Energy Efficient Knowledge Distribution in Wireless Sensor Networks for Query Analytics

Kurt Portelli

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

July 26, 2016

EU funds for Malta | 2014 2020

The research work disclosed in this publication may be partially funded by the
Endeavour Scholarship Scheme (Malta). Scholarships may be considered for
part-financing by the European Union - European Social Fund (ESF) -
Operational Programme II – Cohesion Policy 2014-2020
"Investing in human capital to create more opportunities and promote the well-being of society".

European Union – European Structural and Investment Funds
Operational Programme II – Cohesion Policy 2014-2020
*"Investing in human capital to create more opportunities
and promote the well-being of society"*
Scholarships may be considered for part-financing by the
European Union - European Social Funds (ESF)
Co-financing rate: 80% EU Funds;20% National Funds

Abstract

Systems using Wireless Sensor Networks (WSN) are shaped with the consideration of a great number of factors. These include power consumption, lifetime, network topology, responsiveness and transmission errors. Thus, several research challenges are introduced. We propose a system which lets each network node locally gather knowledge and distribute it amongst its peers. Then, based on this knowledge it evaluates the error and decides whether it should send the updated knowledge or not, minimizing power consumption. Using various techniques we will be querying the learnt shared knowledge and evaluating the accuracy of each to study what the effects of increasing the permissible error are.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: Kurt Portelli Signature: K.Portelli

Acknowledgements

I wish to express my sincere gratitude to Dr. Christos Anagnostopoulos for his guidance and expert advice throughout all the various stages of this project. Apart from that I am also thankful to him for making me appreciate even more the subject of data science and machine learning.

Furthermore, I would like to thank my family for all their continuous support in this first year living abroad.

Contents

1 Introduction 5

1.1 Problem Synopsis 6

1.2 Project Aims 6

1.3 Document Outline 7

2 Preliminaries 8

2.1 On-line Algorithms 8

2.1.1 K-Means 8

2.1.2 Adaptive Resonance Theory (ART) 9

2.1.3 Linear Regression 9

2.1.4 Mean and Variance 10

2.2 Probability Density Function (PDF) 11

2.3 Normalization 11

2.4 K Nearest Neighbours (KNN) 11

3 Literature Review 13

3.1 Introduction 13

3.2 Wireless Networks 13

3.3 Edge Computing 13

3.4 Predictive Analysis 13

4 Contributions **14**

4.1 Model Description 14

4.2 Network Efficiency 15

4.3 Ensemble Learning 16

4.4 Adding the “Reliability” variable 17

5 Design and Implementation **20**

5.1 Dataset 20

5.2 Network Architecture 21

5.3 Query Analytics 22

5.4 Statistics Gathering 24

6 Evaluation and Case Study **25**

6.1 THETA study 25

6.1.1 Probability Density Functions 27

6.2 Query Accuracy Evaluation 29

6.3 Network Efficiency 33

7 Conclusion **34**

7.1 Achievements 34

7.2 Future Work 34

7.3 Final Remarks 34

A First appendix **35**

Chapter 1

Introduction

The Internet of Things (IoT) is a rapidly growing area, with it more data is being stored and made accessible which previously has never even been thought possible. IoT is composed of billions of devices that can gather, share information and sometimes even capable of performing operations on this information. IoT devices are physical objects connected with the internet able to share information. This can include anything ranging from smartphones to Radio Frequency Identification (RFID) tags found in everyday products. As more data is made available for anything imaginable that affects our daily lives, opportunities arise for applications that extract this information and make use of it. Y.Wang et al. [29] state that the smart grid has been recognized as an important form of IoT. It allows a two-way information flow which produces new perspectives in energy management. Making all this information useful is very challenging as it needs to be collected, transferred, stored and made accessible. [4]

Since these IoT devices are generally wireless, compact and have very limited resources they can be considered as WSNs. REFERENCE ? WSNs are made up of a large number of nodes, which are capable of wireless communication and minimal computation capabilities. [2] Each sensor node measures the temporal-spatial field of the area around it. These are called the contextual scalar parameters and can be of any form and value, example; humidity, temperature, acidity, etc.. This contextual information is relayed to a "sink", which can referred to as a central node.

In a naive system all these devices generate massive amounts of data which is periodically sent to central node with more computing resources. This in turn, might restructure the data more effectively to be sent to another node or make it available for querying. All this network transfer drains power and as the network size increases this effect is emphasized even more. This has created a need and opportunity for machine and statistical learning algorithms.[8] According to L.Bottou et al.[8] these technological improvements have outrun the exponential evolution of computing power. Thus, we must rely more on on-line learning algorithms to process these large amounts of data with comparatively less computing power.

The vision of IoT is to allow any device to be continuously connected sharing its knowledge about the surroundings. Machine learning then uses this stream of data to deduce results, infer new knowledge, reason about contextual events or make assumptions. The possibilities for such systems are endless, taking, for instance, the case of thermometers in rooms next to each other and based on previous information the temperature of various rooms can be predicted based on the temperature of adjacent ones.

1.1 Problem Synopsis

One of the factors in IoT systems is the collection of contextual information from certain sources in an IoT environment.[4] IoT devices communicate in an ad-hoc manner with the Central nodes, also known as collectors to send information. The objective of each collector is that it has the latest up-to-date information/context. This would allow the IoT system to further process the context and make use of it, example environmental monitoring.

In this IoT environment the IoT devices are not able to communicate directly together due to the limited resources, so instead they communicate with the collectors. Even if each device has an infinite power supply, a naive system which periodically transmits the context is not feasible especially as the network grows in size. Collectors would become overloaded with messages from all devices and incur a performance degradation as network size increases. Transmitting wireless messages is very costly and in most cases IoT devices run on limited battery power. These devices need to make intelligent decisions whether to send data or not to conserve battery power and increase their lifetime. In this naive system there is no concept of knowledge, thus withholding data would result in the collectors having gaps of knowledge.

Once the collectors have access to the context of the connected devices, they need to make it available for querying. A naive system stores all the data items inside the context for each device. This would result in a very resource hungry collector due to the huge amount of processing and storage capabilities it would need to support. C.Anagnostopoulos [4] states that the contextual sensor data exhibits high complexity, dynamism, accuracy, precision and timeliness. This is due to the huge volumes, interdependency relationships between devices and continuous updates performed in real-time. From this statement he then argues that “an IoT system should not concern itself with the individual pieces of contextual sensor data: rather, context should be intelligently collected and interpreted into a higher, domain relevant concept.”[4]

Using this concept, instead of having access to all the raw data, collectors would contain multiple intelligent contexts. This raises the challenge on whether for each query there exists an optimal set of intelligent contexts that increase the accuracy. In other words it opens the opportunity for ensemble learning [11]. This means that we cannot guarantee to find the best hypothesis (intelligent context) that gives the best accuracy from all contexts. Ensemble learning tries to intelligently choose contexts and assign weight depending on their likelihood of providing the best result. Hence, it aims at reducing the bias and variance of the learning algorithms used to construct the context.[11]

The research challenge we will be looking at is the building of an energy efficient network which minimises power consumption while simultaneously minimising transmission error. Furthermore, we will investigate the use of ensemble learning on the local contexts to achieve more accurate queries. This system must make use of very little computing resources to maximise its lifetimes while at the same time support scalability and real-time querying for environment monitoring.

1.2 Project Aims

We argue that instead of directly sending the contextual information, devices should locally learn their context using machine learning algorithms and then transmit this knowledge. If we transmit the local model to the central node each time it is updated then we would not save any power consumption. Thus, we go a step further and set up the notion of a permissible error, which allows the local and central models to have a limited amount of discrepancy. This allows us to avoid unnecessary transmissions which do not "teach" the collector any valuable knowledge, and in turn reduce the power consumption. Altering the permissible error should determine the percentage of messages being sent, thus directly affecting power consumption.

We will be using on-line linear regression to learn the context of each device with real-time constraints. Primarily we will be investigating how the error discrepancy between devices and collectors affect battery consumption and network lifetime. Secondly we will study how altering this error discrepancy has an effect on the querying accuracy of the collectors.

The primary objectives for this project may be defined as follows:

- Obtain a clear understanding on how IoT systems operate and what they try to achieve.
- Implement On-line machine learning and clustering algorithms. In particular, we will be focusing on a 3-dimensional implementation.
- Build a system capable of simulating a real-life IoT system containing collectors and devices. We will be using the U-Air [31] dataset, which contains air quality data from real-life stations in Beijing and Shanghai.
- Integrate the capability of transmitting both raw context and knowledge (with a flexible degree of error) to investigate energy efficiency of our implementation.
- Evaluate the query accuracy performance of both naive system and our implementation.

1.3 Document Outline

The content of this report is organized into the following chapters:

- **Chapter 2** provides the reader with the background knowledge required for understanding the basic principles and algorithms behind this work. In particular, we introduce the general theory behind on-line algorithms, linear regression and normalization.
- Afterwards, in **Chapter 3** we analyse the related work in this field to provide us with the required knowledge to improve upon what there already is.
- **Chapter 4** describes our major contributions to the research challenges, namely network efficiency and ensemble learning.
- In **Chapter 5** we describe the major design and implementation decisions of our contributions and simulation system.
- **Chapter 6** contains the evaluation and performance analysis of our work.
- Lastly in **Chapter 7** we identify possible future extensions, and conclude the presented work.

