A communication architecture for UAV swarms

A thesis submitted in fulfilment of the requirements

for the degree of Master of Technology

by

Sai Aditya Chundi

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

May 2019

Certificate

It is certified that the work contained in this thesis entitled ”A communication architecture for UAV swarms” by ”Sai Aditya Chundi” has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

Ketan Rajawat

Professor

Department of Electrical Engineering

Indian Institute of Technology Kanpur

May 2019

Abstract

UAVs have received considerable attention in the last decade, both in industry and academia. Potential applications are wide and varied, encompassing both military and domestic spaces. Search and rescue missions during disasters, environmental monitoring and surveillance, precision agriculture and farming are some of the domestic applications, while the scope of their usage in military space can be readily appreciated.

Communication is a critical component in realizing swarms of UAVs. There are two aspects of the communication architecture while considering UAV swarms. While there needs to be a reliable communication between the UAVs and a ground station, communication between individual UAVs is essential in enabling a distributed architecture for swarm applications. Wireless ad-hoc networks offer an appealing solution for inter UAV communication. While there have been works which used 802.11 based ad-hoc networks for the communication in multi UAV setups, these are short range links which are not suitable for the long range link between UAVs and the ground station. On the other hand, the commonly used 900 Mhz based radios for the communication between UAVs and the ground station in long range, are not well suited for inter UAV communication.

In this work, we present a novel communication architecture for UAV swarms, which combines both the long range and short range architectures. Moreover, our communication architecture is based on Robot Operating System(ROS), which ensures that any distributed application can be easily integrated into, and extended by the capabilities of ROS.

Acknowledgements

First and foremost, I would like to express my sincere gratitude and appreciation to my supervisor Dr. Ketan Rajawat. I am forever indebted to him for his expert guidance, continuous encouragement and steadfast belief in me, when even I could not believe in myself. I feel extremely fortunate and am grateful to him for giving me an oppurtunity to work in this area, during which I had a fantastic learning experience.

I particularly offer my gratitude to Mohan and Lavish, who have been extremely supportive of me. The discussions we have had on diverse topics, throughout my stay at SPIN lab are invaluable to me. I would also like to thank Sanku and Anway with whom I have had a great pleasure of working. I also offer my thanks to all my peers in SPIN lab.

I fondly cherish my memories in IITK with my friends Suraj and Venkatesh, without whom these past two years would have much more difficult.

Last but not least, I express my profound gratitude to my parents, siblings and siblings-in-law, without whose unconditional love and unwavering support, this work would not have been possible.

Contents

Certificate i

Abstract ii

Acknowledgements iii

Contents iv

List of Figures vi

1 Introduction 1
- 1.1 Rise of Unmanned Aerial Vehicles . . . 1
- 1.2 The Challenge of Communications in UAV swarms . . . 2
- 1.3 Motivation and the Current Work . . . 4
- 1.4 Organizaiton of the thesis . . . 5

2 The Mesh Architecture 6
- 2.1 IEEE 802.11s . . . 6
  - 2.1.1 Routing in 802.11s . . . 7
  - 2.1.2 open80211s . . . 9
- 2.2 Implementation . . . 9
  - 2.2.1 OpenWrt . . . 10
  - 2.2.2 Hardware . . . 11
    - 2.2.2.1 Tp-Link WR902AC . . . 11
    - 2.2.2.2 Alfa Tube 2H . . . 12
  - 2.2.3 Network Configuration . . . 13
    - 2.2.3.1 LUCI . . . 13
    - 2.2.3.2 802.11s mesh configuration . . . 13
    - 2.2.3.3 Routing protocols . . . 15
    - 2.2.3.4 Mesh networks among the host machines . . . 15
- 2.3 Integration with ROS . . . 16
  - 2.3.1 Multimaster_fkie . . . 17

2.3.1.1 master_discovery . . . 17
2.3.1.2 master_sync . . . 18

3 Long Range Communication Architecture 19
3.1 Hardware . . . 19
3.1.1 RFD900X . . . 20
3.1.1.1 Peer to Peer Firmware . . . 21
3.1.1.2 Synchronous Mesh Firmware . . . 22
3.1.1.3 Asynchronous Mesh Firmware . . . 23
3.2 Architecture and Integration with ROS . . . 24
3.2.1 ROS . . . 25
3.2.1.1 serialros . . . 25
3.2.2 Static Leader . . . 26
3.2.3 Dynamic Leader . . . 27
3.2.3.1 link_status . . . 28
3.2.3.2 get_leader service . . . 29
3.2.3.3 uav.py . . . 29
3.2.3.4 RFD configuration . . . 30

4 Conclusion 32
4.1 Future Work . . . 33

5 Preliminaries 34

Bibliography 35

List of Figures

1.1 Possible network configurations for communications in UAVs 3

2.1 A comparison of 802.11 wlan and 802.11s mesh networks 7
2.2 open80211s stack on Linux kernel 9
2.3 logo of OpenWrt 9
2.4 An ssh shell on an openwrt flashed router 10
2.5 Tp-Link WR902AC 11
2.6 Alfa Tube 2H 12
2.7 LUCI web interface for configuring openwrt routers 14
2.8 Mesh architecture with bridging of wireless and ethernet interfaces in routers 16
2.9 An illustration of ROS single master and multimaster setups 17

3.1 RFD900x along with antennas and FTDI cables 20
3.2 modems in peer to peer configuration 21
3.3 modems in synchronous mesh configuration 22
3.4 modems in asynchronous mesh configuration 23
3.5 serialros, a serial interface to ROS 25
3.6 Static Leader Architecture 26
3.7 Dynamic Leader Architecture 28
3.8 Flowchart depicting changing mode of a UAV 30

5.1 Components in a general autonomous UAV 34

I dedicate this work to my parents and teachers

Chapter 1

Introduction

1.1 Rise of Unmanned Aerial Vehicles

Unmanned aerial vehicles or UAVs have become ubiquitous in the past decade, both in research and industry. A myriad of applications involving UAVs in diverse fields has made them quite popular. Potential domestic applications include Environmental monitoring and surveillance [11], traffic management [25], remote sensing [29], precision agriculture and farming [10], disaster management [12], to name a few. The use of UAVs for defence purposes is only poised to grow in the next decade. It is not hard to see the appeal of UAVs in the defence sector with applications ranging from simple surveillance and reconnaissance missions to offensives like *search and destroy* missions and targeted hits.

Much of the rise of this interest in UAVs can be attributed to associated advances in robotics, largely driven by the progress in robust and cheap sensors and communication technology. The emergence of scalable and extensible software architectures like the Robot Operating System(ROS), which enables easy integration of various subsystems further pushed the progress in these domains.

While early applications of UAVs were single UAV based, the focus is now shifting towards applications involving multiple UAVs, cooperatively completing tasks. There are several advantages of multi-UAV systems over single UAV systems. In certain applications like search and rescue missions, for instance, the use of multiple UAVs for surveying an area can significantly reduce the time taken to complete the task [30]. As given by [33], other advantages of multi-UAV systems over single UAV systems include cost, scalability, survivability and speed.

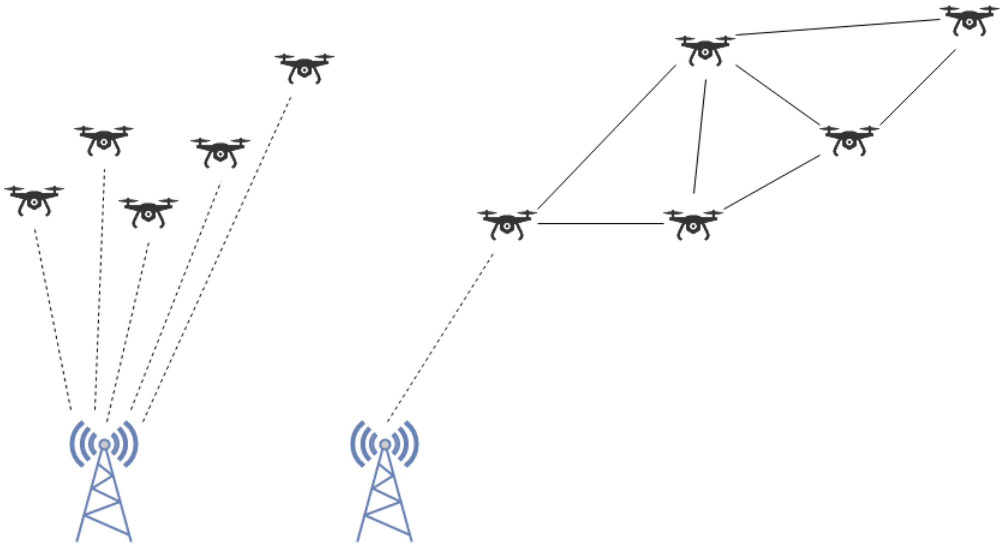
Though UAV swarms have promising capabilities, they have quite a few challenges to overcome. Communication is one of the main hindrances to realize robust swarms of UAVs.

1.2 The Challenge of Communications in UAV swarms

While there is much literature regarding communication in UAV systems, [33] and [13] are two good survey articles regarding the current trends and issues in communication in UAV swarms. Besides pointing out the shifting interest towards multi UAV systems, due to their advantages over single UAV systems, they also identify the challenges in these systems, communication among the UAVs being the most notable one.

Since the first applications were single UAV based, communication architecture in these systems was simple and straightforward. Often, there only needed to be a communication link between the UAV and the ground station. In some cases where the UAV has to cover a larger area, there could be multiple ground stations, and the UAV would have to communicate with the ground station near it. Even then, the overall architecture was pretty basic. As we noted earlier, there is a rapidly growing interest in realizing swarms of cooperative and collaborative UAVs, accomplishing complex tasks. Robust and reliable communication among the UAVs is an essential and critical component in enabling cooperative and collaborative behaviour in these UAV swarms.

While there is some interest in infrastructure based communication in UAVs, like using cellular infrastructure for UAVs [32], much of the research community considers enabling UAV communications in infrastructure less environments, more significant. These so called



(A) Star Configuration (B) Mesh Configuration

FIGURE 1.1: Possible network configurations for communications in UAVs

adhoc networks are important, for instance in disaster management scenarios where the existing infrastructure is damaged, or in military applications where there is no pre-existing infrastructure. The authors in [13] try to identify several possible communication architectures for multi UAV networks.

Figure 1.1a shows the star topology, where all the UAVs are connected to the ground station. In this configuration, the necessary communication among UAVs also needs to be routed through the ground station. Figure 1.1b shows the mesh topology, where the UAVs communicate among themselves using an adhoc network, they themselves form. Typically, the ground station is also a part of the adhoc network. In the star configuration, since all the data has to be routed through the ground station, there would be higher latency and congestion in the network.

