School of Electronics and Computer Science
Faculty of Engineering and Physical Sciences
UNIVERSITY OF SOUTHAMPTON

Author: Daniel George Trickey
August 11, 2020

Project supervisor: Professor Kirk Martinez
Second examiner: Dr Nick Harris

An investigation into RIOT-OS for use in 6LoWPAN sensor networks

A project report submitted for the award of
BSc Computer Science

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

A project report submitted for the award of BSc Computer Science

by Daniel George Trickey

Internet Protocol Version 6 (*IPv6*) over Low-Power Wireless Personal Area Networks (*6LoWPAN*) is used for communication in wireless sensor networks (*WSNs*), often in industrial or environmental sensor networks. *RIOT* is an Open Source operating system for embedded Internet of Things (IoT) devices, with support for *6LoWPAN*. This paper investigates the suitability of *RIOT* for use in *WSNs*, evaluating the available modules in *RIOT* for the various components of the system. A *practical deployment* is built from the selected components and evaluated. *RIOT* is found to be a good alternative to other operating systems available for building *6LoWPAN* networks, such as *Contiki*. It is found that the support for *GoMACH* is suitable for replacing *ContikiMAC*, enabling networks to be built on *RIOT* with duty-cycle support to save power.

Contents

Acknowledgements

This project is dedicated to Kajetan Champlewski, who would always listen to me talk about this project, ask me difficult technical questions and help me further my understanding.

I would like to thank Kirk, my supervisor, for all of his help and support throughout this project and for introducing me to a new area of computer networking.

Statement of Originality

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

Chapter 1

Introduction

IPv6 over Low-Power Wireless Personal Area Networks (*6LoWPAN*) is a standard that enables the transmission of Internet Protocol Version 6 (*IPv6*) over *IEEE 802.15.4* radio networks [1]. It is often used for communication in environmental monitoring and industrial sensor networks.

There are a number of hardware devices available that have *IEEE 802.15.4* compatible radios, and it is useful to be able to easily port software between devices. Portability of code between devices can be achieved by using one of a number of small operating systems, that provide an abstraction layer over the underlying hardware. Use of such an operating system also makes it possible to build sensor networks without having studied the data-sheet of the radio chip-set in detail, opening up development to those with less knowledge of electronics.

There are a number of such operating systems available for embedded devices, including *Contiki*, *ContikiNG*, *TinyOS*, *Zephyr OS* and *RIOT* [2].

In some environments, energy may be a restricted resource, relying on off-grid power sources such as photovoltaic solar panels or batteries. In these situations, it is useful to use as little energy as possible. However, many of the existing implementations of *6LoWPAN* continuously use energy to power the radio or microprocessor unnecessarily.

This project investigates the use of *RIOT* in *6LoWPAN* networks, including the use of some more modern protocols that have been implemented to build a practical system. Different technologies and methods for various network requirements will be investigated and compared.

The suitability of *RIOT* for use in *6LoWPAN* networks will be determined by testing various parts of *RIOT* to determine the best combination of components and then building a proof of concept network based on them. It will explore technologies used to transfer data on and off the network, as well as how data is moved around the network.

A test *practical network* will also be built using a selection of these technologies and the performance of the test network will be evaluated.

Chapter 2

Background

This chapter discusses background information required to understand the findings in this project.

2.1 OSI Model

The *Open System Interconnection* (*OSI*) *Model* defines seven layers describing how networked applications communicate with each other. It does not refer to specific technologies, but a technology can usually be recognised as being at a specific layer, or covering multiple layers [3].

Whilst the *OSI Model* defines 7 layers, only the first 5 are useful for this project. Layers 5 to 7 are often the same application or protocol. Descriptions of the layers, with examples are given in Table 2.1.

OSI Layer	**Description**	**Examples**
Layer 1	Physical Layer	Ethernet, IEEE 802.15.4
Layer 2	Data Link Layer	Ethernet, IEEE 802.15.4
Layer 3	Network Layer	IPv4, IPv6
Layer 4	Transport Layer	TCP, UDP
Layers 5-7	Application Layers	CoAP, HTTP

Table 2.1: OSI Model Layers

2.2 RIOT

RIOT is an Open Source operating system for embedded Internet of Things (*IoT*) devices. It implements a number of technologies that are useful in building wireless

sensor networks, including *6LoWPAN*, *RPL* and *CoAP*. It was built on foundations laid by *Contiki*, but notably takes a different approach to scheduling and code structure.

Contiki offers an event-based kernel using co-operative multi-threading, which leads to a number of limitations where we cannot guarantee when code will be run. *RIOT* uses a "tickless" kernel, with context switching driven by hardware interrupts. This allows for minimum length context switching, and as the scheduler is aware of thread priorities, it is able to use preemptive multitasking, so that a thread can be guaranteed a "slice" of processor time [4]. Additionally, when the processor has no other tasks to perform an "idle" pseudo-thread is scheduled that puts the device into a low power state, increasing battery life of remote devices. *RIOT*'s approach to scheduling gives a number of "real-time" features that enable reliable systems to be built with better guarantees in place.

RIOT has a modular code structure which breaks down code into modules which can be dependent on each other. This makes it possible to have a high degree of code reuse throughout the operating system, with a flexible layer of abstraction. The abstractions allow code written for *RIOT* to easily be ported to a number of "boards". Parameters can be changed as part of the build process, which is a standard open source tool-chain based on the GNU Compiler Collection (GCC) and GNU Make.

Whilst the main hardware requirement for running *RIOT* on a device is at least 1.5kB of Random Access Memory (RAM), it does not require more complex features such as a Memory Management Unit (MMU) or a Floating Point Unit (FPU). However, thanks to the abstractions made by *RIOT*, the operating system is able to make use of features on specific platforms to improve its efficiency. A notable example of this is the use of a Vectored Interrupt Controller (VIC), as found on many Arm processors, for use in the scheduler; this allows the processor to go into deep sleep when the idle thread is scheduled [2]. It is desirable for the processor to be in deep sleep for as much time as possible to extend sensor battery life.

2.3 IEEE 802.15.4

IEEE 802.15.4 is a standard that defines both a physical layer (PHY) and medium access control (MAC) layer for low bandwidth, low power, wireless communication [5]. The standard operates on up to 27 channels, at approximately 868MHz, 900MHz or 2.4GHz, depending on regional restrictions of the unlicensed spectrum [6]. Whilst the MAC layer is defined in the standard as either Carrier Squelch Multiple Access / Collision Avoidance (CSMA/CA), or Time Domain Multiple Access (TDMA) [6], other standards for the MAC layer have also been developed such as ContikiMAC [7] and GoMACH.

ContikiMAC was developed by The Contiki Project as a MAC layer with a radio duty cycling system, allowing nodes to keep their radios off for approximately 99% of the time

[7]. However, the MAC continuously re-sends a packet, waiting for the recipient node to wake up its radio and acknowledge before sending the next packet in the sequence. Alternatives to *ContikiMAC* include *6TiSCH*, which uses the Time Slotted Channel Hopping (TSCH) mode of *IEEE 802.15.4e* [8]. This schedules the radio time into time slots and frequency slots. This results in the radio being off for the majority of the time, but turning on at the right frequency and time to communicate, thus receiving packets without duplication [9].

