



FACULTY OF SCIENCE & TECHNOLOGY

BSc (Hons) Computer Networks
May 2019

An Automatic Peer to Peer Network Configuration of
Autonomous Devices

by

Jamie Murray

Faculty of Science & Technology
Department of Computing and Informatics
Final Year Project

Abstract

Infrastructure based wireless networks restrict the freedom of movement of connected nodes as they all need to be within range of a centralised network device to communicate. Peer to Peer wireless mesh networks do not suffer from this restriction allowing nodes to communicate directly or by relaying traffic through network peers. The configuration of wireless networks can also be a time-consuming process and if not done properly a network may not even be formed, an automated solution would alleviate these problems. This project focuses on researching, designing, implementing and evaluating an automated Peer to Peer wireless networking solution to overcome the stated limitations.

The autonomous devices at the centre of the project are Sun Founder Pi Car S robots, each containing a Linux based Raspberry Pi. The robots are used by Bournemouth University and other educational establishments as a platform for Science, Technology, Engineering and Maths (STEM) learning and robotics projects. By simplifying and speeding up the network configuration phase the end users can get straight down to business.

The solution consists of a combination of a newly created system service, Python and Bash scripting, and the B.A.T.M.A.N Adv layer 2 routing protocol. All the must have project requirements have been achieved, extensive testing carried out and passed. The result is a scalable and robust networking solution providing a fully converged ad-hoc mesh network in under 25 seconds. It contains a unique addressing method, a means of IP discovery and a simple method of updating or defaulting the network configuration.

Dissertation Declaration

I agree that, should the University wish to retain it for reference purposes, a copy of my dissertation may be held by Bournemouth University normally for a period of 3 academic years. I understand that once the retention period has expired my dissertation will be destroyed.

Confidentiality

I confirm that this dissertation does not contain information of a commercial or confidential nature or include personal information other than that which would normally be in the public domain unless the relevant permissions have been obtained. In particular any information which identifies a particular individual's religious or political beliefs, information relating to their health, ethnicity, criminal history or sex life has been anonymised unless permission has been granted for its publication from the person to whom it relates.

Copyright

The copyright for this dissertation remains with me.

Requests for Information

I agree that this dissertation may be made available as the result of a request for information under the Freedom of Information Act.

Signed: Jamie Murray

Name: Jamie Murray

Date: 08-05-19

Programme: BSc (Hons) Computer Networks

Original Work Declaration

This dissertation and the project that it is based on are my own work, except where stated, in accordance with University regulations.

Signed: Jamie Murray

Name: Jamie Murray

Date: 8-05-19

Acknowledgments

I would like to thank my fabulous project supervisor Dr Natalia Chechina for all the excellent support and guidance throughout this project and helping make it a success.

Melanie Coles, from day one to the very end, you have supported me and inspired me to achieve. It is true to say that I wouldn't be where I am today without your help.

John Murray, my eldest son and my best friend, there through the good times and the bad times. It's an honour and a privilege to be your Dad.

Thank you all from the bottom of my heart.

TABLE OF CONTENTS

1	INTRODUCTION	1
1.1	Project Background & Context	1
1.2	Aims & Objectives	1
1.3	Success Criteria.....	2
1.4	Risk Analysis	2
2	BACKGROUND STUDY	3
2.1	Networks Overview	3
2.1.1	Wireless Local Area Network (WLAN) Standards	3
2.1.2	Wireless Personal Area Network (WPAN) Standards.....	3
2.1.3	Low Powered Wireless Local Area Network (LP-WLAN) Standards.....	3
2.1.4	The OSI Model.....	4
2.1.5	Network Topologies	4
2.2	Connectivity Technology in IoT & Distributed Robotics	7
2.3	Standards & Protocols	8
2.3.1	IEEE 802.11 Standard	8
2.3.2	Bluetooth Standards	9
2.3.3	IEEE 802.15.4 Standard	10
2.3.4	IEEE 802.11 Wireless Mesh Routing Protocols.....	11
2.4	Sun Founder Pi Car S Robots.....	13
2.4.1	Raspberry Pi.....	14
2.4.2	Raspbian	15
2.5	Automatic Configuration & Tools.....	17
2.5.1	Azure Portal IoT Hub	17
2.5.2	Ansible.....	18
2.5.3	Custom Scripting	18
2.6	Summary	19
3	METHODOLOGY	20
3.1	Waterfall	20
3.2	Agile	21
3.3	Summary	22
3.4	Chosen Methodology	23
4	REQUIREMENTS	24
5	ITERATION 1.....	25
5.1	Design & Implementation Overview	25
5.1.1	Automation Phase 1	25
5.1.2	Automation Phase 2	27
5.1.3	Configure Network	28
5.1.4	IPv4 Addressing with Avahi.....	31
5.1.5	Changing the Hostname	31
5.2	Evaluation.....	32
5.2.1	Test 1	32
5.2.2	Test 2	33
5.2.3	Test 3	33
5.2.4	Test 4	33
5.3	Discussion	34
5.3.1	Avahi	34
5.3.2	Interface Name Discovery.....	35
5.3.3	WiFi Country Code	35
5.3.4	Proposed Changes	35
6	ITERATION 2.....	36
6.1	Design & Implementation Overview	36
6.1.1	Creating Custom Log Files	37

6.1.2	Adding a File Check.....	38
6.1.3	Addressing.....	38
6.1.4	Adding the Gateway Functionality.....	43
6.2	Evaluation.....	44
6.2.1	Experiments.....	44
6.2.2	Complete System Check	45
7	CONCLUSIONS	46
7.1	Future work.....	46
7.1.1	Set Up an Automatic Virtual Private Network (VPN) Connection.....	46
7.1.2	Improve Wireless Security	47
7.1.3	Improve IPv4 Addressing.....	47
8	REFERENCES	48
9	APPENDIX A – Project Proposal	55
10	APPENDIX B – Risk Register.....	60
11	APPENDIX C – Project Requirements.....	61
12	APPENDIX D – Network Design.....	62
13	APPENDIX E – Experiment 1	63
14	APPENDIX F – Experiment 2	66
15	APPENDIX G – Experiment 3.....	68
16	APPENDIX H – User Guide.....	71
17	APPENDIX I – Ethics Checklist	76
18	APPENDIX J – USB and SD CARD Contents	78

LIST OF FIGURES

Figure 1 The OSI Model Showing Addressing & Encapsulation	4
Figure 2 Topology Comparison	5
Figure 3 A Typical 802.11 Network with Gateway	5
Figure 4 Multihop and Star Comparison	6
Figure 5 802.15.4 Star & P2P Topologies (IEEE 2015)	10
Figure 6 PAN Networks (IEEE 2015)	11
Figure 7 B.A.T.M.A.N Adv Layer Integration (Hundebøll and Ledet-Pedersen 2011)	12
Figure 8 Sun Founder Pi Car S (Sun Founder 2019)	14
Figure 9 The Raspberry Pi 3B (Raspberry Pi Foundation 2018).....	14
Figure 10 TIOBE Index (TIOBE 2019).....	16
Figure 11 Hello World Comparison	17
Figure 12 Project Management Success Rate Comparison (PMI 2019).....	20
Figure 13 2019 Project Management Methodology Statistics (PMI 2019).....	20
Figure 14 The Waterfall Methodology (Royce 1970)	21
Figure 15 Agile Values (Beck et al 2001)	21
Figure 16 The Agile Approach (Rasmusson 2019).....	22
Figure 17 Modified Waterfall	23
Figure 18 The MoSCoW Prioritisation Model (Agile Business 2019)	24
Figure 19 Iteration 1 Overall Process Flow Diagram	25
Figure 20 autop2p.service	26
Figure 21 Iteration 1 Script Process Flow Diagram	28
Figure 22 Example Python os Module Import and Command.....	28
Figure 23 ifconfig Output	29
Figure 24 Example Python commands Module Import & Command	29
Figure 25 Example Python re Module Import & Command	30
Figure 26 Wireless Interface Assessment	30
Figure 27 Example of Leveraging Error Messages.....	30
Figure 28 Example Python string.replace()	31
Figure 29 batctl Tests	33
Figure 30 Iteration 2 Overall Design.....	36
Figure 31 Iteration 2 Script Process Flow.....	37
Figure 32 SLAAC Global Addressing Format (Internet Society 2019)	38
Figure 33 SLAAC Link-Local Addressing Capture.....	39
Figure 34 Bournemouth University IPv4 Addressing	39
Figure 35 IPv4 Private Address Ranges (Rekhter et al 1996)	40
Figure 36 MAC Address and IPv4 Address Format	40
Figure 37 Binary, Hexadecimal & Denary Comparison.....	41
Figure 38 Hexadecimal to Denary Conversion	41
Figure 39 Addressing Process Flow.....	42
Figure 40 B.A.T.M.A.N Adv OGM Message Capture.....	44
Figure 41 Experiment 1 Map	64
Figure 42 Experiment 3 Network Settings	69
Figure 43 Fern WiFi Cracker	70
Figure 44 Raspbian Version.....	71
Figure 45 fdisk Output.....	72
Figure 46 File Location.....	73
Figure 47 Wireshark Popup	74

LIST OF TABLES

Table 1 Standards & Protocols Comparison.....	8
Table 2 Wireless Mesh Routing Protocol Comparison Table.....	12
Table 3 Service File Locations (Debian 2019c)	26
Table 4 File Permission.....	32
Table 5 Comparison of Number of Host Addresses	35
Table 6 Experiment Details	44
Table 7 Experiment 1 Loss	64
Table 8 Experiment 1 Latency.....	64

1 INTRODUCTION

1.1 Project Background & Context

In recent years wireless connectivity technologies in computer networks have evolved and diversified to adapt to the unique challenges faced in emerging sectors such as, distributed robotics and the Internet of Things (IoT).

Wireless communication allows the free movement of mobile devices, removing the need for cables, enabling networks to be deployed quicker and in a greater range of environments. These benefits can be further increased by removing the need for a central intermediate network device such as a wireless access point and utilising a Peer to Peer (P2P) network topology.

The Pi Car S is a robot car kits purpose is to provide a platform to facilitate Science, Technology, Engineering and Maths (STEM) learning (Sun Founder 2018). Bournemouth University Computing Department have a collection of these kits that are utilised for various robotics and coding-based projects.

To reduce the amount of time and effort expended in the initial network set up stage and remove the reliance on a central intermediate network device, an automated P2P networking solution would be beneficial.

1.2 Aims & Objectives

The aim of this project is to design and implement an automated P2P wireless networking solution that provides all the required functionality from the previous wireless access point-based solution, but without the need for any configuration.

The project objectives are:

- Investigate the specifications and limitations of the project hardware and software.
- Research the existing wireless technologies available.
- Analyse current automated configuration methods.
- Select the best suited technologies based on the background study findings.
- Design, implement and evaluate a custom solution.

All the objectives need to be accomplished within the time frames allocated in the project plan that is attached as part of appendix A.

1.3 Success Criteria

To be considered a success the project must accomplish the following:

- The project must include a preliminary research phase into the areas stated in the project objectives and the possible solutions identified must be evaluated and compared to assess their suitability to the project environment.
- The solution must meet the must have functional and non-functional project requirements that are attached to this report as appendix C.
- The solution must pass a series of evaluation tests to ensure it is a reliable, and capable of running the required Pi Car S software.

1.4 Risk Analysis

Prior to any detailed planning project risks were identified and documented in the risk matrix attached to this report as appendix B. The risk matrix contains an impact assessment and a risk response for each individual risk to provide situational awareness and to maximise the chances of the project being a success.

2 BACKGROUND STUDY

There are many ways to establish a wireless network, but to achieve a fitting solution for inter-robot connectivity extensive research was necessary. The Pi Car S contains sensors, actuators and control boards, built around a Raspberry Pi Model 3 B (Sun Founder 2019). Essentially, they are a combination of distributed robotics and a mobile wireless sensor network (WSN). Connectivity technologies in IoT and distributed robotics were the main areas of interest for connectivity research as these types of networks have certain unique characteristics.

To have an automated solution, research was carried out into how the operating system automates other tasks on start-up and potential scripting solutions. The functions and limitations of the Raspberry Pi also needed to be fully understood as well as the operation of the software that will be run over the network. The following chapter covers the findings from the research.

2.1 Networks Overview

The following section will provide insight into the networking standards, protocols and topologies suitable for use with this project. A P2P wireless solution is necessary to allow free movement of the robots. The wireless standards available to this project can be broken down into 3 main areas:

2.1.1 Wireless Local Area Network (WLAN) Standards

WLAN standards are intended to provide the functionality a local-area network (LAN) using Ethernet and as such high data rates are required (Carrol 2009).

2.1.2 Wireless Personal Area Network (WPAN) Standards

WPAN standards typically have lower data rates and effective range. Some can operate in dynamically formed mesh topologies and relay data to extend range (Carrol 2009).

2.1.3 Low Powered Wireless Local Area Network (LP-WLAN) Standards

LP-WLAN standards are intended for use in networks with limited power and relaxed throughput requirements. The main objectives of LR-WPAN standards are ease of installation, reliable data transfer, extremely low cost, and a reasonable battery life, while maintaining a simple and flexible protocol (IEEE 2011).

2.1.4 The OSI Model

The Open Systems Interconnection (OSI) model is a conceptual model that provides an abstract view of the processes involved in network communications. Data is encapsulated by the lower layers before being transmitted over the physical media (Dye et al 2008). For the purpose of this project the focus is mostly on the bottom 3 layers which are responsible for network communication. **It is important to understand the encapsulation mechanics and addressing required at layers 2 and 3.** Figure 1 below shows the layers, where the referenced addressing fits in and the encapsulation process.

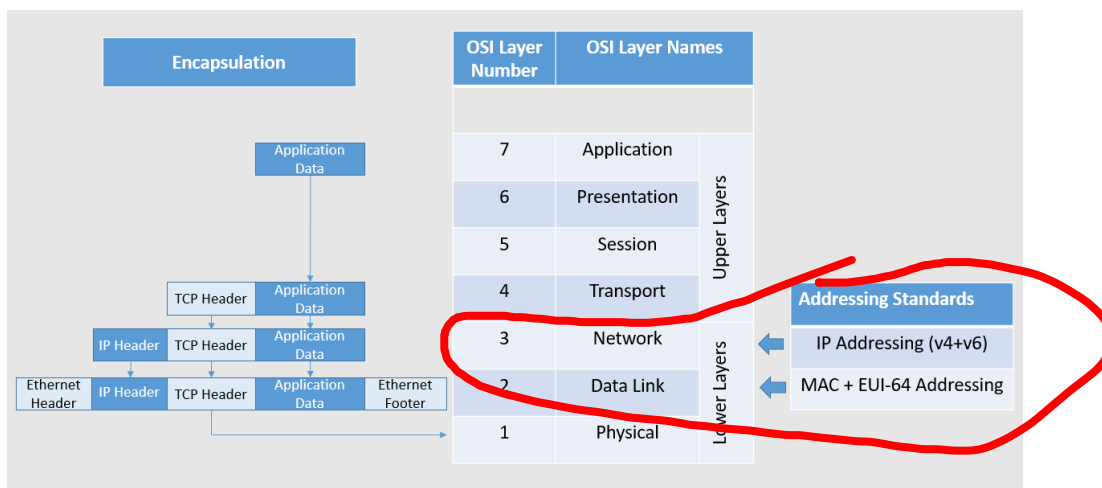


Figure 1 The OSI Model Showing Addressing & Encapsulation

2.1.5 Network Topologies

A network topology describes the arrangement and interlinking of devices within that network. A network in its simplest form consists of 2 devices communicating directly with one another over a medium. In the case of wireless communications this medium is the atmosphere. **There are 3 network topologies that are relevant to this project (fig 2).**

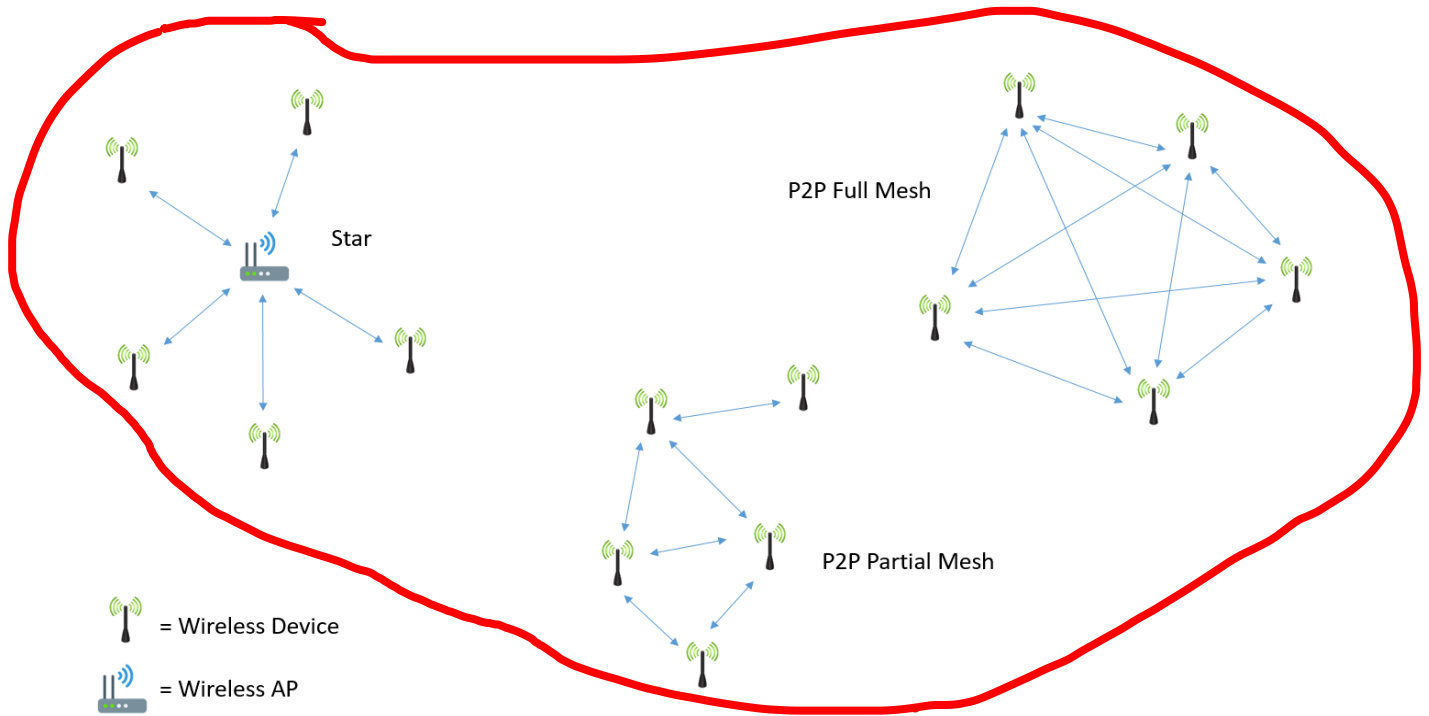


Figure 2 Topology Comparison

2.1.5.1 Star

A star topology is a network that has a central intermediate device. It is the most common WLAN topology as it is easy to install, scalable and easy to troubleshoot (Cisco Networking Academy 2014). A typical example of a star topology would be a home network as shown in figure 3. **At the centre of the network sits the multifunctional wireless router providing the functionality of a wireless access point (WAP), a modem, a switch and a router.**

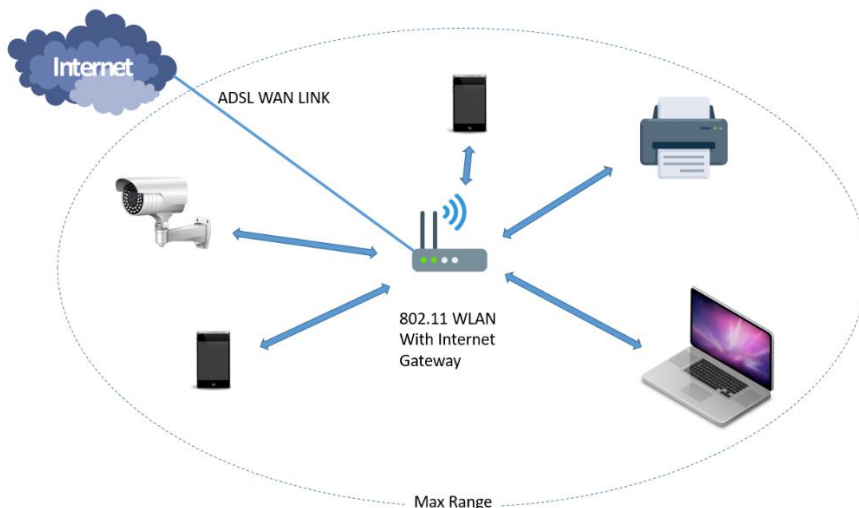


Figure 3 A Typical 802.11 Network with Gateway

The primary role of the device is to forward IP packets (layer 3) and frames (layer 2) to their destination. In this scenario the wireless router also provides a wide area network (WAN) gateway service, in this case to the Internet.