On top of that, if the UAVs are considerably far from the ground station, the network in star configuration may not provide reasonably high bandwidth for the communication among the UAVs. Moreover, if any of the UAVs loses its link to the ground, it cannot be accessible by any other UAVs, even if they are close. It is clear to see that the mesh configuration has a considerable advantage over the star configuration. Indeed, there is a consensus in the literature that realizing adhoc networks is the best solution for communication in multi UAV systems. There is a growing intereset in FANETs, the Flying Adhoc Networks. There is much literature in the filed and [33] introduces and outlines the current and future trends in FANETs.

1.3 Motivation and the Current Work

With the advent of Internet of Things(IOT), there are now several communication standards which enable mesh networks like Bluetooth mesh networks [9], Thread [4, 28] and Zwave [31], where the last two protocols are based on the IEEE standard 802.15 [3]. Despite these, mesh networks which are based on the wifi, as standardized in IEEE 802.11s, better suit the needs of UAVs because of their high data rate and range.

While there are previous works which have used wifi based adhoc networks in UAVs like AUGNET [6] and UAVNET [21], they have several issues, not least of which is the range of the networks. Though the mesh networks enable reliable communication among the UAVs at an acceptable range, the link between the UAVs and the ground station is much more vulnerable. While the distance between the neighbouring UAVs can be kept relatively small, one cannot restrict the distance between the ground station and a UAV. A typical scenario would be that of a group of UAVs flying as a swarm far from the ground station. Even though the UAVs are in range and allow communication among themselves via the wifi mesh, the ground station at a larger distance can no longer be a part of the mesh. This is certainly a limiting factor in mesh networks for UAVs. This is one of the motivations for this work, in which we address this issue by using two different radio modules, a long range and a short range one(the wifi mesh). We propose a new architecture which blends short range and long range aspects of the communication. Note that the long range radios typically provide low bandwidth and are not suited for robust inter communication among the UAVs, demanded by many emerging distributed applications. This is not a concern in the case of the link between the UAVs and the ground station as long as, much data is not needed at the ground station, which is the case in autonomous applications, where much of the decision making happeds on board the UAV and ground station is merely used for monitoring.

Another aspect of motivation for this work is its integration with the Robot Operating System or ROS. ROS is a widely accepted software platform for research in robotics, which became popular as *Linux for Robotics.* ROS, with its distributed and modular architecture, active development, collabarative environment and vibrant community has seen widespread adoption by individuals and organizations, across the board in robotics research. Thus the goal was to integrate our communication architecture into ROS, enabling development of distributed applications among UAVs without the need to worry about the underlying communication architecture.

1.4 Organizaiton of the thesis

We have introduced and motivated the problem of communication in UAV swarms in chapter 1. In chapter 2, we shall look at the development of the mesh architecutre. We shall look at the long range communication architecture in chapter 3. We shall look at some future work and finally conclude this thesis with chapter 4.

Chapter 2

The Mesh Architecture

In this chapter, we will look at the short range communication architecture for inter communication among the UAVs which entails realizing a wifi mesh network based on the IEEE standard 802.11s. We shall also look at how it is integrated into ROS.

2.1 IEEE 802.11s

The IEEE WLAN 802.11 standard is quite a popular solution for high bandwidth networking services at reasonable distances. But, it is based on a centrailized architecture, where every networking node is a *Station*, STA. In the simplest architecture, one of the STAs acts as an *Access point*, AP, a centralized node which provides integration services to other STAs. It is a single hop architecture where *STAs* directly communicate with *APs*, and all the communication between *STAs* is routed through the *AP* to which they are connected. Essentially, it is a network in star configuration, at MAC layer.

With growing demand for more diverse wireless infrastructure and multihop networks, 802.11s [16] emerged as an ammendment to the original standard to accommodate mesh networking architecture. The 802.11s standard extends the 802.11 MAC layer, allowing a MAC based multihop architecture.

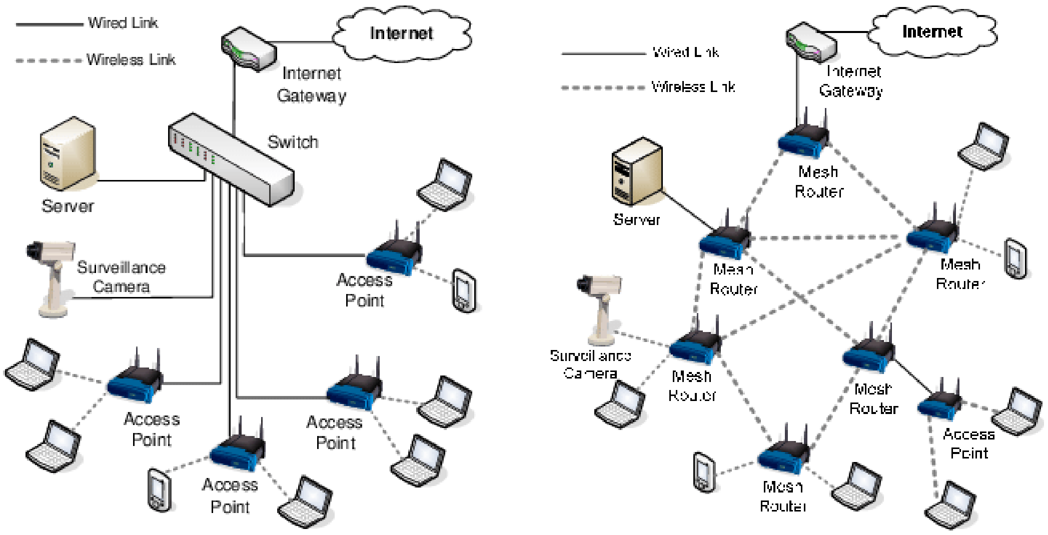


Figure 2.1: A comparison of 802.11 wlan and 802.11s mesh networks

source: Portmann, Marius. (2006). Wireless Mesh Networks for Public Safety and Disaster Recovery Applications. 10.1201/9781420013542.ch16.

2.1.1 Routing in 802.11s

Though there are a lot of routing protocols proposed in wireless mesh networks, the field is still active in research. What routing protocol to use closely depends on the application, and choice of the routing protocol may significantly impact the performance of the system. Different routing protocols can be mainly categorized into two types.

- ***Proactive Routing Protocols:*** Proactive routing protocols are those protocols where the nodes keep track of the routes of all the accessible nodes. This is typically done by storing tables of all the routes based on the topology of the network. The tables are updated as the topology changes. While transmitting data, the node can instantly find the next hop from the routing table, which allows little or no latency in the sytem. On the other hand, to keep the tables updated with the topology , a lot of messages have to be sent between the nodes, which decreases the effective bandwidth for data. Moreover, these protocols tend to be slow to adapt to fast changes in topology. Some of the popular proactive routing protocols include *Optimized Link State Routing* (OLSR) [8], *Destination Sequenced Distance Vector*

(DSDV) [23], *Better Approach To Mobile Adhoc Networking* (BATMAN) [17], to name a few.

- ***Reactive Routing Protocols:*** Where proactive routing protocols keep track of the routes based on topology, reactive routing protocols are on-demand routing protocols where a route between two nodes is found only when there needs to be communication between them. These protocols are defined to address the bandwidth issues of the proactive protocols. As they don't need to keep track of the routes, periodic flooding the network with messages to find the current topology, is not needed, which saves the bandwidth. On the other hand, these protocols tend to introduce more latency into the system as a route to a node has to be found before transmission of data. Some of the reactive routing protocols include *Dynamic Source Routing* (DSR) [18], *Adhoc On Demand Distance Vector* (AODV) [7].

Due to significant research in routing protocols for mobile adhoc networks, there have been other classes of protocols as well, apart from the two main categories above. There are *Hybrid Routing protocols* like *Zone Routing Protocol* (ZRP) [14] and *Temporarily Ordered Routing Algorithm* (TORA) [22], which try to blend both low latency and bandwidth efficiency of proactive and reactive protocols together. There are also a class of *Geographic Routing protocols* [5, 19, 20] which are based on the geographic positions of the nodes, like GPS data for instance.

The 802.11s standard defines a mandatory default routing scheme called *Hybrid Wireless Mesh Protocol* (HWMP), which is a hybrid protocol combining procative and reactive ascpects of routing protocols. However, it is not mandatory that one has to use the default HWMP and can implement any routing protocol while using 802.11s PHY and MAC layers.

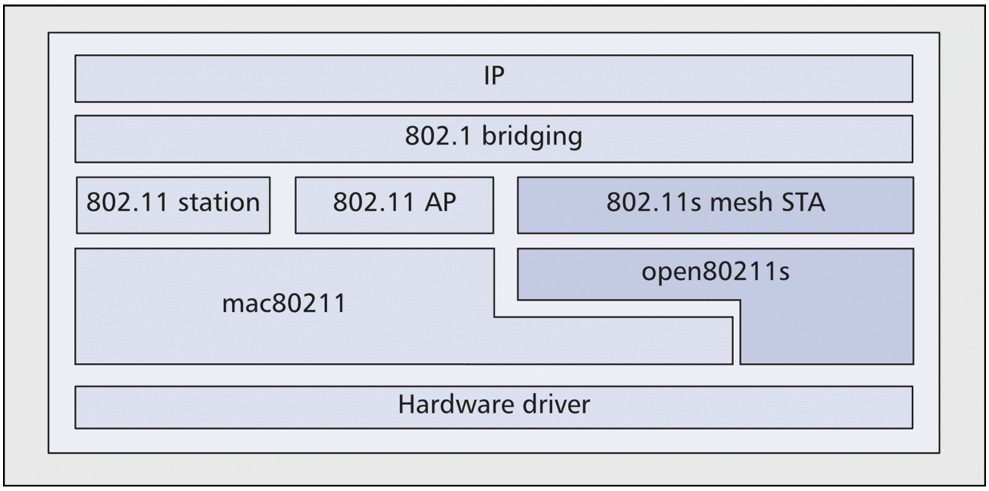


Figure 2.2: open80211s stack on Linux kernel

source: G. R. Hiertz et al., "IEEE 802.11s: The WLAN Mesh Standard," in IEEE Wireless Communications, February 2010.

2.1.2 open80211s

open80211s stack is an open source implementation of the 802.11s standard on the Linux kernel. It is based on mac80211 wireless stack, which implements the standard 802.11 on Linux kernel. Since it is based on mac80211 and makes minimal changes to the 802.11 MAC layer, it is supported on all legacy hardware that supports mac80211.

