DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Simulation & Analysis of Peer-to-Peer Network Quality for Measurement Scheduling

Online algorithms, Application for Network QoS Monitoring

GUSTAF LINDSTEDT

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Simulation & Analysis of Peer-to-Peer Network Quality for Measurement Scheduling

Online algorithms, Application for Network QoS Monitoring

GUSTAF LINDSTEDT

Supervisor: Stefano Markidis
Examiner: Anders Lansner
Supervisors at RISE SICS: Ian Marsh, Liam McNamara

January 1, 2019

Abstract

With the growing dependency on Internet connectivity in our daily lives, monitoring connection quality to ensure a good quality of service has become increasingly important. The CheesePi project aims to build a platform for monitoring connection quality from the home user's perspective. And with peer to peer technologies becoming more prevalent the need for quality of service monitoring between peers become more important. This thesis analyses the problem of scheduling connection quality measurements between peers in a network. A method is presented for scheduling measurements which make use of statistical models of the individual links in the network based on previous measurement data. The method applies the ADWIN1 adaptive windowing algorithm over the models and decides a priority based on the relative window sizes for each link. This method is evaluated against a round-robin scheduler through simulation and is shown to provide a better scheduling than round-robin in most cases in terms of achieving the most "information gain" per measurement iteration. The results show that for sudden changes in a network link the scheduler prioritises measurements for that link and therefore converge its view of the network to the new stable state more quickly than when using round-robin scheduling. The scheduling method was developed to be practically applicable to the CheesePi project and might effectively be deployed in real systems running the CheesePi platform. The thesis also contains an evaluation of two online algorithms for mean and variance as to how they react to change in the data source from which the samples are taken.

Sammanfattning

Med det ökade beroendet på uppkoppling till internet i vårt dagliga liv har det blivit allt viktigare att kontrollera uppkopplingskvaliteten för att säkerställa att slutanvändaren får en bra service. CheesePi-projektet har som mål att bygga en plattform för att monitorera uppkopplingskvaliteten från en hemanvändares perspektiv. I samband med att peer-to-peer teknologier förekommer mer blir det också allt viktigare att säkerställa uppkopplingskvaliteten mellan hemanvändare. Den här rapporten analyserar problemet med att planera mätningar av uppkopplingskvaliteten mellan hemanvändar-noder i ett nätverk. En metod för att planera mätningar presenteras, som använder sig av statistiska modeller av de individuella länkarna i nätverket som baseras på tidigare mätdata. Metoden applicerar ADWIN1 algoritmen, som använder adaptiva fönster, över de statistiska modellerna och bestämmer en prioritet baserat på fönstrens relativa storlek för varje länk. Denna metod utvärderas mot en "round-robin"-planerare genom simulering och demonstreras ge bättre planeringsresultat än "round-robin" i de flesta fall, när det kommer till att uppnå bäst "informations-ökning" varje mätcykel. Resultaten visar att för plötsliga förändringar i en nätverkslänk prioriterar planeraren mätningar för den länken, och därför konvergerar dess vy av nätverket till det nya stabila tillståndet fortare än för "round-robin"-planeraren. Planeringsmetoden har utvecklats för att vara användbart för CheesePi-projektet och har en möjlighet att användas på riktiga system som kör CheesePi-plattformen. Rapporten innehåller också en utvärdering av två "online"-algoritmer för att beräkna medeltalet och variansen, med avseende på hur de reagerar till förändringar i datakällan som mätvärdena utvinns från.

Contents

Contents **vii**

1 Introduction **1**
1.1 Background . . . 1
1.1.1 The Internet . . . 1
1.1.2 Autonomous System . . . 1
1.1.3 IP Addresses . . . 2
1.1.4 Network Address Translation (NAT) . . . 2
1.1.5 MAC Addresses . . . 3
1.1.6 The CheesePi Platform . . . 3
1.1.7 The Raspberry Pi . . . 3
1.1.8 Python . . . 3
1.2 Motivation . . . 4
1.3 Problem . . . 5
1.4 Ethics and Sustainability . . . 5
1.5 Contributions . . . 6
1.6 Outline . . . 6

2 Literature Study **7**
2.1 Measurement Agents . . . 7
2.1.1 DIMES . . . 7
2.1.2 Neubot . . . 7
2.1.3 Dasu . . . 8
2.2 Measurement Scheduling . . . 8

3 Methods **11**
3.1 Information . . . 11
3.1.1 Ping Delay as a Metric . . . 11
3.1.2 Online Calculation of Mean and Variance . . . 13
3.1.2.1 The Definition of Mean and Variance . . . 13
3.1.2.2 Online Algorithms . . . 14
3.2 Simulation . . . 16
3.2.1 Defining Useful Information . . . 16

3.2.2 Generating Measurement Schedules . . . 17
3.2.2.1 The ADWIN1 algorithm . . . 18
3.3 Scheduling System Design . . . 19
3.3.1 The CheesePi . . . 19
3.3.2 The Scheduling Server . . . 20
3.3.2.1 Client-Server Communication . . . 20
3.3.2.2 Server Structure . . . 20
3.3.2.3 Database Structure . . . 20

4 Design and Implementation **23**
4.1 Simulation Framework . . . 23
4.1.1 Designing Simulation Test Cases . . . 24
4.1.2 Simulation Setup Format . . . 25
4.2 Plotting . . . 26
4.3 Software Environment . . . 26
4.3.1 Python Packages . . . 26
4.3.2 Database . . . 26

5 Experimental Environment **27**
5.1 Hardware . . . 27
5.2 Metrics for Evaluation . . . 27
5.3 Online Algorithms for Mean and Variance . . . 28
5.3.1 Uniformly Weighted Mean and Variance . . . 28
5.3.2 Exponentially Weighted Mean and Variance . . . 28
5.3.3 Comparing Performance for Different Distribution Parameters 28
5.3.4 Comparing Reaction to Change in Distribution Parameters . 29
5.4 Improving Information Gain Through Scheduling . . . 29
5.4.1 Three Peer Simulation with Round-Robin scheduling . . . 30
5.4.2 ADWIN1-Based Scheduling Using Three Peers . . . 30
5.4.3 ADWIN1-Based Scheduling Using Ten Peers . . . 31

6 Results **33**
6.1 Simulation Performance . . . 33
6.2 Online Algorithms for Mean and Variance . . . 33
6.2.1 Uniformly Weighted Mean and Variance . . . 33
6.2.2 Exponentially Weighted Mean and Variance . . . 36
6.2.3 Comparing Performance for Different Distribution Parameters 38
6.2.4 Comparing Reaction to Change in Distribution Parameters . 40
6.3 Improving Information Gain Through Scheduling . . . 46
6.3.1 Three Peer Simulation with Round-Robin Scheduling . . . 46
6.3.2 ADWIN1-Based Scheduling Using Three Peers . . . 48
6.3.2.1 Small Change . . . 48
6.3.2.2 Large Change . . . 50
6.3.2.3 Gradual Change . . . 52

6.3.3 ADWIN1-Based Scheduling Using Ten Peers 54
6.3.3.1 Small Change 54
6.3.3.2 Large Change 56
6.3.3.3 Gradual Change 58
6.3.3.4 Multiple Simultaneous Changes 60

7 Discussion & Conclusion **63**
7.1 Discussion 63
7.1.1 Future Work 64
7.2 Conclusion 65

A Simulation Configurations **67**

Bibliography **69**

Chapter 1

Introduction

This chapter gives an overview of the motivations and goals of the thesis, along with a section outlining the structure of the report.

1.1 Background

In this section some background is given on a few topics that are relevant to the rest of the paper.

1.1.1 The Internet

The Internet is, in essence, precisely what the word implies: a set of many interconnected networks. These networks are controlled by a huge number of different entities. Some of them are Internet Service Providers (ISP) who provide internet access to regular customers, both corporate and private, while others might be universities or other more high level entities that provide and manage the infrastructure for a whole country's, or even several countries' connection to the rest of the world.

1.1.2 Autonomous System

The many networks that make up the Internet are grouped in so called Autonomous Systems (AS), which are groups of one or more IP address prefixes that have a single clearly defined routing policy.[1] An Autonomous System can potentially be run by several operators, however it is common that an ISP has full control over its own Autonomous Systems.

Since different Autonomous Systems are managed by different entities they can differ in how they route traffic. They might also route traffic inconsistently, which might be due to load balancing between alternate paths since routing decisions are done independently for each packet. This can lead to packet routing asymmetry where a packet travelling from point **A** to point **B** can take a different path than a packet travelling in the opposite direction.[2]

1.1.3 IP Addresses

In order to identify devices on a network, either a local network or the Internet, the Internet Protocol (IP) is used. The Internet Protocol uses IP addresses as unique identifiers for devices in a network. In the Internet Protocol version 4 (IPv4), the version that is most commonly used today, the address space consists of 32 bits. An IPv4 IP address is usually written as four numbers separated by dots with each number representing one byte of the 32 bits in the address space. For example: `127.0.0.1`, `192.168.0.1` etc. The numbers representing the bytes can be any value in the range 0–255, although some IP address ranges are reserved by the specification of IPv4. A device can have a public IP address, meaning that it is globally accessible on the Internet. However, most devices only have local IP addresses which means that they are only accessible within their own network, most commonly a network set up by a home router. For example the above address `192.168.0.1` belongs to the address space `192.168.0.0-192.168.255.255` (commonly written as `192.168.0.0/16`, with 16 denoting the bitmask of the significant bits).[3] This address space is reserved in the IPv4 specification for private networks, which means it is primarily used to identify devices on a home network, and no server on the Internet can use any of these addresses.

IP addresses can change, and frequently do. This is usually not a problem because public servers make use of domain names to identify themselves to the user. These domain names are then resolved using the Domain Name System (DNS) to get the IP address. The Domain Name System is a network of distributed servers which keep records of which domain names map to which IP addresses. If a server maintainer changes the IP of a server, the maintainer has to update the DNS record of the DNS server which is responsible for that domain name to point to the new one. End users normally don't need to worry if their public IP address is changed because as a rule almost all internet traffic is client initiated.

1.1.4 Network Address Translation (NAT)

Network Address Translation is a technique that is most commonly used by routers to map local network IP addresses to a public address. For example most home users have one public IP address designated to them from their ISP. In order to have multiple devices at home connect to the Internet the router keeps a table of active connections. It then dynamically assigns a connection from a local IP address to an IP address on the Internet to a specific port of the router's public IP address. Now the device on the other end communicates with the router, on that port, which in turn forwards those packets to the initiating device on the local network. This is a very useful technique and solves many problems. However, it also creates many problems whenever communication does not follow the traditional client-server pattern, for example with peer-to-peer communication.

There are a few techniques for end peers who are both on private networks to be able to contact each other such as STUN[4] and TURN[5], however these methods

are not trivial to implement. There are many different types of NAT and every implementation can differ subtly in behaviour. This makes it hard to create a catch-all solution to NAT traversing.

1.1.5 MAC Addresses

At the lower layer, below the IP addresses, MAC addresses are used to uniquely identify network interfaces of devices (MAC is an acronym for Media Access Control). This means that a device's Ethernet port and Wi-Fi card have separate MAC addresses. MAC addresses are also supposed to be globally unique, meaning that no two network interfaces should have the same MAC address. The MAC address space consist of 48 bits and is usually represented as six groups of two hexadecimal digits separated by colons or hyphens, e.g. `01:23:45:67:89:ab` or `01-23-45-67-89-ab`.

1.1.6 The CheesePi Platform

The CheesePi platform[1][6] is a broadband monitoring platform with the stated goal of attempting to objectively characterise the quality of the service that the user receives. The platform primarily targets the Raspberry Pi device and specifically the Raspbian Linux distribution, however the platform aims to be as platform agnostic as possible. In order to be platform agnostic and to ease maintainability CheesePi is for the most part implemented using Python. Python is a widely used high level programming language which uses an interpreter, eliminating the need to compile the source code for specific platforms.

1.1.7 The Raspberry Pi

The Raspberry Pi[7] is a line of small and affordable general purpose computers developed by the Raspberry Pi Foundation. It was first developed with the intent of making a cheap hackable device in order to promote teaching of basic computer science in schools and developing countries, however it was enthusiastically received by hobbyists as well who wanted the same kind of devices for various projects ranging from robots to emulators and home entertainment systems.

The various Raspberry Pi models all run on ARM CPUs, most have Ethernet sockets (the recently released Raspberry Pi 3 even has built in Wi-Fi[8]), and they all run Linux without issue. Since a Raspberry Pi is so cheap and easily available to consumers, it is a good primary hardware platform for the CheesePi. Odds are that many people who would also be interested in running a CheesePi node might already own a Raspberry Pi that they can deploy to.

1.1.8 Python

Python is a high-level, general-purpose, interpreted programming language. It has many features such as garbage-collection and a dynamic type system, and the syntax

[1] `http://cheesepi.sics.se`

is expressive and easy to read. In addition to its language features Python also has a package manager called 'pip' for installing third party packages. The community around Python is large and active, which is reflected in the large number of high quality packages available in the pip repositories.

There are two main versions of Python, Python 2 (2.7.11 at time of writing) and Python 3 (3.5.1 at time of writing). Python 3 is meant to be the successor of Python 2 and has been developed alongside maintaining Python 2. However, adoption of Python 3 has been slow and still a lot of packages only fully support Python 2. Python 2 is also still the default version for many distributions of Linux. For this reason, the CheesePi platform is for the most part written in Python 2, therefore all code written for this thesis will also be written in Python 2 compatible code.

1.2 Motivation

The Internet is being used by people all over the world, and its usage increases day by day. As technology progresses, with an increasing amount of web based services and all of our devices becoming connected, the amount of Internet traffic increases as well, both for individual users and for the infrastructure enabling it. Monitoring the quality of connections over the Internet is becoming increasingly important due to these trends, not only for the sake of consumers being able to evaluate that they are getting the connection that their Internet Service Providers claims, but more importantly to ensure the quality and availability of the infrastructure so that we can continue to rely on Internet-based services such as email and banking which have become crucial to everyday life.

A common source of frustration for end users is when their Internet connection does not behave as expected. This may be due to a number of various causes, such as traffic congestion, route changes, or simply a bad WiFi signal. It can be hard for end users to know what the root cause for the disruption in the service is. In order for both end users and regulators to make sure that the service provided by the ISPs is the same as advertised there needs to be monitoring of the quality of service (QoS). Furthermore, monitoring data is valuable to the ISPs as well so that they can evaluate the service that they provide and detect when there is a need to upgrade infrastructure or perform other kinds of maintenance.

Recent years have seen an increase in applications that use peer-to-peer technology for communication between clients, for example protocols such as BitTorrent[9] and WebRTC[10]. This move away from the traditional client-server model means that it is no longer sufficient to have a good connection to the servers that host the websites and services you use, but to have a high quality connection to any device on the Internet, including your friend's home router.

When performing measurements on a path between two points on the Internet, there is no guarantee that the result will be the same in both directions due to packet routing asymmetry. This greatly increases the amount of measurements that

needs to be performed and results that needs to be analysed to get an accurate view of the network, which includes any asymmetrical properties.

Even without assuming that we cannot trust connections over the internet to be symmetrical in terms of speed and quality, platforms such as the CheesePi will undoubtedly face issues with resource limitations when attempting to expand to peer-to-peer measurements for evaluating and analysing the QoS.

In order to provide useful insights into the QoS an end user is receiving, measurements need to be collected and analysed and continuously updated. However every peer cannot feasibly perform continuous measurements against all other peers. There has to be some sort of prioritisation, where the obvious and easy solution would be to schedule at random. However there might be better solutions, limiting the amount of measurements each peer has to perform while still gathering the information needed to provide useful insights into the QoS.

1.3 Problem

The problem of scheduling measurements with the intent of maximizing the amount of information gained presents a few questions:

- We need to establish what kind of information we have to base our scheduling decisions on. Our only dataset is the collection of samples we've gathered from previous measurements, but we need to find out what information we can infer from them.
- After we've identified what information we have we need to find ways in which this information can give us suggestions of which measurement targets would give us the most new information for our models.

Throughout this we will need to keep in mind the assumed use case for peer-to-peer QoS monitoring for the CheesePi. Therefore we need to consider the efficiency of the scheduling so that it is potentially usable in a real world scenario. We also need to be aware of the fact that due to the dynamic nature of the Internet, the sources we are measuring can change over time.

To limit the scope the thesis is based entirely around the ping round-trip time metric as the measure for quality of service, but the aim is to keep it generally applicable to other metrics.

1.4 Ethics and Sustainability

The work presented in this thesis is not considered to have any ethical implications, either from the work itself or from the conclusions drawn from it. Sustainability is one of the main factors for the problem statement of this thesis, where the need to optimise the usage of limited resources for performing measurements is expressed.

A good measurement scheduler would lead to fewer measurements being needed, which would free up more computing power and network bandwidth for other tasks.

1.5 Contributions

This thesis provides the following contributions:

- A comparison of two online algorithms for calculating mean and variance in terms of convergence and stability when the underlying distribution changes.
- A simulation based method for evaluating different methods of prioritising measurement targets by comparing the relative information gain on subsequent measurements.
- A method for prioritising measurement targets based on data acquired from previous measurements, which provides a better information gain than equal prioritisation of targets.

1.6 Outline

A brief overview of some related work in terms of QoS agents and scheduling of measurements is found in Chapter 2. The overarching methods used in this thesis to explore the question as well as some background will be discussed in Chapter 3. Then in Chapter 4 the implementation details for the system developed to simulate and test the methods will be presented. In Chapter 5 the details regarding the setup used when running the simulations, as well as the configurations of the simulations themselves and the metrics used for the results are presented. Following that Chapter 6 will present the results for the simulations previously described. After that will follow a brief conclusion and discussion in Chapter 7 followed by the Bibliography. Appendix A contains the simulation configurations used for the results shown in Chapter 6.

Chapter 2

Literature Study

In this chapter some related work is presented in the fields of Internet quality of service measurement agents as well as sensor measurement scheduling.

2.1 Measurement Agents

2.1.1 DIMES

DIMES[11] was a distributed measurement infrastructure project with the aim of measuring and mapping the structure and evolution of the Internet, both at the AS level and individual router nodes. It was launched in 2004 and had grown to 2300 users with over 3600 agents after nine months of operation, in 70 different countries.

