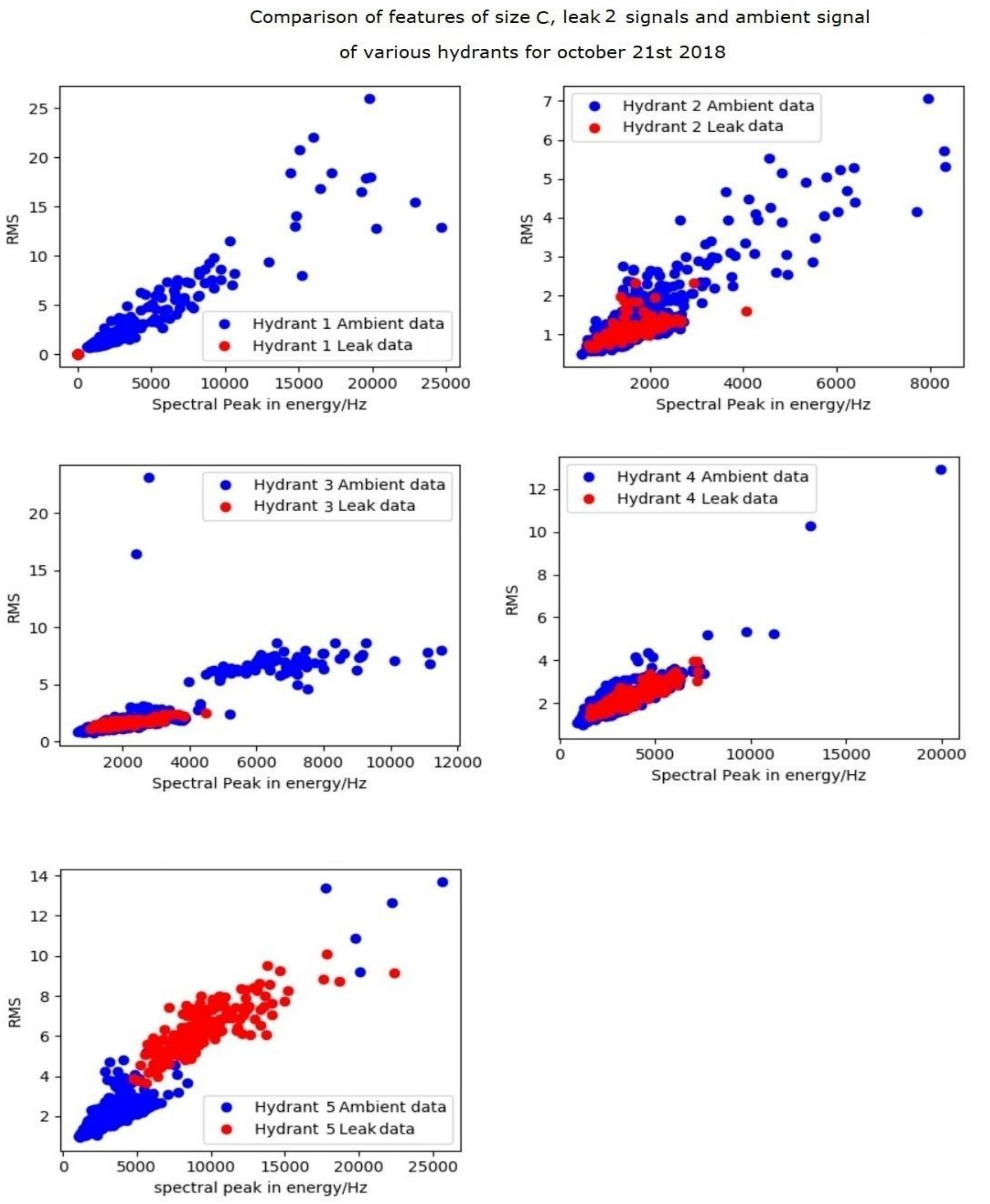


Figure 4.9 Feature graphs of ambient and leak (size C) data from October 21st, for various hydrants.

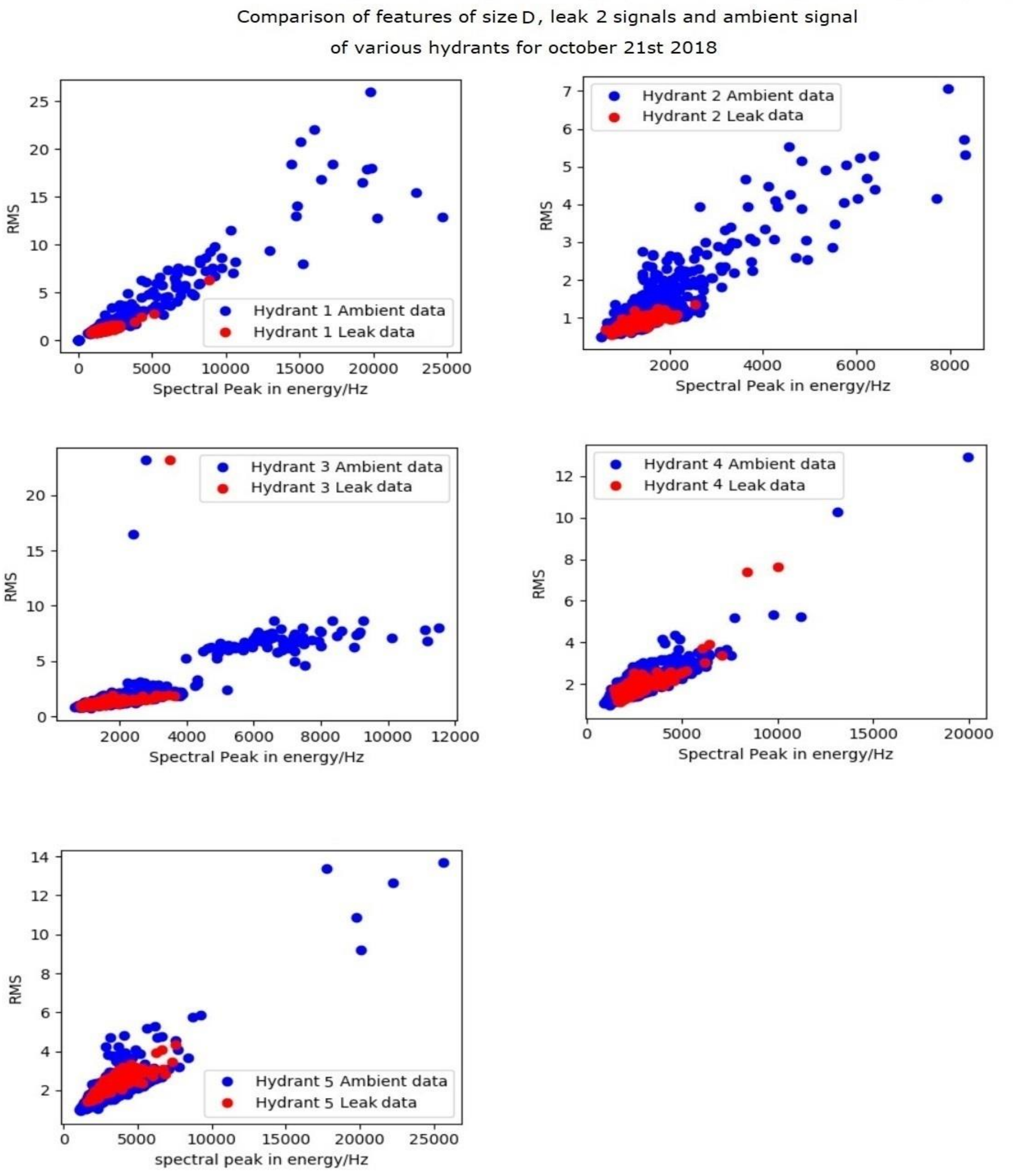


Figure 4.10 Feature graphs of ambient and leak (size D) data from October 21st, for various hydrants.

4.6 Training OCSVM and prediction

The model used in this thesis is based on OCSVM which is based on Scikit-learn Python library. Though there are many leak detection methods such as artificial neural network (ANNs) (Mounce, S. R. et al., 2010, Aksela, K. et al., 2009, Romano, M. et al., 2011, Yang, J. et al., 2010) available, they need a significant amount of historical data for training. Previous studies employing leak detection performed on the same data sets showed that the semi-supervised OCSVM method provided good anomaly detection results (Cody, R. et al., 2018). OCSVM is a well-established anomaly detection algorithm (Scholkopf et al., 2001) which only requires training data from the baseline state of the system in order to determine if the new data belongs to the known baseline data class or if it is unknown. There are a number of kernels available to use with this algorithm such as linear, polynomial, sigmoidal and Gaussian radial basis functions (RBF). The linear kernel can classify very large leaks (from Figures 4.3 and 4.7) as the classes are clearly separable. However, to classify smaller leaks (of sizes B and C, where the classes show some overlapping as in Figures 4.4, 4.5 and 4.9), RBF is more suitable as it can classify datasets which are non-linearly separable. Since RBF can classify more cases (sizes A, B, and C) compared to linear kernel (size A), RBF is selected as the kernel for OCSVM training.

The algorithmic details of OCSVM are provided in Appendix B. The OCSVM algorithm is trained with only the ambient data on the test day. For example, to test the performance of OCSVM on the October 13th, ambient data from the October 13th is treated as the baseline data and used to train the OCSVM algorithm for various values of hyper parameters (the parameters for tuning the performance of the algorithm), gamma and nu. For the current OCSVM implementation, the parameter gamma takes 8 values on an equally spaced logarithmic scale ranging between -6 and 1 (1e-06 to 10 on a linear scale). The parameter nu takes 10 values on an equally spaced linear scale ranging between 0.01 and 0.99. Hence, for 80 combinations of gamma and nu, the OCSVM is trained (grid search) and for each model or for each combination of gamma and nu, the test set consisting of 50% ambient case and 50% leak case of size A (in this case Oct 13th test data) is presented and the labels (+1 for ambient and -1 for Leak case) are predicted. These predicted labels along with the true labels (the expected labels of the test data) are used to check the performance of the trained model.

Along with prediction accuracy (true positive), other metrics are used to demonstrate the performance of the OCSVM model. In order to take false negatives and false positives also into account along with true positives, the metrics Precision, Recall and F1 scores are calculated. The definitions for the aforementioned three scores, precision, recall and F1, are given next.

Precision: This score is defined as the ratio of number of true positive labels to the sum of the number of true positives and false positive labels in the prediction result (given in the mathematical expression 3). This is calculated using the below mathematical expression

$$\text{Precision} = t_p / (t_p + f_p) \tag{3}$$

With respect to leak, t_p here is number of leak cases that are predicted correctly and f_p is the number of ambient cases that are predicted as leak cases. The precision score is a floating-point number which can take values between 0 and 1, 0 being the worst and 1 being the best. Intuitively, precision score is an indicator of the ability of the trained model (the classifier) not to label as positive, a sample that is negative. For example, if the precision score is high for the case of a leak, then the model is most likely to predict the case as leak.

Recall: This score is defined as the ratio of true positives to the total number of true positives and false negatives (given in the mathematical expression 4). With respect to a leak, this is the number of leak cases predicted correctly to the sum of number of leak cases predicted correctly and number of leak cases that are predicted as ambient cases. Intuitively, the recall score indicates the ability of the trained model to find all the positive samples (or the sample of interest, leak in this case). Recall score is a floating-point number which can take values between 0 and 1, 0 being the worst and 1 being the best.

$$\text{Recall} = t_p \; / \; (t_p + f_n) \tag{4}$$

F1: This is the weighted average score of Precision and Recall, which is a floating point number which can take values between 0 and 1, 0 being the worst and 1 being the best. The relative contribution of precision and recall to F1 score is equal and is given by the mathematical expression 5.

$$\text{F1} = 2 * (\text{precision} * \text{recall}) \; / \; (\text{precision} + \text{recall}) \tag{5}$$

Though the F1 score is often adequate to evaluate the performance of the trained model, Precision and Recall scores are also used to provide better insight into the model. These scores are calculated for both ambient and leak cases separately.

In the process of finding the best generic model which can correctly classify leak and ambient cases, the model which results in the best F1 score for size A leak is selected after the Precision score, Recall score and F1 scores are calculated for each combination of the hyper parameters (gamma and nu). The model with the highest F1 score is then tested for sizes B, C and D leaks. From the results in Tables 4.3 and 4.4, it can be seen that sizes A, B and C leaks (for both Leak 1 and 2) are predicted with high F1 scores as their features were clearly separable from the features associated with the ambient data (in Figure 4.3 through 4.5 and 4.7 through 4.9), whereas for size D leak case, the F1 scores are relatively low as their features are not clearly separable in the feature space (in Figures 4.6 and 4.10). The prediction results for Leak1 for the instrumented Hydrant 3 corresponding to the October 13th, 2018 test date shown in the Table 4.3 was obtained by training the OCSVM algorithm only with the ambient data corresponding to the October 13th for the instrumented Hydrant 3. For the leak sizes A, B and C, the Recall scores are high, however the Precision scores are lower compared to the Recall scores. This means that most of the leak cases were identified as leaks (and a small number of leak cases were identified as ambient), and many ambient cases were identified as leaks.