2.4 IPv6

IPv6 is the latest version of the *Internet Protocol* (IP), and the successor to version 4 (*IPv4*) [1].

2.4.1 Addressing

IPv6 has an address space of 128 bits, which is significantly larger than IPv4's 32-bit address space. The use of IPv6 means that is now possible to give every device in a network a globally-routable IP address, with no need for techniques such as *Network Address Translation* (*NAT*) [10]. Avoiding the use of *NAT* allows for traffic to be globally routed to individual devices.

The 128 bits of address space that *IPv6* has are split up into sub-networks (subnet) or prefixes of various length. A prefix indicates that the first `n` bits are the same, and the remaining `128-n` bits can vary within the network and can be assigned to devices. Typical subnet sizes are 64 bits long, as this allows for the use of *Stateless Address Configuration* (*SLAAC*), which is a scheme allowing devices to automatically determine their own address in the subnet, usually based on their *Ethernet* MAC address.

IPv6 addresses are written using hexadecimal in groups of 16 bits, with a colon separating every 16 bits. The largest section of the address that is all zeros can be abbreviated as `::`. Within any of the groupings, preceding zeros can also be omitted to reduce the length of the written address. For example `2001:db8::8335` is expanded to `2001:0db8:0000:0000:0000:0000:0000:8335`. A prefix is notated using Classless Inter-Domain Routing (CIDR) Notation, which specifies the length of the prefix as an integer after a slash, for example `2001:db8::/32`. It is also common to denote a arbitrary prefix of a certain length as a /n, pronounced "slash n", where n is an integer depicting the length of the prefix.

IPv6 packets are typically sent directly from one device to another device, this class of traffic is referred to as "uni-cast". However, it is possible to send messages to a group of devices, which is referred to as "multi-cast".

The *IPv6* address space is split into a different types of addresses, detailed in Table 2.2.

IPv6 Prefix	Name	Purpose
2000::/3	Global Unicast	Globally routable unicast addresses
fc00::/7	Unique Local Addressing (ULA)	For use in private networks, not globally routable
fe80::/8	Link-local Addressing	For use on layer 2 links only. Not routable
ff00::/8	Multicast addresses	Multicast address space

Table 2.2: Common IPv6 address types.

2.4.2 Neighbour Discovery Protocol (NDP)

The *Neighbour Discovery Protocol* (*NDP*) is a protocol that allows IPv6 enabled devices to discover other hosts and routers on the same network segment. It is the successor to the *Address Resolution Protocol* (*ARP*) that is used for *IPv4* networks [11].

*Router Advertisement*s (RAs) are a part of NDP that is used by IPv6 routers to advertise their presence to other routers and hosts. *RAs* are sent to the all-nodes multi-cast address, which will reach all hosts and routers on the same network segment. *RAs* are also sent in response to a *Router Solicitation* message.

A *Router Advertisement* contains information about the router that sent it, including any prefixes that are on-link, as well as the routes that the router has to other networks. This is usually advertising a default route to the internet. A *RA* also contains a flag to instruct hosts to calculate their address using *SLAAC*.

2.5 6LoWPAN

6LoWPAN defines a format for transmitting *IPv6* packets over *IEEE 802.15.4* networks [12]. This allows for the assignment of global IP addresses to individual nodes in a sensor network. The *Maximum Transmission Unit* (*MTU*) of an *IPv6* packet is 1280 bytes, which is much larger than the *IEEE 802.15.4* PHY packet size of 127 bytes [12]. After the overhead for frame headers and encryption has been accounted for, there are 41 bytes for data per packet. This means that packets over this size are subject to fragmentation, so it is advantageous to keep them below this size [13]. Figure 2.1 shows the fragmentation of a packet in more detail. Applications of *6LoWPAN* include, but are not limited to, home automation, real-time environmental monitoring, smart metering and smart grid infrastructure [14].

A typical *6LoWPAN* network consists of a *Border Router* that bridges between *6LoWPAN* and an *Up-link Router*. The *Up-link Router* has access to wider networks, usually

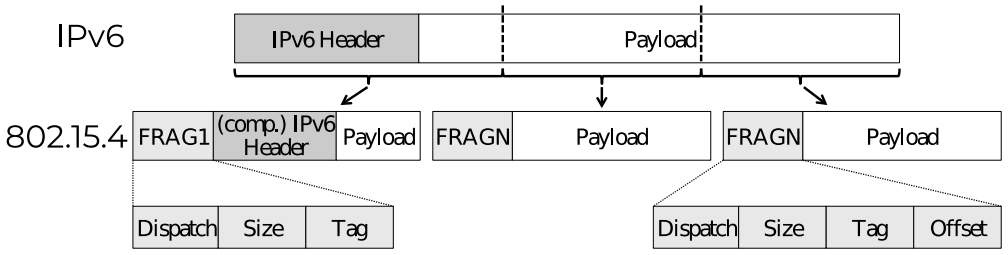
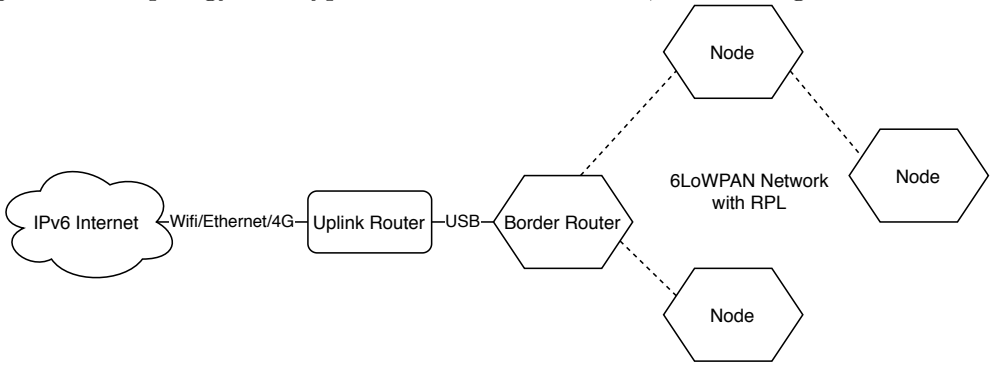


Figure 2.1: Fragmentation of *IPv6* over 802.15.4, adapted from *6LoWPAN*: The wireless embedded Internet [13]

the Internet. The *border router* then connects to a number of remote nodes using *6LoWPAN*, and routes traffic between them and the wider world, as shown in Figure 2.2. Routing protocols such as *RPL* can also be used on the *6LoWPAN* network so that nodes that are not in direct range of the border router can still communicate with it [15].

It is also possible to build networks that have multiple *border routers*, each with a different prefix. Multiple *border routers* can be used to build up-link redundancy into a network, and can be distributed throughout the network to mitigate a physical fault.

Figure 2.2: Topology of a typical *6LoWPAN* network, with a single border router.