The DIMES project chose a distributed infrastructure because using only a few dozen or hundreds of vantage points would lead to a strong bias towards customer-provider links in the mapped topology, missing many peer-to-peer links. The system was designed to enable simultaneous experiment deployments by many researchers using a scripting language called PENny. A centralized management system was chosen for DIMES due to the security and complexity issues inherent to a distributed system. The management system would maintain a system of queues with many-to-many relationships between agents and scripted experiments.

Because of the fact that an agent might not be active at all times and would be vulnerable to hijack attacks if it was listening for connections on a known port, as well as the possibility that agents could be behind firewalls which block incoming connections, it was decided that the communication between the agents and the central system would be agent initiated.

2.1.2 Neubot

Neubot[12] is a software architecture, and agent, for distributed measurements of Internet access quality and network neutrality. The agent is run by volunteers on their home computer which periodically monitors quality of service. All results are then collected to a centralized server and the data is made publicly available.

A similarity with the CheesePi project is that it is meant to permanently monitor the QoS instead of performing isolated experiments, and that it is from an end-user perspective.

Neubot has two modes of measurement operation, the first is client-server, where the agent performs measurements against a set of test servers deployed at internet exchanges. The second mode is client-client, where measurements would be performed between agents assuming they are not firewalled.

2.1.3 Dasu

Dasu[13] is a distributed measurement experimentation platform that was launched in July 2010 and was installed by over 90 000 users across 1802 networks and 147 countries. One of the stated goals was to create a platform with a model that aligns the objectives of both the platform hosts and the experimenters. With this in mind Dasu was built to support both broadband characterization and measurement experimentation.

The Dasu client is implemented as an extension to BitTorrent, one of the most popular peer-to-peer networks. This provided additional benefits in being able to take advantage of BitTorrent users' comparatively long session times. Although the clients are distributed the platform uses a host of centralized servers for configuration, experiment administration and coordination, as well as data gathering. In the interest of security the experiments are implemented using a rule-based declarative syntax which is limited in functionality so as to hinder the ability to write malicious experiments.

A few case studies are highlighted in the Dasu paper which were performed using the platform, one of which with the purpose of studying routing asymmetry. In the study they found that approximately 28% of the paths tested between clients and server nodes had routing asymmetries.[13]

2.2 Measurement Scheduling

There has been a bit of research done in the field of sensor selection in sensor networks in the interest of maximising accuracy and minimising cost. In this context resources could be constrained by the cost of transmitting measurement data between sensor nodes, or through the inability to perform simultaneous measurements due to sensors interfering with each other, for example with sonar.

One suggested method is selecting sensors based on their locality with respect to the perceived target, using nearest neighbour or Mahalanobis distance. Zhao et. al. suggest an improvement, and demonstrates its effectiveness, to such methods by using a heuristic which limits the amount of times a sensor can be selected in order to introduce more sensory diversity.[14]

Another method entails selecting sensors at random according to a probability distribution, where the probability distribution is chosen so as to minimize the expected steady state error covariance.[15]

A common theme is to try to find a definition of the expected information utility, or information gain, of selecting a specific sensor. Zhao et. al. describes one method of attempting to predict the expected belief state if a simulated measurement of a sensor in the current belief state was incorporated. The information utility of that sensor could then be quantified by the entropy on the new distribution from the simulated measurement.[14]

These methods however depend on the assumption that the sensors are measuring the same process. This is however not the case for the problem this thesis deals with, where each sensor is assumed to measure a unique process.

Chapter 3

Methods

This chapter describes the approach taken to attempt to solve the problem statement of the thesis.

3.1 Information

The information that we have to work with given the problem statement are the previous samples that our measurement agents have gathered when performing measurements towards their peers. We will only consider the ping round-trip time (ping RTT, or ping delay) as the metric for measuring connection quality. Therefore the collection of previous samples are assumed to be results of ping measurements. There are many other ways to evaluate connection quality, however ping RTT is one of the simplest, and deciding which metrics should be used to measure different qualities of a connection is out of scope for this thesis.

The ping RTT is the time it takes for a packet to travel from its source, to its destination host, and back again. The time it takes depends on a number of things, such as physical distance, the amount of 'hops' the packet has to travel (i.e. how many routers the packet has to pass through to get to the destination), packet congestion along the path (i.e. if a network node along the path is routing more traffic than it can handle) and so on.

3.1.1 Ping Delay as a Metric

If you perform a series of pings, the distribution of the ping delay results usually follows a pattern where there's a hard lower bound which represents the physical limits of the route (i.e. physical distance and cable/signal performance). Most of the delay results tend to group close to this lower bound, with a tail of results falling off towards longer delay times. This is demonstrated in Figures 3.1 and 3.2, which are histograms of ping measurements of 1000 packets each to two different destinations.

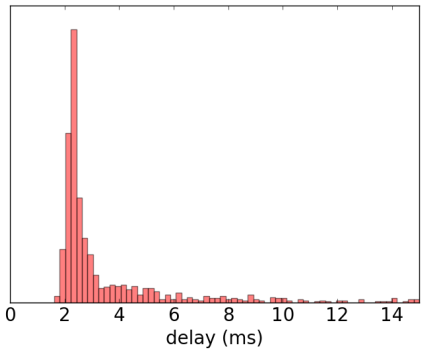


Figure 3.1: Histogram of the delay to www.theonion.com.

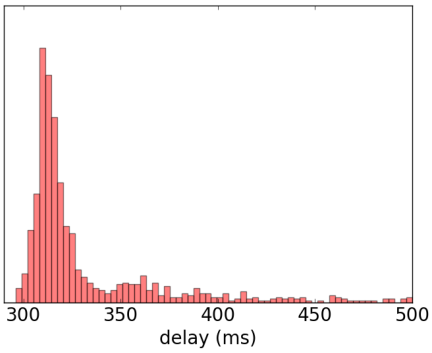


Figure 3.2: Histogram of the delay to www.baidu.cn.

There are a few properties of ping RTT that are worth noting if we want to use it as a metric to measure connection quality. A metric in the mathematical sense is defined as a function over a set defined as fulfilling four basic properties, which are listed below:

1. $d(x, y) \geq 0$ **non-negativity**
2. $d(x, y) = 0 \iff x = y$ **identity of indiscernibles**
3. $d(x, y) = d(y, x)$ **symmetry**
4. $d(x, z) \leq d(x, y) + d(y, z)$ **triangle inequality**

However, due to the structure of the Internet and how different ISPs route traffic as described in Section 1.1.1, we cannot guarantee that a packet travelling from point A to point B on the network will take the same path as a packet travelling in the opposite direction. Therefore we cannot guarantee that ping RTT satisfies **symmetry**. Furthermore, since we cannot be certain that packets take the shortest possible route to their destination, we cannot guarantee that the **triangle inequality** holds.

If we are attempting to measure connection quality based on ping RTT, these properties limit the type of conclusions we can draw from the measurement data. Most notably the lack of symmetry introduces the concept of directionality and so doubles the amount of paths that needs to be measured (two directions for every node pair). This greatly increases the size of the problem if our intent is to try to approach a 'complete' view of the network through our measurement data. The number of unique paths grow according to how many permutations of size two that can be selected from the pool of measurement agents:

$$\frac{n!}{(n-k)!} \quad \text{where} \quad \begin{aligned} n &= \text{the number of unique objects} \\ k &= \text{the size of the permutations} \end{aligned} \tag{3.1}$$

For example, given a network of 100 measurement agents, we already have $\frac{100!}{98!} = 9900$ possible paths to measure using Equation (3.1). This leads us to the main question that this thesis is trying to answer: How do we choose measurements to perform in order to gain as much knowledge as possible using our limited resources?

3.1.2 Online Calculation of Mean and Variance

One way to try to determine if a certain link is worth performing measurements on is to calculate and store a few statistical values based on previous data gathered. The mean and variance of the measurement results are particularly interesting because they have clear interpretations, the mean is the average result of the ping RTT, and the variance is a measure for the spread of results around the mean, i.e. how much fluctuation there is in the sample set. The mean and variance are traditionally calculated by iterating through all samples of the sample set. However, as more and more samples are gathered for a certain link we don't want to constantly make database lookups for the entire set of previous samples we've received just because we want to update a statistical value. Fortunately there are a few online algorithms for calculating the mean and variance, which are presented in Section 3.1.2.2. But for some background we should first cover the definition of mean and variance.

3.1.2.1 The Definition of Mean and Variance

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} \qquad \text{sample mean} \tag{3.2}$$

$$\mathrm{E}[X] = \sum_{i=1}^{n} p_i x_i \qquad \text{expected value of } X \tag{3.3}$$

$$\mu = \mathrm{E}[X] \qquad \text{population mean} \tag{3.4}$$

The sample mean $\bar{x}$, as defined in equation (3.2), converges towards the population mean μ when the number of samples are increased, and can therefore be used to estimate the population mean. The population mean in turn is the expected value $\mathrm{E}[X]$ of the random variable X. Intuitively the expected value is the average of the population.

The variance is a measure of how much a set of numbers are spread out. It is defined as the expected squared difference of a random variable X from the mean, as seen in equation (3.5). The standard deviation σ is defined as the square root

of the variance. Standard deviation is sometimes preferred over variance since it is expressed in the same units as the data.

$$\mathrm{Var}(X) = \mathrm{E}[(X - \mathrm{E}[X])^2] \qquad \text{definition of variance} \tag{3.5}$$

$$\mathrm{Var}(X) = \mathrm{E}[(X - \mu)^2] \qquad \text{from (3.4)} \tag{3.6}$$

$$\mathrm{Var}(X) = \sum_{i=1}^{n} p_i (x_i - \mu)^2 \qquad \text{from (3.3)} \tag{3.7}$$

$$\mathrm{Var}(X) = \sigma^2 \qquad \text{relation to standard deviation} \tag{3.8}$$

$$\sigma = \sqrt{\mathrm{Var}(X)} \qquad \text{from (3.8)} \tag{3.9}$$

Since samples are drawn in proportion to their probability from a given distribution we can simplify equation (3.7) by removing the probability factor p_i. However we must then normalise the sum by the number of samples:

$$\mathrm{Var}(X) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2 \tag{3.10}$$

3.1.2.2 Online Algorithms

Since we want to continually update our estimates of mean and variance we don't want to have to keep looking at old data over and over again in order to incorporate new results into the estimate. Fortunately there are a few online algorithms for calculating mean and variance. When describing algorithms the term 'online' refers to the algorithm not having to access old input in order to process new input, i.e. it can be forgetful of previous data while still producing viable results. In short such algorithms are well suited for cases where there's a potentially unending stream of data to be processed.

A relatively simple online algorithm for mean and variance is described by Welford[16], it is well known and has been studied[17]. Shown below is an example implementation in Python:

```
1  def online_mv(samples):
2      n    = 0
3      mean = 0.0
4      M2   = 0.0
5
6      for x in samples:
7          n = n + 1
8          delta = x - mean
9          mean = mean + delta / n
10         M2 = M2 + delta * (x - mean)
11
12     # normalise sum of squares to get variance, see equation  (3.10)
13     variance = M2 / (n - 1)
14
15     return (mean, variance)
```

Listing 3.1: Online mean and variance

Note that this example only demonstrates how it could be used as a 'one pass' function, but it is easy to see how it could be implemented for a stream of potentially unending data.

A potential issue with this algorithm, depending on the use case, is that it gives old and new samples equal amount of weight when estimating the mean and the variance. But what happens then if something about the data source changes, i.e. the 'real' mean and variance of the underlying distribution change? Presumably it will take some time for the algorithm to correct its values to the new 'reality'. We can surmise that the amount of time it takes to correct the estimates is likely proportional to the amount of data previously gathered, since the data from the new source needs to be in majority to the old data in order for the new state to 'weigh' more in the equation. The old data has left its mark on the estimate and will stay indefinitely. This might be beneficial for detecting a change in the data source, but it means that the approximation would have to be reset every time we think that change has occurred.

Another way to approach this issue is to make sure that the samples are not uniformly weighted. For example we might want to continuously favour new samples in our estimate. Finch[18] describes one such algorithm using exponential weighting, i.e. the weighting follows an exponential curve, where the impact of old samples on the estimate are continuously reduced in favour of new ones. Below is an example implementation in Python:

```
1  def exp_weighted_mv(samples, alpha):
2      # alpha is a value between 0 and 1, which dictates how
3      # quickly the impact of old samples are diminished
4
5      mean = 0.0
6      variance = 0.0
7
8      for x in samples:
9          delta = x - mean
10         increment = alpha * delta
11         mean = mean + increment
12         variance = (1 - alpha) * (variance + delta * increment)
13
14     return (mean, variance)
```

Listing 3.2: Exponentially weighted online mean and variance

The algorithm is fairly simple, however it introduces the α variable, where $0 < \alpha < 1$, which tweaks the weighting bias of newer over older samples (or older over newer). From the pseudocode one can deduce that lower values of α makes the estimates more rigid, i.e. old samples retain their influence on the estimate for a longer period. Finch does not mention appropriate values for α, probably because it depends on what data is being examined and which behaviour is desirable. Through the process of writing this thesis, by way of trial and error, an α value of around 0.1 or less seems to give the best results for the given use case.

3.2 Simulation

In order to compare methods for prioritising measurement targets we need have some notion of the information gained from performing a measurement, so that we can evaluate if our prioritisation was a good one or not. If we know what the actual value of what we are trying to approximate through measurements is, we can then estimate the information gain of a single measurement provides to our approximation model. The problem is that in a real world situation we don't have an absolute source of truth on the actual value. A simple way to get around this is to evaluate our methods in a simulated environment, where we control the source of truth, so that we can see how our approximation differs from the actual value. This allows us to evaluate which methods might work better, by comparing how efficient they are at approximating the true value.

3.2.1 Defining Useful Information

Under the assumption that we have unlimited resources, we would perform measurements on every path using every conceivable metric at all times. In doing this we could say that we have a reasonably 'good' view of the network (in terms of the

metrics we have chosen). Now this scenario is not feasible in reality, but assuming we have to cut down on monitoring in order to save resources we arrive at the question: Which of these measurements are least important in keeping our view of the network as 'good' as possible? One answer could be to discard the measurements that continue to give the same, or sufficiently similar, results every time we measure. For example, two nodes which are only at a one hop distance from each other with a high speed link connecting them would probably yield similar results each time the path was measured. Furthermore, nodes that are far apart but have a 'stable' link, i.e. the link seems impervious to fluctuations due to load or other causes, would also yield similar results each time. Results from these links, once we have gathered the bare minimum, does not add much new information to our view of the network.

We now have a notion of what type of information we can do without, for the sake of saving resources. The question is then: How do we decide, before knowing the information yield of individual measurements, which of the paths that are worth performing measurements on?

What we need to do is implement some sort of prediction mechanism, which will take into account any previous results and give a value for how potentially interesting a new measurement sample on this link might be. By using such a value we could then rank potential measurements and choose the most promising ones.

3.2.2 Generating Measurement Schedules

Before we get into trying to schedule measurements in an intelligent way we need to consider the simplest solutions, so that we have something to compare to. There are two simple, naive ways to schedule measurements:

Random Choose a random link to measure out of all available.

Round-Robin Choose the link that was least recently measured (or have not been previously measured at all).

Using a round-robin scheduling scheme is beneficial in terms of fairness, in that links are measured in equal amounts. A random scheduling scheme also retains a certain amount of fairness, but is less rigid in the time it takes for a link to be measured again.

For our scheduler, in order to make the most out of our limited resources we would like to filter out links which are mostly 'stable', while prioritising links where we think we lack information (either due to the link being new or because of a recent change in its behaviour). This introduces the question of how we quantise how 'stable' a link is, compared to others.

A very naive scheduler could for example bias towards scheduling measurement of paths which have a high variance in the metrics we're considering, because sources with a high variance intrinsically needs to be sampled to a greater extent than say a source with the same constant result when sampled. A flaw with this method is that it would continue to bias sources with a high variance even though they have been

extensively sampled. It doesn't take into consideration whether the high variance is normal for a specific data source.

In essence we need a method which does some form of analysis on new samples in relation to previously gathered samples from that source. We also need to be able to identify if a change has occurred in the source, so that we can change our priorities dynamically.

3.2.2.1 The ADWIN1 algorithm

The ADWIN1 algorithm described by Bifet and Gavalda[19] is an adaptive windowing algorithm designed to deal with distribution change in data sequences over time. We can use this algorithm to gauge whether our view of a sample source, e.g. a link that is being measured, is undergoing change or if it is stable. If we apply the ADWIN1 algorithm by letting it manage an adaptive window for the estimated variance of our statistical model, then the algorithm will be able to detect whenever our model exhibits a significant change with respect to previous iterations of the model.

The ADWIN1 algorithm keeps a sliding window W of the most recently read input values. The window is used to detect whether a new value represents a statistical change having occurred in the data source. The change detection is based on the null hypothesis that if there has been no change, the (unknown) mean value μ_t has remained constant for all values in W. This hypothesis is tested by calculating the observed mean values $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ for each subwindow partition $W = W_0 \cdot W_1$ and checking if the difference between them is greater or equal to the bound ϵ_{cut}, in which case a change has occurred. This bound depend on the individual sizes n_0 and n_1 of the subwindows and on a chosen confidence value δ, and is calculated as shown in equation (3.12). In simpler terms this means that a change is assumed to have occurred if two "large enough" subwindows have a "distinct enough" difference in their averages.

$$m = \frac{1}{1/n_0 + 1/n_1} \qquad \text{harmonic mean of } n_0 \text{ and } n_1 \tag{3.11}$$

$$\epsilon_{cut} = \sqrt{\frac{1}{2m} \cdot \ln \frac{4}{\delta'}} \ , \quad \delta' = \frac{\delta}{n} \qquad \text{where } n \text{ is the size of } W \tag{3.12}$$

The sliding window is maintained by the algorithm so that it always passes this test. When new data is received the algorithm adds it to the head of the sliding window, and proceeds to adjust the size of the window. It drops elements from the tail as needed, in order to make sure that the difference between the observed mean values of the subwindows of any window split is less than or equal to the ϵ_{cut} value. Pseudo-code is shown in Algorithm 1.