Typically, these devices are also responsible for automated addressing, most commonly using Dynamic Host Configuration Protocol (DHCP) with IPv4 addressing.

The transmission range of the wireless access point and potential RF interference limits the effective range of the connected stations. Even if two stations are within range of each other, if one is out of range to the WAP then they cannot communicate.

2.1.5.2 Peer to Peer (P2P)

A P2P network consists of 2 or more end devices connected to form a network. A P2P network differs from a Star network in that it does not require a central network device such as a WAP, this style of network is also known as ad-hoc (Dye et al 2008).

P2P connections can be quickly established anywhere between 2 compatible devices without the need for a WAP. Many of the protocols used for this type of communication however are based around file transfers or the streaming of media. Bluetooth Basic Rate (BR), Enhanced Data Rate (EDR) and WiFi Direct are all used in this way but are not suitable for this project as they do not allow connections between more than 2 devices (Bluetooth SIG 2019).

2.1.5.3 Mesh

A mesh topology consists of multiple P2P connections. They can be either full, where every station can communicate directly with one another, or partial where not all the stations can communicate directly. In this scenario an intermediate station may relay traffic, this is called a multi-hop scenario (fig 4). This is a critical enabling feature in dynamic distributed robotics and IoT networks to overcome range issues associated with using low power transmissions.

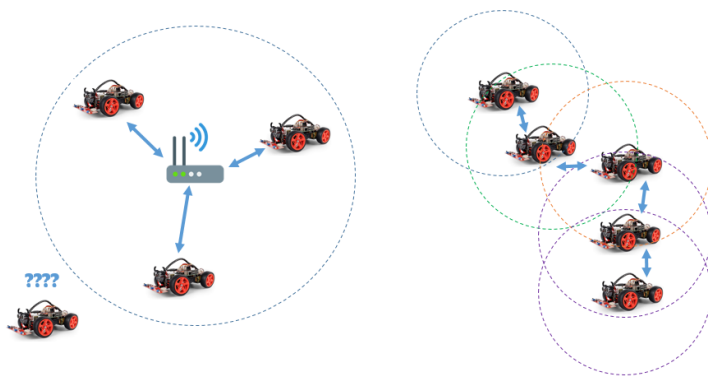


Figure 4 Multihop and Star Comparison

A network that has multiple routes to a destination is also more resilient as it provides redundancy should one link fail or become congested.

2.2 Connectivity Technology in IoT & Distributed Robotics

Robotics and IoT are two terms each covering a myriad of technologies and concepts. The IoT provides services for pervasive sensing, tracking and monitoring while robotics focuses on producing action, interaction and autonomous behaviour. Connectivity both within the separate communities and together is necessary to enable them to achieve their individual goals and to work in synergy. IoT networks feed sensor data into monitoring and analytical software or Artificial Intelligence (AI) platforms. This data can then be processed, and the output can provide informed, near real time instructions, relayed to situational aware robots (Simeons et al 2018).

The potential benefits are huge, connectivity is key. IoT nodes use both wired and wireless standards, connecting through mobile networks, over satellite links and established IP based network infrastructure. They make use of a diverse range of connectivity technologies including more recent IoT specific protocols and standards and other well-established wired and wireless protocols.

To enable remote connectivity each node requires unique addressing, IPv6 can provide this. The combination of these methods of connectivity and standards is seen as the enabling technologies that make IoT applications possible. The real-world benefits of the IoT have already been realised and a vast amount of research and development has taken place in industry and academia (Patel and Scholar 2016).

IoT deployment is increasing at an exponential rate with the number of deployed devices is doubling year on year, causing a greater demand for skilled staff and driving up financial overheads. A vast range of devices are now connected to converged global networks (Cisco 2016). Configuring networks is a time-consuming process requiring a level of expertise and travel to remote locations. An automatic configuration process would save time and money in the deployment phase of a network and the nodes could be deployed by virtually anyone. Further financial savings could be made by reducing the amount of skilled staff required. This is important in a project environment as a project may not be initiated if the return on investment is insufficient or the costs too high.

The “things” that form the IoT consist of varying hardware platforms running an array of both proprietary and non-proprietary operating systems and embedded software. These are often low powered, mobile devices powered by batteries over loss prone and sub optimal wireless networks. In these environments traditional 802.11 standards may not suffice. However, for many devices integrating into an existing WiFi or Ethernet network is the best solution as they are already established, and power is readily available. With such a diverse range of options and complex

technologies to connect it is vital to understand the characteristics and specifics of the standards and protocols.

2.3 Standards & Protocols

Research into wireless communication standards and protocols is initially focused around utilising the project hardware if possible and potentially utilising 802.15.4 standards with additional hardware if necessary. The findings of the research are summarised in table 1 detailing the characteristics and capabilities from the project perspective.

	802.11n	Bluetooth LE	Bluetooth BR/EDR	802.15.4
Max Data Rates	600 Mb/s	2 Mb/s	3 Mb/s	250 Kb/s
Max TX Power	+30dBm	+20dBm	+20dBm	+20dBm
Frequency Band	2.4 GHz	2.4GHz	2.4 GHz	868 - 868.6 MHz
UK	5 GHz			2.4 GHz
Classification	WLAN	WPAN	WPAN	LP-WLAN
Optimised For	High Data Rates	Short Burst TX (IoT)	Constant TX	Short Burst TX (IoT)
Supports Mesh	N	Y	N	Y
Supports Multi-hop	N	N	N	Y
Supported by BCM43438	Y	Y	Y	N
Supported by RA5370	Y	N	N	N
Requires Extra Hardware	N	N	N	Y

Table 1 Standards & Protocols Comparison

2.3.1 IEEE 802.11 Standard

Institute of Electrical and Electronics Engineers (IEEE) developed the 802.11 specification to define a standard means of passing data within a wireless local area network (WLAN) using radio frequencies (RF).

The transmissions, as with all the standards covered in this report are half-duplex, meaning stations need to take turns to transmit. As the medium (the atmosphere) is shared and contended there's potential in a multi stations scenario for simultaneous transmissions that cancel each other out. A solution to this problem was created called carrier-sensing multiple access collision avoidance (CSMA CA). This method of carrier sensing detects collisions and uses an algorithm to invoke random back off periods to ensure simultaneous transmissions can be avoided and overcome.

Authentication is carried out either with a pre-shared key or by using a certificate based method like Remote Authentication Dial in User Service (RADIUS) as described in RFC2865. RADIUS requires a centralised server, both the server and the client need to have a shared secret that

never gets passed over the network, this makes it more secure than a pre-shared key approach (Rigney et al 2000).

There are 3 pre-shared key methods of security, WEP, WPA and WPA2. Wired Equivalency Privacy (WEP) is now depreciated and has many weaknesses, its RC4 method of encryption is only 24 bits so keys get reused, once discovered the remaining traffic in a stream can be decrypted (Interopnet Labs 2008).

WiFi Protected Access (WPA) was developed to resolve the issues with WEP, it uses Temporal Key Integrity Protocol (TKIP) for encryption. TKIP uses the same RC4 cipher method as WEP and suffers the same weaknesses. WPA2 uses Advanced Encryption Standard (AES) encryption and was considered secure initially but has also been found to have many weaknesses. Vulnerabilities such as key reinstallation attacks (KRACKs) that manipulate the initial 4-way handshake used by WPA2 are widely documented all over the Internet and automated hacking tools freely available online (Vanhoef 2019; Kali Tutorials, 2014).

802.11n can use frequencies in both the 2.4GHz and 5GHz ranges. For the purpose of this project the UK 2.4 GHz range is of interest as this is the frequency range supported by the hardware. The 2.4 GHz range consists of 13 overlapping channels with 20 MHz bandwidth, the frequency range being 2.412 – 2.472 GHz (Ofcom 2019).

The effective range varies greatly dependant on many factors as with all RF communications. It is commonly stated by manufactures that a typical coverage of 100m+ can be achieved with the 802.11n standard. In real world studies in indoor environments coverage of 802.11 has been shown to be far less (Sandra et al. 2010).

Data rates for 802.11n are stated to be up to 600Mbps. This is achieved using higher bandwidth 40 MHz channels in the 5GHz range and a feature called Multiple in Multiple out (MiMo). MiMo uses multiple transmit and receive antennas to increase the number of transmit and receive streams and eliminate some causes of interference (IEEE 2009; Carrol, B 2009).

2.3.2 Bluetooth Standards

Bluetooth is classed as a WPAN standard. The IEEE 802.15.1 working group defined the standards for Bluetooth until 2005 and since then the Bluetooth Special Interest Group (SIG) has managed and developed the evolving standard. The wireless chipset on the Raspberry Pi supports Bluetooth Low Energy (LE), the Basic Rate (BR) and the Enhanced Data Rate (EDR) shown in figure 3. The LE standards low power consumption is intended for use with IoT applications

requiring low power consumption and relatively low bandwidth. BR and EDR have numerous abilities, from streaming audio to P2P file transfers. Maximum data rates are considerably lower than 802.11 as the intended applications are different.

Another pertinent feature is that all mentioned standard supports point to point and LE supports mesh networking but not multi-hop. A basic level of security can also be provided in the form of AES 128-bit encryption. Bluetooth overcomes RF interference issues using something called Adaptive Frequency Hopping (AFH) which changes the operating frequency during transmissions.

Bluetooth nodes are not IP addressable and as such would require a gateway to achieve external connectivity. (Cypress Semiconductor Corporation 2018; Bluetooth SIG 2019).

2.3.3 IEEE 802.15.4 Standard

The 802.15.4 LP-WLAN standard was designed to meet the new challenges of IoT networks. Globally unique addressing is provided at layer 2 by a 64-bit Extended Unique ID (EUI-64) as opposed to the 48-bit Ethernet Media Access Control (MAC) address (fig 1). It uses the same CSMA-CA collision avoidance method as the previously mentioned 802.11 standard.

Devices are broken down into 2 types. Either a Full-Function Device (FFD) that can act as a Personal Area Network (PAN) coordinator or a Reduced-Function Device (RFD). An RFD only cares about having a connection to an FFD and carrying out its own task, thus keeping its energy usage to a minimum (fig 5).

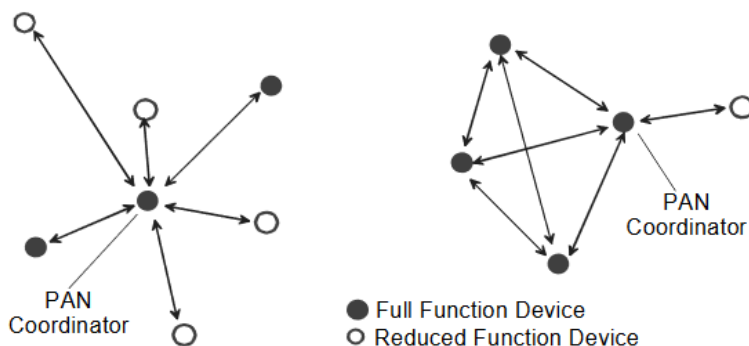


Figure 5 802.15.4 Star & P2P Topologies (IEEE 2015)

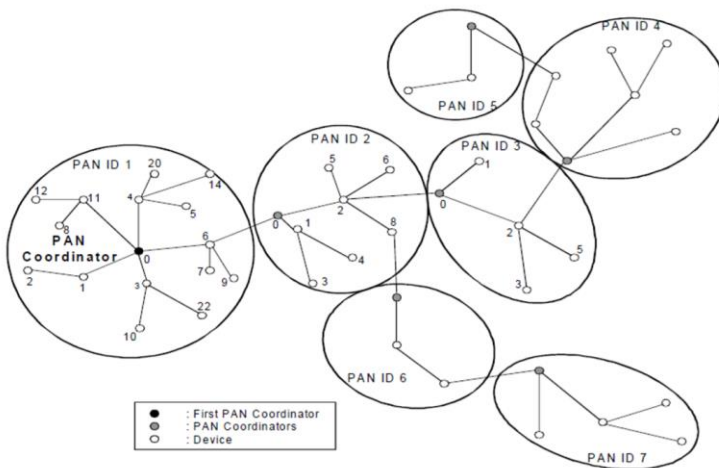


Figure 6 PAN Networks (IEEE 2015)

Each PAN has a unique identifier which allows communication both within that PAN and between devices on other PANs (fig 6). In the 2.4 GHz frequency range 16 channels of 5 MHz bandwidth are used. 250Kbps is the maximum physical data rate however the majority of this is used in overheads due to small physical packet sizes of 127 octets (IEEE 2015).

2.3.3.1 IEEE 802.15.4 Wireless Mesh Protocols

Several IoT specific protocols such as Thread, ZigBee and 6LowPAN have been created to operate over the on top of the 802.15.4 MAC and PHY layers. 6LowPAN is better suited to this project as it can interact well with other protocols.

IPv6 over Low-Power Wireless Personal Area Networks (6LowPAN) as the name suggests is LP-WPAN standard capable of utilising IPv6 addressing. The minimum IPv6 packet size is 1280 octets, to overcome this limitation a fragmentation and reassembly adaptation layer is provided at the layer below IP and header compression is used.

Either EUI-64 addressing can be used or a 16-bit short address further reducing overheads but further overheads. As mentioned, the overheads take up a considerable amount of space in the header, this limits how security within a 6LowPAN network can be achieved. (IEEE 2015; Montenegro et al 2017).

2.3.4 IEEE 802.11 Wireless Mesh Routing Protocols

The problem with classical routing protocols is that they are typically not well suited for wireless ad-hoc networks. This is because such networks are unstructured, dynamically change their topology, and are based on an inherently unreliable medium (Neumann et al, 2007).

Several wireless mesh specific protocols are available that operate over the 802.11 standard. The 2 deemed most suitable that work over both wired and wireless mediums and provide multihop mesh capabilities are, Babel and B.A.T.M.A.N Adv.

Table 2 details a summary of the findings from research into wireless mesh routing protocols.

	B.A.T.M.A.N Adv	Babel	ZigBee	6LowPAN
Compatible Standards	Bluetooth 802.11	802.11	802.15.4	802.15.4
IPv6 compatible	Y	Y	N	Y
OSI Layer	2	3	3+	3
Supports Mesh	Y	Y	Y	Y
Supports Multi-hop	Y	Y	Y	Y
Supported by BCM43438	N	N	N	N
Supported by RA5370	Y	Y	N	N
Requires Extra Hardware	N	N	Y	Y

Table 2 Wireless Mesh Routing Protocol Comparison Table

2.3.4.1 Babel

Babel is a layer 3 distance vector routing protocol; it works with wired and wireless mesh networks. As it uses periodic routing table updates rather than triggered updates it doesn't scale well in larger networks as it generates a lot of traffic overheads. This traffic would also impact on power consumption in a low powered IoT environment (Chroboczek 2011).

2.3.4.2 B.A.T.M.A.N Adv

The Open Mesh project manages the Better Approach to Mobile Ad-hoc Networking Advanced (B.A.T.M.A.N Adv) open source routing protocol. It's a multi-hop, ad-hoc mesh networking protocol that operates at layer 2 of the OSI model **and has been part of the Linux kernel since version 2.6.38.** It is integrated as a layer between the network interface drivers and a virtual interface created by the protocol which can be bridged with 802.11, Ethernet and Bluetooth physical interfaces (fig 7).

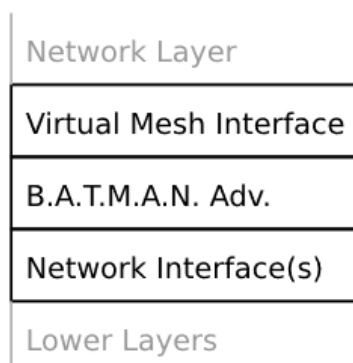


Figure 7 B.A.T.M.A.N Adv Layer Integration (Hundebøll and Ledet-Pedersen 2011)

B.A.T.M.A.N Adv is network layer agnostic allowing layer 3 protocols to be ran over the top if required, layer 3 addressing is not required for the nodes to communicate. Connected nodes are completely unaware of the B.A.T.M.A.N Adv protocol running below and can communicate as if connected to a big virtual switch.

Configuration and debugging can be achieved with the use of a command line tool called batctl.

This works in conjunction with B.A.T.M.A.N Adv providing tools for monitoring the state of the mesh and a command line interface to add or remove interfaces and alter parameters.

B.A.T.M.A.N Adv nodes use Originator Messages (OGMs) to inform neighbours about their existence, these are small 52-byte packets with the addition of IP and User Data Protocol (UDP) headers. OGMs contain sequence numbers to avoid loops and are also used to assess link quality. B.A.T.M.A.N Adv does not keep a map of the whole network, it only cares about the link quality to its direct neighbours.

There are issues surrounding the amount of network latency a general-purpose mesh protocol such as B.A.T.M.A.N Adv introduces in larger networks (500+), for the scope of this project however that is not of concern. **No** wireless security is built in but can be implemented at the higher layers if required. Virtual Local Area Networks (VLANs) are supported allowing traffic segregation at layer 2 (Lang 2019; Hundebøll and Ledet-Pedersen 2011).

2.4 Sun Founder Pi Car S Robots

The Pi Car S robot (fig 8) can be programmed with either Python or Dragit, a drag and drop Snap-based graphical interface. The kit contains 3 sensor modules, an ultrasonic sensor, a line following sensor and a light following sensor. The steering is achieved with a servo motor and two small electric motors drive the rear wheels.

There are 2 daughter boards that control the steering and sensor inputs that connect into a module that plugs directly onto the GPIO pins of a Raspberry Pi. A Ralink RT5370 802.11n wireless adapter is also provided (Sun Founder 2018).



Figure 8 Sun Founder Pi Car S (Sun Founder 2019)

2.4.1 Raspberry Pi

The Raspberry Pi 3 Model B (fig 9) is an extremely versatile credit card sized, single board computer (SBC). The Raspberry Pi project was started by a British computer scientist named Eben Upton and was marketed as an educational aid to learn computer science. The first Raspberry Pi was released in 2012, since then it has been used in a vast range of projects due to its low cost, portability, programmability and both wired and wireless connectivity. It runs Linux and other freely available operating systems from a micro SD card. With the addition of a keyboard, mouse and micro USB power supply it can be connected to a television or monitor and used as a fully functioning desktop computer (The Raspberry Pi Foundation 2015).

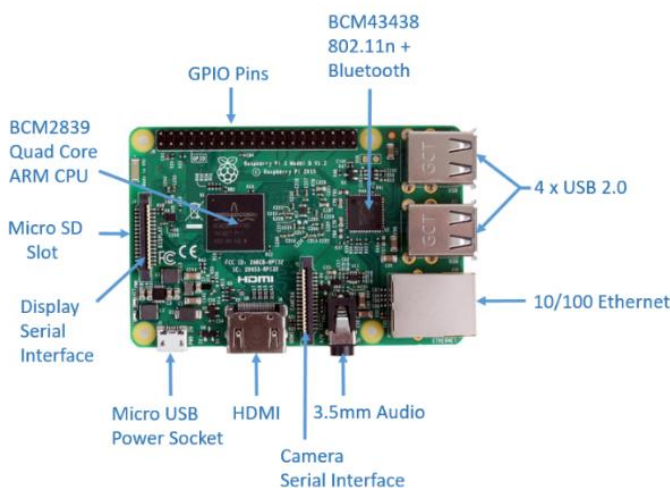


Figure 9 The Raspberry Pi 3B (Raspberry Pi Foundation 2018)

The Model 3 B features a quad core 1.2GHz Broadcom BCM2837 64bit CPU and 1GB of RAM. It also features an Ethernet port and a Broadcom BCM43438 wireless LAN and Bluetooth chip, enabling multiple methods of network connectivity. It has ample processing power and memory for it to run one of several operating systems available. (Raspberry Pi Foundation 2018).

2.4.1.1 Broadcom BCM43438

The BCM43438 chip can provide both single band 2.4GHz 802.11 b/g/n and Bluetooth. These standards share a single antenna which restricts its ability to utilise MiMo technology. The claimed data rate of the chip using the 802.11n standard is 200Mbps and the maximum channel bandwidth is 20 MHz. While this is deemed ample for this project it is significantly less than other devices using this standard, which claim up to 600Mbps. This is due in part to this chipset being unable to use the 5 GHz band that allows double the amount of bandwidth (40 MHz) and because of the lack of MiMo support. It does not allow for an increase in MTU size above the default 1500.

The BCM43438 can operate as a Bluetooth class 1 or class 2 transmitter, class 1 provides a theoretical range of up to 100M. It also supports Bluetooth 4.2 LE (Low energy) and is backwardly compatible with previous Bluetooth standards (Cypress Semiconductor Corporation 2018).

2.4.1.2 Ralink RT5370

The Ralink RT5370 is also a single band 2.4GHz 802.11 b/g/n device. Being single band and having only a single antenna it suffers from the same limitations as the BCM43438 in terms of data rates and channel bandwidth. The RT5370 does however allow an increased MTU size if required (MediaTek 2019).