Chapter 2

Preliminaries

In this chapter, we present an overview of the concepts and base principles we mention and make use of in our work. We develop a system which deals with a continuous data stream and thus, has to use specialized algorithms which treat each data item individually without storing past data items. Various algorithms are used which extract knowledge from each individual sensor by minimizing loss and then quantize the input.

2.1 On-line Algorithms

According to M.Karp [17] an on-line algorithms is one which receives a sequence of requests and performs an immediate action to each request. These are useful in situations where decisions must be made without any future knowledge. Their effectiveness is measured by the comparing them with their off-line (batch) counterparts. In this dissertation we make use of well known common techniques already proven to work.

2.1.1 K-Means

The K-means algorithm attempts to find K representatives (centroids) of a data set in such a way as to best represent the data. [6] Although it is algorithmically simple, it still manages to be relatively robust and give close to optimal results on a wide range of data sets. Its disadvantage is that one has to predefine K. Furthermore, the initial K centroids chosen have a drastic effect upon the clusters chosen. The first k inputs initialize the codebook vectors (centroids) $m_j, j = 1, ..., k$. The rest of the inputs $x^t \in X$ are used for the update function.

$$i \longleftarrow argmin_j \parallel x^t - m_j \parallel \tag{2.1}$$

$$m_i \longleftarrow m_i + \eta(x^t - m_i) \tag{2.2}$$

Equation 2.1 finds the closest centroid to the input. Then the closest centroid is updated as shown in equation 2.2 depending on the learning rate defined by η. A large η enhances accuracy but makes m_i oscillate. Therefore to reach convergence one needs to use a small or decaying η.

2.1.2 Adaptive Resonance Theory (ART)

ART[9] is another unsupervised learning algorithm similar to k-means. It does not require the number of clusters to be predefined before hand. Instead one has to define *vigilance* which represents the degree of similarity between points. An data item $x^t \in X$ is a member of a cluster m_j only if the distance between them is less than vigilance. Similar to the K=means procedure, once m_j is chosen, equation eq:updateCluster is used to update the centroid. If no centroid satisfies this constraint, then a new cluster is created. This allows a better flexibility but has the disadvantage of creating an infeasible and unpredictable amount of clusters in certain situations.

2.1.3 Linear Regression

Once it is proven that there is a statistically significant correlation between two variables, linear regression allows us to make predictions about one variable based only on the other variable. The relationship between these two values must be linear, else this technique would not be able to model the data accurately.

To perform this supervised learning we must first start by defining the hypothesis example the linear function 2.3.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{2.3}$$

The θ_i's represent the weights of the linear function, thus, determining the mapping between X and Y. Let $x_0 = 1$, we then rewrite equation 2.3 to a more compact form represented by equation 2.4.

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x \tag{2.4}$$

For this hypothesis to be useful, the θ parameters need to be trained to fit a particular dataset. Conceptually the closest $h(x)$ is to Y, the better we are at representing the data. Thus, we define the cost function equation 2.5 which measures the total discrepancy for each input $x^{(i)}$ with output $y^{(i)}$.

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{2.5}$$

Naturally, we would like to choose a theta which minimizes the error. One such technique which does this is the gradient descent rule. It calculates the gradient of the error and takes a step proportional to the negative gradient. To achieve this we need to use the partial derivative of equation 2.5. For all θ the update rule then becomes equation 2.6. This update rule is called least mean squares (LMS) and it has a very interesting property. The magnitude of the update is proportional the the error. This means that the greater the error, the steeper the descent towards the minima. After completing the derivative we are left with the complete equation 2.7. This has to be repeated for each θ until convergence. Meaning until we reach the best possible theta values.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{2.6}$$

$$\text{Repeat until convergence: (For each j) } \theta_j := \theta_j - \alpha \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \tag{2.7}$$

One might argue that with this iterative approach we might get stuck at a local minima instead of finding the optimal minima. In this particular scenario J is a convex quadratic function, meaning it has only one minima. Thus, gradient descent will always converge at the global minima assuming α is not too large.

Multivariate Stochastic Gradient Descent

The previous algorithm is called batch gradient descent as for each update step it uses all all the training set. This would not be feasible to compute especially as the dataset size increases. Another issue would be that previous data items have to be stored, and this is not always possible. In fact we will be using the more lightweight and on-line version called stochastic gradient descent. In batch gradient descent we go through all the data set for a single update step while when using equation 2.8 we start making progress right away. We use this update rule for each new training item we encounter. This in fact has the tendency of getting close to θ much faster than batch gradient descent, however it may never converge and instead keep oscillating around the minimum. The reason it is called multivariate is that we will be using multiple inputs (more than one x) to get y.

$$\text{(For each j) } \theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \tag{2.8}$$

2.1.4 Mean and Variance

Generally the naive way of calculating the mean and variance require knowledge of the whole data set. This is not possible in systems where data cannot be stored or there are other resource limitations. Welford [30] developed algorithm 1 which calculates this in a single pass by inspecting one value at a time only once. This algorithm is implemented used in various integral parts of our system.

```
Function online_variance(data)
    $n = 0$;
    $mean = 0.0$;
    $M2 = 0.0$;
    forall $x$ in data do
        $n+ = 1$;
        $delta = x - mean$;
        $mean+ = delta/n$;
        $M2+ = delta * (x - mean)$;
        if $n < 2$ then
            return float('nan');
        else
            return $M2/(n - 1)$;
        end
    end
```

Algorithm 1: Online Mean and Variance [30]

2.2 Probability Density Function (PDF)

The PDF [18] of a continuous random variable is a function that describes the likelihood for the variable to be assigned a certain value. The reason it is called a continuous random variable is because it takes on an uncountably infinite number of possible outcomes. The PDF of the normal distribution shown in equation 2.9 is built using the mean and variance denoted by μ and σ respectively.

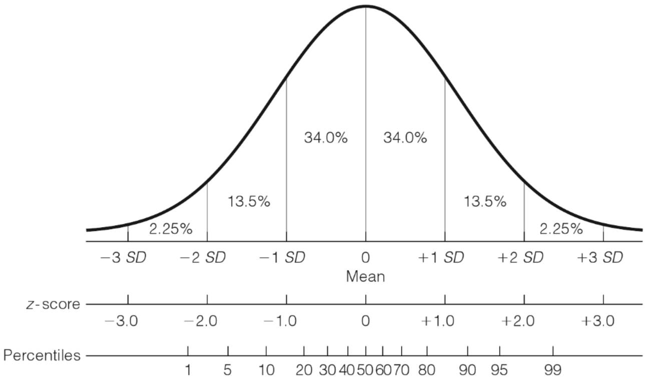
$$f(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{2.9}$$

2.3 Normalization

We use the standard score [12] to normalize the dataset. This is done by subtracting each item x by the mean and then dividing it by the standard deviation of the dataset as shown in equation 2.10. Performing this normalization allows us to better understand the results as it gives a meaning to a raw value. Figure 2.1 demonstrates the distribution of z, a negative value means that x was less than the mean. Furthermore, the magnitude of this value represents the distance from the mean in terms of standard deviation.

$$z = \frac{x-\mu}{\sigma} \tag{2.10}$$

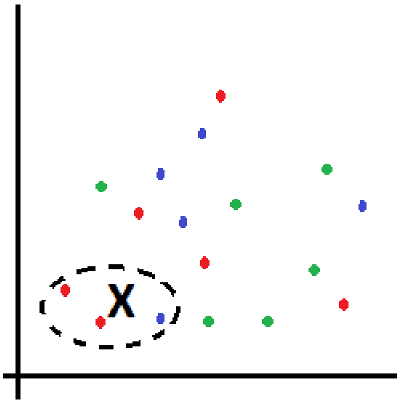
Figure 2.1: Standard score normalization [12]



2.4 K Nearest Neighbours (KNN)

KNN [1] is a straightforward classification technique, in which given an input it is able to classify it to certain class. As the name implies it takes the closest K neighbours and depending on the majority of their classes it determines the class of the input. Figure 2.2 demonstrates KNN ($K = 3$) while classifying X using a data set containing 3 different classes. In this case input X is classified as Red due to the fact that the majority of its 3 closest neighbours are Red.

Figure 2.2: At KNN=3, Input X is classified as Red



Chapter 3

Literature Review

3.1 Introduction

In this chapter we describe the studies and research works that have been carried out in the field of energy efficient sensor networks. Additionally we shall delve into other areas such as ensemble learning and edge computing. We then evaluate the other works whilst comparing them with our approach.

3.2 Wireless Networks

[22] [15] [3] [19] [29] [28] [24] [27] [16] [20] [10] [5]

3.3 Edge Computing

[23] [7] [25]

3.4 Predictive Analysis

[21] [13]

Chapter 4

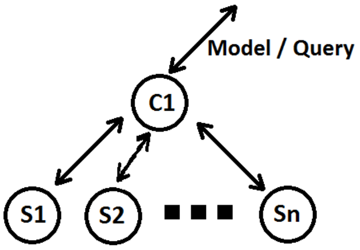
Contributions

In this chapter we discuss our major contributions with the aim of improving WSNs. More specifically how we can increase network efficiency, which extends the devices' life time. We shall delineate a strategy for creating knowledge from the raw context, by locally learning the context in real-time using on-line stochastic gradient descent. Based on this knowledge, we then decide whether it is worth it to transmit data or not. Additionally we look into how we can minimize the error when querying the knowledge found inside the collectors.