Note: *there seems to be an error in the pseudo-code of the original paper where it says* $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_{cut}$ *instead of* $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \leq \epsilon_{cut}$

Algorithm 1 ADWIN1

Initialize Window W
for each $t > 0$ **do**
  $W \leftarrow W \cup \{x_t\}$
  repeat
    Drop the tail element of W
  until $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \leq \epsilon_{cut}$ holds for every split of W into $W = W_0 \cdot W_1$
  output $\hat{\mu}_W$
end for

This algorithm simply outputs the observed mean based on the values present in the sliding window. Although this observed mean is not specifically what we are after since we already keep our own approximation model of the value, there is another interesting property that we can gather from it, namely the size of the sliding window. If a link is 'stable' the sliding window will be large, since it does not have to drop that many elements in order to stay consistent. However if a link recently experienced a significant change, we can assume that the window will be smaller in order to stay consistent. If we keep an adaptive sliding window for every link a node has, we can look at the lengths of the windows and find a scheduling priority based on that.

3.3 Scheduling System Design

Since the context for this thesis is the potential use case of having peer-to-peer measurements as part of the CheesePi platform the implementation of the scheduler will be done in the form of a server which could potentially be used for the CheesePi. In this section we give a brief outline of the architecture of the server implementation and how peers would interact with it.

3.3.1 The CheesePi

The CheesePi platform is designed to periodically perform a set of defined measurement tasks, and gather the results in a local database. The database used is InfluxDB[20], which is a database specialising in storing time series, i.e. data that carries a time stamp, and is therefore well suited for monitoring values over time. The measurement tasks can either be defined by the user or left to the default settings of the CheesePi platform. The idea which is the basis for what this thesis explores is that some of these tasks, given the users consent, could be scheduled dynamically by a central server, with the client node uploading its results back to the server on task completion. This exchange with the server of scheduled tasks and their results would happen on a regular basis, for example daily or once per week.

3.3.2 The Scheduling Server

In order for any kind of measurement planning to occur there has to be communication between the client nodes and the entity doing the planning. The traditional Client-Server model is the easiest to implement, not only because it is conceptually easier but also for practical reasons. An alternative could be a decentralized network of clients communicating through Client-Client interaction. However such a solution would need to account for dynamically changing IP addresses, Network Address Translation (NAT) issues, and a host of other difficulties. Using a centralized server is a trade off where we choose a single point of failure in order to avoid having to deal with the inherent issues of implementing peer-to-peer communication. Having a centralized server also greatly simplifies planning of measurements since all gathered data is available in one place.

3.3.2.1 Client-Server Communication

The basic idea is that the Clients (i.e. the CheesePis) should contact the Server on a regular basis primarily to do one of three things:

Beacon	Report that the CheesePi is active and running, so that the server can keep track of active peers.
Get Tasks	Ask the server if it has any measurement tasks this CheesePi should perform during its next measurement cycle.
Upload Results	Upload the results of the measurements it has performed.

3.3.2.2 Server Structure

The Server is divided into three logical parts: the Control Server, the Upload Server and the Storage Server. These three parts may reside on the same physical machine, however they are intended to be logically decoupled in a way so that they do not have to be. The CheesePi clients only care and know about the Control Server and the Upload Server. The Storage Server is essentially just the database that the Upload Server and the Control Server interacts with.

The Upload Server only exists to receive dumps of result data that will be processed and put into the database of the Storage Server. The Control Server however is the main line of communication with the CheesePi clients. The communication is always Client initiated, where the Client on a regular basis (or whenever needed) contacts the Control Server to retrieve some information, such as scheduled tasks, and notify the Control Server of its existence. When the Client is done with a task scheduled by the Control Server, it uploads the results to the Upload Server.

3.3.2.3 Database Structure

The Server database's primary purpose is to store results of measurements performed by the client nodes. It also needs to keep track of the different client nodes that

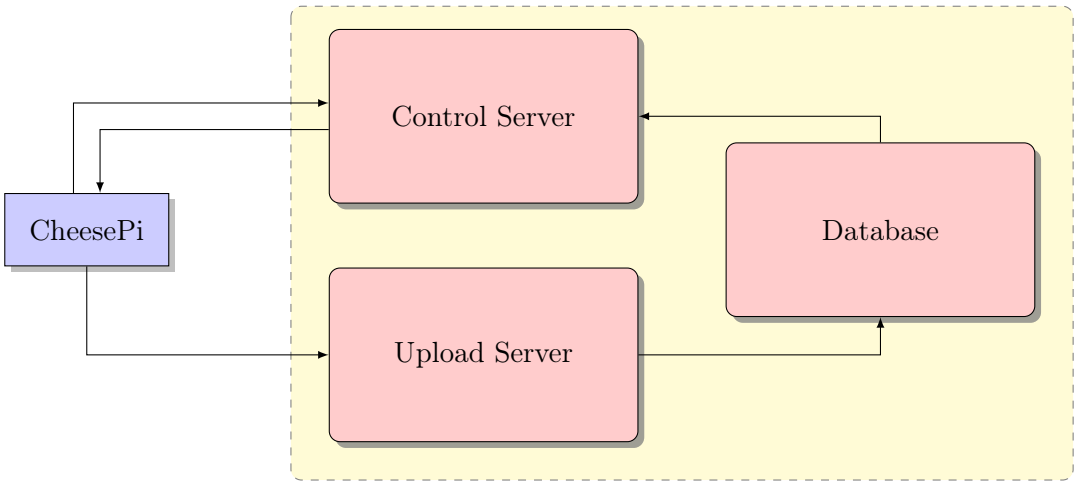


Figure 3.3: System diagram of the client and server interaction.

communicate with the server, so that it can know which result sets come from the same client node. Client nodes identify themselves using a random UUID[21] that is generated on first startup of the client node, so if a client node changes IP address the server will still recognize it.

Results that are uploaded to the server will be processed and put directly into the database. During processing a query is made to the database to see if there are previous statistics available for this specific path. Then a new set of statistics will be calculated, using previous values as starting points, and put into the database.

The database contains two document collections (similar to tables in SQL terms). One collection is simply to hold information about a peer, such as its UUID and last known ip address. The second collection holds documents with all the data gathered from the peers, such as raw results returned by the peer, and accumulated statistics for that peer.

The two collections could be merged into only one collection with all the data in it, which was the case in the initial implementation. However it was deemed an appropriate separation of concerns that one collection would only care about the identity and basic information of a peer. Furthermore the second collection should, for the same reasons, be split up into two collections. One collection with the results and a separate collection with the statistics.

Chapter 4

Design and Implementation

This chapter describes specific implementation solutions that might be of interest, and also the software environment used and any relevant aspects to it.

4.1 Simulation Framework

The simulation of a network of peers is done using a Python script. The script takes a file in JSON format as input which defines the whole setup for the simulation. The format of this file is described in Section 4.1.2.

In order for the simulation script to run the local machine must have the control server, upload server and the database running on it. The simulation script interacts with all these elements to mock the peers against the server code, and queries the database to gather data to be used for plotting results.

At first the script initialises all of the nodes in the mock network with the distribution models that will produce the mock measurement samples, as defined in the simulation definition file. The script then registers all of the nodes' UUIDs with the control server so that they are eligible for scheduling.

The script now enters the main loop, which simulates a number of iterations as defined in the simulation setup file. An iteration is a concept used in the simulation to denote time, one iteration is one quantised step for the simulation. For every such iteration, the script iterates over all the nodes in the simulated network and performs the following steps:

1. Ask the control server for targets that this node should perform measurements against.
2. Mock measurements for the given targets using the samples generated by the distribution models for those targets.
3. Package and upload the results to the upload server.
4. Fetch the now updated models for all the links this node has from the database.
5. Calculate some values from these new models, such as the change from the previous models, and save all the data in the script's local object for the node.

This way we keep a record of the state for each node at every iteration.

After all iterations are complete, the database is queried once more for the final states of the internal models. After which the data saved for all the nodes by the script are gathered and inserted into serializable structures which are then saved to disk using a Python module called pickle (see Section 4.3.1) for later plotting.

By storing the data to disk we separate data generation from data plotting, so that if any modification needs to be made to how the data is plotted the plotting code can be modified without having to worry about losing data generated by the simulation.

4.1.1 Designing Simulation Test Cases

In order for us to reason about how to make good decisions when scheduling measurements we need to analyse the behaviour of the metrics we will base those decisions on. In order to do so we will attempt to simulate different scenarios and observe how the metrics react to them.

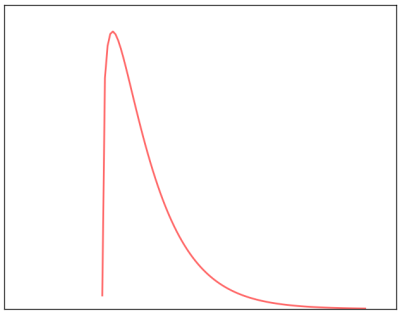


Figure 4.1: An example Gamma distribution

To generate test data that at least to some degree reflects the real world we need to pull our samples from a statistical distribution that mimics how real world results might look like. For ping measurements, typically there would be a hard minimum round trip time which is bound by physical constraints such as the medium through which the connection is made and the physical distance. Usually most samples would be concentrated close to this lower limit, with a slow drop off towards longer round trip times. This type of probability distribution can be modeled using a Gamma distribution or a Log-normal distribution. The Gamma distribution was chosen in this case, with an example shown in Figure 4.1. The choice was arbitrary, since the accuracy of the underlying distribution is not considered to be important for the purpose of this thesis. Furthermore, we cannot be sure every real world link follows the same exact distribution, however the Gamma distribution was considered to be a close enough approximation.

A Gamma distribution is primarily defined by two parameters, a ‘shape’ parameter and a ‘scale’ parameter. An important aspect for our purposes is that obtaining the mean and variance of a Gamma distribution is well defined. So if we know the parameters of a Gamma distribution it is easy to calculate the true mean and variance. Thus we can have a distribution where we know exactly what mean and variance it has, then generate samples from it, and finally compare how accurate our approximations of the mean and variance are.

4.1.2 Simulation Setup Format

In order to have reproducible simulations they are defined using JSON[22] files. The simulation definitions used in this thesis can be found in Appendix A. A simulation file defines a few global values for how it should run:

- `sample_size`: The amount of samples to fabricate for every measurement.
- `iterations`: How many iterations the simulation should run for.
- `schedule_size`: The size of the schedule, i.e. how may other peers a peer should "measure" against in one iteration.
- `schedule_method`: The method that should be used for scheduling. The accepted values are `random`, `roundrobin` and `smart`.
- `full_coverage_start`: Either `True` or `False`, depending on whether it is important that all the nodes start off with some history about each other. If it is set to true the first iteration will have every node measure every other node, so that some starting data is introduced into the system.
- `peers`: The list of peers that should be simulated. A peer object has two fields:
  - `uuid`: The UUID of the peer. This is needed since UUIDs are used by the server as a unique identifier.
  - `links`: A list defining the links to the other peers. A link object contains two fields:
    - `uuid`: The UUID of the end destination of the link.
    - `dist`: A list of distribution definitions that define how the samples for this link shall be generated. A distribution object contains a few optional parameters, if no parameters are given (an empty object), the default values will be chosen:
      - `shape`: The shape parameter of the Gamma Distribution (default = 2).
      - `scale`: The scale parameter of the Gamma Distribution (default = 3).
      - `loc`: The location parameter of the distribution, effectively shifts the entire distribution along the X-axis (default = 10).
      - `start_iter`: The iteration that this sample generator should start operating, effectively replacing the previously active distribution (default = 0).

The `start_iter` value of a distribution object allows us to define an arbitrary number of changes in the underlying distribution of a link by defining new ones with different parameters at different starting iterations.

4.2 Plotting

Plotting the data is done using the Python package matplotlib (Section 4.3.1). All the data generated from the tests were serialized and saved in binary format, and every simulation generated a lot of files. A GUI tool was built using the Tkinter (Section 4.3.1) package to aid in reading these binary files and plotting the data, and to configure the plots.

4.3 Software Environment

This section details some of the software used for the implementation of the scheduling system and the simulation framework.

4.3.1 Python Packages

To implement some aspects of the scheduling server for this thesis, specifically the client-server communication, the Python package Twisted[23] was used. Twisted implements an event-loop to enable concurrency through event-driven programming. It is mainly intended for network applications and used to implement servers. The actual communication protocol between the client and the server was implemented using txmsgpackrpc[24] which is a library that runs on top of Twisted and implements Remote Procedure Calls (RPC) using MessagePack[25] over TCP, SSL, UDP and UNIX sockets.

For the simulation framework SciPy[26] was used for implementations of statistical distributions, which were used to generate the sample data for the tests in this thesis (see Section 4.1.1). SciPy is a Python package that includes a huge amount of scientific tools for Python. SciPy builds on the NumPy[27] package which is the fundamental Python package for scientific computing.

To plot the results of the simulation tests the package matplotlib[28] was used. matplotplib is a versatile plotting library for Python which is the de-facto standard for plotting with Python. In order to save the simulation data to file before plotting it was serialized using the pickle[29] module, which is a standard Python module that can serialize and deserialize Python objects. In order to aid with plotting the data the TkInter[30] package was used to implement a simple interface for exploring the data. The TkInter package provides a thin layer on top of the open-source and cross-platform Tcl/Tk[31] GUI toolkit.

4.3.2 Database

In order to allow for rapid prototyping MongoDB[32] was chosen for the server database. MongoDB and InfluxDB (which is the main database on the client side) both store data as JSON, which makes it easier to write the glue for transferring data from a client database to the server database.

Chapter 5

Experimental Environment

This chapter describes in detail the setup used when running the simulations, including which metrics that are collected for the results, as well as the parameters for the actual simulations being run.

5.1 Hardware

All simulations are run on an Asus U36SD laptop running Linux, with 8GB of RAM and an Intel i5–2410M CPU clocked at 2.3 Ghz.

5.2 Metrics for Evaluation

Here we describe some metrics we will use to evaluate how the different approaches perform in relation to each other and our source of truth (i.e. the sample distributions).

- **Raw Values**
  The obvious and simple way is to plot the calculated mean and variance approximated by the models at each iteration, along with the true mean and variance, and see how they differ.
- **Delta Values**
  A potentially useful way to evaluate the stability of a model and to see how a value converges towards the true value is to plot the model's delta for each iteration, i.e. the difference between each approximated value and the previous approximation.
- **Aggregate Error of the Approximations**
  In order to better evaluate how the global scope is affected, and not only the link local scope, we can aggregate every link's difference between the calculated and true values of mean or variance at every iteration. We can then see how 'good' our *entire* network view is over time, to make sure that link local benefits

are not offset by inefficiencies when measuring other links. We will call this the *aggregate error* of the network.

5.3 Online Algorithms for Mean and Variance

For the first part of our results we want to investigate how the online algorithms described in Section 3.1.2.2 perform, in order to establish their usefulness as base for scheduling decisions.

For our simulations designed to evaluate the online algorithms we will consider a network with only two nodes. This case is useful because scheduling is not applicable, since there is always only one option to choose. Given that there is no choice involved in this setup we can also establish baseline values for `sample_size` and `iterations` for subsequent simulations and can evaluate the online algorithms for calculating mean and variance discussed in Section 3.1.2 in isolation.

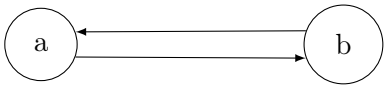


Figure 5.1: Two Peer Network

5.3.1 Uniformly Weighted Mean and Variance

The uniformly weighted online algorithm as described in Section 3.1.2.2 will be used when establishing the baselines for `sample_size` and `iterations` to see how the model converges for the different values. It is presumed that somewhere between 100 and 1000 iterations are needed to compare and identify the core characteristics of the two algorithms. Furthermore a sample size of 100 is assumed to be a reasonable value for a potential real world case, however we will test simulations with values of 10 and 1000 to see if it has any greater effect on the results.

5.3.2 Exponentially Weighted Mean and Variance

After establishing the values for `sample_size` and `iterations` to use in our simulations we will want to investigate how the exponentially weighted online algorithm as described in Section 3.1.2.2 compares to the uniformly weighted one and how to best choose the value of the α variable for our simulations.

5.3.3 Comparing Performance for Different Distribution Parameters

In order to arrive at a conclusion on which online algorithm to use as a basis for scheduling decisions we now want to start comparing their behaviour for different distributions. We want to see if the two algorithms are equivalent in how accurate the estimates are and how quickly they converge (if at all) towards the true values.

Simulations will be run with different values for the `shape`, `scale` parameters of the Gamma distribution (4.1) that is used to generate the fake samples to see how the two models behave under different circumstances.

Additionally, the Gamma distributions can also be shifted on the x-axis using the `loc` parameter in the simulation definition (see Section 4.1.2). However this will not be tested since the effect of this would only result in shifting the mean value by a constant amount, and so should have no impact on the algorithms as the variance remains the same.

5.3.4 Comparing Reaction to Change in Distribution Parameters

Building on the previous results we also want to see how the algorithms react if we change what the underlying distribution is for a given source of samples. To introduce change in the underlying distribution we will change the `shape`, `scale` or `loc` parameters of the Gamma distribution at a certain iteration in the simulation, and see how it affects the models calculated by the algorithms.

5.4 Improving Information Gain Through Scheduling

Now we turn to the main subject of this thesis, whether we can improve information gain through scheduling decisions based on our approximation models.

The reference for comparison will be to use round-robin scheduling because of two reasons. For one, round-robin scheduling is predictable and easily reproducible and therefore provides a stable point of reference, in contrast to for example random scheduling. Second it distributes attention evenly, and is the most naive method assuming there is no prioritising to be made.

For these simulations we introduce another node to form a network of three measurement peers. This network is interesting because it is the smallest one where a choice can be made about which link to measure. Every peer has two outgoing links, and so assuming a peer can only measure one link at a time, there is a binary choice.

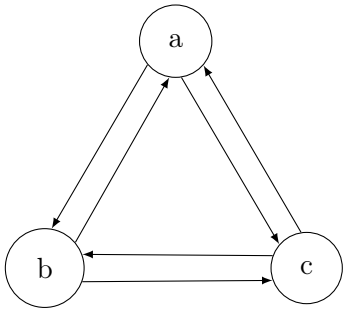


Figure 5.2: Three Peer Network

To evaluate the performance of the scheduler we will introduce change into the network by modifying the source distribution of one of the paths. The change will be in the `loc` parameter which shifts the mean along the x-axis, which in this case represents the simulated RTT delay in milliseconds. This is to simulate a service degradation that would increase the average RTT delay. Three scenarios will be evaluated, a small instant

change, a large instant change, and a small gradual change. They are listed below in Table 5.1.

