Investigation Into Building Large Scale Wireless Networks with Ubiquitous Access

A report
submitted in fulfillment
of the requirements for the degree
of
Master of Science in Computer Science
at
The University of Waikato

by

Scott Yearbury

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Department of Computer Science
Hamilton, New Zealand
November 8, 2020

Abstract

This thesis presents a design and implementation of a novel method of device authentication in Institute of Electrical and Electronics Engineers (IEEE) 802.11 Wi-Fi networks and a method of access point configuration and management. The novel method of authentication provides a mix of properties from both Wi-Fi protected access (WPA) - Personal and WPA-Enterprise. It allows for per user authentication and allows for device ownership to be tracked like WPA-Enterprise while providing the ease of use of and ubiquitous support of WPA-Personal. It also allows for a network covering a wide area to use a single Service Set Identifier (SSID) and for seamless authentication and roaming between access points. This project also provides a highly scalable method of centralised access point configuration and management based upon the OpenSync access point management software and a custom OpenSync controller. The access point configuration and management and Wi-Fi authentication methods provided are designed so that it supports this project being extended to provide user based network segmentation such that each user appears to have their own virtual layer 2 network as they roam between access points.

Acknowledgements

I would like to thank everyone in WAND who helped me in any way with this thesis, in particular Brad Cowie and my supervisor Dr Richard Nelson.

I would also like to thank my family, in particular my parents, for their support throughout my entire time at university.

Contents

List of Figures

List of Acronyms

AC Access Controller

AP Access Point

CAPWAP Control And Provisioning of Wireless Access Points

DHCP Dynamic Host Configuration Protocol

DTLS Datagram Transport Layer Security

EAP Extensible Authentication Protocol

GTK Group Temporal Key

IANA Internet Assigned Numbers Authority

IEEE Institute of Electrical and Electronics Engineers

IP Internet Protocol

IPSK Identity Pre-Shared Key

JSON JavaScript Object Notation

JSON-RPC JSON - Remote Procedure Call

LAN Local Area Network

MAC Medium Access Control

MIC Message Integrity Check

NAS Network Access Server

OSI Open Systems Interconnection

OVSDB Open vSwitch Database

OVS Open vSwitch

PMK Pairwise Master Key

PSK Pre-Shared Key

PTK Pairwise Transient Key

RADIUS Remote Authentication Dial In User Service

RFC Request for Comments

SDN Software Defined Networking

SSH Secure Shell

SSID Service Set Identifier

TCP Transmission Control Protocol

UCI Unified Configuration Interface

UDP User Datagram Protocol

VAP Virtual Access Point

WLAN Wireless Local Area Network

WPA Wi-Fi Protected Access

WTP Wireless Termination Point

Chapter 1

Introduction

1.1 Overview

1.1.1 The Problem

Over the past decade, Internet access has become ubiquitous and essential to life in the modern world. During this time Institute of Electrical and Electronics Engineers (IEEE) 802.11 Wi-Fi [13] and GSM and CDMA based cellular networks have become the dominant methods for wirelessly connecting to the Internet. In home environments Wi-Fi is generally the preferred option, when available, due to its usually faster speeds [22], lower data costs [20] and because it allows people to access their Local Area Networks (LANs). However, when out of the home, people often use slower, more expensive cellular data connections. This can be for a range of reasons including Wi-Fi not being available where they are, an unwillingness to constantly connect their devices to new Wi-Fi networks or an unwillingness to trust an unknown network. Cellular networks don't suffer from these same problems. Unlike Wi-Fi, people's devices usually only connect to a single cellular network that covers almost everywhere they go. Their devices then automatically authenticate and connect to new cellular towers that are part of the same network, as they go out of range of previous ones to ensure they stay connected.

