



University | School of
of Glasgow | Computing Science

Title of project placed here

Chongbin Ren

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

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

Date of submiss

Abstract

abstract goes here

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: _____

Acknowledgements

Contents

1	Introduction	5
1.1	Context	5
2	Background	6
2.1	Robotics Overview	6
2.1.1	Multi-Robot Systems	6
2.1.2	Raspberry Pi	7
2.1.3	SunFounder Raspberry Pi Smart Video Car	7
2.2	Robot Software and Operating System	8
2.2.1	ROS	8
2.2.2	ROS 2	8
2.2.3	Ubuntu	9
2.3	Communication on ROS vs ROS2	9
2.3.1	Star Network Topology	9
2.3.2	TCP and UDP in ROS	10
2.3.3	Data Distribution Service (DDS) on ROS2	11
3	Installation and configuration	12
3.1	Ubuntu and ROS system installation	12
3.1.1	Ubuntu installation	12
3.1.2	ROS1 Installation	12
3.1.3	ROS2 Installation	13

3.2 Router network configuration	13
3.3 Communication between different ROS hosts	13
4 Experiment and Analysis	14
4.1 Experiment 1: Investigating different ROS1 version message latency with previous experiment	14
4.2 Experiment 2: ROS1 and ROS2 message latency analysing	18
5 Conclusion	21
5.1 Summary	21
5.2 Future work	21
A First appendix	22
A.1 ROS1 installation command steps	22
A.2 ROS2 installation steps	23
B Second appendix	24

Chapter 1

Introduction

1.1 Context

Chapter 2

Background

2.1 Robotics Overview

Robotics is a branch of engineering that involves the conception, design, manufacture, and operation of robots[1]. Robots are machines that can be used to do jobs. Some robots can do work by themselves. Other robots must always have a person telling them what to do. Todays robotic systems generally consist of many small autonomous systems working together to form a coherent whole[2].

This chapter aims to provide the background knowledge required to understand the results presented in this paper. Section 2.1 discusses the robotics overview and hardware in this experiment. Section 2.2 describe the operating system and robotic middleware when constructing a robotic system. Next, section 2.3 introduces network topology and communication protocol used by different robotic operating system.

2.1.1 Multi-Robot Systems

Given the rate of technological advancements over the past decade, the adoption of robotics has become increasingly widespread. In most domains, the performance of multiple robots outperforms a single robot, both with respect to cost, efficacy and domain potential [3]. This has led to multirobot systems becoming a key area of research within the field of robotics in general [4].

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. These multirobot are distinct from a multi-agent system in which individual nodes are generally stationary, as each agent in a multi-robot system is mobile[5]. Mobile robotics has been made more possible recently by advances in battery and wireless communication technologies[6].

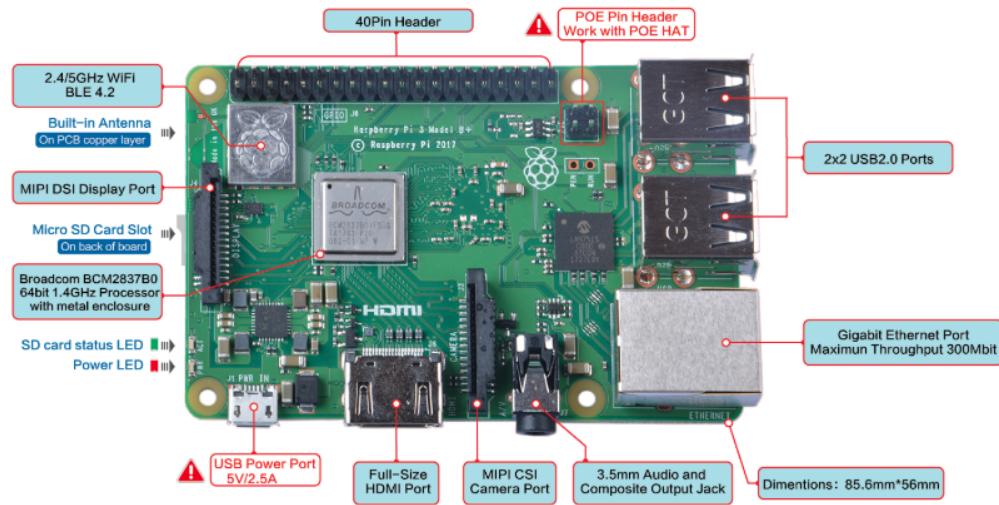
2.1.2 Raspberry Pi

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries.[7]

The Raspberry Pi is small sized like credit card, single board computer. Its capable of doing everything that a desktop computer can do, from browsing the internet and installing applications, even playing games. It runs Linux and other available operating systems from a micro SD card. With the addition of a keyboard, mouse and micro USB power supply it can be connected to a screen monitor and used as a fully functioning desktop computer. In short, the Raspberry Pi is a small computer with relatively limited memory and performance, but it contains all needed functions.

The first Raspberry Pi was released in 2012, since then it has been used in a vast range of projects due to its low cost, portability, programmability and both wired and wireless connectivity[8]. Several generations of Raspberry Pis have been released. The Raspberry Pi 3 and Pi Zero W (wireless) are equipped with 2.4 GHz WiFi 802.11n (150 Mbit/s) and Bluetooth 4.1 (24 Mbit/s) based on the Broadcom BCM43438 FullMAC chip with no official support for monitor mode. The Raspberry Pi 3B+ features dual-band IEEE 802.11b/g/n/ac WiFi, Bluetooth 4.2, and Gigabit Ethernet[8].