Name	**Total change** (`loc`)	**Relative %**	**Change rate** (iterations)
Small	$10 \rightarrow 20$	100	1
Large	$10 \rightarrow 100$	900	1
Gradual	$10 \rightarrow 20$	100	45 (5 per increment of 1)

Table 5.1: Test scenario definitions for evaluating measurement scheduling. The total change refers to the `loc` parameter of the source gamma distribution for one path in the network. The change rate is the amount of iterations it takes for the entire change to take effect.

5.4.1 Three Peer Simulation with Round-Robin scheduling

First we will run a static simulation where nothing changes to see how the round-robin approach affects the results of the uniform and exponential algorithms, as well as the aggregate error of the network. After we have a baseline for the aggregate error for this setup we will introduce a change to one of the links during the simulation, and see how this affects the aggregate error graph.

5.4.2 ADWIN1-Based Scheduling Using Three Peers

Once we know how round-robin scheduling behaves we will attempt to schedule the measurements using the ADWIN1 algorithm as described in Section 3.2.2.1. ADWIN1 will manage a sliding window of the variance of each network path as calculated by the exponentially weighted online algorithm described in Section 3.1.2.2. The maximum length of the sliding window is arbitrarily chosen to be 20.

The tests will be simulating a change in one of the links in the three peer network to see if we can improve the performance when compared to round-robin scheduling. The three test cases defined in Table 5.1 will be used for simulating the change in the network and will be evaluated separately.

Simulations will be run with different δ-values for the ADWIN1 algorithm. For each case the raw values of the approximated variance using the exponential approximation model will be plotted in two plots. The first plot comparing the round-robin scheduling against ADWIN1 scheduling using $\delta = 10^{-1}$. The second plot will compare the ADWIN1 simulation with $\delta = 10^{-1}$ against a simulation where $\delta = 10^{-50}$ in order to see how the scheduling is affected when adjusting the confidence value.

The aggregate error of the network will also be plotted for every test case comparing the error for round-robin scheduling and ADWIN1 using $\delta = 10^{-1}$.

5.4.3 ADWIN1-Based Scheduling Using Ten Peers

The same tests that was done for the three peer network will be performed using a network of ten simulated peers. In this case each node will get to choose three other peers to measure against. This is done in order to verify that any benefits through scheduling also applies to a bigger pool of possible choices in each decision.

A fourth case will also be considered where we run a simulation where multiple simultaneous changes occur in the network, one small, one large and one gradual, to see what happens when multiple links need extra measurement attention. The three changes listed are the same ones as defined in Table 5.1.

Chapter 6

Results

In this chapter the results from the simulation experiments described in Chapter 5 are presented.

6.1 Simulation Performance

The time complexity of the simulation framework is exponential to the number of network nodes that it simulates. Therefore the performance rapidly decrease when increasing the amount of nodes. This can be improved slightly by lowering the number of iterations and number of samples per iteration when increasing the number of nodes to simulate.

Running a simulation with 50 nodes, 100 iterations, 10 samples per iteration and a schedule size of 1 would take on average 4 minutes to complete, whereas a simulation with the same number of nodes but 500 iterations and 100 samples per iteration would take up to 30 minutes.

6.2 Online Algorithms for Mean and Variance

In this section we present the results for the tests described in Section 5.3 which compare the two algorithms described in Section 3.1.2.2 for calculating online mean and variance.

6.2.1 Uniformly Weighted Mean and Variance

First we plot the gamma distributions probability density function along with a histogram of the samples generated to verify that the data is correctly generated. We can also see that the mean calculated by the uniformly weighted algorithm 3.1.2.2 is very close to the actual mean. This test was run using 100 iterations, where for every iteration 100 samples were generated for every simulated path.

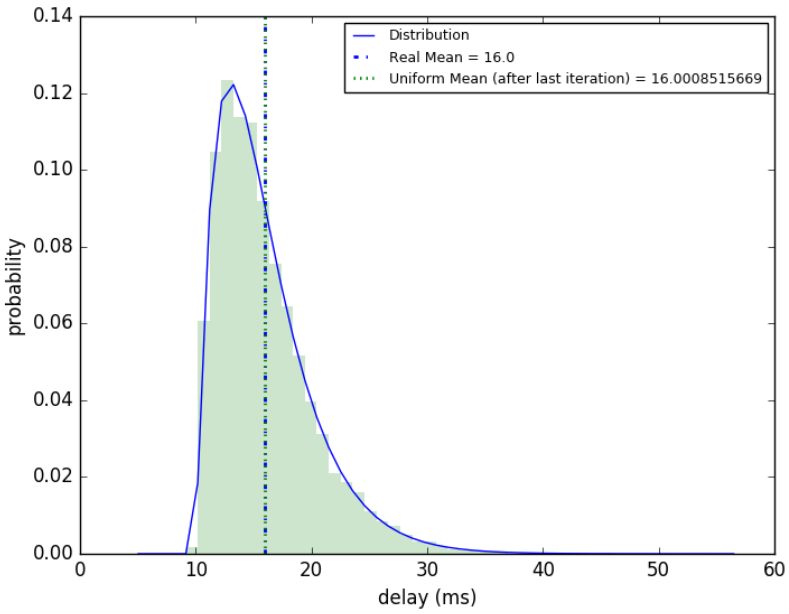
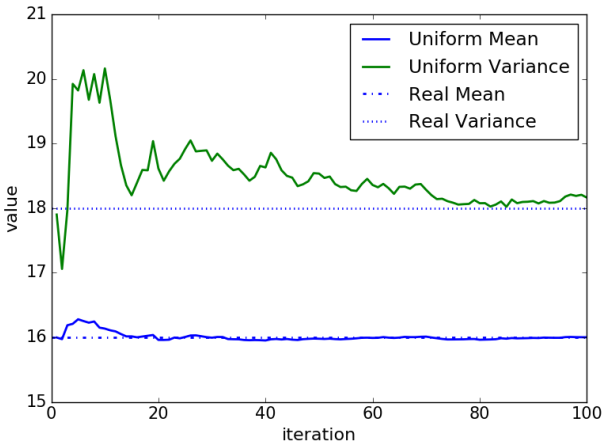
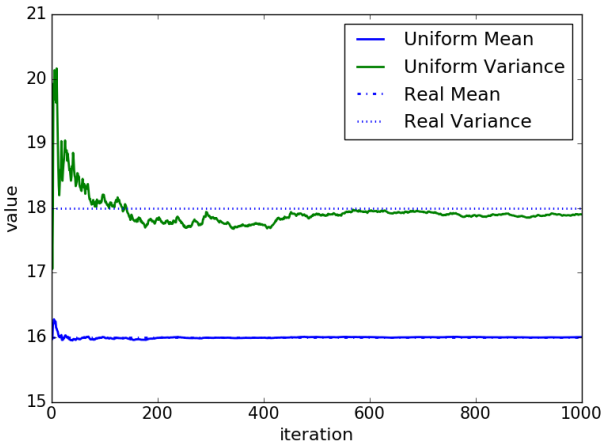


Figure 6.1: Gamma distribution and histogram of generated samples



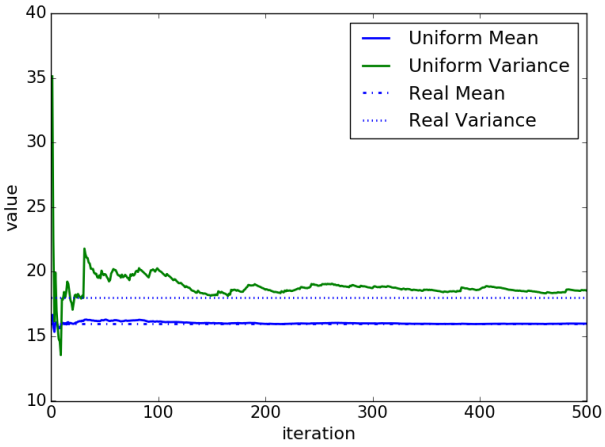
(a) 100 Iterations



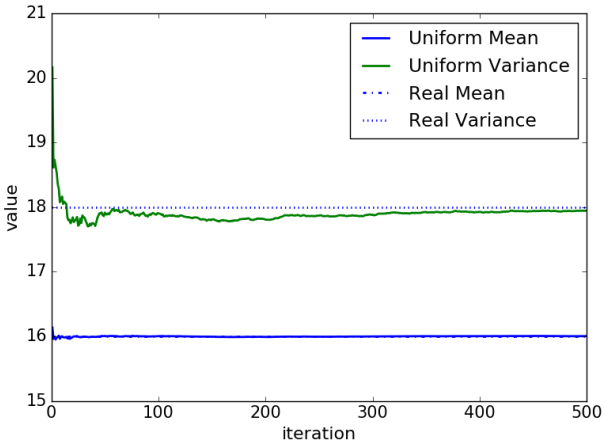
(b) 1000 Iterations

Figure 6.2: Convergence of the approximated mean and variance calculated using the uniformly weighted online algorithm (3.1) for 100 iterations and 1000 iterations.

The graphs in Figure 6.2 show how the mean and variance values changes when new samples are added when using the algorithm with uniform weighting to calculate them. The most drastic change happens within the first 100 iterations (10,000 samples in total). There is some adjustments affecting the variance after that, however looking at Figure 6.2b we can conclude that it is pretty much stable after around 500 iterations.



(a) 10 Samples



(b) 1000 Samples

Figure 6.3: Convergence of the approximated mean and variance calculated using the uniformly weighted online algorithm (3.1) for 10 and 1000 samples per iteration.

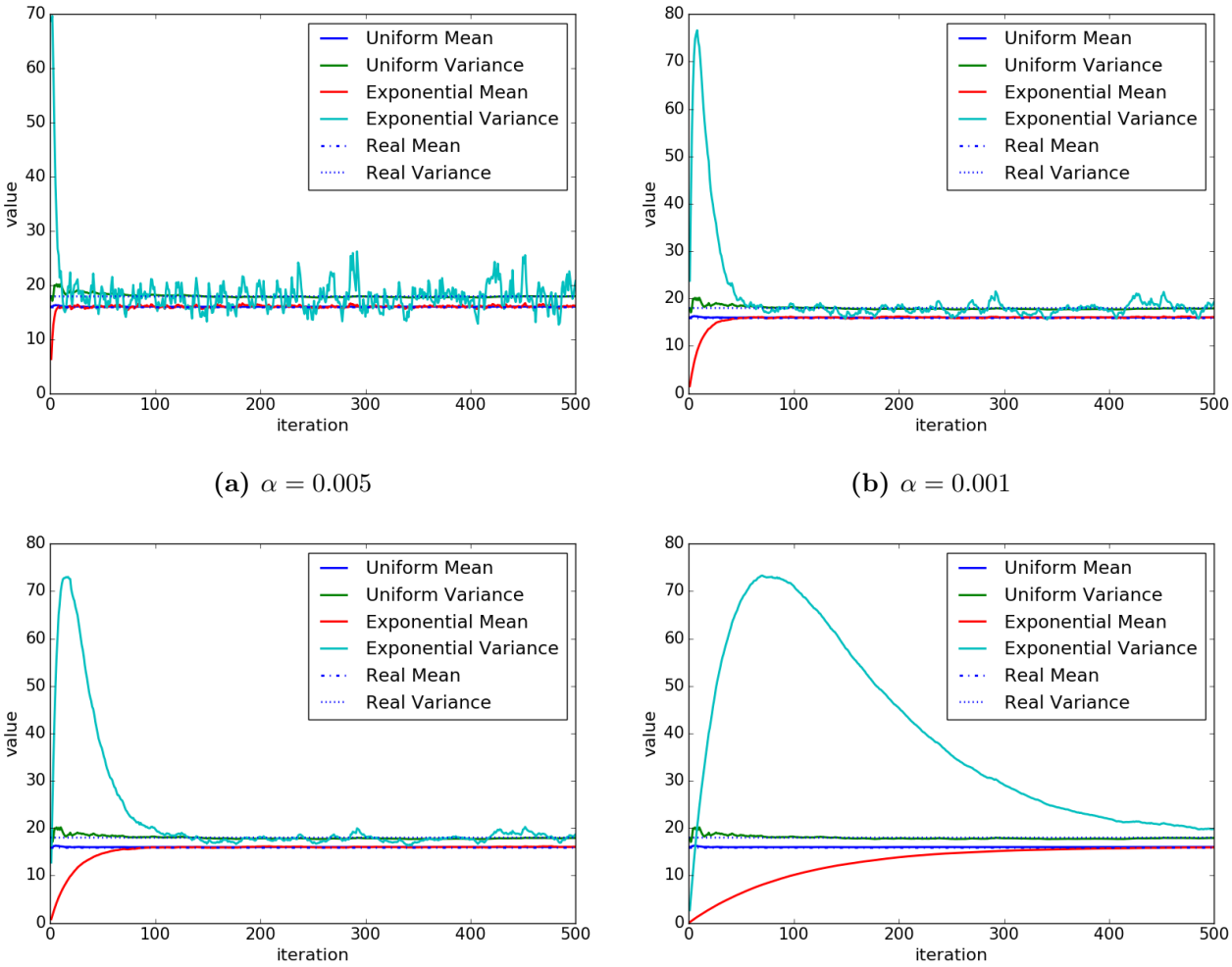
In Figure 6.3 we can see how different values for `sample_size` affect the results. With 10 samples we see a larger fluctuation and much less long term stability

compared to larger sample sizes. A `sample_size` value of 1000 predictably provides greater stability and better convergence toward the actual value. However a `sample_size` of 100, as used in Figure 6.2, is a more plausible number for a real world measurement task, and seems to be adequate in showing the behaviour of the algorithm based on the samples.

Note: *From now on all the simulations will be run using 500 iterations and 100 samples per mock measurement as established in this section, unless otherwise specified.*

6.2.2 Exponentially Weighted Mean and Variance

Now we want to compare how the exponentially weighted algorithm 3.2 compares to the uniformly weighted one. We start by testing how the value of α impacts the results.



(a) $\alpha = 0.005$

(b) $\alpha = 0.001$

(c) $\alpha = 0.0005$

(d) $\alpha = 0.0001$

Figure 6.4: Comparison of the convergence between the uniformly weighted (3.1) and the exponentially weighted (3.2) algorithms for online mean and variance, for different values of α in the exponential algorithm.

We can see in Figure 6.4 that the α value determines how much of an impact new values have on the estimation. We can also see that the uniform algorithm performs better in terms of closeness of the approximation to the actual value, as well as stability. Unsurprisingly, when tweaking the α value of the exponential algorithm there is a trade-off between warm-up, i.e. how fast an equilibrium is found, and stability. For this particular test it seems like a good α is somewhere around 0.001 and 0.0005. From now on $\alpha = 0.001$ will be used unless otherwise specified.

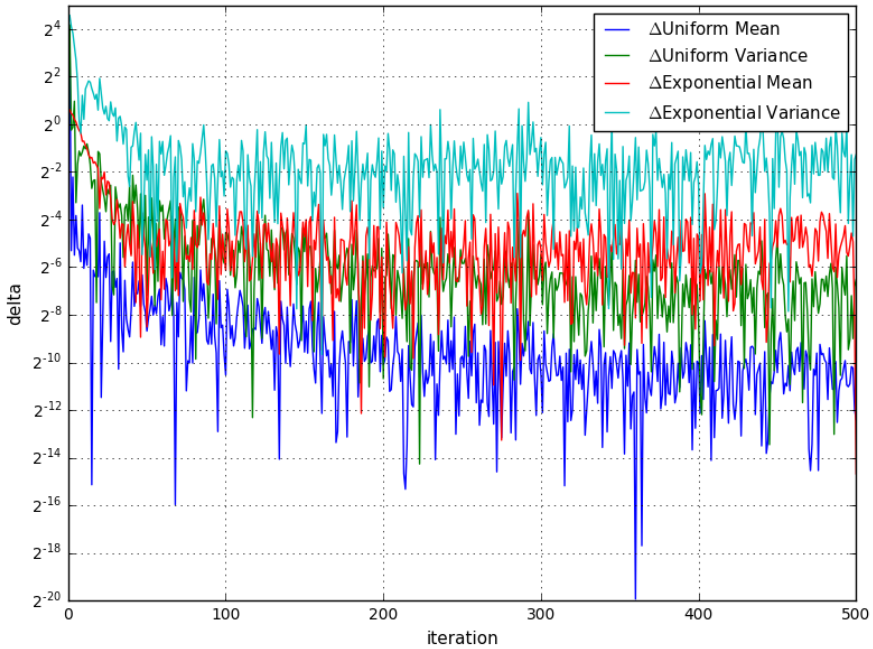


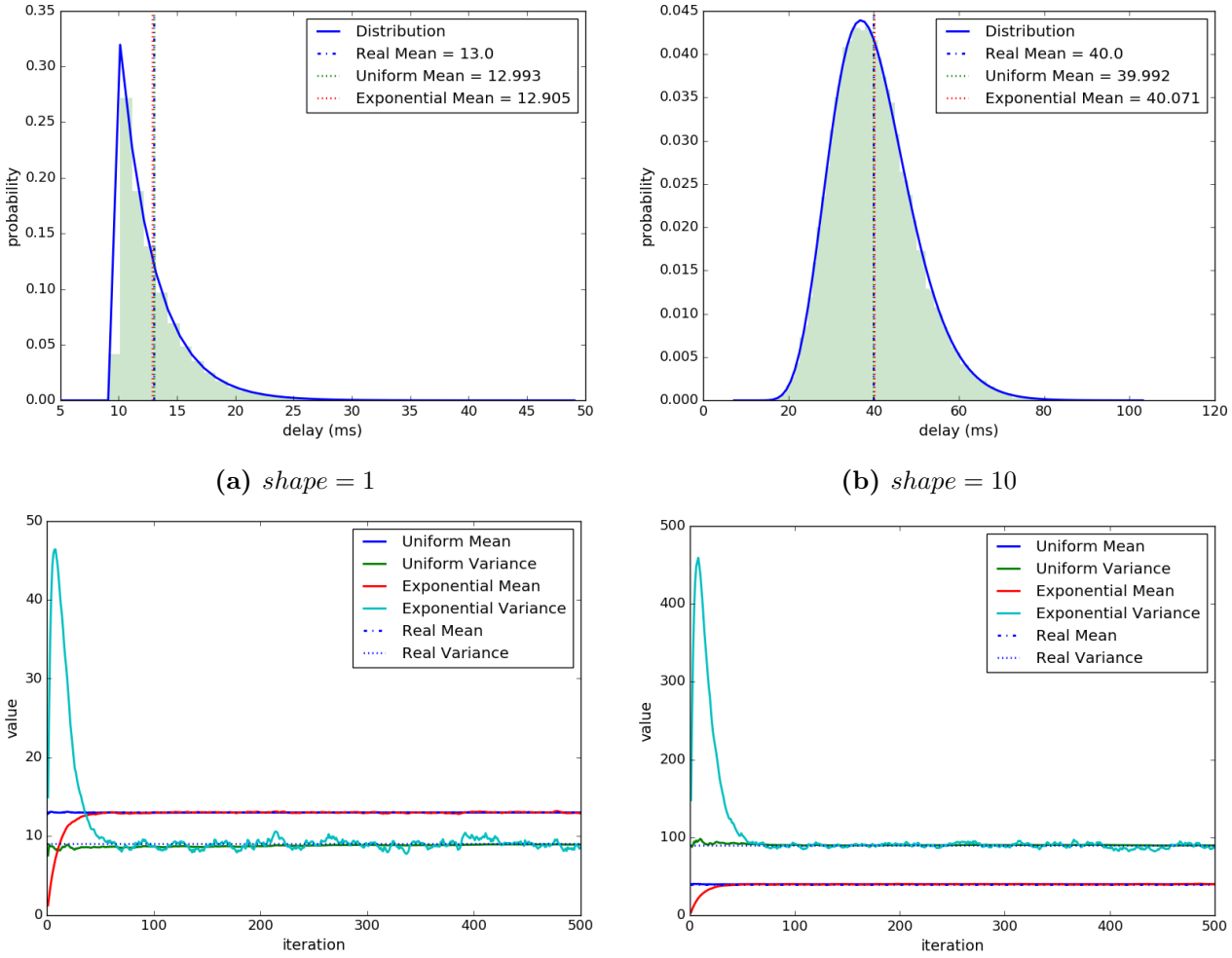
Figure 6.5: The delta of the mean and variance values between each iteration and the previous for the uniform and exponential online algorithms.