4.1 Model Description

We start by considering an IOT system composed of sensor nodes and a collector depicted in figure 4.1. Sensor nodes are only capable of communicating with collectors. Collectors can then be queried or in turn be considered as sensor nodes and connected to larger tree structure, creating a larger more complex network. Resources are reserved for communication, processing and data sensing. Message transmission consumes much more energy when compared to the processing and data sensing tasks. [22] Thus, we argue that if we let the nodes continuously sense data and only transmit this information when it is essential, we could create a more energy efficient network. A.Manjeshwar et al. [22] states that this is possible because sensor networks are "data centric", meaning that data is requested based on certain attributes and not requested for a certain node. The requirements of each WSN change according to the application as some nodes might be static while others mobile. When nodes are adjacent to each other certain similarity might occur, thus, data can be aggregated. A common query in these systems is for example which area has humidity $< 10\%$ rather than what is the humidity of area X. We can use this knowledge to our advantage and define a system that can minimize the number of messages transmitted while at the same time still provide accurate results from queries.

Figure 4.1: Model Example



4.2 Network Efficiency

Consider a sensor node i which receives a new data item $p(t)$ at time t. Our system needs to decide whether it is worth transmitting this new data or not. To achieve this, we create a local model based on linear regression. The function we compute will enable us to predict the error and have a quantifiable way of deciding whether transmitting $p(t)$ is worth it or not. Since each sensor node has limited resources, both computational and storage limitations we use on-line stochastic gradient descent to calculate the regression line. A full detailed description of on-line stochastic gradient descent can be found in the Preliminaries chapter, section 2.1.3. Using this algorithm we avoid storing a history of values and instead we consider each new data item alone, then discard it.

Each sensor i locally and incrementally builds its own model from sensing the pairs (x, y). After a period of learning, each sensor sends its model parameters to a centralized concentrator. Take for instance weights $w = [w_0, w_1, w_2]$ from equation 4.1.

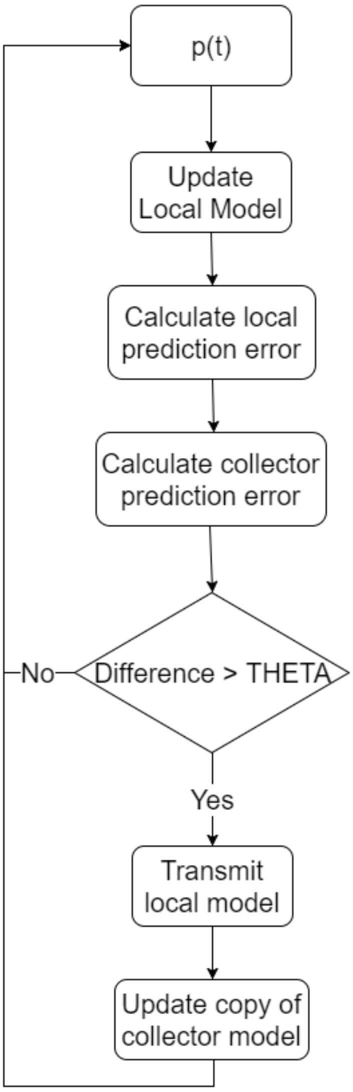
$$f[i](x) = w_0 + w_1x_1 + w_2x_2 \tag{4.1}$$

Once a sensor i has sent its local model $F[i](x)$ to the concentrator, then for every new (x, y) data pair it receives, it is capable of updating the local model and deciding of whether to update the concentrator model as well. For every new (x, y) the w parameters might change when compared to the original model sent previously to the concentrator. Sensor i upon receiving an (x, y) pair it updates the model and predicts the output $y*$ from its current local $f[i](x)$ with parameters w' and then measures locally the error $e* = |y - y*|$. If the model has not significantly changed the 'obsolete' model in the concentrator will experience a similar error. Sensor i keeps a copy of the original w (obsolete model in concentrator), thus, knows exactly the error the concentrator would experience. Hence, let us denote this error from the original local model as $e\# = |y - y\#|$ where $y\#$ is the prediction of y when involving the original (obsolete) model $f[i](x)$ with parameter w. At that time, sensor i has two prediction errors: $e*$ and $e\#$. The $e\#$ is the error that concentrator would experience if the same input x was used.

$$De = |e * -e\#| \tag{4.2}$$

Using Equation 4.2 we are able to get the discrepancy in error between the 2 models. We then use the condition $De > THETA$ to determine whether it is worth it to update the obsolete model or not. A $THETA = 0$ would mean that the local and concentrator model would be synchronized at all times. On the other hand when we increase $THETA$ we transmit less messages, thus, increase network efficiency. This would also mean we are now susceptible to an increase in error between the 2 models. Figure 4.2 shows a simplified flowchart of the algorithm just described in this section.

Figure 4.2: Sensor Node Flowchart


p(t)
Update Local Model
Calculate local prediction error
Calculate collector prediction error
Difference > THETA
No
Yes
Transmit local model
Update copy of collector model

4.3 Ensemble Learning

The previous section explains our algorithm which increases network efficiency by transmitting less messages to the concentrator. This is done by altering $THETA$ which determines the acceptable error. An increase in network efficiency means that sensors working on battery power can last longer without the need of recharging. As previously mentioned the purpose of an IOT system is that it can be queried from the "root" (concentrator). In this section we aim to extend the model defined in the previous section to reduce the error at the concentrator level.

Naively the concentrator would have all the (x, y) data pairs and when queried with an arbitrary x, it can directly search in its storage for the best match and return y accordingly. In our model the concentrator collects a function $f[i](x)$ for each sensor i which simulates the data collected. The ideal scenario is that when the concentrator receives query x, it would also know which function f it should use to predict y. Unfortunately, in a real-life scenario this is not possible as we would not know which sensor x is related to.

One solution is that the concentrator proceeds with averaging the prediction results from all the collected functions $f[i](x)$. The concept behind averaging all the predictions ensures that the over-estimates and under-estimates are balanced out. We argue that this might not always produce an optimal result and it can further be improved. There exists scenarios where an input x is not observed by some of the functions, thus, leading to poor predictions. Take for instance the case where multiple thermometers are sparsely placed around a building. Sensors next to a kitchen might in general observe temperature readings in a higher range than sensors placed outside. Given a high temperature x, would it be better to average the predictions from all the sensors or use only the kitchen sensors?

With this concept in mind, we extend the previous model and add data representatives of the input space. Using these data representatives, the system would be able to determine whether a function f is familiar to the query x or not. To elect data representatives we use an on-line clustering algorithm which quantizes the input space. We used both K-Means and ART which are described in detail in the Preliminaries chapter; section 2.4 and 2.1.2. When a data pair (x, y) is used to update the model, we also update the centroids in the clustering algorithm. The centroids are transmitted to the concentrator along with local model.

The concentrator now has the necessary information to determine what the input space of each function is. When query x is submitted, the concentrator can select the functions which have data representatives in the vicinity. The assumption is that these functions were generated from values similar to the query, thus, have more reliable knowledge.

4.4 Adding the “Reliability” variable

Right now the concentrator is able to select which functions to use based on the distance between query x and centroids. We believe the distance alone might not be sufficient enough to describe the input space of each function. Our intuition is that some sensors produce more accurate results than others. To study this, we performed an analysis on the Beijing Air Quality dataset [31] and trained a regression model for each sensor. We then isolated 2 sensors at a time and predicted the y using 3 different methods;

- Endogenous error - In this test we predict y of sensor i using regression function $f(i)$.
- Exogenous error - In this test we predict y of sensor i using regression function $f(i + 1)$.
- Fusion error - In this test we predict y of sensor i using both regression functions $f(i)$ and $f(i + 1)$ and then take the mean.

Figure 4.3: Endogenous, Exogenous, Fusion errors

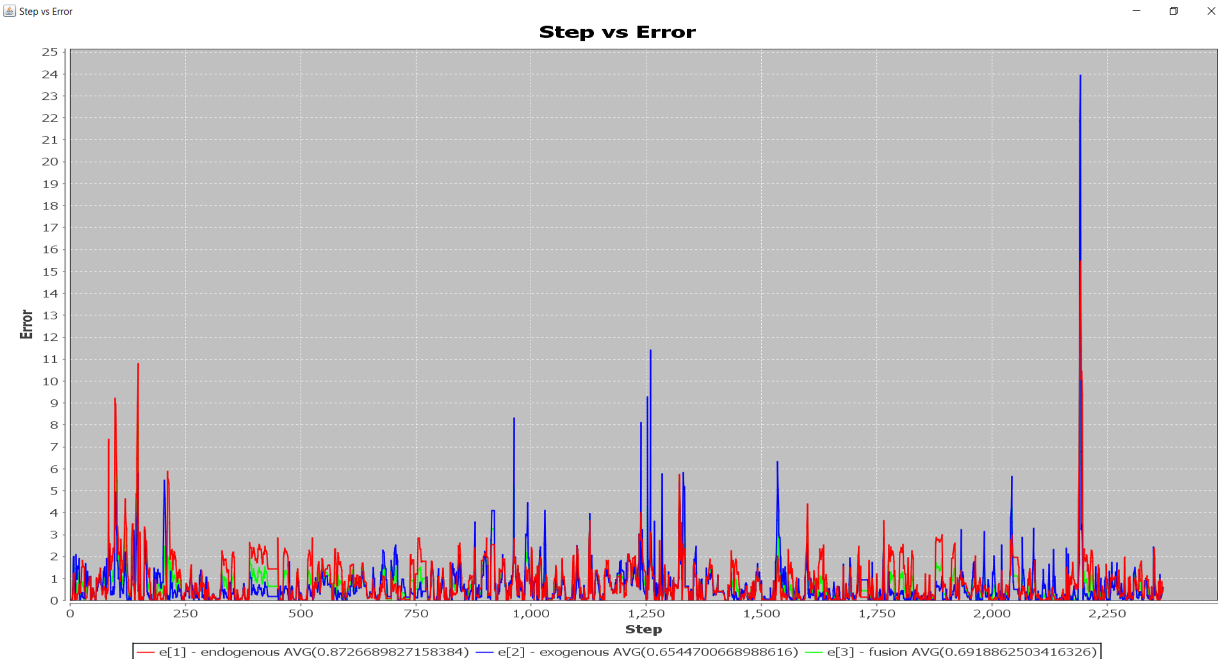
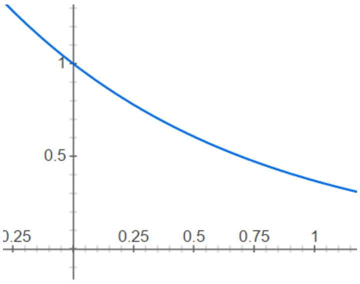