Raspberry Pi 3B+ is used in this paper. The structure picture is as follows:



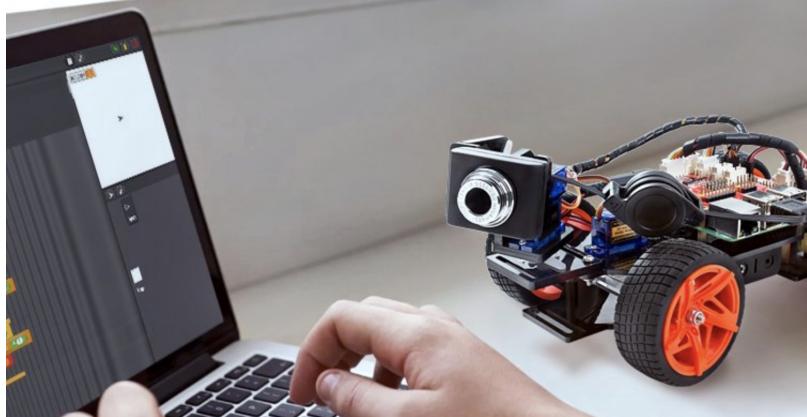
2.1.3 SunFounder Raspberry Pi Smart Video Car

SunFounder is a technology company focused on Raspberry Pi and Arduino open source community development. The Pi Car-S Robot is a smart car which work based on a Raspberry Pi.

The raspberry pi car comes with three sensor modules including ultrasonic obstacle avoidance, light follower, and line follower. The car robot has durable and shatterproof plate and new style Servo, the MAX torque of the clutch gear digital servo is up to 1.6KG[9]. It rotates from 0-180 degrees.

Python code is provided for the car, and users can also program and debug it with Dragit, a Snap-based graphical interface, by just simple dragging and dropping the code blocks for more complex

functions. A user can learn the programming conveniently on how to control the car.



2.2 Robot Software and Operating System

The following sections introduce the Robot software and operating system that used in this project. The version of ROS and ROS2 is under development and changes frequently these years.

2.2.1 ROS

Robot Operating System (ROS or ros) is robotics middleware (i.e. collection of software frameworks for robot software development). ROS is not an operating system. It is a set of software libraries and tools that help people build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools and its all open source[11]. ROS is a software framework meant to allow people to write applications which operate robotic hardware. As its most fundamental level, it is an abstraction layer - offering hardware abstraction.

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 a common interface design so that no matter what hardware is producing the data, the results are distributed in a consistent manner[10].

Robot Operating System is mainly composed of 2 things[12]:

1. A core (middleware) with communication tools.
2. A set of plug and play libraries.

2.2.2 ROS 2

ROS2, which is a new version of ROS that is under heavy development. ROS2 is the next generation robot operating system, and is actively being developed to fully replace ROS in the near future. The goal of the ROS2 project is to adapt to previous changes, leveraging what is great about ROS and improving what is not[13].

A further reason to build ROS 2 is to take advantage of the opportunity to improve our user-facing APIs. A great deal of the ROS code that exists today is compatible with the client libraries as far back as the 0.4 Mango Tango release from February 2009. That's great from the point of view of stability, but it also implies that we're still living with API decisions that were made several years ago, some of which we know now to be not the best[14].

There are some obvious differences between ROS and ROS2:

1. ROS1 is targeting Python 2, but ROS2 requires at least Python version 3.5.
2. Catkin make is gone: Catkin has been replaced by colcon. This new building system is in essence a universal building system for the ROS ecosystem.
3. ROS2 uses rclpy as Python client library which replaces roscpp in ROS.

2.2.3 Ubuntu

Ubuntu is a complete Linux operating system, which is an open source Debian-based Linux distribution. Ubuntu is now the most used Linux operating system, both for desktops and servers.

Ubuntu is built on Debian's architecture and infrastructure, and comprises Linux server, desktop and discontinued phone and tablet operating system versions.[15] Ubuntu releases updated versions predictably every six months, and each release receives free support for nine months[16]. Downloading, installing, and using Ubuntu Linux doesn't cost a penny. Installing Ubuntu does not need high-end system requirements. So it is an ideal operating system for Raspberry Pi, as the Raspberry Pi has limited memory CPU performance.

Ubuntu is available in a number of different flavours, each coming with its own desktop environment. Ubuntu MATE takes the Ubuntu base operating system and adds the MATE Desktop. The MATE Desktop is one such implementation of a desktop environment and includes a file manager which can connect you to your local and networked files, a text editor, calculator, archive manager, image viewer, document viewer, system monitor and terminal[17]. And Ubuntu MATE offers specific downloading/installing package for Raspberry Pi. This can be downloaded from the official site.

Furthermore, Ubuntu is also the recommended operating system in ROS. ROS offers more support for a few Ubuntu versions. There is another operating system called Raspbian which is widely installed on Raspberry Pi. However, Raspbian is not well supported by ROS and it may occur many problems when installing ROS in Raspbian. Therefore, Ubuntu is used in this experiment for Raspberry Pi as operating system.