In Figure 6.5 the delta values between each iteration, i.e. the difference between the previous value and the new, have been plotted on a logarithmic scale. We can see that the convergence pattern is pretty similar, though offset from each other. However, if we look carefully the uniform algorithm converges in a slightly more curved fashion, where the exponential algorithm remains pretty linear after the warm-up period if you disregard the noise.

6.2.3 Comparing Performance for Different Distribution Parameters

In Figure 6.6 we can see how the *shape* parameter affects the distribution. A lower *shape* value skews the probability density function to the left, and the higher the *shape* value gets the closer it gets to a normal distribution. From the plots of the values we can observe that the two algorithms aren't much affected by changes in

the skewness of the source distribution. The variance is slightly more stable when the distribution is not skewed.

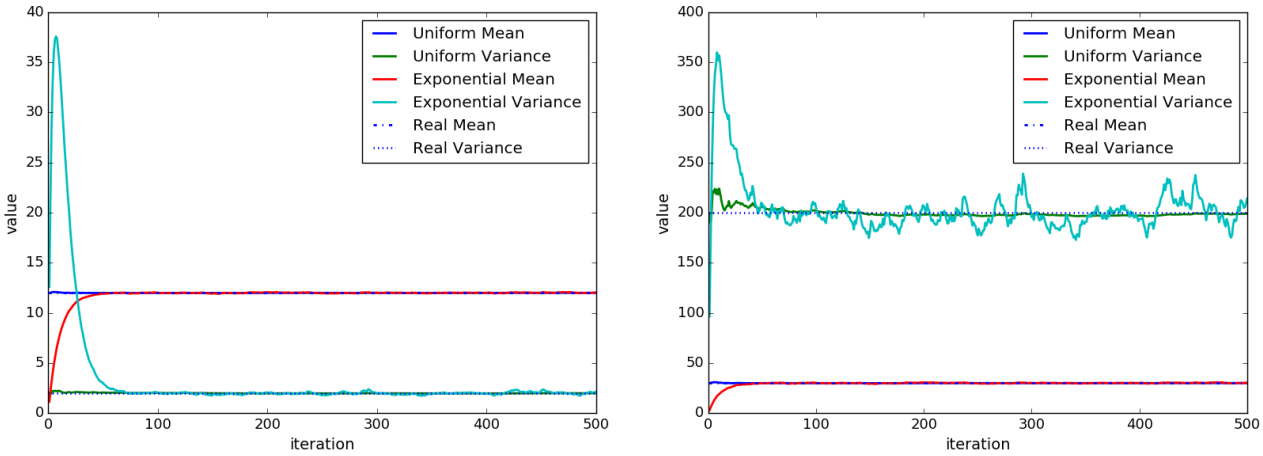


(a) $shape = 1$ **(b)** $shape = 10$

(c) $shape = 1$ **(d)** $shape = 10$

Figure 6.6: Gamma distributions with different values for the *shape* parameter and how the online algorithms for mean and variance converge on the true values for those distributions. $\alpha = 0.001$ for the exponential algorithm.

In contrast to the *shape* parameter, changing the *scale* parameter affects the results to a much greater extent, as seen in Figure 6.7. The *scale* parameter of a Gamma distribution effectively scales the distribution's probability density function on the x-axis, so a larger value of the *scale* parameter stretches out the distribution upwards the x-axis. It is therefore not strange that the values are affected in this way. There is a larger window of possible values, which makes it harder to establish a good estimate for the boundaries (which in a sense is what variance is).

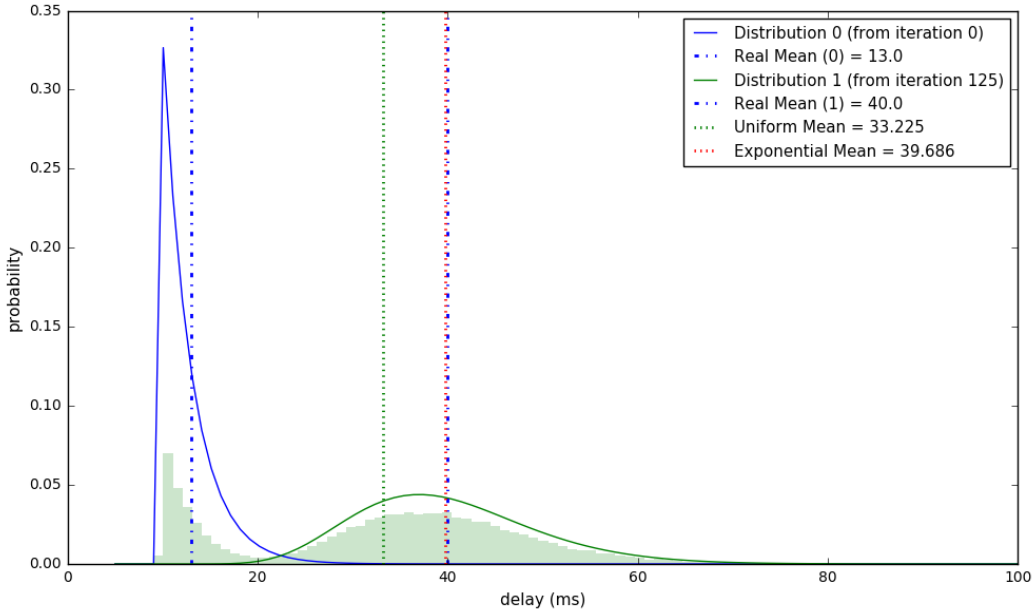


(a) $scale = 1$ **(b)** $scale = 10$

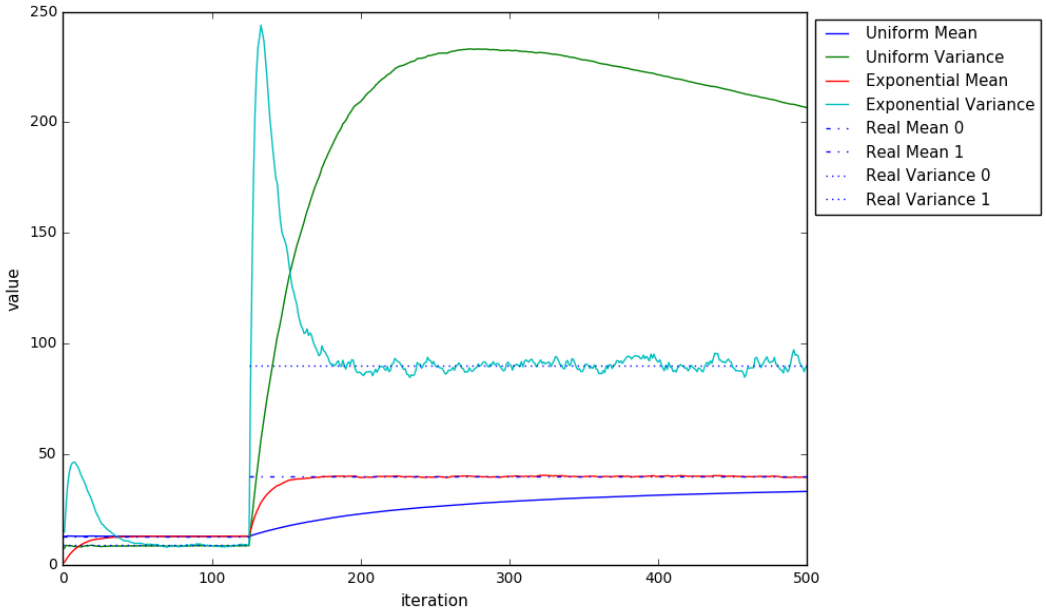
Figure 6.7: Convergence of the online algorithms for gamma distributions with different values for the *scale* parameter. $\alpha = 0.001$ for the exponential algorithm.

6.2.4 Comparing Reaction to Change in Distribution Parameters

In Figure 6.8 we see the reaction when the *shape* parameter is changed from a value of 1 to 10. Here we start to see the strength in the exponentially weighted algorithm. Even though it has a longer warm-up period and less accuracy than the uniformly weighted algorithm it is able to adapt a lot quicker to changes in the data. The uniformly weighted algorithm however completely breaks down and from what we can see it needs significantly more than twice the amount of data from the new source configuration compared to the old one in order to stabilize on the new values.



(a) Distribution with histogram

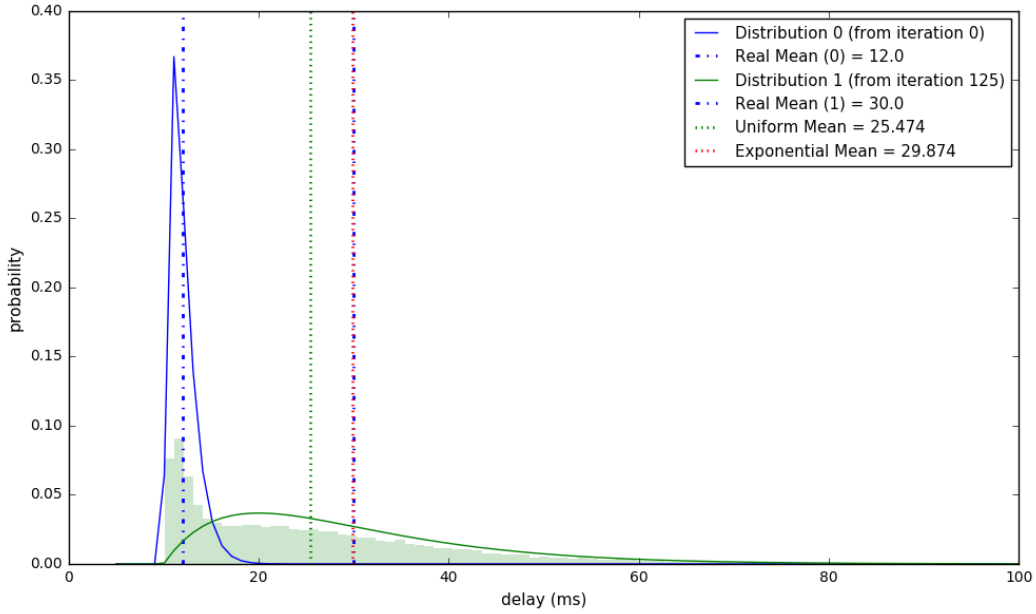


(b) Values

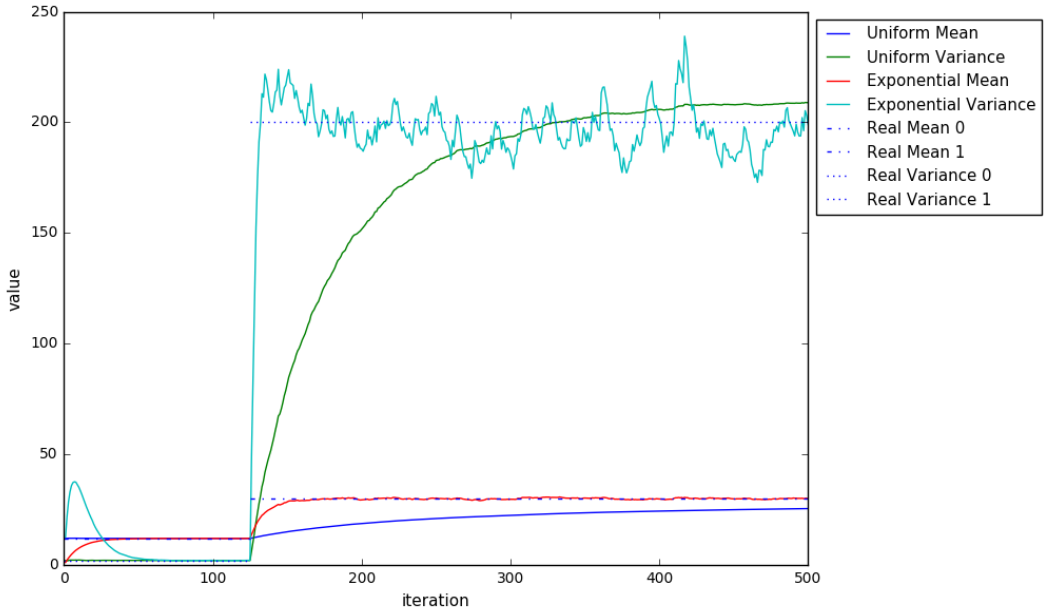
Figure 6.8: Distribution and convergence of the online algorithms for a gamma distribution where the *shape* parameter is changed $1 \rightarrow 10$ at iteration 125. $\alpha = 0.001$ for the exponential algorithm.

When we change the *scale* parameter of the distribution (which effectively stretches out the distribution over the x-axis) we see behaviour in the two algorithms similar

to when changing the *shape* parameter, as can be seen in Figure 6.9. Interestingly since the variance changes so significantly we can spot the increased noise in the exponential algorithm when estimating the variance, although it is still a lot quicker to respond to the change than the uniformly weighted algorithm.



(a) Distribution with histogram

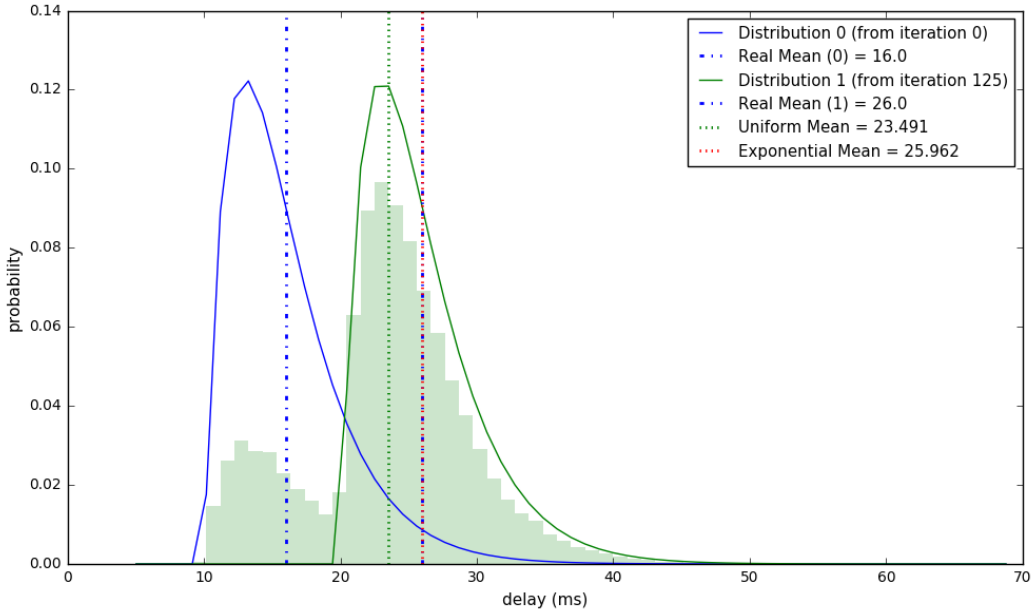


(b) Values

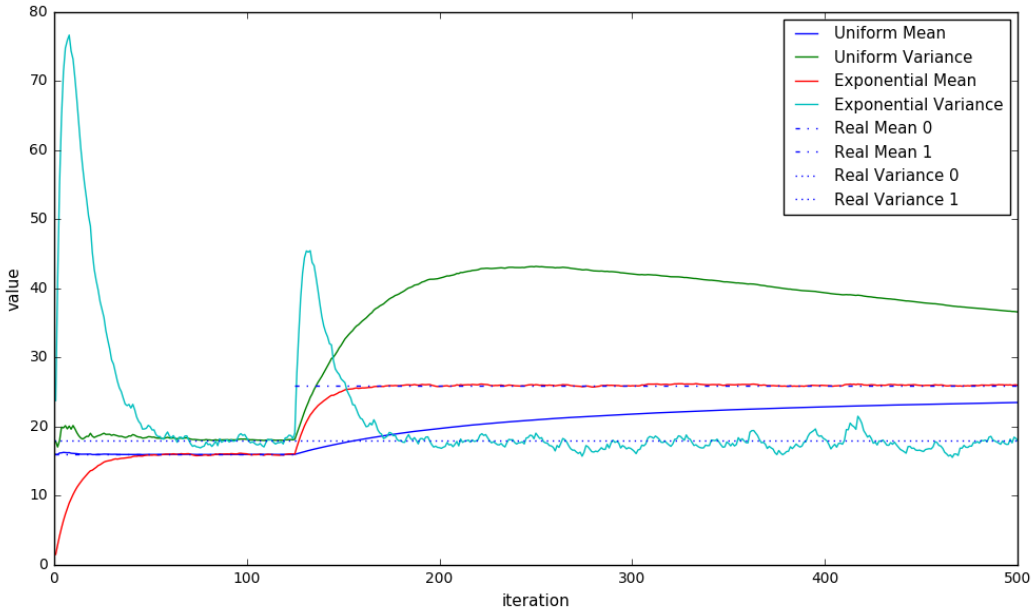
Figure 6.9: Distribution and convergence of the online algorithms for a gamma distribution where the *scale* parameter is changed $1 \rightarrow 10$ at iteration 125. $\alpha = 0.001$ for the exponential algorithm.