The goal of this project is to develop a Wi-Fi network which has properties of both Wi-Fi and cellular networks. A singular Wi-Fi network would be deployed that would cover a wide area such as an apartment building and the suburb around it. Users would then connect their devices to this single network and the devices would seamlessly roam between Access Points (APs)

to stay connected. This would allow devices to always be connected to a trusted network with no ongoing effort required from the user.

Deploying a single network over a large area also provided other opportunities that this project makes use of. When a user leaves their traditional home Wi-Fi network, they leave behind their home LAN, meaning they can no longer access other devices and resources on it. There are ways to overcome this and access devices and resources on a LAN over the Internet but these are often complex to set up. This project allows for a single network that enables users to access their own home LAN wherever they connect to the network. This allows them to access any devices and resources on their home LAN from outside their home with no more difficulty then when they are in their home. Another benefit that comes from deploying a single network over a wide area is that it would be centrally managed. This is in contrast to how Wi-Fi is usually deployed today which can be problematic, especially in high density housing environments. In apartment blocks it is common for the residents of every apartment to have their own Internet connection and AP broadcasting their own private Wi-Fi network. If APs are not configured correctly, many of them can be left operating on the same or neighbouring channels causing interference and degraded performance. Due to the technical nature of this problem many users aren't even aware it exists, let alone how to fix it.

There are many problems that must be overcome to be able to implement this single large network that is deployed over a wide area and provides a good user experience. Having a single network without any network segmentation, which everyone connects their devices to, would both have a poor user experience and have massive security issues. It would allow any device to talk to any other device, and there would be no way to prevent malicious actors from accessing the network. This could have consequences ranging from annoying to serious. For example, devices that advertise themselves on the LAN using multicast packets, such as Google Chomecasts, would able to be seen and controlled by everyone on the network. A more serious example would be malicious users being able to access devices such as unsecured security cameras over the network. A common solution to this problem, often used on public networks, is to prevent devices on the network from being able to talk to each other and only to allow traffic to and from the Internet. This can be suitable for networks such as corporate networks but would not be suitable for someone's home as it would prevent them from being able to use local network resources such as network printers or Google Chromecasts.

Providing a single network that everyone connects to and that provides no per-user authentication would also cause issues. There would be no way to control who has access to the network and no way to stop someone from being able to access the network. There are existing ways the network could be secured but each of these has its own limitations. Using Wi-Fi Protected Access (WPA) Personal [13] would secure the network with a single pre-shared key (PSK) that every user would use. This PSK would have to be known by everyone who should have access to the network, whilst not being shared with anyone who should not have access to the network. The only way to remove an existing user from the network would be to change the PSK, which would require all users to enter the new PSK in their devices, and hope the person is not able to find out the new PSK. This makes WPA-Personal or any method of authentication that does not provide per-user authentication not at all suitable for this use. There are other methods of authentication to Wi-Fi networks that do provide per-user authentication, such as WPA-Enterprise [13] and using a captive portal. These, however, have limited support by devices and can be difficult for users to use also making them impractical for a network which everyone is supposed to connect all their devices.

Having a single Wi-Fi network cover a larger area requires the deployment and management of many APs. There are many different existing solutions to manage large deployments of APs, although the majority of these are proprietary and not open source.

As outlined above, there are three main issues that need to be solved to make the use of a single Wi-Fi network that covers a wide area and has many different people connecting their devices to it practicable. These are as below.

- The network needs to be segmented such that each user appears to have their own private network, like they would if they were connected to their own traditional home Wi-Fi network.
- The network would need an authentication method that is easy to use, widely supported by all types of devices and provides a way for individual user's access to the network to be controlled.
- There would need to be a method of centralised AP configuration and management suitable for the large number of APs that would be required to cover a large area.

1.1.2 Contribution

This thesis proposes a design for a wide area Wi-Fi network that meets two of the three requirements outlined above. It meets the requirement that it provides an easy to use authentication method that is suitable for a wide area Wi-Fi network. It also meets the requirement that it provides a method of centralised AP configuration and management. It also partially meets the requirement to segment networks such that each user appears to have their own private network.