2.3 Communication on ROS vs ROS2

2.3.1 Star Network Topology

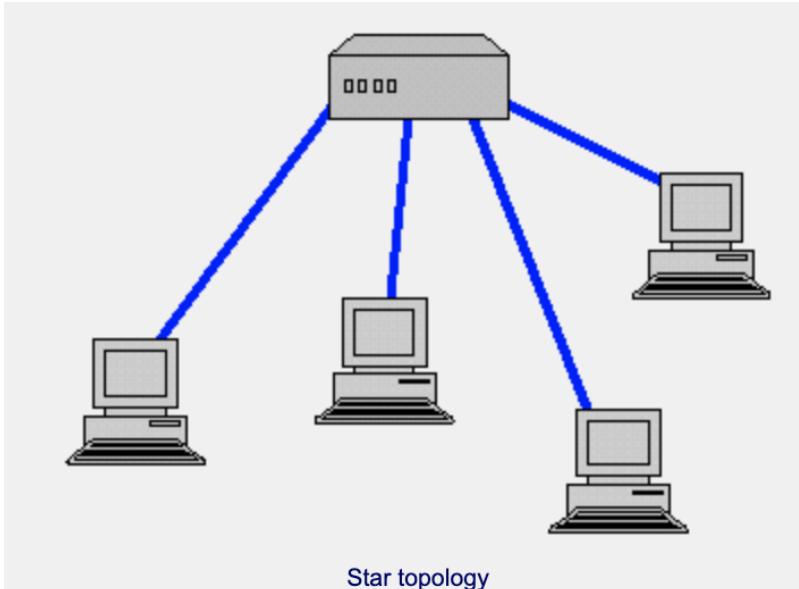
A star topology is a topology for a Local Area Network (LAN) in which all nodes are individually connected to a central connection point, like a hub or a switch. A star takes more cable than e.g. a bus, but the benefit is that if a cable fails, only one node will be brought down[18].

All traffic emanates from the hub of the star. The central site is in control of all the nodes attached to it. The central hub is usually a fast, self contained computer and is responsible for routing all traffic to other nodes. The main advantages of a star network is that one malfunctioning node does not affect the rest of the network[19]. However this type of network can be prone to bottleneck and failure problems at the central site.

It is the most common network topology in real life, as it is easy to install, scalable and easy to troubleshoot. At the centre of the network sits the multifunctional wireless router providing the functionality of a wireless access point (WAP), a modem, a switch and a router.

Star network topology is also used in this experiment for testing message latency communicating in different devices with a center router.

The network graph is as follows:



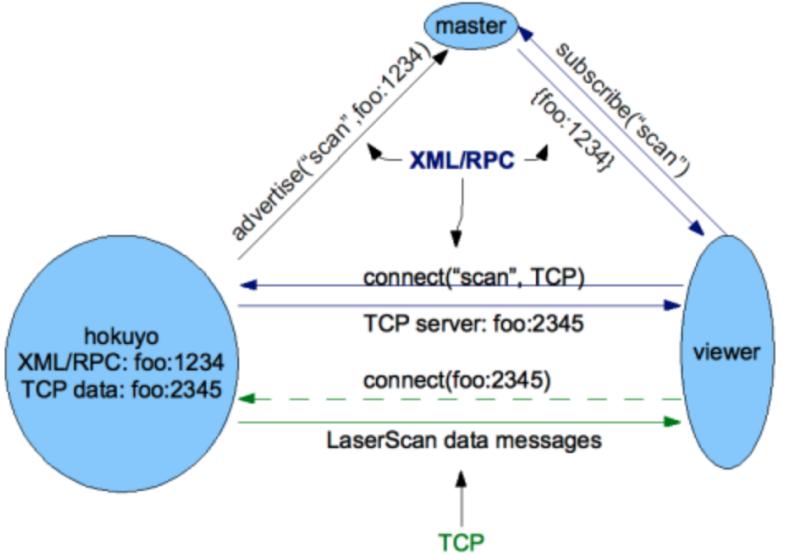
2.3.2 TCP and UDP in ROS

ROS currently supports TCP/IP-based and UDP-based message transport. The TCP/IP-based transport is known as TCPROS and streams message data over persistent TCP/IP connections. TCPROS is the default transport used in ROS and is the only transport that client libraries are required to support. The UDP-based transport, which is known as UDPROS and is currently only supported in roscpp, separates messages into UDP packets. UDPROS is a low-latency, lossy transport, so is best suited for tasks like teleoperation.

ROS nodes negotiate the desired transport at runtime. For example, if a node prefers UDPROS transport but the other Node does not support it, it can fallback on TCPROS transport[20]. This negotiation model enables new transports to be added over time as compelling use cases arise.

TCPROS is a transport layer for ROS Messages and Services. It uses standard TCP/IP sockets for transporting message data. Inbound connections are received via a TCP Server Socket with a header containing message data type and routing information[21].

Therefore, in the experiment, ROS use TCPROS to communicate between different machines by default. The TCPROS communication structure is as follows:



2.3.3 Data Distribution Service (DDS) on ROS2

When exploring options for the next generation communication system of ROS, the initial options were to either improve the ROS 1 transport or build a new middleware using component libraries such as ZeroMQ, Protocol Buffers, and zeroconf (Bonjour/Avahi)[22]. However, in addition to those options, during the research, one middleware that stood out was DDS.

The benefit of using DDS, is that there is much less code to maintain and the behavior and exact specifications of the middleware have already been distilled into documentation. In addition to system-level documentation, DDS also has recommended use cases and a software API. With this concrete specification, third parties can review, audit, and implement the middleware with varying degrees of interoperability. This is something that ROS has never had, besides a few basic descriptions in a wiki and a reference implementation[23].

ROS2 using DDS as message transport service. DDS provides a publish-subscribe transport which is very similar to ROSs publish-subscribe transport. It allows any two DDS programs to communicate without the need for a tool like the ROS master[24]. This makes the system more fault tolerant and flexible.

The default implementation of DDS is over UDP, DDS implementations use (UDP) broadcast traffic for peer discovery. By default, when sending message, the ROS2 communication will broadcast to the local subnet (ie: all IPs in the same range).