2.6 Mesh Networks and Routing

A typical sensor network will have nodes that are in range of at least one other node, which can then establish wireless links between them. A routing protocol can be used to facilitate the transmission of packets to a remote node where a direct link does not exist [16]. This is generally known as a mesh networking topology. The *Routing Protocol for Low-Power and Lossy Networks* (*RPL*) is designed for use in low power, lossy and low bandwidth networks. It is self-healing, meaning that it can continue working around failed nodes, and can support multiple border routers for redundancy [17]. Networks using *RPL* can be simulated in software, for example using *Cooja*,[18] which allows

modifications to the protocol and algorithms to be easily tested without deploying a physical network.

2.7 Protocols above the network layer

The *User Datagram Protocol* (*UDP*) is a communications protocol that allows sending of data over IP with a relatively low frame overhead [19]. Given that the *6LoWPAN* overhead only leaves 41 bytes available per packet, efficiency is essential. Thus *UDP*, with a header size of only 8 bytes, is preferable to alternatives such as *Transmission Control Protocol* (*TCP*) where the header size is a minimum of 20 bytes [20].

The *Constrained Application Protocol* (*CoAP*) is a UDP-based transfer protocol inspired by the Hypertext Transfer Protocol (HTTP) [21]. Designed for low power devices communicating over low bandwidth links, it offers some interoperability with HTTP through the use of a proxy server. A compact encoding scheme reduces the minimum header size to 10 bytes, allowing CoAP messages to be sent in a single *IEEE 802.15.4* packet [22]. This allows real-time systems to be designed to use less power than if using alternatives such as HTTP or MQTT [23].

Chapter 3

Ingress and Egress of Data

For most applications of *6LoWPAN* networks, it is beneficial to be able to move data on and off the network, taking data beyond the *border router* and onto larger networks. This usually requires that a Layer 2 connection is made between the *border router* and the *gateway router*, as the Layer 1 connection is usually dependent on the hardware of the *border router*. The Layer 2 connection can then be used as a point to point link that IP traffic can be routed over.

This chapter explores a number of candidate protocols for the up-link connection, and compares them in this projects use case in WSNs.

3.1 Point-to-Point Protocol

The Point-to-Point Protocol (PPP) is an internet standard for the encapsulation of multi-protocol datagrams over point to point links, such as serial [24]. PPP can be used as the layer 2 link in an IP stack, encapsulating the IP packets within it. PPP can be used to establish links over various media, historically this was mostly serial connections. More recently, PPP is often used as part of broadband services to establish and authenticate connectivity over a Digital Subscriber Line (DSL) [25]. This means that with a suitable PPP implementation and modem, *RIOT* could directly establish a broadband internet connection.

RIOT has some support for PPP in the main code tree. There are also a number of pull requests on *GitHub* that contain never-finished implementations of a full network stack implementation of PPP.

3.2 Serial Line IP

Serial Line IP (*SLIP*) is a de facto protocol standard for transmitting IP packets over serial interfaces. It originated in the 1980's for Berkeley UNIX systems and was documented in RFC 1055.[26] *SLIP* simply defines a sequence of characters to frame IP packets over the serial link, relying on the reliability built into Internet Protocol to ensure delivery of data.

The *SLIP* implementation in RIOT was originally imported from *Contiki* in 2015 [27] and was only compatible with the `gnrc` network stack in *RIOT*. It was eventually ported to the more general purpose `netdev` network layer in 2017 [28].

In testing, the *SLIP* implementation in *RIOT* was not found to be suitable for the use cases considered in this project. When it was tested, it broke *IPv6* connectivity on the Linux-based *up-link router* due to enabling packet forwarding for all interfaces. Linux requires that forwarding is enabled for all interfaces at the same time, it is not possible to enable forwarding for a specific interface. By default, Linux will ignore *RAs* on interfaces with forwarding enabled, thus breaking connectivity. An issue was raised on *GitHub* for this, which has yet to be resolved at the time of writing.

The *SLIP* implementation also required manual setup of IP addresses on both the *border router* and *up-link router*, as *SLIP* is a layer 3 interface.

3.3 Ethernet over Serial

Ethernet over Serial (*ethos*) is a protocol developed for RIOT as a replacement for the `gnrc_slip` network driver. It implements a `netdev` network layer driver, and allows for the transfer of layer 2 Ethernet frames over a serial link. A terminal can also be multiplexed over the interface so that both Ethernet and a serial console can be run over the same link [29].

Ethos requires software support on the *border router*, as well as the *up-link router*. A user-space driver on the up-link router is used to interface with the Linux kernel networking using a "tap" interface. As "tap" is specific to Linux, it is not possible to build an *up-link router* based on Mac OS or Windows using ethos.

3.3.1 Ethernet over Serial with Global Addresses

When using *ethos* with globally routable *IPv6* addresses, it was found that remote nodes were unable to reach the wider internet, whereas the Border Router was able to. This was due to a property of *RIOT*'s source address selection when routing. For remote

nodes, a link-local address was set as the source address and for the *border router*, a global address was used as the source address. The *up-link router* refuses to route any packets with a source address of a link-local due to the configuration of the Linux kernel (the packets would be dropped by the ISP anyway).

In order to resolve this, we must give the *border router* a higher priority address on the up-link interface. This can be achieved by setting it manually adding addresses in the same subnet (usually of size `/127`) on both the Up-link and Border Router. The *border router* will not allow this by default in *RIOT*, so it must be compiled with the following compiler flag added in the `Makefile`: `CFLAGS += -DCONFIG_GNRC_NETIF_IPV6_ADDRS_NUMOF=3`. This flag instructs the compiler to allocate additional memory to store more addresses on the interface. We need at least three addresses because two are taken up by the link-local and multi-cast all-nodes address, meaning that a third is required for the global address.

An alternative way to add addresses to the link is to send *Router Advertisements* from the *up-link router*. On Unix-based *up-link routers* this can easily be achieved using `radvd`, which is included in most distributions. *RIOT* will listen to the *RAs* and assign an address on the up-link interface using *SLAAC*.

3.4 USB Ethernet

RIOT has recently added native support for *Universal Serial Bus* (*USB*) using the "usbus" module [30], which means that it is possible to use protocols that use USB rather than serial to communicate with the up-link router. Using this module, it is possible to configure the device with a USB Communications Device Class Ethernet Network Control Model (CDC-ECM) type endpoint, usually just referred to as USB Ethernet. This allows the node to appear as a USB Ethernet adapter to the up-link router, so Ethernet traffic can be passed over the interface.

A nice property of using USB Ethernet is that the majority of operating systems include support for it natively, meaning that no user-space software is needed on the up-link router. As part of this project, USB Ethernet support was added to the border router example code on *RIOT*, giving it equivalent support from *RIOT* as *ethos* and *SLIP* [31]. This means that the example is now able to setup and use the CDC-ECM network interface simply by passing an argument when flashing the node.

As USB Ethernet requires use of a native USB interface, it requires the use of hardware that supports that. For example on the SAMR21 board, you need to use two micro USB cables to connect to your computer, one for network and one for serial, flashing and debugging. However, this also allows use of devices that do not have a USB serial connection exposed, such as the NRF 52840 Dongle from Nordic Semiconductor.

USB Ethernet was found to work very well in testing, with essentially no configuration required on the *up-link router* or *border router*.