It provides a novel method of authenticating devices that is based upon WPA-Personal, making it compatible with virtually all devices that support Wi-Fi. The authentication does this while being transparent to the user so that it appears to be standard WPA-Personal making it very simple to use. The authentication method also generates a database of devices and the owners of those devices. This is important to allow the requirement of segmenting the network based on device ownership to be met.

This thesis also provides a solution for the requirement to provide a method of AP centralised configuration and management suitable for the large number of APs. It does this by building on the open source OpenSync management software.

1.2 Document Structure

Chapter 2 outlines the requirements for authentication on the network, investigates the options available and selects one for use.

Chapter 3 outlines the requirements for AP configuration and management, investigates the options available and selects one for use.

Chapter 4 gives a detailed description of the components chosen for this and how they work together to meet the goals outlined in Chapters 2 and 3.

Chapter 5 explains how this project was tested to ensure that the goals outlined in chapters 2 and 3 are met.

Chapter 6 evaluates the project to see if it has met its goals. It also discusses the challenges encountered while developing this project and the future work that could be done to expand on it.

Chapter 7 is a conclusion, providing a summary of the work done in this project and the outcomes of it.

Chapter 2

Investigation: Authentication

This chapter outlines the requirements that need to be met for an authentication method to be suitable to meet the goals of this project. It then discusses the options that were looked at and selects the most suitable option.

2.1 Requirements

The requirements of an authentication method to be suitable for this project are outlined below.

- The authentication method must use a secure and widely supported standard so that virtually all Wi-Fi enabled devices can be authenticated with it.
- The authentication method must also provide a way to identify who the owner of each device is in order to allow devices to be segmented onto the correct virtual layer 2 network.
- The authentication method must be simple enough that a person without technical skills can connect new devices.
- The authentication method must allow devices to connect and disconnect as well as roam between APs without any actions required from the user and while still being able to be identified.

2.2 Options

This section discusses each of the options that were looked at for authentication for this project. It discusses if and how they meet the requirements for authentication and their overall suitability for this project.

2.2.1 Captive Portal

In a captive portal system, devices connect to an open Wi-Fi network on which all non-web traffic is blocked. All web traffic is redirected to a single web page until a user goes to that web page and authenticates the device, usually by logging in with a username and password. Once the device has been authenticated, the device is given access to the network [19].

Captive portal based authentication is not, unlike the other authentication methods looked at, a specific authentication protocol that is required to be supported by devices. Instead it only requires a device to be able to browse to and interact with a website. This is, however, in itself a problem that causes it to not be able to meet the requirements. It is not possible on all devices to browse to and interact with a website, especially many Internet of Things devices which do not even have screens, for example, many Google Nest devices [39]. It also fails to meet the requirement to be simple enough that a person without technical skills can connect new devices. This is because if the user attempts to browse to a website that uses HTTPS, when their request is redirected, their browser might detect this as a man in the middle attack and not allow it to happen [15]. This will likely leave the user confused by the warning their browser is showing and unable to connect.

This method of authentication does meet the requirements to be able to identify the owner of a device. When the user authenticates their device on the web page, their devices Medium Access Control (MAC) address could be saved to a database along with who authenticated the device. This database could also be used to help meet the goal of allowing devices to seamlessly roam between APs as it could be used to identify already authenticated devices.

Using a captive portal would only meet two of the four requirements for an authentication method to be used in this project and so isn't suitable to be used.

2.2.2 Wi-Fi Protected Access

There are three versions of Wi-Fi Protected Access (WPA), WPA, WPA2 and WPA3 [32][30][3], this project uses WPA2. WPA3 is the latest version of WPA but it is not used in this project because many devices still do not support it and widespread support is one of the requirements for authentication in this project [36]. WPA can operate in two different modes [33]. The most common mode is WPA-Personal in which authentication is based around a Pre-Shared Key (PSK) known by both the connecting device, known as the station, and AP. The other mode is WPA-Enterprise which utilises 802.1x [21] and Extensible Authentication Protocol (EAP) [4] to provide per user authentication.