Chapter 3

Installation and configuration

3.1 Ubuntu and ROS system installation

3.1.1 Ubuntu installation

1. Download the latest version image of Ubuntu Bionic (64-bits) for Raspberry Pi from official website.
2. Write the image to Raspberry Pi 3B+ by using SD card.
3. Run it with screen and configure all necessary dependency and install "openssh" server for remote control.

Issues and Solution

1. Other operating system like Raspbian 10(buster) is not well supported by ROS system. There are some outdated dependencies which is needed in installing ROS. Therefore, there are many bugs and manually installation steps to solve when user want to install ROS in Raspbian operating system. Then the Ubuntu is recommend operating system as official website said.
2. Find the 64-bits version operating system because ROS2 only can run on 64-bits operating system.

3.1.2 ROS1 Installation

1. Follow the step details at first appendix A1 ROS1.
2. Configure Ubuntu repositories to allow "restricted," "universe," and "multiverse."
3. After installation, use *roscore* to test whether ROS system exist.

3.1.3 ROS2 Installation

1. Installing ROS2 via Debian Packages.
2. Follow the step details at first appendix A2 ROS2.
3. Using *ros2 run XXpackage XXfile* to test whether ROS2 can run successfully.

3.2 Router network configuration

1. Each Raspberry Pi should connect to the same LAN.
2. Assign static IP address to each Raspberry Pi for ssh remote control and management.
3. Using one computer to join this LAN for controlling the all Raspberry Pi.
4. Remote control other Raspberry Pi by ssh using *ssh hostname@ip address* with password.

3.3 Communication between different ROS hosts

Change the host file

Each host configuration file can be modify by *sudo vim /etc/hosts*. Add ip address and hostname in */etc/hosts* file for Raspberry Pi. Each ip address respond to one hostname which you give. Example hosts file is in the Appendix.

Running Talker and Listener

Enter ROS system in every Ubuntu system and ssh to com1. Then ping com1 and com2. If ping successfully meaning the network is fine, otherwise check the router and network problem.

The host name and ip address is respond to each Raspberry Pi, but the Master URI should be identical. Create a *.sh* file to export environment paths. See the appendix

Configure the file and execute this file in every Raspberry Pi host. Then run the talker and listener python file seperately in different hosts to check if listener can receive messages from talker.

Issues and Solution

Make sure setting the ROS hostname and IP address correctly and must explicitly set it. Otherwise the listener cannot receive talker because the host cannot find it in the network. The Talker will send messages continuously but listener can not receive these messages.

Check the hostname and ip address setting can use:

```
echo $ROS_HOSTNAME  
echo $ROS_IP
```

Chapter 4

Experiment and Analysis

4.1 Experiment 1: Investigating different ROS1 version message latency with previous experiment

Objective

This experiment is aiming to investigate the message sending performance in different ROS version with different OS platform by using Wi-Fi network connection through a router. This experiment was executed by Issac in the previous and I execute this experiment in different software and hardware and compare the results to the previous one[25].

Rationale

In order to acquire a detailed understanding of the performance characteristics of ROS communication channels, this experiment use two ROS hosts to communicate with each other and get the message latency results in different message frequency. I reproduced Issac's experiment to see what is the difference between my experiment and the previous one then analyze the reason to cause these differences.

Procedure

1. Using 'roscore' to start the ROS master node.
2. Start one publisher node on one host and send different frequency message to a echoer node on another host by using Wifi network.
3. Run the python code which send timestamped string messages from the publisher node to the echoer node. Then the echoer node receive message and send it back to publisher node.
4. The master node will receive the message and then record message id and time in sending and receiving into a file.
5. Message frequency is from 200 to 2000Hz (200,400,600...2000)which is same with the previous experiment.
6. Every frequency will be recorded 3 times with a range of message frequencies to obtain averaged results for each message frequency.

7. Running this experiment in different ROS1 version and operating system.

Hardware Configuration

4 Raspberry Pi 3B+ (1.4GHz 64-bit quad-core ARM Cortex-A53 CPU , 1GB RAM LPDDR2)
TP-Link 150 Mbps Router

Software Configuration

Ubuntu 18.04 Bionic (64-bit)
ROS Melodic (Latest, Released May, 2018)

Ubuntu 16.04 Xenial (32-bit)
ROS Kinetic Kame (Released May, 2016)

ROS Melodic is installed on Ubuntu 18 Bionic; ROS Kinetic is installed on Ubuntu 16 Xenial.

Hypothesis

1. The message latency should be like the previous experiment results. Although I use different hardware and software, but the message latency should not have very obvious difference with the previous experiment.
2. Different message frequency may perform different latency. For example, lower message frequency can have lower message latency.

Results

The previous experiment from Issac is Figure 4.1. I plot six different frequency instead of Issac 10 frequency because it may make the graph more clear to check out. Figure 4.2 is message latency from ROS kinetic on Ubuntu Xenial and the Figure 4.3 is ROS Melodic on Ubuntu Bionic.

1. The message latency from previous experiment by Issac is all almost under 20ms lower latency like 200Hz and 400Hz are very low latency and perform better than higher frequency like 1800Hz and 2000Hz.
2. In my experiment message latency is high and very unstable in ROS kinetic on Ubuntu Xenial 32bits. Message latency can be up to hundreds millisecond which is ten times than previous experiment message latency.
3. Then in ROS Melodic on Ubuntu Bionic 64bits, the message latency is low and the result is very like the previous Issac's one. In lower frequency 200Hz and 400Hz show under 10ms latency and in higher message frequency, latency can be vary at 15-20Hz.