2.2 Implementation

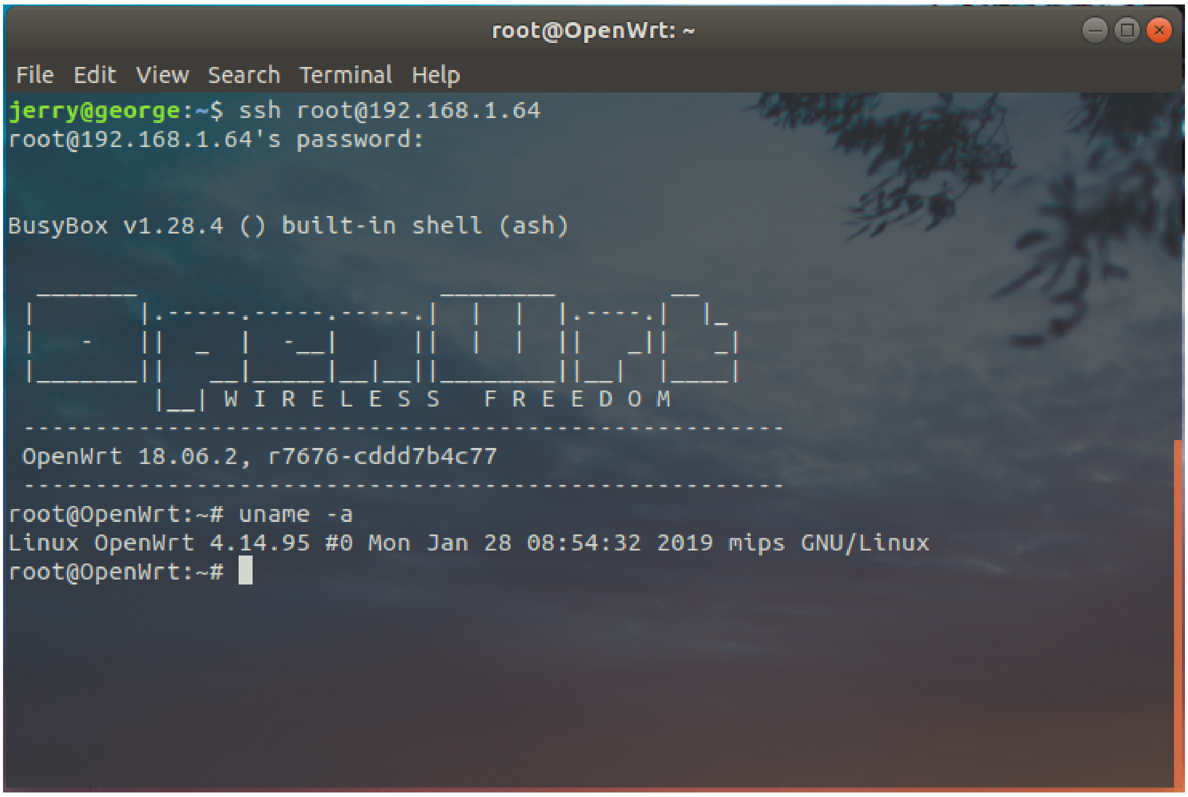
Figure 2.3: logo of OpenWrt

source: https://openwrt.org/

In this section, we shall look at the hardware used as well as how we actually implemented an 802.11s mesh network. As we mentioned in the last section, the open80211s stack can be implemented on most linux machines with legacy wireless cards which are compatible with the standard mac80211 wireless stack. There is a lack of embedded wireless hardware for building 802.11s mesh networks. We wanted to build 802.11s mesh networks with easily available commercial 802.11 legacy hardware and we found a solution with OpenWrt.

2.2.1 OpenWrt

Openwrt [1] is an embedded operating system based on linux kernel, meant for wireless routers. Basically, it is an open source project in which the community releases ports of the operating system for specific wireless routers, whose wireless hardware is compatible and whose memory and firmware allow reflashing the default firmware, these devices come with. Since, not all wireless routers can be flashed with openwrt, we would have to choose from a list of openwrt compatible hardware.



```
jerry@george:~$ ssh root@192.168.1.64
root@192.168.1.64's password:


BusyBox v1.28.4 () built-in shell (ash)

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 OpenWrt 18.06.2, r7676-cddd7b4c77
 -----------------------------------------------------
root@OpenWrt:~# uname -a
Linux OpenWrt 4.14.95 #0 Mon Jan 28 08:54:32 2019 mips GNU/Linux
root@OpenWrt:~# 
```

FIGURE 2.4: An ssh shell on an openwrt flashed router

Once we obtain an openwrt compatible wireless router, we would have to flash the router with an image of the openwrt operating system. Typically, different routers have different instructions for flashing the image and are generally listed along with other details on the device page, specific to the router. Once the router is flashed with openwrt, the device will be running a full-fledged operating system based on linux kernel, albeit one with limited capabilities. When we power on the router, it boots up into openwrt with a default ip address of 192.168.1.1/24. If we connect the router to a host machine via ethernet and *ssh* into the said ip address, we would log into the openwrt with a secure shell. Figure 2.4 shows a terminal on a wireless router, flashed with openwrt and logged in via ssh, from an host pc connected to it by ethernet.

2.2.2 Hardware

Let us now look at the actual routers we have used to realize the mesh networks. We have tested the entire setup with two wireless routers. Both the routers are compatible with openwrt and were decided upon going through the list of supported hardware on the webpage `https://openwrt.org/toh/start`.

2.2.2.1 Tp-Link WR902AC

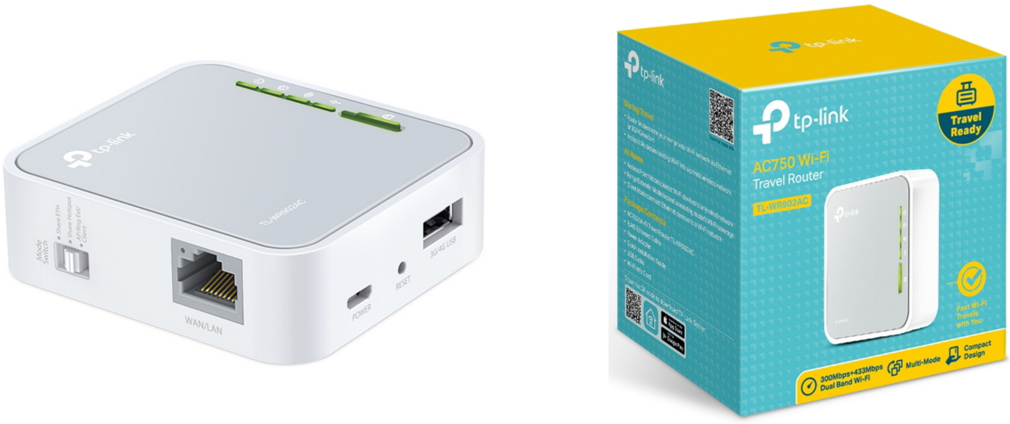


Figure 2.5: Tp-Link WR902AC

The first router is Tp-Link TL-WR902AC v3. Its device page on openwrt can be found at `https://openwrt.org/toh/hwdata/tp-link/tp-link_tl-wr902ac_v3`. The router has a 10/100 Mbps ethernet port, a 2.0 USB port and micro usb port. It needs externel power supply of 5V/2A via the micro USB port. The default firmware the device comes with supports the standards IEEE 802.11ac/n/a @ 5GHz and IEEE 802.11b/g/n @ 2.4 GHz, making it a dual band router. It can support wireless speeds up to 300 Mbps @ 2.4GHz and 433 Mbps @ 5GHz. The transmit power is <20dBm @ 2.4GHz and <23dBm @ 5GHz. The openwrt port for the device doesn't support the device operating at 5GHz.

2.2.2.2 Alfa Tube 2H

The second router we tested is Alfa Tube 2H. The corresponding device page on openwrt can be found at `https://openwrt.org/toh/hwdata/alfa_network/alfa_network_tube2h`. This router too has a 10/100 Mbps ethernet port. It also has an N-type male antenna port to connect external antennas. It supports passive *Power Over Ethernet*(POE) and is powered using ethernet port itself. Unlike the Tp-Link device, it is a single band router, which supports IEEE 802.11b/g/n, operating at 2.4GHz. It can support wireless speeds up to 150Mbps. Its transmit power can go upto 500mW or 27dBm. The Alfa Tube 2H offers two main advantages as compared to WR902AC. First, its transmit can go upto 27dBm whereas the WR902AC can only support a transmit power upto 20dBm. This translates to higher range for communication. Second, unlike the WR902AC, the Tube 2H supports external antennas which can further increase the range of the device.

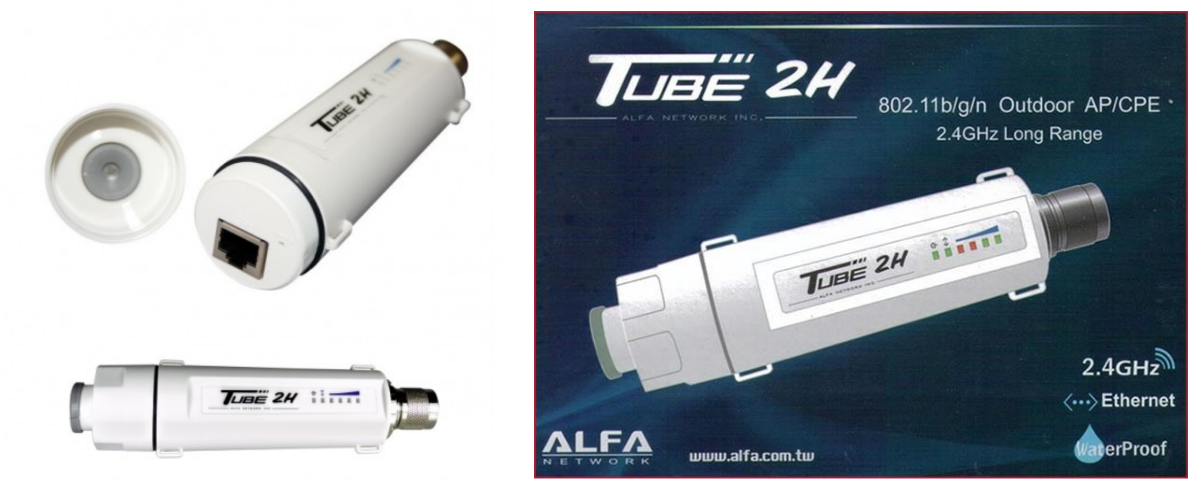


Figure 2.6: Alfa Tube 2H

2.2.3 Network Configuration

Since Openwrt is a full fledged operating system based on the linux kernel, it offers tremendous flexibility in configuring the wireless devices for the network. The directory structure of the system is quite similar to normal linux distributions except a few minor changes. For instance, all the configuration files for different subsystems can be found in the folder */etc/config*. Changing these files and restarting the respective services, or the entire system would change the configuration of the system.