Figure 4.3 shows one of the studies done using the first 2 sensors. In this case using just the regression model from another sensor (exogenous) proved to more accurate rather than using its own model. The reason might be because some centroids are more popular, while others yield less error in certain situations. This is were ensemble learning is useful, and we can leverage the knowledge the concentrator has about every sensor attached to it. The concentrator should try to choose a subset from all the sensors to provide more accurate results. Centroids which are frequently updated tend to have a higher average error due to the frequent updates to the regression model. Hence, we should consider the number of times each centroid is used to determine the popularity. Thus, along with each centroid we include an error value and number of times used.

$$weight = \frac{e^{-distance} + e^{-error} + used}{3} \tag{4.3}$$

Figure 4.4: e^{-x} Function



Equation 4.3 defines how we assign a weight to each centroid in relation to a given query. Once we sort the centroids in descending order according to their weight, we would get an amalgamation of

the 3 properties; closest, lowest error and most popular. Note, that the values of *used* are normalized between 0 and 1 so as to makes use of the exponent function shown in figure 4.4. As distance and error approach 1 (highest possible due to normalization), the exponent function inverts them and gives them a low value. On the other hand as *used* approaches 1 we would like to give it a higher weight, thus we leave it as is.

Chapter 5

Design and Implementation

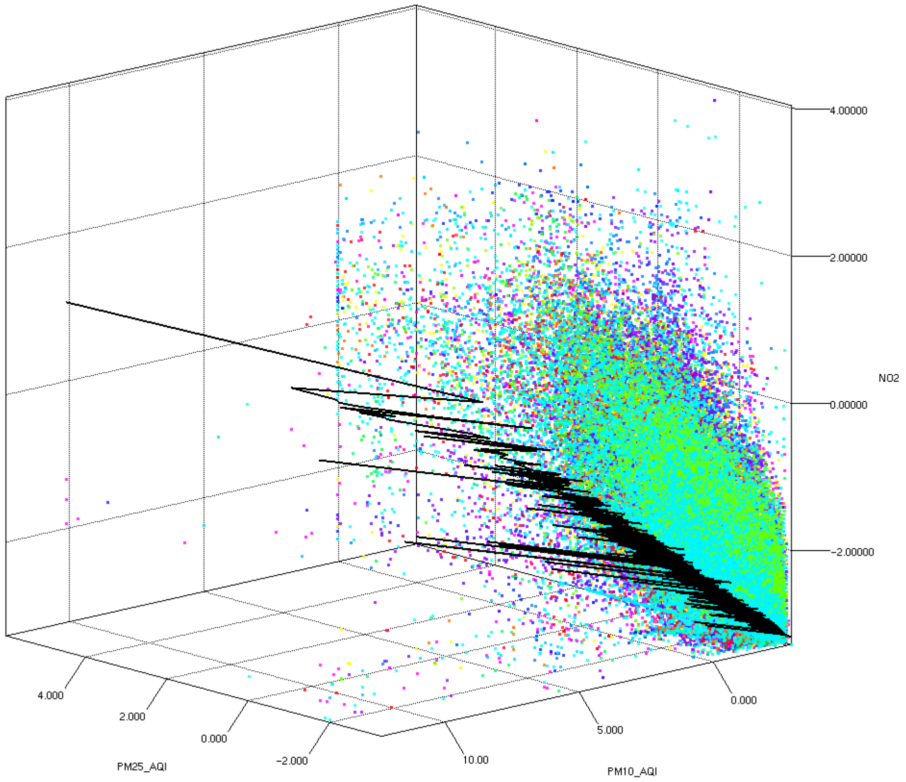
In the previous chapter we laid the theoretical groundwork and intuitions of our system, and now we proceed to examine how they were implemented into fully working algorithms. We begin by explaining the dataset that we shall be using and how it is structured as it will affect the design of the system. Afterwards, we explain how we implemented a system capable of simulating an IoT environment. This includes the network aspect and statistics gathering. We then explain in detail our contributions regarding network efficiency and ensemble learning mentioned in the previous chapter.

5.1 Dataset

We will be using the air quality data[31] collected from air quality monitoring stations in Beijing. There are 36 independent sensors which can be considered as our IoT devices. All of these devices will need to transmit the knowledge to the concentrator. We chose 3 fields from the dataset and verified there was a correlation among them. Let x_1 be `PM25_AQI`, and x_2 be `PM10_AQI`. These values represent the concentration levels of fine particulate matter (air pollutant) with aerodynamic diameter of less than 2.5, 1.0 respectively. We will consider y as the `NO2_AQI` which represents the concentration levels of Nitrogen Dioxide. These gases are emitted by all combustion processes and have a negative impact on human life, example Nitrogen dioxide is linked with the summer smog.[26]

We normalize the data using the standard score as explained in the Preliminaries chapter. The normalized x_0, x_1, y will be used during the implementation and evaluation phases. Figure 5.1 displays the whole normalized dataset with a general linear regression model amalgamating all the sensors together.

Figure 5.1: 3D graph plot of dataset along with linear model



5.2 Network Architecture

We first started by creating a list of requirements of the system and then designed the system's components around it. We needed to simulate multiple sensors reading information at unpredictable times. This information had to be made accessible through a concentrator for querying. Additionally we needed to gather statistics so as to be able to analyze the performance and compare different simulations. We decided to us Java to implement the simulator as it felt very intuitive to read the data into multiple threads, add our contributions and analyze them.

A `SensorManager` class was created that handles the reading of new information from the dataset for each sensor. A `Sensor` thread is then initiated for each sensor. Each sensor reads data from the `SensorManager` and acts on it as described in algorithm 2. Algorithm 2 is the implementation of all the contributions described in the previous chapter.

```
$w' = (w'_0, w'_1, w'_2)$ (local weights);
$w = (w_0, w_1, w_2)$ (concentrator weights);
$f' = w'$ (local model);
$f = w$ (concentrator model);
$centroids$ = create centroids list;
while data is available do
    $x_1, x_2, y = p(t)$ (read from sensor);
    update $w'$ of $f'$ model using $x_1, x_2, y$;
    $c$ = closest centroid to $x_1, x_2$ from $centroids$;
    if learning phase finished then
        $Y' = f'(x)$;
        $Y = f(x)$;
        $e' = (Y' - y)^2$ (local error);
        $e = (Y - y)^2$ (concentrator error);
        associate $e'$ to $centroids[c]$;
        if $e > e'$ then
            $discrepancy = e - e'$;
            if $discrepancy > THETA$ then
                Transmit $w'$,$centroids$ (and error associated with each centroid);
                $f$ = $f'$;
            end
        end
    end
end
```

Algorithm 2: Transmission algorithm

Each sensor stores the local model $f'(x)$ and a copy of the concentrator model $f(x)$. When a new data item is read, $f'(x)$ is updated and the closest cluster determined. After the local and concentrator errors are calculated, an error is associated with the closest cluster so as to have the notion of a reliability factor as explained in the contributions chapter. Lastly the algorithm checks the discrepancy between local and concentrator errors, and if it is larger than $THETA$ is transmits the local knowledge to the concentrator.

5.3 Query Analytics

The `Concentrator` class is tasked with continuously receiving local models in no particular order and storing them. Furthermore, the concentrator needs to make its knowledge available for querying. Thus, we expose several methods which allow querying using different algorithms. This allows us to take measurements and then evaluate them to determine which gives the minimum error. Algorithm 3 is the naive approach which predicts y' using all the models and then calculates the mean.

```
query = $x_1, x_2$;
$f$ = List of sensor models;
$y' = 0$;
foreach f[i] do
    $y' += f[i](x_1, x_2)$;
end
calculate mean $y'$
```

Algorithm 3: Averaging models

Algorithm 4 uses the centroids to calculate the distance between the query and each centroid. Then the nodes are sorted by distance in ascending order and the first K models are used to predict y'. The concept is that the we only use the functions which "are built from that quantized space". This should skip the outliers and return a more accurate result when compared to the naive approach (algorithm 3).

```
query = $x_1, x_2$;
$f$ = List of sensor models;
$centroids$ = list of centroids;
$distances$ = list of distances;
foreach $centroids[i]$ do
    $distances[i] = distance(query, centroids[i])$;
end
sort $distances$ ascending;
$y' = 0$;
for int i=0; i < K; i++ do
    $y' += f[distances[i].nodeId](x_1, x_2)$;
end
calculate mean $y'$;
```