The prediction results for Leak 1, for the October 21st test data, are shown in the Table 4.4. These results were obtained by training the OCSVM algorithm with the ambient data from the October 13th, 19th, 20th test dates and a small portion of the ambient data from October 21st. It can be seen from the results in Tables 4.3 and 4.4 that the F1 scores for sizes A, B and C are better compared to the results from the October 13th. Similarly, Table 4.5 shows that the prediction results can be improved when training data from previous days are also included to train the prediction model for the Leak2 case. These observations show that as more training data is made available, the prediction model becomes more generic and provides good prediction capability and so, the model needs to be updated frequently for achieving good prediction performance. However, the fine-tuning of the model and a detailed investigation into the factors governing leak detection performance is considered beyond the scope of this thesis and these results are provided only for the sake of completeness of the IoT implementation, the latter being the main focus of this thesis.

For 13th ambient train data, 13thambient test and leak test data				
	gamma =0.000001, nu =0.2278			
size of the leak	precision score	recall score	F1 score	overall accuracy
A	0.79	1	0.88	0.87
B	0.79	0.99	0.88	0.86
C	0.78	0.96	0.86	0.85
D	0.36	0.15	0.21	0.44

Table 4.3 Accuracies of prediction using OCSVM for various leak sizes for Leak 1, Hydrant 3 for October 13th training and testing data.

For 13th, 19th, 20th, 21st ambient train data, 21st ambient test and leak test data				
	gamma = 0.001, nu = 0.1189			
size of the leak	precision score	recall score	F1 score	overall accuracy
A	0.98	1	0.99	0.99
B	0.98	1	0.99	0.99
C	0.98	1	0.99	0.99
D	0.95	0.27	0.42	0.62

Table 4.4 Accuracies of prediction using OCSVM for various leak sizes for Leak 1, Hydrant 3 for October 13th, 19th, 20th and 21st training data and October 21st testing data.

19th, 20th, 21st ambient training, 21st ambient test and leak test				
	gamma = 0.000001, nu = 0.1189			
	precision score	recall score	F1 score	overall accuracy
A	0.9	1	0.95	0.95
B	0.9	1	0.95	0.95
C	0.9	1	0.95	0.94
D	0.53	0.11	0.18	0.51

Table 4.5 Accuracies of prediction using OCSVM for various leak sizes for Leak 2, Hydrant 5 for October 19th and 20th training and 21st October testing data

From the results in Tables 4.3, 4.4 and 4.5, the overall accuracies (which is a combination of ambient and leak test case accuracies) match the corresponding F1-scores. Hence, overall accuracy can also be used to evaluate the performance of OCSVM. F1-score, precision score and recall score provide additional insight into the performance of the algorithm.

4.7 Summary

The following observations can be made based on the results of development and testing of the leak detection IoT framework using GCP. The cloud technology was successfully deployed in solving the challenge of storing and processing a large amount of data collected from the IoT edge devices (emulators). The cloud processing pipeline was able to perform prediction on both live data (simulated) and batch data. The protocol such as HTTP was necessary for the communication between different systems (data sources (emulators) and data processing system (GCP)) and to ensure interoperability. Reducing the raw data to features was necessary to reduce the processing and transmission cost. The selected features RMS and spectral peak when used together successfully identified very large (200 litres/minute outflow), large (100 litres/minute outflow) and medium (50 litres/minute outflow) leaks. OCSVM algorithm with RBF kernel performed very well in identifying large and medium sized leaks. Recall and Precision scores provided valuable insight into the performance of prediction algorithm OCSVM and the sensitivity analysis showed that the distance of an instrumented hydrant from a leak plays an important in determining whether features extracted from the data collected from that hydrant show sensitivity towards leak. The closer the hydrants are to a leak, the better the sensitivity (towards that leak) exhibited by the features extracted from the hydrants' data.

5 Conclusions and Recommendations

5.1 Conclusions

This thesis focuses on an IoT implementation using Google Cloud Platform (GCP) and provides a detailed design and implementation of the cloud framework that is scalable, secure and able to analyze both real time as well as batch data collected from IoT devices attached to water distribution network for leaks. The processing capabilities of the framework shows that the concept of cloud based IoT solution for leak detection in WDN is viable and can provide a powerful means for utilities to carry out intervention strategies in continuous monitoring settings. The main conclusions of this study along with the conclusions resulting from the implementation are described next.

5.1.1 Overall framework performance

In terms of the overall performance of the IoT based leak detection framework presented, the following main conclusions can be drawn:

- The framework was able to process data from two different emulators running two different platforms (Hologram Cloud and a custom Python program running on windows OS), which shows that this framework supports data from multiple sources through HTTP protocol.
- Feature extraction at the sensor node level allows for lower data transmission overhead and processing costs by reducing raw time-series data to only the pertinent information required for the machine learning algorithm. This could have significant implications for real-world implementation employing hundreds and possibly thousands of IoT sensors at a city scale.

- Security is ensured at each stage on GCP. For example, the data ingestion layer on GCP embeds two types of authentication such as API key authentication and OAuth2.0 authentication. Similarly, on *Cloud Storage*, client-side encryption and server-side encryption ensures data security. In *Cloud Dataflow* pipeline communication with GCP sources and sinks are encrypted and the communication is carried over HTTPS while all inter-worker communication is carried over a private network.
- Though the framework used emulators to emulate live stream of acoustic pressure data coming from hydrophones attached to hydrants, the thesis did not employ synthetic data and the decision making sub-system of the framework was tuned using actual data. Also, the classification algorithm used in this thesis does not require all possible cases to set a baseline. Hence the framework can be confidently used to classify the data in a live WDN when emulators are replaced by the actual data collection sub system, without making any changes to the overall framework.
- Since GCP is integrated with easy to use and powerful machine learning API (since it is integrated with Sci-kit learn and Tensorflow Python libraries), the task of data analysis was relatively straight-forward.
- Auto-scaling infrastructure on GCP provided scalability at every stage of data life cycle (such as data ingestion, storage, processing and result visualization) without user intervention.
- Since the GCP is based on distributed computing, it is fault tolerant and so are the data analysis and decision-making layers in the leak detection framework.
- Decision support sub-system used a simple, yet powerful decision-making algorithm, OCSVM with Gaussian RBF kernel. The model used in this thesis was able to detect very large, large, and medium-sized leaks and ambient cases. However, a model which can classify

very small leaks remains to be developed. OCSVM was able to detect a majority of the leak types and ambient cases with minimal training (only with ambient labeled data). This is ideal for use in live WDNs, where it is extremely difficult to obtain labeled data for the leak cases whereas obtaining ambient data is relatively simple.

5.2 Recommendations for future work

This thesis presented an effective, scalable, and secure framework which can be deployed at scale to detect leaks in WDNs. This work is by no means exhaustive and there is still room for improvement in many areas. The following list summarizes some of the key areas for improvement.

- While leak detection was demonstrated and the instrumented hydrant (from where the data was collected) that is close to the leak was identified, leak localization (the exact leak location) was not implemented, which could be a future task.
- Live testing (in a live WDN) was not undertaken in this thesis, mainly due to the difficulty in finding industry partners. This framework could be deployed in an actual WDN and its performance could be evaluated in the future.
- The framework can currently support data sources which rely on the HTTP protocol. The framework can be broadened to support other communication protocols such as MQTT and to support greater interoperability.

References

Wu, Z.Y., Farley, M., Turtle, D., Kapelan, Z., Boxall, J., Mounce, S.R., Dahasahasra, S., Mulay, M. and Kleiner, Y., (2011), "Water loss reduction", USA: Bentley Institute Press, 1-68.

Hunaidi, O., (2000), "Detecting leaks in water-distribution pipes ", Institute of Research and Construction (IRC), October, 2000, ISSN 1206-1220.

Whittle, A. J., Allen, M., Preis, A. and Iqbal, M., (2013), "SENSOR NETWORKS FOR MONITORING AND CONTROL OF WATER DISTRIBUTION SYSTEMS", The 6th International Conference on Structural Health Monitoring of Intelligent Infrastructure Hong Kong.

Karray, F., Garcia-Ortiz, A., Jmal, M.W., Obeid, A.M., Abid, M., (2016), "EARNPIPE: A Testbed for Smart Water Pipeline Monitoring using Wireless Sensor Network", Procedia Computer Science 96 (2016)285–294.

Feng, N., (2019), "Online Monitoring Framework for Pressure Transient Detection in Water Distribution Networks", Masters thesis.

Adedeji, K., Hamam, Y., Abe, B. and Abu-Mahfouz, A., (2016), "Wireless Sensor Network-based improved NPW Leakage detection algorithm for real-time application in pipelines" by, Southern Africa Telecommunication Networks and Applications Conference (SATNAC) 2016.

Arangoa, I.M., Izquierdo, J.S., Campbell, E.O.G. and Pérez-Garcíab, R., (2014), "Cloud-based Decision Making in Water Distribution Systems", Procedia Engineering 89(2014)488 – 494.

Puust R., Kapelan Z., Savic D.A. and Koppel T., (2010), "A review of methods for leakage management in pipe networks" Urban Water Journal vol 7, no 25, pp25-45.

Ramos, H., Reis, C., Ferreira, C., Falcao, C. and Covas, D., (2001), "Leakage Control Policy within operating management Tools" Proceedings of the 6th International Conference on Computing and Control in the Water Industry, Leicester, UK.

Kim, M. and Lee, S., (2009), "Detection of leak acoustic signal in buried gas pipe based on the time-frequency analysis". Journal of Loss Prevention in the Process Industries, 990-994.

Tapanes, E., (2001), "Fibre optic sensing solutions for real-time pipeline integrity monitoring", In Australian Pipeline Industry Association National Convention, Vol. 3, pp. 27-30.

Furness, R.A., Reet, J.D., 2009, "Pipeline leak detection techniques". Pipeline rules of thumb handbook.

Geiger, G., (2006), "State-of-the-Art in Leak Detection and Localisation", Pipeline Technology 2006 Conference.

Lowry, W. E., Dunn, D., Walsh, R., Merewether, D., and Rao, D. V., (2000). “Method and system to locate leaks in subsurface containment structures using tracer gases”. U.S. Patent No. 6,035,701. Washington, DC: U.S. Patent and Trademark Office.

USDT, (2007), “Leak Detection Technology Study For PIPES”, Act. Tech. rep., US Department of Transportation.

Hauge, E., Aamo, O., and Godhavn, J., (2007), “Model based pipeline monitoring with leak detection”. In Proc. 7th IFAC Symp. on Nonlinear Control Systems.

Carbó-Bech, A., Las Heras, S. A. D. and Guardo, A., (2017), "Pipeline Leak Detection by Transient-Based Method Using MATLAB R Functions", doi:10.20944/preprints201706.0007.v1, Preprints.

Zhang, J., and Di Mauro, E. (1998), “Implementing a reliable leak detection system on a crude oil pipeline”, Dubai, UAE: Advances in Pipeline Technology.

Mpesha, W., Gassman, S., and Chaudhry, M. (2001), “Leak detection in pipes by frequency response method”. Journal of Hydraulic Engineering, 127, 134-147.

Liu, J., Yao, J., Gallaher, M., Coburn, J. and Fernandez, R., (2008), “Study on Methane Emission Reduction Potential in Chinas Oil and Natural Gas Industry”. Tech. rep.

Golmohamadi, M., (2015), “Pipeline leak detection” Masters Thesis.

Gamboa-Medina, M.M., Ribeiro Reis, L.F., Capobianco Guido, R., (2014), “Feature extraction in pressure signals for leak detection in water networks”, Procedia Engineering, 70 (2014)688–697.

Eyuboglu, S., Mahdi, H., Al-Shukri, H. and Rock, L., (2003), "Detection of water leaks using ground penetrating radar", Proceedings of the Third International Conference on Applied Geophysics, Orlando-FL; p. 8-12.

Ayala–Cabrera, A., Herrera, M., Izquierdo, J., Oca˜na–Levario, S.J. and P´erez–Garc´ıa, R., (2013),"GPR-Based Water Leak Models in Water Distribution Systems", sensors ISSN 1424-8220.

Abouhamad, M., Zayed, T., and Moselhi, O., (2016), "Leak Detection in Buried Pipes Using Ground Penetrating Radar—A Comparative Study", Pipelines, pp. 417-424.

Weil, G. J., (1993), “Non contact, remote sensing of buried water pipeline leaks using infrared thermography. In Water Management in the’90s: A Time for Innovation” (pp. 404-407). ASCE.

Jackson, C.N.; Sherlock, C.N. (1998): “Non-destructive Testing Handbook: Leak Testing”, page 519, Library of Congress Cataloging-in-Publication Data, 2008.

Agapiou, A., Themistocleous, K., Alexakis, D., Hadjimitsis, D.G., (2014), “Water leakage detection using remote sensing, field spectroscopy and GIS in semiarid areas of Cyprus” Urban Water Journal.

Hadjimitsis, D. G, Clayton, C. R. I, & Toulios, L. (2010a). A New Method for Assess- ing the Trophic State of Large Dams in Cyprus Using Satellite Remotely Sensed Data, Water and Environment Journal, doi:10.1111/j.1747-6593.2009.00176.x., 24, 200-207.

Geiger, G. and Werner, T., (2003), "Leak detection and location- A Survey," in Proc. PSIG Annual Meeting, Bern, Switzerland, Oct. pp.1-11.

