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 8, 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 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).

2.2 Multi-Robot Systems

Insert multi-robot systems blurb here!

2.3 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.

Insert middlware overview here!

2.4 ROS (Robot Operating System)

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

2.5 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

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.

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[**?**]. 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.

Chapter 4

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.

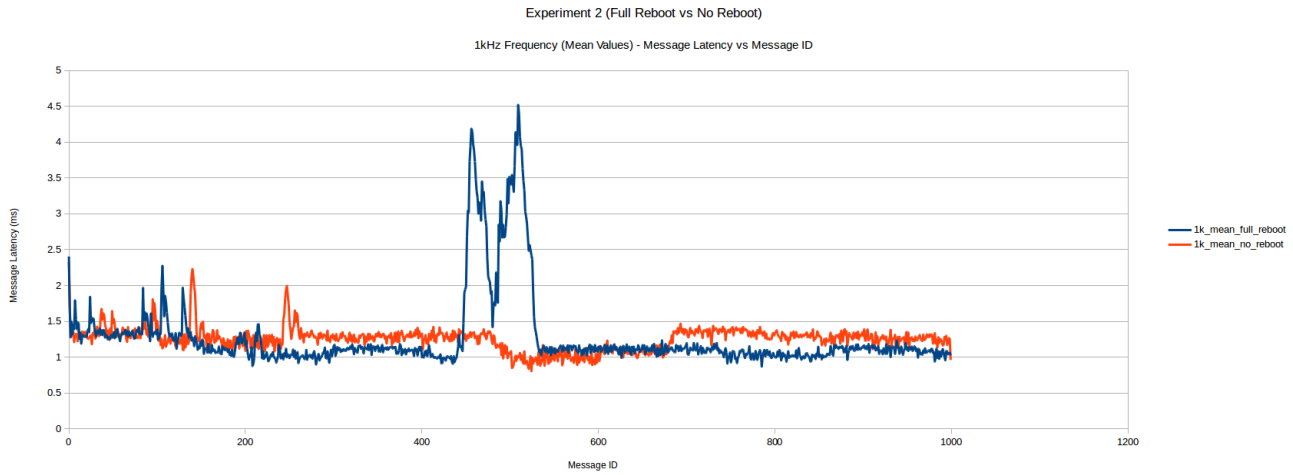

Figure 4.1: Experiment 2 - Mean Message Latency 1KHz Message Frequency

Figure 4.1 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 4.2 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.