Conclusion

Firstly, the ROS version and operating system version can have influence on the communication performance between different hosts. In my experiment, ROS kinetic on Ubuntu 16 shows very unstable message latency. It performs worse than using ROS Melodic on Ubuntu 18.

Secondly, in my experiment ROS melodic Ubuntu 18 can be more stable when sending message than previous experiment ROS kinetic on Raspbian. Because there is no obvious spike on ROS melodic Ubuntu 18. However, Router could also be a factor that have slightly impact on message latency.

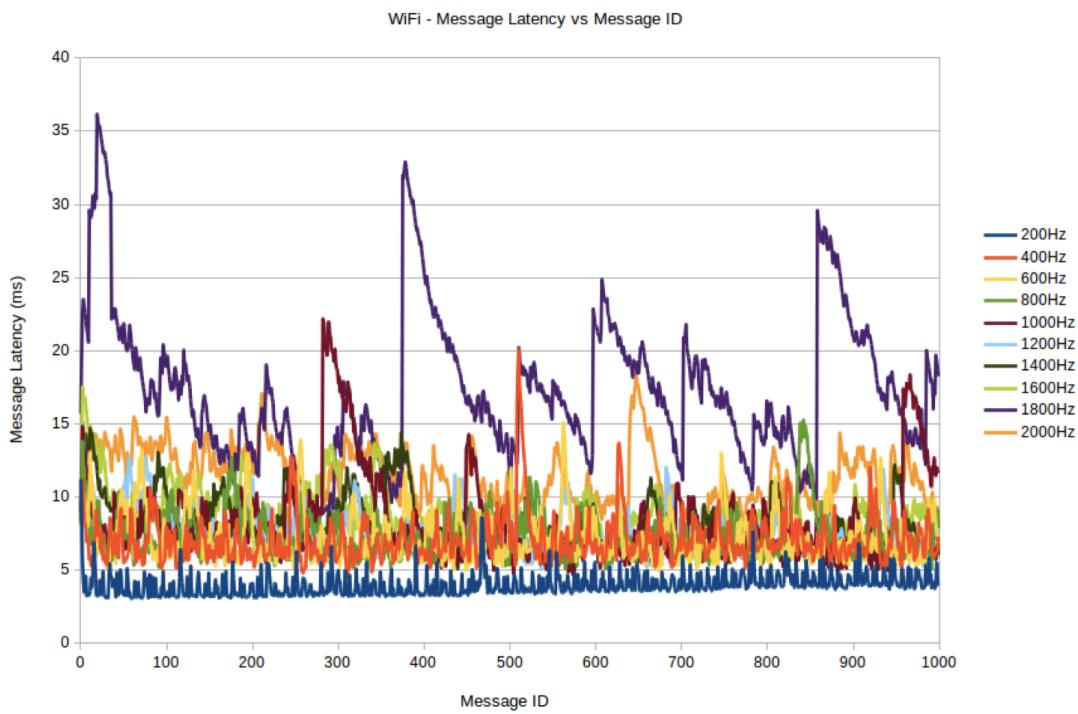


Figure 4.1: Message latency for WiFi frequency in Issac experiment

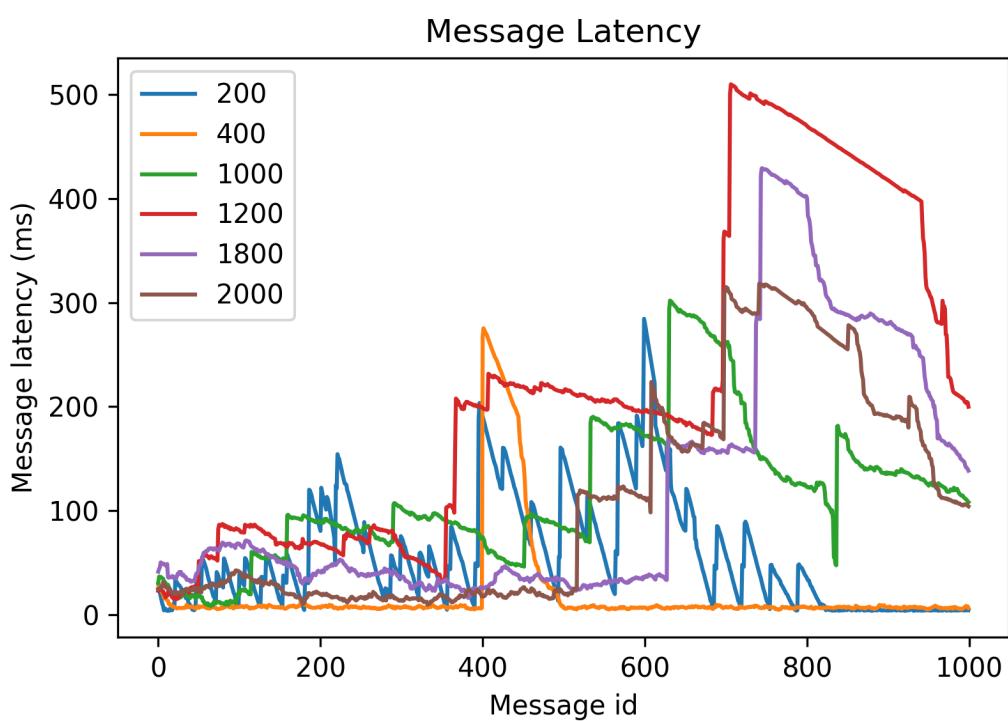


Figure 4.2: Message latency for ROS kinetic on Ubuntu Xenial 32-bit

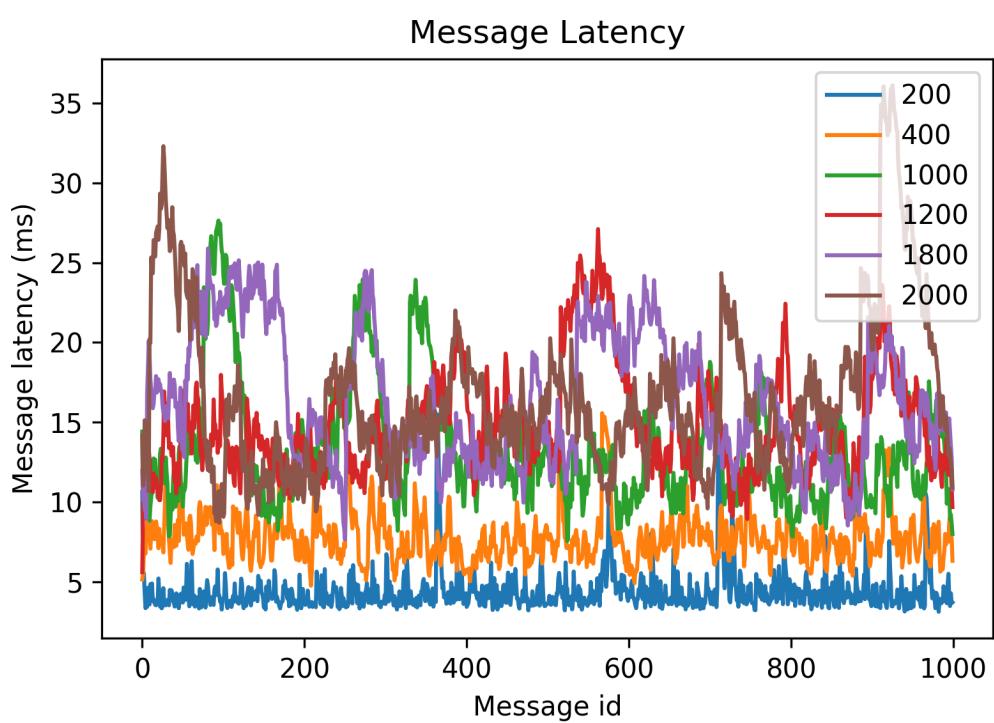


Figure 4.3: Message latency for ROS Melodic on Ubuntu Bionic 64-bit

4.2 Experiment 2: ROS1 and ROS2 message latency analysing

Objective

The aim for this experiment is to analyse message latency performance in communication between ROS1 and ROS2 version with all other same condition. Then collect different message frequency data and analyse data to measure communication performance.

Rationale

ROS2 has more improvement comparing with ROS1 and it use different api and protocol when sending message. So in this experiment, I am going to test the message latency in different frequency and analysing what happen in different ROS version.

Procedure

1. There is no master node on ROS2 so you can directly start one node on one Raspberry Pi and send different frequency message to another host through wifi router.