2.4.2 Raspbian

The Raspberry Pi can run many different operating systems, they include: Windows 10 IoT core, Risc OS, Arch Linux, Ubuntu and many others. Raspbian, a Linux based operating system is the most popular and is also Sun Founders recommended choice for use with the Pi Car S robots used in this project. Raspbian is an unofficial port of Debian that is in continual development by a team of volunteers and is free to download and use.

Raspbian is necessary because the standard versions of Debian are incompatible with the Raspberry Pi ARM architecture. The Raspbian team aim to keep the distributions as close to Debian as possible and a high percentage of the packages available for Debian have been optimised for Raspbian. Debian has millions of users, excellent documentation and many online knowledge bases, as such it is easy to find solutions to a vast range of issues. As with all Linux distributions free, open source software packages can be sourced from online repositories.

Automated system tasks are accomplished within the operating system by using services. Raspbian services are managed and monitored by `systemd` and `systemctl`. Raspbian also comes with several programming languages preinstalled to enable further automation, including Python, C, C++ and Java (Raspbian 2018).

2.4.2.1 Programming Languages

Automating the network configuration requires some form of scripting that performs the tasks the end user would during the setup process. Commands are usually issued into the command line shell used by Raspbian, Bourne-again Shell (Bash). Bash is a programming language in its own right and compatible with most UNIX-like operating systems. Bash scripts are frequently used to automate system tasks and can be used in conjunction with other programming languages (Arch Linux 2019).

The use of another programming language would improve the portability of the code, enabling reuse on multiple platforms. Python, C, C++ and Java are all well-established and popular general-purpose programming languages capable of achieving the project goal.

Figure 10 shows the TIOBE community index ratings at the time of the project. It is based on the number of skilled engineers, courses, 3rd party vendors and search engine calculations. The top 4 places are taken by 4 of the pre-installed languages, with Java and C taking nearly a 30% share.

Apr 2019	Apr 2018	Change	Programming Language	Ratings	Change
1	1		Java	15.035%	-0.74%
2	2		C	14.076%	+0.49%
3	3		C++	8.838%	+1.62%
4	4		Python	8.166%	+2.36%

Figure 10 TIOBE Index (TIOBE 2019)

The popularity is not an indicator of performance or suitability for a specific task, it is something to consider regarding the future maintainability of the code. Ease of use and readability are another important consideration, below is a comparison of the code from the 4 most popular programming languages required to print “Hello World” to the console (fig 11).

Java

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

C++

```
#include <iostream>

int main()
{
    std::cout << "Hello World";
    return 0;
}
```

C

```
#include <stdio.h>

int main(void)
{
    printf("Hello World");
}
```

Python 2,7

```
print "Hello World"
```

Figure 11 Hello World Comparison

Studies carried out have found that beginner programmers found Python easier to understand and implement than C++ as it has a relatively simple format and syntax (Ateeq et al 2014; Foster 2014).

2.5 Automatic Configuration & Tools

Research into already existing solutions revealed 3 main approaches to automatic configuration.

- Cloud based IoT platforms such as IBM Watson, Cisco IoT Cloud Connect and the Microsoft Azure Portal.
- Agentless automation languages, such as Puppet and Ansible.
- Custom scripting.

2.5.1 Azure Portal IoT Hub

Microsoft's Azure portal is an example of a cloud-based subscription service and can provide a vast range of resources and services including virtual machines, cloud storage, and many IoT specific services. Of interest for this project is its cloud-based configuration tool called IoT Hub used in the automatic configuration, monitoring and managing of Windows 10 IoT devices.

It is an example of a Function as a Service (FaaS) platform that provides a centralised graphical user interface. One desirable feature it provides is individual device authentication and access control; this is particularly beneficial in IoT networks as remote, unattended nodes could be an easy way for an attacker to gain entry to a network.

For a business already taking advantage of other Azure features and IoT core devices, a cloud-based subscription service could be a good choice. The need for dedicated hardware, software, and future upgrades would be offloaded to the service provider. Cloud based services provide excellent scalability with the ability to quickly add or remove services and increase or decrease utilisation (Microsoft 2019).

2.5.2 Ansible

Ansible is a simple automation language used to create something called an Ansible playbook; it is also the underlying engine that runs the playbooks. Playbooks contain user created tasks that are human and machine readable, tasks contain modules, the tasks run sequentially.

Ansible can be used with Ansible Tower, a system used to manage and control Ansible automation. Network configuration is just one of a range of automated tasks, application deployment, configuration management, security compliance and many other tasks are possible.

Connections are made to hosts using SSH or PowerShell from a device running Ansible, called the control node. Ansible has modules for many different vendors including Juniper and Cisco. The Ansible components required to automate the configuration of a small Raspberry Pi network are freely available (Red Hat 2019).

2.5.3 Custom Scripting

For the configuration to be truly automatic, no external input should be necessary from the end user other than connecting the peripherals and powering on the robots. A script that runs on system start-up could carry out the automation of the network configuration, written in any of the languages pre-installed on the Raspberry Pi.

An initial configuration of the device is required however to install the required software and the network configuration script. This initial configuration can also be automated by using an install script. This approach would speed up deployment and if combined with a user guide enable someone with limited technical ability to configure the devices.

2.6 Summary

The project hardware and operating system provide a versatile platform enabling many ways to accomplish an automated network configuration with a range of viable wireless technologies available. While an 802.15.4 solution combined with 6LoWPAN has many benefits additional hardware is required adding another layer of complexity and cost. The low data rates also limit the scope of future projects using the Pi Car S.

The BCM43438 chip on board the Raspberry Pi cannot achieve the optimal multi-hop mesh topology when used with its native Bluetooth or 802.11 standards. This can however be accomplished by combining a wireless mesh protocol running over 802.11 hardware.

B.A.T.M.A.N Adv stands out from the crowd as the preferable wireless mesh protocol, effectively all nodes in a B.A.T.M.A.N Adv mesh network act as if connected to a big virtual switch. Being a network agnostic layer 2 protocol allows user applications and layer 3 networking configurations to run seamlessly over the top. WPA2 security is the current highest level of security using a pre-shared key, certificate based authentication methods (RADIUS) require a server.

B.A.T.M.A.N Adv adds an extra 32 bits to the standard 1500-bit Ethernet frame. To avoid fragmentation which impacts on network performance the RA5370 provided with the Pi Car S kit should be used over the on-board BCM42438, as the later doesn't allow an MTU increase. An added benefit of this to the end user is that the BCM42438 will be free for additional Bluetooth or 802.11 connectivity.

The automation of the network configuration will be achieved in 2 phases, the first phase being carried out by a service that will start on system start up. Services can be monitored, and logs are created by the system that aid troubleshooting. The chosen language for the automation of the network configuration is Python 2.7. It comes pre-installed on Raspbian, has the simplest syntax and an expansive library. Some of the code will also be portable for reuse on different operating systems.

3 METHODOLOGY

A 2019 survey carried out by the Project Management Institute (PMI) shows that employing a formal project management methodology maximises the chances of project success (fig 12).



Figure 12 Project Management Success Rate Comparison (PMI 2019)

The following sections explore the options considered when selecting an appropriate process model. This project which has a preliminary research and analysis phase but also contains some elements of software development.

3.1 Waterfall

The Waterfall methodology uses a linear sequential approach to project management. The stages of the project are carried out in order from project initiation to closure in a single pass; this is often referred to as a predictive methodology. Predictive methodologies still account for the largest proportion of projects overall (fig 13).

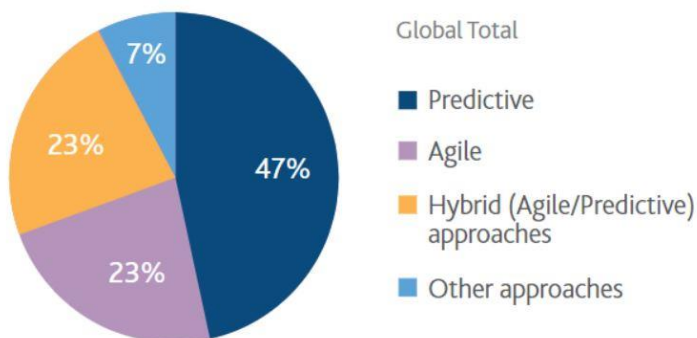


Figure 13 2019 Project Management Methodology Statistics (PMI 2019)

One of Waterfalls benefits is its simplicity (fig 14), only a basic knowledge of its structure and implementation are required. Predictive methodologies are well suited to non-software projects where material costs are high, and the chances of change are low.

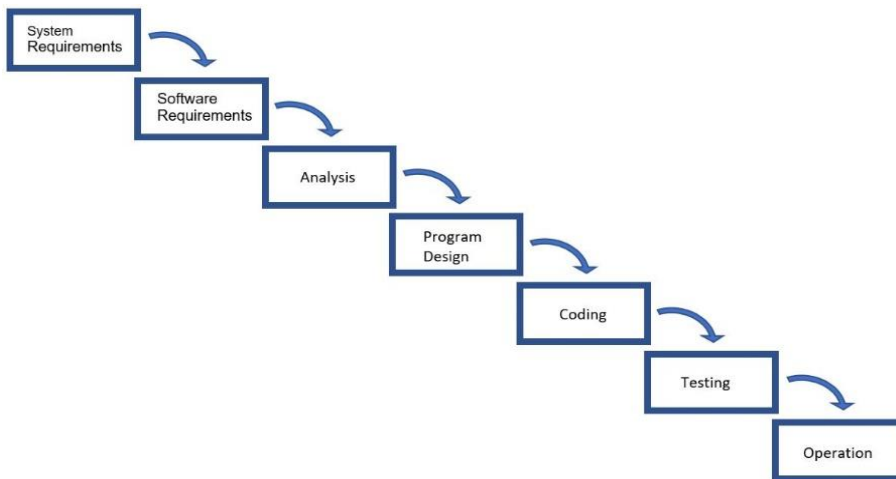


Figure 14 The Waterfall Methodology (Royce 1970)

No regular risk analysis is carried out with a basic Waterfall project and it is not well suited to high-risk projects where requirements may change. If time is short, the final testing phase may be cut short delivering low quality software.

3.2 Agile

Agile is an iterative approach to managing software development projects defined in the Agile Manifesto. The manifesto states that Agile is not a methodology but a set of values and principles that the experienced developer authors have found that if followed, lead to the successful delivery of better-quality software. The Agile beliefs are that while there is value in the items on the right in the following list, the items on the left are more important (Beck et al 2001).

- | | |
|--------------------------------|-------------------------------|
| • Individuals and interactions | • Processes and tools |
| • Working software | • Comprehensive documentation |
| • Customer collaboration | • Contract negotiation |
| • Responding to change | • Following a plan |

Figure 15 Agile Values (Beck et al 2001)

In addition to the values, there are 12 principles that support the values. The principles are very general and are not intended to provide a ridged framework but rather intended to provide guidance when making project decisions.

Figure 16 illustrates Agile's iterative approach in comparison to traditional single pass approaches such as Waterfall. Breaking down the development into smaller chunks allows problems and risks to be identified earlier and allows a team to be more adaptable to change.

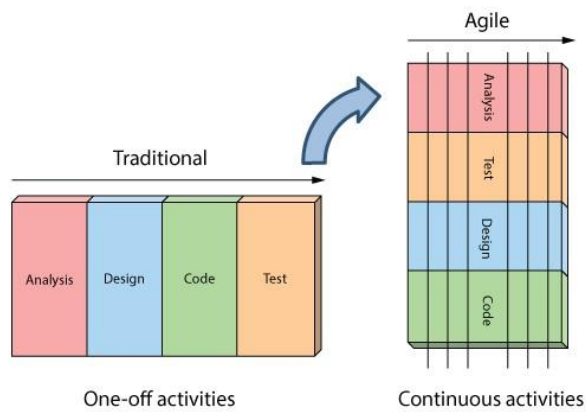


Figure 16 The Agile Approach (Rasmusson 2019)

There are several different implementations of Agile, one of the most popular being Scrum.

3.2.1.1 Scrum

Scrum is a team focused and is well suited to software development projects as it inherits the Agile beliefs, values and iterative approach. Scrum has its own lightweight documentation structure and teams have the following predefined roles and responsibilities:

- Scrum Master – Protects the team from distractions and helps them perform to their highest standards.
- Project Owner – Maintains something called the continually evolving project backlog containing the project requirements. Aims to satisfy all stakeholders and ensures that everyone knows the priorities.
- The Development Team – Empowered to organise and manage their own work, to increase efficiency and effectiveness.

There are short daily team meetings every day to assess progress toward the Sprint Goal. Sprints are normally no longer than 2 weeks (Scrum Alliance 2019).

3.3 Summary

There are benefits to each of the methodologies, while Agile based approaches such as Scrum are clearly better suited to a project containing some software development, they are very team based.

Working as an individual removes the need for regular team meetings and role allocation but increases the overall level of responsibility as the risks and deadlines still exist. The simplistic Waterfall approach could provide the structure required but lacks flexibility and regular risk assessment required for risky software-based projects.

3.4 Chosen Methodology

The chosen methodology is a hybrid of an initial predictive Waterfall approach followed by an iterative Agile approach to the development phase consisting of frequent yet small modular releases.

The initial part of the project has set deadlines and deliverables, a project proposal containing a detailed project plan is attached to this report as appendix A. The plan provides set time frames to enable planning and monitor progress. A risk register is also attached as appendix B, this document promotes situational awareness and contains details of how the known project risks are managed with an appropriate risk response. Frequent risks assessment is conducted through-out the project in a less formal way using a project white board.

The design and implementation phase consist of 2 iterations with modular testing and end of iteration tests at the completion of each cycle (fig 17).

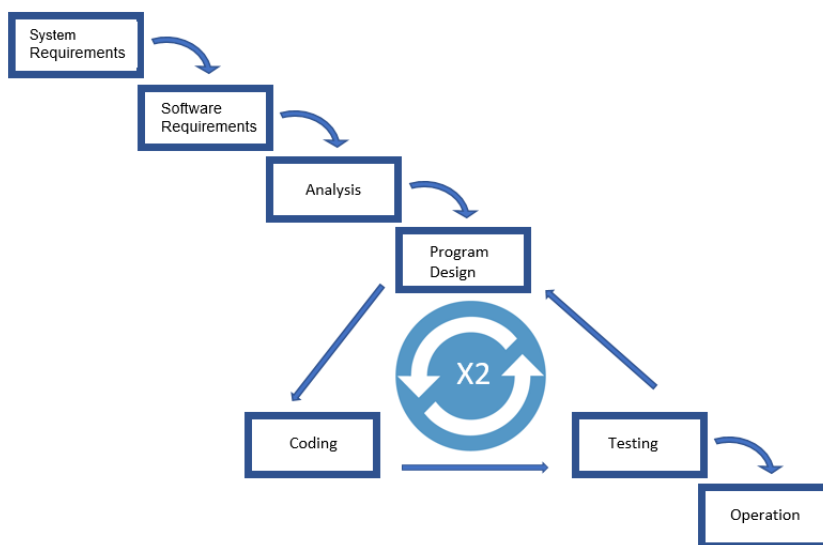


Figure 17 Modified Waterfall

4 REQUIREMENTS

The project requirements were established at the beginning of the project and assessed using the MoSCoW methodology to balance work prioritisation and avoid scope creep (fig 18).

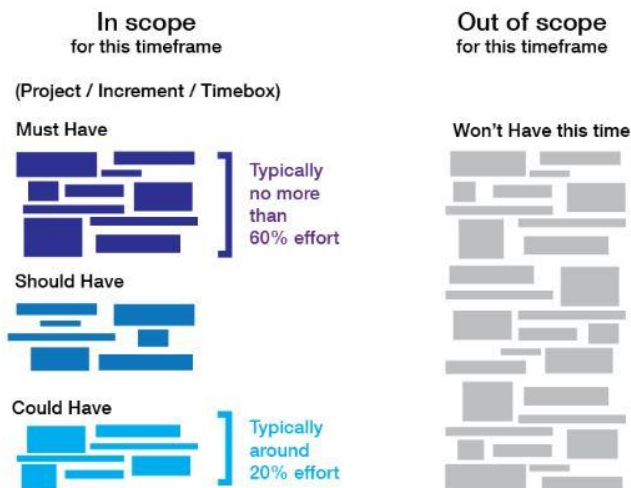


Figure 18 The MoSCoW Prioritisation Model (Agile Business 2019)

They consist of functional and non-functional requirements and are attached to this report as appendix C.

5 ITERATION 1

5.1 Design & Implementation Overview

This chapter describes the first of 2 iterations of design and implementation. Design and implementation were approached in an iterative manner, while the majority of the issues encountered were resolved before the end of the iteration some required further work in iteration 2.

The initially chosen approach for addressing proved to be unsuitable as it lacked a means of IP discovery and produced unnecessary network traffic. Some further refinements to the network configuration method were also identified in regards to interface discovery. WEP security was implemented instead of WPA2 due to implementation issues.

The issues and the chosen approach to rectify them are detailed in the discussion section (5.3) along with a brief discussion of the other challenges resolved during the iteration.

The initial designs aim is to achieve the project requirements using the best suited technologies discovered during the background research phase. The network design diagram is attached to this report as appendix D. The functions that need to be carried out at each stage were identified and the processes visualised in the process flow diagram below (fig 19).

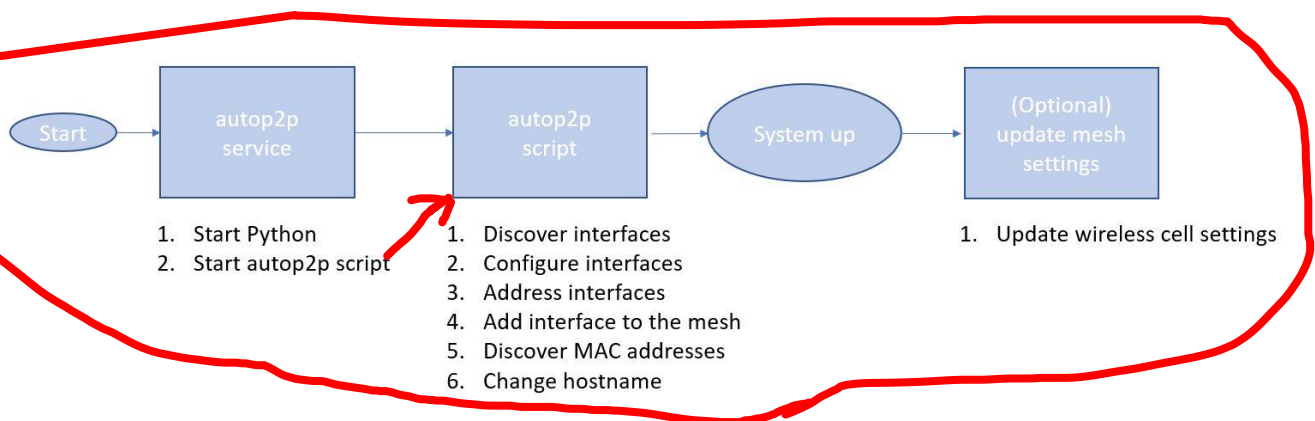


Figure 19 Iteration 1 Overall Process Flow Diagram

The design consists of 3 main components:

- A new service (automation phase 1).
- A script to carry out the automated network configuration (automation phase 2).
- A means of updating the network settings.

5.1.1 Automation Phase 1

Creating a new service is the first of 2 phases required to accomplish an automated network configuration. A service is a process that runs either on system start-up or on demand. Services are the operating systems way to automate the tasks necessary to get a computer to an

acceptable start state for a user. A service may perform its task and then stop or run constantly in memory listening out for requests. The latter is normally referred to as a daemon (Debian 2017b).

`systemd` is the system and service manager used by Raspbian and many other recent versions of Linux. `systemd` is controlled and monitored using a command line tool called `systemctl`. The services files are written in plain text and appended with the `.service` file extension. The new service file is classed as a local configuration and its recommended location is in the `/etc/systemd/system/` directory path. All service file locations are listed below in table 3.

Directory Path	Contents
<code>/lib/systemd/system/</code>	Units of installed packages
<code>/run/systemd/system/</code>	Runtime units
<code>/etc/systemd/system/</code>	Local configuration

Table 3 Service File Locations (Debian 2019c)

`systemd` not only manages the services but also creates a journal of events and failures. These can be viewed to enable troubleshooting or to confirm everything is working as it should using `systemctl` commands and in further detail with another command line tool called `journalctl` (Debian 2019c).

`systemd` tasks are organised as units, services are a type of unit. Targets are groups of units; the units may be of different types that are grouped together to accomplish a mutual goal.

Each service is part of a dedicated control group (`cgroup`) with the same name as the service (Debian 2019c).