The process by which devices connect to a Wi-Fi network secured by WPA2 is shown in figure 2.1. The first step is for the station to discover the AP, or vice versa. This can be done either through passive or active scanning. When using passive scanning the station listens for beacon frames that are broadcast periodically by APs. These frames contain details of the Wi-Fi network being broadcast by the AP such as the Service Set Identifier (SSID) and supported protocols. Alternatively, a device can broadcast a probe request frame, which contains information such as supported data rates and protocols. The device then listens for a probe response frame from an AP. These frames are sent by APs and contain the SSID of the network among other information. They are only sent by the AP if the station is compatible with the network being broadcast by the AP, which is determined using information provided in the probe request.

Once the devices have discovered each other, the WPA four-way handshake begins. The four-way handshake consists of four messages being sent between the AP and station. The first of these messages contains the ANonce which is a random value generated by the AP and is sent to the station. The station then uses the ANonce value along with the SNonce it generates itself, its MAC address, the APs MAC address and the Pairwise Master Key (PMK) to generate the Pairwise Temporal Key (PTK). When using WPA-Personal the PMK which is generated from the PSK using a SHA-1 hash, when using WPA-Enterprise the PMK is generated by the EAP authentication process. The station then replies with the SNonce it generated and used to generate the PTK, this message also includes the Message Integrity Check (MIC), allowing the AP to detect if the message has been corrupted or interfered with. Once

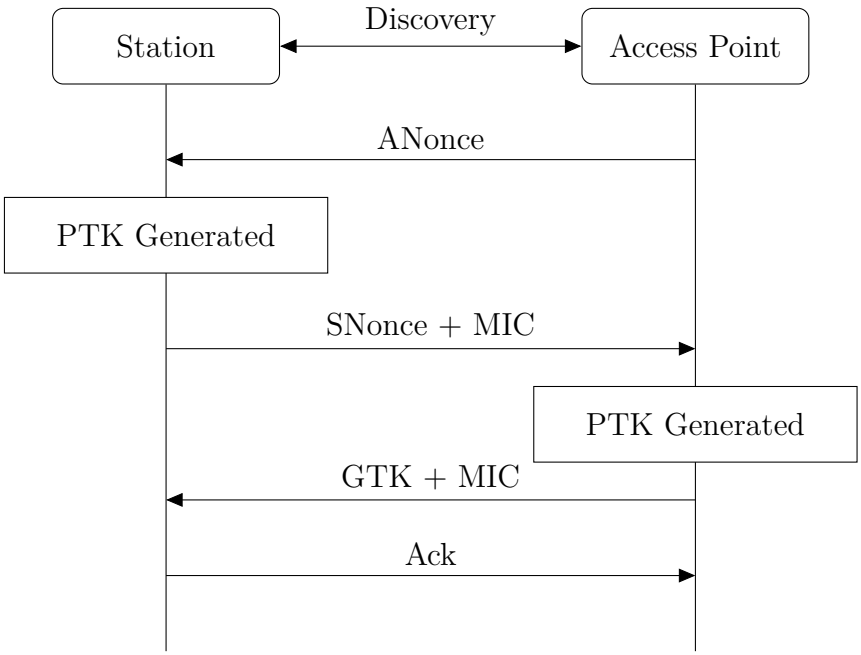

Figure 2.1: WPA 4-way Handshake

the AP has received this message, it now has enough information to, and does, generate the PTK using the same method as the station. At this point both the AP and station have the PTK and use this to encrypt all unicast traffic between the two devices from then on. The AP then sends the Group Temporal Key (GTK) which is generated by the AP and is used by the AP to encrypt all multicast traffic it sends. This message is encrypted by the PTK and has a MIC to protect the GTK from being intercepted by any listening devices to ensure that it is only known by devices that are connected to the Wi-Fi network. The final message sent in the four-way handshake is the an acknowledgement sent from the station to the AP to confirm that the new keys have been installed. To prevent SSID spoofing attacks being able to discover the PSK from devices attempting to connect to a spoofed network, the PSK is never sent between the station and the AP. Instead it is used in generating the PMK and PTK ensuring that both devices have the same PSK, otherwise the four-way handshake is not able to be completed.