We can see in Figure 6.10b where the *loc* parameter is changed how the algorithms react much in the same way as when changing the *shape* and *scale* param-

eters, the exponential algorithm is quick to adapt and arrive at an estimate of the new value while the uniform algorithm creeps ever so slowly towards its new goal. What's interesting to note is that even though the variance does not change we see a noticeable spike in the exponential estimate of the variance, this could be helpful when trying to detect when a change occurs.



(a) Distribution



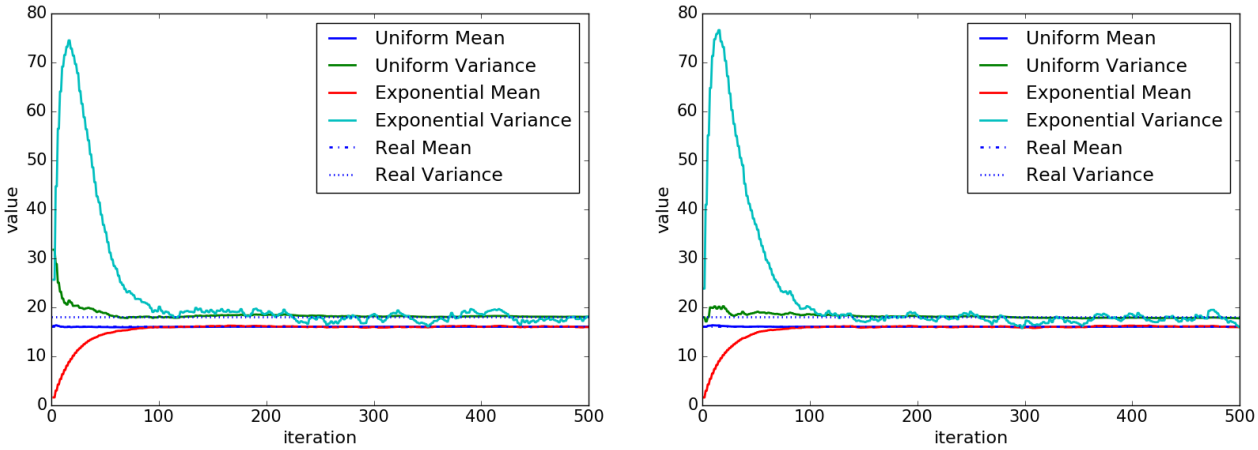
(b) Values

Figure 6.10: Distribution and convergence of the online algorithms for a gamma distribution where the *loc* parameter is changed $10 \rightarrow 20$ at iteration 125. $\alpha = 0.001$ for the exponential algorithm.

6.3 Improving Information Gain Through Scheduling

This section presents the results for the simulation tests evaluating different measurement scheduling methods as described in Section 5.4.

6.3.1 Three Peer Simulation with Round-Robin Scheduling



(a) Path 1 **(b)** Path 2

Figure 6.11: Convergence of the online algorithms for two simulated links from one peer in a three peer network using round-robin scheduling where the schedule size is 1.

In Figure 6.11 we see the two outgoing links from one peer when using round-robin scheduling. As we can see they receive equal amounts of attention in terms of measurements having been scheduled.

We can see in Figure 6.12 an example of how the aggregate error can look like, as described in Section 5.2. In this particular scenario all of the six paths have the same distribution, they are however initialized with different seeds for the random number generator.

In Figure 6.13 we have introduced a small change to one of the paths (as defined in Table 5.1), and we can clearly see how this affects the global aggregate error. The effect of the change on the estimates for the two algorithms is similar to what we've already seen in Section 6.2.

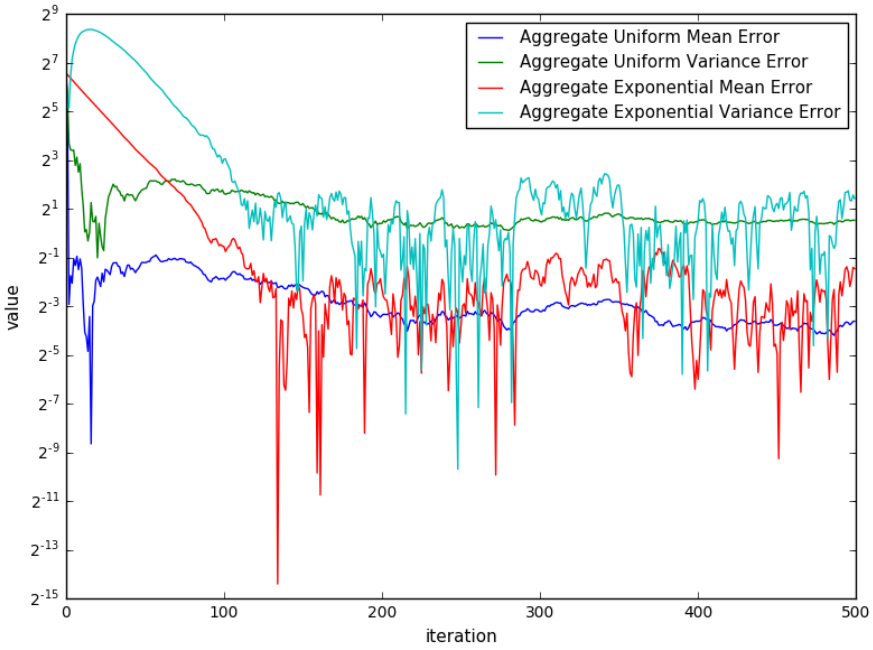


Figure 6.12: Aggregate error of the online algorithms for a simulated three node network where all paths have the same distribution.

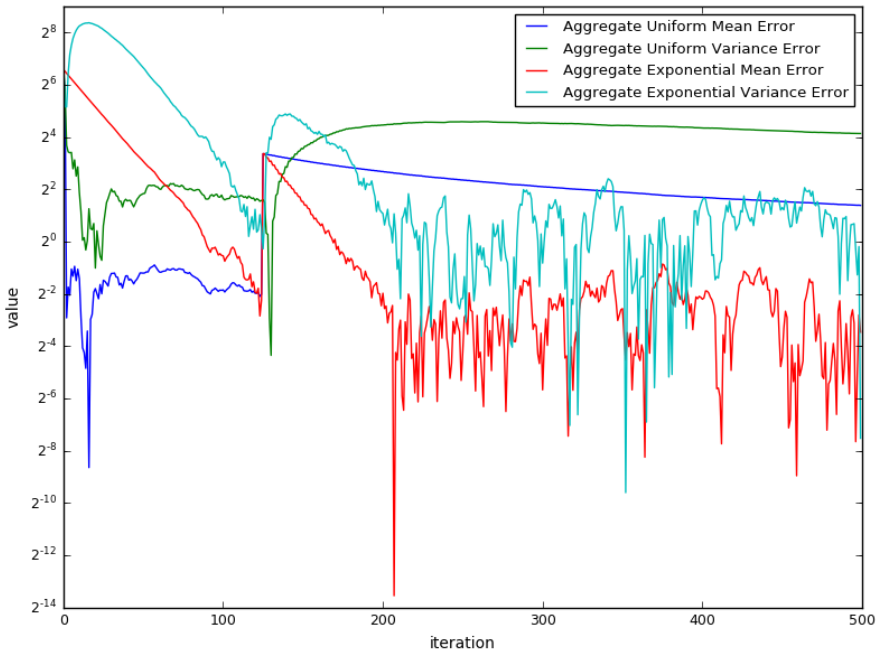


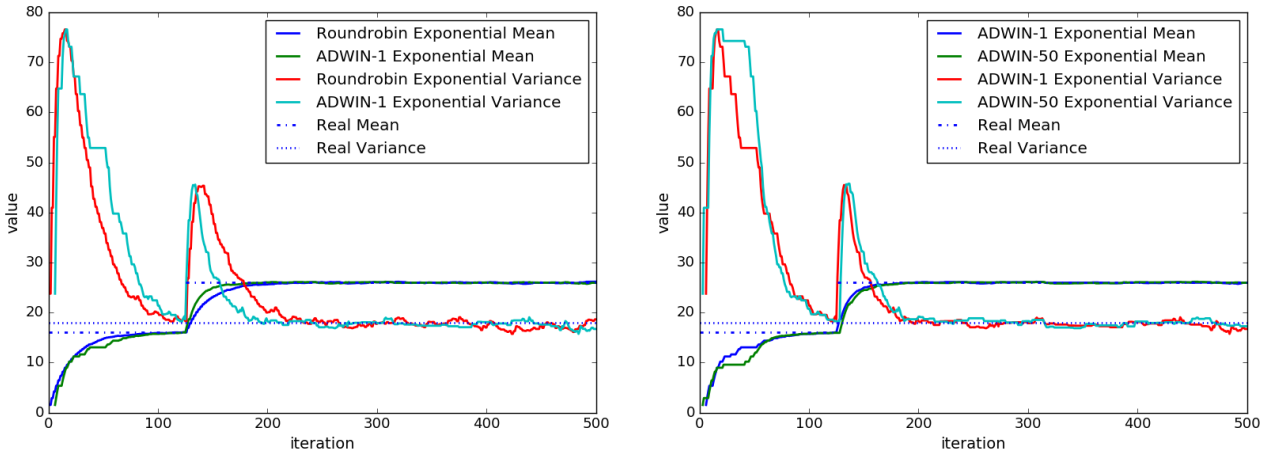
Figure 6.13: Aggregate error of the online algorithms for a simulated three node network where one path changes its *loc* parameter $10 \rightarrow 20$ at iteration 125.

6.3.2 ADWIN1-Based Scheduling Using Three Peers

In this section the results from the tests described in Section 5.4.2 are presented, where the ADWIN1-based scheduling method is compared to roundrobin scheduling for a network of three peers. The changes introduced into the network to evaluate the scheduling are the small, large and gradual changes defined in Table 5.1.

6.3.2.1 Small Change

In Figure 6.14a we can see that using ADWIN1 to determine the schedule improves the recovery time when compared to using round-robin scheduling. Worth noting is that ADWIN1 actually has worse performance in the beginning before getting the approximation model to a stable state. In Figure 6.14b we see that using a smaller δ makes the scheduling slightly more cautious before reacting to change, but still better than the round-robin case.



(a) Roundrobin and ADWIN where $\delta = 10^{-1}$.

(b) ADWIN where $\delta = 10^{-1}$ and $\delta = 10^{-50}$.

Figure 6.14: Mean and variance values for an exponentially weighted model of a single link where a small change occurs in a three node network.

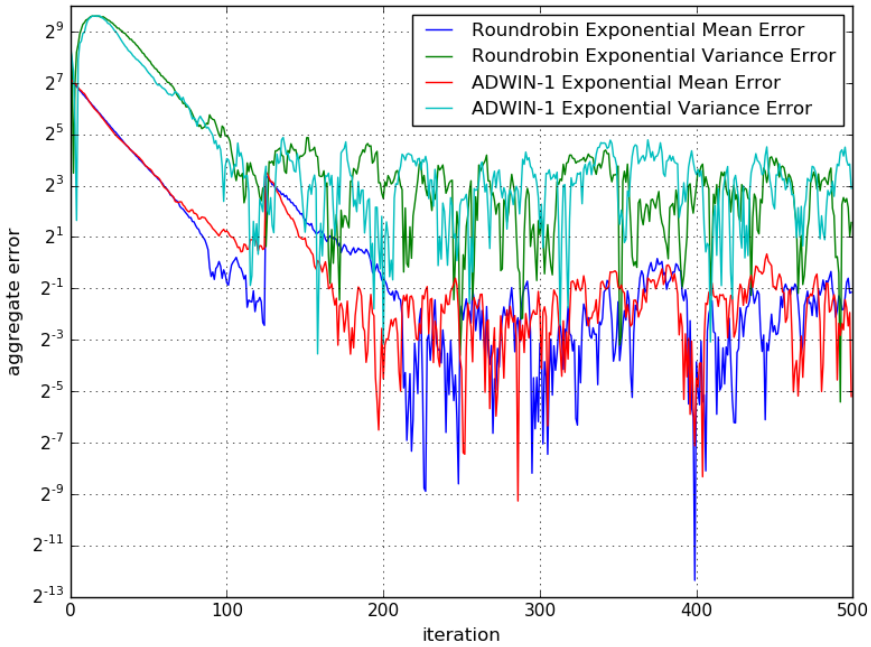
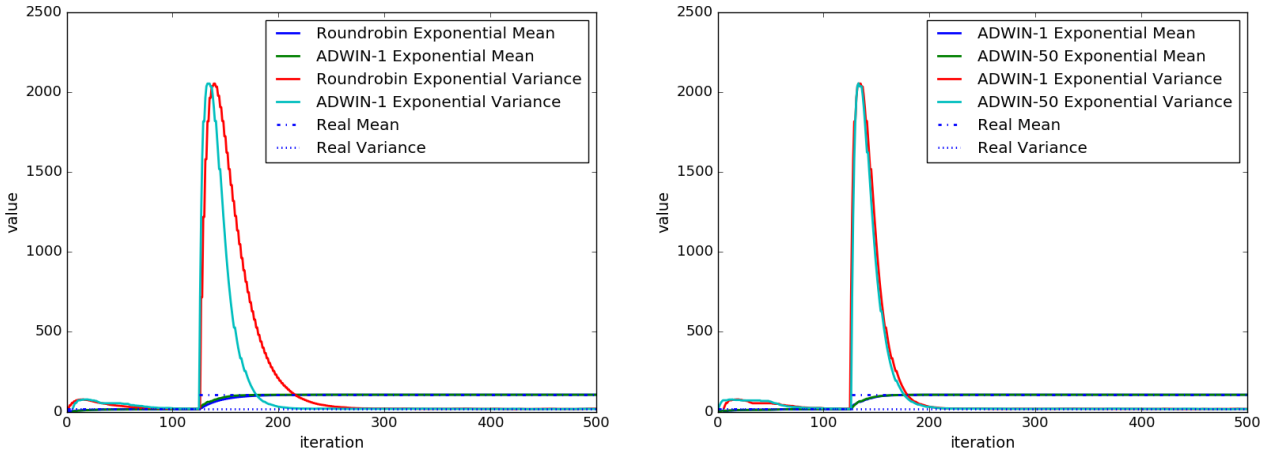


Figure 6.15: The aggregate error for a three node network where a small change occur when using roundrobin scheduling and scheduling based on ADWIN where $\delta = 10^{-1}$.

6.3.2.2 Large Change

We can see in Figures 6.16a and 6.16b that similar results can be observed for a large change as for a small change. In Figure 6.17 we can also see the delta values for the simulation corresponding to Figure 6.16a, i.e. the difference between the current approximation and the previous over time. We can observe that the ADWIN1 scheme reaches a more stable state, keeping steady beneath the 2^{-3} line once stable. In Figure 6.18 we can see a slight advantage in the overall aggregate error of the approximations over the simulated network, though at times it does favour measuring the changed link to the detriment of the other links.



(a) Roundrobin and ADWIN where $\delta = 10^{-1}$.

(b) ADWIN where $\delta = 10^{-1}$ and $\delta = 10^{-50}$.

Figure 6.16: Mean and variance values for an exponentially weighted model of a single link where a large change occurs in a three node network.

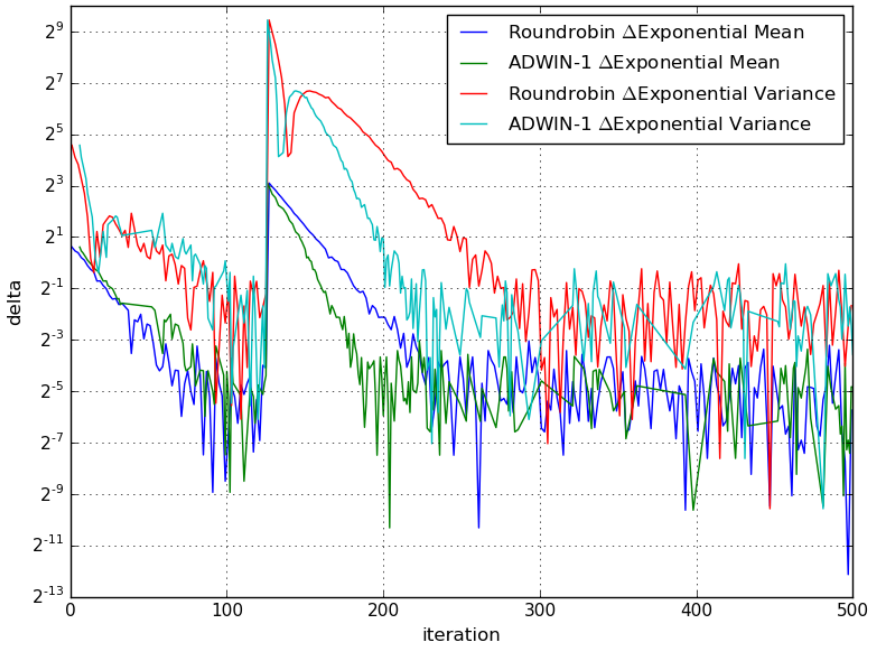


Figure 6.17: Deltas for an exponentially weighted model of a single link where a large change occurs in a three node network using roundrobin scheduling and scheduling based on ADWIN where $\delta = 10^{-1}$.