A service file consists of 3 sections as shown in figure 20, `[Unit]`, `[Service]` and `[Install]`.

```
[Unit]
Description=Auto P2P
Before=network-online.target

[Service]
Type=simple
ExecStart=/usr/bin/python /home/pi/autop2p.py
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

Figure 20 autop2p.service

The `[Unit]` section needs a description that is relevant and unambiguous as this is the description shown in the system logs, in this case Auto P2P. The order in which the services are started is very important as some tasks are dependent on other tasks being accomplished before they can work, this is called a dependency. An example of this is that the network interfaces need to be discovered by the system before they can be configured. The order of execution is controlled using `Before` or `After` instructions in the `[Unit]` section in conjunction with a unit or target name (Debian 2019e).

The second section of a service file is the `[Service]` section. Several different types of service exist, if none is stated then it is assumed it is a `simple` service. For `simple` services it is expected that the main process of the service is configured with `ExecStart`. This is where the path to executable resources is entered. This is the link to the second phase of automation, for the project first an instance of Python is started and then the `autop2p` script (Debian 2019d).

The `[Install]` section carries installation information for the unit, this is not interpreted by `systemd` during run time but when the service is enabled or disabled using the `systemctl` tool. Symbolic links are created at that point in `.wants/` or `.requires/` subdirectories of the specific unit. These instructions dictate when the service starts in relation to system run levels. Essentially by specifying `multi-user.target` in the `[Install]` section it is telling the operating system to start the service at system boot (Debian 2019c; Debian 2019d).

5.1.2 Automation Phase 2

The second phase of automation is carried by an executable script written in Python 2.7.

The Python script is responsible for all the initial network configuration and the required subtasks.

The process flow diagram in figure 21 shows the overall structure and the order of execution of the initial script. From this the programming constructs, variables, inputs and outputs could be identified.

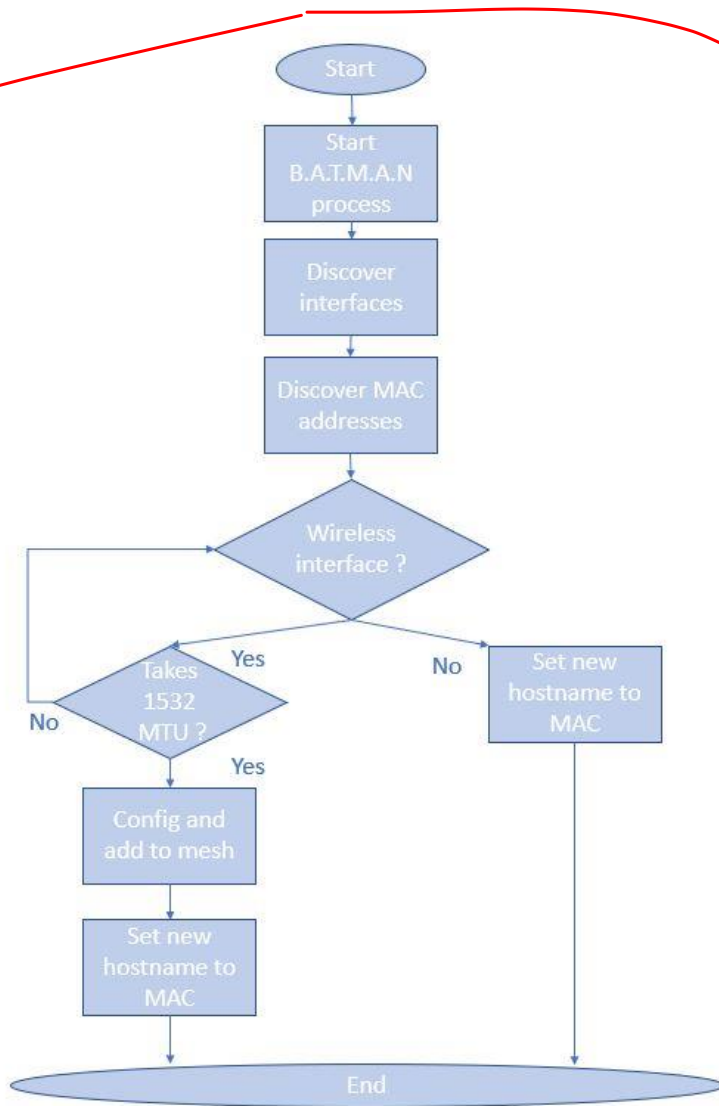


Figure 21 Iteration 1 Script Process Flow Diagram

Using the online documentation from the Python Software Foundation (2019) and many other online resources the individual functions to achieve the requirements were created. The script was written and tested in an iterative and modular way, and upload frequently to the online Version Control System and repository, GitHub. This provided a centralised storage solution and easy access to previous versions following unsuccessful changes (GitHub 2019).

5.1.3 Configure Network

Configuring the network and other system settings manually requires commands to be entered into the terminal. This can be carried out in Python by importing the `os` module (Python 2019c) and issuing an associated command in conjunction with the required system command, as shown in figure 22.

```
import os

os.system("echo Hello World")
```

Figure 22 Example Python `os` Module Import and Command

Many of the configuration commands require root privileges, this can be achieved using the preinstalled `sudo` package by prefixing a command with `sudo` (Arch Linux 2019). To carry out interface configuration the interface name needs to be discovered and assigned to a variable. Interface names can be discovered with the `ifconfig` command. This output also contains a lot of other unwanted information as shown in figure 23.

```
pi@000f008elb90:~ $ ifconfig
bat0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.100.1.229  netmask 255.255.255.0  broadcast 10.100.1.255
    inet6 fe80::e049:1bff:fe4b:35e5  prefixlen 64  scopeid 0x20<link>
    ether e2:49:1b:4b:35:e5  txqueuelen 1000  (Ethernet)
    RX packets 6399  bytes 316230 (308.8 KiB)
    RX errors 0  dropped 2  overruns 0  frame 0
    TX packets 126  bytes 19824 (19.3 KiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 23 `ifconfig` Output

By discovering the format of the output from a command, a unique identifier from the line containing the information required can be used by `grep`, a utility used to search for specified patterns in text. When a pattern is matched `grep` returns the full line of text. This can then be processed by `awk`, which sees the line as fields separated by white space. By specifying the field position that field with the can then be printed (Debian 2017a; Debian 2019a). The Python `commands` module has a `.getoutput` function that takes a system command containing `grep` and `awk` and returns the output generated, which can then be assigned to a variable (Python 2019b).

This is demonstrated in figure 24 in which `grep` searches for the “BROADCAST” pattern, `awk` is then used to print out the first word of the returned line, which in this case would be the interface name `bat0`. This is then assigned to the variable `interface_name` by `commands.getoutput`.

```
import commands

interface_name = commands.getoutput('sudo ifconfig bat0 | grep BROADCAST | awk "{ print $1 }"')
```

Figure 24 Example Python `commands` Module Import & Command

This way of discovering information can be problematic and result in some oddly named variables taken from unintended output. This is handled by checking the variables contents with regular expressions using the `re` module (Python 2019e). A regular expression specifies the format of a string. This can be used in conjunction with `if` statements to carry out a task if a specified regular expression is. Figure 25 is a regular expression checking for a MAC address format.

```
import re

if re.match('^[a-zA-F0-9]{2}[:?]{5}[a-zA-F0-9]{2}$'):
```

Figure 25 Example Python `re` Module Import & Command

Initially, to identify if an interface was wireless or Ethernet the string that was extracted using the `grep` and `awk` functions and tested against `if` statements matching the Raspbian interface naming convention, as demonstrated in figure 26.

```
test_int = 'wlan0'

if 'wlan' in test_int:
    print "This is a wireless interface"
else:
    print "This is not a wireless interface"
```

Figure 26 Wireless Interface Assessment

Error messages are printed to the terminal when carrying out manual configuration. The `os` module can be used to assess if an error has occurred during the script (Python 2019c). If an `os` command is successful it outputs 0. This is utilised to assess if the wireless interfaces will accept an increase in MTU as per the project requirements. In the example in figure 27 the `os` module is imported, the output from a command to increase the MTU is assigned to the `error_message` variable and `if` statements used to assess if the variable is equal to 0 or not.

```
import os

error_message = os.system('sudo ifconfig wlan0 mtu 1532')
if error_message == 0:
    print "Continue with confirmation"
else:
    print "This is not a wireless interface"
```

Figure 27 Example of Leveraging Error Messages

The `ifconfig` command is used to configure general interface properties, e.g:

Increase the MTU.

Bring the interfaces up and down.

Discover interface names and addressing.

The `iwconfig` command is used to configure wireless interfaces properties, e.g:

The interface mode (ad-hoc).

The essid.

The cell id.

The wireless channel.

Set the transmission power.

Apply WEP security.

The `batctl` tool is used to bridge physical interfaces with the `bat0` virtual interface.

5.1.4 IPv4 Addressing with Avahi

There is a chicken and egg issue surrounding addressing in P2P networks. The issue is that to join a network you need an address, but how do you get an address without being connected to a network?

While it would be possible to configure one of the robots as a DHCP server this introduces a single point of failure on the network and the network would no longer be truly P2P. If the robot providing the DHCP addressing were out of range or not functioning, no additional robots would be able to get an address and join the network. IPv4 Link Local addressing as described in RFC 3927 details how a network hosts can automatically configure its own address within the `169.254.0.0 /16` address range (Cheshire et al 2005).

A package called Avahi-autoipd was downloaded and installed to provide the link-local addressing. It is primarily intended to be used to provide automatic addressing in ad-hoc networks which lack a DHCP server (Debian 2015). This was implemented on the `bat0` virtual interface created by B.A.T.M.A.N Adv.

5.1.5 Changing the Hostname

All the hostnames on default Raspbian images are set to `raspberrypi`. The reason for changing this is to provide a uniquely identifiable, unambiguous hostname for each robot on the network. The hostname, where possible, is changed to the MAC address of the wireless interface that connects to the mesh network to simplify identification. If no suitable wireless interfaces are detected, then the hostname is set to the Ethernet ports MAC address. These are discovered using the previously mentioned combination of `grep`, `awk` and `re`. Colons are not permitted in a valid hostname, so these are stripped out using the Python `string.replace()` method like the example in figure 28 (Python 2019f).

```
mac_addr = '11:22:33:44:55:66'
mac_addr = mac_addr.replace(':', '')
```

Figure 28 Example Python `string.replace()`

To change the hostname 2 files, need to be updated:

```
/etc/hosts
/etc/hostname
```

A `hostname` command followed by the new hostname is issued and the networking service restarted, removing the need to reboot (Tobias et al 2009).

The file permissions need to be set to enable them to be written too. This was done using the `chmod` command followed by a 3-digit number. The number required can be worked out using table 4 below. Each number represents either the owner, group or user in that order. The individual numbers are worked out by adding together each permission the individual group or user has.

Owner		Group		User	
Read	4	Read	4	Read	4
Write	2	Write	2	Write	2
Execute	1	Execute	1	Execute	1

相加求和为权限

Table 4 File Permission

For the host files it is necessary to give all 3 write permission using:

```
sudo chmod 666
```

The files are then in turn read into memory using the `.read` function where the hostnames were replaced and then the new data saved over the original data.

5.2 Evaluation

To evaluate the success of the newly created script and service the following test were carried out throughout the implementation phase and at the end. For the tests static IP addresses were used. Throughout the project a network protocol analyser called Wireshark was used to visualise and inspect the network traffic (Wireshark 2019).

使用wireshark来进行网络流量测试，对配置进行评估

5.2.1 Test 1

Checking the function of the service with `systemctl` and `journalctl`:

```
sudo systemctl status autop2p.service
```

```
sudo journalctl -u autop2p.service
```

检查功能文件状态

The results from the end of iteration tests were error free.

没有错误

5.2.2 Test 2

Checking the B.A.T.M.A.N Adv Neighbour and Originator tables using the `batctl` tool.

```
sudo batctl n
```

检查batman的邻居表

```
sudo batctl o
```

Both tables showed the neighbouring nodes proving they had all joined the mesh.

5.2.3 Test 3

Checking basic layer 2 connectivity, latency and loss with `batctl` layer 2 pings.

```
sudo batctl ping {bathost name | MAC address}
```

Ping测试来测试丢包

Connectivity was good between all nodes with no packet loss.

5.2.4 Test 4

Basic layer 3 connectivity, latency and loss with standard pings.

```
ping {IPv4 address/mask}
```

Pings were successful between all nodes with no packet loss and low latency.

```

pi@000f008e1b90:~$ sudo batctl o
[B.A.T.M.A.N. adv 2017.3, MainIF/MAC: wlan1/00:0f:00:8e:1b:90 (bat0/1e:65:bb:f0:a5:e7 BATMAN_IV)]
  Originator      last-seen (#/255)  Nexthop      [outgoingIF]
  00:0f:00:8a:c4:67 0.150s (172) 00:0f:00:79:cf:26 [wlan1]
  00:0f:00:8a:c4:67 0.150s (174) 00:0f:00:a3:c2:b8 [wlan1]
  00:0f:00:8a:c4:67 0.150s (232) 00:0f:00:8a:c4:67 [wlan1]
  00:0f:00:a3:c2:b8 0.240s (178) 00:0f:00:79:cf:26 [wlan1]
  00:0f:00:a3:c2:b8 0.240s (178) 00:0f:00:8a:c4:67 [wlan1]
  00:0f:00:a3:c2:b8 0.240s (237) 00:0f:00:a3:c2:b8 [wlan1]
  00:0f:00:79:cf:26 0.580s (172) 00:0f:00:a3:c2:b8 [wlan1]
  00:0f:00:79:cf:26 0.580s (175) 00:0f:00:8a:c4:67 [wlan1]
  00:0f:00:79:cf:26 0.580s (232) 00:0f:00:79:cf:26 [wlan1]

pi@000f008e1b90:~$ sudo batctl n
[B.A.T.M.A.N. adv 2017.3, MainIF/MAC: wlan1/00:0f:00:8e:1b:90 (bat0/1e:65:bb:f0:a5:e7 BATMAN_IV)]
  IF      Neighbor      last-seen
  wlan1  00:0f:00:79:cf:26 0.810s
  wlan1  00:0f:00:8a:c4:67 0.140s
  wlan1  00:0f:00:a3:c2:b8 0.240s

pi@000f008e1b90:~$ sudo batctl if
wlan1: active

pi@000f008e1b90:~$ sudo batctl tg
[B.A.T.M.A.N. adv 2017.3, MainIF/MAC: wlan1/00:0f:00:8e:1b:90 (bat0/1e:65:bb:f0:a5:e7 BATMAN_IV)]
  Client      VID Flags Last ttvn      Via      ttvn (CRC)
  * 56:f8:b2:2c:02:ea -1 [...] ( 1) 00:0f:00:a3:c2:b8 ( 1) (0x94b50289)
  * c6:33:66:8b:7a:2c -1 [...] ( 1) 00:0f:00:8a:c4:67 ( 1) (0x2ea4a9cd)
  * e6:07:78:9f:a7:71 -1 [...] ( 1) 00:0f:00:79:cf:26 ( 1) (0xffa6ad22)

pi@000f008e1b90:~$ sudo batctl ping 56:f8:b2:2c:02:ea
PING 56:f8:b2:2c:02:ea (00:0f:00:a3:c2:b8) 20(48) bytes of data
20 bytes from 56:f8:b2:2c:02:ea icmp_seq=1 ttl=50 time=8.54 ms
20 bytes from 56:f8:b2:2c:02:ea icmp_seq=2 ttl=50 time=1.37 ms
20 bytes from 56:f8:b2:2c:02:ea icmp_seq=3 ttl=50 time=13.50 ms
20 bytes from 56:f8:b2:2c:02:ea icmp_seq=4 ttl=50 time=17.95 ms

```

Originator table

Neighbor table

Bridged with bat0

Bat hostnames

Layer 2 Pings

Figure 29 `batctl` Tests

The output in figure 29 shows the results from a selection of `batctl` commands and how B.A.T.M.A.N Adv stores multiple routes to other nodes in the Originator table and allocates costs out of 255 to evaluate the best path to a node. All 4 nodes were within range of each other when these tests were carried out so that is why there are 3 routes to each node in the Originator table. The highest value denotes the best route, these are also in this case the direct paths and have a star prefix.

The Neighbour table is far simpler showing the neighbours RT5370 MAC address, which interface it is connected to and the last-seen value which is based on the last time an OGM “hello” message was received from that node, by default they are sent once a second.

Wireshark was also used to examine the network traffic, identify any issues and assess if any changes are required. The captures showed a lot of unnecessary address resolution protocol (ARP) broadcasts from link-local addresses allocated from Avahi. These far exceeded the OGM messages and represented 97% of all traffic on the unutilised network. Avahi automatically allocates addresses if no other means of addressing has been implemented and sees the physical interfaces bridged with the `bat0` virtual interfaces as needing addressing. This in turn generates traffic in the form of link-local ARP broadcast and service discovery messages.

5.3 Discussion

5.3.1 Avahi

Avahi-autoipd implementation was problematic. An Avahi daemon is already installed on the stock Raspbian image which wasn't identified in the background study. The Avahi-autoipd package just adds extra functionality. The main problem was that Avahi runs as a daemon with its own non-root user account. When issuing commands within the script the root privileges required for all the subsequent non-Avahi configuration were dropped, causing them to fail which caused confusion and cast doubt on the rest of the code until error messages regarding privileges in the `systemctl` log which were linked to Avahi. The system logs were invaluable during the implementation phase.

During implementation Wireshark captures showed a lot of multicast traffic is generated by Avahi even when on an isolated network. When a robot was connected wirelessly to the mesh and to a home network by Ethernet, link-local traffic from devices on the home network was passed onto the mesh as all link-local devices use the same subnet. While these small protocol overheads are perfectly acceptable in a high bandwidth, mains powered home network this could cause issues in a low power sensor network as each transmission will consume a small amount of power.

Discovering neighbouring robots IP addresses was also not a simple process, the default addressing uses a /16 subnet mask. Table 5 shows how the IPv4 address space is split down using a subnet mask. The zeros in the binary notation field denote which bits are available for host addressing, indicating that there are 65534 potential host IP addresses for a /16 address. This made any Internet Control Message Protocol (ICMP) based discovery, such as a ping sweep impractical.

Subnet Mask	CIDR	Binary Notation	Number of Host Addresses
255.255.0.0	/16	11111111.11111111.00000000.00000000	65534
255.255.255.0	/24	11111111.11111111.11111111.00000000	254

Table 5 Comparison of Number of Host Addresses

Options were found to set a smaller address scope for Avahi and also for it not to drop root privileges, but it was decided that another means of addressing was required due to a combination of the stated issues.

5.3.2 Interface Name Discovery

Initially the script used hardcoded interfaces names and it was assumed that the interfaces would always take the same names, they didn't. To address this lack of flexibility a more robust approach involving an initial interface name discovery function is required.

5.3.3 WiFi Country Code

When connecting the newly built Raspberry Pi to a monitor it was discovered that they were unable to connect to a standard 802.11 wireless network using the network manager. Because the country code had not been set then the wireless was disabled. This was due to the initial GUI start-up wizard not being completed as the Raspberry Pi's were configured via Secure Shell (SSH).

5.3.4 Proposed Changes

The following additions will be added to iteration 2:

- A custom addressing method and the complete removal of Avahi.
- An interface discovery function and a dynamically formed list containing the names.
- A function to check the WiFi country code and if necessary, set to GB.
- An automatic configuration of the B.A.T.M.A.N Adv gateway function.
- An IP discovery function.

6 ITERATION 2

6.1 Design & Implementation Overview

This chapter covers the second and final iteration of design and implementation. Iteration 2 consists of the proposed changes following iteration 1 and the pre-scheduled tasks from the project requirements list. Following the evaluation at the end of iteration 1 the overall design process flow diagram was updated to include the new addressing function following the system initiation, changes are shown in green (fig 30).

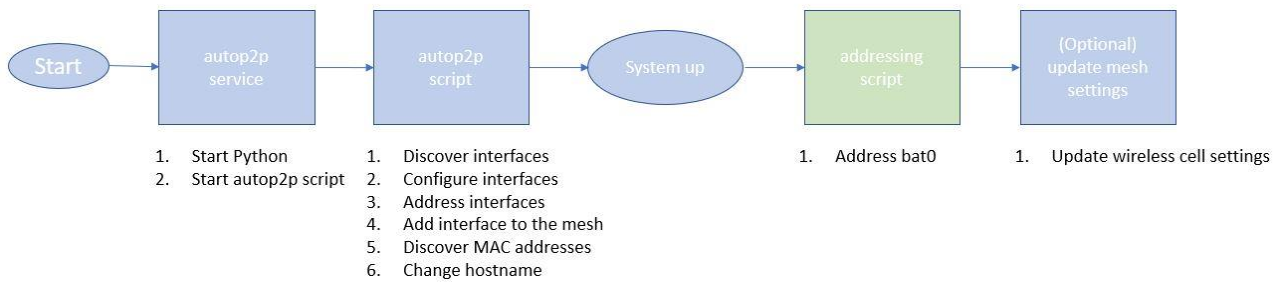


Figure 30 Iteration 2 Overall Design

The other proposed changes occur within the `autop2p.py` script. The script process flow diagram was updated to include the additions, shown in figure 31.