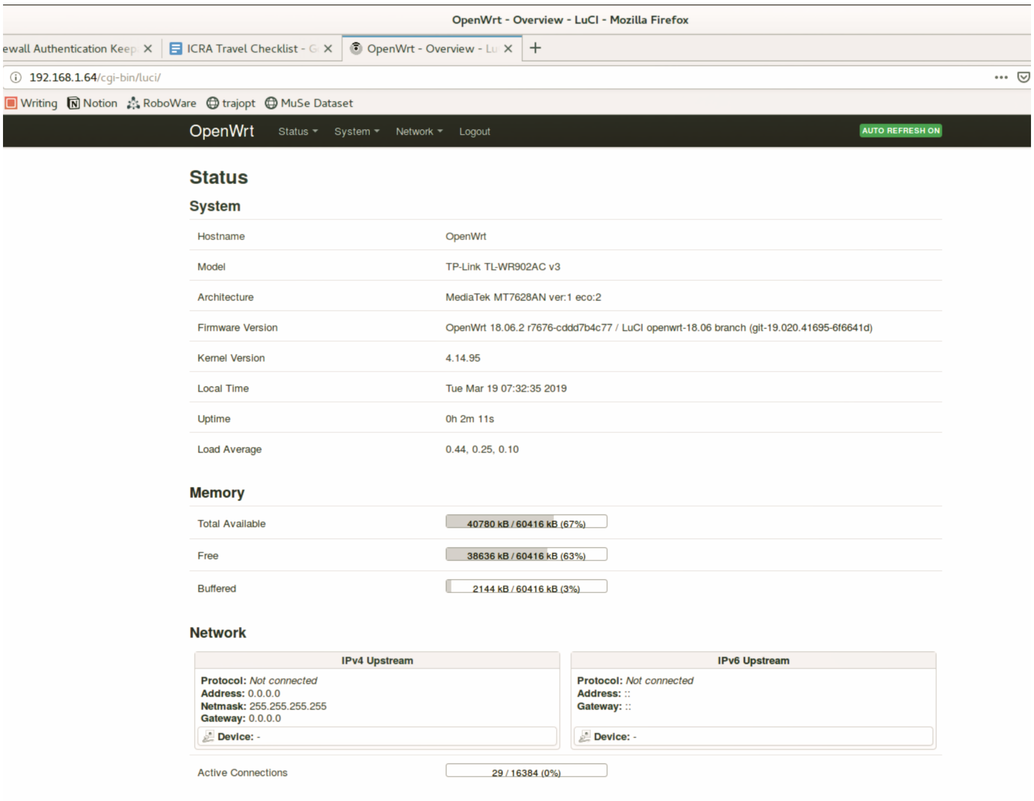
To connect the router to a wifi access point, we would first have create a device interface in *client* mode in */etc/config/wireless*. We would also have to specify the *ssid* of the *AP* and provide authentication, if any. Then, we would have to create a network interface for the above created device, assign an IP address to it in case of static address or specify DHCP etc., in */etc/config/network*. Restarting the system or the network services, then , would connect the router to the specified *AP*.

2.2.3.1 LUCI

Openwrt also provides an easy gui interface to do the network configuration. The interface is called LUCI, which can be accessed by typing in *https://<IP address of the router>*, in a web browser, of the host pc, connected to the router, via ethernet. The interface can be used to connect the router to an access point, or make the router an access point and share its network. However, luci has limited functionality and cannot be used to configure 802.11s mesh networks. We would have to do that by manually changing the configuration files. Figure 2.7 shows the luci interface opened in a web browser.

2.2.3.2 802.11s mesh configuration

After the routers are flashed with an image of openwrt and booted up, they do not have the capability of supporting 802.11s mesh by default. Some setup and configuration is needed to implement 802.11s on these devices. First, we connect the router to the internet


OpenWrt - Overview - LuCI - Mozilla Firefox
ewall Authentication Keep
ICRA Travel Checklist - G
OpenWrt - Overview - Lu
192.168.1.64/cgi-bin/luci/
Writing
Notion
RoboWare
trajopt
MuSe Dataset
OpenWrt
Status
System
Network
Logout
AUTO REFRESH ON

Status

System

Hostname	OpenWrt
Model	TP-Link TL-WR902AC v3
Architecture	MediaTek MT7628AN ver:1 eco:2
Firmware Version	OpenWrt 18.06.2 r7676-cddd7b4c77 / LuCI openwrt-18.06 branch (git-19.020.41695-6f6641d)
Kernel Version	4.14.95
Local Time	Tue Mar 19 07:32:35 2019
Uptime	0h 2m 11s
Load Average	0.44, 0.25, 0.10

Memory

Total Available	40780 kB / 60416 kB (67%)
Free	38636 kB / 60416 kB (63%)
Buffered	2144 kB / 60416 kB (3%)

Network

IPv4 Upstream	IPv6 Upstream
Protocol: *Not connected* **Address:** 0.0.0.0 **Netmask:** 255.255.255.255 **Gateway:** 0.0.0.0 **Device:** -	**Protocol:** *Not connected* **Address:** :: **Gateway:** :: **Device:** -

Active Connections	29 / 16384 (0%)

FIGURE 2.7: LUCI web interface for configuring openwrt routers

by connecting it to any wifi access point, via luci. Next, the package *wpad-mini* which is installed by default should be removed and the package *wpad-mesh* needs to be installed. This can be done by using the default package manager of openwrt, opkg.

Now, we could implement 802.11s on these routers. We create a device interface in mesh mode, specify the name of the mesh network in wireless configuration file. Then, we create a network interface for the mesh interface in network configuration file. In our case, we decided to use static ip addresses for the routers with the network address 192.168.1.0 and a subnet mask of 24. So, the ip addresses assigned to different routers are of the form 192.168.1.xxx.

After the setup, if two are more routers with the same configuration, but different ip addresses are powered on, they should be forming a mesh network among themselves. This can be verified by connecting one of the routers to a host pc, *sshing* into the router and pinging the other routers. The ping would be getting responses from the other routers.

2.2.3.3 Routing protocols

As we have noted in an earlier section, the standard 802.11s allows different routing protocols to be used on top of the 802.11 MAC layer, even though it defines a default routing protocol, HWMP. While it is theoretically possible to implement a custom protocol, there are a few routing protocols with open source implementations one can readily use. Among them the protocols BATMAN and OLSR are popular and have implementations for linux as well. Openwrt supports both the protocols and documents how to configure the 802.11s mesh to use these protocols. In our case, we have used BATMAN as the routing protocol as it is shown that BATMAN outperforms OLSR in real world scenarios [2].

2.2.3.4 Mesh networks among the host machines

In the last few sections, we saw how we implemented 802.11s mesh networks on commercially available wireless routers. In the end, after everything is configured, we would just power on the routers and a mesh network would be established among them. But, we would not just like to create a mesh network among the routers themselves, but would like to create mesh networks among the machines to which these routers are connected.

The routers mainly have two interfaces within them, the ethernet interface and the wireless interface. The ethernet interface may be connected to a host machine, while the mesh network is formed on the wireless interface. In case we want to create a mesh network among the host machines themselves, we would need to have a way to share the mesh network on the wireless interface with the ethernet interface. For this, we bridge both the wireless and the ethernet interface. Bridging both the interfaces creates a single logical network interface. Though there are two device interfaces, there would only be one logical network interface after bridging, to which we would assign the IP address. Thus, bridging enables mesh networks among host machines themselves as long as they are connected to the so configured mesh routers. Figure 2.8 illustrates the bridging of the wireless and ethernet interfaces in routers, where they established a mesh network among themselves and each of them is connected to a host machine via the ethernet port.

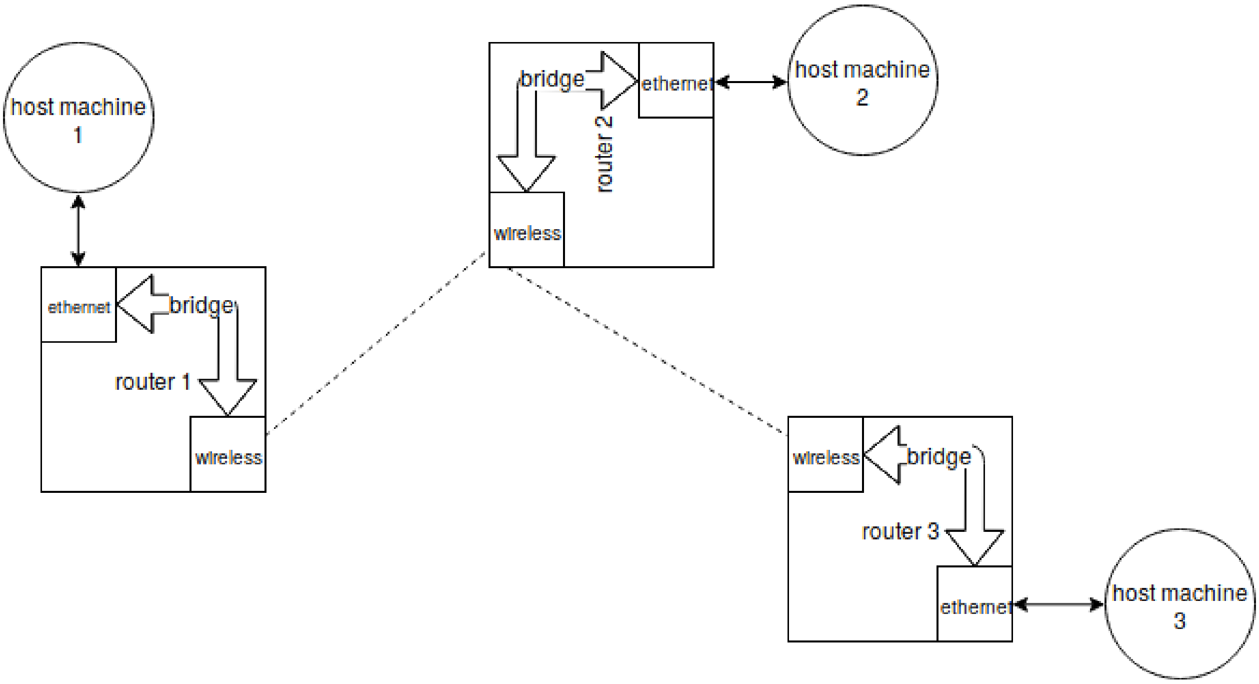


FIGURE 2.8: Mesh architecture with bridging of wireless and ethernet interfaces in routers

2.3 Integration with ROS

One of the motivations of this work is to integrate the communication architecture into ROS. ROS is quite a flexible framework allowing to run it on multiple machines pretty easily. ROS fits well in the application layer in the traditional OSI layers and uses TCP/IP for sending and receiving messages on ROS topics and services. Thus, as long as a TCP/IP stack is installed on the machines and the network between them is estabished, it is trivial to setup ROS on these machines.

The standard way to do it is to specify one of the machines as a ROS master. This is specified by setting the environmental variable ROS_MASTER_URI to the IP address of the designated master on all the other machines. But, this setup establishes a star configuration in application layer. When ROS is run on multiple machines in the standard way, designating one of the machines as the ROS master, all the communication between nodes is coordinated through the ROS master. This is very inefficient both bandwidth wise and latency wise. Moreover, if, for whatever reason, the msaster node becomes unreachable, it would effect the entire system. Thus, it would be better to implement a multimaster version of ROS, where all the host machines will be running as a ROS master and still be able to access ros topics and services of the other masters as required.