Chapter 4

Radio Medium Access Control

In IEEE 802 standards, including *IEEE 802.15.4*, the Medium Access Control (MAC) sub-layer is the layer that is responsible for controlling the underlying hardware of the link, usually wireless, wired or optical. The MAC sub-layer is part of Layer 2 of the *OSI Model*. It is often used to control access to the medium being used to transmit data. In radio networks, extra care needs to be taken when designing MACs as radio is a shared medium that could be in use by multiple nodes at once. If multiple nodes transmit simultaneously, it could cause loss of data.

This chapter investigates some of the MAC implementations that are available, along with their advantages and disadvantages.

4.1 Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA)

Carrier-Sense Multiple Access with Collision Avoidance (*CSMA/CA*) is the default MAC for the `gnrc` *6LoWPAN* implementation in *RIOT*. *CSMA/CA* attempts to avoid collisions with other devices by first checking to see if another device is transmitting, and only transmitting when the channel is determined to be "idle".

Usually *CSMA/CA* includes an additional mechanism to check if the airspace is available using request-to-send (RTS) and clear-to-send (CTS) messages. However, these are not included in *CSMA/CA* for *IEEE 802.15.4* [32].

CSMA/CA is not particularly power efficient as the radio needs to remain on to listen for incoming transmissions from other nodes.

4.2 IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH)

IPv6 over the TSCH mode of IEEE 802.15.4e (*6TiSCH*) is a MAC that uses Time Slotted Channel Hopping (TSCH). TSCH makes use of Time Division Multiple Access (TDMA), where a channel is split into a number of slots, usually about 10 milliseconds in length. It also utilises multiple channels (16 for 802.15.4), giving a number of slots per channel. Nodes will sleep, transmit or receive according to the *schedule* [33].

RIOT has support for *6TiSCH* by utilising the *OpenWSN* implementation, which was created by University of California Berkeley. It should be noted that the integration into *RIOT* is intended to be used as an entire package with the rest of the *OpenWSN* stack [34]. Support for *OpenWSN* in *RIOT* was added since this project began, so it was not covered in the initial research for this project.

4.3 GoMACH

GoMACH builds on the work of *6TiSCH* and *ContikiMAC* to combine the TSCH mode of *IEEE 802.15.4e*, with the duty-cycle sleep mode of *ContikiMAC* to build an efficient and reliable MAC with good power efficiency [35]. *GoMACH* is implemented as a module in *RIOT* and integrates well with *6LoWPAN* and *RPL*.

It is quite simple to add support for *GoMACH* to an existing *RIOT* project by adding `USEMODULE += gnrc_gomach` to the `Makefile`. It should be noted that this should be present on all nodes in the network, as it is not compatible with the default `gnrc` implementation.

Chapter 5

Address Assignment

Once a layer 2 link has been established, we also need to establish connectivity at Layer 3. In *6LoWPAN*, this is always *IPv6*, although *RIOT* does support other Layer 3 protocols also, such as LoRa and LoRaWAN.

It is important that both the *up-link router* and *border router* know what *IPv6* prefix is to be used on the wireless network. This includes the setup of appropriate routing tables so that traffic can go back and forth.

5.1 Static Addressing

At a high level, the simplest option appears to be setting the address prefix on the *border router* statically, hard-coding it in the firmware when the node is flashed. Static routes can then be set on the *up-link router* using tools such as "iputils". However, *RIOT* does not support setting this information in such a manner so easily, so this requires calling internal functions to the OS that are not guaranteed to be stable between versions because they are not part of the public API. These functions must be called as early as possible when the *border router* is booted, with a /64 set on the wireless interface, so that the prefix can be propagated to other nodes on the network. When RPL is used on the network, it must also be configured manually.

Using static addressing does introduce a degree of inflexibility, as the *border router* must be re-flashed if the prefix changes. This would cause issues when deploying networks where the prefix is not constant, which could occur if a WSN is using a residential broadband connection to reach the Internet.

5.2 Dynamic Host Configuration Protocol for IPv6 (DHCPv6)

Dynamic Host Configuration Protocol for IPv6 (*DHCPv6*) is a protocol for the management of *IPv6* devices on a network, most commonly used to inform devices of Domain Name System (DNS) servers and assign *IPv6* addresses. It is often used as an alternative to *SLAAC* on networks that require closer monitoring of devices [36].

DHCPv6 Prefix Delegation (*DHCPv6PD*) is an extension to *DHCPv6* that allows a requesting router to request a *IPv6* prefix from a delegating router. The delegating router will route all traffic for this prefix to the requesting router, and does not assume any knowledge of the network topology behind the requesting router [37].

RIOT can act as the requesting router to request an *IPv6* prefix from the *up-link router*. Using *DHCPv6PD*, the configuration on both the *up-link router* and *border router* can be entirely automatic. A *DHCPv6* server will need to be run on the *up-link router*, for which there is open source software available, such as *Kea* [38].

5.3 Micro Host Configuration Protocol (UHCP)

Micro Host Configuration Protocol (*UHCP*) is a simple protocol developed for *RIOT* that will automatically configure an *IPv6* prefix on a *border router* [39].

UHCP requires support on both the *border* and *up-link routers*. The *up-link router* should run the `uhcpd` daemon with the desired network prefix configured, and the *border router* should have the `gnrc_uhcpc` module included when the firmware is flashed. `uhcpd` will listen on the multi-cast address `ff15:abcd` for *UHCP* requests over *UDP* port 12345 from the *border router*. The *border router* will send UHCP requests every 10 seconds until it receives a *UHCP* push message, which contains a prefix. Once a prefix has been received, the *border router* will make requests every 60 seconds and update the prefix.

It should be noted that *UHCP* is specific to *RIOT*, and is not supported in any other operating system. An *up-link router* running `uhcpd` can only provide connectivity to exactly one *border router*, and can only give out a /64 prefix to the *border router*.

5.4 Use of Public IP Space

When deploying real-world sensor networks using *6LoWPAN*, it is often desirable to make use of public uni-cast *IPv6* space. This makes it possible to send a message to a specific node on your network from any computer with *IPv6* connectivity.

This would enable nodes to send data directly back to academic institutions for processing in real-time. *CoAP* requests could also be made directly to nodes in the network, without the need for a HTTP to CoAP or IPv4 to IPv6 proxy.

It should be noted that if public *IPv6* space is used for a network, a firewall with appropriate rules should be in place. Ideally, requests should be rate-limited or limited to requests from trusted devices and institutions.

5.4.1 DHCPv6

It is also common for typical Internet Service Provider (ISP) Customer-Premises Equipment (CPE) to run a *DHCPv6* server with support for prefix delegation. Thus, if a layer 2 connection can be established between the *border router* and the CPE, the prefix can be directly allocated by the CPE. A simple way to achieve this is using a small Linux computer, such as a Raspberry Pi, and creating a "Bridge" interface, which acts as a virtual *Ethernet switch*. Using this technique can make it very easy to use public *IPv6* space on field deployments.

Creating a bridge can be performed on any Linux system with `bridge-utils` installed, although firewall rules on the system should be checked to ensure that they do not block traffic. This can be achieved by running the following commands:

```
# brctl add br0
# brctl addif br0 <interface-to-cpe>
# brctl addif br0 <border-router-interface>
# ifconfig up br0
```

If your system is running *NetworkManager*, you will need to configure it to ignore interfaces used by RIOT, as it will interfere with the configuration. It should also be noted that DHCPv6 does not work with SLIP.