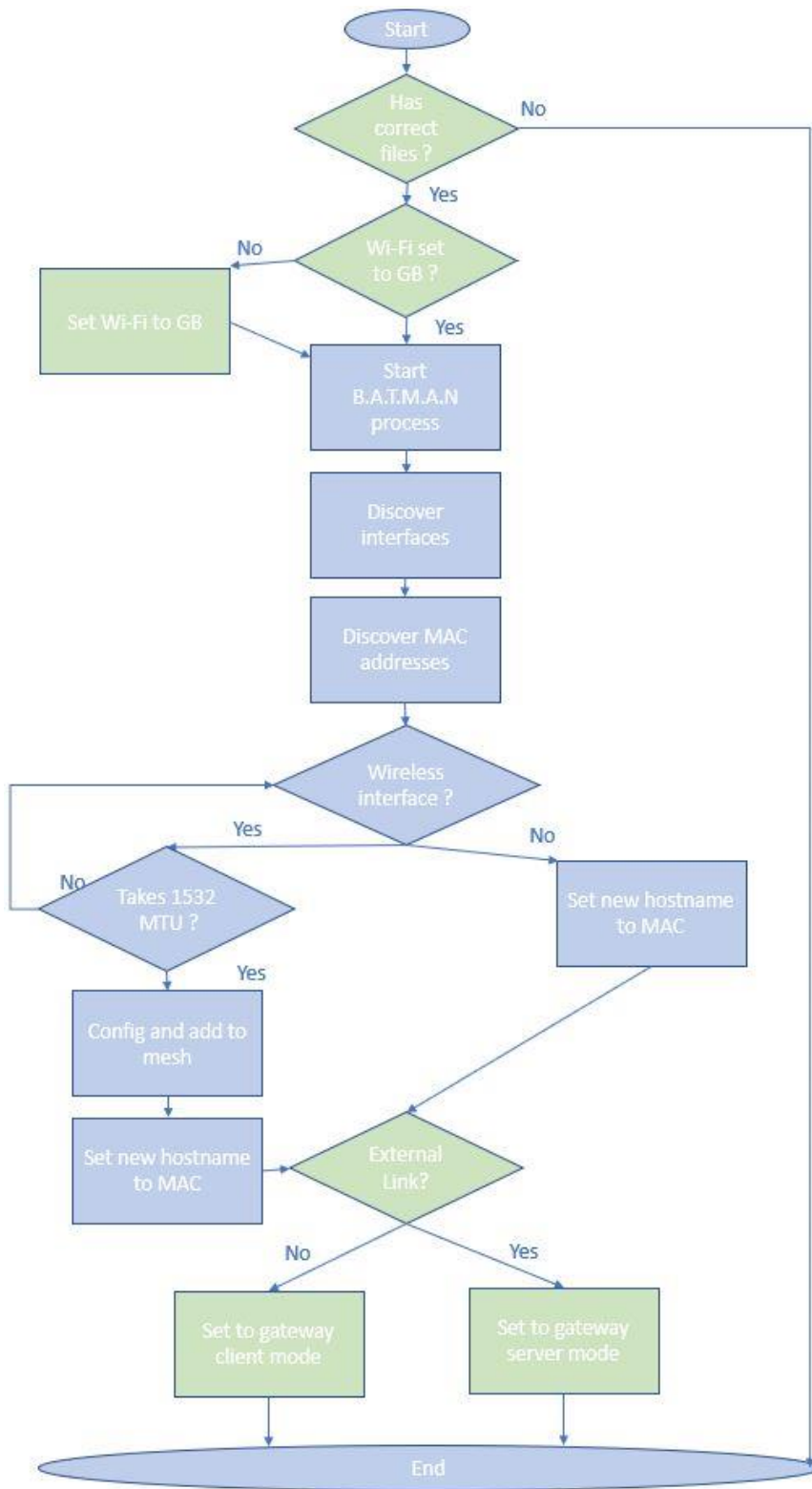


Figure 31 Iteration 2 Script Process Flow

6.1.1 Creating Custom Log Files

The creation of the log files is to aid troubleshooting and keep a log of IP addresses discovery within the network. The logs are created by the `autop2p.py` script and its contents deleted every time the script is ran. This is accomplished with the standard library and using Python's built-in file object functions `.write` and `.close` (Python 2019a).

6.1.2 Adding a File Check

The file check has 2 benefits, if there were no check and the file wasn't in the right place then an `IoError` exception would occur. All possible exceptions need to be considered and handled to ensure that the script either continues or takes an alternate course of action. Carrying out the file check at the beginning negates the requirement to handle the exception and avoids any wasted processing between the script initiation and the file editing function. Python 2.7 has an `os.path.exists` function that exists for just this task. If a file exists, then a `true` value is returned and if not, the `false` value is returned (Python 2019d).

6.1.3 Addressing

The IP addressing solution requirements are as follows:

- Capable of providing both IPv6 and IPv4 addressing.
- Carry out the addressing with minimal network overheads.
- Contain a method to discover both the host and neighbouring IP addresses.
- Contain a way to avoid duplicate addressing.

6.1.3.1 IPv6 SLAAC

IPv6 uses something called Stateless Address Auto Configuration (SLAAC) to achieve automated link-local or global network addressing without the need for a centralised server as described in RFC4862 (Hinden and Deering 2007). This is built into the kernel, so no extra configuration is required to achieve IPv6 addressing.

Global IPv6 addressing is achieved by taking the interfaces gateway, learned via Router Advertisements (RAs), and an interface's MAC address with `ff:ee` added in the middle of it (fig 32).

```
2001:db80:0000:0000:0000:0000:0001 Gateway
0123:4567:89ab          MAC
0123:45ff:fe67:89ab      MAC after padding
2001:db80:0000:0000:0123:45ff:fe67:89ab Final IP address
```

Figure 32 SLAAC Global Addressing Format (Internet Society 2019)

Link-local addressing is achieved by prefixing the address with `fe80::`, the right most bits are the interface MAC address with `ff:ee` added in the middle just like in a global address, the remaining space is padded with zeros. In IPv6 strings of consecutive zeros can be abbreviated by leaving just the colons.

Figure 33 is a Wireshark summary of the IPv6 addressing from within a capture. The addressing has been allocated by SLAAC to both the virtual `bat0` interface and the physical wireless interface.

The destination address `fe02::2` is an IPv6 multicast address for all routers within a link-local network. The summary shows both the virtual and physical interfaces of the nodes trying to find a gateway from which they would be able to retrieve the details for a global address (IANA 2018).

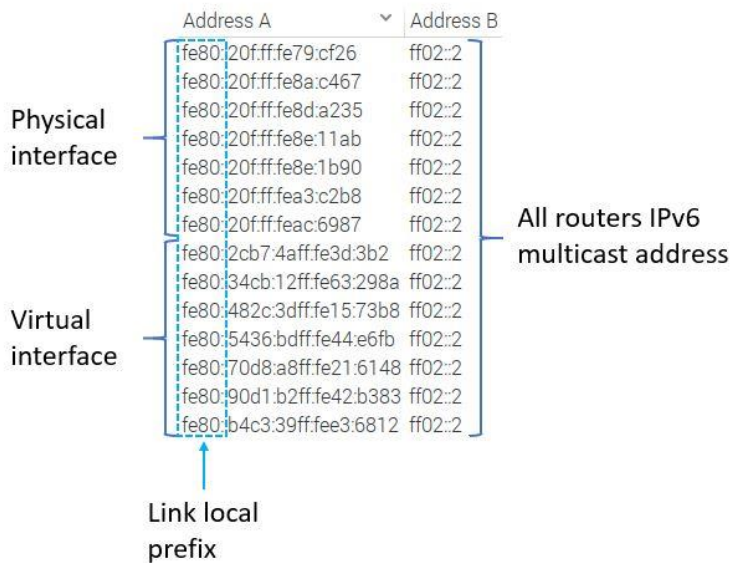


Figure 33 SLAAC Link-Local Addressing Capture

Upon checking the interface configuration with `ifconfig`, no global addressing had been assigned as there is no gateway, therefore nodes can only communicate within the local network.

SLAAC causes no issues with IPv4, both IPv6 and IPv4 addressing can coexist on a network as IPv6 is not backwardly compatible with IPv4. An interface can also have both an IPv6 address and an IPv4 address, this is called dual stacking (Nordmark and Gilligan 2005).

6.1.3.2 IPv4 Addressing

While IPv6 brings many benefits over IPv4 it is less commonly used, Google statistics at the time of writing claim only 23.86% of UK IP addressing is IPv6 (Google 2019). Bournemouth University also used IPv4 for its internal private addressing (fig 34). For these reasons and to enable integration without the need for address translation an automated method of IPv4 addressing is required.

```
IPv4 address:      10.72.38.8
IPv4 DNS servers:  10.254.10.13
                  10.254.10.8
Primary DNS suffix: student.bournemouth.ac.uk
Manufacturer:     Intel
Description:       Intel(R) Ethernet Connection (5) I219-LM
Driver version:    12.17.10.6
Physical address (MAC): 18-60-24-E5-E8-B7
```

Figure 34 Bournemouth University IPv4 Addressing

For private networks 3 address blocks are defined in RFC1918 shown in figure 35 (Rekhter et al 1996).

10.0.0.0	-	10.255.255.255	(10/8 prefix)
172.16.0.0	-	172.31.255.255	(172.16/12 prefix)
192.168.0.0	-	192.168.255.255	(192.168/16 prefix)

Figure 35 IPv4 Private Address Ranges (Rekhter et al 1996)

To avoid addressing conflicts when connecting the robots to Bournemouth University's internal network a network address of 10.100.1.0 /24 was decided upon. A /24 subnet mask allows up to 254 robots to be part of each cell. There are 2 reserved addresses, 255 is the broadcast address and 0 is the network address.

To completely stop link-local addressing and other unwanted addressing issues an implicit deny statement is used at the start of the system interface configuration method in a file called `dhcpcd.conf`. This successfully stops all system configuration but has 2 unwanted side effects. The first effects WPA2 configuration, this uses a service called WPA supplicant which relies on the interfaces being configured initially by the `dhcpcd.conf` file. An additional section of code is also required to bring the interfaces into an up state or the `autop2p.py` script fails.

Inspired by the IPv6 method of referencing the MAC address to provide uniqueness the IPv4 solution does something similar. A MAC address is broken down into 2 parts, the Organisationally Unique Identifier (OUI) is vendor specific and takes up the first 3 octets. As the project hardware is all from the same vendor these values are all identical. The remaining octets are the Network Interface Controller (NIC) value is unique to the vendor, these are the octets from which the IPv4 addressing is derived. Only 1 of these 3 octets is used for the host part of the IPv4 address but needs to be unique within the network (fig 36).

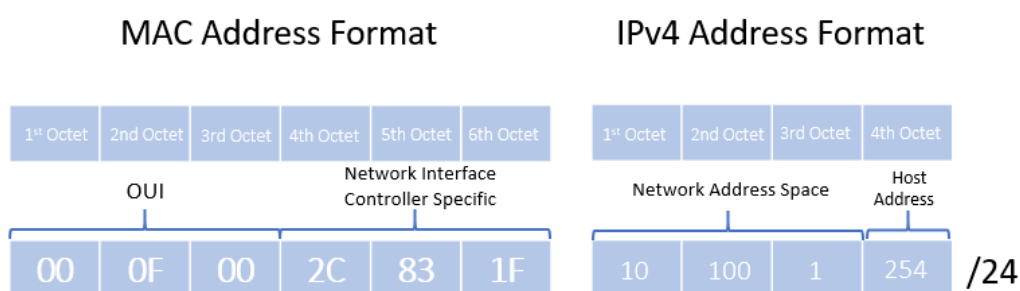


Figure 36 MAC Address and IPv4 Address Format

An octet gets its name from its binary notation in which it has 8 bits. The underlying binary structure of the octets in IPv4 and the MAC address are the same, they are just represented differently. Figure 37 shows how each representation is derived from the same binary octet.

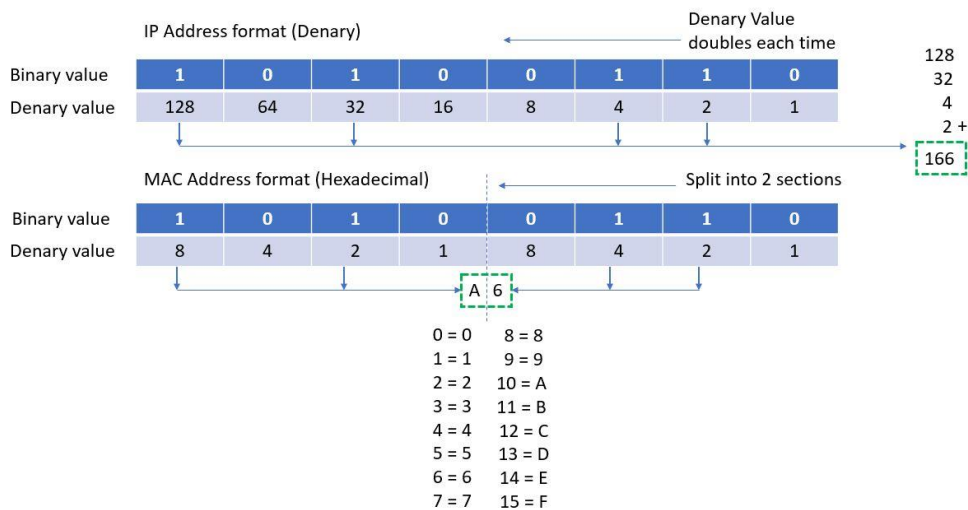


Figure 37 Binary, Hexadecimal & Denary Comparison

Hexadecimal numbering has 16 possible values ranging from 0 to F, in the example the left half of the hexadecimal representation is equal to 10, which is A in hexadecimal. The right-hand half is 6 which is also 6 in hexadecimal. Denary is base 10, each value can be 0 to 9, the same value is represented as 166 in denary.

Hexadecimal can be converted to a denary string within Python using the method shown in figure 38.

```
test_hex = 'A6'
test_hex_to_denary = str((int(test_hex, 16)))
```

Figure 38 Hexadecimal to Denary Conversion

The neighbouring nodes interface MAC addresses is extracted from the B.A.T.M.A.N Adv Originator table. The neighbouring MAC address octets are checked against the hosts to ensure they are unique before converting to denary and applying the IP address. The addressing process flow below depicts the sequential octet comparison and addressing mechanics (fig 39).

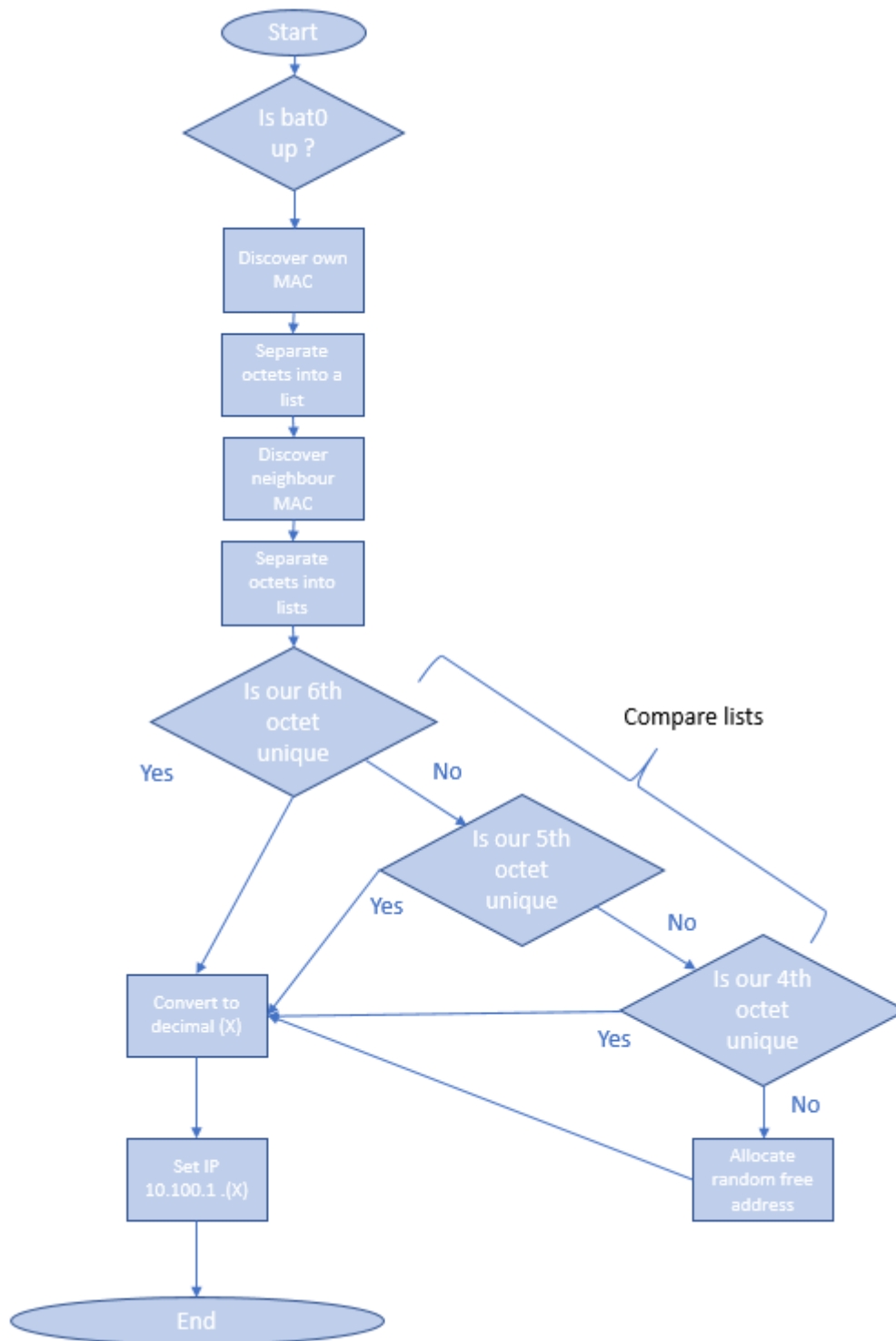


Figure 39 Addressing Process Flow

If none of the octets are unique an `addressing_backstop` function allocates a random free address based on the other addresses that are already known. The downside of this is that it isn't derived from the MAC so isn't as easily traceable to the specific host. Initially the addressing was carried out with a separate script started by the `rc.local` service, to allow variables to be shared it made sense to move the addressing into the main `autop2p.py` script (Raspberry Pi Foundation 2019).

6.1.3.3 IP Address Discovery

A simple Python script was created that uses the same approach as the addressing function to extract the last octets of both the host and neighbouring hosts MAC addresses from the Originator table output and convert them to denary.

6.1.4 Adding the Gateway Functionality

For an IoT deployment the robots must have an Internet connection. The first step to achieving this requires the setup of a B.A.T.M.A.N Adv gateway function which helps nodes find the best path to an external network or the Internet. This is achieved by setting a node to either 1 of 2 settings, a gateway client or a gateway server. The gateway function is built around the assumption that the gateway server who has internet/external connectivity is also a DHCP server, which in this case it isn't.

The reason it is of benefit to this project is that the protocol lists the gateway servers in a gateway table. The addresses could be extracted for use in another script or service that runs as a daemon. By leaving the Ethernet or spare 802.11 interface in DHCP mode and with some further Network Address Translation (NAT) configuration and port forwarding the nodes behind the server could share the internet connection.