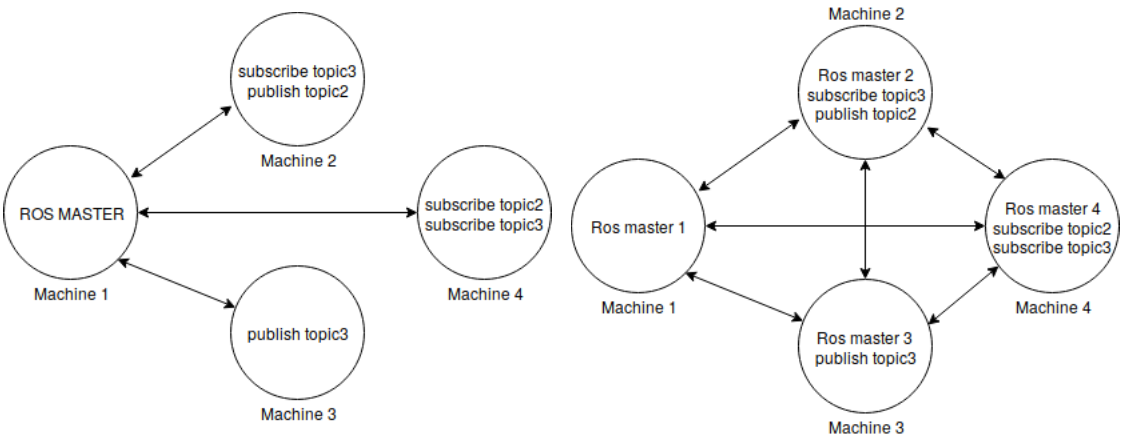


FIGURE 2.9: An illustration of ROS single master and multimaster setups

2.3.1 Multimaster_fkie

Multimaster_fkie [15, 27] is a ROS package that allows running different ROS masters on multiple machines and still enable access of required ROS topics and services among the different masters. It consists of two main nodes, *master_discovery* and *master_sync*.

2.3.1.1 master_discovery

The *master_discovery* node is responsible for discovery of other ros masters in the network. As we shall see shortly, it relies on multicast feature of IPV4 stack to do this. It periodically sends a multicast message to notify its presense on the network to other ros masters. Furthermore, it monitors incoming multicast messages to discover new masters and keep track of changes to their states.

As stated above, the *master_discovery* node uses multicast addressing for its functionality. Since multicast feature is disabled by default on many systems, the feature needs to be enabled, if not. While all muticast enabled devices are added to the multicast group *224.0.0.1*, by default, one can add all the machines to any feasible multicast group of their choice. After everything is configured, pinging the multicast address should give you replies from the other machines added to the same multicast group.

Once the configuration is done on all the machines, we could run the *master_discovery* nodes on all the machines, with the parameter *_mcast_group* set to the chosen multicast group. The node would be reporting the ros masters it discovers on the network. The node also provides a ros service *list_masters*, which upon running would produce a list of all the ros masters discovered.

2.3.1.2 master_sync

While the *master_discovery* node discovers other masters, it is the *master_sync* node which handles the data transfer between the nodes of different masters. It is responsible for registering the topics and services of foreign masters on the local master. It can be configured to select which hosts, nodes, topics and services are to be synchronized. Thus, one could only select only the required topics and services and ignore all the remaining data.

Once the configuration on all the machines is done, we would, first, run the *master_discover* node, with multicast group as the parameter and then, run the *master_sync* node on required machines. The *master_sync* node synchronizes the topics and services on the local machine i.e., if you run the *master_sync* node on machine1, the topics and services of the other masters can be accessed on machine1 only. To access machine1's topics and services on other machines, *master_sync* nodes have to be run on those machines.

Chapter 3

Long Range Communication Architecture

In this chapter, we shall look at the long range communication architecture, menat for communication between ground station and the UAVs, and how it is integrated with the mesh network and ROS.

3.1 Hardware

Among the existing solutions for long range communication, radios based on 900Mhz spectrum seem popular. Compared to 2.4Ghz radios, there are a few reasons for this. First, the path loss for 2.4Ghz radios is higher than that of 900Mhz radios, reducing the received signal strength significantly over long distances. With the use of high gain directional antennas, the gap in the received signal strength between the radios of two spectra, can be eliminated, or even reversed. While this makes the 2.4Ghz radios favorable than 900Mhz radios in static point-to-point links, 900Mhz radios still have a much longer range than 2.4Ghz radios. Second, the 2.4Ghz spectrum is more vulnerable to the issues of penetration and blockages in the face of obstacles, while 900Mhz is more robust to them.

Note that much of 2.4Ghz hardware consists of wifi based solutions, based on the 802.11 hardware, and 802.11 was designed to be indoor, with short range, but high bandwidth. Consequently, hardware based on 802.11 is inefficient for use over long distances. While there are implementations of custom protocol stacks, instead of 802.11, on 2.4Ghz spectrum, these are almost always propietary and consequently, the modules are quite expensive. Considering all things, 900Mhz radios are still offer quite a promising solution for communication over long distances. One downside is that the 900Mhz radios typically offer lower bandwidth on the order of a few 100 Kbps, compared to 2.4Ghz solutions. Nonetheless, this should not be a problem as long as not too much data is demanded at the ground station, which is true in most applications.

3.1.1 RFD900X

RFD900X is third in line of popular 900Mhz radios by the company, RFDesign, the first two being RFD900 and RFD900+. The radio operates on frequency range 902-928 Mhz. It allows selectable transmit power upto 30dBm, in steps of 1dBm. It supports multiple air data rates from 4Kbps to 500Kbps. It has a UART interface supporting multiple baud rates from 9600 to 1000000. It has an advertised range of more than 50Kms in line of sight conditions, depending on antennas. The radios support a set of AT commands for configuring different parameters. There is decent documentation regarding the radios that can be found at `http://files.rfdesign.com.au/docs/`.

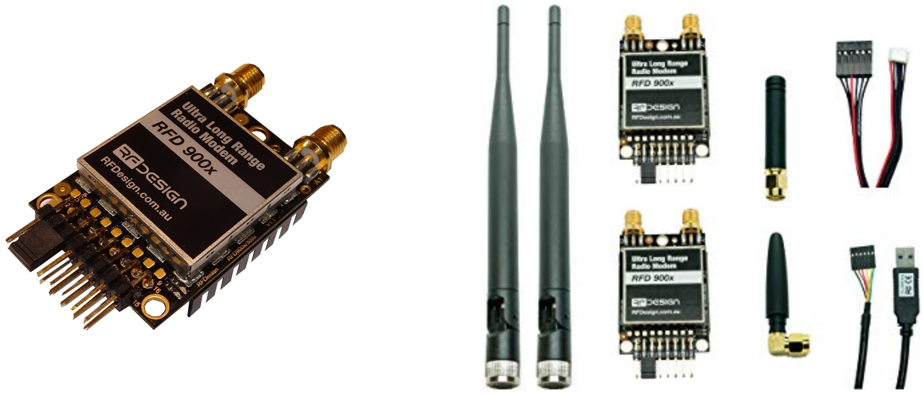


Figure 3.1: RFD900x along with antennas and FTDI cables

The radios come with a default peer to peer firmware, which means that the radios can only be used in pairs. There are two other firmwares that can be flashed on radios, synchronous mesh and asynchronous mesh, which are point to multipoint firmwares. These firmwares can be flashed by a gui program called 'Modem Tools' that can downloaded from `http://files.rfdesign.com.au/`.

3.1.1.1 Peer to Peer Firmware

As mentioned, the radios come flashed with the peer to peer firmware by default. We may need to configure the radios with some parameters via the AT commands. For instance, the parameters NETID, SERIAL_SPEED, AIR_SPEED should be same on both the modems, among other things. Radios with same NETID can talk to each other. Essentially, any raw bytes input to the serial UART interface at one radio can be received at the serial interface of the other radio. Different pairs of radios with different NETIDs can only send and receive data to and from the radio with the same NETID. Even though you can have multiple pairs of radios wiht different NETIDs, as the total number of radios increase, interference also increases, increasing the errors in transmitted bytes.

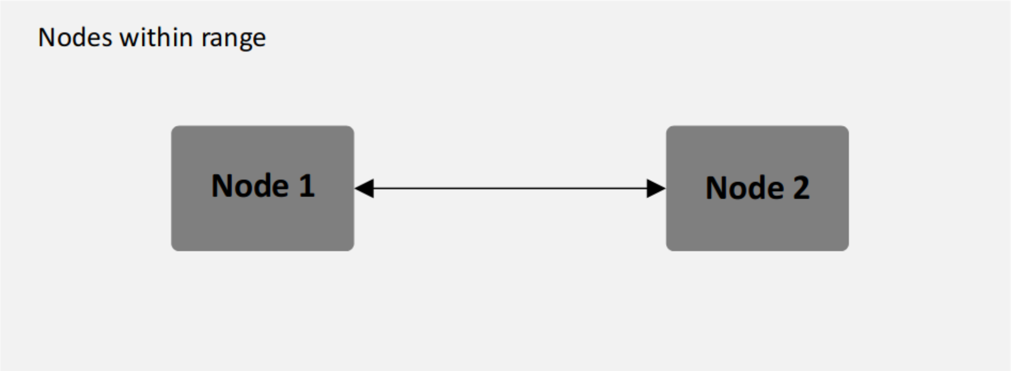


FIGURE 3.2: modems in peer to peer configuration

source: http://files.rfdesign.com.au/Files/documents/RFD900x%20DataSheet%20V1.1.pdf

3.1.1.2 Synchronous Mesh Firmware

While the peer to peer firmware allows communication only between a pair of radios, the asynchronous firmware is a point to multipoint firmware. Apart from the NETID parameter, there is a NODEID parameter, NODEDESTINATION parameter, NETCOUNT parameter among others, that can be set by the AT commands. Unlike the peer to peer firmware, more than two radios can have the same NETID. While using the synchronous firmware, one of the radios needs to be as a base node. This can be done by setting the NETID parameter to 0 and NODEID parameter to 1 on the base radio. We should also specify the number of radios with the parameter NETCOUNT. Each radio should be assigned a unique NODEID, from 1 to 15. By, specifying the NODEDESTINATION, the radio transmits the data to that node. If the NODEDESTINATION is set as 255, it enables broadcast mode and transmits to all the nodes.

In this configuration, the base radio assigns time slots to each of the other radios to communicate. Essentially, each radio transmits data only during its assigned time slot. So, even if other radios are not transmitting data, a radio waits for its time slot to transmit its data. This is inefficient when either not many radios are transmitting or they are not transmitting too often.