St. Großwig, A., Graupner, E., Hurtig, K. Kühn and A. Trostel, "Distributed fibre optical temperature sensing technique- a variable tool for monitoring tasks", Proceedings of the 8th International Symposium on Temperature and Thermal Measurements in Industry and Science 19 – 21 June 2001

Lambert, A. (1994), "Accounting for losses: The Burst and background concept", (BABE) IWEM Journal, April 1994, 8(2), 205-14.

MacDonald, G. (2005) "DMA Design and Implementation", a North American Context. Leakage Conference, IWA.

Zan, T., H. Lim, K. Wong, J. Whittle, and B. Lee, (2014), "Event detection and localization in urban water distribution network." IEEE Sensors Journal 14: 4134–4142. doi:10.1109/JSEN.2014.2358842.

Wu, Y., and S. Liu., (2017), "A review of data-driven approaches for burst detection in water distribution systems." Urban Water Journal 14: 972–983. doi:10.1080/1573062X.2017.1279191.

Pudar, R.S., Liggett, J.A., (1992), "Leaks in pipe networks", Journal of Hydraulic Engineering, ASCE 118 (7), 1031-1046.

Kapelan, Z.S., Savic, D.A., Walters, G.A., (2003), "A hybrid inverse transient model for leakage detection and roughness calibration in pipe networks", Journal of Hydraulic Research, IAHR 41 (5), 481-492.

Kapelan, Z.S., Savic, D.A., Walters, G.A., (2002), "Hybrid GA for calibration of water distribution models", In: Proc. EWRI, Roanoke, VA, May, 2002.

Vitkovsky, J., Simpson, A.R., Lambert, M.F., (2000), "Leak detection and calibration using transients and genetic algorithms Journal of Water Resources Planning & Management", 126(4):262-265.

Wang, X.J., Lambert, M.F., Simpson, A.R., Liggett, J.A., Vitkovsky´, J.P., (2002), "Leak detection in pipelines using the damping of fluid transients", Journal of Hydraulic Engineering, ASCE 128 (7), 697-711.

Karney, B., Khani, D., Halfawy, M.R. and Hunaidi, O., (2009), "A Simulation Study on Using Inverse Transient Analysis for Leak Detection in Water Distribution Networks", R235-23. doi: 10.14796/JWMM.R235-23., www.chijournal.org ISSN: 2292-6062 (Formerly in Conceptual Modeling of Urban Water Systems. ISBN: 978-0-9808853-2-3).

Pothof, I. and Karney, B., (2012), "Guidelines for Transient Analysis in Water Transmission and Distribution Systems",DOI: 10.5772/53944.

Colombo, A.F., Lee, P., Karney, B.W., (2009), “A selective literature review of transient-based leak detection methods”, Journal of Hydro-environment Research 2 (2009) 212-227.

Boulos, P. F., Karney, B. W., Wood, D. J., Lingireddy, S., (2005), "Hydraulic Transient Guidelines for Protecting Water Distribution Systems", American Water Works Association. Journal; May 2005; 97, 5; ProQuest.

Chang, Y.C., Lai, T.T., Chu, H.H. and Huang, P., (2009), “Pipeprobe: Mapping spatial layout of indoor water pipelines”. In: Mobile Data Management: Systems, Services and Middleware, MDM’09. Tenth International Conference on. IEEE; p. 391–392.

Nixon, W., Ghidaoui, M., (2006), “Range of validity of the transient damping leakage detection method”. Journal of Hydraulic Engineering, ASCE 132 (9), 944e957.

Seto, L., Ross, T., (2013). “Development of a Long Duration, Free Swimming, Inline Acoustic Leak Detection Inspection Tool”.

Hunaidi, O.; Wang, A.; Bracken, M.; Gambino, T.; Fricke, C, (2004), “Acoustic methods for locating leaks in municipal water pipe networks”, NRCC-47062.

Pal, M., Dixon, N., Flint, J., (2010), “Detecting & Locating Leaks in Water Distribution Polyethylene Pipes”, ISBN: 978-988-18210-7-2, ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online).

Choi, J., Shin, J., Song, C., Han, S. and Park, D., (2017). “Leak Detection and Location of Water Pipes Using Vibration Sensors and Modified ML Prefilter”, www.mdpi.com/journal/sensors, doi:10.3390/s17092104.

Zhang, L.; Wu, Y.; Guo, L.; Cai, P., (2013), “Design and Implementation of Leak Acoustic Signal Correlator for Water Pipelines”. Inf. Technol. J. 2013, 12, 2195–2200, doi:10.3923/itj.2013.2195.2200.

Cheung, P., Lemos, V.B. and Andrade, B., (2014), "Cloud Computing Solutions For Aiding Urban Water Management", http://academicworks.cuny.edu/cc_conf_hic/327.

Gao, J., Liu, J., Nori, R., Rajan, B., Fu, B., Xiao, Y. and Liang, W., (2013), Chen, C.L.P. "SCADA communication and security issues", Security Comm. Networks 2014; 7:175–194.

Romano, M., Kapelan, Z. and Savic, D., (2013), “Automated Detection of Pipe Bursts and other Events in Water Distribution Systems”, Journal of Water Resources Planning and Management.

Barnaghi, P., Wang, W., Henson, C. and Taylor, K., (2012), "Semantics for the Internet of Things: Early Progress and Back to the Future", (IJSWIS) 8(1), 10.4018/jswis.2012010101.

Shadroo, S. and Rahmani,A.M., (2018),"Systematic survey of big data and data mining in internet of things", Computer Networks, 139 (2018) 19-47.

Christodoulou, S., Agathokleous, A., Kounoudes, A., and Milis, M., (2010), "Wireless Sensor Networks for Water Loss Detection" European Water 30:41-48.

Mohamed, N., Jawhar, I., Al-Jaroodi, J. and Zhang, L., (2011), “Sensor Network Architectures for Monitoring Underwater Pipelines” Sensors, 11, 10738-10764; doi:10.3390/s111110738.

AL-Kadi, T., AL-Tuwaijri, Z and AL-Omran, A., (2013), "Wireless Sensor Networks for Leakage Detection in Underground Pipelines: A Survey Paper", Procedia Computer Science 21(2013)491–498.

Kartakis, S., Abraham, E., and McCann, J.A., (2015), “Waterbox: A testbed for monitoring and controlling smart water networks.” In: Proceedings of the 1st ACM International Workshop on Cyber-Physical Systems for Smart Water Networks. ACM; p. 8.

Stoianov, I., Nachman, L., Madden, S. and Tokmouline, T., (2007), "PIPENET: A wireless sensor network for pipeline monitoring", DOI: 10.1109/IPSN.2007.4379686 · Source: IEEE Xplore.

Sadeghioon, A.M., Metje, N., Chapman, D. and Anthony, C., (2018), “Water pipeline failure detection using distributed relative pressure and temperature measurements and anomaly detection algorithms”, Urban Water Journal 2018, Vol. 15, No. 4, 287–295.

Abdelhafidh, M., Fourati, M., Fourati, L.C. and Abidi, A., (2017), "Remote Water Pipeline Monitoring System IoT-based Architecture for new Industrial era 4.0", 2161-5330/17 $31.00 © 2017 IEEE, DOI 10.1109/AICCSA.2017.158.

Mois, G., Sanislav, T. and Folea, S.C. (2016), "A Cyber-Physical System for Environmental Monitoring", IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 65, NO. 6.

Zhao, J., Li, D., Qi, H., Sun, F., and An, R., (2010), “The Fault Diagnosis Method of Pipeline Leakage Based On Neural Network,” Proceedings of International Conference on Computer, Mechatronics, Control and Electronic Engineering, pp. 322-325.

Ferrante, M., Brunone, B., and Meniconi, S., (2007), “Wavelets for The Analysis of Transient Pressure Signals for Leak Detection,” J Hydraul Eng ASCE 133(11), pp. 1274–1282.

Chen, H., Ye, H., Chen, L., and Su, H., (2004), "Application of Support Vector Machine Learning to Leak Detection and Location in Pipelines," In Instrumentation and Measurement Technology Conference, Proceedings of the 21st IEEE, Vol. 3, pp. 2273-2277.

Billmann, L. and Isermann, R., (1987), "Leak Detection Methods for Pipelines," Automatica, pp. 381-385.

Hadjimitsis, D.G., Agapiou, A., Themistocleous, K., Toulios, G., Perdikou, S., Toulios, L., and Clayton, C., (2013), "Detection of water pipes and leakages in rural water supply networks using remote sensing techniques", In D.G. Hadjimitsis (Ed.), Remote Sensing of Environment - Integrated Approaches (pp. 155–180). Rijeka, Croatia: InTech.

Koo, D., Piratla, K. and Matthews, J. C., (2015), "Towards Sustainable Water Supply: Schematic Development of Big Data Collection Using Internet of Things (IoT)" Procedia Engineering 118 (2015) 489 – 497.

Rohit, A., Sakthivel, S., Sandheep, T.J. and Saraswathi, V., (2018), "Water Leakage System Using IoT" by ISSN: 2350-0557, Volume-5, Issue-2, March-2018.

Mounce, S.R, Pedraza, C., Jackson, T., Linford, P. and Boxall, J.B., (2015), "Cloud based machine learning approaches for leakage assessment and management in smart water networks", Procedia Engineering 119(2015)43–52.

Mounce, S.R., Machell, J. and Boxall, J.B., (2007), "An Artificial Neural Network/Fuzzy Logic system for DMA flow meter data analysis providing burst identification and size estimation", CCWI2007, SUWM2007 (pp 313-320).

Mounce, S. R., Boxall, J. B. and Machelle, J., (2010), "Development and Verification of an Online Artificial Intelligence System for Detection of Bursts and Other Abnormal Flows", Journal of Water Resources Planning and Management, vol 136, no. 3, pp. 309-318, 2010.

Mounce, S., R. Mounce, and J. Boxall, (2010), "Novelty detection for time series data analysis in water distribution systems using support vector machines." Journal of Hydroinformatics 13: 672–686. doi:10.2166/hydro.2010.144.

Mounce, S.R., Mounce, R.B., and Boxall, J.B. (2011). "Novelty detection for time series data analysis in water distribution systems using support vector machines". Journal of Hydroinformatics. 13 (4): 672-686.

Ahmed, K.E.U., Gregory, M.A., (2011), "Integrating Wireless Sensor Networks with Cloud Computing" 011 DOI: 10.1109/MSN.2011.86.

Anzelmo, E., Bassi, A., Caprio, D., Dodson, S., Kranenburg, R.V., Ratto, M., (2011), "Discussion Paper on the Internet of Things" commissioned by the Institute for Internet and Society, Berlin.

Akbar, A., Khan, A., Carrez, F. and Moessner, K., (2017), "Predictive Analytics for Complex IoT Data Streams", IEEE INTERNET OF THINGS JOURNAL, VOL. 4, NO. 5.

Kim, Y., Lee, S., Park, T., Lee, G., Suh, J. and Lee, J, (2016), "Robust leak detection and its localization using interval estimation for water distribution network." Computers & Chemical Engineering 92: 1–17. doi:10.1016/j.compchemeng. 2016.04.027.

Aksela, K., Aksela, M. and Valhala, R., (2009), "Leak detection in a real distribution network using SOM", Urban Water Journal, vol 64, (January), pp. 279-289, 209, 2009.

Romano, M., Kapelan, Z. and Savic, D., (2011), "Real-Time Leak Detection in Water Distribution Systems", Journal of Water Distribution Systems Analysis, pp. 1074-1082, 2011.

Yang, J., Wen, Y. and Li, P., (2010), "Approximate entropy-based leak detection using artificial neural network in water distribution pipelines", International Conference on Control Automation Robotics Vision (ICARCV), pp. 1029-1034.

Salam, A. E. U., Tola, M., Selintung, M. and Maricar, F., (2010), "A leak detection system on the Water Pipe Network through Support Vector Machine method", Makassar International Conference on Electrical Engineering and Informatics (MICEEI), pp. 161-165, 2014.

Mashford, J., De Silva, D., Burns, D. and Marney, D., (2012), "Leak Detection in Simulated Water Pipe Networks Using SVM," Journal of Applied Artificial Intelligence, vol 26, no. 5, pp. 429-444.

Zhang, Q., Wu, Z. Y., Zhao, M. and Qi, J., (2010), "Leak Zone Identification in Large-Scale Water Distribution Systems Using Multiclass Support Vector Machines", Journal of Water Resources Planning and Management, vol 142, no. 11, pp. 1-15.

Terao, Y. and Mitra, A., (2008), "Robust water leakage detection approach using the sound signals and pattern recognition", Sensors and Smart Structures Technologies for Civil, Mechanical and Aerospace Systems, vol 6932, pp. 69322D-69322D-9.

Rashid, S., Akram, U. and Khan, S. A., (2015), "WML: Wireless sensor network based machine learning for leak detection and size estimation," Procedia Computer Science, vol 63, pp. 171-176.

Colombo, A.F., Lee, Pedro., Karney, B. W. (2009), "A selective literature review of transient-based leak detection methods", Journal of Hydro-environment Research 2 (2009) 212e227 213.

PureTechnologies, Smartball for Water Leak Detection, (2009), http://www.puretechnologiesltd.com>

Liu, Z., Kleiner, Y.,(2013), "ground penetrating radar and infrared imaging".State of the art review of inspection technologies for condition assessment of water pipes", Measurement 46 (2013) 1–15.