WPA-Personal

WPA-Personal is the simpler mode of WPA and is the most widely used mode. It is based around a PSK which the AP is configured with and is also entered into any device connecting to the network. A SHA-1 hash is then used to

convert the PSK into the PMK for use in the four-way handshake shown in figure 2.1.

WPA-Personal meets the requirements of being widely supported by virtually all devices. It is also a very simple method of authentication that virtually anyone could use and allows for seamless roaming between APs. It doesn't, however, provide per user authentication as every user has to enter the same PSK, this makes it unsuitable for this project. This is because without per user authentication there is no way to identify device owners to segment the network nor is there a way to add and remove a user's access to the network.

WPA-Enterprise

WPA-Enterprise is a mode of WPA which allows the use of IEEE 802.1x [21] authentication to authenticate devices connecting to a Wi-Fi network. The 802.1x protocol is used to provide port-based network access control on local area networks through the use of the EAP [4]. When a device connects to a Wi-Fi network that is secured with WPA-Enterprise, it is first authenticated with 802.1x authentication. In 802.1x authentication there are three devices: the supplicant, which is the connecting device/station and is being authenticated, the authenticator, which is the AP when using WPA-Enterprise, and the authentication server which the authenticator uses to request the requirements for connecting deices from, usually a Remote Authentication Dial In User Service (RADIUS) server.

When a device connects to an IEEE 802.11 Wi-Fi network secured using WPA-Enterprise first authentication using EAP occurs, then a WPA four-way handshake occurs. This is shown in figure 2.2 When the supplicant first connects to the authenticator all non-EAP traffic is blocked. The authenticator sends an EAP-Request Identity frame to the supplicant. The supplicant then replies to the authenticator with an EAP-Response Identity frame. That response is then sent to the authentication server in a RADIUS Access-Request packet. The authentication server then uses this information to begin EAP based authentication, through the authenticator, with the supplicant. If the EAP authentication is successful, the authentication server will send the generated PMK to the authenticator. The PMK is then used in the WPA four-way handshake which validates the PMK and generates keys to be used to encrypt the Wi-Fi connection.