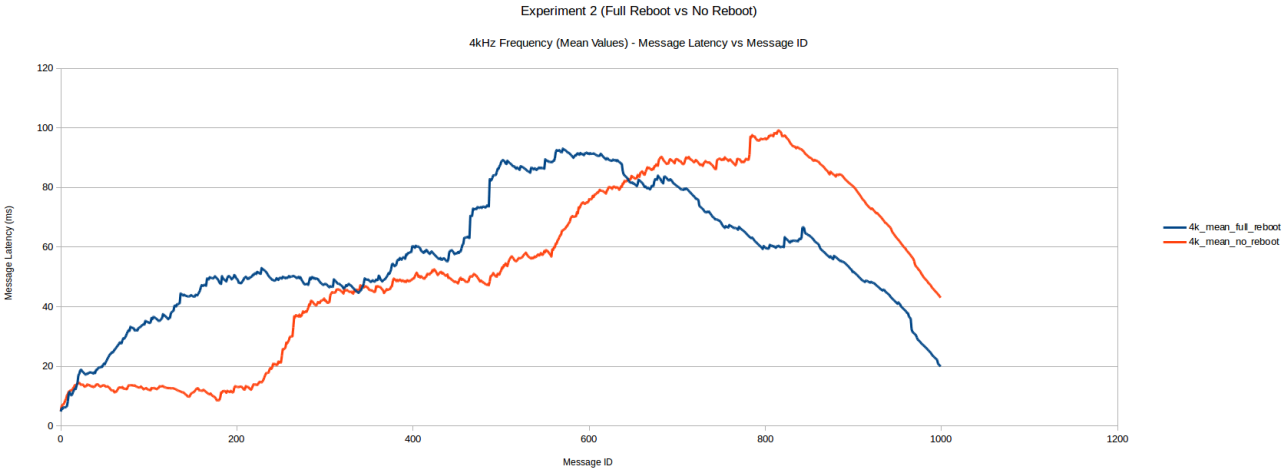

Figure 4.2: Experiment 2 - Mean Message Latency 4KHz Message Frequency

Appendices

Appendix A

Experiment 2 Other Graphs

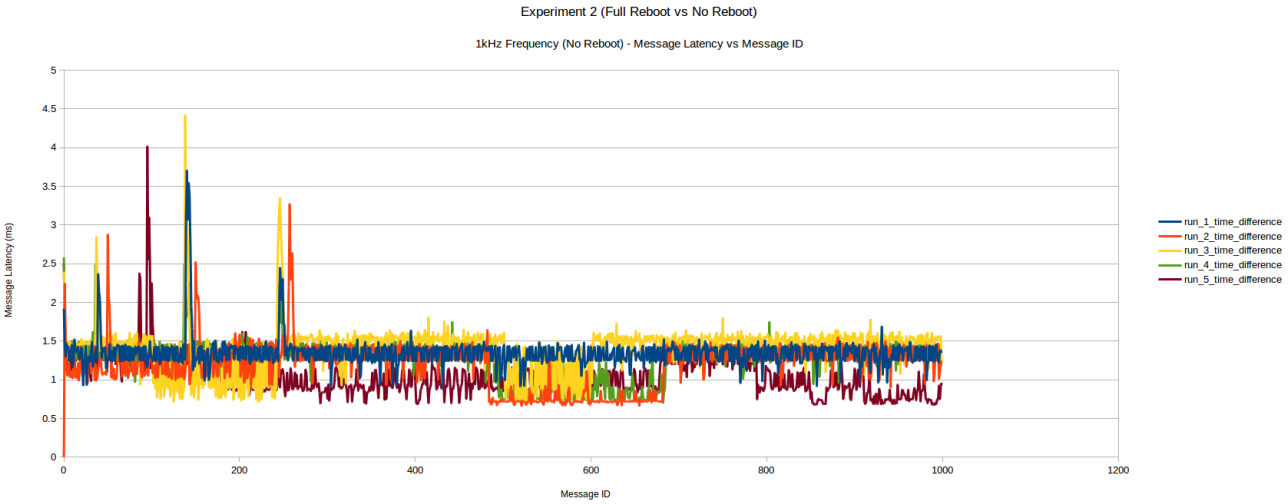

Figure A.1: Experiment 2 - No Reboot 1KHz Message Frequency

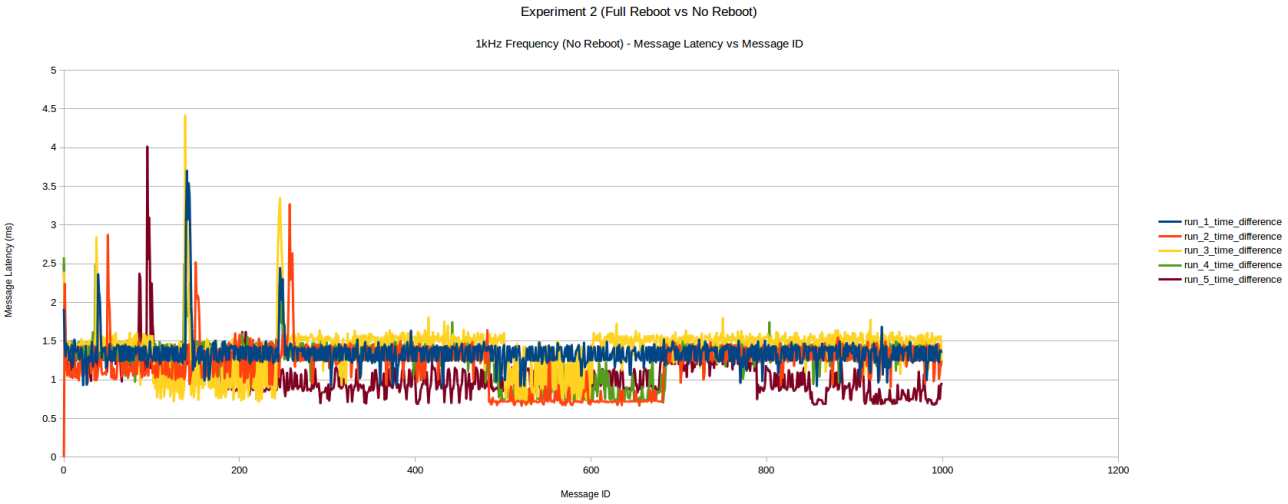

Figure A.2: Experiment 2 - No Reboot 1KHz Message Frequency