Fielding, R. T.; Gettys, J., Mogul, J. C. Nielsen, H. F., Masinter, L., Leach, P. J., Berners-L. T., (June 1999). Hypertext Transfer Protocol – HTTP/1.1. IETF. doi:10.17487/RFC2616. RFC 2616.

Shabnam Shadroo, Amir Masoud Rahmani, (2018), "Systematic survey of big data and data mining in internet of things", Computer Networks, 139 (2018) 19-47.

Yokotani, T., Sasaki, Y., (2016), "Comparison with HTTP and MQTT on Required Network Resources for IoT. The 2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC).

Cody, R, Narasimhan, S, Tolson, B, (2017), "one-class SVM – leak detection in water distribution systems", CCWI 2017 – Computing and Control for the Water Industry

Cody, R, Harmouche, J and Narasimhan, S, (2018), "Leak detection in water distribution pipes using singular spectrum analysis", URBAN WATER JOURNAL, 2018, VOL. 15, NO. 7, 636–644, https://doi.org/10.1080/1573062X.2018.1532016

Cooley, J. W., and John W. Tukey, (1965), "An algorithm for the machine calculation of complex Fourier series," Math. Comput. 19: 297-301.

Press, W., Teukolsky, S., Vetterline, W.T., and Flannery, B.P., (2007), Numerical Recipes: The Art of Scientific Computing, ch. 12-13. Cambridge Univ. Press, Cambridge, UK.

Pedregosa, F., Varoquaux,G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., (2011), "Scikit-learn: Machine Learning in Python", JMLR 12, pp. 2825-2830.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., M¨uller, A.C., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., Vanderplas, J., Joly, A., Holt, B., Varoquaux, G., (2013), "API design for machine learning software: experiences from the scikit-learn project", European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases, Prague, Czech Republic. ffhal-00856511f.

Scholkopf, B., J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson, (2001), "Estimating the support of a high-dimensional distribution." Neural Computation 13: 1443–1471. doi:10.1162/089976601750264965.

Steven, W. S, (1999). "Chapter 8: The Discrete Fourier Transform". The Scientist and Engineer's Guide to Digital Signal Processing (Second ed.). San Diego, Calif.: California Technical Publishing. ISBN 978-0-9660176-3-2.

IEEE Standard Glossary of Software Engineering Terminology.

1. http://hacklerplumbingmckinney.com/water-leak-detection-equipment/
2. http://yunik.co.za/services/pipe-leak-sealing-technology/
3. https://www.youtube.com/watch?v=5tEumKrxjFM

4. http://www.optimale.com.br/
5. https://www.youshare.ac.uk/
6. https://azure.microsoft.com/en-us/
7. https://developers.google.com/identity/protocols/OAuth2
8. https://cloud.google.com/docs/
9. https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

Appendix A

Details of Implementation

Introduction

The details of implementation of emulators (Python client on Raspberry Pi, Hologram cloud, Python client on a laptop computer*), Cloud Functions*, *Cloud Dataflow*, *Cloud ML*, *Cloud SDK*, *BigQuery* and *AI Platform Notebook* are provided in Appendix A.

Data acquisition layer

The initial phase of data collection from WDN, data pre-processing and feature extraction were discussed in chapter 4. So only the implementation details of the rest of the IoT components are given here.

Emulators

Python client on Raspberry Pi

Common IoT device, the Raspberry PI was also used in the implementation pipeline. Python 3.7 along with the Hologram Python SDK was installed to enable communication between the Raspberry Pi and a USB cellular modem, Hologram Nova. The features calculated from the hydrophone data were sent in the JavaScript Object Notation (JSON) format using `hologram.sendMessage` function syntax of the Hologram SDK as follows:

```
hologram.sendMessage("{‟rms”:  ″10.11″,  ″spectral_peak″:″33.23″,  ″label″:″-1″,
‟hydrant″:″h3″}")
```

JSON is a language independent data format that is human readable and consists of data in key value pairs. In the above JSON message example, label 1 is associated with ambient data and -1 for leak data. Data is identified by its location, `hydrant` value, for example `‟h3″` is the data captured from the field hydrant `h3`, which is located at the address, 30 Paulstown in Guelph, as described in Chapter 3.

Hologram Cloud

Hologram Cloud was configured to automatically transmit the data received from Raspberry Pi to the GCP *Cloud Function* by configuring a set of parameters for a webhook (an HTTP POST request provided in the form of web UI). The configuration included providing the right destination URL, I.e. GCP *Cloud Function* URL along with a device identification number of the Hologram modem, timestamp of receiving JSON data and the API key provided by GCP. They were separated by the symbol “&”. In the header section, x-auth-token was provided, which consisted of a base-64 encoded username and password for the Hologram cloud. The following sample explains the route configuration.

Fields	**Subfields**	**Values**
URL		https://us-central1-uw-cive-700-1.cloudfunctions.net/Login?device=<<device_id>>&timestamp=<<received>>
PAYLOAD		<<decdata>>
HEADERS	content-type	application/json

	x-auth-token	dGVzdC50ZXN0QHRlc3QuY29tOnRlczEyMzoi
	Apikey	MIzaMyCugA9sAgpUhGVH0KtXbACPyHP8OPrpNLX

Table 1 Route configuration for the webhook on hologram cloud

The features received by Hologram from Raspberry Pi 3 is transmitted by HTTP POST request to GCP, which is explained in the next section.

Python client on laptop

The laptop used as a Python client runs Python 3.7 on a 64-bit, Windows 10 OS, which sends the data using HTTP POST requests to *Cloud Functions*. Emulators first send the authentication information to GCP and upon passing this authentication stage the features are transmitted to the GCP. The implementation of two types of authentication are discussed next.

(i) API key authentication: A typical HTTP POST request with API key authentication looks like:

```
res=requests.post("https://us-central1-uw-cive-700-1.cloudfunctions.net/Api_Key_Demo_ML?key=AIzaSyAvLhEDXnG2h8ES6He76eW8wkFyOm6jiqE&device=156790&timestamp=31-07-2019T12:12:12", json={"rms": "10.11", "spectral_peak": "33.23", "label": "-1", "hydrant" : "h3"}, headers={'X-Auth-Token': base64.b64encode(bytes(combined_string, encoding='utf8'))})
```

The `requests.post` is a python function syntax to send an HTTP POST request consisting of a website address as the first argument which represents GCP *Cloud Function*, which is an ingestion component of the GCP. The URL also consists of an API key, device identification number, and timestamp corresponding to the time of request to the GCP. The second argument in the `requests.post` is the data represented in JSON. The third argument of `requests.post` function syntax is the header which consists of the base64 encoded username and password.

Base-64 is a binary-to-text encoding scheme where each digit represents 6-bits of data. If an ASCII (subset of UTF-8) string is represented by three 8-bit bytes total 24 bits), then the Base64 converted string contains four 6-bit data (total 24 bits). A simple example is shown in Table 2, where the string “Cat” is converted to “Q2F0” in Base64 encoding.

ASCII or UTF-8	character	C								A								T							
	8-bit	67 (0x43)								97 (0x61)								116 (0x74)							
Bits		0	1	0	0	0	0	1	1	0	1	1	0	0	0	0	1	0	1	1	1	0	1	0	0
Base64	6-bit	16						54						5						52					
	Character	Q						2						F						0					
		81(0x10)						50 (0x36)						70 (0x05)						48 (0x34)					

Table 2 UTF-8 and base64 character encoding.

(Note: UTF-8 is a type of encoding defined by Unicode which defines standards to maintain consistency in representing text in the computing industry around the world.)

(ii) OAuth2.0 authentication: `requests.post` function syntax for OAuth2.0 authentication is of the format similar to that of API_Key request.post, except for the header section and a couple of other arguments such as Device Id and API_KEY as the `requests.post` is sent from emulator after the authentication process is completed. A typical requests.post for Oauth2.0 is as follows:

```
 res=requests.post(“https://us-central1-uw-cive-700-
1.cloudfunctions.net/OAuth_Demo_ML?timestamp=31-07-2019T12:12:12”,  json={"rms":
"10.11", "spectral_peak": "33.23", "label": "-1", "hydrant" : "h3"}, headers={})
```

Data analysis and decision-making layer

To use the GCP subcomponents, the first step is to enable them on the GCP website. The coding details on various GCP components were based on the documentation provided by Google on GCP[8].

Ingestion

Cloud Function

Cloud Function acts as the ingestion component for GCP, which is a web application written in Flask framework running Python code. *Cloud Function* in this thesis is customized to handle HTTP POST requests using the trigger type HTTP. To run the Python code, the library dependencies or the additional code needed were added in requirements.txt on *Cloud Functions* user interface(UI), as shown below:

```
# Function dependencies, for example:
# package>=version
google-cloud
google-cloud-pubsub
```

requirements.txt file is a file containing a list of items to be installed by pip install (which is a package manager for Python) before the Python code is actually run on VM. The above code snippet shows that Google Cloud and Cloud Pub/Sub packages need to be installed on the virtual machine (VM) where the code is run on the cloud. So before the code is run the above mentioned packages are installed on the VM when the Cloud Function code is run on VM.

For the API key case, environmental variables `API_KEY`, `DEVICE_ID`, `X_AUTH_TOKEN` `wer`e specified. `API_KEY` was used to check whether the POST request sent contained the correct key.

The `DEVICE_ID` was used to check whether the device identification numbers sent by the emulators are correct. For OAuth2.0 request, API key check and device identification check are not made, as the authentication check is made by the Google server only once and only if the client passes the authentication the HTTP request reaches *Cloud Function*.

Once the HTTP client was authenticated, the JSON data was verified by checking whether all the key values such as `rms, spectral_peak, label, hydrant` were present along with checking for the presence of time stamp that is a part of the URL in the HTTP request. If all the checks passed, then an `‘200: okay’` response was sent back to the calling function, either the Python client (in case of laptop emulator) or to a Hologram application (in case of Raspberry Pi 3 emulator). If the checks failed, then corresponding error messages were sent back to the calling function. If all the checks were passed, before sending `‘200:ok’` response back to calling function, *Cloud Function* created a new JSON and converted it into a byte string (as *Cloud Pub/Sub* could take the data only in byte string format as shown below) for Cloud Pub/Sub to ingest the data. This new JSON consisted of all the key value pairs from the emulators along with the timestamp that was a part of the URL in the HTTP post request sent by emulators.

```
b'{"rms":"10.11","spectral_peak":"33.23","label":"-
1","hydrant":"h3","date_time":"2019-06-30T00:21:28"}'
```

The JSON that was forwarded to *Cloud Pub/Sub* or any error message sent back to calling function could all be seen in *Cloud Function* log files.

Storage

Cloud Pub/Sub

On *Cloud Pub/Sub* UI, a topic “hydrophone” was created prior to the first request from the Python/Hologram client. When the messages arrive at *Pub/Sub*, it can be viewed on Pub/Sub page on GCP website. When the *Cloud DataFlow* job (which is a subscriber to the topic “hydrophone”) was created, it appeared on the list of subscribers on *Cloud Pub/Sub* subscriber list.

Cloud storage

As mentioned previously, *Cloud Storage* serves as both permanent and temporary storage functions in the GCP. On the GCP web UI, a bucket was created to act as a container for both folders and files by specifying a project ID as the name. Subsection *Cloud ML* describes how the ambient and leak data cases were uploaded along with a model file, stored and then used to train the *Cloud ML* for predictions on stream and batch data. Similarly, a temporary folder was created on *Cloud Storag*e and used to store the files generated during *Cloud DataFlow* execution. To create temporary storage, the storage location is provided as a command line parameter (as `"--temp_location gs://uw-cive-700-1/tmp/")` during the executing of *Cloud DataFlow* code. These files and folders are accessed by the programs by specifying the fully qualified name of that file or folder, as given by: `gs://<bucket-name>/foldername1/foldername2/../filename`.

Data analysis

Cloud Dataflow

Cloud Dataflow constitutes the data analysis component in the overall GCP architecture. Two types of *Cloud Dataflow* pull subscribers are used in this thesis:

(i) Case 1: First pull subscriber calculates prediction on each data element of the stream and outputs the prediction to a *BigQuery* table.

(ii) Case 2: The second subscriber maps the incoming JSON to an output table from which a Python script running on *AI Platform Notebook* selects a set of rows between time stamps specified and performs batch prediction on the selected data.