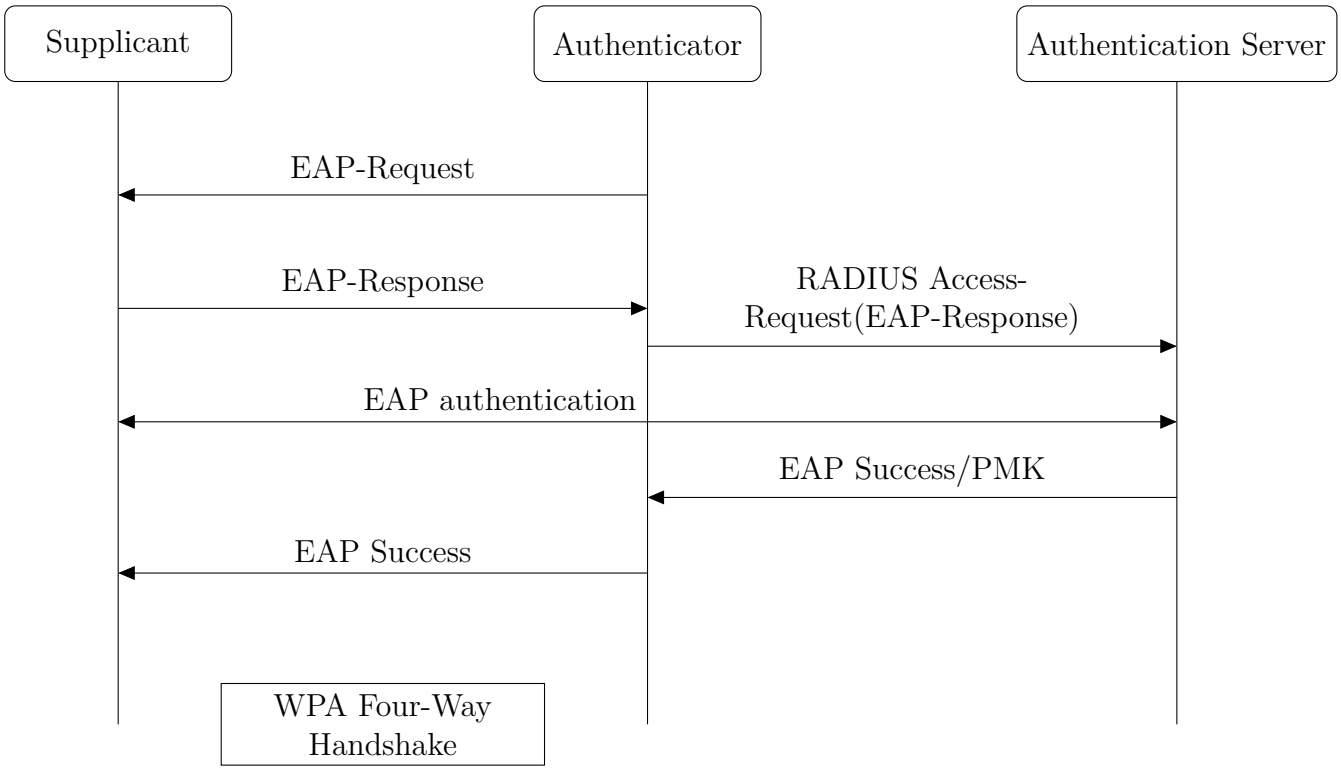

Figure 2.2: 802.1X Authentication in WPA-Enterprise

WPA-Enterprise is quite widely supported but there are some common devices, such as the the Microsoft Xbox One, which don't support it [37]. This means it fails the requirement for it to be a widely supported standard so that virtually all Wi-Fi enabled devices can be authenticated with it. It does meet the requirements to provide per user authentication and allow for seamless roaming between APs. Whether it meets the requirement to be simple enough that a person without technical skills could connect new devices is not entirely straightforward. Depending on the type of EAP authentication used, it can either be relatively straightforward to use or quite complex. However, even if WPA-Enterprise can be configured to be simple to use, it still fails the requirement to be supported by virtually all devices and so is not suitable for this project.

Identity PSK

Identity PSK (IPSK) [10][17], unlike the other WPA based authentication options, is not a single standard but instead a modified version of WPA-Personal that adds more features. It is implemented by many different vendors and with a variety of different names.

IPSK operates like WPA-Personal, but it doesn't require there to be a single PSK which all devices must use to access the network. Instead there is a RADIUS server with a database which stores MAC addresses and PSK pairings for devices. When a device first associates with an AP, the AP sends a RADIUS Access-Request message to the RADIUS server which either replies with an Access-Accept message, containing the PSK for that device, or an Access-Reject message. If the AP receives an Access-Accept message, it uses the PSK returned in that message when performing the four-way handshake. If the AP receives an Access-Reject, depending on the implementation being used can either be configured to use a default PSK configured on the device or to just block the connection. IPSKs operation is shown in figure 2.3.