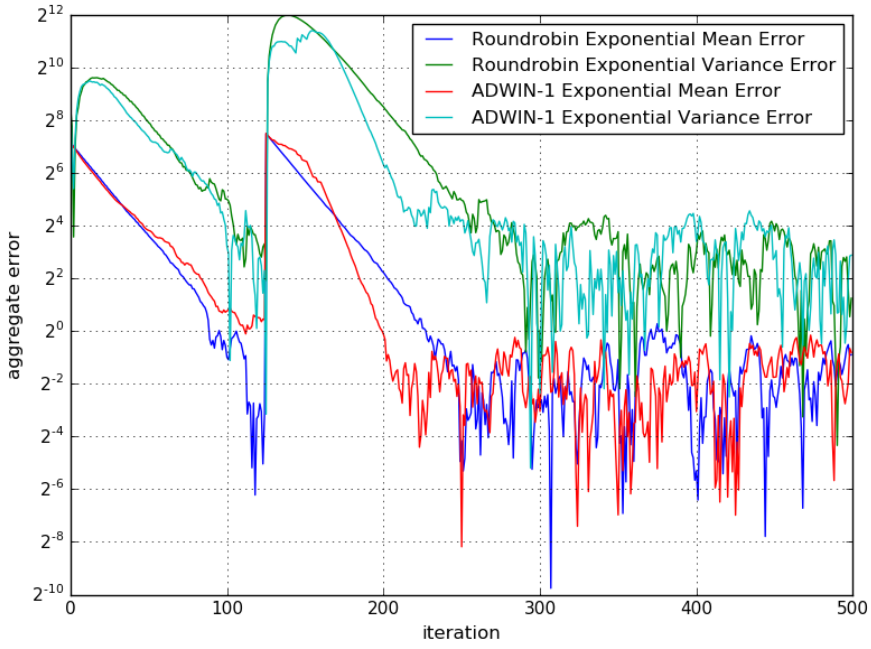
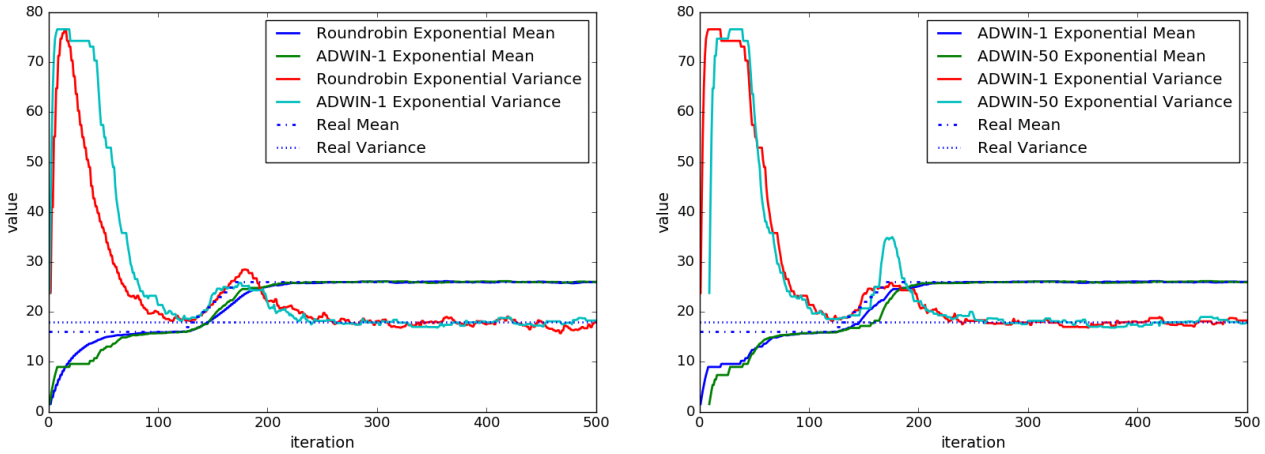


Figure 6.18: The aggregate error for a three node network where a large change occur when using roundrobin scheduling and scheduling based on ADWIN where $\delta = 10^{-1}$.

6.3.2.3 Gradual Change

In Figure 6.19a we can see that the ADWIN1 scheduler has a slight advantage when dealing with a gradual change. Figure 6.19b shows that using a δ value of 10^{-50} only serves to hinder the scheduler since it needs a higher confidence that a change has occurred before prioritising a link. The aggregate error in Figure 6.20 is inconclusive and doesn't show any significant improvement for ADWIN1 over roundrobin based scheduling.



(a) Roundrobin and ADWIN where $\delta = 10^{-1}$.

(b) ADWIN where $\delta = 10^{-1}$ and $\delta = 10^{-50}$.

Figure 6.19: Mean and variance values for an exponentially weighted model of a single link where a gradual change occurs in a three node network.

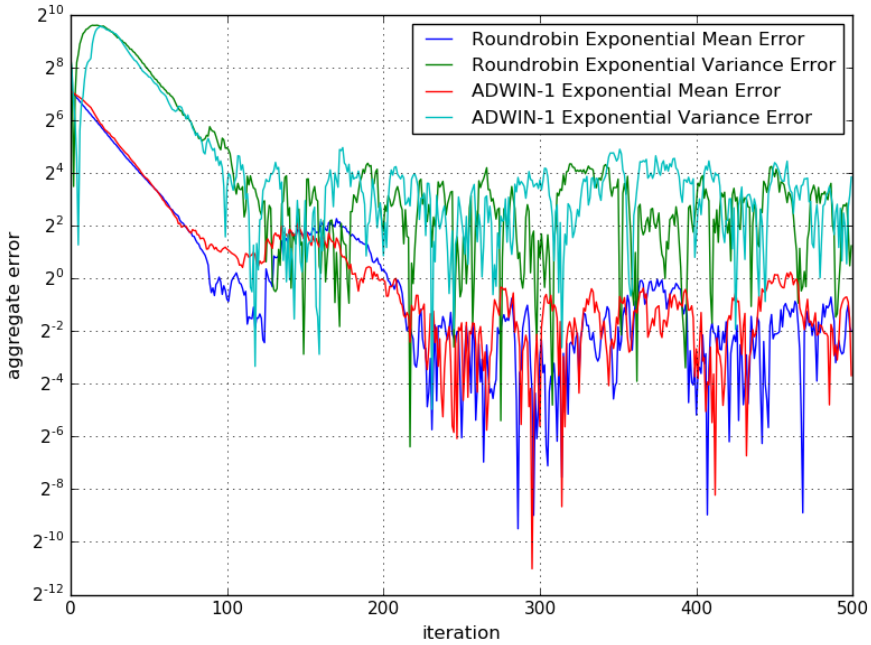


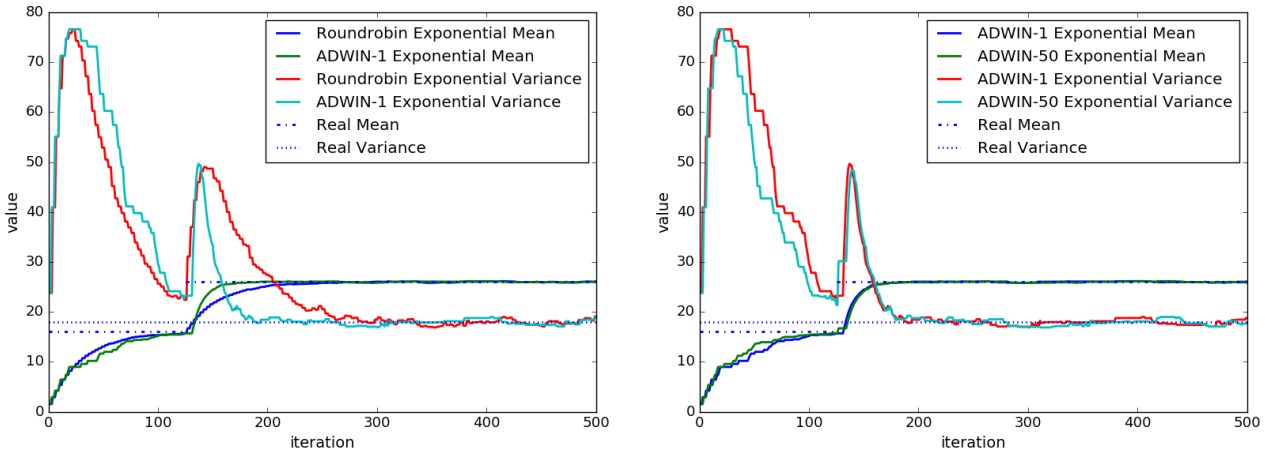
Figure 6.20: The aggregate error for a three node network where a gradual change occur when using roundrobin scheduling and scheduling based on ADWIN where $\delta = 10^{-1}$.

6.3.3 ADWIN1-Based Scheduling Using Ten Peers

In this section the results from the tests described in Section 5.4.3 are presented, where the ADWIN1-based scheduling method is compared to roundrobin scheduling for a network of ten peers. The changes introduced into the network to evaluate the scheduling are the small, large and gradual changes defined in Table 5.1.

6.3.3.1 Small Change

In Figure 6.21a we can see that ADWIN1 still has an advantage over roundrobin when increasing the size of the network for a small change in one link. However in Figure 6.22 we can see that the aggregate error over the entire network is inconclusive.



(a) Roundrobin and ADWIN where $\delta = 10^{-1}$.

(b) ADWIN where $\delta = 10^{-1}$ and $\delta = 10^{-50}$.

Figure 6.21: Mean and variance values for an exponentially weighted model of a single link where a small change occurs in a ten node network.

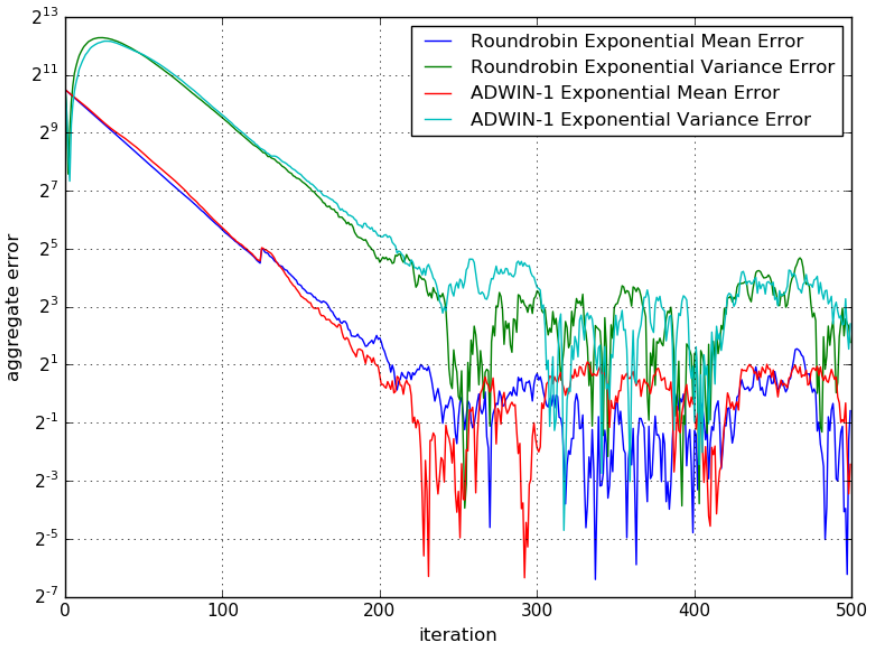
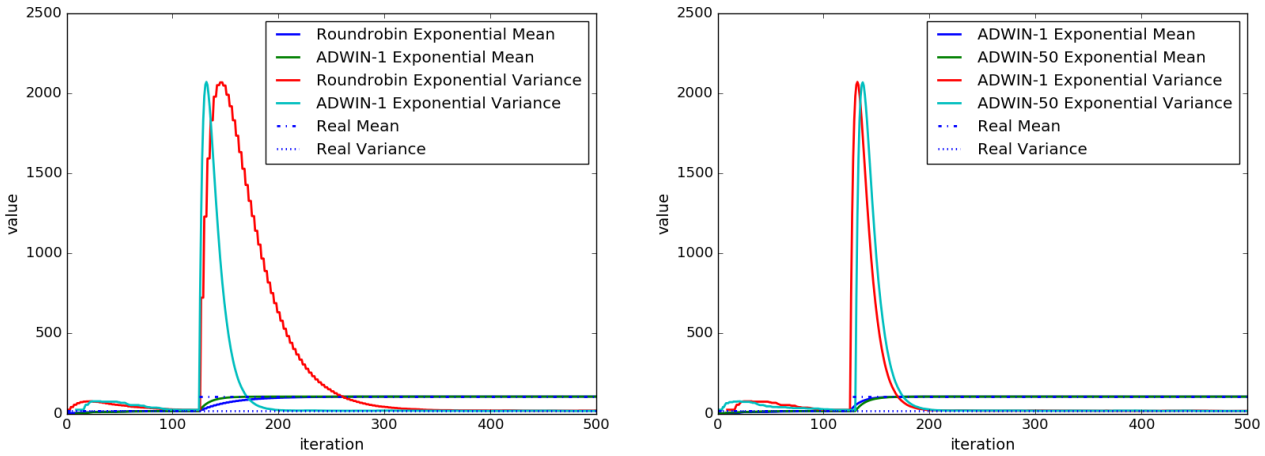


Figure 6.22: The aggregate error for a ten node network where a small change occur when using roundrobin scheduling and scheduling based on ADWIN where $\delta = 10^{-1}$.

6.3.3.2 Large Change

ADWIN1 outperforms roundrobin also for a large change in a 10 peer network, as can be seen in Figure 6.23a. In this case we can also observe a significant difference in the aggregate error, as seen in Figure 6.25, which is likely due to the increased impact that the changed link has on the aggregate error when the change is large.



(a) Roundrobin and ADWIN where $\delta = 10^{-1}$.

(b) ADWIN where $\delta = 10^{-1}$ and $\delta = 10^{-50}$.

Figure 6.23: Mean and variance values for an exponentially weighted model of a single link where a large change occurs in a ten node network.

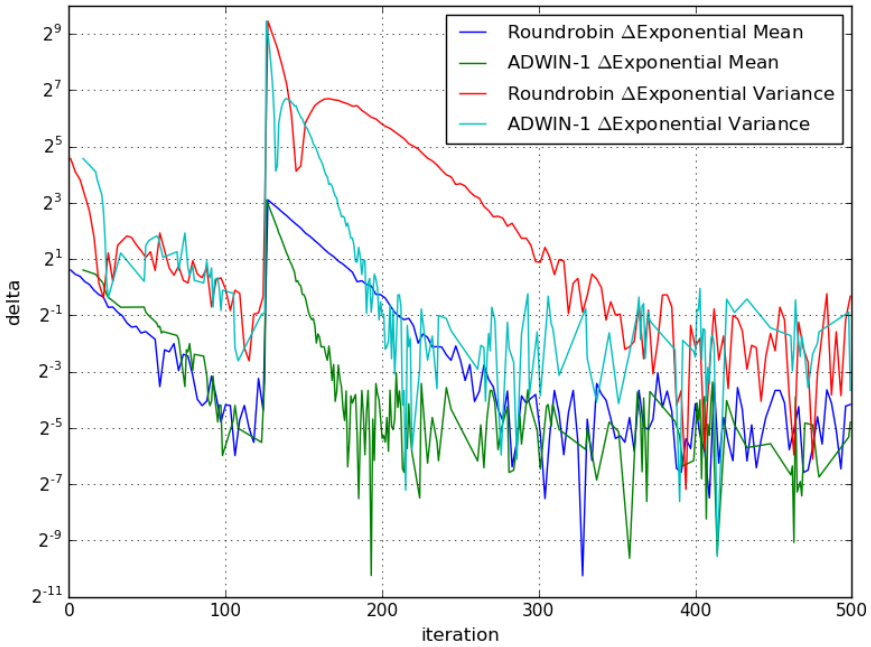


Figure 6.24: Deltas for an exponentially weighted model of a single link where a large change occurs in a ten node network using roundrobin scheduling and scheduling based on ADWIN where $\delta = 10^{-1}$.

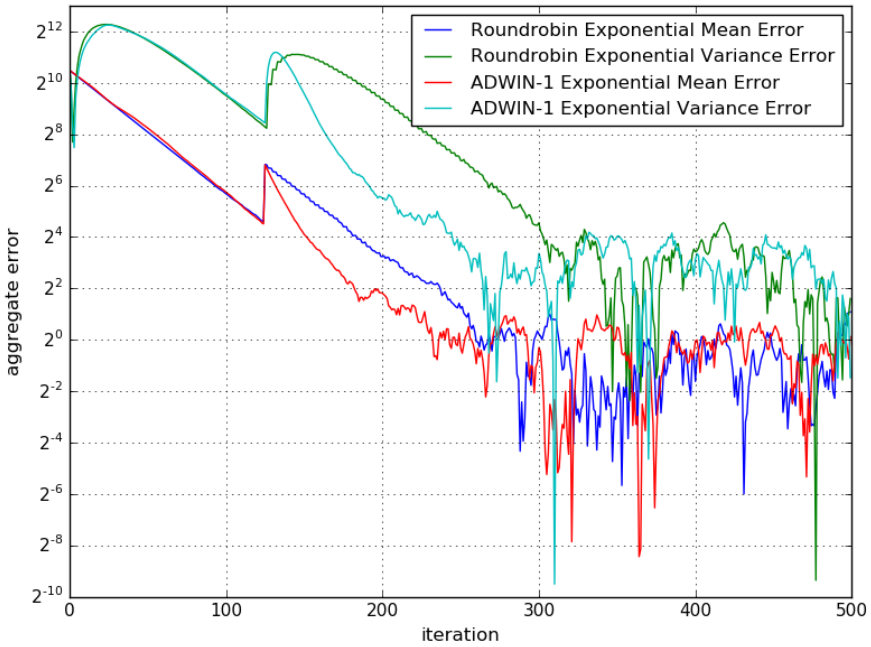
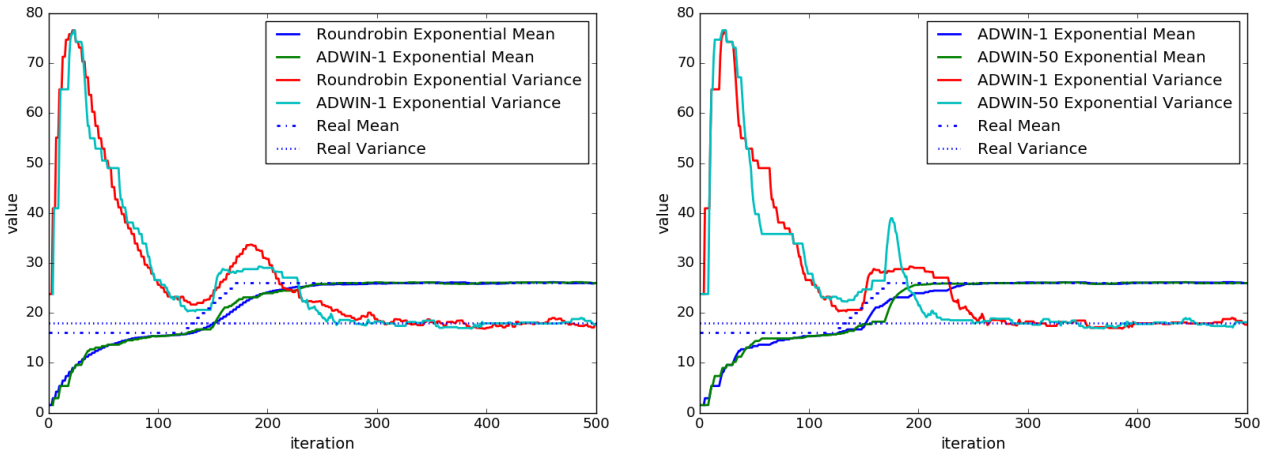


Figure 6.25: The aggregate error for a ten node network where a large change occur when using roundrobin scheduling and scheduling based on ADWIN where $\delta = 10^{-1}$.