External connectivity is checked on all nodes using an external ping to 8.8.8.8 (Googles primary domain name server). The output is checked using a similar method as previous functions with grep and awk. Dependant on the output the interfaces are then set to the required gateway mode with the following command.

```
sudo batctl gw mode {client|server}
```

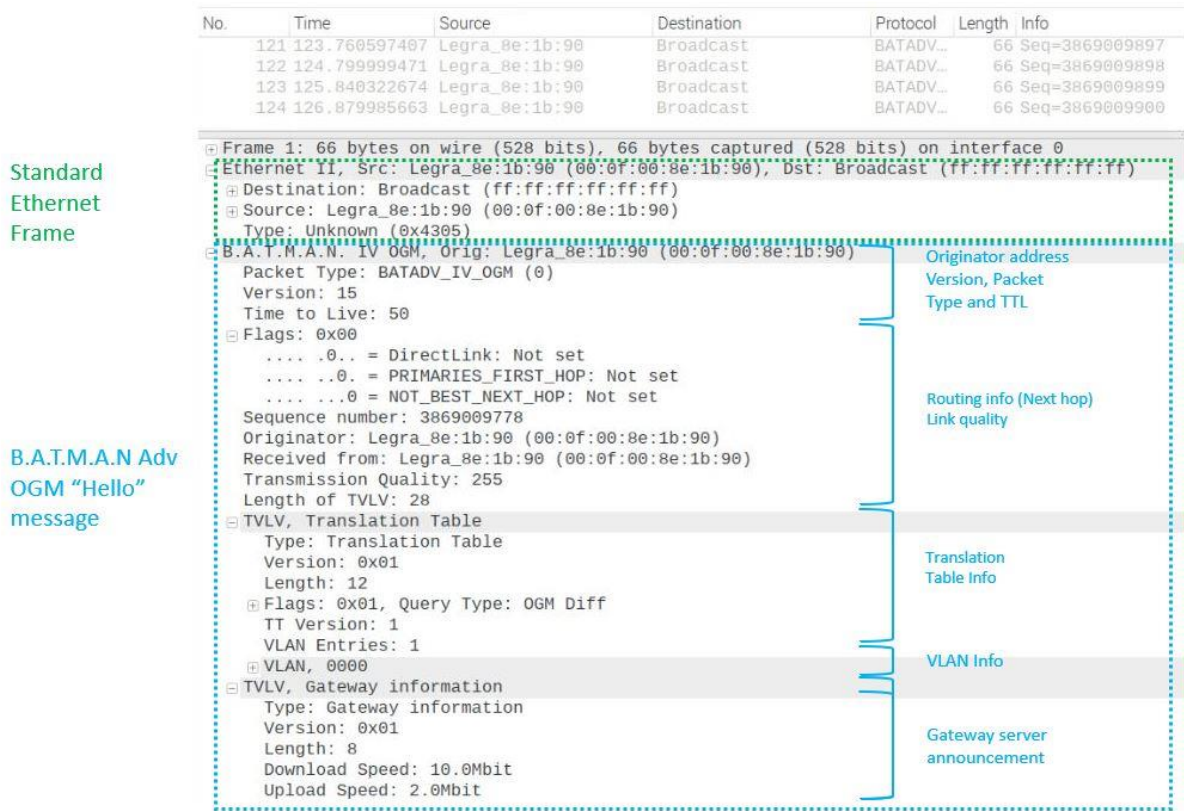


Figure 40 B.A.T.M.A.N Adv OGM Message Capture

The gateway server announcement in the capture above (fig 40) confirms that the gateway settings have been applied.

6.2 Evaluation

The evaluation consists of 3 experiments and a complete system check.

6.2.1 Experiments

The previous tests carried out during and at the end of iteration 1 were carried out throughout iteration 2

To evaluate the final solution 3 experiments, assess the overall performance of the solution were conducted, table 6 shows the experiment details and where they are attached within this report.

3 experiments were carried out as shown in table 6.

Experiment Number	Experiment Name	Appendix
1	Evaluating a multihop wireless network for latency and loss	E
2	Evaluating the time taken from powering on to full connectivity	F
3	Evaluating the WEP network security	G

Table 6 Experiment Details

6.2.2 Complete System Check

To fully test all the separate components of the AutoP2P tool a complete rebuild was carried out with 4 robots. This was done using a stock Raspbian image and installing the project solution using the `install.sh` script. The `ipcheck.py` script was used to identify neighbouring IP addresses and the end of iteration 1 tests carried out successfully.

2 tests were ran using the Sun Founder software that are used for testing the front steering servo and the motors of the Pi Car S robots, both were successful. The 2 logs were also checked for accuracy, no errors were found.

7 CONCLUSIONS

The project has been a success overall as it meets the success criteria outlined in section 1.3. As per the project objectives (1.2) an in-depth background study was carried out which led to a competent design utilising the feature rich B.A.T.M.A.N Adv protocol combined with several unique features. A custom IPv4 addressing solution was created which removes the reliance on link-local addressing and is combined with an IP address log and IP discovery script which simplifies the user experience.

The chosen method of network automation has proven to be reliable and during the evaluation was found to be capable of establishing an ad-hoc multi-hop wireless mesh network in under 25 seconds by doing nothing more than powering the robots on. Once established the network appears to the upper layer applications that each node is connected to a big virtual switch, this includes the Pi Car S software, achieving the project aim (1.2).

While the solution meets all the project requirements (appendix C) the WEP security standard implemented does provide a level of security but has been proven to be weak and easily breakable, improving security should be a priority for future work. The addressing function introduces no network overheads but has a potential flaw in that it cannot detect out of range nodes that have been dropped from the Originator table. This is not deemed to be a major issue as the network is not mission critical, it could however be improved if required.

The chosen project methodology provided the structure required to meet the project deadlines while allowing for some flexibility when the inevitable project challenges occurred. At several times during the project issues occurred that required more than the allocated time to rectify. This extra time may not be available in some situations, the lessons learnt during this project will be of great benefit in future projects.

7.1 Future work

7.1.1 Set Up an Automatic Virtual Private Network (VPN) Connection

A secure external connection could be established automatically by adding some additions to the `autop2p.py` script and using a solution such as PiVPN. PiVPN automates the configuration of an OpenVPN server and produces all the necessary certificates and `.ovpn` configuration files for use with remote clients. The VPN configuration could be configured on the nodes that have detected external connectivity and are automatically configured as gateway servers. This would provide end to end security through the local host network and over the Internet and enable secure IoT connectivity (The PiVPN Project 2019).

7.1.2 Improve Wireless Security

Improving the security of the network should be a priority for any future work. WEP is notoriously weak, WPA 2 is considered a better alternative but is also flawed when using a pre-shared key. A certificate-based solution like RADIUS that works in a P2P network needs to be found or created.

7.1.3 Improve IPv4 Addressing

To enable network segregation and increase the number of subnets from 1 to 13 the addressing function could be altered to change the 3rd octet of the host IPv4 address to the wireless channel the host cell is operating on. This would also allow an increase in the number of possible hosts from 254 to 3302

The project method of IPv4 addressing has no way of detecting nodes that are out of range at the time the script executes. This could be improved by creating and distributing a list of neighbouring IP addresses to all nodes or adding a way of detecting IP address conflicts and automatically re-addressing to an unused IP. Any network wide distribution will add network overheads that will increase as the network grows, this is a design decision to consider for any future development.

Word count (main body of the report): 11235

Word count (artefact): 3760

Total: 14995

8 REFERENCES

Agile Business, 2019. *MoSCoW Prioritisation* [online]. Available from:

<https://www.agilebusiness.org/content/moscow-prioritisation>. [Accessed 15th Feb 2019].

Arch Linux, 2019. *Bash* [online]. Available from: <https://wiki.archlinux.org/index.php/Bash>.

[Accessed 14th Apr 2019].

Arch Linux, 2019. *Sudo* [online]. Available from: <https://wiki.archlinux.org/index.php/Sudo>.

[Accessed 18th Apr 2019].

Ateeq, M., Habib, H., Umer, A. and Rehman, M. 2014. *C++ or Python? Which One to Begin with: A Learner's Perspective* [online]. Available from: <https://ieeexplore.ieee.org/document/6821830>.

[Accessed 11th Apr 2019].

Al-Fuqaha, A., Mohsen, G., Mohammadi, M., Aledhari, M. and Ayyash, M. 2015. *Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications* [online]. Available from:

<https://ieeexplore.ieee.org/document/7123563>. [Accessed 25th Feb 2019].

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. 2001. *Manifesto for Agile Software Development* [online]. Available from: <https://agilemanifesto.org/>. [Accessed 27th Apr 2019].

Red Hat, 2019. *Red Hat Ansible* [online]. Available from: <https://www.ansible.com/>. [Accessed 2nd Apr 2019].

Bluetooth SIG, 2019. *Core Specifications* [online]. Available from:

<https://www.bluetooth.com/specifications/bluetooth-core-specification>. [Accessed 17th Feb 2019].

Carrol, B. 2009. *CCNA Wireless Official Exam Certification Guide*. 1st ed. Indianapolis: Cisco Press.

Cheshire, S. Aboba, B. and Guttman, E. 2005. *Dynamic Configuration of IPv4 Link-Local*

Addresses [online]. Available from: <https://tools.ietf.org/html/rfc3927>. [Accessed 18th Apr 2019].

Chroboczek, J. 2011. *The Babel Routing Protocol* [online]. Available from:

<https://tools.ietf.org/html/rfc6126#section-1.2>. [Accessed 8th Apr 2019].

Cisco, 2016. *Internet of Things*. [online]. Available from: <https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.pdf>. [Accessed 5th Mar 2019].

Cypress Semiconductor Corporation, 2018. *Single-Chip IEEE 802.11 b/g/n MAC/Baseband/Radio with Integrated Bluetooth 4.2* [online]. Available from: <https://www.cypress.com/file/298076/download>. [Accessed 4th Feb 2019].

Debian, 2015. *Avahi-autoipd* [online]. Available from: <https://manpages.debian.org/jessie/avahi-autoipd/avahi-autoipd.8.en.html>. [Accessed 18th Apr 2019].

Debian, 2017a. *Awk* [online]. Available from: <https://manpages.debian.org/testing/original-awk/awk.1.en.html>. [Accessed 17th Apr 2019].

Debian. 2017b. *Daemon* [online]. Available from: <https://wiki.debian.org/Daemon>. [Accessed 6th May 2019].

Debian, 2019a. *Grep* [online]. Available from: <https://packages.debian.org/stable/grep>. [Accessed 17th Apr 2019].

Debian, 2019b. *Runlevel(8)* [online]. Available from: <https://manpages.debian.org/stretch/systemd-sysv/runlevel.8.en.html>. [Accessed 18th Feb 2019].

Debian, 2019c. *Systemd* [online]. Available from: <https://wiki.debian.org/systemd> [Accessed 18th Feb 2019].

Debian, 2019d. *Systemd.Service(5)* [online]. Available from: <https://manpages.debian.org/stretch/systemd/systemd.service.5.en.html>. [Accessed 18th Feb 2019].

Debian, 2019e. *Systemd.Unit(5)* [online]. Available from: <https://manpages.debian.org/stretch/systemd/systemd.unit.5.en.html>. [Accessed 18th Feb 2019].

Dye, M., McDonald, R. and Ruff, A. 2008. *Network Fundamentals: CCNA exploration companion guide*. 10th ed. Indianapolis: Cisco Press. p53,74-75

- Foster, E. 2014. *A Comparative Analysis of the C++, Java, and Python Languages* [online]. Available from: https://www.researchgate.net/publication/320407173_A_COMPARITIVE_ANALYSIS_OF_THE_C_JAVA_AND_PYTHON_LANGUAGES. [Accessed 14th Apr 2019].
- GitHub, 2019. *GitHub Help* [online]. Available from: <https://help.github.com/en>. [Accessed 29th Apr 2019].
- Google, 2019. *IPv6 Statistics* [online]. Available from: <http://www.google.com/intl/en/ipv6/statistics.html#tab=per-country-ipv6-adoption&tab=per-country-ipv6-adoption>. [Accessed 23rd Apr 2019].
- Hinden, R. and Deering, S. 2007. *IPv6 Stateless Address Autoconfiguration* [online]. Available from: <https://tools.ietf.org/html/rfc4862#section-5.3>. [Accessed 23rd Apr 2019].
- Hundebøll, M. and Ledet-Pedersen, J. 2011. *Inter-Flow Network Coding for Wireless Mesh Networks* [online]. Available from: https://downloads.open-mesh.org/batman/papers/batman-adv_network_coding.pdf. [Accessed 8th May 2019].
- Hundebøll, M. 2014. *Better Approach To Mobile Adhoc Networking B.A.T.M.A.N Adv Kernel Space L2 Mesh Routing* [online]. Available from: <https://www.open-mesh.org/attachments/download/416/slides.pdf>. [Accessed 11th Mar 2019].
- IANA, 2018. *IPv6 Multicast Address Space Registry* [online]. Available from: <https://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml>. [Accessed 23rd Apr 2019].
- IEEE, 2003. *802.15.4 -2003 - IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)* [online]. Available from: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1237559>. [Accessed 6th Mar 2019].
- IEEE, 2009. *IEEE 802.11n – 2009* [online]. Available from: <https://ieeexplore-ieee-org.libezproxy.bournemouth.ac.uk/stamp/stamp.jsp?tp=&arnumber=5307322>. [Accessed 8th Apr 2019].

IEEE, 2015. *IEEE Standard for Low Rate Wireless Networks* [online]. Available from: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7460875&tag=1>. [Accessed 5th Mar 2019].

Internet Society, 2019. *Privacy Extensions for IPv6 SLAAC* [online]. Available from: <https://www.internetsociety.org/resources/deploy360/2014/privacy-extensions-for-ipv6-slaac/>. [Accessed 23rd Apr 2019].

Interopnet Labs. 2008. *Whats wrong with WEP ?* [online]. Available from: <http://www.opus1.com/www/whitepapers/whatswrongwithwep.pdf>. [Accessed 9th May 2019].

Kali Tutorials, 2014. *Hack WPA/WPA2 PSK Capturing the Handshake* [online]. Available from: <https://www.kalitutorials.net/2014/06/hack-wpa-2-psk-capturing-handshake.html>. [Accessed 7th May 2019].

Kali Tutorials. 2014. *Wifite : Hacking Wifi The Easy Way : Kali Linux* [online]. Available from: <https://www.kalitutorials.net/2014/04/wifite-hacking-wifi-easy-way-kali-linux.html>. [Accessed 9th May 2019].

Kurth, O. 2018. *Avahi-autoipd* [online]. Available from: <https://linux.die.net/man/8/avahi-autoipd>. [Accessed 1st Apr 2019].

Lang, J. 2019. *Open Mesh Wiki* [online]. Available from: <https://www.open-mesh.org/projects/open-mesh/wiki>. [Accessed 11th Mar 2019].

Mediatek, 2019. *RT5370 High-performance 802.11n Wi-Fi with antenna diversity switching* [online]. Available from: <https://www.mediatek.com/products/broadbandWifi/rt5370>. [Accessed 8th Apr 2019].

Microsoft, 2019. *IoT Hub* [online]. Available from: <https://azure.microsoft.com/en-gb/services/iot-hub/>. [Accessed 2nd Apr 2019].

Montenegro, G., Kushalnagar, N., Hui, J. and Culler, D. 2017. *RFC4944 Transmission of IPv6 over IEEE 802.15.4 Networks* [online]. Available from: <https://tools.ietf.org/html/rfc8137#ref-IEEE802.15.4>. [Accessed 5th Mar 2019].

Neumann, A., Aichele, C. and Lindner, M. 2007. *B.A.T.M.A.N Status Report* [online]. Available from: <https://downloads.open-mesh.org/batman/papers/batman-status.pdf>. [Accessed 8th Apr 2019].

Nordmark, E. and Gilligan, R. 2005. *RFC 4213 Basic Transition Mechanisms for IPv6 Hosts and Routers* [online]. Available from: <https://tools.ietf.org/html/rfc4213>. [Accessed 30th Apr 2019].
OpenCV team, 2019. *About OpenCV* [online]. Available from: <https://opencv.org/about.html>. [Accessed 30th Mar 2019].

Ofcom. 2019. *Ofcom UK Frequency Allocation* [online]. Available from: <http://static.ofcom.org.uk/static/spectrum/fat.html>. [Accessed 6th May 2019].

Patel, K. and Scholar, P. 2016. *Internet of Things - IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges* [online]. Available from: <http://ijesc.org/upload/8e9af2eca2e1119b895544fd60c3b857.Internet%20of%20Things-IOT%20Definition,%20Characteristics,%20Architecture,%20Enabling%20Technologies,%20Application%20&%20Future%20Challenges>. [Accessed 5th Mar 2019].

The PiVPN Project, 2019. *PiVPN* [online]. Available from: <http://www.pivpn.io/>. [Accessed 7th May 2019].

Python, 2019a. *Built-in Types* [online]. Available from: <https://docs.python.org/2/library/stdtypes.html?highlight=write#file.write>. [Accessed 30th Apr 2019].

Python, 2019b. *commands* [online]. Available from: <https://docs.python.org/2/library/commands.html?highlight=getoutput#commands.getoutput>. [Accessed 18th Apr 2019].

Python, 2019c. *os* [online]. Available from: <https://docs.python.org/2/library/os.html?highlight=os%20system#module-os>. [Accessed 29th Apr 2019].

Python, 2019d. *os.path* [online]. Available from: <https://docs.python.org/2/library/os.path.html>. [Accessed 23rd Apr 2019].

Python, 2019e. *re* [online]. Available from: <https://docs.python.org/2/library/re.html?highlight=re#module-re>. [Accessed 29th Apr 2019].

Python, 2019f. *string* [online]. Available from: <https://docs.python.org/2/library/string.html>. [Accessed 29th Apr 2019].

PMI, 2018. *Pulse of the Profession, the 10th Global Project Management Survey* [online]. Available from: <https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2018.pdf>. [Accessed 28th Apr 2019].

Rasmusson, J. 2019. *How is Agile different* [online]. Available from: http://www.agilenutshell.com/how_is_it_different. [Accessed 2nd May 2019].

Raspberry Pi Foundation, 2015. *What is a Raspberry Pi?* [video, online]. YouTube. Available from: https://www.youtube.com/watch?time_continue=100&v=uXUjwk2-qx4 [Accessed 26th Feb 2019].

Raspberry Pi Foundation, 2018. *Raspberry Pi 3 Model B* [online]. Available from: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. [Accessed 4th Feb 2019].

Raspberry Pi Foundation. 2019. *rc.local* [online]. Available from: <https://www.raspberrypi.org/documentation/linux/usage/rc-local.md>. [Accessed 9th May 2019].

Raspbian, 2018. *Raspbian FAQ* [online]. Available from: <https://raspbian.org/RaspbianFAQ>. [Accessed 27th Feb 2019].

Rekhter, Y., Moskowitz, B., Karrenberg, D., Groot, G. and Lear, E. 1996. *Address Allocation for Private Internets* [online]. Available from: <https://tools.ietf.org/html/rfc1918>. [Accessed 23rd Apr 2019].

Rigney, C., Willens, S., Rubens, A. and Simpson, W. 2000. *Remote Authentication Dial In User Service (RADIUS)* [online]. Available from: <https://www.ietf.org/rfc/rfc2865.txt?number=2865>. [Accessed 9th May 2019].

Royce, W. 1970. *Managing the development of large software systems* [online]. Available from: <http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf> [Accessed 17th Feb 2019].

Sendra, S., Fernandez, P., Turro, C. and Lloret, J. 2010. *IEEE 802.11a/b/g/n Indoor Coverage and Performance Comparison* [online]. Available from: <https://ieeexplore-ieee-org.libezproxy.bournemouth.ac.uk/stamp/stamp.jsp?tp=&arnumber=5629058>. [Accessed 3rd Mar 2019].

Simeons, P., Dragone, M. and Saffiotti, A. 2018. *The Internet of Robotic Things: A review of concept, added value and applications* [online]. Available from: <https://journals.sagepub.com/doi/pdf/10.1177/1729881418759424>. [Accessed 7th Mar 2019].

Scrum Alliance. 2019. *Learn About Scrum* [online]. Available from: <https://www.scrumalliance.org/learn-about-scrum>. [Accessed 2nd May 2019].

Sun Founder. 2018. *SunFounder PiCar-S Kit V2.0 for Raspberry Pi 3/2/B+* [online]. Available from: <https://www.sunfounder.com/picar-s-kit.html>. [Accessed 26th Apr 2019].

Tobias, P., Eckenfels, B. and Meskes, M. 2009. *Hostname* [online]. Available from: <https://manpages.debian.org/stretch/hostname/hostname.1.en.html>. [Accessed 29 Apr 2019].

The Python Software Foundation, 2019. *Python 2.7.16 documentation* [online]. Available from: <https://docs.python.org/2/index.html>. [Accessed 20th Mar 2019].

Vanhoef, M. 2017. *Key Reinstallation Attacks* [online]. Available from: <https://www.krackattacks.com/>. [Accessed 9th May 2019].

Wireshark. 2019. *About Wireshark* [online]. Available from: <https://www.wireshark.org/>. [Accessed 5th May 2019].

Wu, P. 2019. *Platform-Specific information about capture privileges* [online]. Available from: https://wiki.wireshark.org/CaptureSetup/CapturePrivileges#GNU.2FLinux_distributions.2C_Wireshark_is_installed_using_a_package_manager. [Accessed 13th Mar 2019].

9 APPENDIX A – PROJECT PROPOSAL

BU Computing Programmes 2018-2019

Undergraduate Project Proposal Form

Degree Title: Computer Networks	Student's Name:
	Jamie Murray
	Supervisor's Name:
	Dr N Chechina
	Project Title/Area:
	An Automatic P2P Network Configuration of Autonomous Devices

Section 1: Project Overview

1.1 Problem definition - use one sentence to summarise the problem:

Setting up a peer to peer wireless network on autonomous devices can be too complex and time consuming.

1.2 Project description - briefly explain your project:

Create a solution for automatically connecting two Raspberry Pi based devices straight out of the box. This will be accomplished using bash and python scripting.

