University of Glasgow | School of Computing Science

On The Scalability of ROS

Isaac Jordan

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

Level 4 Project — November 10, 2016

Abstract

Robots, distributed systems, and middleware.

Education Use Consent

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

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

1.1 Context

Robotics is the field of creating physical systems to automate tasks. As a concept, automation dates back hundreds of years - however the idea of replicating biological systems seen in nature (for example, humans) was not first seen until the 20th century. Robotics was mainly considered a theoretical or fictitious fantasy, however the main discussers of the concept (such as Isaac Asimov) assumed technological capability would inevitably reach the level where robotics would either be inseperable from society, or be banned due to it's impact.

In the 19th century the beginnings of robotics could be seen in the creation of vehicles which could be remotely operated using electric signals. By the early 20th century, wireless radio guidance systems were advanced enough that remotely operated aircraft could be demonstrated in 1917 (by Archibald Low [citation needed]).

Technology advanced so rapidly during the 20th century that by the 1970s the Soviet Union could explore the surface of the moon using a remotely operated vehicle robot.

However, these uses of robotics displayed no 'intelligent behaviour' on behalf of the robotic system. Beginning in the 1980s, the growing field of Artificial Intelligence (A.I.) was revived and began creating systems which could intelligently answer domain-specific questions - called expert systems.

Robotics is a fast-progressing field which has seen major advances the past decades. From obvious examples such as Amazon's item pickers, to more integrated applications such as autonomous cars - the scale of robotics is increasing.

1.2 Aims and Objectives

If I had an aim it would go here.

1.3 Achievements

If I had achieved anything it would go here.

Chapter 2

Background

2.1 Robotics

Robots are the future. (Why robots are important)

Robotics is the field concerning autonomous computer systems, usually involving physical interaction with it's environment. Robotics lies at the intersection between computing science, eletrical and mechanical engineering. Robotics has been common in popular culture since the 1940s, however the physical realisation of these systems did not begin to be practical until the 2000s when increases in computing power, sensor accuracy, research investment, and software algorithms allowed for useful robotics systems to be created.

Today's robotic systems generally consist of many small autonomous systems working together to form a coherent whole. For example, a particular sensor (say a camera) may be constantly recording data and storing it in some buffer (erasing the oldest when full). This subsystem does not depend on any others to complete it's task (recording the envionrment), but other subsystems may rely on it's output (such as a computer vision package which needs video frame inputs).

Robotics is part of a field called cyber-physical systems (CPS) that concerns computer systems that interface with the physical world. These can include Internet Of Things (IOT) networks which can (for example) be used to control things around the household, such as light switches, central heating systems, and hoovering. This is distinct from robotics as CPS generally embody a 'think globally, act locally' approach (such as using data from many sources to improve the CPS' performance in each), compared to robotics which utilises a more 'think locally, act locally' strategy - meaning that each instance of a robot generally makes it's own decisions, and acts upon them solely.

2.2 Multi-Robot Systems

Multi-robot systems are specific instances of mult-agent systems. They represent a joint problem space between robotics, artificial intelligence, and distributed systems. Multi-robot systems as a formal concept is a recent development with a IEEE technical committee only being formed in 2014[2].

Multi-robot systems can consist of many intelligent agents (each of which may be comprised of many small autonomous systems) working to solve a task that any one system may not be able to solve alone. One such example is a warehouse management system.

In a warehouse, millions of items can be spread throughout miles of shelving and the requirement is that a random subset of items (those that have been bought) must arrive at a specific point (the delivery pick-up point) at a specific time (when the van is there). This task previously could be solved by having human pickers wander the isles searching for items - however given recent increases in the size of online shopping this is no longer feasible. Now, online retailers are using thousands of individual robots to intelligently move the shelving around and bring the correct items to stationary human pickers. This system requires the coordination of the individual robots, but they must all act independently in order to efficiently keep pace with the items ordered.

2.3 Scalability in Robotics

When creating multi-robot systems, scalability becomes an important concern. At what level does a system architect choose to switch from small highly-powered robot systems to a larger number of simpler robot systems. Several factors to consider include the processing power of each robot, the communication framework required to organise them, the cost of each robot, and the suitability of the problem to a multi-robot system.