Algorithm 4: Closest K nodes

Algorithm 5 is an extension of algorithm 4. Rather than considering only the distance from the query we try to materialize a "reliability" factor. With each centroid we store an error value and the number of times the centroid was used. This information is then used to calculate the weight. The equation is explained in detail in the contributions chapter; section 4.4. A low `error`, low `distance`, and a large `used` value would result in large weight. Thus, we sort the weight in descending order and choose the first K models. We also normalize the first K weights so as to adjust the predict value accordingly. The first K models should be the most reliable and give us a more accurate result when compared to algorithm 4.

```
query = $x_1, x_2$;
$f$ = List of sensor models;
$centroids$ = list of centroids;
$errors$ = list of errors;
$used$ = list of times used;
$weights$ = list of weights;
foreach $centroids[i]$ do
    $d = distance(query, centroids[i])$;
    $e = errors[i]$;
    $u = used[i]$;
    $weights[i] = \frac{e^{-d}+e^{-e}+u}{3}$;
end
sort $weights$ descending;
$normalizedWeights$ = normalize first K $weights$;
$y' = 0$;
for int i=0; i < K; i++ do
    $y' + = f[weights[i].nodeId](x_1, x_2) * normalizedWeights[i]$;
end
calculate mean $y'$;
```

Algorithm 5: Most reliable K nodes

5.4 Statistics Gathering

Lastly, we needed a way to gather performance statistics for each run configuration. Each class mentioned previously keeps track of various statistics such as messages sent, errors, etc.. The `QueryPerformer` class was created with the simple task of gathering all the performance statistics in one place and saving it into a readable text file for visualization purposes. This class is initialized once all sensors have read the last piece of data and finished running their logic.

Chapter 6

Evaluation and Case Study

In the introduction of this report we pointed out the project objectives which were;

- The implementation of a system capable of simulating an IoT environment and maximise network efficiency.
- Intelligent sensors capable of extracting context and decide whether it is worth to transmit this knowledge or delay.
- Evaluate the query accuracy of our system by comparing it to a naive system which sends data continuously without regarding battery consumption.

Up to now we presented our algorithms and described the implementation of our system. In this chapter, we evaluate the performance of our system using the Beijing Air Quality data[31] and comment on the results. This includes the evaluation of both the network efficiency and the query accuracy contribution.

6.1 THETA study

Before measuring the network efficiency we study the effects of altering $THETA$. This is the variable which determines the allowed discrepancy between local and concentrator models. As we increase $THETA$ the number of messages sent drastically decreases but the error between local and concentrator models should increase. This can in fact be seen in the Figures 6.1 and 6.2.

Figure 6.1: $THETA$ vs Messages Sent %

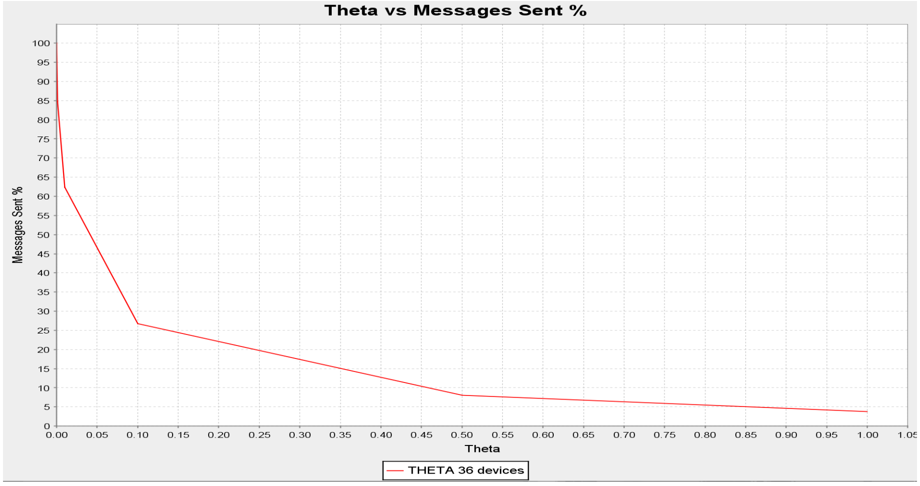
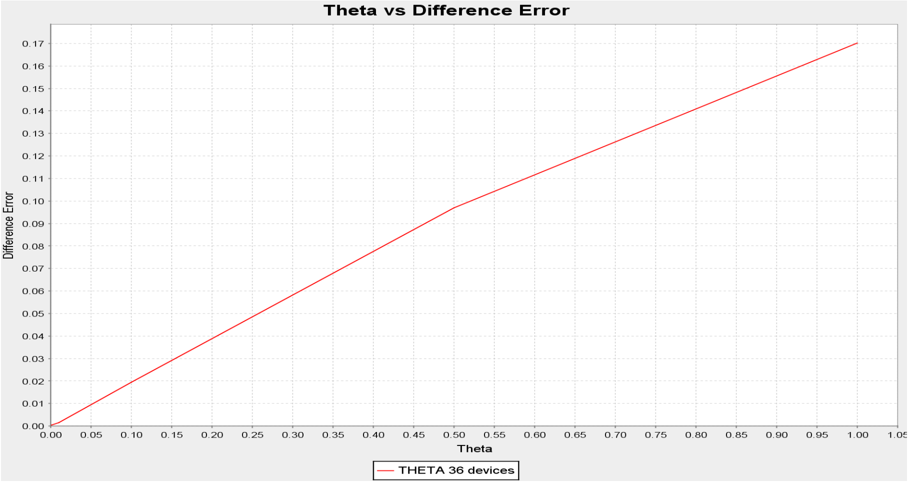
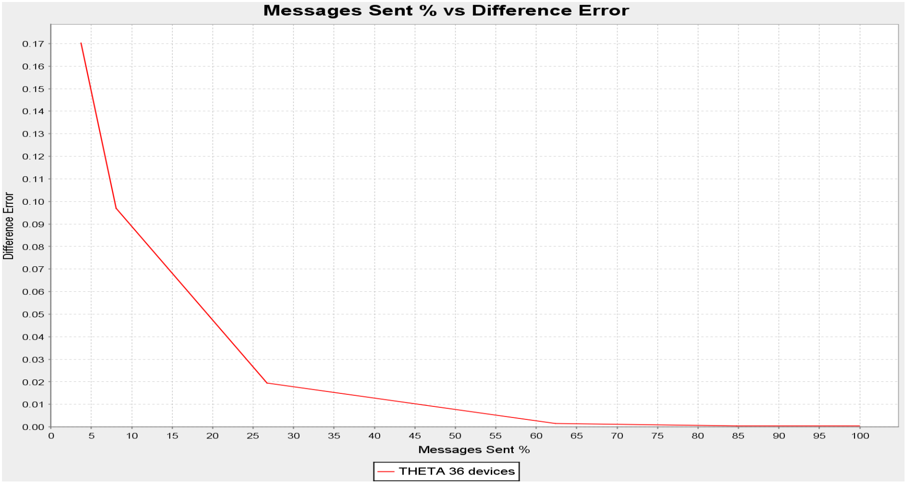


Figure 6.2: $THETA$ vs Difference in Error



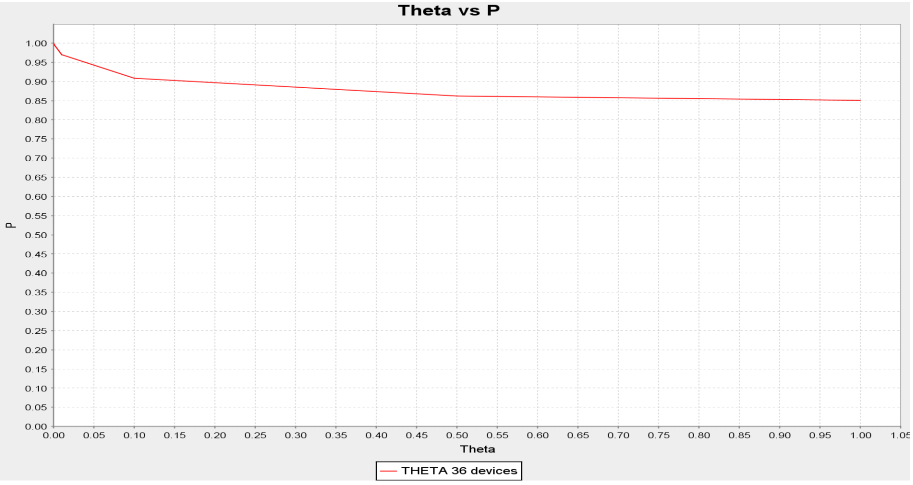
In figure 6.3 we eliminate $THETA$ and directly study the relationship between `Messages Sent` and `Difference in Error`. One notices that there is barely any difference in error when sending 60% messages instead of sending all messages (100%). This means that around 40% of the messages have no useful information to the concentrator. The concentrator is able to predict y with the same error rate as the local model. We are capable of achieving the usefulness of messages due to the introduction of a learning model. In this chapter; section 6.3 we use a real case study in which we exploit our algorithm to evaluate the effect of transmitting less messages on the devices' battery life.

Figure 6.3: Messages Sent % vs Difference in Error



Furthermore, we stored the number of times the local model was better than the central model for each $THETA$. This is shown in figure 6.4. P is the probability that the local model is better than the central model. It is interesting to note that as $THETA$ increases the probability that the local model is better decreases. We believe that this is because as $THETA$ increases the central model is updated less frequently, thus, encapsulating a certain level of generality. This is interesting as it might open up future work regarding whether it would be worthwhile to predict results from a history of models.