5.4.2 UHCP

It is also possible to configure static addressing using *UHCP*. A route of at least /64 to the *up-link router* will need to be configured, and traffic should be allowed through the firewall as desired. The *UDP* port 12345 should be accessible from the *border router* through any firewalls so that UHCP can allocate the network prefix.

To use CDC-ECM as the up-link on the `gnrc_border_router` example, the following command should be run:

```
make IPV6_PREFIX=2a02:8010:659d:8001::/64 UPLINK=cdc-ecm flash term
```

If using `ethos`, add `CFLAGS += -DCONFIG_GNRC_NETIF_IPV6_ADDRS_NUMOF=3` to the Makefile.

```
make IPV6_PREFIX=2a02:8010:659d:8001::/64 UPLINK=ethos flash term
```

The `ifconfig` tooling will then need to be used on the *RIOT* command line to configure an additional public address on the wired interface. This circumvents an issue where *RIOT* chooses a link-local address as the source address, preventing traffic from reaching beyond the up-link router. A packet capture of this issue is depicted in Figure 5.1.

5448	1509.3100199…	fe80::299:71ff:febf:bd4	2001:470:694c:80::1	ICMPv6	66 Echo (ping) request id=0x33bf, seq=0, hop limit=64 (no response found!)
5449	1509.3100444…	fe80::b02b:e3ff:fe3d:8647	fe80::299:71ff:febf:bd4	ICMPv6	114 Destination Unreachable (Beyond scope of source address)
5450	1510.3141612…	fe80::299:71ff:febf:bd4	2001:470:694c:80::1	ICMPv6	66 Echo (ping) request id=0x33bf, seq=1, hop limit=64 (no response found!)
5451	1510.3141882…	fe80::b02b:e3ff:fe3d:8647	fe80::299:71ff:febf:bd4	ICMPv6	114 Destination Unreachable (Beyond scope of source address)
5452	1511.3185047…	fe80::299:71ff:febf:bd4	2001:470:694c:80::1	ICMPv6	66 Echo (ping) request id=0x33bf, seq=2, hop limit=64 (no response found!)
5453	1511.3185256…	fe80::b02b:e3ff:fe3d:8647	fe80::299:71ff:febf:bd4	ICMPv6	114 Destination Unreachable (Beyond scope of source address)

Figure 5.1: Link-local source address issues when using ethos as the up-link

Chapter 6

Building a Practical Network

This chapter explores building a "practical" *6LoWPAN* network using SAMR21 XPlained Pro Evaluation boards running *RIOT* as the underlying OS.

6.1 Network Setup

The following technologies and protocols were selected for the network:

- *USB Ethernet* to link between the *up-link router* and *border router*.
- *UHCP* to assign the PAN subnet from global address space.
- *GoMACH* as the *IEEE 802.15.4* MAC.
- *RPL* as a routing protocol to allow the network to continue functioning when a node fails (fail-over) and mesh routing within the network.

UHCP was chosen over *DHCPv6PD* as `gnrc_uhcpc` supports automatically setting up *RPL* with the prefix, whereas that is not the case for `dhcp6c`.

A basic *CoAP* server was run on nodes in the test network to ensure connectivity was working.

The Makefile and C code used for this test is available in the Data and Design Archive.

The Code used for the test was based on the `gnrc_border_router` example, using the `cdc-ecm` up-link that was implemented for chapter 3. The CoAP code was adapted from the `nanocoap_server` example included in *RIOT*.

6.2 Up-link Router

A laptop running *Arch Linux* was used as the *up-link router*, with the Internet connection over WiFi. A static *IPv6* route for `2a02:8010:659d:6000::/56` to the *up-link router* was configured on my ISP-supplied router CPE so that a global prefix could be used for the *6LoWPAN* subnet.

As this was a testing environment, the laptop was not configured as a router so a few adjustments to the configuration needed to be made. Firstly, the laptop was running *NetworkManager* which was configured to ignore any USB Ethernet devices so that it did not interfere with the configuration. This was achieved using a `udev` rule that added `ENVNM_UNMANAGED="1"` to any devices that loaded the `cdc_ether` kernel module.

In order for packets to be routed from the network to the wider internet, *IPv6* forwarding needed to be enabled on the *up-link router*, which can be achieved by changing the Linux kernel parameter `sudo sysctl net.ipv6.conf.all.forwarding` to be `1`.

Forwarding must be enabled on all interfaces, which means that *Router Advertisements* are also ignored on all interfaces. To re-enable listening for *RAs*, the kernel parameter `net.ipv6.conf.<uplink-interface>.accept_ra=2` must be set. If this is not done, then the *up-link router* is not able to route to the internet.

Firewalling should be setup appropriately on the *up-link router* to ensure that packets can be routed. As this test was occurring in a trusted environment, the firewall was just disabled, although it should be noted that this is not good practice and should never be done on an untrusted network.

Once the SAMR21 for the *border router* is plugged in with both USB cables, it can be flashed and start running as the *border router*. This can be seen in Figure 6.1.

```
sudo make IPV6_PREFIX=2a02:8010:659d:6001::/64 UPLINK=cdc-ecm flash term
```

The Remote Nodes for the network were flashed and will automatically connect to the network when started. It may be used to check the serial terminal on them to retrieve their IP address for testing.

6.3 Evaluation of setup

This deployment was tested with two remote nodes, and one *border router* as shown in Figure 6.2.

Bidirectional connectivity between all nodes on the network was achieved, as well as between remote nodes and the *up-link router*. This was tested by sending *Internet*

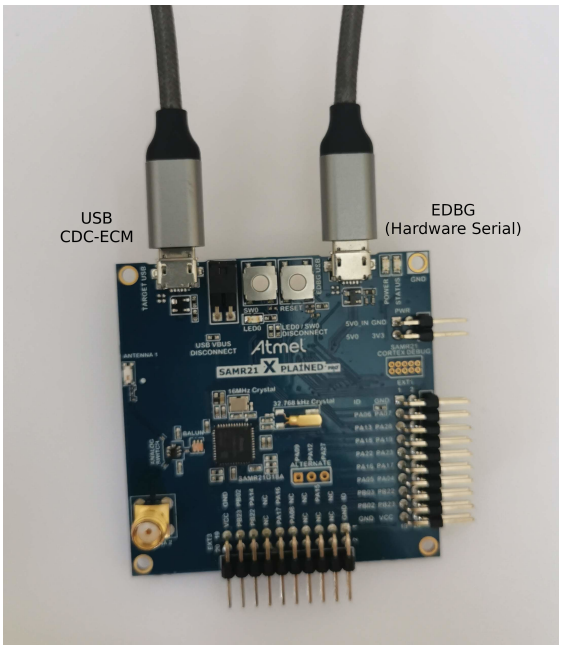


Figure 6.1: SAMR21 Evaluation Board in use as an up-link router.