Swarm robotics is the extreme approach of creating many very simple robots which individually could not solve tasks or survive environments - however when they work together as a form of society they can work efficiently.

2.4 Robotic Middleware

Robot designers want to work at a high level. Sensors are low level. (Why is middleware needed, whats available)

Robotic middleware is a software infrastructure that is intended to provide convenient abstraction and communication paradigms for facilitating this multi-subsystem approach. In general, a robotics middleware would provide interfaces for defining each subsystem, and defning how each subsystem communicates with others.

Robots consist of many individual sensors, for example a camera that can record 720p RGB video at 30fps, a LIDAR range sensor that can measure distances of up to 30m at a frequency of 1-500Hz, a tri-axis gyroscope and accelerometer capable of measuring up to 16g of force with a measurement frequency of 40Hz. These devices use a range of sophisticated technologies to measure and analyse the physical world, each recording a different data format at a different data rate. These sensor modules are all independently designed, resulting in a wide array of different software and hardware interfaces - meaning that each robot system that wishes to use these sensors must create the software to communicate with these sensors, consume their data, and then write the robotic software that makes use of it. This results in a wide variety of implementations for manipulating the same data on each sensor, increasing the likelihood of implementation mistakes, misunderstandings, and wasting researchers' time.

Software called middleware is the solution to this problem. A middleware provides a common interface design so that no matter what hardware is creating the data, the results are distributed in a consistent manner. This means that hardware manufacturers (or users) need only implement one software system for each sensor that can them be distributed and reused amongst all users of that sensor, as long as those users are using that middleware. This means that creators of systems utilising that middleware have easy access to the data created by a particular sensor as they know the software the create will be able to easily consume the sensor's data stream.

Another benefit of middleware is that this communication interface can be reused within the robotic system for communicating between distinct modules within the system. For example, a researcher may create a computer vision module that processes a video stream and returns a data stream containing a description of the objects in the

video stream at each frame. When utilising a middleware, the researcher can make the result of their computer vision module available in a similar fashion to the sensor's video stream - meaning that higher level software can utilise the results as if there was an 'object-detecting hardware sensor'. These levels of abstractions make software much more maintainable, and reusable.

There are a wide variety of robotic middlewares in use currently. Many employ a free (libre) approach to software by making the source code available online, whereas others are created for commercial licensing using propriatary source.

2.4.1 MOOS (Mission Oriented Operating Suite) [3]

Design

The designers of MOOS claim that MOOS is 'a suite of libraries and executables designed and proven to run a field robot in sub-sea and land domains'[3]. MOOS is one of the few robot middlewares to explicitly target robots in a sub-sea domain. MOOS is designed to be platform independent, allowing for flexibility in the user's choice of operating system. Other design details considered during it's creation are that vehicles should be run using a network of communicating processes, each of which is specialised for a specific task. The communication should be robust failures in the stop and starting of individual processes, and processes should have no shared header file dependencies[3] - this last point means that each process can be compiled separately from any processes that it makes use of.

Communication Topology

2.4.2 YARP (Yet Another Robot Platform)

2.4.3 Orocos (Open RObot Control Software)

2.4.4 CARMEN (Carnegie Mellon Robot Navigation Toolkit)

2.4.5 Orca

2.4.6 Microsoft Robotics Developer Studio (v4)

2.4.7 OpenRTM-aist

2.4.8 Player

2.4.9 ROS (Robot Operating System)

2.5 ROS (Robot Operating System)

ROS is what I'll be testing out. (What is it)

2.6 Configuration of Robots

The robots used in this project are 9 identical robot cars with front-wheel steering. (What is their set up)

Chapter 3

Experimental Analysis

3.1 Experiment 1

The first experiment was designed to analyse the transfer time of messages between two machines at varying message frequencies. This would highlight whether there was some limit as to how often ROS could send and receive messages on it's topics.