1.3 Background - please provide brief background information, e.g., client:

The use case scenario for the project is based around 2 Raspberry Pi based robot casualty recovery vehicles that will carry out a searching function. They need to be able to communicate to alert each other of when they find a casualty. The network setup phase needs to be as simple as possible to speed up deployment and ideally contain some form of redundancy and failure recovery.

1.4 Aims and objectives – what are the aims and objectives of your project?

The aim is to create a script that runs on system start-up that enables a wireless interface, creates unique ad hoc addressing, searches for other devices and connects to them. A process needs to be in place to periodically check connectivity and if necessary re-establish links. Some form of alert needs to be activated to confirm to the operator that setup has been successful. The success criteria for the project will be primarily the successful creation of the P2P network, then if possible to build in a failure recovery mechanism and fault tolerance for example the relaying of messages via an alternative route using another device connected to the network. The project will require a large amount of research, an overview of the findings from this research will be part of the dissertation and the knowledge extracted will direct

BU Computing Programmes 2018-2019

the project design. A critical analysis of how the project went, successes, failures and possible future areas of interest will also be included.

Section 2: Artefact

2.1 What is the artefact that you intend to produce?

An automated script that establishes network communications between 2 devices.

2.2 How is your artefact actionable (i.e., routes to exploitation in the technology domain)?

Once created the code will be able to be reused to test functionality. All the test specifications and code will be documented within the project.

Section 3: Evaluation

3.1 How are you going to evaluate your work?

A series of tests will be created to check each part of the design specification have been met. The network communications can be captured with Wireshark and the error logs from the device can be accessed with ease. S.M.A.R.T goals will be set.

3.2 Why is this project honours worthy?

It will demonstrate an in-depth understanding of networking protocols and the ability to automate networking tasks.

3.3 How does this project relate to your degree title outcomes?

The project is based around modern network communication standards and protocols.

3.4 How does your project meet the BCS Undergraduate Project Requirements?

The project addresses a real-world problem, is substantial and requires creativity. It will require the practical application of a project methodology and skills learnt on the degree course as well as analytical skills.

It will allow me to demonstrate my networking, coding, research and project management ability. The project will contain a critical appraisal and all the required content stated in the project handbook that is derived from the official BCS guidelines.

BU Computing Programmes 2018-2019

3.5 What are the risks in this project and how are you going to manage them?

I will keep a risk register containing identified risks and grade them on the likely hood of them occurring and the impact if they do. For each risk one of the 5 risk responses will be employed, either avoid, mitigate, transfer, defer or just accept the risk. As this project has no external stakeholder involvement the risks are mainly centred around my own time management, ability to perform and ensure that my work is backed up in multiple places to prevent a catastrophic failure. To ensure that my time management is accurate I have created a Gantt chart and will factor in some extra time for the risky phases like the development of the solution for example. Time management will be key to this projects success and the effective prioritising of tasks will be carried out in advance. A basic to do list will be utilised to keep track of tasks so none get forgotten. This combined with regular meetings with my project supervisor should keep the project timeline within acceptable levels until completion.

To ensure that my solution is complete and fully functioning a lengthy literature review and study of the areas surrounding the scripting, the relevant wireless protocols, Linux and the hardware specific issues of a Raspberry Pi. One of the main plus points to this project is that everything is so readily available. The task of creating the final design will be broken down into smaller stages and the code submitted to a personal git hub account.

To prevent data loss GitHub, OneDrive and the Bournemouth University servers will be used to back up my work. I will also keep at least one offline copy and ensure that they are saved with a date time group.

Section 4: References

4.1 Please provide references if you have used any.

None used.

Section 5: Ethics (please delete as appropriate)

5.1 Have you submitted the ethics checklist to your supervisor? **Yes**

5.2 Has the checklist been approved by your supervisor? **Yes**

BU Computing Programmes 2018-2019

Section 6: Proposed Plan (please attach your Gantt chart below)

[illegible]

10 APPENDIX B – RISK REGISTER

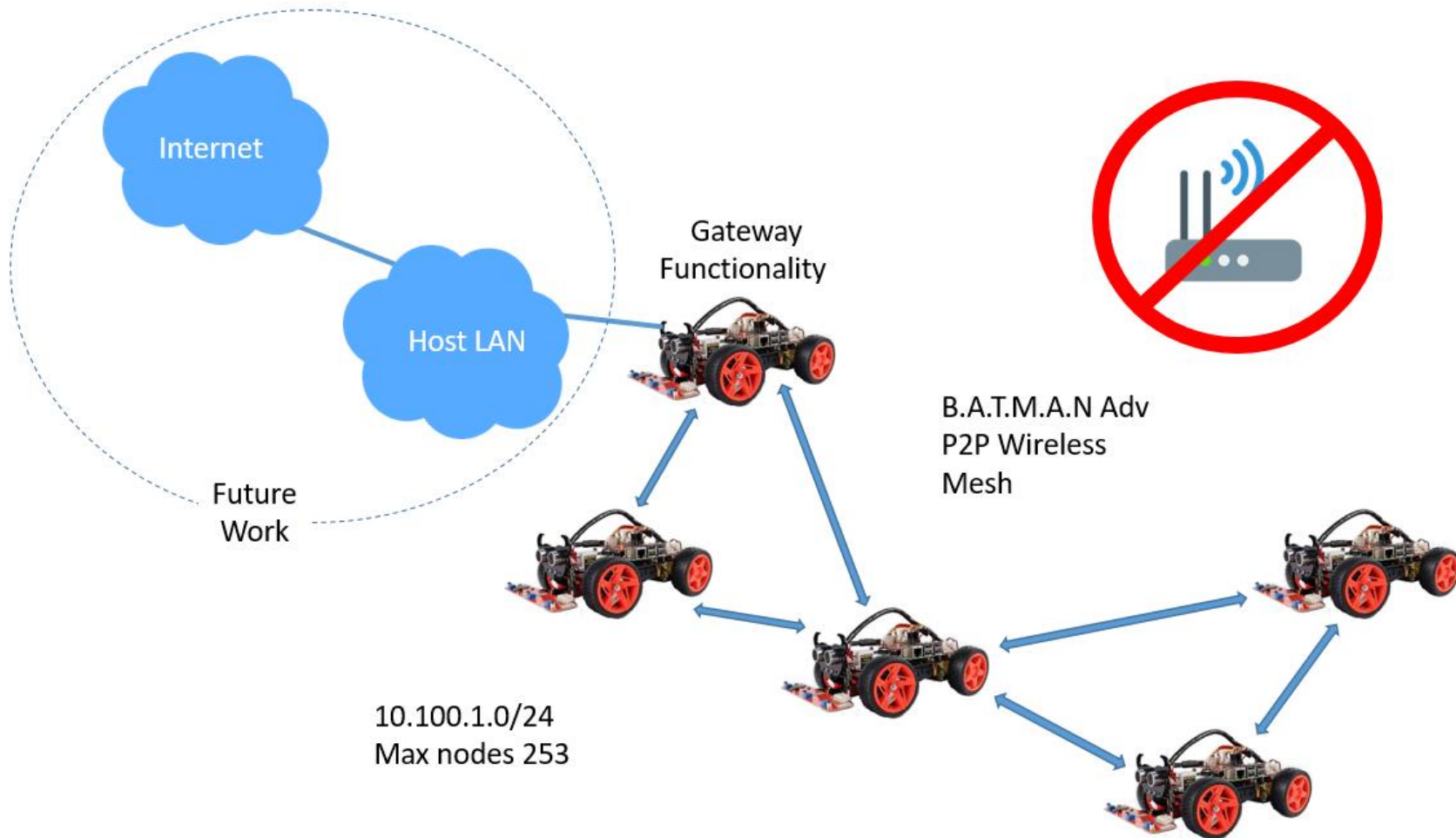
Risk ID	Risk	Effect	Likelihood 1 -3	Impact 1 - 3	Overall Risk Score	Risk Response
1	Unable to code the required functions due to lack of technical ability	The solution will be of poor quality or ineffective	2	2	4	Mitigate by carrying out research prior and during the project.
2	The chosen solutions are not capable of meeting the requirements	The solution will lack essential functionality	1	2	3	Mitigate by ensuring that the capabilities of each component are fully understood prior to selection.
3	The project will run over time due to illness or unforeseen circumstances	An incomplete report or solution will be submitted, or the deadline will be missed	1	3	4	Accept. The University has a mitigating circumstances procedure.
4	The project report or code will become corrupt or be accidentally deleted	The work will need to be redone impacting on project time	2	2	4	Avoid by backing up to GitHub, OneDrive and an external HDD.
5	The printing company chosen to print the report will have problems causing a delay	The report will not be submitted in time	1	3	4	Avoid by ensuring that printing is carried out 24 hours prior to hand in.
6	Poor time management / Lack of commitment	The deadline will be missed, or the project will be of poor quality or ineffective	1	2	3	Carry out frequent progress checks and create a realist schedule. Drop all non-essential activities. Have frequent weekly meetings with my supervisor.

11 APPENDIX C – PROJECT REQUIREMENTS

	Functional Requirements	Iteration	MoSCoW
R1	The solution starts automatically on system boot	1	M
R2	The solution provides peer to peer network connectivity	1	M
R3	The solution requires no configuration from once enable on a device	1	M
R4	The solution can detect compatible network interfaces	1	M
R5	The solution can configure compatible network interfaces	1	M
R6	The solution must be able to provide unique IPv4 addressing	1	M
R7	The solution must be able to provide unique IPv6 addressing	1	M
R8	The solution must ensure the hostname is unique	1	S
R9	The solution provides network security	2	S
R10	The solution provides a means of error logging	2	S
R11	The solution can automatically configure gateway and client mode	2	S
R12	The solution provides remote connectivity		W
R13	The solution has a graphical user interface		W

	Non-Functional Requirements	MoSCoW
RNF1	The solution is scalable	M
RNF2	The solution is maintainable	M
RNF3	The solution is reliable	M
RNF5	The solution has documentation	S
RNF6	The solution is portable	S

12 APPENDIX D – NETWORK DESIGN



13 APPENDIX E – EXPERIMENT 1

Evaluating a multihop wireless network for latency and loss

Objective

To evaluate the effects of varying amounts of hops in a wireless mesh network effects latency and loss.

Rationale

Mobile wireless networks are prone to interference and loss and each extra hop within the wireless mesh will add additional delay due to the additional processing and retransmission required. Therefore, it is important to make an accurate assessment of the impact additional nodes on network performance and reliability.

Procedure

7 Raspberry Pi will be configured using the project solution and network hardware. The experiment will be conducted in an open-air environment using building to shield the RF signal to prevent hops being by passed. The transmission power of the wireless interfaces was reduced to 10db using the following command substituting the wireless interface name where necessary:

```
sudo iwconfig wlan0 txpower 10
```

The experiment layout will be assessed by using the `batctl traceroute` command and ensuring traffic is not shortcutting its direct neighbour.

This IP addresses will be discovered using the `ipcheck.py` script. Each node in turn will have 60 pings sent to it using the following command but changing the IP address to match the node being tested:

```
ping 10.100.1.10 -c 60
```

This will be carried out 3 times for each node. The mean average of the latency and the percentage of packet loss will be recorded for each hop. The experiment map shows nodes numbered from 1 to 7, node 1 was where the pings were initiated from.

Hardware Configuration

- 1 x Pi-Top laptop
- 7 x Raspberry Pi (Model 3 B)
- 7 x Ralink RT5370 wireless adapters
- 7 x Battery packs

Software Configuration

Raspbian Desktop November 2018

B.A.T.M.A.N Adv

AutoP2P tool

Wireless transmission power limited to 10db

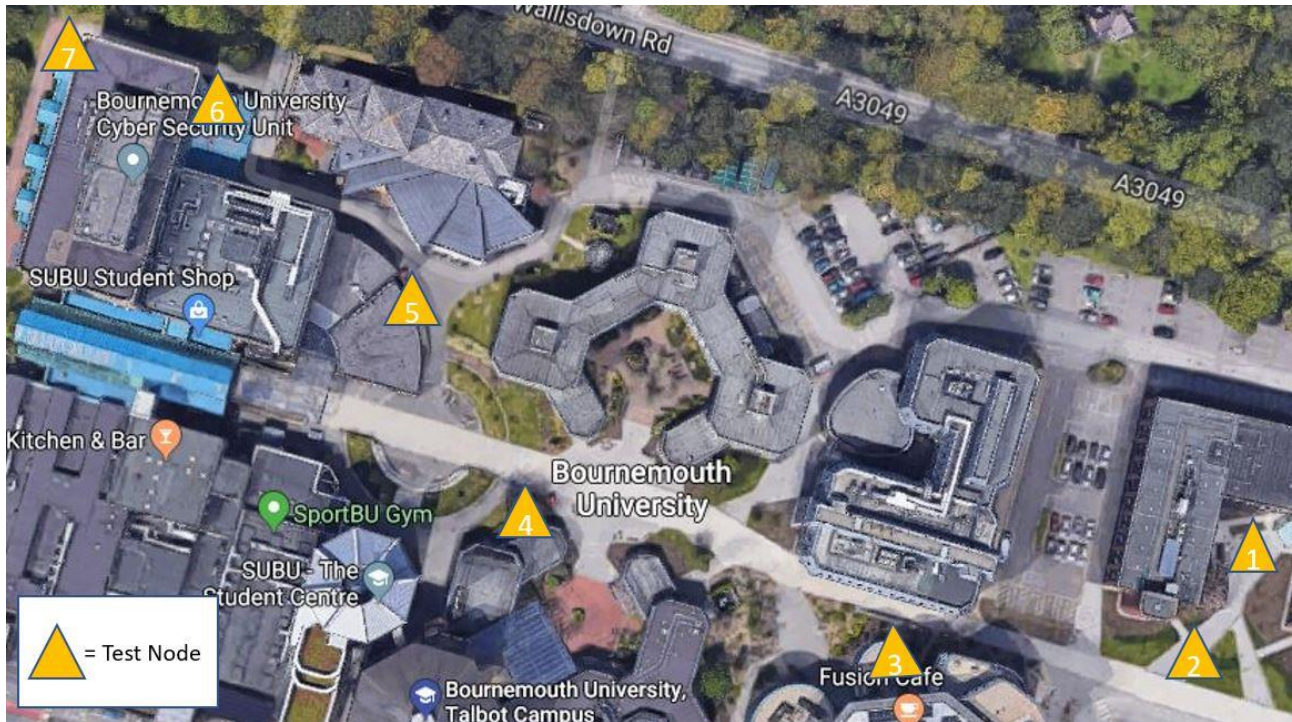


Figure 41 Experiment 1 Map

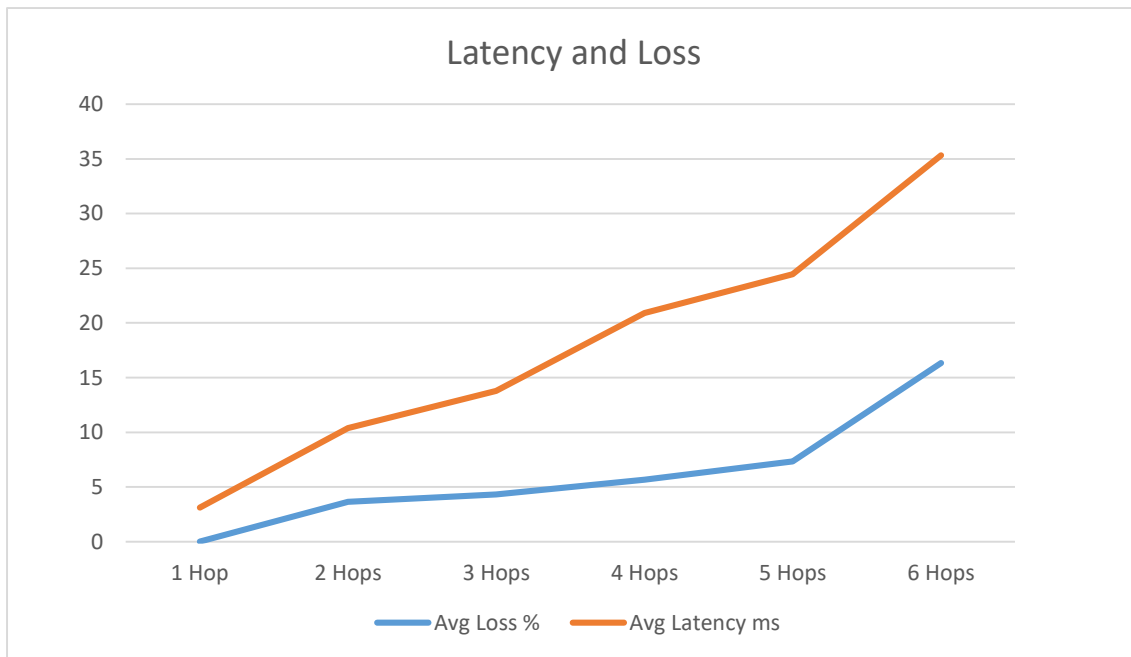
Results

	1 Hop	2 Hops	3 Hops	4 Hops	5 Hops	6 Hops
Test 1 Loss (%)	0	5	4	11	3	13
Test 2 Loss (%)	0	2	4	5	3	20
Test 3 Loss (%)	0	4	5	1	16	16
Average Loss (%)	0	3.66	4.33	5.66	7.33	16.33

Table 7 Experiment 1 Loss

	1 Hop	2 Hops	3 Hops	4 Hops	5 Hops	6 Hops
Test 1 Latency (ms)	1.833	6.615	8.864	14.863	18.944	17.324
Test 2 Latency (ms)	5.961	7.522	8.534	9.445	9.927	20.399
Test 3 Latency (ms)	1.523	6.092	10.986	21.428	22.480	19.243
Average Latency (ms)	3.11	6.74	9.46	15.25	17.12	18.99

Table 8 Experiment 1 Latency



Graph 1 Latency and Loss Results

Conclusion

Both latency and loss increase at each hop. There is some variation in the results and some anomalies. This is most likely down to RF specific issues such as interference, loss, reflection or refraction. The results show neither a truly linear or exponential curve for either latency or loss. Anomalies aside both latency and loss appear to be increasing in a more linear way. The amount of loss is not parallel with latency, as ping packets are unreliable retransmissions cannot be the reason. This could be due to the delay due to buffering, processing and retransmission time required at each hop. Further tests may confirm this.

Despite the transmission power of each wireless interface being changed from 30db to 10db the coverage from the RT5370 is excellent in outdoor environment.

14 APPENDIX F – EXPERIMENT 2

Evaluating the time taken from powering on to full connectivity.

Objective

To evaluate the time taken between powering on a small group of nodes until full IP connectivity.

Rationale

The automated solution causes a small amount of delay in the boot up time. This delay needs to be assessed to gain an accurate understanding of how the network functions.

Procedure

The IP addresses of all 4 wireless interfaces will be discovered using the `ipcheck.py` script. All the Raspberry Pi apart from one will be powered off. A constant ping will be started to each address. The Raspberry Pi will all be turned on simultaneously and a stopwatch started. Once the last IP address has returned a successful ping the stopwatch will be stopped, and the result recorded. This will be carried out 10 times, the mean average will be worked out and used as the result.

Hardware Configuration

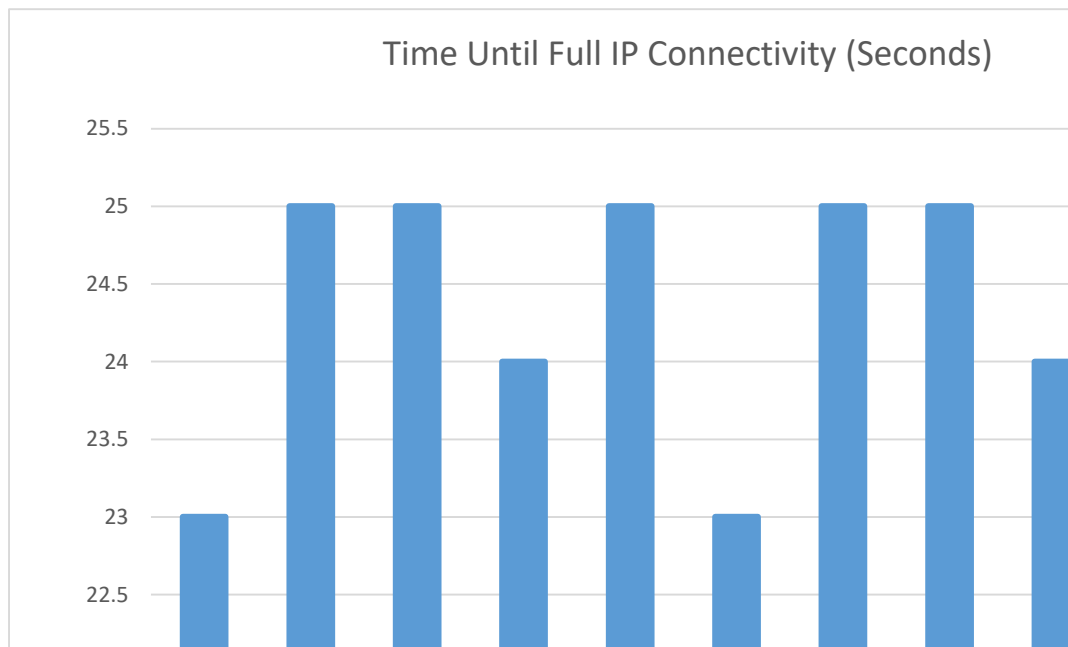
- 1 x Pi-Top Laptop
- 4 x Raspberry Pi (Model 3 B)
- 4 x Ralink RT5370 wireless adapters
- 4 x Battery packs
- 4 x Micro SD cards

Software Configuration

- Raspbian Desktop November 2018
- B.A.T.M.A.M Adv
- AutoP2P configuration tool

Results

Attempt Number	Time (Seconds)
1	23
2	25
3	25
4	24
5	25
6	23
7	25
8	25
9	24
10	23



The average delay is = **24.2 seconds**

Conclusion

The results show a reasonably consistent start up time with no anomalies. This gives a reasonably accurate measurement of delay and shows that the solution is reliable and fast.