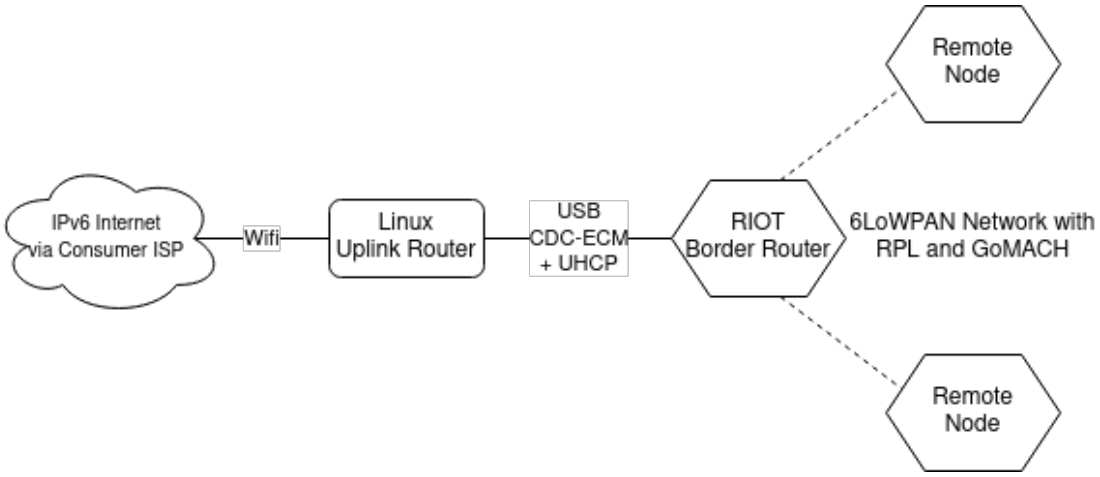


Figure 6.2: Topology of the test network.

Control Message Protocol (*ICMP*) ping requests. *CoAP* was also tested from the *up-link router* to the nodes.

Nodes were unable to reach the wider internet and vice versa on the test setup. However, packets were successfully leaving the network and being sent to the CPE from the *up-link router*. The problem was most likely caused by an improperly configured firewall on the CPE, which could not be adjusted. It was found that other devices behind the ISP Router were able to access the *6LoWPAN* network.

On a previous, similar setup that was built as part of this project, full internet connectivity was achieved. This was achieved using infrastructure that was not available for the

final test setup, and did not use *GoMACH* or *USB Ethernet*, instead using *CSMA/CA* and *ethos*.

GoMACH reported that the achieved Duty Cycle was 0%, although this was due to the debug shell being enabled on all nodes. As the shell process was constantly running, the nodes were never able to run the `idle` thread, so did not enter sleep mode.

Chapter 7

Progress

7.1 Achievement

Throughout the project, the following was achieved:

- A review of background literature and reading of relevant topics, including technical documents underlying many of the protocols that are implemented in RIOT and utilised in this project.
- Testing and selection of appropriate hardware for the project. Hardware tested included the Zolertia Z1, Atmel SAMR21 Xplained Pro Evaluation Board and the Nordic Semiconductor NRF52840.
- Testing of configurations of the CDC-ECM implementation in RIOT in order to get it working reliably. This was tested against existing link types in RIOT.
- Writing an extension for the RIOT `gnrc_border_router` example code to add CDC-ECM as an up-link. This was submitted to the RIOT GitHub project, accepted and merged, see Figure 7.1. This extension was later used to build the *practical network*.
- Reading on how to configure Linux as a router, for use in building Up-link Routers. This was then used to build a Virtual Private Network (VPN) configuration to route public IPv6 space to my laptop for testing purposes. Unfortunately this was not available throughout the entire project as the infrastructure that it was hosted on became unavailable.
- Investigation and testing of a number of MAC implementations in RIOT.
- Building a practical network using the technologies that were investigated. This was tested as much as was possible.

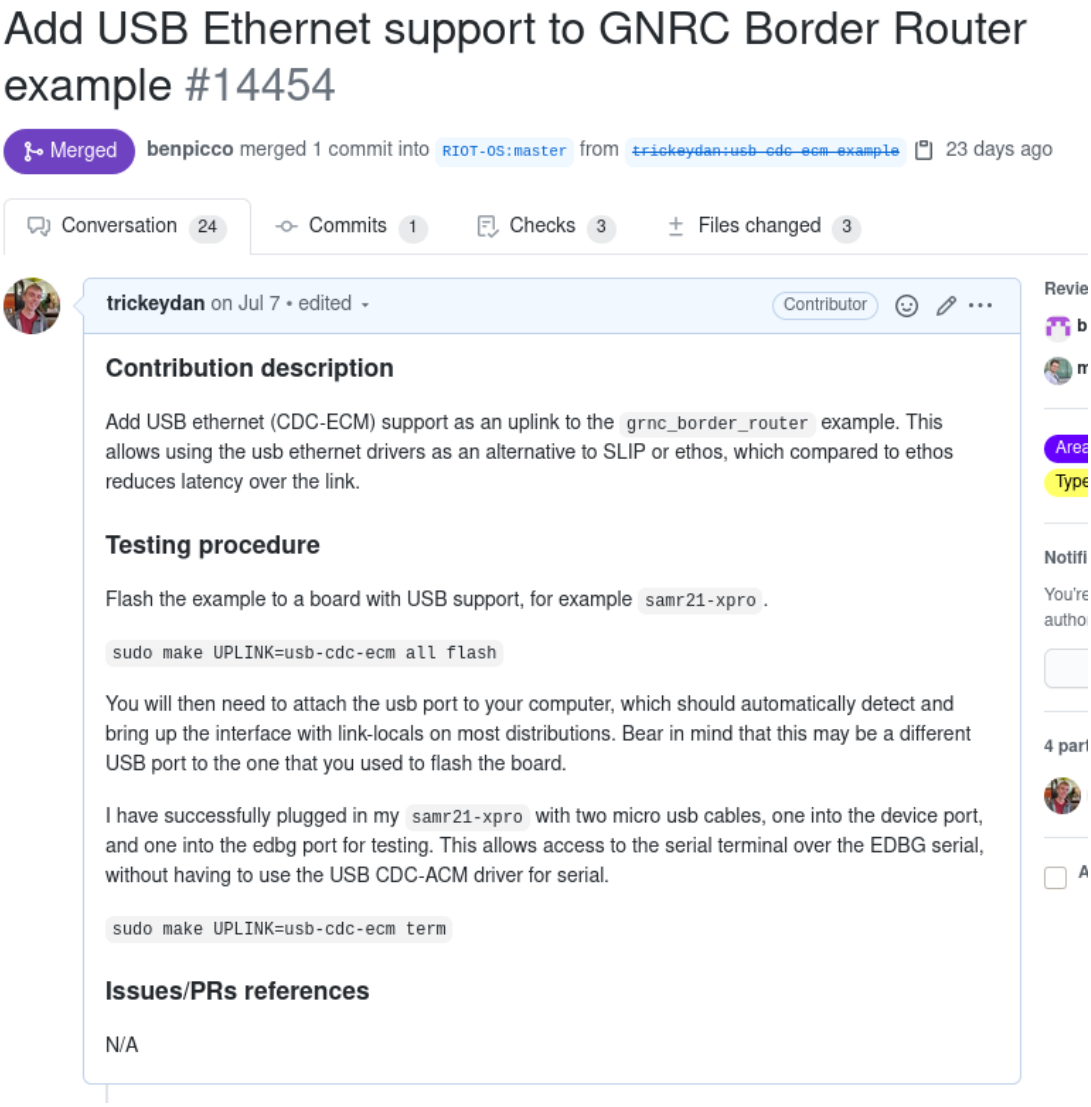


Figure 7.1: Pull Request for CDC-ECM Up-link

Unfortunately, I was unable to do everything that I wanted to do in this project due to limitations from the COVID-19 pandemic. This included, but was not limited to:

- Power usage measurements of different MACs were not taken, as access to the lab bench equipment required was not possible.
- The meshing and fail-over capabilities of RPL when used with different MACs could not be adequately tested. This was caused by there not being enough space in my house to reduce the signal strength enough such that not all nodes could reach the border router. Originally, I had planned to place nodes spaced throughout one of the buildings at the university to achieve the desired range.
- When testing the *practical network*, global internet connectivity was restricted by my residential ISPs firewall.

7.2 Project Management

7.2.1 Schedule

A schedule for the project was originally laid out in the progress report, at which time the aim of the project was closer to developing a duty-cycle scheme similar to *GoMACH*, albeit with larger periods. It was determined that the original scope of the project was too large, so I instead focused on the use of *RIOT* for sensor networks with a particular focus on getting *USB Ethernet* working reliably.

A timeline of the progress of the project can be found in Figure 7.2. This can be compared to Figure 7.3 which shows the original plan for the project.

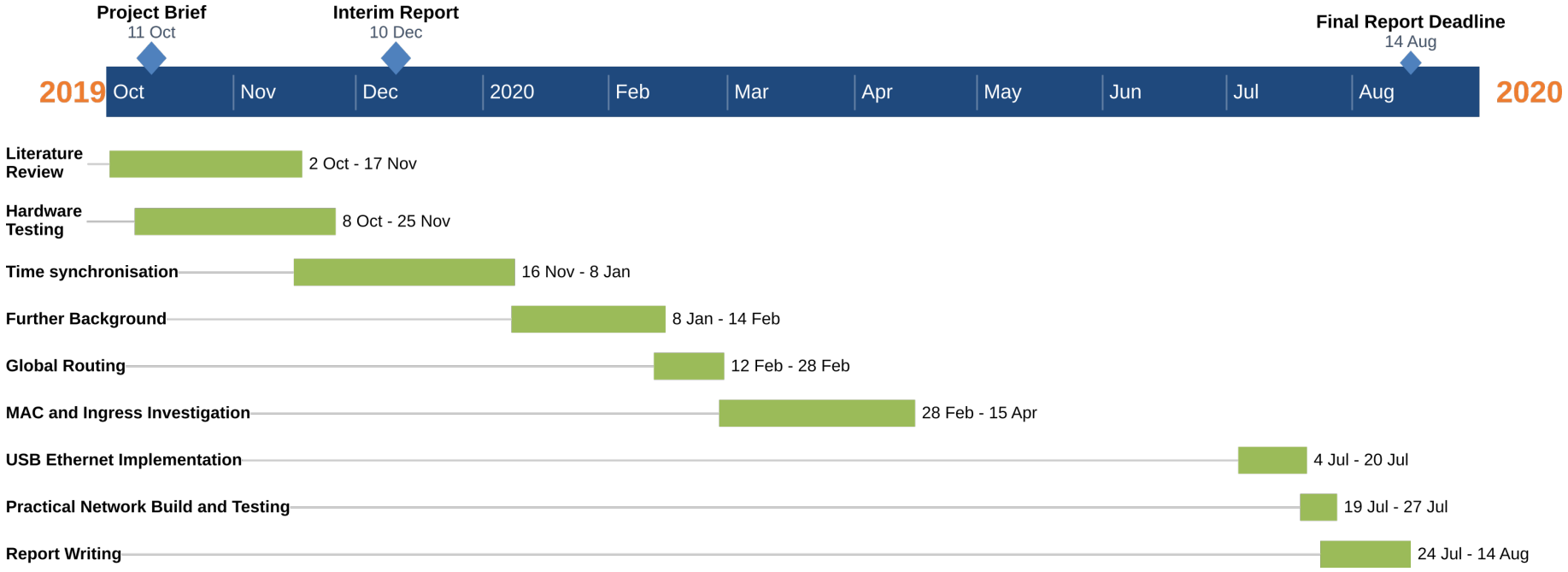


Figure 7.2: Gantt Chart showing actual progress

Figure 7.3: Gantt Chart showing planned progress

7.2.2 Risks

Risks for the project were evaluated in the early stages of the project and it was determined that there were no major risks at that time. Minor risks such as stress from other deadlines and hardware failure were considered but were not determined to be significant. In March 2020, it became clear that the *COVID-19* pandemic presented a significant risk to the project; however a global pandemic was deemed to be unlikely at the time of risk assessment.

It should be noted that there was a large period of no progress on this project between April and June. This was caused by the unexpected passing of a close friend of mine, resulting in a long period where I was unable to work and reduced productivity when work on the project resumed. The impact of these factors on the project was mitigated by a deadline extension granted by the Special Considerations Board.

7.2.3 Tools

In addition to the Gantt charts above, I mainly used GitHub Projects as a Kanban-style board to help manage the remaining tasks for the project. This was chosen because it is a technique I have used before both individually and in teams. Use of a project board like this allows for an overview of progress to be seen at a glance, with details available when a card is selected.

Chapter 8

Conclusions and Further Work

The investigations conducted during this project have shown that *RIOT* is suitable for building sensor networks. It works particularly well when combined with using *USB Ethernet* as an up-link, which in combination with *DHCPv6* can be used to easily build sensor networks that use global *IPv6* addresses. This makes it possible for nodes to be queried over the internet, and also for nodes to directly submit data back to a server that is gathering data, greatly simplifying the build of networks. This will mean that it is possible to spend less time on building infrastructure for sensor networks, instead focusing on designing and building the sensor nodes.

There exists a number of paths that could be taken for future work on this area. Whilst this project tested the implementation of *GoMACH* in *RIOT*, and found it to work over short distances, it was not possible to test it in a more realistic scenario. *GoMACH* should be tested to see if it adversely affects *RPL*, and how well the modulation works over long distances. It would also be beneficial to measure the power usage of *GoMACH* against the default *CSMA/CA* implementation to see if the additional complexity is worth it.

Whilst it was found to be possible to get *DHCPv6* and *RPL* to work on *RIOT*, it did require manual configuration of *RPL*. Whereas for *UHCP*, the configuration of *RPL* is implemented automatically. It would be beneficial to build similar functionality for *DHCPv6*, as this would make it easier to build completely auto-configuring *border routers*.

In this project, the *up-link router* was manually configured for testing purposes. However, for real-world deployments this is not practical, so work could be done to build an *up-link router* that is very easily configured for field use. This could be based off a single board computer such as a Raspberry Pi, and provide monitoring and statistics from the *6LoWPAN* network.