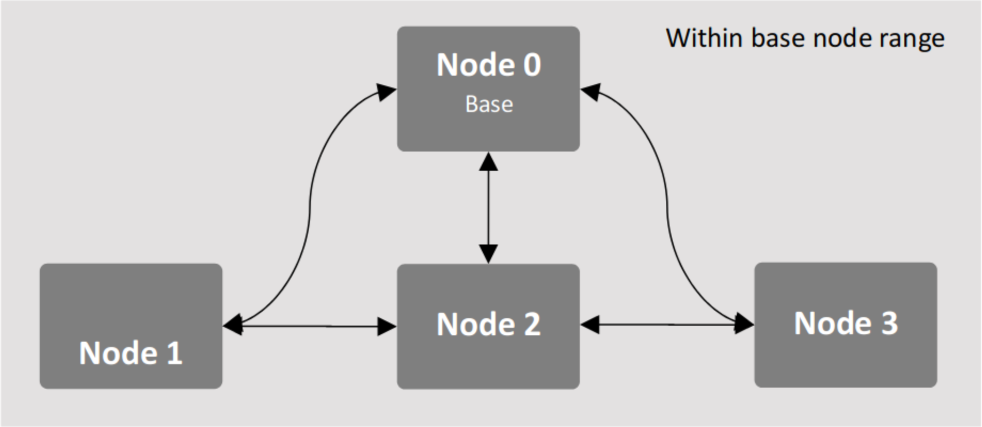


Figure 3.3: modems in synchronous mesh configuration

source: http://files.rfdesign.com.au/Files/documents/RFD900x%20DataSheet%20V1.1.pdf

3.1.1.3 Asynchronous Mesh Firmware

Like synchronous firmware, this a point to multipoint firmware. But, in its configurable parameters, there is no NETCOUNT pararmeter, and instead of NODEDESTATION, there is a DESTID parameter. Here, the NODEID can be set to any unique value between 1 to 32767. The radio transmits to a receiver radio, specified by setting the parameter DESTID to the NODEID of the receiver radio. If the DESTID parameter is set as 65535, broadcast mode is enabled and the radio transmits to all the radios in the network.

While, in the synchronous firmware, each radio in the network, is assigned a time slot, for transmitting data, coordinated by the base radio, in the case of asynchronous firmware, there is no base radio which assigns time slots and the radios can transmit any time, asynchronously. If two radios transmit simultaneously and a packet collision happens, the radios retry to transmit the packet, multiple times, based on the parameter, MAX_RETRIES.

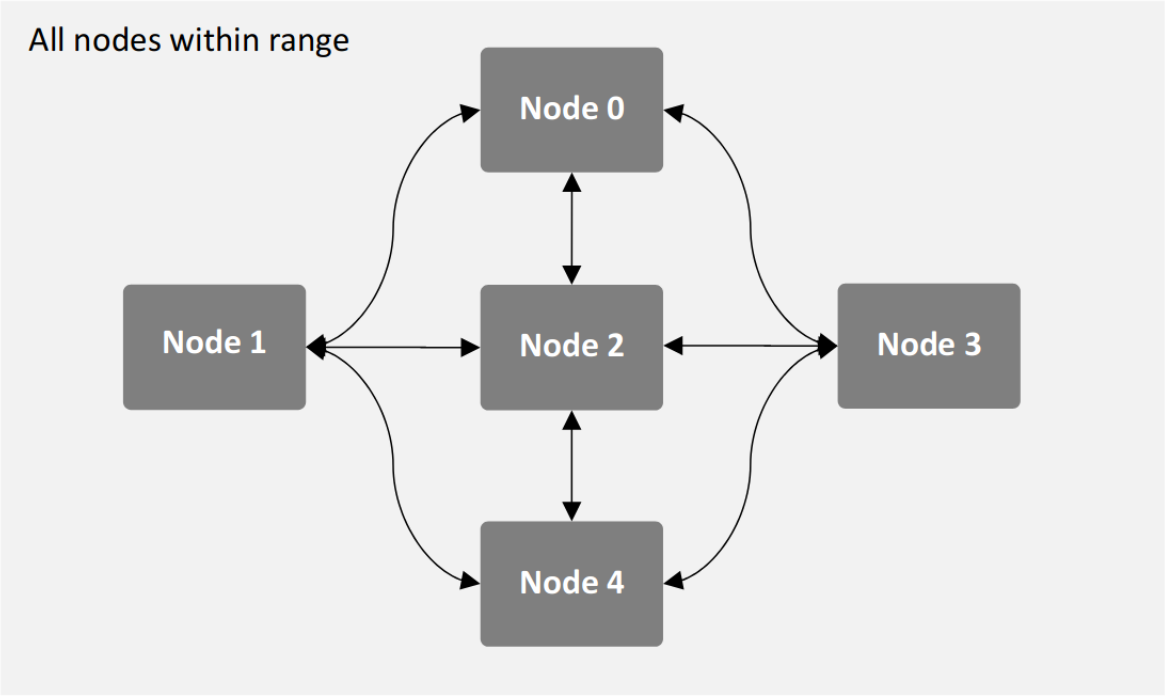


Figure 3.4: modems in asynchronous mesh configuration

source: http://files.rfdesign.com.au/Files/documents/RFD900x%20DataSheet%20V1.1.pdf

3.2 Architecture and Integration with ROS

Let us give a brief outline of the entire architecture, along with wifi mesh and the RFDs. As we described in the last chapter, the wifi routers connected to the host machines, will establish a mesh network among the host machines, which are running ROS. Thus, ros topics and services of any host machine can be acessed in any other host machine, as required. Now, the assumption is that, while the mesh network enables communication among the host machines, which are mounted on the UAVs, the communication between the UAVs and the ground station cannot be acheived by the mesh network as the ground station may be far away and out of range of the mesh. This is where the RFDs come in.

However, RFDs have a few downsides. First, they offer much lower data rate, which means that the data that is being sent down by the UAVs to the ground has to be minimal. Second, if all the UAVs are simultaneously transmitting data to the ground station, it would overwhelm the system in one way, or the other. This motivated us to come up with an architecture, where not all the UAVs are transmitting data down to the ground station, at any given time. We designate one of the UAVs as a leader, in the sense that it is the only UAV, that sends data down to the ground station. The leader, itself takes care of all the data, of the remaining UAVs as well, that needs to be sent to the ground station. This is possible since the leader has access to the data of all the remaining UAVs, as they are all connected via the mesh network.

One of the key assumptions is that all the UAVs are always connected via the mesh network, allowing the leader to access their data, as required. But, it may happen that some of the UAVs are out of range of the mesh network and are not accessible by the leader, or the leader itself, may come out of range, disconnecting it with the rest of the UAVs. This prompted us to extend the proposed architecture to address these problems.

In the coming sections, we propose and describe two architectures. First one is a *Static Leader* architecture, where there is only one fixed leader, in the network. Clearly, this assumes that all the other UAVs are accessible by the leader via the mesh network. The second architecture, which we call *Dynamic Leader* architecture is an extension of the first one, designed to address the shortcomings of the second one.

Before proceeding further, let us look at how ROS fits into the architecture, and the interfaces we have had to write for ROS.

3.2.1 ROS

ROS, running on the host machines of the UAVs, communicates with the onboard autopilot, which is connected to various sensors, and publishes all the sensor data and the state of the UAVs, on various ros topics. It also makes various ros services available, which upon calling would call corresponding services on the autopilot. Our aim was to make all the required data of all the UAVs available at the ground station, as ros topics and services.

Since the RFDs are serial radios and do not have a TCP/IP protocol stack, these radios cannot be integrated into ROS, as easily as the wifi mesh network. The reason for this is that, while ROS allows access of ros topics and services via TCP/IP, it does not have a serial interface. By serial interface, we mean the access of ros topics and services via a serial communication link. This has led us write our own serial interface for ROS. We have written this interface as an independent open source ROS node in a package, that anyone can use in their own applications. Both the node and the package are called *serialros*.

3.2.1.1 serialros

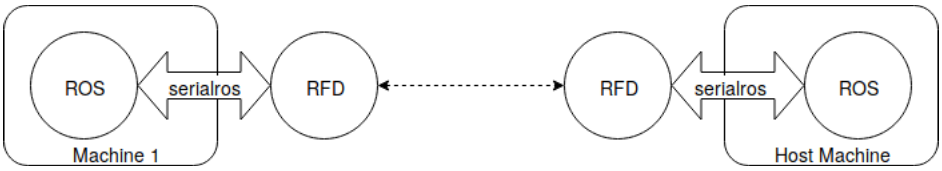


Figure 3.5: serialros, a serial interface to ROS

The *serialros* package allows sending and receiving of arbitrary ros topics via a serial communication link. It also allows access of ros services via a serial link. The node, when launched, monitors the serial port as specified and publishes any incoming messages, on their corresponding topics in the local machine. However, it does not transmit any ros messages of any topics, that are published on the local machine, via the serial port. For transmitting messages, from a topic that is being published on the local machine, via the serial port, the topic has to be specified in a configuration file. Similarly the services of a remote machine that we want to access on the local machine needs to be specified in the configuration file.

The package has a launch file *serialros.launch*, that can be launched with *roslaunch*, which launches the relevant nodes and starts publishing any incoming ros topics on the serial port, transmits any specified ros topics from the local machine and makes specified services from remote machines accessible in the local machine. The source code and instructions on how to use the package can be found at `https://github.com/saiadityachundi/serialros.git`.

3.2.2 Static Leader

In this architecture, we designate one of the UAVs as a leader. We assume that all the UAVs are always connected via the mesh network. As we saw in the last chapter, as long as all the UAVs are connected, all the ros topics and services of any of the UAVs can be echoed and accessed at the leader.

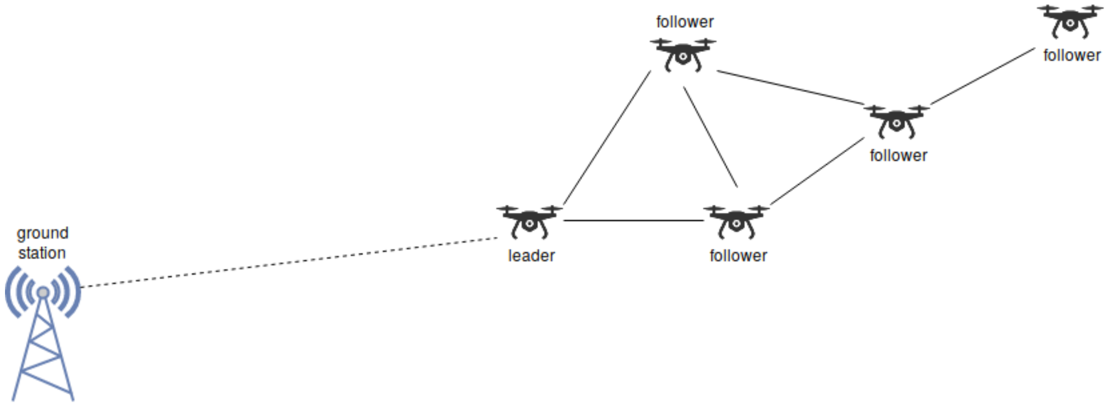


FIGURE 3.6: Static Leader Architecture

The linux machine on the leader runs the *serialros* node, as described in the last section. The *serialros* node is also run on the ground station. This allows transfer of specified ros topics and services between the leader and the ground station. And since, the leader has access to all the ros topics and services of all the UAVs, sepcifying these topics in the *serialros* configuration file would send these topics to the ground station, in the downlink. Typically, the ground station would not need to send any topics to the UAVs, in the uplink. It may need to access some ros services of the UAVs. This can be done by specifying these remote services in the configuration file of *serialros*, at the ground station.