3.1.1 Configuration

These two machines were Raspberry Pi 3 Model Bs connected via ethernet to an Asus router.

The messages were sent and received using ROS Kinetic, running on the Raspbian OS.

The experimental setup was that one Raspberry Pi would run a ROS master node, another Raspberry Pi runs a sender program that notes down the message-sent time, and sends it to a 3rd Raspberry Pi which merely echoes the message back to the sender. Upon receiving the message, the sender/receiver notes the current time, and writes 'message X which was sent at Y, was received at Z' to a text file. The resulting Round Trip Time (RTT) for each message would therefore be the difference between the message-sent time and the message-received time.

The expected result of the experiment was that message latency (RTT) would be the same across all lower frequencies, until some bottleneck was reached that would then cause message latencies to exponentially increase due to congestion.

Code had been written prior to execute this experiment, so initially this was used[1]. However, this gave results that were contrary to the hypothesis. An increase in message frequency resulted in a reduction in message latency. A number of messages on higher frequencies were also dropped, and never received. As this was the opposite of the hypothesis, the first step was to critique the experiment code.

This review highlighted two major issues, the first was the echoing machine had a delay similar to the sender when the experiment design mandated that the echoer always respond as fast as it can. The second issue was the the maximum message queue size in ROS (how many messages can be buffered at once to compensate for a slow subscriber) was set equal to the message frequency of that run.

These issues were resolved by removing the code that executed the delay in the echoer, and by setting the maximum queue size to be equal to 1000 in every experiment (the number of messages expected to be sent).

The experiment was then repeated using this code, and the results from these runs agreed with the hypothesis.

3.2 Experiment 2

Experiment 2 was an investigation in to whether the performance of ROS messages was affected by previous messages sent on the system. This was tested by comparing the performance of ROS messages of 5 consecutive message passing runs vs 5 message passing runs with system reboots between runs. This was repeated at 1KHz, 4KHz, 7KHz, and 10Khz frequencies for two reasons. The first was to investigate whether areas in which we observed consistent performance before would exhibit any difference between reboot and no reboot. Secondly, it would give further insight in to where the exact barrier between 'good, consistent performance' and 'poor, erratic performance' is.

Rebooting the systems involved has the opportunity to affect performance by stopping any background processes, interrupting slow processing messages from previous runs, and resetting any message caches and buffers in memory.

The result of the experiment was hypothesised to demonstrate no significant difference between rebooting and not-rebooting at any message frequency.