15 APPENDIX G – EXPERIMENT 3

Evaluating WEP security with automated attack tools.

Objective

To evaluate the effectiveness of WEP wireless security when used for ad-hoc connections using freely available automated tools.

Rational

WEP has been found to have many flaws and has long been depreciated. The project solution uses WEP as there were issues with implementing WPA2. It is important to assess just how quickly if possible, it can be broken.

Procedure

Two of the project robots will be powered on and check that they are have successfully established and joined a B.A.T.M.A.N Adv mesh network. The wireless attack will be started on a nearby host running Kali Linux. An automated WEP attack will be started using the default settings and with using a fully automated setting to replicate an inexperienced, opportunist hacker. The time it takes to crack the WEP encryption will be recorded. If time allows multiple attempts will be made changing password parameters.

Hardware Configuration

B.A.T.M.A.N Adv hosts	Kali based desktop
2 x Raspberry Pi (Model 3 B)	1 x AMD Ryzen 5 2600 CPU
2 x Ralink RT5370 wireless adapters	16 GB RAM
2 x Micro SD cards	Alfa Networks AR9271 USB Adapter

Software configuration

B.A.T.M.A.N Adv hosts	Kali based desktop
Raspbian Desktop November 2018	Kali Linux 19.1a
B.A.T.M.A.M Adv	Fern WiFi Cracker 1.4
AutoP2P configuration tool	Wifite

Network configuration

```

ssid = "Batmesh"
channel = "1"
mode = "ad-hoc"
cell_id = "00:11:22:33:44:55"
key = "1234567890123"

```

Figure 42 Experiment 3 Network Settings

RESULTS

Run 1

A wireless mesh network was established using the project solution and checked to confirm connectivity. Wifite was started at 22:00, the target network was selected, and the automated attack commenced using a guide found on the Kali Tutorials website (Kali Tutorials 2014). Wifite carried out the following types of attack:

- ARP replay
- Fragment
- Chop chop
- Caffe Latte
- Hirte

At 07:30 the next day it was still running. The test was aborted. It was discovered that the amount of IVs captures was far lower than expected, this was because the network was generating hardly any traffic.

Run 2

Wifite was replaced with an alternative tool called Fern WiFi Cracker. The ARP replay option was selected and a fully automated attack started. A constant ping was started between the two host to increase the number of IVs being captured. After 2 hours the test was aborted as over 10000 IVs were captured and the password still had not been cracked.

Run 3

To increase network traffic a simple python script was written that created hundreds of instances of pings between the hosts with a reduced 200ms gap. The ARP replay option was selected again and a fully automated attack started. The script significantly increased the amount of network traffic

and increased the CPU utilisation to between 50% and 60% on both hosts. The test was stopped after 3 hours when the number of IVs captured reached 64400048.

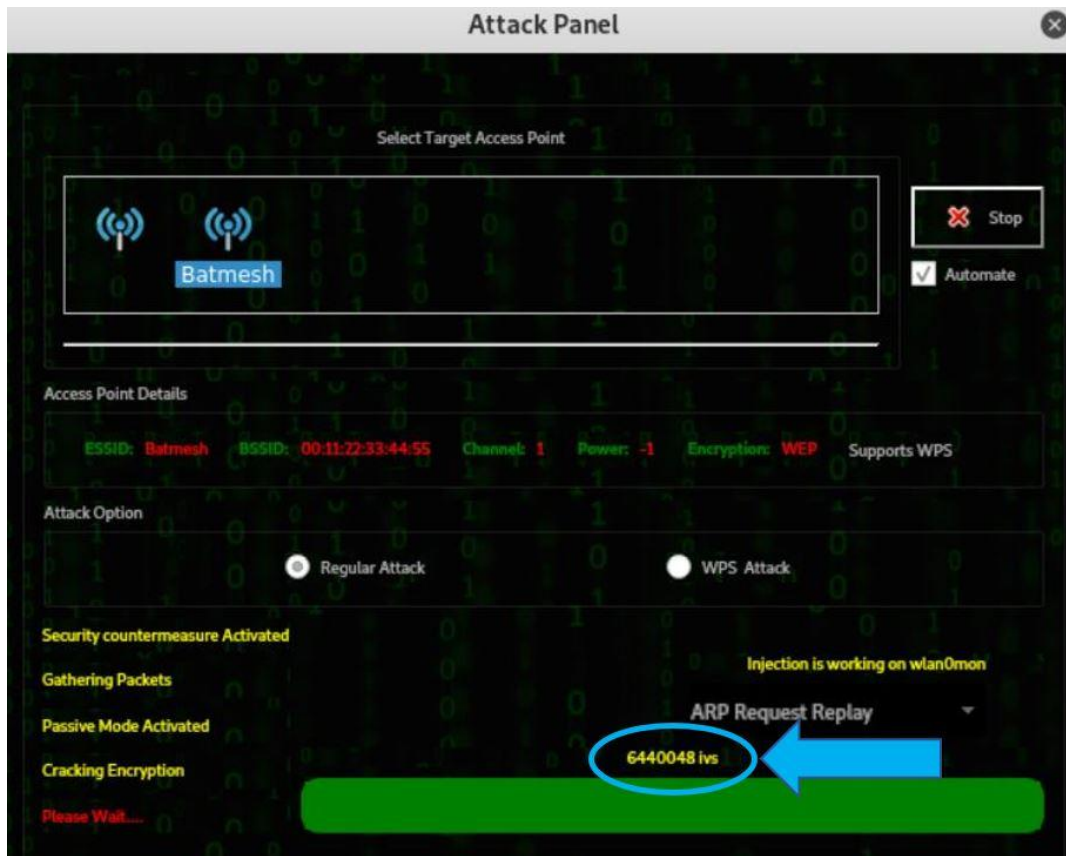


Figure 43 Fern WiFi Cracker

Conclusion

The network security takes longer to crack in a quiet network with less traffic.

Under the test conditions the WEP key wasn't cracked. This could be down to

- Using a longer 13-character key.
- An incorrect setting.
- The interfaces of the mesh being setup as ad-hoc when the software is aimed at an infrastructure-based network.

There is no doubt that WEP has been found to be weak and it would be preferable to use an alternate means to secure the network. In this scenario it has provided a level of security however which is better than none. The tests were carried out using the default settings with little experience of carrying out penetration tests.

16 APPENDIX H – USER GUIDE

AutoP2P Configuration Guide

This document is intended to provide step by step instruction on how to configure the AutoP2P network configuration tool on a Raspberry Pi Model 3B running the Raspbian operating system. All the required software for use with the Pi Car S is downloaded and installed by the install script.

If your installing the tool on an already built system, then skip straight to step 8.

Equipment Required

1 Sun Founder Pi Car-S
 1 Raspberry Pi Model 3B
 1 Micro SD card (Class 10 or above + Minimum 8GB is recommended)
 An Internet connection or offline copies of the Raspbian OS and required packages.
 1 USB to SD converter
 1 Ralink RT5370 USB adapter

Software & Packages Required

Raspbian Stretch with Desktop (Nov 2018 version used for this guide)
 Automatic P2P network tool files

Step 1 Download Raspbian

Download Raspbian Desktop from:
<https://www.raspberrypi.org/downloads/raspbian/>

Figure a below shows the details of the version used with this guide. If this version is no longer available, then legacy versions are available for download from:
<https://downloads.raspberrypi.org/raspbian/images/>

Newer versions may work but this cannot be guaranteed. It may be preferable to use a newer version if new features are required or vulnerabilities have been discovered.

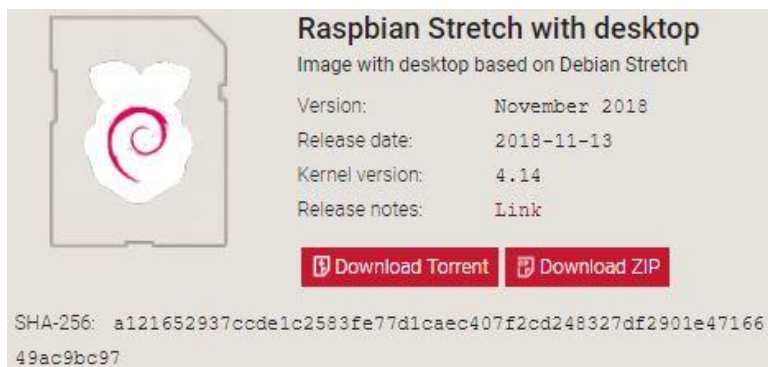


Figure 44 Raspbian Version

Step 2 Check File Integrity

The file can be downloaded using a torrent client or directly, both methods deliver a compressed file. The file should be checked for integrity using the SHA-256 hash provided on the relevant download page. To do this open terminal and change to the directory that contains the file, then run the `sha256sum` command.

```
cd /Downloads
sha256sum 2018-11-13-raspbian-stretch.zip
```

Step 3 Unzip the File

If the hashes match then the copy is good, if not download the file again as it is corrupted. The next step is to un-zip the file, this takes a few minutes to complete

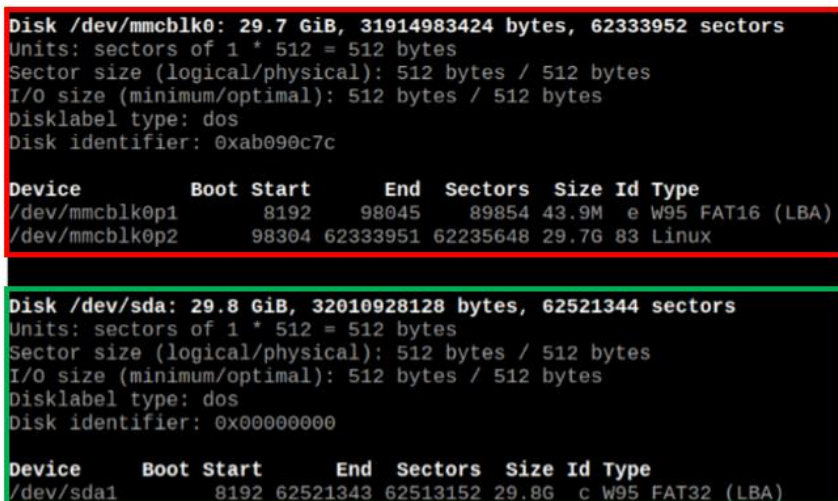
```
unzip 2018-11-13-raspbian-stretch.zip
```

Step 4 Identify the SD Card

Once un-zipped the file will be an image file, this is now ready to be written to the micro SD card. It is important at this stage to ensure that the correct drive is identified to be written too, before inserting the card enter the following command.

```
sudo fdisk -l
```

Insert the SD card into the USB converter and plug it into a USB port on the Raspberry Pi and run the exact same command again. The new entry will be the SD card, its capacity will be shown to confirm. If in doubt at this stage repeat all this step again.



```
Disk /dev/mmcblk0: 29.7 GiB, 31914983424 bytes, 62333952 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xab090c7c

Device            Boot Start      End  Sectors  Size Id Type
/dev/mmcblk0p1          8192    98045    89854  43.9M  e W95 FAT16 (LBA)
/dev/mmcblk0p2   98304 62333951 62235648  29.7G  83 Linux

Disk /dev/sda: 29.8 GiB, 32010928128 bytes, 62521344 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00000000

Device            Boot Start      End  Sectors  Size Id Type
/dev/sda1          8192 62521343 62513152  29.8G  c W95 FAT32 (LBA)
```

Figure 45 fdisk Output

In the example in figure b the output in the red box is the SD card used for the operating system of the Raspberry Pi in use and was present when entering the fdisk command for the first time.

The output in the green box is the target SD card which only appeared in the output once the SD card was inserted. This gives us the device location which in this case is `/dev/sda`.

Step 5 Write the Image to the SD Card

To write the image to the card enter the following command changing the device location to match that discovered in step 4.

```
sudo dd bs=4M if=2018-11-13-raspbian-stretch.img of=/dev/sda conv=fsync
```

Once the image has been written remove the USB converter from the Raspberry Pi.

Step 6 First Boot

Insert the newly imaged SD card into the host Raspberry Pi. Connect the Raspberry Pi to a power supply, monitor, keyboard and mouse. If a wired internet connection is being used, then ensure the cable is connected. Turn on the Raspberry Pi, the first boot normally takes longer than normal as the file system needs to expand.

If using a WiFi connection to gain internet access connect now.

Step 7 Change Password

Using the default passwords is a security risk. To change the default user password, enter the following command:

```
sudo passwd pi
```

You will then be prompted to change your password.

Step 8 Copy Files To

Insert the USB device that has the AutoP2P files into the Raspberry Pi. Drag and drop the files into the root of the pi user home drive `/home/pi/` as shown in figure c.

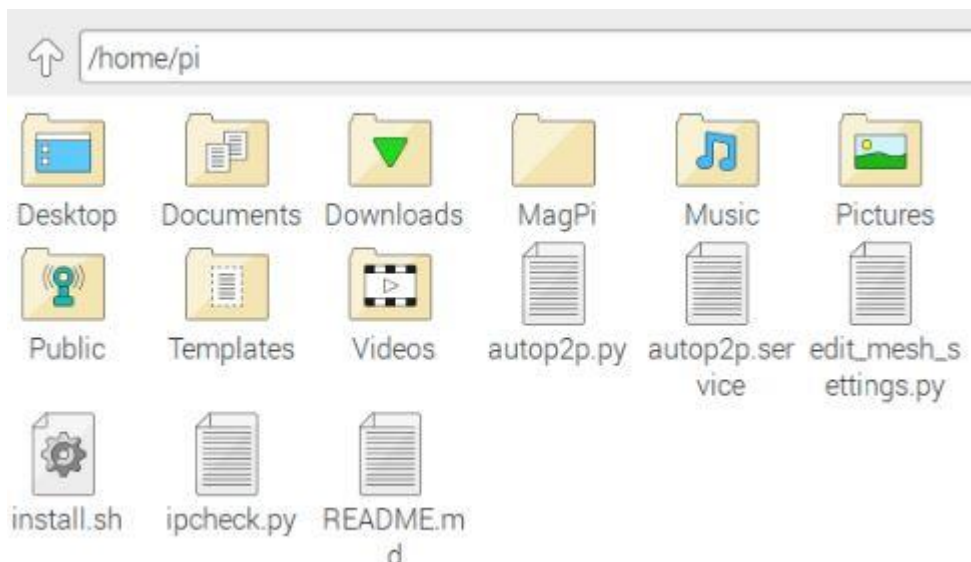


Figure 46 File Location

Step 9 Run the Install Script

Enter the following 2 commands into the terminal:

```
sudo chmod +x install.sh
sudo bash install.sh
```

During the installation a pop-up window will appear as shown below in figure d, click yes.

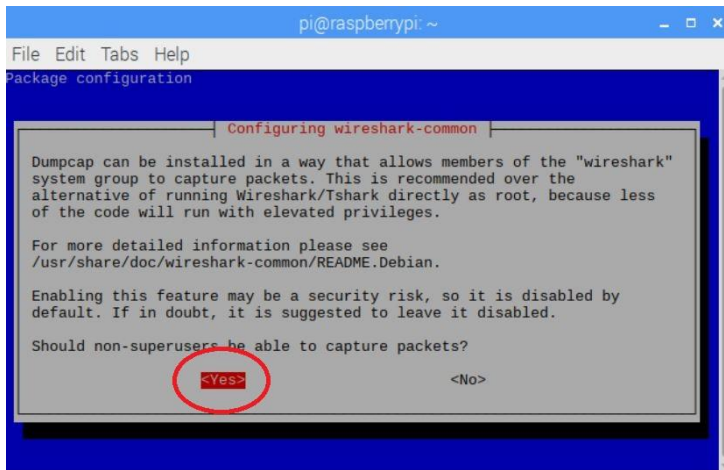


Figure 47 Wireshark Popup

Wireshark is a network protocol analyser used for network troubleshooting.

The installation will take a few minutes, once complete you will be given the option to reboot. Enter yes, once the system reboots the autop2p tool is installed and enabled.

It is now possible to connect using SSH with the following login details.

Username = pi
Password = raspberry

Useful Commands

Check the autop2p.service has started correctly

```
sudo systemctl autop2p.service
```

or

```
sudo journalctl -f -u autop2p.service
```

Check IP addressing

```
python ipcheck.py
```

Edit the mesh settings

```
python edit_mesh_settings.py
```

Check the log files

Either click on the log files to open them if directly connected or:

```
nano batlog.txt
```

and

```
nano batips.txt
```

Check B.A.T.M.A.N Adv Originator table

```
sudo batctl o
```

Check B.A.T.M.A.N Adv Neighbour table

```
sudo batctl n
```

Check which interface is being used by B.A.T.M.A.N Adv

```
sudo batctl if
```

Test basic Pi Car functionality

```
picar front-wheel-test
```

Cloning SD cards using SD Card Copier

To clone an image, connect a Raspberry Pi to a monitor along with a keyboard and mouse. Insert the SD card you want to clone into the Raspberry Pi and power it up. In the applications menu and open Accessories/SD card copier.

Connect the USB to SD card adapter with the SD card you wish to clone the image too. Identify the correct cards using the procedure outlined in step 4. Check the box that asks if new UUID's are required. Click star

17 APPENDIX I – ETHICS CHECKLIST



Research Ethics Checklist

About Your Checklist	
Reference Id	22589
Date Created	10/10/2018 19:51:03
Status	Approved
Date Approved	11/10/2018 10:08:55
Date Submitted	10/10/2018 20:24:10

Researcher Details	
Name	Jamie Murray
Faculty	Faculty of Science & Technology
Status	Undergraduate (BA, BSc)
Course	BSc (Hons) Computer Networks
Have you received external funding to support this research project?	No

Project Details	
Title	An Automatic P2P Network Configuration of Autonomous Devices
Start Date of Project	28/01/2019
End Date of Project	10/05/2019
Proposed Start Date of Data Collection	28/01/2019
Supervisor	Natalia Chechina
Approver	Natalia Chechina
Summary - no more than 500 words (including detail on background methodology, sample, outcomes, etc.)	
<p>The aim of the project is to create a software solution to establish a wireless P2P network between 2 autonomous devices without any user configuration. The project will use Raspberry Pi 3 single board computers, they are very popular low cost computing solution utilised in various different computing roles. The wireless network will not require any extra hardware other than that of the Raspberry Pi. It is expected that this solution will be created using bash and python scripting.</p>	

None of the questions apply to my study
<p>I am confirming that my proposed project does not:</p> <ul style="list-style-type: none"> • Involve human participants • Involve the use of human tissue • Involve medical research requiring NHS ethical / REC Approval

- Involve the use of animals (or tissues/fluids derived from animals)
- Involve access to identifiable personal data for living individuals not already in the public domain
- Involve increased danger of physical or psychological harm for researcher(s) or subject(s)
- Raise any ethical issues associated with the use of genetically modified organisms

On this basis, my proposed project does not require a formal ethics review.

If any changes to the project involve any of the criteria above, I undertake to resubmit the project for formal ethical approval.

18 APPENDIX J – USB AND SD CARD CONTENTS

A USB drive can be found attached to the report. It contains an electronic version of this report:

An Automatic Peer to Peer Network Configuration of Autonomous Devices.doc

It also contains a folder called autop2p_tool containing the 6 project artefact files, which are:

README.txt

autop2p.py

autop2p.service

ipcheck.py

edit_mesh_settings.py

install.sh

A micro SD card is also attached, this is stock Raspbian, updated and with the project artefact pre-installed. This can be cloned using the Accessories/SD Card Copier software that comes built into Raspbian.

The script requires an RT5370 network adapter that comes as part of the Pi Car S kit or an adapter with the same capabilities to accommodate an increase in MTU.

Username: pi

Password: raspberry