Figure 6.4: $THETA$ vs P



6.1.1 Probability Density Functions

Additionally, we measured the mean and variance of the error using an on-line algorithm described in the Preliminaries chapter; section 2.1.4. The mean and variance allow us to the plot the probability density function which is also described in the Preliminaries chapter. This visually shows the likelihood of a certain random variable taking a given value. In our case the value with the highest likelihood is 0 since we used the standard score normalization.

Figure 6.5: Local Model PDF (Error e')

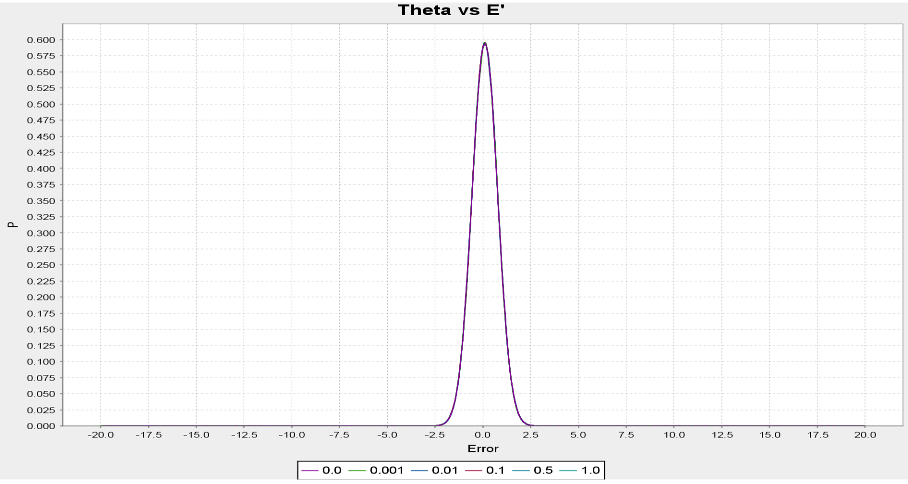
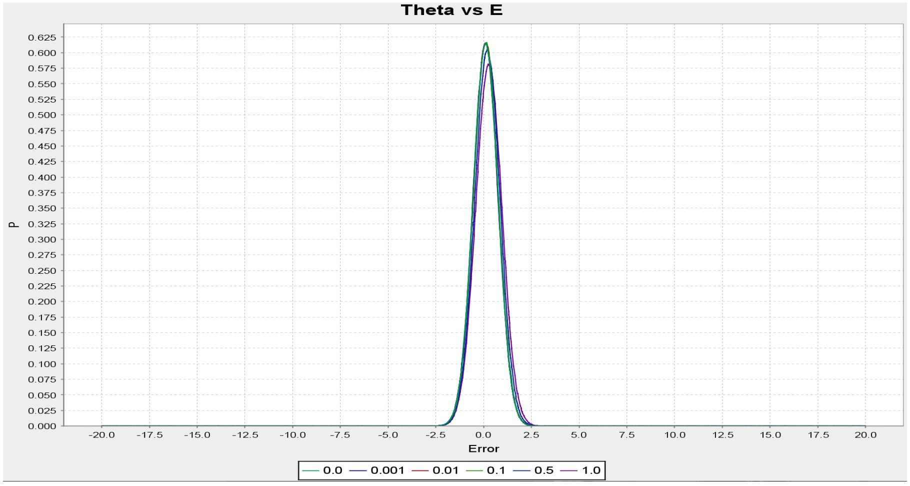
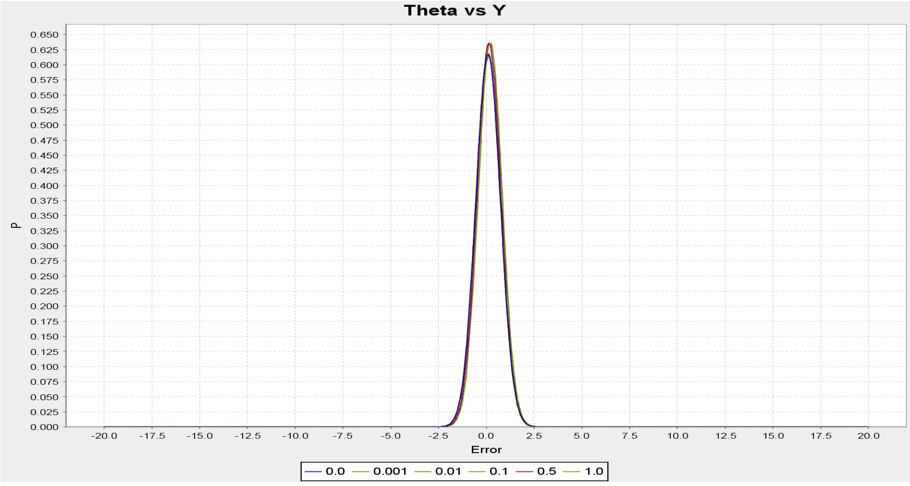


Figure 6.6: Concentrator Model PDF (Error e)



Figures 6.5 and 6.6 show the local and concentrator probability density functions respectively as we increase $THETA$. One notices that the mean of the concentrator shifts to the right (increases from 0) as we increase $THETA$ meaning that the error increases. Figure 6.7 shows the PDF of the difference between Concentrator and Local model.

Figure 6.7: $Concentrator - Local$ Model PDF



6.2 Query Accuracy Evaluation

In the previous section we demonstrated how our system gains network efficiency at the expense of data accuracy. This data accuracy value does not depict the full story. This is due to the fact that IoT systems are queried at a concentrator level rather than at device level. When data from several models (nodes) is aggregated together the error might be reduced as all these models contain distributed knowledge about the environment. Thus, from all the data set we randomly select queries and use them to query the concentrator after it finishes learning. This test simulates the typical usage of an IoT system. It lets us study the effect of transmitting less messages on the query accuracy. Furthermore, in the implementation chapter we included 2 variables which will directly affect the query accuracy. These are; the K number of clusters (centroids) per device, and the number of models chosen at the concentrator for prediction (Defined as KNN in section 5.3).

As a result of this, we will be running tests changing these 2 variable and $Theta$. For each test the following algorithms (described in the implementation chapter) are used to predict the result of the query. Algorithms:

- Base Line - (most expensive algorithm) All data is transmitted to the concentrator and a single model is built.
- Ideal - Since this is a test scenario we know from which sensor the query is taken. Only the model for that sensor is used to predict the result.
- Average - All the models stored in the concentrator are used to calculate the mean.
- Distance - The closest K models are used to calculate the result. (section 4.3)
- Reliability factor - The first K models with the highest weight are used to calculate the result. (section 4.4)

Figures 6.8, 6.9, 6.10, 6.11, 6.12, 6.13 demonstrate our results from these tests. They are 3D graphs, with the x-axis being the KNN, z-axis the K (clusters) and y-axis the error. The different colors show the algorithm used for that particular result.

Figure 6.8: $THETA = 0.0$ (99.98% messages)

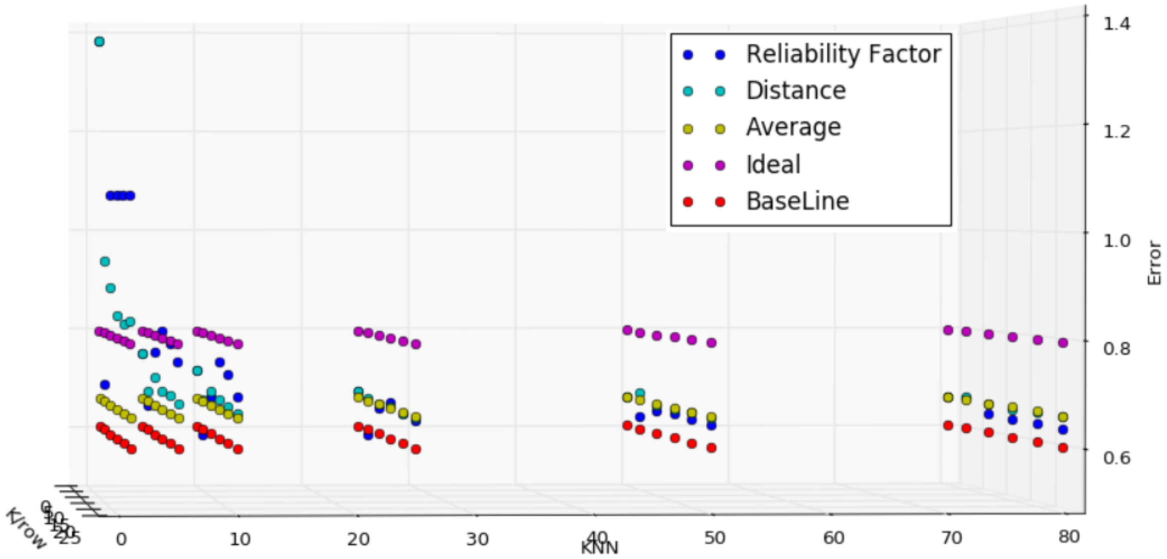


Figure 6.9: $THETA = 0.001$ (84.7% messages)