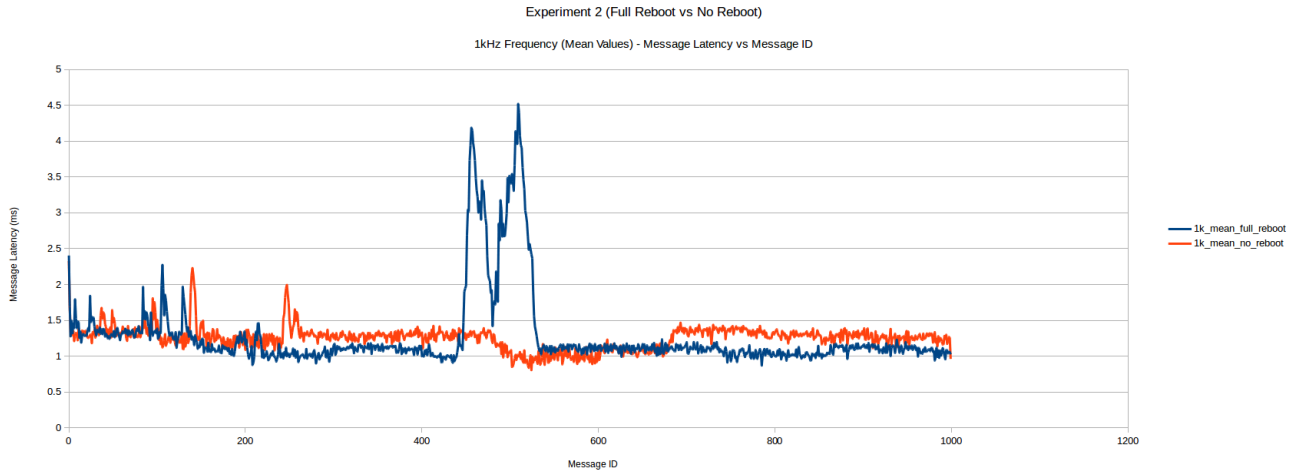

Figure 3.1: Experiment 2 - Mean Message Latency 1KHz Message Frequency

Figure 3.1 demonstrates that for relatively low message frequencies the mean message latency was consistently 1 - 1.5ms (the peak around message 500 in the full reboot data was due to 1 erroneous run at that data point). Figure 3.2 is characteristic of the higher frequency runs - the no reboot runs generally gave equal or better performance compared to the full reboot runs. See Appendix A for other mean graphs, and individual run graphs.