IPSK provides features from both WPA-Personal and WPA-Enterprise. It is secure and is supported by any device that supports WPA-Personal, which is virtually all Wi-Fi enabled devices. This is because, as can be seen when comparing figures 2.1 and 2.3, from the point of view of the station, there is no difference to connecting to a network secured with WPA-Personal or IPSK. IPSK also provides a way to identify devices and their owners, this is because it requires a database of every device and its PSK to be able to function. This same database can be used to identify the owner of a device and then that ownership information can be used to segment the network. IPSKs main limitation is that it doesn't provide any way to automatically generate the database of MAC addresses and PSKs. It would be beyond the knowledge of many people to be able to identify the MAC address of all their devices and to then enter that into a database through a portal or some other method. This means that IPSK doesn't meet the requirement to be simple enough that a person without technical skills can connect new devices. If all APs use the same database to lookup device's PSKs when they connect, IPSK can provide seamless roaming between APs. IPSK meets three of the four requirements for authentication in this project, with only its requirement of having a database of device MAC addresses preventing it from being suitable to be used as the authentication method in this project.

Identity PSK with a Default PSK

So far IPSK has come the closest of all the authentication methods discussed to meeting the requirements of authentication in this project. It doesn't, however, provide a simple way for new devices to be onboarded on to the Wi-Fi network.