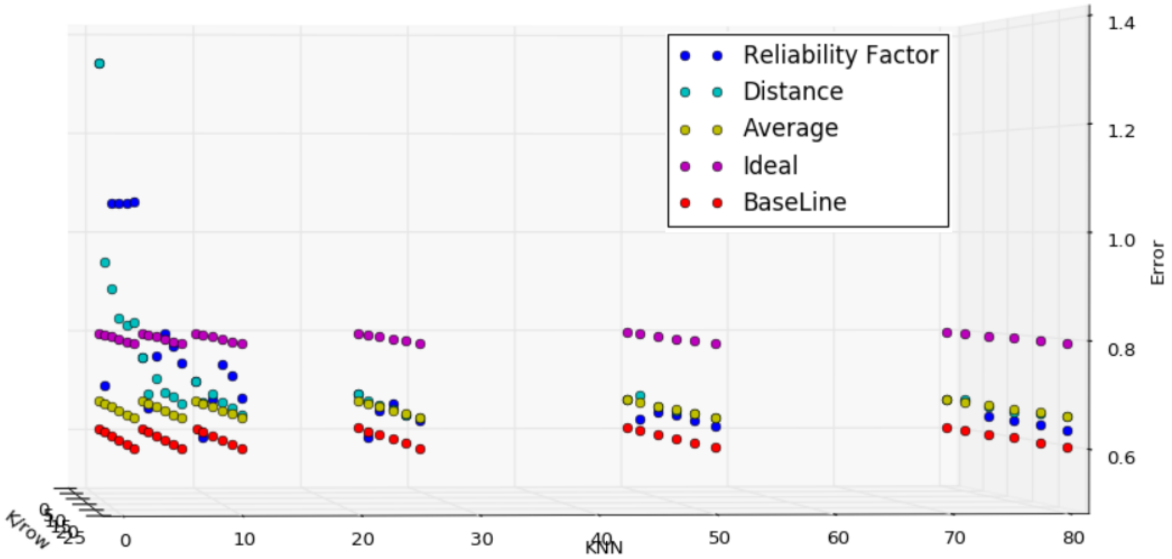


Figure 6.10: $THETA = 0.01$ (62.3% messages)

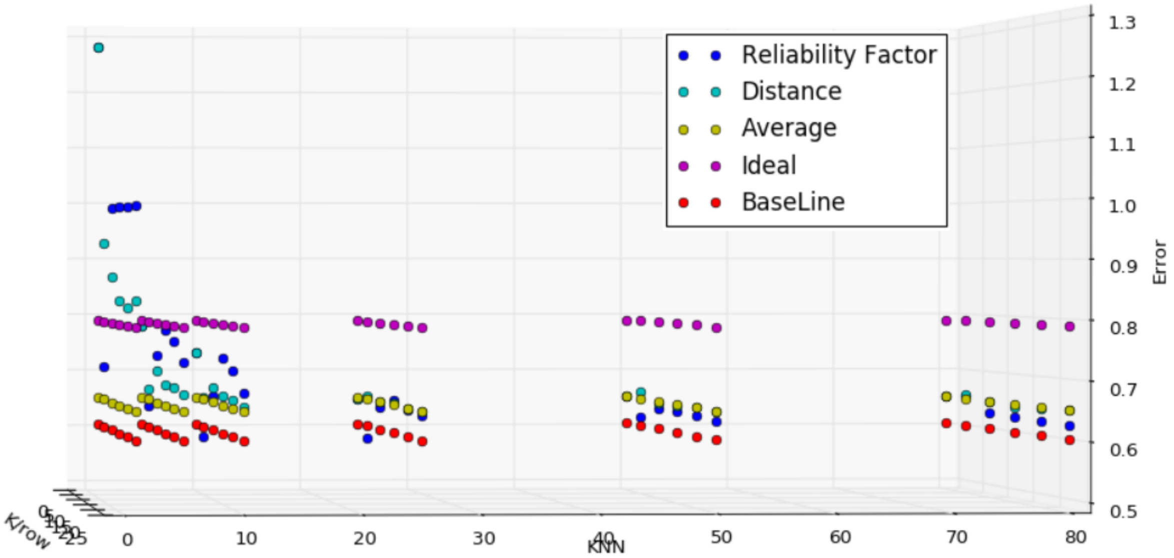


Figure 6.11: $THETA = 0.1$ (26.7% messages)

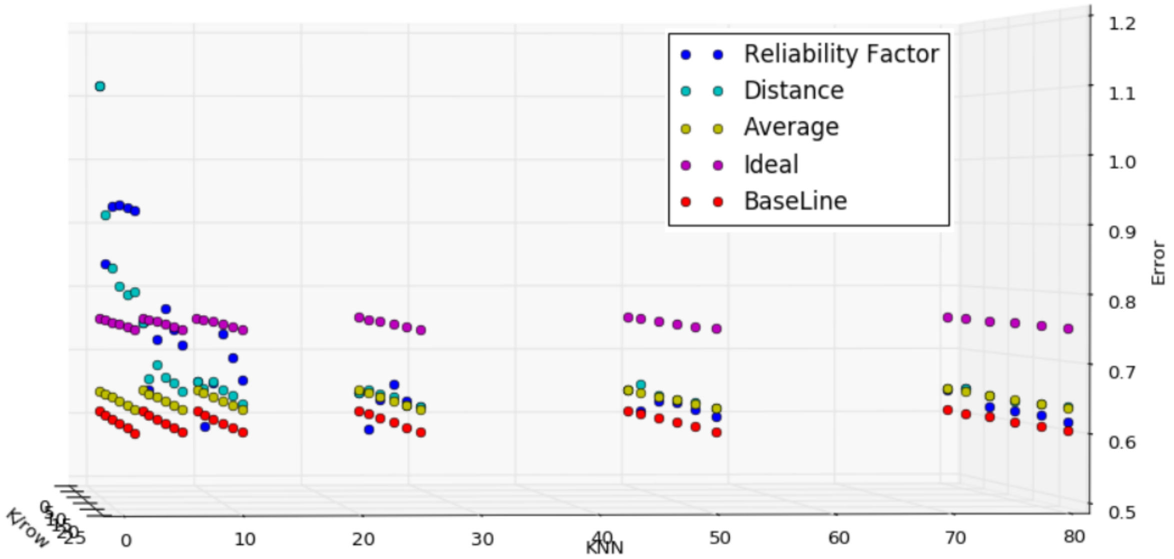


Figure 6.12: $THETA = 0.5$ (8.1% messages)

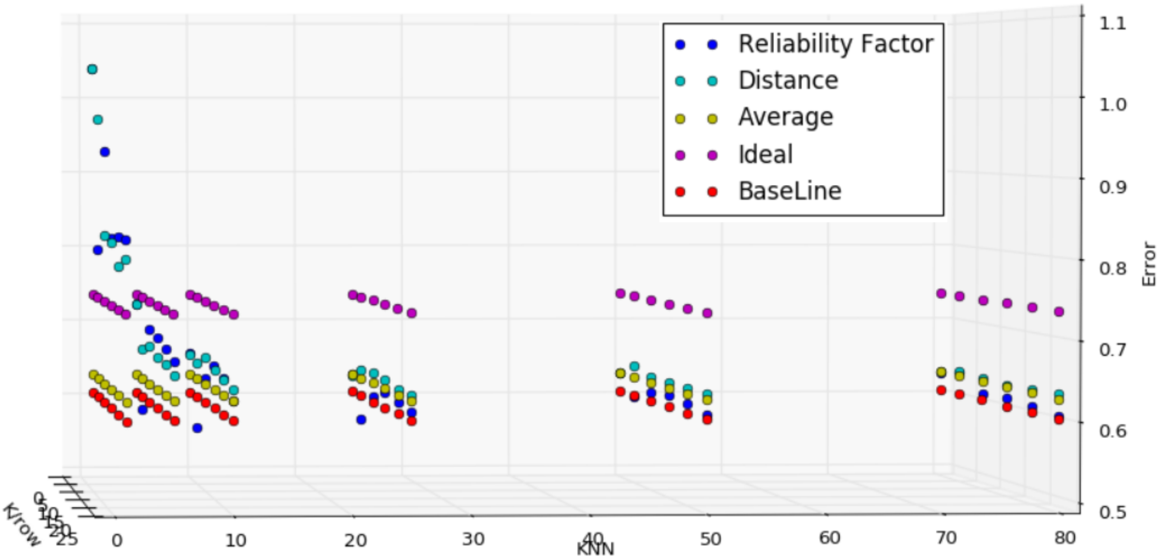
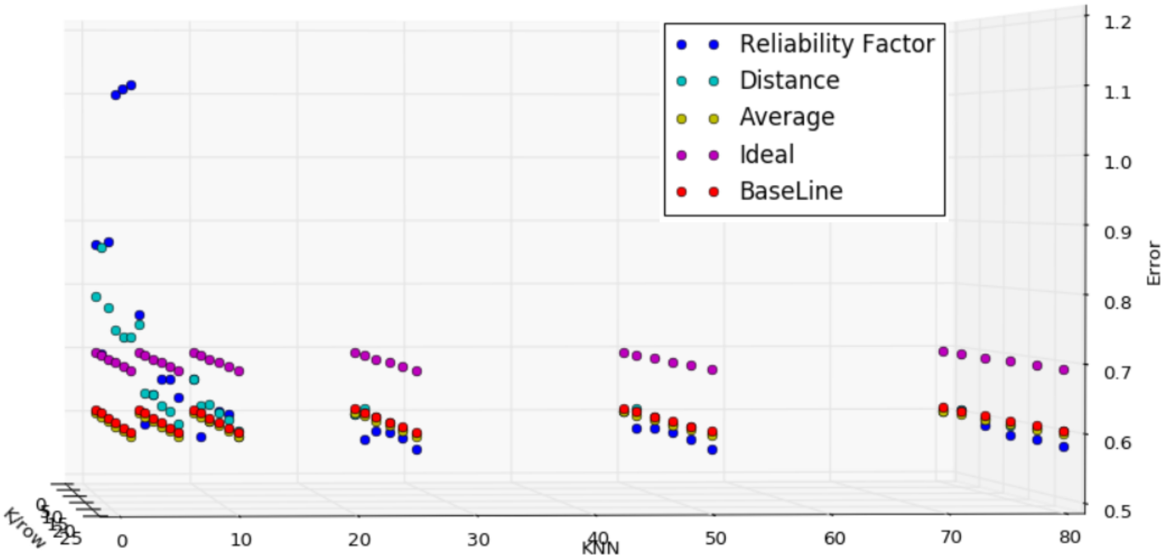


Figure 6.13: $THETA = 1.0$ (3.7% messages)



From these graphs we can draw a number of conclusions. Firstly, the `ideal` algorithm seems to perform the worst on average. This means that using the correct sensor only leads to a high error. On the other hand when we naively take the `average` of all sensors we get much less error. This implies that knowledge is distributed amongst all the sensors and an optimal subset of sensors should exist to get the least amount of error. The `baseline` demonstrates the scenario where we have access to 100% of the data and in fact produces the lowest error in most cases. Although one notices that as we increase $THETA$ each algorithm approaches the baseline and in some cases we surpass it. This is due to the fact that with a large $THETA$ the model inside the concentrator only captures the essential information and is much more general than the local models.