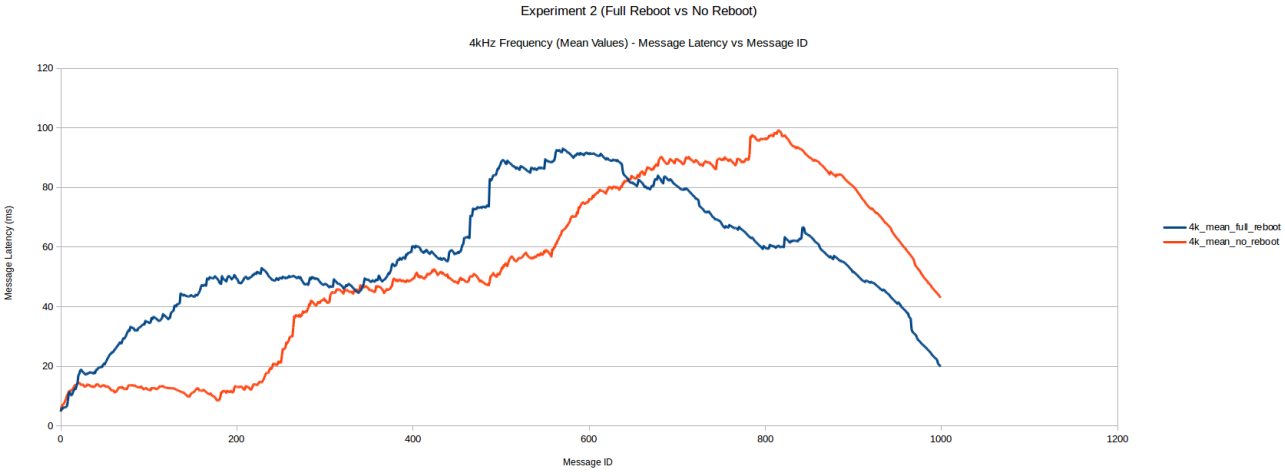

Figure 3.2: Experiment 2 - Mean Message Latency 4KHz Message Frequency

Appendices

Appendix A

Experiment 2 Other Graphs

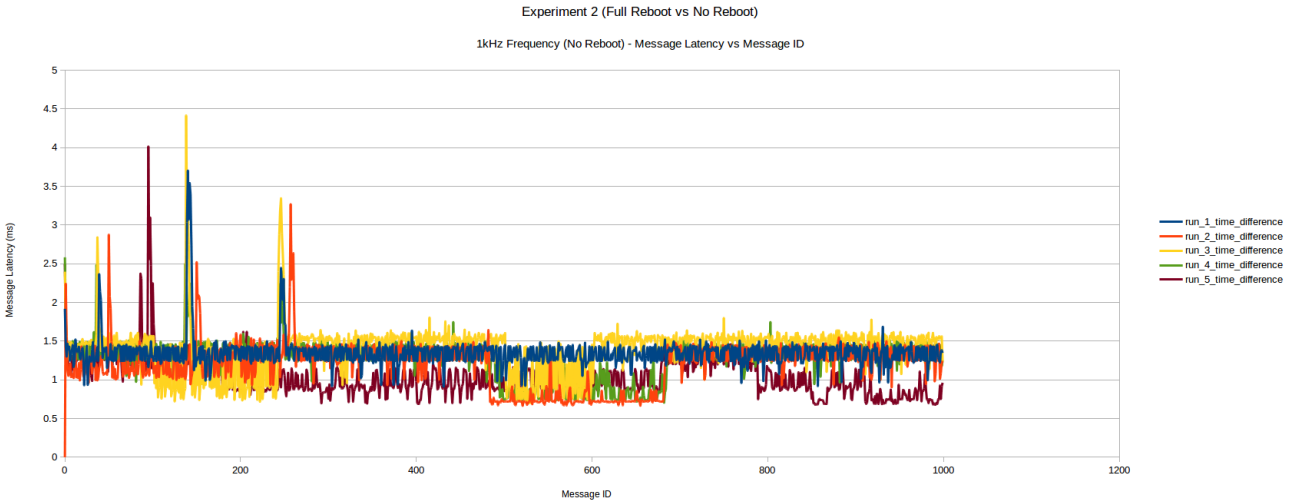

Figure A.1: Experiment 2 - No Reboot 1KHz Message Frequency

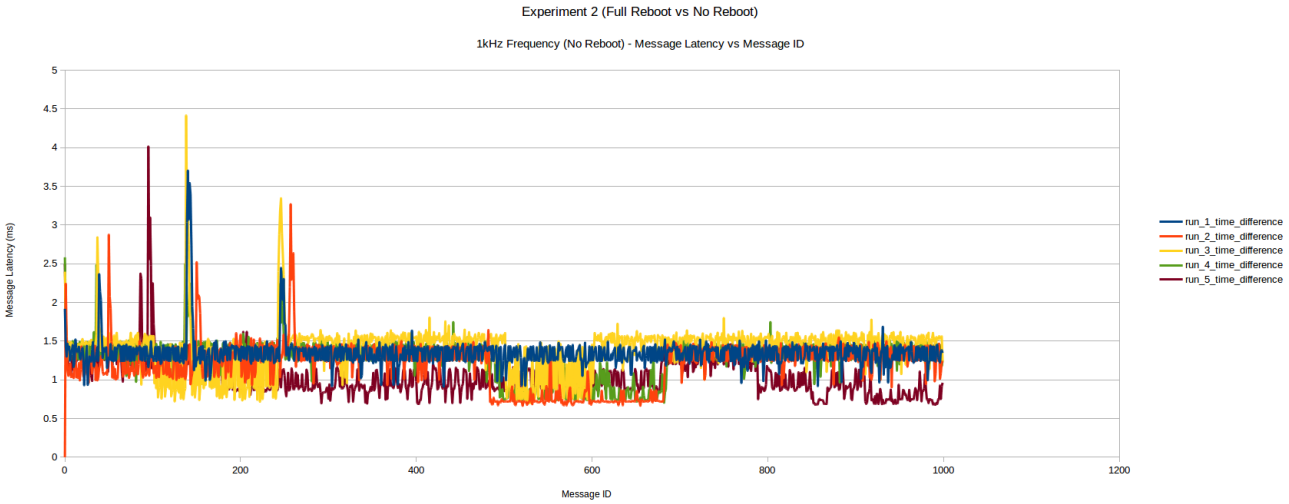

Figure A.2: Experiment 2 - No Reboot 1KHz Message Frequency

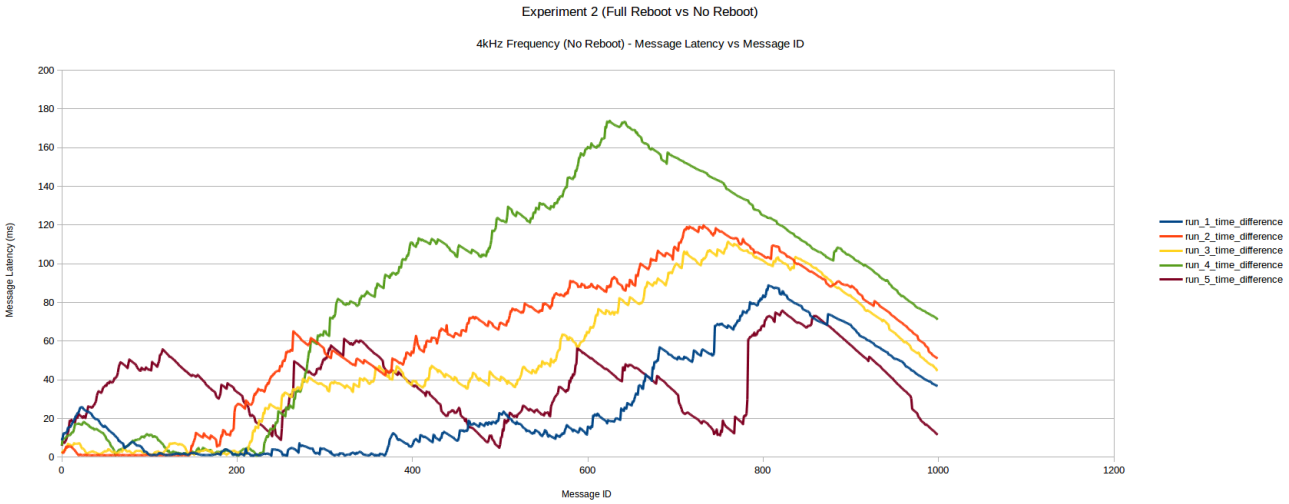

Figure A.3: Experiment 2 - No Reboot 4KHz Message Frequency