2. Running python code which send string timestamped messages from the sender host to the echoer host. Then the echoer host send message back to sender host.
3. The sender node will receive the message and then record message id and time elapsed in communication into a file.
4. Message frequency is from low frequency (1Hz and 10Hz), middle frequency(100 and 200Hz) and high frequency (1000 and 2000Hz). Every frequency will be run 3 times and get a mean time results.
5. Using "ros2 run" in ROS2 run python3 code which also send and receive message and then record the results data.
6. Record all result data and plot by using matplotlib python in Jupyter notebook.

Hardware Configuration

4 Raspberry Pi 3B+ (1.4GHz 64-bit quad-core ARM Cortex-A53 CPU , 1GB RAM LPDDR2)
D-link Wireless AC1200 Dual Band Gigabit Router

Software Configuration

ROS1 Melodic (Latest, Released May, 2018) on Ubuntu 18.04 Bionic (64-bit)
ROS2 Crystal Clemmys (Released December 2018) on Ubuntu 18.04 Bionic (64-bit)

Hypothesis

The hypothesis in this experiment can be that ROS2 communication performs better than ROS1 because it is new ROS version and improve in communication between different ROS version.

Results

1. From the experiment, overall, in Figure 4.4 and 4.5, ROS1 is less stable than ROS2 from low frequency to high frequency. All message latency is under 25ms in ROS1 comparing under 10ms in ROS2.
2. In low frequency, in Figure 4.6 and 4.7, 1Hz and 10Hz comparing, ROS1 latency can vary between 4ms to 15ms. However, in ROS2, message latency can vary from 2ms to 10ms.
3. Then in middle frequency, in Figure 4.8 and 4.9, from 100Hz to 1000Hz, ROS1 message is increasing with higher message frequency. However, ROS2 shows a stable and low message latency under 8ms.