6.3.3.3 Gradual Change

In Figure 6.26a we can see that also in the ten peer case the ADWIN1 scheduling has a slight advantage over roundrobin for a gradual change. The aggregate error shows no significant improvement however, and ADWIN1 using $\delta = 10^{-50}$ performs worse than $\delta = 10^{-1}$ as in the other simulations.



(a) Roundrobin and ADWIN where $\delta = 10^{-1}$.

(b) ADWIN where $\delta = 10^{-1}$ and $\delta = 10^{-50}$.

Figure 6.26: Mean and variance values for an exponentially weighted model of a single link where a gradual change occurs in a ten node network.

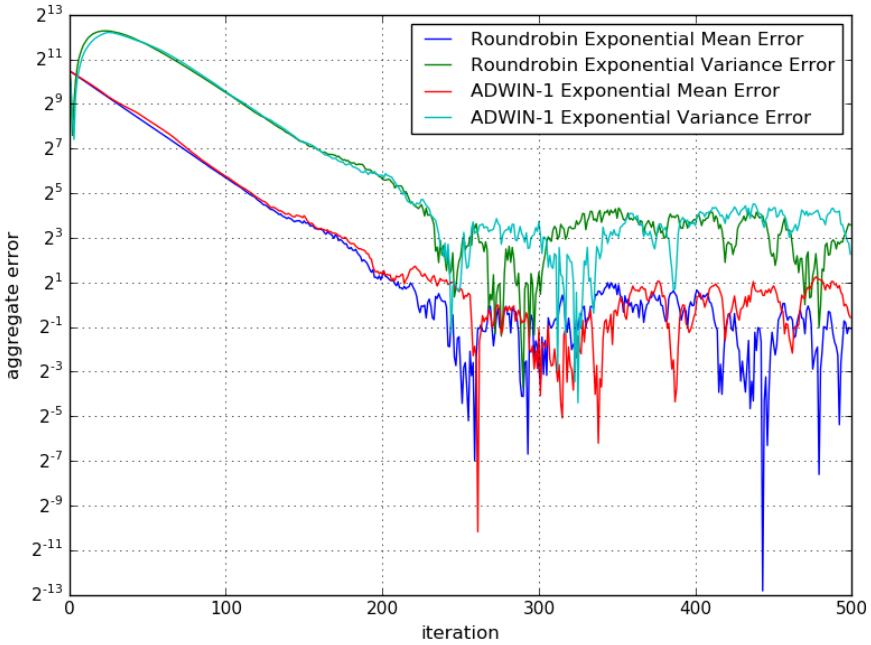
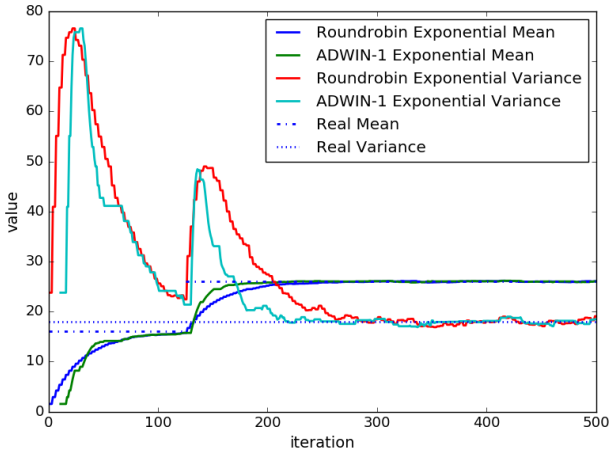


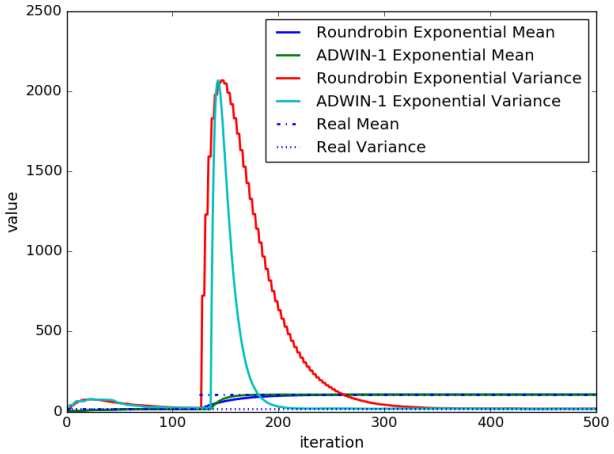
Figure 6.27: The aggregate error for a ten node network where a gradual change occur when using roundrobin scheduling and scheduling based on ADWIN where $\delta = 10^{-1}$.

6.3.3.4 Multiple Simultaneous Changes

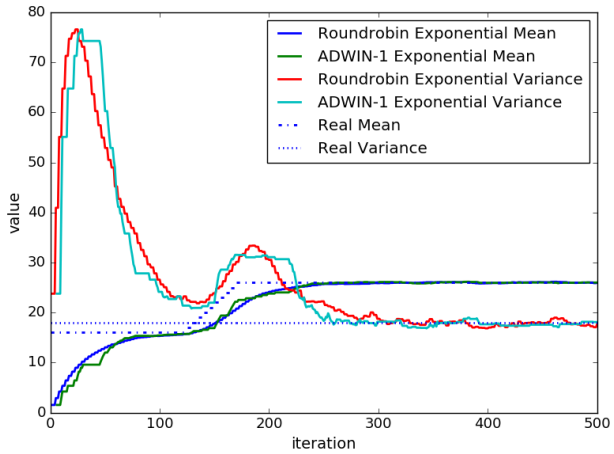
With the exception of the gradual case, as seen in Figure 6.28c, the performance of the ADWIN1 based scheduler outperforms the roundrobin one in the case where several simultaneous changes are happening in the network. We can also observe in Figure 6.29 that the aggregate error is decreased significantly faster by the ADWIN1 scheduler after the changes have occurred.



(a) Small change, ADWIN $\delta = 10^{-1}$.



(b) Large change, ADWIN $\delta = 10^{-1}$.



(c) Gradual change, ADWIN $\delta = 10^{-1}$.

Figure 6.28: Mean and variance values for exponentially weighted models of links where a small, large and gradual change occurs at the same time in a ten node network.

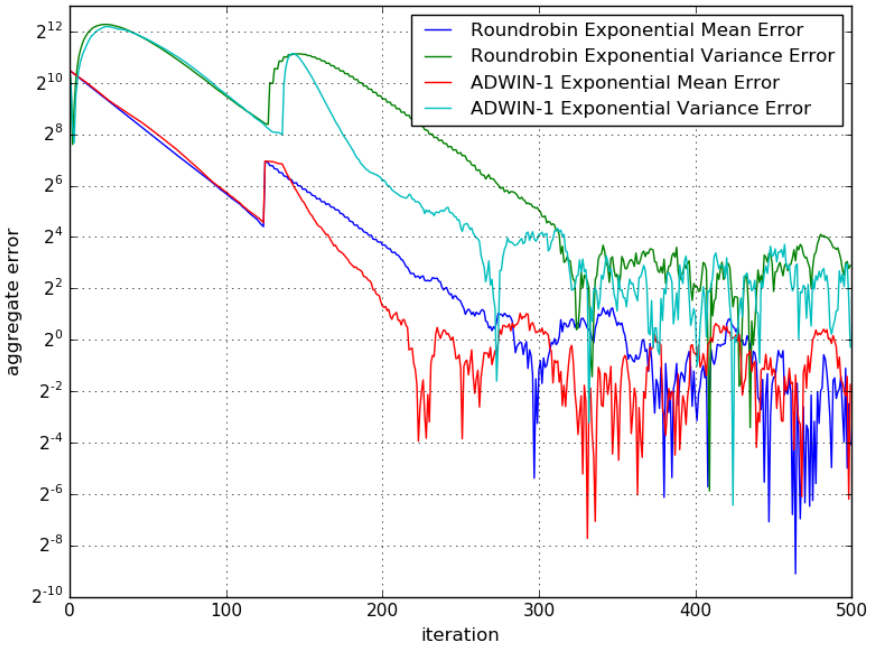


Figure 6.29: The aggregate error for a ten node network where a node simultaneously experiences a small, a large and a gradual change in three of its links, when using roundrobin scheduling and scheduling based on ADWIN where $\delta = 10^{-1}$.

Chapter 7

Discussion & Conclusion

7.1 Discussion

The results in Chapter 6 show that the method presented in this thesis for scheduling measurements is more efficient in terms of maximising information gain than roundrobin scheduling for a sudden small or large change (as defined in Table 5.1) in increased mean delay in a simulated network path.

The results for a gradual change are inconclusive as to whether the method provides any significant benefit over roundrobin scheduling. In Figure 6.26a there appears to be a slight advantage on the single link level, however the aggregate error as seen in Figure 6.27 does not conclusively show the method to be better or worse than roundrobin.

Although results are positive, the method used in this thesis to evaluate the proposed measurement scheduling scheme does have a few limitations:

- It is assumed that the delay patterns are relatively regular and stable over longer time frames, i.e. they should be stable over multiple measurement cycles. The evaluation does not consider cyclic variations such as daily or weekly congestion patterns.
- The test cases only cover *increases* in delay and does not evaluate how a decrease might affect the scheduling. It is assumed that it would provide similar results.
- The way in which change is introduced, by shifting a gamma distribution along the x-axis, is a bit simplistic and may not reflect how changes in real networks behave.

The results show that the proposed method for scheduling measurements does work in the simulated test cases, however there are some limitations that could pose some issue when applying it to real world networks:

- The maximum window size for the ADWIN1 algorithm was arbitrarily chosen to be 20 with the motivation that it would be sufficient for providing results, while not being overly excessive considering possible storage constraints.
- The value for the α variable of the exponentially weighted online algorithm described in Section 3.1.2.2 was chosen to fit the given test cases. However the value chosen might be wrong for some real world links, and there might not be a one size fits all answer to what α should be for this use case. There might have to be some adjustments on a per link basis to achieve the best results.
- The value chosen for the δ variable in the ADWIN1 algorithm was chosen to fit the given test cases. The algorithm might have different performance for different links depending on their stability, i.e. how large the variance is. Tweaking the δ value for each link might help reduce false positives in change detection for links where a high variance is the norm.

7.1.1 Future Work

An area of future work would be to explore how to improve the measurement scheduling method. Some variables used in different parts of the method can be tweaked and there might be more optimal values than the ones used here. Examples are the α value of the exponentially weighted algorithm for mean and variance, the δ confidence value in the ADWIN1 algorithm, and also the maximum length of a sliding window managed by the ADWIN1 algorithm. There might also be different "optimal" values depending on the distribution of the data, so it could prove beneficial to use different values per link based on some heuristic.

One interesting prospect would be to investigate if you could improve the scheduler by letting it dynamically tweak the α value when adding new data to an existing model for a path. For example if the window is short, i.e. a change has likely occurred, then it might choose to give larger weight to newer data by tweaking the α value to quickly converge to the new normal, after which it would return α to the default value.

To expand on this and get conclusive results on the effectiveness of the scheduling method it needs to be tested with real connections going over the internet. However, it will be hard to evaluate since in such cases we don't know the underlying distribution for the delays we are measuring. A possible way of evaluating would be to test it out in a small scale where you could feasibly perform measurements for all links all the time with high granularity and use those results as the "ground truth". One could then run the intended scheduler and measurements in parallel and evaluate the resulting view of the network against the "ground truth" and see how big the discrepancy is.

As in a lot of cases there might not exist a "one size fits all" solution to this specific scheduling problem. In many of the results the round-robin scheduler had a more stable convergence during the warm-up period. To get the best overall result it might be beneficial to use round-robin scheduling in the beginning and then use

a different method after relative stability has been achieved. Finding a balance between different scheduling methods to achieve better results is an area still left to explore.

7.2 Conclusion

From the results we can conclude that if we keep sliding windows of estimated values for the variance of network paths, and let the ADWIN1 algorithm manage the length of these windows, then the length of the windows can be used as a basis for measurement scheduling. The results show that for these simulation experiments this method leads to faster recovery in general, i.e. more efficient information gain from subsequent measurements, after changes occur in the network when compared to a round-robin scheduling scheme.

Appendix A

Simulation Configurations

The configurations files used for the simulations can be found at the following link: `https://gitlab.com/glindstedt/thesis-data`.

Bibliography

[1] J. Hawkinson and T. Bates. *Guidelines for creation, selection, and registration of an Autonomous System (AS)*. BCP 6. http://www.rfc-editor.org/rfc/rfc1930.txt. RFC Editor, Mar. 1996. URL: http://www.rfc-editor.org/rfc/rfc1930.txt.

[2] Wolfgang John, Maurizio Dusi, and Kimberly C Claffy. "Estimating routing symmetry on single links by passive flow measurements". In: *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*. ACM. 2010, pp. 473–478.

[3] Yakov Rekhter et al. *Address Allocation for Private Internets*. BCP 5. http://www.rfc-editor.org/rfc/rfc1918.txt. RFC Editor, Feb. 1996. URL: http://www.rfc-editor.org/rfc/rfc1918.txt.

[4] J. Rosenberg et al. *Session Traversal Utilities for NAT (STUN)*. RFC 5389. http://www.rfc-editor.org/rfc/rfc5389.txt. RFC Editor, Oct. 2008. URL: http://www.rfc-editor.org/rfc/rfc5389.txt.

[5] R. Mahy, P. Matthews, and J. Rosenberg. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. RFC 5766. http://www.rfc-editor.org/rfc/rfc5766.txt. RFC Editor, Apr. 2010. URL: http://www.rfc-editor.org/rfc/rfc5766.txt.

[6] Liam McNamara and Ian Marsh. "CheesePi: A Raspberry Pi based measurement platform". In: *Research and Applications of Internet Measurements (RAIM)*. 2015.

[7] Raspberry Pi Foundation. *Raspberry Pi - Teach, Learn, and Make with Raspberry Pi*. URL: https://www.raspberrypi.org/ (visited on 06/03/2018).

[8] Raspberry Pi Foundation. *Raspberry Pi 3 On Sale Now At $35*. 2016. URL: https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/ (visited on 02/29/2016).

[9] Bram Cohen. *BitTorrent Protocol Specification*. URL: http://www.bittorrent.org/beps/bep_0003.html (visited on 10/07/2017).

[10] WebRTC Official Website. *WebRTC Official Website*. URL: https://webrtc.org/ (visited on 10/07/2017).

[11] Yuval Shavitt and Eran Shir. "DIMES: Let the Internet measure itself". In: *ACM SIGCOMM Computer Communication Review* 35.5 (2005), pp. 71–74.

[12] Simone Basso, Antonio Servetti, and Juan Carlos De Martin. "The network neutrality bot architecture: a preliminary approach for self-monitoring of Internet access QoS". In: *Computers and Communications (ISCC), 2011 IEEE Symposium on*. IEEE. 2011, pp. 1131–1136.

[13] Mario A Sánchez et al. "Dasu: Pushing Experiments to the Internet's Edge." In: *NSDI*. 2013, pp. 487–499.

[14] Feng Zhao, Jaewon Shin, and James Reich. "Information-driven dynamic sensor collaboration". In: *IEEE Signal processing magazine* 19.2 (2002), pp. 61–72.

[15] Vijay Gupta et al. "Sensor scheduling algorithms requiring limited computation [vehicle sonar range-finder example]". In: *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*. Vol. 3. IEEE. 2004, pp. iii–825.

[16] BP Welford. "Note on a method for calculating corrected sums of squares and products". In: *Technometrics* 4.3 (1962), pp. 419–420.

[17] Tony F Chan, Gene H Golub, and Randall J LeVeque. "Algorithms for computing the sample variance: Analysis and recommendations". In: *The American Statistician* 37.3 (1983), pp. 242–247.

[18] Tony Finch. "Incremental calculation of weighted mean and variance". In: *University of Cambridge* 4 (2009), pp. 11–5.

[19] Albert Bifet and Ricard Gavalda. "Learning from Time-Changing Data with Adaptive Windowing." In: *SDM*. Vol. 7. SIAM. 2007, p. 2007.

[20] *InfluxDB*. URL: `https://influxdata.com/` (visited on 06/03/2018).

[21] Paul J. Leach, Michael Mealling, and Rich Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. RFC 4122. `http://www.rfc-editor.org/rfc/rfc4122.txt`. RFC Editor, July 2005. URL: `http://www.rfc-editor.org/rfc/rfc4122.txt`.

[22] *JSON*. URL: `https://json.org` (visited on 06/03/2018).

[23] *Twisted*. URL: `https://twistedmatrix.com` (visited on 06/03/2018).

[24] *txmsgpackrpc*. URL: `https://github.com/jakm/txmsgpackrpc` (visited on 06/03/2018).

[25] *MessagePack*. URL: `https://msgpack.org/` (visited on 06/03/2018).

[26] *SciPy*. URL: `https://www.scipy.org/` (visited on 06/03/2018).

[27] *NumPy*. URL: `http://www.numpy.org/` (visited on 06/03/2018).

[28] *Matplotlib*. URL: `https://matplotlib.org` (visited on 06/03/2018).

[29] *Pickle*. URL: `https://docs.python.org/2/library/pickle.html` (visited on 06/03/2018).

[30] *TkInter*. URL: `https://wiki.python.org/moin/TkInter` (visited on 06/03/2018).

[31] *Tcl/Tk*. URL: `https://www.tcl.tk/` (visited on 06/03/2018).

[32] *MongoDB*. URL: `https://www.mongodb.com/` (visited on 06/03/2018).

TRITA -EECS-EX-2018:776