Thus, *Static Leader* architecture combines both long range and short range communication aspects. While the wifi mesh network enables communication among the UAVs for robust distributed applications, the leader established long range communication with the ground station. One of the advantages of this architecture is that the long range radios are only needed at the leader. Also note that, since there are only two long range radios operating, one at the ground station and one on the leader, we can use RFDs, in peer to peer configuration, which offers best performance than either of the point to multipoint firmwares.

However, as mentioned earlier, this architecture assumes that all the UAVs are always connected via the mesh network, which may turn out be false. Moreover, since there is only one leader, there is a single point of failure in the system. In a case where the leader fails, for whatever reason, the ground station link would be gone. The need for addressing these issues brings us to the next section, *Dynamic Leader*

3.2.3 Dynamic Leader

In *Dynamic Leader* architecture, all the UAVs are mounted with RFDs. The RFDs are configured in asynchronous point to multipoint firmware, instead of the default peer to peer firmware. Unlike the *Static Leader* architecture, we do not designate any UAV as a leader in this architecture. We have written a set of ROS nodes, complementing the *serialros* package, which will set the leader dynamically. Let us now, briefly get an overview of this system.

We define three modes for all the UAVs in this architecture; *leader*, *follower* and *emergency*. Except at the time of initiatialization, every UAV has to be in one of these three modes, at any given time, for the entire duration. During operation, at any instant, there is supposed to be only one *leader*, and the remaining UAVs would be *followers*. The third mode, *emergency* is meant for those UAVs, which may have gotten disconnected with the mesh network.

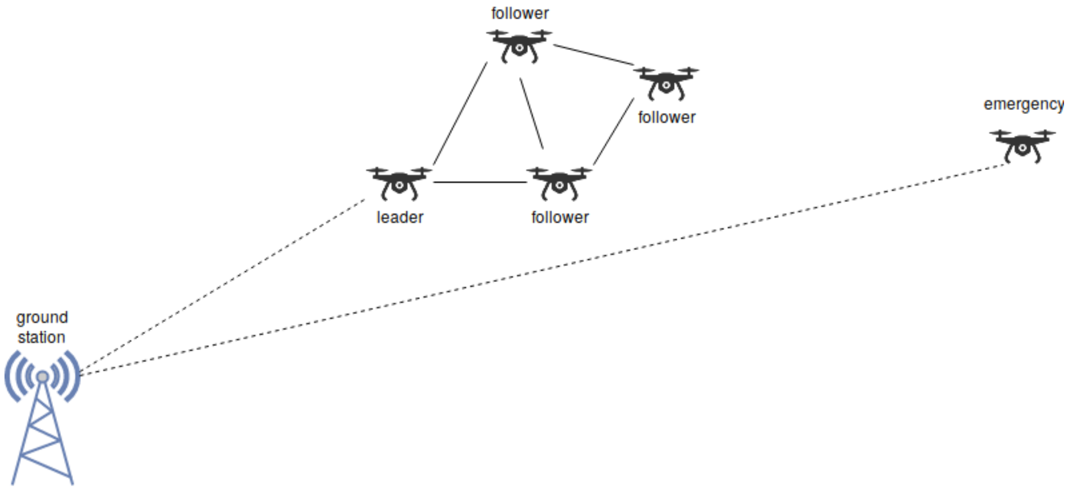


FIGURE 3.7: Dynamic Leader Architecture

3.2.3.1 link_status

Our aim was to design a robust system, where there is no single point of failure. Similar to the *Static Leader* architecture, at any given moment, only the leader *leader* sends the data to the ground station, on behalf of all the UAVs. If any of the UAVs gets disconnected from the wifi mesh network, it detects that it is not in the mesh anymore and goes into *emergency* mode. While in *emergency* mode, the UAV sends its data itself to the ground station. The functionality of checking if the UAV is in the mesh network or not is implement in the ROS node *link_status.py*.

3.2.3.2 get_leader service

Apart from the *leader*, one of the UAVs is designated as a *standby leader*. Every UAV in the mesh network periodically seeks the *leader* in the network. If there is a current *leader* in the network, it would respond with its id. If there is no response and the request is timed out, the current *standby leader* becomes the new *leader*. This functionality of seeking the leader and the leader responding is implemented in the ROS service *get_leader.py*.

3.2.3.3 uav.py

Note that any of the UAVs may go into the *emergency* mode and start transmitting data to the ground station. To allow this, RFDs are mounted on all the UAVs, powered on, ready to send the data. But, to send and receive ros topics and access ros servies via the RFD, the host machines need to the use the *serialros package* as described earlier. While in *leader* mode, the UAVs transmit data to the ground staion, they would not transmit anything while in *follower* mode, and they would, again transmit data if they go into *emergency* mode. Furthermore, they have to send ros topics of all the UAVs while they are in *leader* mode, while they would send only their ros topics when they are *emergency* mode.

All the above functionality requires launching and terminating the *serialros* node arbitrarily. Moreover, the *serialros* node needs to be launched with a different configuration file in each mode, specifying different ros topics and services in the configuration file. To achieve this functionality, we have written a ROS node, *uav.py*, which, using the earlier mentioned nodes *link_status.py* and *get_leader.py*, keeps launching and terminating the *serialros* node with a configuration file depending the current mode of the UAV. Figure 3.8 shows a flowchart depicting the changing mode of a UAV.

All the three ros nodes described above have been written as ROS package, *swarmBaba*. As mentioned, the *uav.py* node uses *serialros* node for sending and receiving topics via the RFD. Thus, this package *swarmBaba* is not independent and *serialros* is a prerequisit

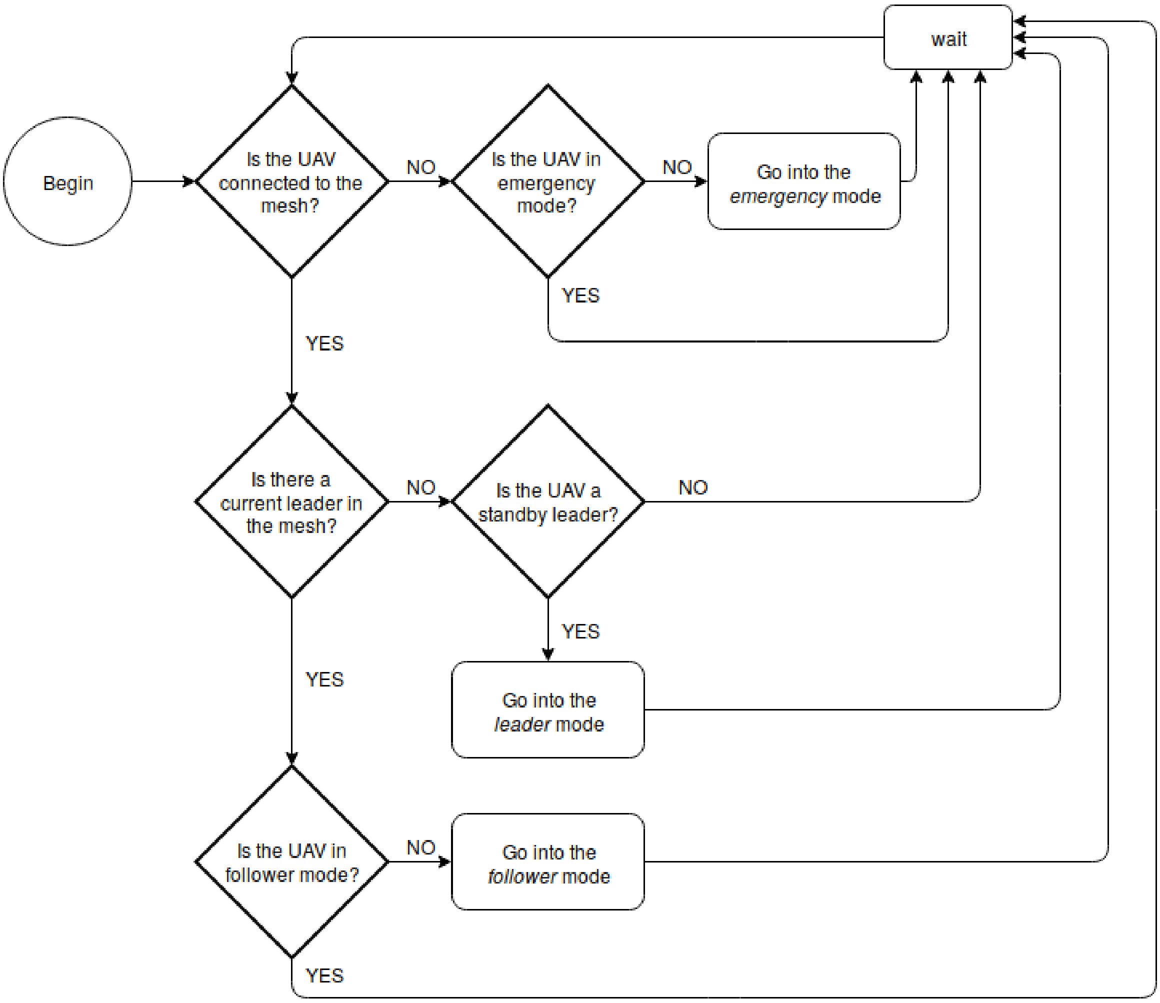

wait
Begin
Is the UAV connected to the mesh?
NO
Is the UAV in emergency mode?
NO
Go into the *emergency* mode
YES
YES
Is there a current leader in the mesh?
NO
Is the UAV a standby leader?
NO
YES
Go into the *leader* mode
YES
Is the UAV in follower mode?
NO
Go into the *follower* mode
YES

Figure 3.8: Flowchart depicting changing mode of a UAV

for using it. The source code and instructions on how to use it can found at `https://github.com/saiadityachundi/swarmBaba`.

3.2.3.4 RFD configuration

Note that in the downlink, all the UAVs which have to transmit to the ground station would have their DESTID parameter set as 1, assuming the RFD at the ground station has its NODEID set as 1. However, for the uplink, the RFD at the ground station sets its DESTID as 65535, configuring the RFD in broadcast mode. No ros topics are sent in the uplink. The uplink is used for accessing ros services offered by the UAVs. Typically, the ground station rarely uses these services. Thus, the long range communication link between the UAVs and the ground station sees much higher data on the downlink than the uplink.