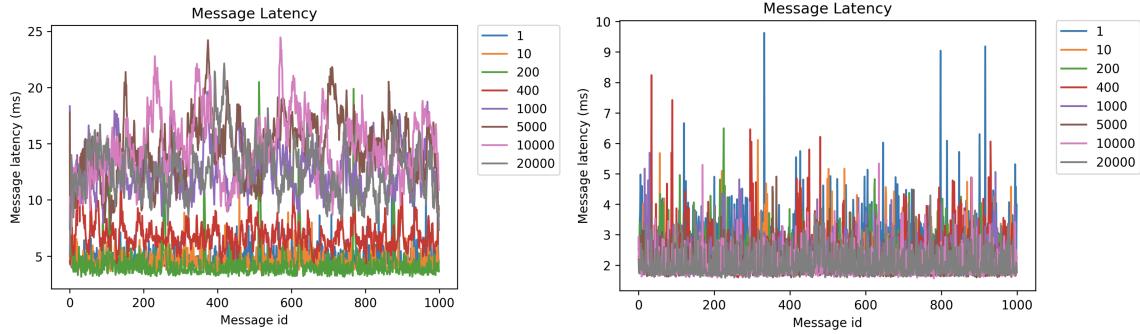


Figure 4.4: ROS1 all frequency

Figure 4.5: ROS2 all frequency

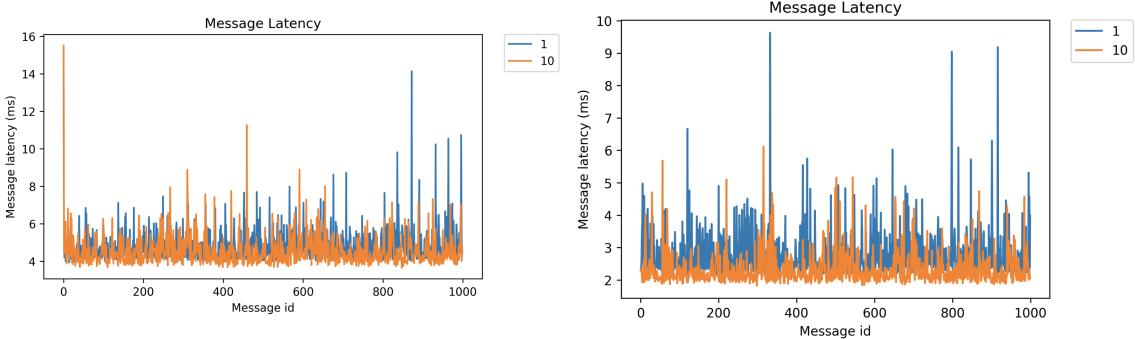


Figure 4.6: ROS1 1Hz and 10Hz

Figure 4.7: ROS2 1Hz and 10Hz

4. In High frequency, in Figure 4.10 and 4.11, 1000Hz to 20000Hz, message latency can be unstable but looks the same vary from 5ms to 20ms in ROS1, however, the message latency can be low in ROS2 high frequency which is under 5ms.

Conclusion

In order to acquire a detailed performance characteristics of ROS1 and ROS2 channels, this experiment uses Raspberry Pi 3B+ which installed operating 64-bit ubuntu system. Then communicating by the same router with ROS1 and ROS2 separately. This can guaranteed the same variable and validity of this experiment.

To conclude, ROS2 can show more stable and less message frequency than ROS1 with the same hardware and operating system. ROS2 can show less message latency in different frequency than

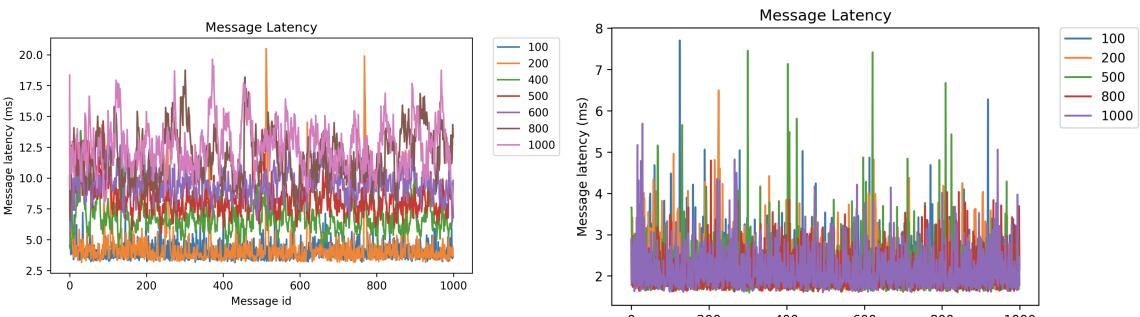


Figure 4.8: ROS1 100Hz and 200Hz

Figure 4.9: ROS2 100Hz and 200Hz

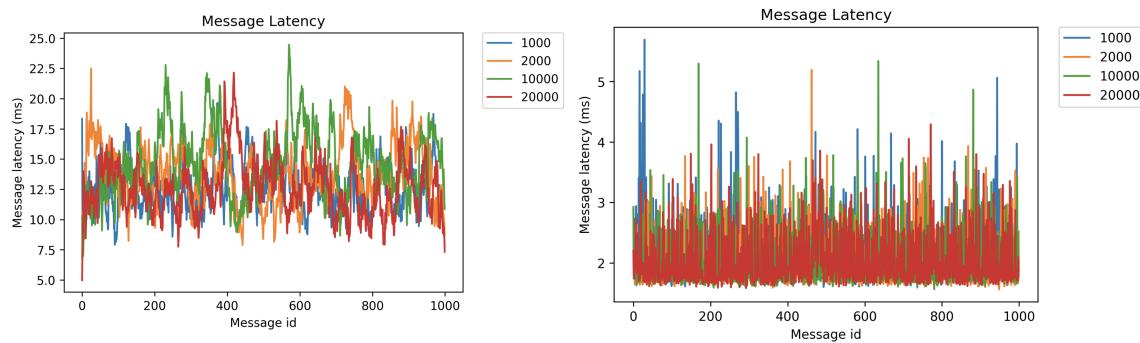


Figure 4.10: ROS1 1000Hz and 2000Hz

Figure 4.11: ROS2 1000Hz and 2000Hz

ROS1. Due to using DDS in ROS2, a more advanced communication protocol so the message communication performance is better than ROS1.