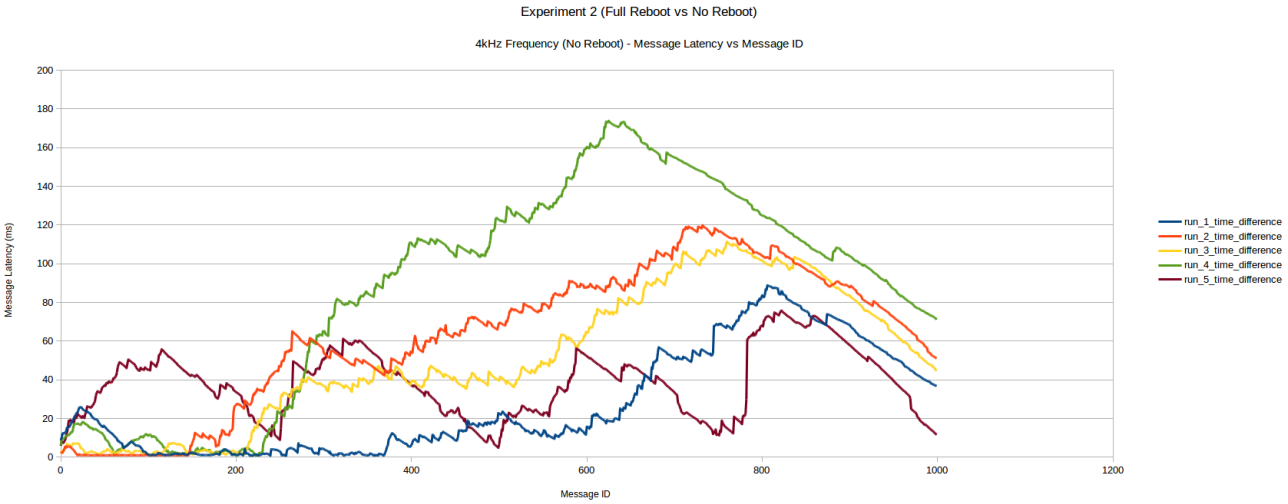

Figure A.3: Experiment 2 - No Reboot 4KHz Message Frequency

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

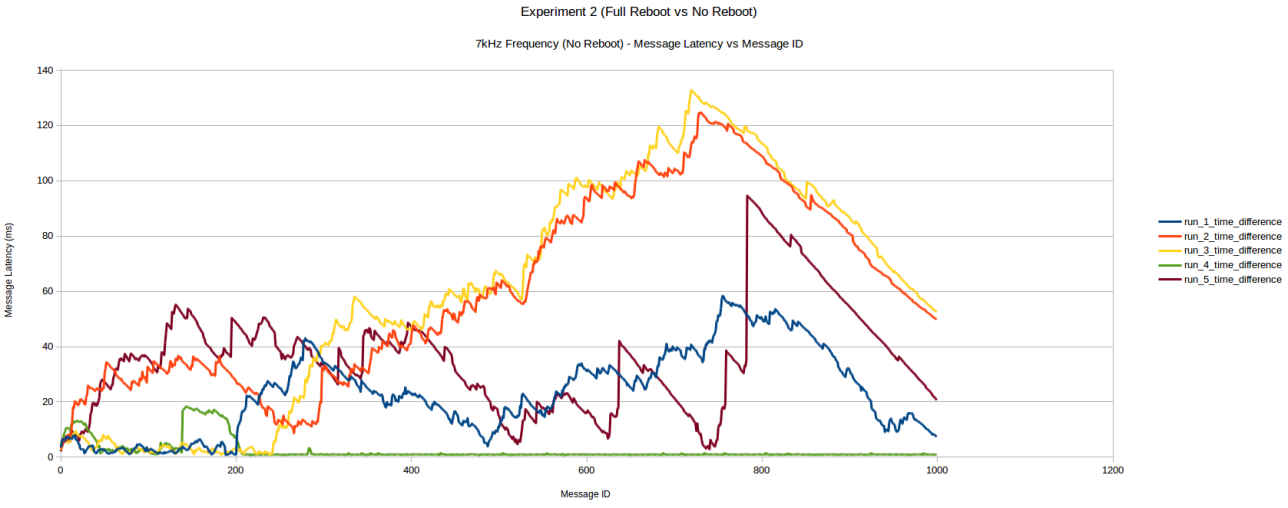

Figure A.5: Experiment 2 - No Reboot 7KHz Message Frequency

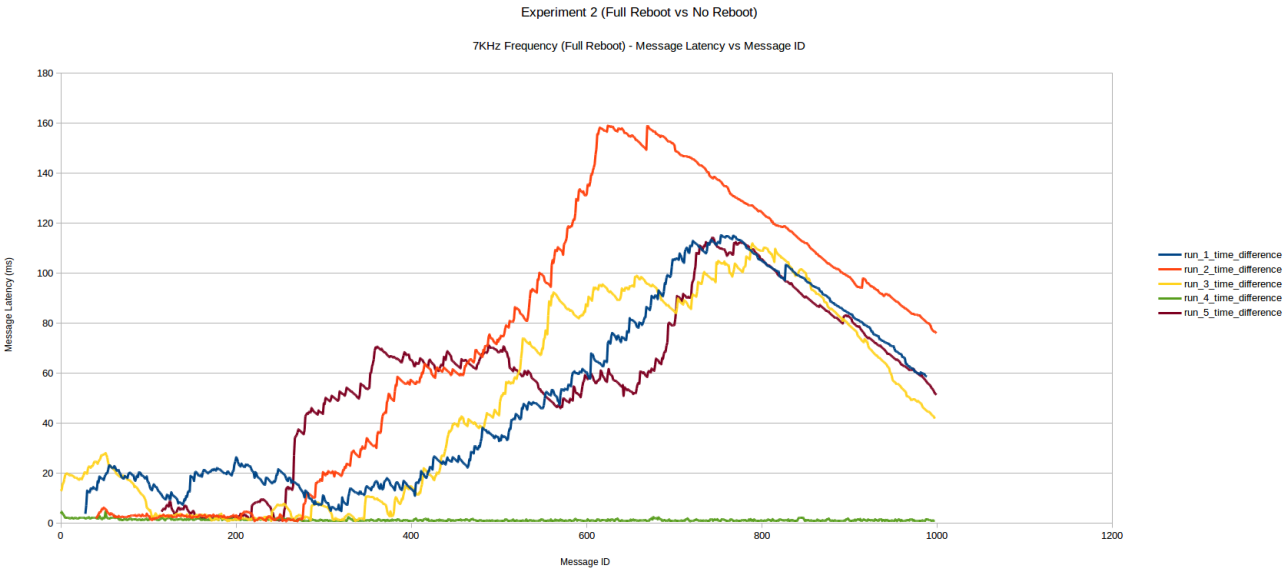