Figure A.4: Experiment 2 - Full Reboot 4KHz Message Frequency

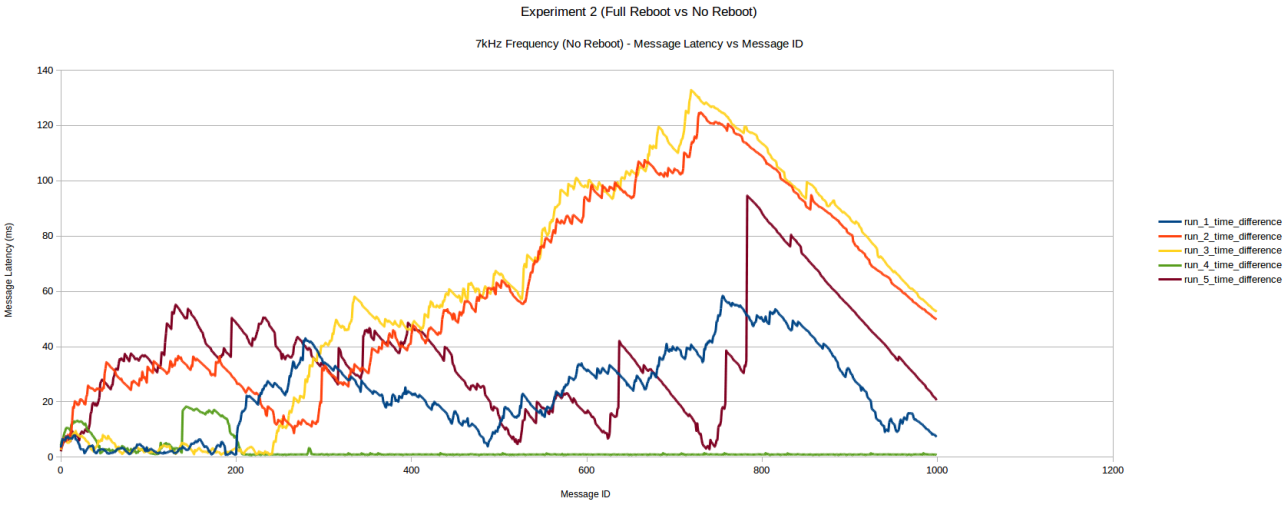

Figure A.5: Experiment 2 - No Reboot 7KHz Message Frequency

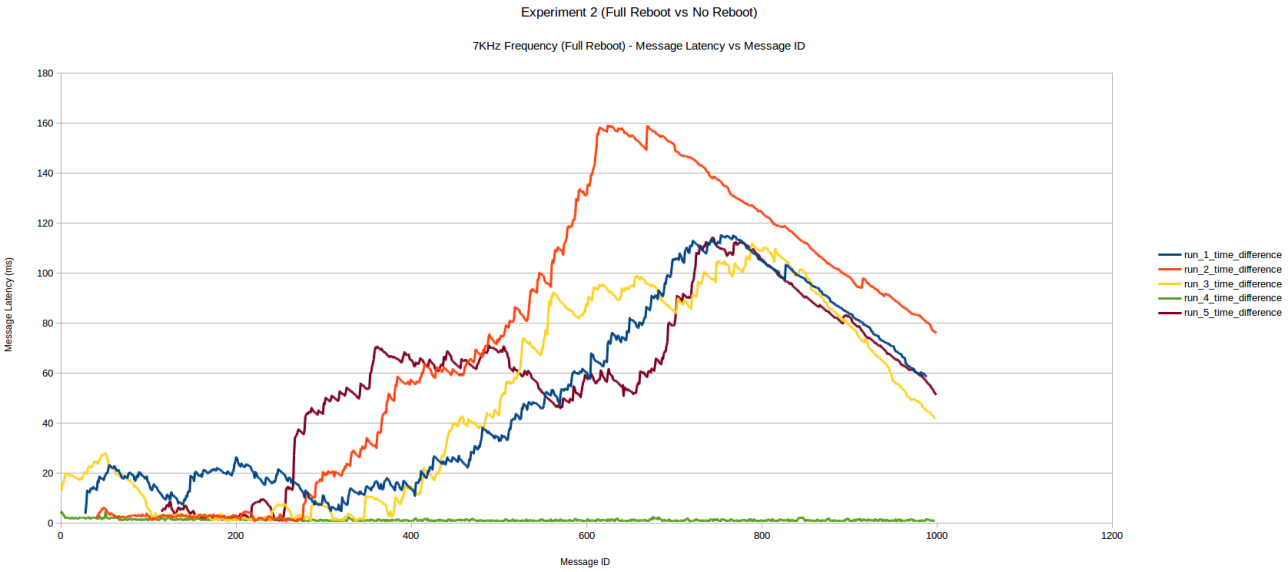

Figure A.6: Experiment 2 - Full Reboot 7KHz Message Frequency

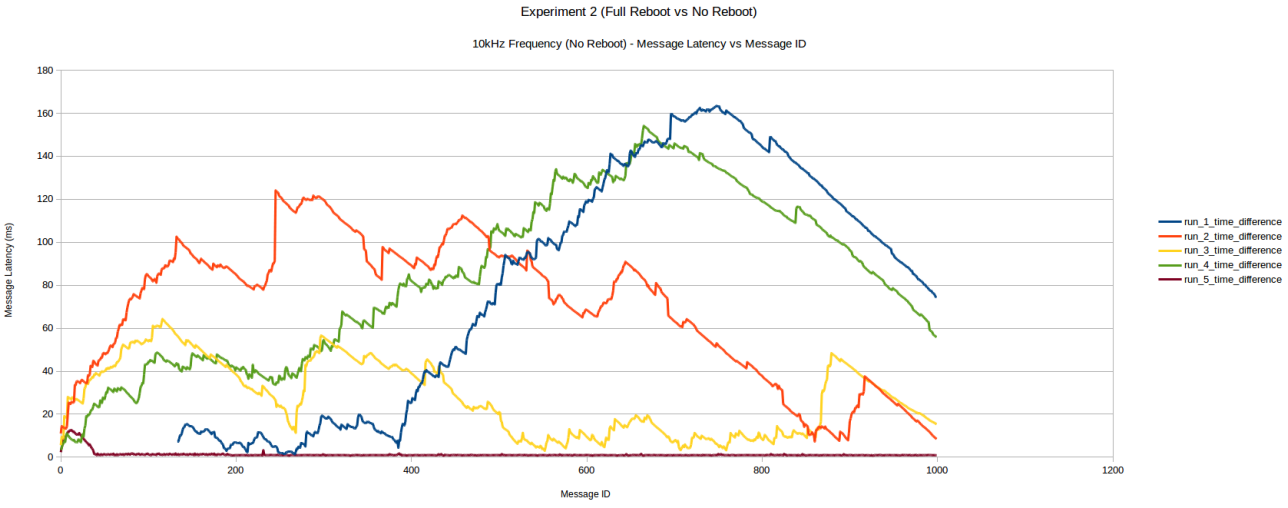

Figure A.7: Experiment 2 - No Reboot 10KHz Message Frequency

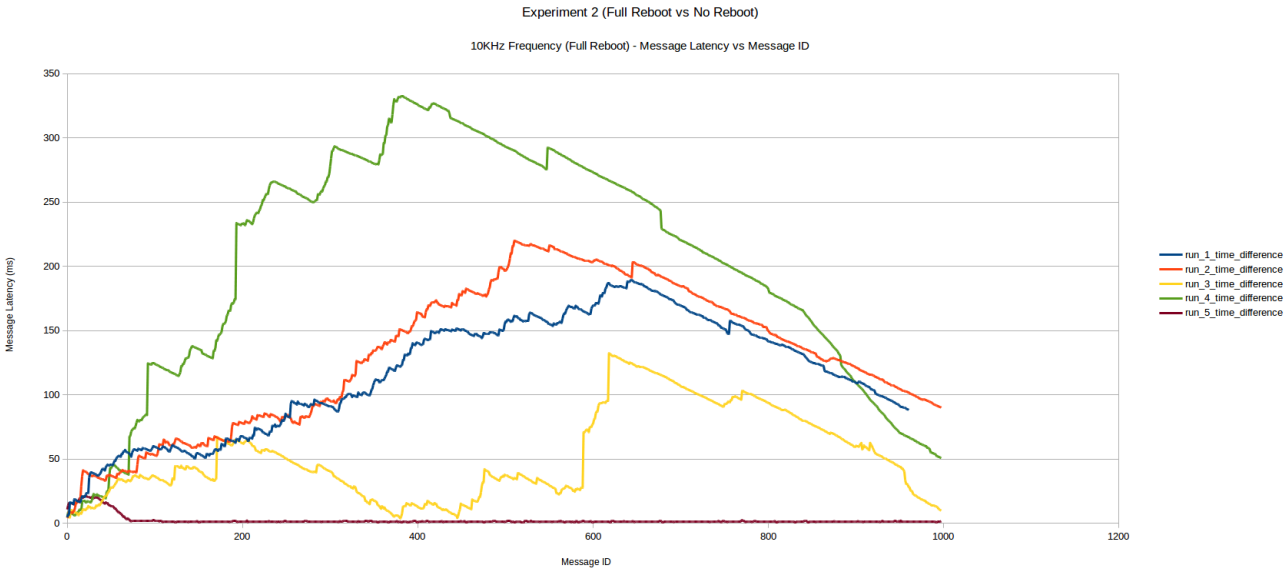

Figure A.8: Experiment 2 - Full Reboot 10KHz Message Frequency

Appendix B

Running the Programs

To compile this dissertation:

```
> pdflatex dissertation
> bibtex dissertation
> pdflatex dissertation
    > pdflatex dissertation
```

An example of running from the command line is as follows:

```
> rosrun rosberry_experiments run_experiment.py
```

Appendix C

Generating Random Graphs (Example Appendix)

```
> java RandomGraph 100 0.9 > 100-90-00.clq
```

Bibliography

[1] Message Latency Initial Code. https://bitbucket.org/natalia-chechina/2016-level4-isaac/src/10f50c8892074fc0d5d35223a367e3443ba60483/experiments/message_latency/andreeas%20version/src/rosber

[2] Technical Committee on Multi-Robot Systems (TC MRS) of the IEEE Robotics and Automation Society. http://multirobotsystems.org/.

[3] Paul Michael Newman. Moos-mission orientated operating suite. *Massachusetts Institute of Technology, Tech. Rep*, 2299(08), 2008.