We introduced the `distance` and `reliability factor` algorithms which in contrast to the other algorithms only use a small subset of the models defined by the KNN. If KNN is equal to the number of clusters, then the predicted result would be equal to the `average`. An important thing to note is that KNN seems to be a more important factor to accuracy rather than the number of clusters. KNN is the variable which defines the number of models to choose for averaging at the concentrator level. If $KNN = 1$, then only the first model is chosen. So, assuming that the algorithm we have in place always returns the ideal sensor, we will approach the `ideal` plane instead of the `baseline` plane.

The predicted result from a very small KNN will be heavily biased and in fact as one can see from the graphs the error is much higher than the average. From these results we found that the ideal sensor does not produce the least error. Thus, we should have a high KNN but at the same time less than the number of clusters so as to be able to choose a subset of the models, and not all models.

A high number of clusters for each device enables the concentrator to know more about the underlying data. One has to keep in mind that for every cluster, each device has to transmit more bits which would decrease the network efficiency. From these graphs we observe that with 5 clusters for each device we are able to approach the `baseline` using the `reliability factor` algorithm. Increasing KNN does not impact network efficiency as everything would have been already stored on the concentrator at this stage. We also noticed that the distance algorithm seems to produce a more stable set of errors.

6.3 Network Efficiency

In this section we examine whether our system achieves efficient knowledge forwarding at the expense of data accuracy. We compare our system against a simple model which forwards all the data. To evaluate the efficiency we adopt the mica2 [14] model. Mica2 operates with a pair of AA batteries that approximately supply 2200 mAh with effective average voltage 3V. The energy costs for a single CPU instruction and transmitting/receiving costs are defined in table 6.1

Node Operation Mode	Energy Cost
Instruction Execution	4 nJ/Instruction
Idle - Stand by	9.6 mJ/s - 0.33 mJ/s
Transmitting - Receiving	720 nJ/bit - 110 nJ/bit

Table 6.1: Mica2 Energy Costs

Algorithm	Messages	Messages %
Simple Model	111101	100%
THETA 0.001	94142	84.7%
THETA 0.01	69282	62.3%
THETA 0.1	29704	26.7%
THETA 0.5	8960	8.1%
THETA 1.0	4126	3.7%

Table 6.2: Theta Messages

Chapter 7

Conclusion

7.1 Achievements

7.2 Future Work

7.3 Final Remarks

Appendix A

First appendix

Bibliography

[1] N. S. Altman. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, 46(3):175–185, 1992.

[2] C. Anagnostopoulos, T. Anagnostopoulos, and S. Hadjiefthymiades. An adaptive data forwarding scheme for energy efficiency in wireless sensor networks. In *2010 5th IEEE International Conference Intelligent Systems*, pages 396–401, July 2010.

[3] Christos Anagnostopoulos. Time-optimized contextual information forwarding in mobile sensor networks. *Journal of Parallel and Distributed Computing*, 74(5):2317 – 2332, 2014.

[4] Christos Anagnostopoulos. Intelligent contextual information collection in internet of things. *International Journal of Wireless Information Networks*, 23(1):28–39, 2016.

[5] R. Arroyo-Valles, A. G. Marques, and J. Cid-Sueiro. Optimal selective transmission under energy constraints in sensor networks. *IEEE Transactions on Mobile Computing*, 8(11):1524–1538, Nov 2009.

[6] WESAM BARBAKH and COLIN FYFE. Online clustering algorithms. *International Journal of Neural Systems*, 18(03):185–194, 2008. PMID: 18595148.

[7] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[8] Lon Bottou and Yann LeCun. Large scale online learning. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schlkopf, editors, *NIPS*, pages 217–224. MIT Press, 2003.

[9] Gail A. Carpenter and Stephen Grossberg. *Adaptive Resonance Theory*, pages 22–35. Springer US, Boston, MA, 2010.

[10] C. T. Chou, R. Rana, and W. Hu. Energy efficient information collection in wireless sensor networks using adaptive compressive sensing. In *2009 IEEE 34th Conference on Local Computer Networks*, pages 443–450, Oct 2009.

[11] Thomas G Dietterich. Ensemble learning. *The handbook of brain theory and neural networks*, 2:110–125, 2002.

[12] R. Charles Fawcett University of Virginia Edward S. Neukrug Old Dominion University. *Essentials of Testing and Assessment: A Practical Guide for Counselors, Social Workers, and Psychologists, 3rd Edition*. Brooks Cole, 3 edition, 2015.

[13] Alex Guazzelli, Kostantinos Stathatos, and Michael Zeller. Efficient deployment of predictive analytics through open standards and cloud computing. *SIGKDD Explor. Newsl.*, 11(1):32–38, November 2009.

[14] Tian He, Sudha Krishnamurthy, John A. Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. Energy-efficient surveillance system using wireless sensor networks. In *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services*, MobiSys '04, pages 270–283, New York, NY, USA, 2004. ACM.

[15] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pages 174–185, New York, NY, USA, 1999. ACM.

[16] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 56–67, New York, NY, USA, 2000. ACM.

[17] Richard M. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? In *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume 1 - Volume I*, pages 416–429, Amsterdam, The Netherlands, The Netherlands, 1992. North-Holland Publishing Co.

[18] Andrey N. Kolmogorov. *Foundations of the Theory of Probability*. Chelsea Pub Co, 2 edition, June 1960.

[19] J. N. Laneman, D. N. C. Tse, and G. W. Wornell. Cooperative diversity in wireless networks: Efficient protocols and outage behavior. *IEEE Transactions on Information Theory*, 50(12):3062–3080, Dec 2004.

[20] Hyunyoung Lee, A. Klappenecker, Kyoungsook Lee, and Lan Lin. Energy efficient data management for wireless sensor networks with data sink failure. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference, 2005.*, pages 7 pp.–210, Nov 2005.

[21] Terje K. Lien, Seung-Jun Shin, Jungyub Woo, and Sudarsan Rachuri. 21st cirp conference on life cycle engineering predictive analytics model for power consumption in manufacturing. *Procedia CIRP*, 15:153 – 158, 2014.

[22] Arati Manjeshwar and Dharma P. Agrawal. Teen: A routing protocol for enhanced efficiency in wireless sensor networks. In *in Proc. IPDPS 2001 Workshops*, 2001.

[23] HweeHwa Pang and Kian-Lee Tan. Authenticating query results in edge computing. In *Proceedings of the 20th International Conference on Data Engineering*, ICDE '04, pages 560–, Washington, DC, USA, 2004. IEEE Computer Society.

[24] J. L. Paul. Smart sensor web: Web-based exploitation of sensor fusion for visualization of the tactical battlefield. *IEEE Aerospace and Electronic Systems Magazine*, 16(5):29–36, May 2001.

[25] Michael Rabinovich, Zhen Xiao, and Amit Aggarwal. *Computing on the Edge: A Platform for Replicating Internet Applications*, pages 57–77. Springer Netherlands, Dordrecht, 2004.

[26] Andreas Richter, John P. Burrows, Hendrik Nusz, Claire Granier, and Ulrike Niemeier. Increase in tropospheric nitrogen dioxide over china observed from space. *Nature*, 437(7055):129–132, Sep 2005.

[27] R. C. Shah and J. M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Wireless Communications and Networking Conference, 2002. WCNC2002. 2002 IEEE*, volume 1, pages 350–355 vol.1, Mar 2002.

[28] M. P. urii, Z. Tafa, G. Dimi, and V. Milutinovi. A survey of military applications of wireless sensor networks. In *2012 Mediterranean Conference on Embedded Computing (MECO)*, pages 196–199, June 2012.

[29] Yu Wang, Shiwen Mao, and R. M. Nelms. A distributed online algorithm for optimal real-time energy distribution in smart grid. In *2013 IEEE Global Communications Conference, GLOBECOM 2013, Atlanta, GA, USA, December 9-13, 2013*, pages 1644–1649, 2013.

[30] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[31] Hsun-Ping Hsieh Yu Zheng, Furui Liu. U-air: When urban air quality inference meets big data. In *Proceedings of the 19th SIGKDD conference on Knowledge Discovery and Data Mining*. KDD 2013, August 2013.