Case 1: This *Cloud Dataflow* code was developed and tested initially on a local machine prior to its deployment on GCP. Since it was a user defined code, a service account was created so that the code could access other Google APIs such as *Cloud Pub/Sub*, *BigQuery*, *Cloud Storage* and *Cloud ML*. Service account is the identity that applications use for authentication with Google API, without embedding any keys or user credentials in the application code. The service account key file `<service-account-filename.JSON>` was obtained by creating a service account and providing the details such as the name of the service account, the level of permission. The environment variable `GOOGLE_APPLICATION_CREDENTIALS` was set to use the service account information in the code, by running the following command on the local command prompt:

```
set GOOGLE_APPLICATION_CREDENTIALS="<path to keyfile.JSON>”.
```

Apache-beam framework required to run the *Cloud Dataflow* code was installed using the following command:

```
<path of the python directory>python.exe –m pip install apache-beam[gcp]
```

Before deploying the *Cloud Dataflow* code on GCP, the code was run on a local machine, using Python which ran on Python Integrated Development Environment (IDE) PyCharm 2018.3.5. In the configuration section, provided the following command line parameters which were needed by the *DataFlow* program.

```
--input_topic projects/uw-cive-700-1/topics/hydrophone --output_table "uw-cive-
700-1:demo3.test"  --project  uw-cive-700-1  --temp_location  gs://uw-cive-700-
1/tmp/ --staging_location gs://uw-cive-700-1/staging/ --runner DataFlowRunner
```

`--input_topic:`	fully qualified path of *Cloud Pub/Sub* topic
`--output_table:`	fully qualified name of the *BigQuery* where the prediction result along with other values are stored.
`--project:`	project name
`--temp_location:`	location on the *Cloud Storage* where the intermediate files from cloud dataflow execution are stored.
`--staging_location:`	*Cloud Storage* path for staging the binary file of the *Cloud Dataflow* code.
`--runner:`	`DirectRunner/DataFlowRunner`

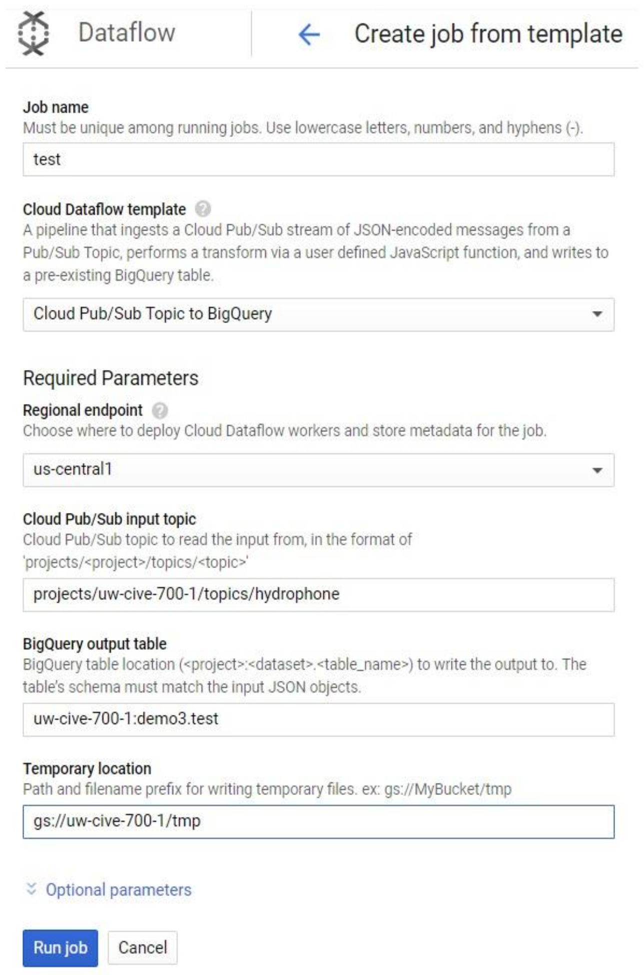
A fully qualified topic has the structure projects/<project-name>/topics/<topic_name>. The project name used in this thesis was uw-cive-700-1 and the topic name is hydrophone. A fully qualified table has the structure <project-name>:<dataset-name>.table-name. A project consists of one or more data sets, where the data set consists of one or more tables.

Dataset-name and table names used were demo3 and test, respectively. So the fully qualified table name is `uw-cive-700-1:demo3.test.` `temp_location` was taken as `gs://uw-cive-700-1/tmp/, where tmp` is a folder under the bucket `uw-cive-700-1`. While the bucket and project names can be different, in this thesis they were retained to be the same. Bucket storage class was regional and us-cental1-a was selected for the region and zone. A folder named `staging` under the bucket was used for staging the *Cloud Dataflow* code executable file. Streaming was enabled in the

Python code for Cloud Dataflow. The *Cloud Dataflow* code was executed on local machine with –runner set to `DirectRunner in the` command line parameters. After it was run successfully, the same *Cloud Dataflow* code was run on local machine with runner set to `DataFlowRunner in` command line parameters. When the code was executed, a pipeline was created on the cloud. Each data element of the incoming stream was decoded to utf-8 and keys and values of JSON were extracted by the *Cloud Dataflow* code. Values of the keys RMS and spectral peak were presented to the machine learning module, which in turn calls the training module and returns the prediction results to *Cloud Dataflow*. JSON data containing results and corresponding features and label, was sent to *Bigquery* to add it to the table `test`. An example of JSON data sent is given below:

```
{"rms":"10.11","spectral_peak":"33.23","label":"-1","hydrant":"h3",″result″:″-1″,"date_time":"2019-06-30T00:21:28"}
```

<u>Case 2</u>: For mapping input stream data to *BigQuery* output table, a readily available dataflow template provided by GCP was used. This type of job was created by selecting “Cloud Pub/Sub topic to BigQuery” option on the template list and providing all the necessary details such as fully qualified *BigQuery* table name, name of the *Cloud Dataflow* job, temporary *Cloud Storage* location, region, fully qualified *Cloud Pub/Sub* topic name as in Case 1. The template to create a *Cloud Dataflow* job using the GCP provided template is shown in Figure 1.



Job name
Must be unique among running jobs. Use lowercase letters, numbers, and hyphens (-).
test

Cloud Dataflow template
A pipeline that ingests a Cloud Pub/Sub stream of JSON-encoded messages from a Pub/Sub Topic, performs a transform via a user defined JavaScript function, and writes to a pre-existing BigQuery table.
Cloud Pub/Sub Topic to BigQuery

Required Parameters

Regional endpoint
Choose where to deploy Cloud Dataflow workers and store metadata for the job.
us-central1

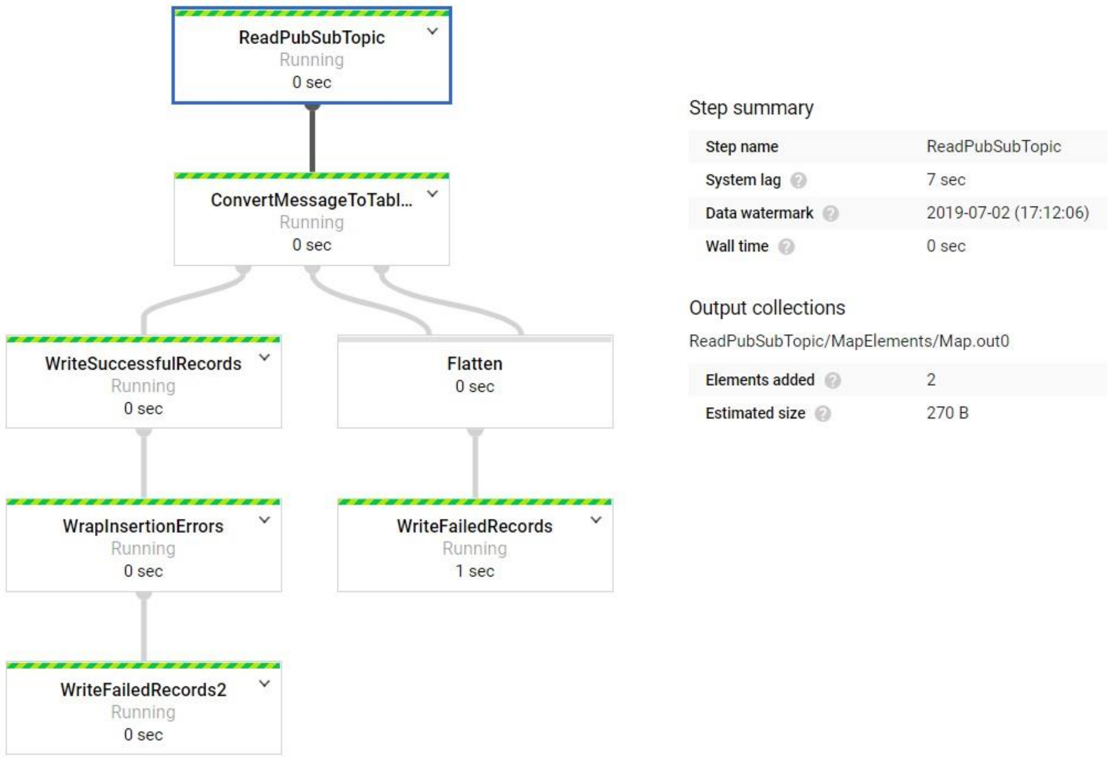
Cloud Pub/Sub input topic
Cloud Pub/Sub topic to read the input from, in the format of 'projects/<project>/topics/<topic>'
projects/uw-cive-700-1/topics/hydrophone

BigQuery output table
BigQuery table location (<project>:<dataset>.<table_name>) to write the output to. The table's schema must match the input JSON objects.
uw-cive-700-1:demo3.test

Temporary location
Path and filename prefix for writing temporary files. ex: gs://MyBucket/tmp
gs://uw-cive-700-1/tmp

Figure 1 Dataflow job creation template.

When the job is running, it can be monitored used the job details shown in the Figure 2(a) and (b). Each operation on the data is represented by a rectangular box, which when highlighted as shown in Figure 2(a) provided the details such as how many data elements are processed and number of bytes processed. Similar pipeline appears for Case 1 as well when the job was deployed on to the GCP. Fig. 2(b) provides the overall job summary including Apache-beam version used, memory usage and CPU usage. Errors/any execution details at any stage of execution could be seen using Stackdriver logs.



Step summary

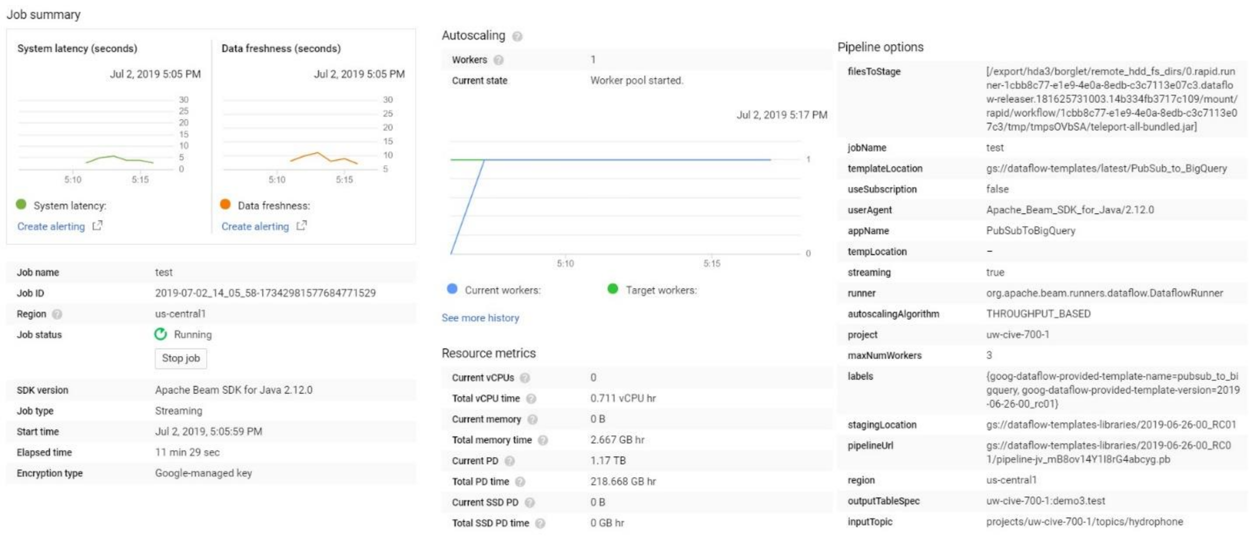
Step name	ReadPubSubTopic
System lag	7 sec
Data watermark	2019-07-02 (17:12:06)
Wall time	0 sec

Output collections

ReadPubSubTopic/MapElements/Map.out0

Elements added	2
Estimated size	270 B

Figure 2 (a). Dataflow job details for monitoring the DataFlow execution



Job name	test
Job ID	2019-07-02_14_05_58-17342981577684771529
Region	us-central1
Job status	Running Stop job
SDK version	Apache Beam SDK for Java 2.12.0
Job type	Streaming
Start time	Jul 2, 2019, 5:05:59 PM
Elapsed time	11 min 29 sec
Encryption type	Google-managed key

Autoscaling

Workers	1
Current state	Worker pool started.

Resource metrics

Current vCPUs	0
Total vCPU time	0.711 vCPU hr
Current memory	0 B
Total memory time	2.667 GB hr
Current PD	1.17 TB
Total PD time	218.668 GB hr
Current SSD PD	0 B
Total SSD PD time	0 GB hr

Pipeline options

filesToStage	[/export/hda3/borglet/remote_hdd_fs_dirs/0.rapid.runner-1cbb8c77-e1e9-4e0a-8edb-c3c7113e07c3.dataflow-releaser.181625731003.14b334fb3717c109/mount/rapid/workflow/1cbb8c77-e1e9-4e0a-8edb-c3c7113e07c3/tmp/tmpsOVbSA/teleport-all-bundled.jar]
jobName	test
templateLocation	gs://dataflow-templates/latest/PubSub_to_BigQuery
useSubscription	false
userAgent	Apache_Beam_SDK_for_Java/2.12.0
appName	PubSubToBigQuery
tempLocation	–
streaming	true
runner	org.apache.beam.runners.dataflow.DataflowRunner
autoscalingAlgorithm	THROUGHPUT_BASED
project	uw-cive-700-1
maxNumWorkers	3
labels	{goog-dataflow-provided-template-name=pubsub_to_bigquery, goog-dataflow-provided-template-version=2019-06-26-00_rc01}
stagingLocation	gs://dataflow-templates-libraries/2019-06-26-00_RC01
pipelineUrl	gs://dataflow-templates-libraries/2019-06-26-00_RC01/pipeline-jv_mB8ov14Y1I8rG4abcyg.pb
region	us-central1
outputTableSpec	uw-cive-700-1:demo3.test
inputTopic	projects/uw-cive-700-1/topics/hydrophone

Figure 2 (b). Dataflow job details for monitoring the *DataFlow* execution.