Figure A.6: Experiment 2 - Full Reboot 7KHz Message Frequency

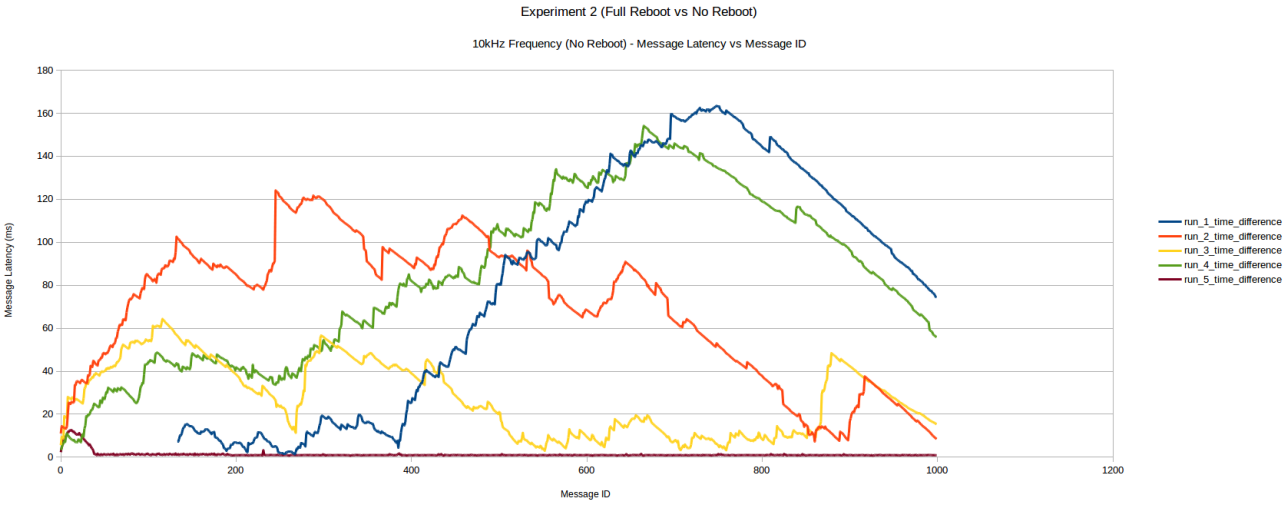

Figure A.7: Experiment 2 - No Reboot 10KHz Message Frequency

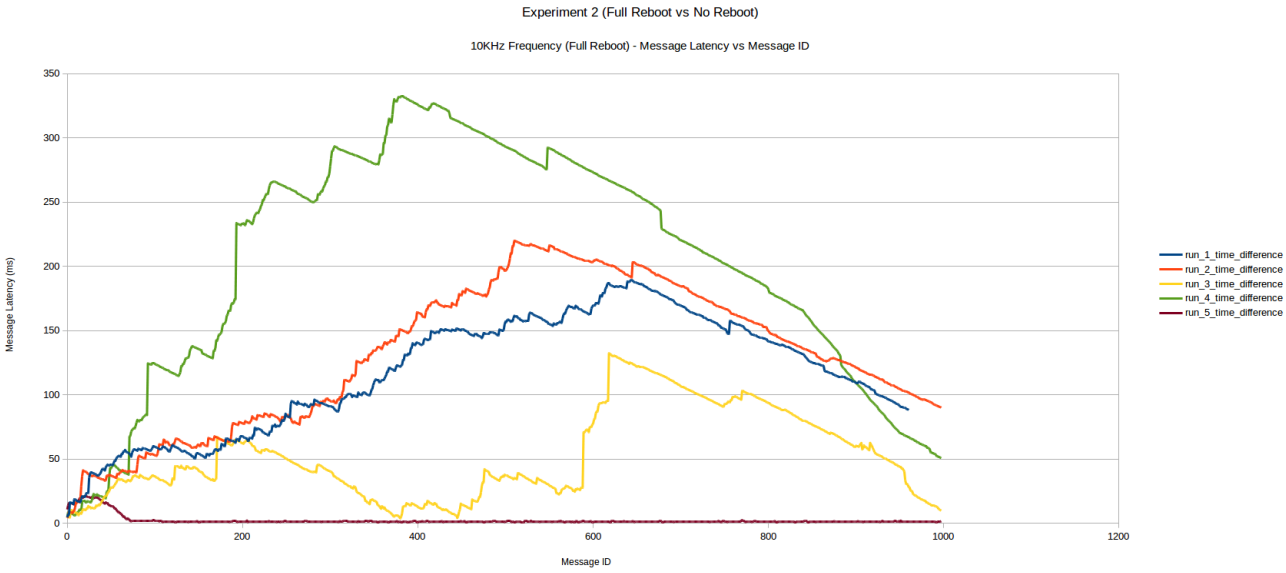

Figure A.8: Experiment 2 - Full Reboot 10KHz Message Frequency

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)

We generate Erdós-Rënyi random graphs $G(n, p)$ where n is the number of vertices and each edge is included in the graph with probability p independent from every other edge. It produces a random graph in DIMACS format with vertices numbered 1 to n inclusive. It can be run from the command line as follows to produce a clq file

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