In the *Dynamic Leader* architecture, only the *leader* would transmit data to the ground station, when all the UAVs are in the network. A UAV which disconnected from the mesh network, would have to transmit data to the ground station. However, that does not often. It is safe to assume that in this architecture, there are not many radios transmitting to the ground station. Thus, using asynchronous firmware for the RFDs makes more sense than the synchronous firmware, because much of the bandwidth would be wasted in the latter case, where not all of the radios are transmitting data.

Chapter 4

Conclusion

We proposed and implemented a novel communication architecture for emerging applications in UAVs, using off the shelf hardware. The architecture blends both short range and long range communication systems. For short range communication system, we realize a wifi mesh network among the UAVs, using commercially available wifi routers. We achieve this by using an embedded linux operating system for routers called, OpenWrt. This mesh network enables robust and reliable communication among the UAVs for distributed applications like swarm formation and control, localization, cooperative search, to name a few.

We integrate this architecture with a long range communication system for communication with the ground station. We use popular 900Mhz radios called RFD900x as our long range modules. We propose two architectures, *Static Leader* and *Dynamic Leader*, where the second one is an extension of the first one. In *Static Leader*, we designate one of the UAVs as *leader* which transmits data to the ground station. In *Dynamic Leader*, the *leader* is not set apriori, but chosen dynamically by the UAVs themselves.

The integration of the long range communication sytem with the short range mesh network enhances the effective range of the operations that can be conducted with the UAVs. Furthermore, our architecture is integrated into ROS. This makes our architecture quite flexible, in the sense that it enables anyone to develop their multiple UAV applications in ROS without worrying about the underlying communication architecture.

4.1 Future Work

While we have designed and implemented this architecture and tested it indoors, we haven't done extensive outdoor testing. Hence, one of the directions for future work is to do extensive outdoor testing and measure the range over which this architecture reliably works.

All the ROS nodes that were described in this work were written in python language. While python is good language for rapid prototyping, c++ is much more efficient than python. We believe migrating the entire code to c++ from python would make the system more robust.

Apart from these, the long range radio modules we have used were RFD900x which were serial radios. There are some alternatives to these radios which offer IP based solutions in 900Mhz spectrum. While it would be much easier to integrate an IP based radio into the ROS framework, it is still worthy exercise. There are other proprietary radio solutions in 2.4Ghz spectrum, apart from the normal 802.11 standard, which claim to offer much longer ranges than traditional wifi. Working with these different radios and implementing a similar architecture is worth pursuing.

Chapter 5

Preliminaries

In this chapter, we shall briefly look at all the components present in a general autonomous UAV.

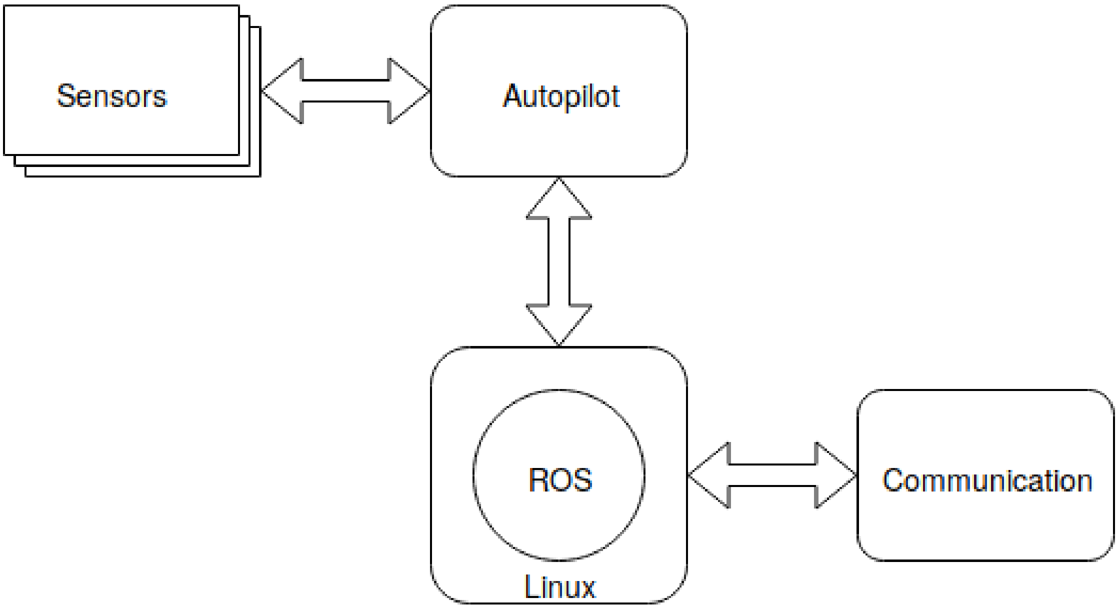


Figure 5.1: Components in a general autonomous UAV

Figure 5.1 shows several subsytems in an an autonomous UAV we consider. It consists of an autopilot which is handles low level control of the UAV dynamics. The autopilot is responsible for attitude stabilization, for instance, in quadrotors. It directly connects with

Bibliography

[1] (2019). https://openwrt.org.

[2] Abolhasan, M., Hagelstein, B., and Wang, J. C. . (2009). Real-world performance of current proactive multi-hop mesh protocols. In *2009 15th Asia-Pacific Conference on Communications*, pages 44–47.

[3] Akyildiz, I. F., Wang, X., and Wang, W. (2005). Wireless mesh networks: a survey. *Computer Networks*, 47(4):445 – 487.

[4] Alm, A. (2019). Internet of things mesh network : Using the thread networking protocol.

[5] Bose, P., Morin, P., Stojmenović, I., and Urrutia, J. (2001). Routing with guaranteed delivery in ad hoc wireless networks. *Wirel. Netw.*, 7(6):609–616.

[6] Brown, T., Argrow, B., Dixon, C., Doshi, S., Thekkekunnel, R.-G., and Henkel, D. (2004). Ad hoc uav ground network (augnet). *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit.*

[7] Chakeres, I. D. and Belding-Royer, E. M. (2004). Aodv routing protocol implementation design. In *24th International Conference on Distributed Computing Systems Workshops, 2004. Proceedings.*, pages 698–703.

[8] Clausen, T. and Jacquet, P. (2003). Optimized link state routing protocol (olsr). https://www.rfc-editor.org/info/rfc3626.

[9] Darroudi, S. M. and Gomez, C. (2017). Bluetooth low energy mesh networks: A survey. *Sensors*, 17(7).

[10] Das, J., Cross, G., Qu, C., Makineni, A., Tokekar, P., Mulgaonkar, Y., and Kumar, V. (2015). Devices, systems, and methods for automated monitoring enabling precision agriculture. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 462–469.

[11] Dunbabin, M. and Marques, L. (2012). Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics & Automation Magazine - IEEE ROBOT AUTOMAT*, 19:24–39.

[12] Erdelj, M., Natalizio, E., Chowdhury, K. R., and Akyildiz, I. F. (2017). Help from the sky: Leveraging uavs for disaster management. *IEEE Pervasive Computing*, 16(1):24–32.

[13] Gupta, L., Jain, R., and Vaszkun, G. (2016). Survey of important issues in uav communication networks. *IEEE Communications Surveys Tutorials*, 18(2):1123–1152.

[14] Haas, Z. J. (1997). A new routing protocol for the reconfigurable wireless networks. In *Proceedings of ICUPC 97 - 6th International Conference on Universal Personal Communications*, volume 2, pages 562–566 vol.2.

[15] Hernández Juan, S. and Herrero Cotarelo, F. (2015). Multi-master ros systems. Technical report, Institut de Robòtica i Informàtica Industrial, Barcelona, Spain.

[16] Hiertz, G., Denteneer, T., Max, S., Taori, R., Cardona, J., Berlemann, L., and Walke, B. (2010). Ieee 802.11s: the wlan mesh standard. *Wireless Communications, IEEE*, 17:104 – 111.

[17] Johnson, D., Ntlatlapa, N., and Aichele, C. (2019). Simple pragmatic approach to mesh routing using batman.

[18] Johnson, D. B., Maltz, D. A., and Broch, J. (2001). Ad hoc networking. chapter DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks, pages 139–172. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[19] Lam, S. S. and Qian, C. (2013). Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch. *IEEE/ACM Trans. Netw.*, 21(2):663–677.

[20] Liu, C. and Wu, J. (2009). Efficient geometric routing in three dimensional ad hoc networks. In *IEEE INFOCOM 2009*, pages 2751–2755.

[21] Morgenthaler, S., Braun, T., Zhao, Z., Staub, T., and Anwander, M. (2012). Uavnet: A mobile wireless mesh network using unmanned aerial vehicles. In *2012 IEEE Globecom Workshops*, pages 1603–1608.

[22] Park, V. D. and Corson, M. S. (1997). A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of INFOCOM '97*, volume 3, pages 1405–1413 vol.3.

[23] Perkins, C. E. and Bhagwat, P. (1994). Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, pages 234–244, New York, NY, USA. ACM.

[24] Portmann, M. (2006). Wireless mesh networks for public safety and disaster recovery applications.

[25] Puri, A. (2005). A survey of unmanned aerial vehicles (uavs) for traffic surveillance. *University of South Florida - internal report.*

[26] Sun, J. and Zhang, X. (2009). Study of zigbee wireless mesh networks. In *2009 Ninth International Conference on Hybrid Intelligent Systems*, volume 2, pages 264–267.

[27] Tiderko, A., Hoeller, F., and Röhling, T. (2016). *The ROS Multimaster Extension for Simplified Deployment of Multi-Robot Systems*, pages 629–650. Springer International Publishing, Cham.

[28] Unwala, I., Taqvi, Z., and Lu, J. (2018). Thread: An iot protocol. In *2018 IEEE Green Technologies Conference (GreenTech)*, pages 161–167.

[29] Whitehead, K. and Hugenholtz, C. H. (2014). Remote sensing of the environment with small unmanned aircraft systems (uass), part 1: a review of progress and challenges. *Journal of Unmanned Vehicle Systems*, 02(03):69–85.

[30] Yang Y, Polycarpou MM, M. A. (2007). Multi-uav cooperative search using an opportunistic learning method. *Journal of Dynamic Systems, Measuremnet, and Control.*, pages 716–728.

[31] Yassein, M. B., Mardini, W., and Khalil, A. (2016). Smart homes automation using z-wave protocol. In *2016 International Conference on Engineering MIS (ICEMIS)*, pages 1–6.

[32] Zeng, Y., Lyu, J., and Zhang, R. (2019). Cellular-connected uav: Potential, challenges, and promising technologies. *IEEE Wireless Communications*, 26(1):120–127.

[33] İlker Bekmezci, Sahingoz, O. K., and Şamil Temel (2013). Flying ad-hoc networks (fanets): A survey. *Ad Hoc Networks*, 11(3):1254 – 1270.