Cloud ML

Prior to deployment on the cloud, the following steps were taken so that *Cloud ML* is accessible by *Cloud Dataflow* to obtain the prediction results.

(i) *AI-Platform Notebook* API was enabled.

(ii) On the web UI, Cloud Shell, which is a command line interface (CLI) with Cloud SDK was used to run the commands in the following steps.

(iii) `gcloud` package was updated by running the command `gcloud components update`.

(iv) Scikit-learn and Pandas packages were installed by running the following command:

```
i. pip install --user scikit-learn pandas
```

(v) Following commands were run to set the environment variables:

```
BUCKET_NAME="uw-cive-700-1"
JOB_NAME="hydrant3_scikit_learn_20190624_115958"
JOB_DIR=gs://$BUCKET_NAME/
MODEL="Hydrant3_train_sklearn"
MODEL_NAME="Hydrant3_train_sklearn"
MODEL_DIR="uw-cive-700-1/hydrant3_20190624_171650"
TRAINING_PACKAGE_PATH="./hydrant3_sklearn_trainer/"
MAIN_TRAINER_MODULE="hydrant3_sklearn_trainer.hydrant4_sklearn"
REGION=us-central1
PYTHON_VERSION=3.7
SCALE_TIER=BASIC
PROJECT_ID="uw-cive-700-1"
FRAMEWORK="SCIKIT_LEARN"
INPUT_FILE="input.json"
TEXT_INSTANCES="text_file"
VERSION_NAME="v1"
RUNTIME_VERSION=1.13
```

(vi) `echo` command was run on each of the above set environment variables to determine if the variables show the values that were set.

(vii) `Hydrant3_sklearn.py` file which trains the model was written under the folder `hydrant3_sklearn_trainer`. The training part of the code was tested on local machine before it was uploaded on to GCP. The program `hydrant3_sklearn.py` contains the code `subprocess.check_call(['gsutil', 'cp', os.path.join(data_dir,'AmbHydr3'), 'AmbHydr4'], stderr=sys.stdout) and subprocess.check_call(['gsutil', 'cp', os.path.join(data_dir,'LeakHydr3'), 'LeakHydr3'], stderr=sys.stdout)` to upload the local training data file `AmbHydr3 and LeakHydr3` on to *Cloud Storage*. The program reads the above mentioned files and trains the model.

`(viii)` The following command was run and its success status was tracked on GCP console/webUI under *AI-Platform Jobs* section. This command exported the model to `model.joblib` file and stored it on Cloud Storage under the path `uw-cive-700-1/hydrant3_20190624_171650`.

```
gcloud ai-platform jobs submit training $JOB_NAME --job-dir $JOB_DIR --package-path $TRAINING_PACKAGE_PATH --module-name $MAIN_TRAINER_MODULE --region $REGION --runtime-version=$RUNTIME_VERSION --python-version=$PYTHON_VERSION --scale-tier $SCALE_TIER
```

AI-Platform organized the trained models and their versions. This was done by creating a model and its version and linking them to `model.joblib`.This `model.joblib` is used by *Cloud Dataflow* and *AI Platform Notebook* Python code for prediction.

Visualization of results

BigQuery

Big query acts as a permanent storage for data and data analysis results. It provides data for batch processing as well as for result visualization. It stores the data in the table format which can accessed using standard query language (SQL) queries. An example of a simple SQL query to retrieve all the rows of a table `test` that belongs to data set `demo3` and project `uw-cive-700-1,` between current time and last 10 minutes is given below,

```
select * from `uw-cive-700-1.demo3.test` WHERE ((DATETIME_DIFF(CURRENT_DATETIME,
PARSE_DATETIME('%Y-%m-%dT%H:%M:%SZ',  date_time), MINUTE))) <= 10;
```

AI Platform Notebook

AI Platform Notebook provides an interactive environment that comes integrated with machine learning tools such as scikit-learn and tensorflow for writing python code to perform data analytics. In this thesis, *AI Platform Notebook* was used to execute both batch analytics and extracting stream processing results from *BigQuery*. To access *BigQuery*, its library was installed by adding a line `!pip install --upgrade google-cloud-bigquery[pandas]` at the top of the *AI Platform Notebook* instance. SQL query similar to the one mentioned in the previous section was used to retrieve table rows that contained the data that was used in batch processing and visualization of results.

Appendix B

One Class Support Vector Machine (OCSVM)

OCSVM[9] is an unspervised outlier detection method. Different kernel methods can be specified for decision making. Various kernel methods available for OCSVM are linear, polynomial, radial basis function (RBF) and sigmoid. They differ in the way they map the training data point to higher dimensions to create hyperplane decision boundary between the classes of data set. The default kernel used for OCSVM is RBF Gaussian. This default setting is used in this thesis. RBF is a real valued function whose value depends only on the distance from some point c in the Euclidean space, so that h (x, c) = h (‖x - c‖). If c is the origin, c=0, and h (x) = h (‖x‖). Commonly used RBF is of the Gaussian form which is:

$$h(x) = \sum_{i=1}^{N} w_i \exp(-\gamma \|x - x_i\|^2)$$

where, x is an observation or data to be evaluated. N is the number of Gaussian kernels which is usually equal to number of data points in the training dataset. x_i is the centre of ith Gaussian curve of the function h(x). w_i is the ith weight that decides influence of Gaussian curve with centre x_i on x. Curve with higher weight has higher influence on x. x belongs to the curve with highest weight which in turn decides which class x belongs to. γ(gamma) must be greater than 0. γ is inversely proportional to the width of Gaussian curve. Higher the value of γ, narrower the curve is. This leads to poor generalization. So care needs to be taken to find a good γ that leads to good generalization. In this thesis, as Python is used in classifying the data, using ocsvm, the details of python OneClassSVM class, its variables, methods are given below. OneClassSVM is a class of sklearn.svm module. The following table provides various variables that belong to the class, their values and meanings.

class sklearn.svm.OneClassSVM(kernel=’rbf’, degree=3, gamma=’auto_deprecated’, coef0=0.0, tol=0.001, nu=0.5, shrinking=True, cache_size=200, verbose=False, max_iter=-1, random_state=None)

Type	**Name**	**Value**	**Meaning**
Parameters	Kernel	string, optional (default=’rbf’)	Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used. If a callable is given it is used to precompute the kernel matrix.
	degree	int, optional (default=3)	Degree of the polynomial kernel function (‘poly’). Ignored by all other kernels.
	Gamma	float, optional (default=’auto’)	Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. Current default is ‘auto’ which uses 1 / n_features, if gamma='scale' is passed then it uses 1 / (n_features * X.var()) as value of gamma. The

			current default of gamma, ‘auto’, will change to ‘scale’ in version 0.22. ‘auto_deprecated’, a deprecated version of ‘auto’ is used as a default indicating that no explicit value of gamma was passed.
	coef0	float, optional (default=0.0)	Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’.
	Tol	float, optional	Tolerance for stopping criterion.
	Nu	float, optional	An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval (0, 1]. By default 0.5 will be taken.
	Shrinking	boolean, optional	Whether to use the shrinking heuristic.
	cache_size	float, optional	Specify the size of the kernel cache (in MB).
	verbose	bool, default: False	Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

	max_iter	int, optional (default=-1)	Hard limit on iterations within solver, or -1 for no limit.
	random_state	int, RandomState instance or None, optional (default=None)	Ignored. Deprecated since version 0.20: random_state has been deprecated in 0.20 and will be removed in 0.22.
Attributes	support_	support_ : array-like, shape = [n_SV]	Indices of support vectors.
	support_vectors_	support_vectors_ : array-like, shape = [nSV, n_features]	Support vectors.
	dual_coef_	array, shape = [1, n_SV]	Coefficients of the support vectors in the decision function.
	coef_	array, shape = [1, n_features]	Weights assigned to the features (coefficients in the primal problem). This is only available in the case of a linear kernel. coef_ is readonly property derived from dual_coef_ and support_vectors_
	Intercept_	array, shape = [1,]	Constant in the decision function
	offset_	float	Offset used to define the decision function from the raw scores. We have the relation: decision_function = score_samples - offset_.

			The offset is the opposite of intercept_ and is provided for consistency with other outlier detection algorithms.

The Following table provides the methods that belongs to OneClassSVM class, their input parameters, return values and the meaning of the method.

Method Name	**Parameters**	**Return values**	**Meaning of the methods**
decision_function (self, X)	X:array-like, shape (n_samples, n_features)	dec: array-like, shape (n_samples,)	Signed distance to the separating hyperplane. Returns the decision function of the samples.
fit(self, X, y=None, sample_weight=None, **params)	X: {array-like, sparse matrix}, shape (n_samples, n_features) Set of samples, where n_samples is the number of samples and n_features is the number of features.	self : object	Detects the soft boundary of the set of samples X. If X is not C-ordered contiguous array it is copied.
	sample_weight: array-like, shape (n_samples,) Per-sample weights. Rescale C per sample. Higher weights force the classifier to put		

	more emphasis on these points.		
	y: Ignored, not used, present for API consistency by convention.		
fit_predict (self, X[, y])	X: ndarray, shape (n_samples, n_features)Input data.	y: ndarray, shape (n_samples,) 1 for inliers, -1 for outliers.	Performs fit on X and returns labels for X. Returns -1 for outliers and 1 for inliers.
	y:Ignorednot used, present for API consistency by convention.		
get_params (self[, deep])	deep: boolean, optional If True, will return the parameters for this estimator and contained sub-objects that are estimators.	params: mapping of string to any Parameter names mapped to their values.	Get parameters for this estimator.
predict (self, X)	X: {array-like, sparse matrix}, shape (n_samples, n_features) For kernel=”precomputed”, the expected shape of X is [n_samples_test, n_samples_train]	y_pred: array, shape (n_samples,) Class labels for samples in X.	Perform classification on samples in X. For a one-class model, +1 or -1 is returned.

score_sample (self, X)	X : array-like, shape (n_samples, n_features)	score_samples: array-like, shape (n_samples,) Returns the (unshifted) scoring function of the samples.	Raw scoring function of the samples.
set_params (self, **params)		self	Set the parameters of this estimator. The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Appendix C

Glossary

API: It is a set of subroutine definitions to communicate between different components of a system.

Containers: These are isolated user space instances that look like real computers from the view point of programs running on them but they are virtual environments provided by an operating system. A real computer can see all the devices connected to it however, a container can see only the devices assigned to it by the operating system.

Distributed computers: These computers are located on different networks but can communicate and co-ordinate their actions by passing messages to one another. They interact with one another to achieve a common goal. Significant characteristics of distributed system are concurrency of computers, lack of a global clock, independent failure of computers or components.

Kernel Methods: In machine learning, kernel methods are a class of algorithms for pattern analysis. The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets. In case of classification, kernels map data points to higher dimensional space via some transformation and linearly separable data points are found that helps to find hyperplane decision boundary that can classify the data into different classes.

Machine Learning: Machine learning (ML) is a subset of Artificial Intelligence and is a scientific study of algorithms and statistical models that are based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.

Micromachines: Micromachines are the mechanical objects generally between 100 nanometres too 100 micrometers in size and are fabricated in the same way as integrated circuits.

Semi-supervised Learning: It is a machine learning task of learning a function from combination of (usually a small amount of) labelled and (a large amount of) unlabelled datasets. When it is not possible to get fully labelled datasets, semi-supervised learning is used.

SPI protocol: It is a synchronous serial communication interface specification used for short distance communication in embedded systems.

Supervised Learning: Supervised learning is the machine learning task of learning a function from labelled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object or vector consisting of features and a desired output value. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples.

Unsupervised Learning: It is a machine learning task of learning a function from unlabelled data. It usually involves grouping of data sets that are not labelled based on commonalities in the data and reacts to new data set based on the presence or absence of those commonalities.