Investigations in this project have focused on networks with a single *border router*, which leaves a single point of failure in the network. This is undesirable for most scenarios, so it would be beneficial to build networks with multiple *border routers*. This would likely require multiple *up-link routers* also, and would require traffic to be dynamically routed to the right destination based on the current environment.

Bibliography

[1] R. Hinden, "Internet protocol, version 6 (ipv6) specification," 2017.

[2] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "Riot os: Towards an os for the internet of things," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2013, pp. 79–80.

[3] N. Briscoe, "Understanding the osi 7-layer model," *PC Network Advisor*, vol. 120, no. 2, pp. 13–16, 2000.

[4] K. Roussel, Y.-Q. Song, and O. Zendra, "Riot os paves the way for implementation of high-performance mac protocols," *arXiv preprint arXiv:1504.03875*, 2015.

[5] R. Heile, "Part 15.4: wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans)," *IEEE Computer Society*, 2003.

[6] J. T. Adams, "An introduction to ieee std 802.15. 4," in *2006 IEEE Aerospace Conference*. IEEE, 2006, pp. 8–pp.

[7] A. Dunkels, "The contikimac radio duty cycling protocol," 2011.

[8] M. R. Palattella, P. Thubert, X. Vilajosana, T. Watteyne, Q. Wang, and T. Engel, "6tisch wireless industrial networks: Determinism meets ipv6," in *Internet of Things*. Springer, 2014, pp. 111–141.

[9] S. Duquennoy, A. Elsts, B. Al Nahas, and G. Oikonomo, "Tsch and 6tisch for contiki: Challenges, design and evaluation," in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2017, pp. 11–18.

[10] T. Savolainen, J. Soininen, and B. Silverajan, "Ipv6 addressing strategies for iot," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3511–3519, 2013.

[11] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Rfc 4861: Neighbor discovery for ip version 6 (ipv6)," *Request for Comments*, 2007.

[12] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Rfc 4944: Transmission of ipv6 packets over ieee 802.15. 4 networks," *Request for Comments*, 2007.

[13] R. Hummen, J. Hiller, H. Wirtz, M. Henze, H. Shafagh, and K. Wehrle, "6lowpan fragmentation attacks and mitigation mechanisms," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*. ACM, 2013, pp. 55–66.

[14] Z. Shelby and C. Bormann, *6LoWPAN: The wireless embedded Internet*. John Wiley & Sons, 2011, vol. 43.

[15] A. Fabre, K. Martinez, G. M. Bragg, P. J. Basford, J. Hart, S. Bader, and O. M. Bragg, "Deploying a 6lowpan, coap, low power, wireless sensor network," in *Proceedings of the 14th ACM conference on embedded network sensor systems CD-ROM*, 2016, pp. 362–363.

[16] J. Hui and J. Vasseur, "Rfc 6553: The routing protocol for low-power and lossy networks (rpl) option for carrying rpl information in data-plane datagrams," *Request for Comments*, 2012.

[17] O. Gaddour and A. Koubâa, "Rpl in a nutshell: A survey," *Computer Networks*, vol. 56, no. 14, pp. 3163–3178, 2012.

[18] N. Tsiftes, J. Eriksson, N. Finne, F. Österlind, J. Höglund, and A. Dunkels, "A framework for low-power ipv6 routing simulation, experimentation, and evaluation," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 479–480, 2011.

[19] J. Postel, "Rfc 768: User datagram protocol," *Request for Comments*, 1980.

[20] ——, "Rfc 793: Transmission control protocol," *Request for Comments*, 1981.

[21] C. Bormann, A. P. Castellani, and Z. Shelby, "Coap: An application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, no. 2, pp. 62–67, 2012.

[22] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained application protocol (coap), draft-ietf-core-coap-13," *Orlando: The Internet Engineering Task Force–IETF*, 2012.

[23] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE international systems engineering symposium (ISSE)*. IEEE, 2017, pp. 1–7.

[24] W. Simpson, "Rfc 1661: The point-to-point protocol (ppp)," *Request for Comments*, 1994.

[25] T. C. Kwok, "Residential broadband architecture over adsl and g.lite (c.992.2): Ppp over atm," *IEEE Communications Magazine*, vol. 37, no. 5, pp. 84–89, 1999.

[26] J. Romkey, "Rfc 1055: A nonstandard for transmission of ip datagrams over serial lines: Slip," *Request for Comments*, 1988.

[27] M. Lenders, "ng_slip: initial import," Feb 2015. [Online]. Available: https://github.com/RIOT-OS/RIOT/pull/2688

[28] ——, "slip: port to be used with netdev," Oct 2017. [Online]. Available: https://github.com/RIOT-OS/RIOT/pull/7381

[29] K. Schleiser, "drivers: add ethernet over serial driver," Feb 2016. [Online]. Available: https://github.com/RIOT-OS/RIOT/pull/4438

[30] K. Zandberg, "Usbus: Initial work towards an usb stack," Jun 2019. [Online]. Available: https://github.com/RIOT-OS/RIOT/pull/10916

[31] D. Trickey, "Add usb ethernet support to gnrc border router example," July 2020. [Online]. Available: https://github.com/RIOT-OS/RIOT/pull/14454

[32] J. Zheng and M. J. Lee, "A comprehensive performance study of ieee 802.15. 4," *Sensor network operations*, vol. 4, pp. 218–237, 2006.

[33] E. Municio, G. Daneels, M. Vučinić, S. Latré, J. Famaey, Y. Tanaka, K. Brun, K. Muraoka, X. Vilajosana, and T. Watteyne, "Simulating 6tisch networks," *Transactions on Emerging Telecommunications Technologies*, 09 2018.

[34] Francisco, "pkg/openwsn: re-integrate the network stack as a package," June 2020. [Online]. Available: https://github.com/RIOT-OS/RIOT/pull/13824

[35] S. Zhuo and Y. Song, "Gomach: A traffic adaptive multi-channel mac protocol for iot," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, Oct 2017, pp. 489–497.

[36] M. Carney, C. Perkins, T. Lemon, B. Volz, J. Bound, and E. R. Droms, "Rfc 3315: Dynamic host configuration protocol for ipv6 (dhcpv6)," *Request for Comments*, 2003.

[37] O. Troan and R. Droms, "Rfc 3633: Ipv6 prefix options for dynamic host configuration protocol (dhcp) version 6," *Request for Comments*, 2003.

[38] Y. Cui, Q. Sun, K. Xu, W. Wang, and T. Lemon, "Configuring ipv4 over ipv6 networks: Transitioning with dhcp," *IEEE Internet Computing*, vol. 18, no. 3, pp. 84–88, 2014.

[39] K. Schleiser, "simplified border router setup," January 2016. [Online]. Available: https://github.com/RIOT-OS/RIOT/pull/4725

Appendix A: Data and Design Archive

The following table contains information about the contents of the Data and Design Archive:

Directory / File Name	**Description**
`border_router`	Firmware for the Border Router used in the "Practical Network" and demo
`cdc-ecm-example.patch`	Git patch to add cdc-ecm support to RIOT's gnrc_border_router example. This was submitted and merged to RIOT.
`remote_node`	Firmware for the remote nodes used in the "Practical Network" and demo
`README.txt`	Description of the contents of the archive