Chapter 5

Conclusion

5.1 Summary

5.2 Future work

Appendix A

First appendix

A.1 ROS1 installation command steps

```
# Install system

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
main" > /etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
sudo apt update
sudo apt install ros-melodic-ros-base

# Install tools and dependencies

sudo rosdep init
rosdep update
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
sudo apt install python-rosinstall python-rosinstall-generator python-wstool
build-essential

# Start ROS service and create workspace

roscore
create catkin folder
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
source devel/setup.bash
```

A.2 ROS2 installation steps

```
# Install
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
sudo apt update && sudo apt install curl gnupg2 lsb-release
curl -s
https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
install
export CHOOSE_ROS_DISTRO=crystal
sudo apt update
sudo apt install ros-$CHOOSE_ROS_DISTRO-ros-base

# Environment setup
sudo apt install python3-argcomplete
source /opt/ros/$CHOOSE_ROS_DISTRO/setup.bash
echo "source /opt/ros/$CHOOSE_ROS_DISTRO/setup.bash" >> ~/.bashrc
sudo apt install python3-colcon-common-extensions

# create workspace
mkdir -p ~/ros2_example_ws/src
cd ~/ros2_example_ws
```

/etc/hosts

```
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

192.168.1.11 com1
192.168.1.12 com2
```

config.sh

```
# /bin/sh
export ROS_HOSTNAME=com1
export ROS_IP=192.168.1.11
export ROS_MASTER_URI=http://com1:11311/
```

Appendix B

Second appendix

```
~\ > ssh comros4@192.168.1.4
comros4@192.168.1.4's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-1032-raspi2 aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

286 packages can be updated.
151 updates are security updates.

Last login: Tue Aug  6 22:29:58 2019 from 192.168.1.51
comros4@comros4-desktop:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  catkin_ws
```

Figure B.1: ubuntu

Bibliography

- [1] Sandra May. What Is Robotics? https://www.nasa.gov/audience/forstudents/k-4/stories/nasa-knows/what_is_robotics_k4.html. Nov. 9, 2009
- [2] B. Bauml and G. Hirzinger. Agile Robot Development (aRD): A Pragmatic Approach to Robotic Software. In 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 37413748, Oct 2006.
- [3] Avinash Gautam and Sudeept Mohan. A review of research in multi-robot systems. In 2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS), pages 15. IEEE, Aug 2012.
- [4] Pedro U. Lima and Luis M. Custodio. Multi-Robot Systems. In Innovations in Robot Mobility and Control, pages 164. Springer, Berlin, Heidelberg, Aug 2005.
- [5] Zhi Yan, Nicolas Jouandea, and Arab Ali Cherif. International Journal of Advanced Robotic Systems, 10(12):399, 2013.
- [6] Deborah Estrin, David Culler, Kris Pister, and Gaurav Sukhatme. Connecting the physical world with pervasive networks. IEEE pervasive computing, 1(1):6263, 2002.
- [7] Cellan-Jones. A15 computer to inspire young programmers. BBC news. 5 May 2011
- [8] Gibbs, Samuel. Raspberry Pi becomes best selling British computer. The Guardian. Retrieved 28 December 2016.
- [9] SunFounder PiCar-S Kit V2.0 page. <https://www.sunfounder.com/picar-s-kit.html>.
- [10] Isaac Jordan. Robotic Middleware. https://github.com/Sheepzez/ros-multirobot-evaluation/tree/master/experiments/message_latency/no_echo_delay_no_disk_write. 2017-03-18.
- [11] ROS Melodic Morenia. wiki.ros.org. Retrieved 10 June 2018.
- [12] The Robotics Back-End Home page. <https://roboticsbackend.com/what-is-ros>. 2019
- [13] ROS2 Overview Home page. <https://index.ros.org/doc/ros2/>

- [14] Brian Gerkey. Why ROS 2?. http://design.ros2.org/articles/why_ros2.html
- [15]"Ubuntu and Debian". Ubuntu.com. Canonical Ltd. Retrieved 14 December 2013.
- [16]"Ubuntu and Debian". Ubuntu.com. Canonical Ltd. Retrieved 14 December 2013.
- [17] What is Ubuntu MATE?. Ubuntu MATE Team. <https://ubuntu-mate.org/what-is-ubuntu-mate/>. 2019
- [18] Star topology. <http://www.telecomabc.com/s/star.html>. 2015
- [19] Star topology. Cisco Networking Academy 2014.
- [20] Topics. <http://wiki.ros.org/Topics>. 2019-02-20
- [21] TCPROS. <http://wiki.ros.org/ROS/TCPROS>. 2013-04-15
- [22] ROS on DSS. https://design.ros2.org/articles/ros_on_dds.html
- [23] Why Consider DDS. ROS on DSS. https://design.ros2.org/articles/ros_on_dds.html
- [24] Where did DDS come from. ROS on DSS. https://design.ros2.org/articles/ros_on_dds.html
- [25] Issac Jordan. Experiment https://github.com/Sheepzez/ros-multirobot-evaluation/tree/master/experiments/message_latency/bag-sender-receiver. Accessed: 2017-03-18.
- [26] Drone (UAV) <https://internetofthingsagenda.techtarget.com/definition/drone>. July 2019
- [27] Sunfounder Smart Video Car Kit for Raspberry Pi. https://github.com/sunfounder/Sunfounder_Smart_Video_Car_Kit_for_RaspberryPi.
- [28] ROS 2 Overview. <https://index.ros.org/doc/ros2/>.