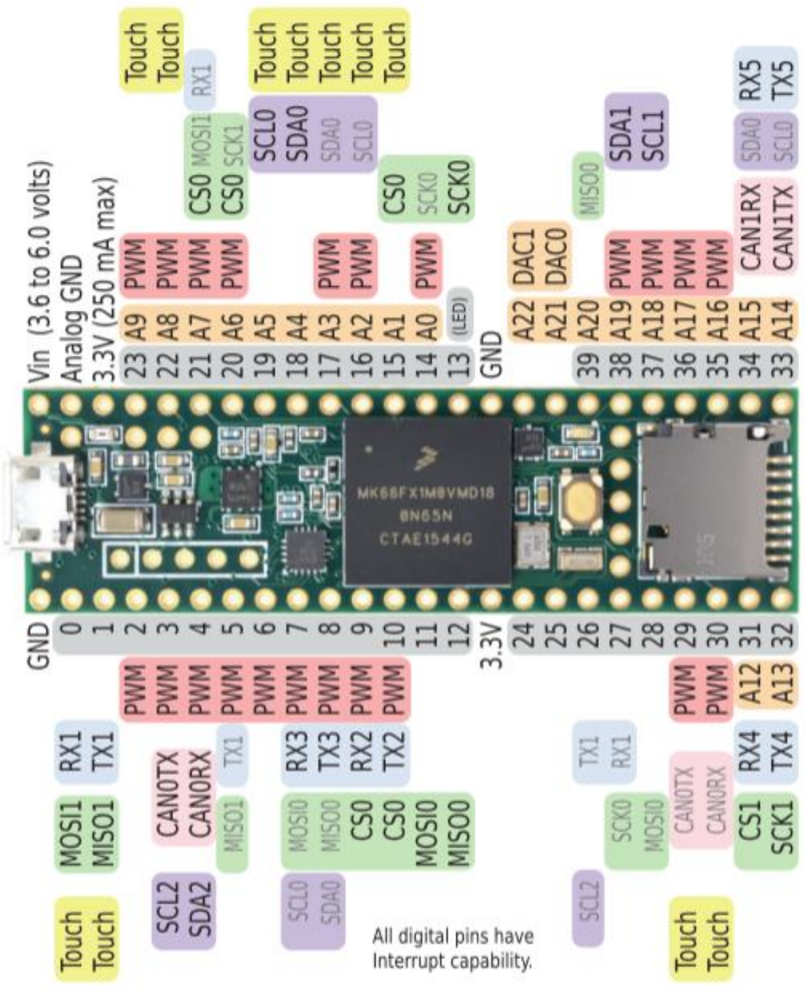
Virtual machine: It is an emulation of a computer system, i.e., it is based on computer architecture and provides functionality of a physical computer. Both hardware and software are involved in virtual machine implementation.

Webhook: It is a way in which one web application provides information to other web application in real time. Data is sent in JSON format by webhooks. Webhooks use HTTP POST request to communicate.

Interoperability: The ability of two or more systems or components to exchange information and to use the information that has been exchanged

Appendix D

Hardware Datasheets



Teensy 3.6 Pinouts

SQ26-13

Tested at 51Hz
Input Impedance of measurment device 30MOhm

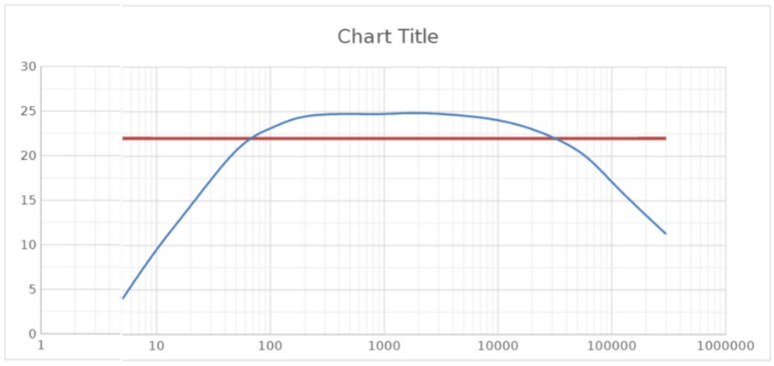
Sensitivity (dB) re 1V/µPa
-175.18

Using 4 Vdc input to the resistive component of RC network.

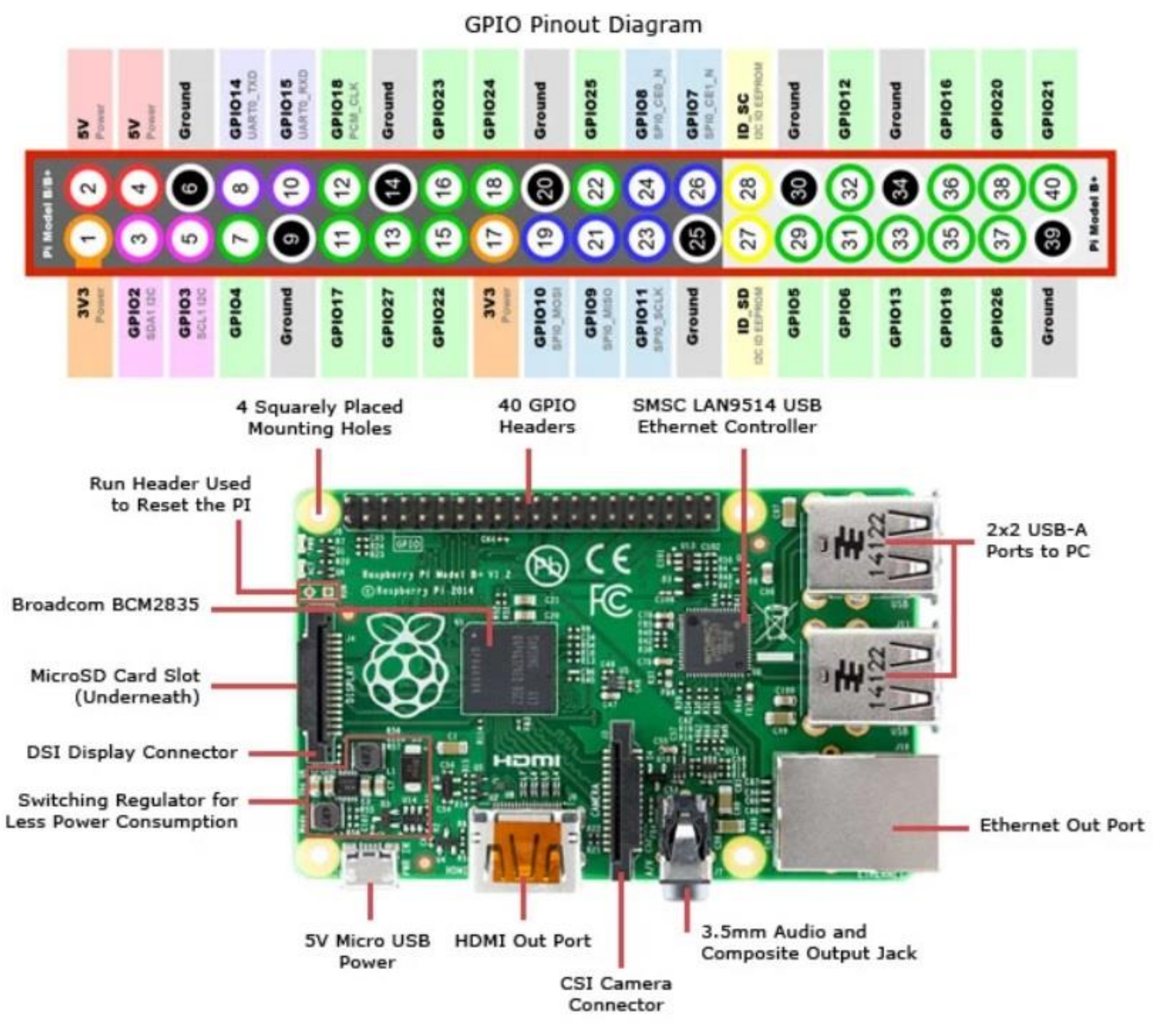
Frequency	Input Volta	Output vol	Gain	3 dB
5	0.0152	0.024	3.967353	22
10	0.0188	0.0563	9.527011	22
20	0.0208	0.11	14.46659	22
50	0.0212	0.228	20.63198	22
100	0.0212	0.304	23.13075	22
200	2.06E-02	0.344	2.45E+01	22
500	2.06E-02	0.356	2.48E+01	22
1000	2.06E-02	0.356	2.48E+01	22
2000	2.06E-02	0.36	2.48E+01	22
5000	2.06E-02	0.348	2.46E+01	22
10000	2.06E-02	0.328	2.40E+01	22
20000	2.06E-02	0.292	2.30E+01	22
50000	2.06E-02	0.222	2.06E+01	22
100000	2.06E-02	0.148	1.71E+01	22
200000	2.06E-02	0.096	1.34E+01	22
300000	0.0202	0.074	1.13E+01	22

Gain = 25 dB
Bandwidth: 65 Hz - 35 kHz
Input Noise at 1 kHz:
Maximum Input/Output Voltage at 1 kHz:

Using a 2k/2.2uF RC network, 20 mVpp, 4 Vdc



Hydrophone, SQ26-13 DataSheet



Raspberry Pi 3 Pinouts

Data Sheet and Hardware Reference

Hologram Nova Global IoT Cellular USB Modem
V1.4

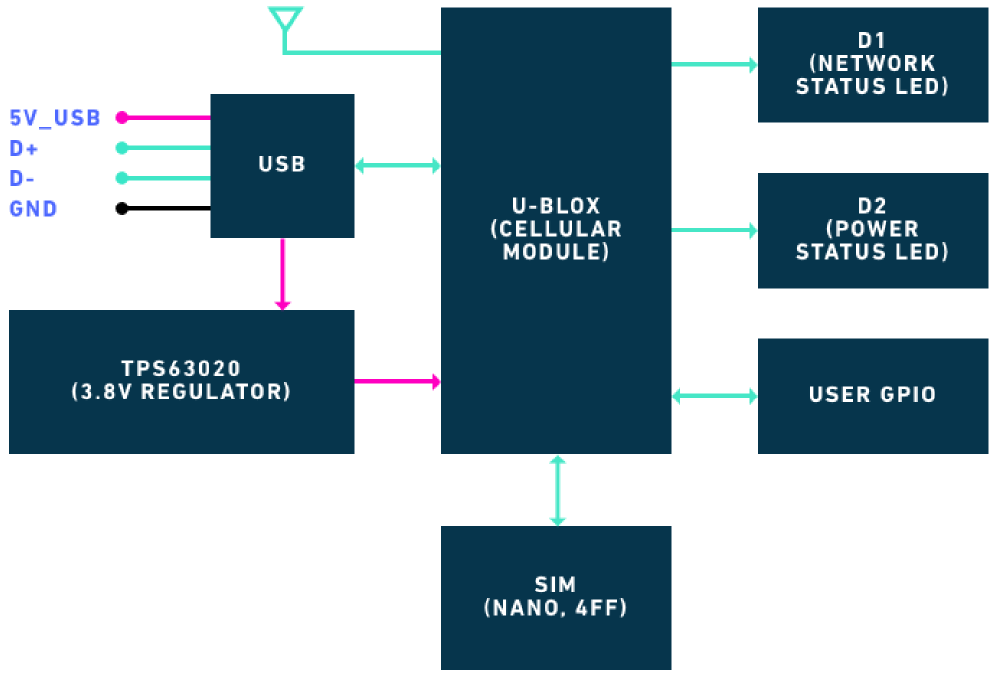
Issue: 04
Date: 2018-09-07

Nova Data Sheet and Hardware Reference

TABLE OF CONTENTS

System Block Diagrams

Block diagram of the Nova board:



Input/Output Characteristics

USB

The Hologram Nova is designed to provide easy access to the u-blox SARA-U201 and SARA-R410-02B USB interface.

SARA series modules include a high-speed USB 2.0 compliant interface with maximum 480 Mb/s data rate. The module itself acts as a USB device and can be connected to any USB host. The USB is the suitable interface for transferring high speed data between SARA-U2 series and a host processor, available for AT commands.

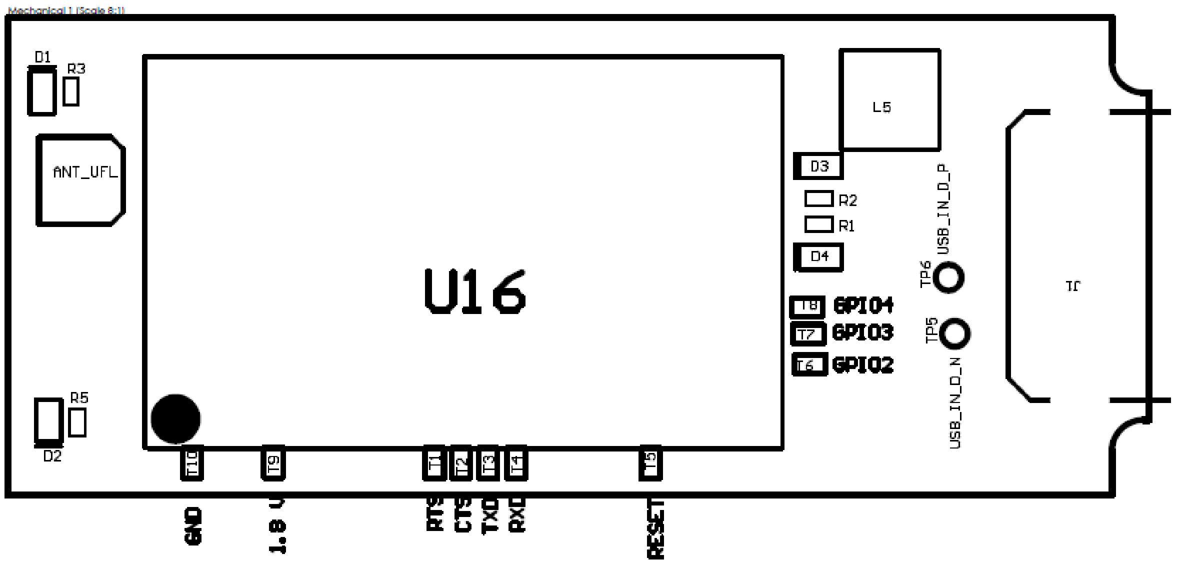
The USB_D+ / USB_D- lines carry the USB serial data and signaling. The USB interface is automatically enabled by an external valid USB VBUS supply voltage (5.0 V typical) applied on the VUSB_DET pin.

For additional details, please see the following datasheets:
u-blox SARA-U201 datasheet
u-b lox SARA-R4 Series datasheet

UART

At Hologram, we believe in providing an open platform for developers to build hardware. To support this mission, the Nova exposes the u-blox modem's UART interface as solderable pads on the top half of the board. For more advanced hardware devlopment, this provides direct access to the u-blox modem which runs at 1.8V

Note: USE UART PADS AT YOUR OWN RISK. Pads are directly connected to the u-blox modem so using these I/O or improperly handling the board runs the risk of damaging the u-blox modem. Additionally, we do not officially provide support this interface.



Technical Specifications

Absolute Maximum Ratings (Power Inputs)

Stressing the device above one or more of the ratings listed in the Absolute Maximum Rating section may cause permanent damage. These are stress ratings only. Operating the device at these or at any conditions other than those specified in the Operating Conditions should be avoided. Exposure to Absolute Maximum Rating conditions for extended periods may affect device reliability.

Symbol	Description	Min.	Max.	Unit
VCC, USB 5V	Input DC voltage at VCC pins	-0.30	5.50	V
USB D+/D- line	Input DC voltage at USB_D+/D- pins	-1.00	5.35	V
UBLOX_RTS UBLOX_CTS UBLOX_TXD UBLOX_RXD	Input DC voltage at u-blox digital interface pins	-0.30	3.60	V
UBLOX_RESET_N	Input DC voltage at u-blox RESET_N pin	-0.15	2.10	V
GPIO2 GPIO3 GPIO4	Input DC voltage at u-blox GPIO pins	-0.30	3.60	V

For power draw characteristics under certain cellular conditions, please view respective u-blox datasheets.

Operating Conditions

The Hologram Nova is designed to operate within temperatures between -45°C to 85°C. It is not designed to withstand material contact with moisture or any other conductors, aside from intended use of the USB. The Hologram Nova may be installed into appropriate enclosures that can protect the device from heat, cold, moisture, and humidity for Industrial use.

If handling the Nova circuit board directly, please do so in an ESD-safe environment and wear ESD protection.

Radio Specifications

The Hologram Nova platform features cellular modems which support a global list of 2G, 3G, and LTE Cat-M1/NB-IoT frequencies.

Nova 3G/2G (SARA-U201)

- 3G Bands:
    - Americas: Band 5 (850MHz), Band 2 (1900MHz)
    - Europe/Asia/Africa: Band 8 (900MHz), Band 1 (2100MHz)
- 2G Bands
    - GSM -850MHz
    - E-GSM -900MHz
    - DCS - 1800 MHz
    - PCS - 1900 MHz

Nova LTE-M & NB-IoT (SARA-R410M-02B)

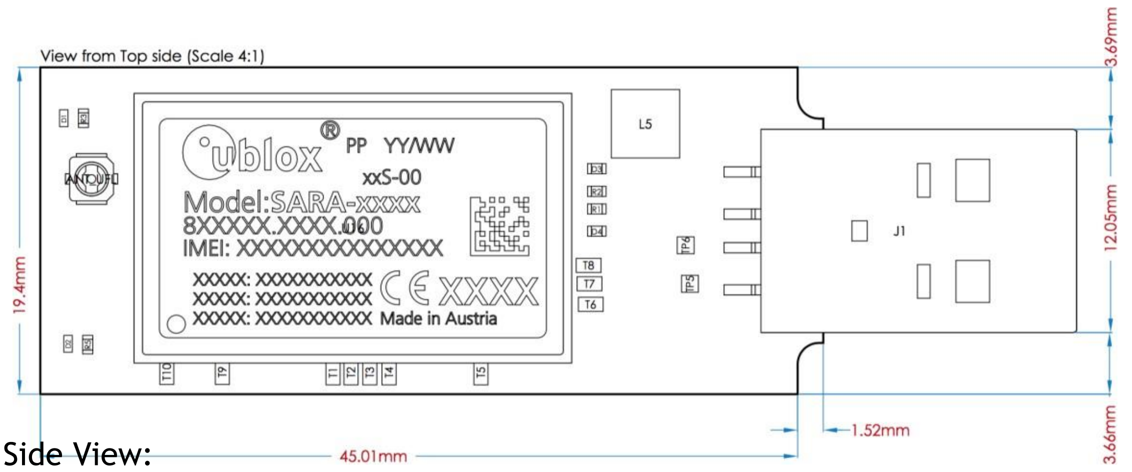
- LTE Cat-M1/N1 Bands:
    - LTE FDD: 1, 2, 3, 4, 5, 8, 12, 13, 17, 18, 19, 20, 25, 26, 28

Mechanical Dimensions

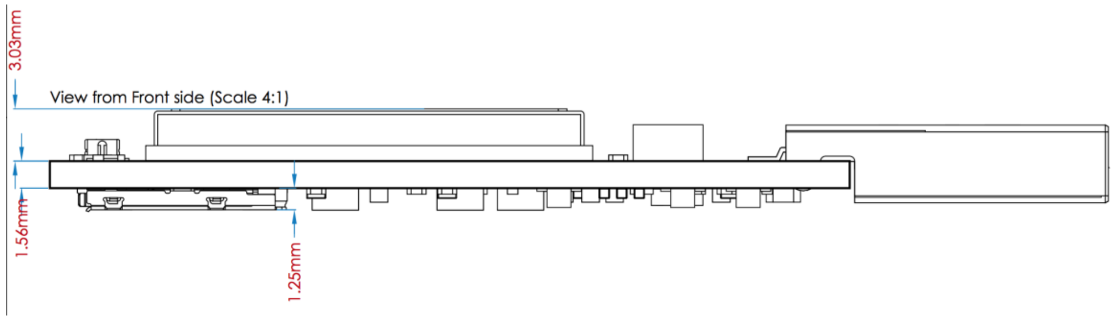
The Hologram Nova board without an enclosure is:

- Length: 61.58 millimeters
- Width: 19.4 millimeters
- Height: 5.84 millimeters (maximum height) Below

are views of the Hologram from the top and side. Top View:



Side View:



LEDs

The Nova has two clear LEDs for providing power and connectivity feedback. A power LED that represents whether the modem is on or off, and a network LED that indicates the current network status.

Note: Nova LTE-M & NB-IoT modem and Nova 3G/2G modem have same placement of LED color indicators but opposite use for power/network indication. Please use below table for reference.

MODEL	POWER LED	NETWORK LED
Nova 3G/2G	Red	Blue
Nova LTE-M & NB-IoT	Blue	Red

- Power LED status indicator (Red – 3G/2G, Blue – LTE-M & NB-IoT)
    - On: USB 5V is connected and the Nova is powered on
    - Off: USB 5V is not connected and the Nova is not powered on (modem may take up to 30s to boot up and power the LED)
- Network LED status indicator (Blue – 3G/2G, Red – LTE-M & NB-IoT)
    - On, solid: Nova connected in active data session
    - On, rapid blink: 3G network detected (3G/2G Nova Only)
    - On, double blink: 2G network detected (3G/2G Nova Only)
    - Off: No network detected
        - Make sure antenna is securely connected, positioned to receive cell signal and SIM properly inserted
        - Device can take up to 200s to detect available networks

Antennas

The Nova is made for ultimate flexibility and this extends to available antennas. Each model's included antenna characteristics are listed below:

Nova 3G/2G (SARA-U201)

- Black, flexible antenna (Sinbon A9702472)
    - Size: 37x7x1mm
    - Weight: <1g
    - Connector: U.FL
    - Mounting: Adhesive 3Mtape
    - Temperature: -40C -+85C

Nova LTE-M & NB-IoT (SARA-R410)

- Black, flexible antenna (Pulse PN W3907B0100)
    - Size: 111.70x20.4x1mm
    - Weight: <1g
    - Connector: U.FL
    - Mounting: Adhesive 3Mtape
    - Temperature: -40C -+85C

The Nova can also be used with additional antennas. If you'd like to use the Nova with an antenna which has an SMA connector, you need to purchase a UFL - SMA adapter.

Bill of Materials

DESIGNATOR	QUANTITY	MFG	MPN
ANT_UFL	1	Amphenol	A-1JB
C2, C6, C48	3	MURATA	GRM155R61C104KA88D
C11	1	SAMSUNG	CL10A225MQ8NNNC
C39	1	MURATA	GRM155R71C103KA01D
C34	1	Murata	GRM188R60J106ME84D
C46	1	AVX/ELCO	04025A150JAT2A
C36, C37, C38	3	MURATA	GRM188R60J226MEA0D
C41, C42, C43, C44, C45 (U201 Nova Only)	5	AVX	04025A470JAT2A
C40	1	KEMET	C0402C560J5GACTU
C47, C49, C50	3	AVX	F950J337MBAAQ2
L7	1	Murata	BLM18KG121TN1D
L5	1	Coilcraft	XFL4020-102MEC
D1	1	VISHAY	VLMB1500-GS08
D2	1	VISHAY	VLMS1500-GS08
D3, D4, D5	3	Littlefuse	PESD0402-140

DESIGNATOR	QUANTITY	MFG	MPN
L1	1	YAEGO	RC0603JR-070RL
R13	1	YAEGO	RC0402FR-071ML
R3, R5	2	YAEGO	RC0402FR-073KL
R4, R6, R7, R8	4	YAEGO	RC0402JR-0710KL
R1, R2	2	YAEGO	RC0402FR-0722RL
R11, R51	2	PANASONIC	ERJ-2GEJ104X
R12	1	YAEGO	RC0402FR-07150KL
R9	1	PANASONIC	ERJ-2GEJ471X
U16	1	U-BLOX	SARA-U260-00S
SIM1	1	GLOBAL CONNECTOR TECHNOLOGY	SIM8050-6-0-14-01-A
U2	1	TI	TPS63020DSJ
Q1, Q2, Q3, Q4	4	ON Semiconductor	MMBT3904LT1G
J1	1	MOLEX	480372200

Regulatory information

Carrier Specific Certifications

NOVA-U201 (3G/2G): AT&T, T-Mobile, PTCRB, GCF

NOVA-R410(LTE-M&NB-IoT):VerizonODI,AT&T,T-Mobile(Inprogress), PTCRB, GCF

Verizon Open Development Device #7721

AT&T Network Compatibility Record: 10bkv4QCDm

Export Control Classification Number (ECCN)

ECCNsarefivecharacteralpha-numericdesignationsusedontheCommerce Control List(CCL)toidentifydual-useitemsforexportcontrolpurposes.An ECCN categorizes items based on the nature of the product, i.e. type of commodity,software,or technologyanditsrespectivetechnicalparameters.

ECCN for All Nova Modems: 5A992.c

RoHS Compliance

TheNovamodemfamilycomplieswiththeRoHS(RestrictionofHazardous Substances)directiveoftheEuropeanUnion,EUDirective2011/65/EU.

Harmonized Tariff Schedule Code (HTS)

HTS Code for All Nova Modems: 8517.62.0010

Interference Statement

This device complies with Part 15 of the FCC Rules and Industry Canada licence-exempt RSS standards. Operation is subject to the following two conditions: (1) This device may not cause harmful interferences, and (2) this device must accept any interference received, including interference that may cause undesired operation.

FCC & ICCompliance

If the modem's antenna is located farther than 20cm from the human body and there are no proximate transmitters, the FCC/IC approvals of the constituent u- blox SARA-U201 or SARA-R410-02B can be reused by the end product.

Should the modems antenna be mounted closer than 20cm from the human bodyor if there are proximate transmitters, additional FCC/IC testing maybe required for the end product.

Nova 3G/2G & Nova LTE-M & NB-IoT modems make use of the underlying u-blox module's FCC & IC identification numbers below.

MODEL	FCC ID	IC ID (CERTIFICATION NUMBER)
Nova 3G/2G	XPY1CGM5NNN	8595A-1CGM5NNN
Nova LTE-M & NB-IoT	XPY2AGQN4NNN	8595A-2AGQN4NNN

Additionally, all Nova modems are compliant with FCC Part 15 Class B

Modification Statement

Hologram has not approved any changes or modifications to this device by the user. Any changes or modifications could void the user's authorization to operate the equipment.

End Product Labeling Requirements

End products utilizing Nova 3G/2G modems should be labeled with the following information:

Device Uses Approved Radio: NOVA-U201

Contains FCC ID:
XPY2AGQN4NNN Contains
IC:8595A-1CGM5NNN

This device complies with Part 15 of the FCC Rules and Industry Canada licence-exempt RSS standards. Operation is subject to the following two conditions: (1) This device may not cause harmful interferences, and (2) this device must accept any interference received, including interference that may cause undesired operation.

End products utilizing Nova LTE-M & NB-IoT modems should be labeled with the following information:

Device Uses Approved Radio: NOVA-R410

Contains FCC ID:
XPY2AGQN4NNN Contains
IC:8595A-2AGQN4NNN

This device complies with Part 15 of the FCC Rules and Industry Canada licence-exempt RSS standards. Operation is subject to the following two conditions: (1) This device may not cause harmful interferences, and (2) this device must accept any interference received, including interference that may cause undesired operation