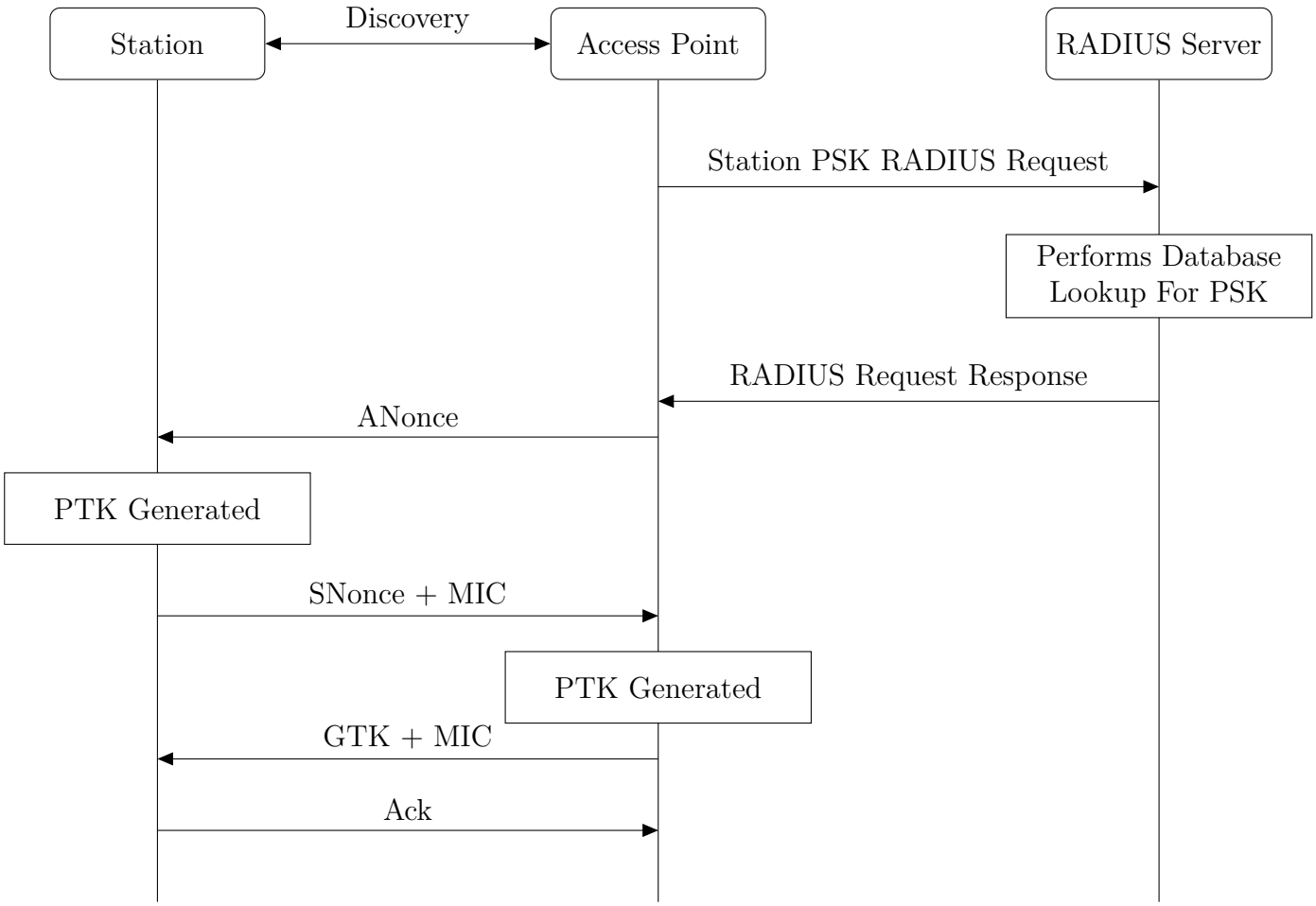

Figure 2.3: WPA 4-way Handshake when using IPSK

The solution to this is to use IPSK with a default PSK, an authentication method which has been developed in this project, and which provides seamless and transparent onboarding on new devices. It is implemented using IPSK and a custom RADIUS server.

When using IPSK with a default PSK, an AP would have to be deployed in every home or apartment. Users would connect their devices to the Wi-Fi network for the first time from their home or apartment so that it connects through their home AP. The AP that a device is first connected to is used to identify the ownership of the device. When a device connects to the Wi-Fi network, the AP will perform standard IPSK authentication and will make a RADIUS request to the custom RADIUS server. This RADIUS request contains the MAC address of the station and information to identify the AP the device is connecting through. How the custom RADIUS server handles authentication requests is shown in figure 2.4. When the RADIUS server

receives a request it makes a query to the device database to find the PSK associated with the device's MAC address. If a PSK is returned, the RADIUS server sends an Access-Accept RADIUS packet with the PSK in an attribute field. If, when the controller makes the query to the database, no entry in the database matches, the RADIUS server makes a second query. This second query uses the details of the AP that are included in the RADIUS Access-Request sent by the AP. If a PSK is returned from this second query, the details of the device that connected to the AP along with the PSK retrieved in the second query are added to the database and then an Access-Accept RADIUS packet, with the PSK as an attribute, is sent to the AP and the device is able to authenticate. If the second query also doesn't return a PSK, then an Access-Reject RADIUS packet is sent to the AP, and the device will not be authenticated. If the AP receives an Access-Accept message from the RADIUS server, the PSK in that message is used by the AP to generate the PMK for the four-way handshake. If the PSK entered in the device by the user matches the PSK retrieved from the database and used by the AP, the device will be authenticated and can be connected to the Wi-Fi network. If the PSKs do not match, the authentication will fail, and the device will not be able to connect to the Wi-Fi network.

This method of authentication meets all the requirements of authentication in this project. Being based around IPSK means that it is both secure and only requires devices to support WPA-Personal which supported by virtually all Wi-Fi enabled devices. This means it meets the requirement to use a secure and widely supported standard so that virtually all Wi-Fi enabled devices can be authenticated with it and also that the authentication must be simple enough that a person without technical skills can connect new devices. The custom RADIUS server creates a database of devices and PSKs when devices are first authenticated to the Wi-Fi, this can be used to identify device owners to allow the network to be segmented. This means it meets the requirement to provide a way to identify who the owner of each device is to allow devices to be segmented onto the correct virtual layer 2 network. Once a device has been registered in the database, it can then roam, using the same PSK, between any AP that shares the same RADIUS server, and therefore database of devices, with the AP it was first connected to. This means that it meets the requirement that it allow devices to connect and disconnect as well as roam between APs without any actions required from the user and while still being able to be identified.

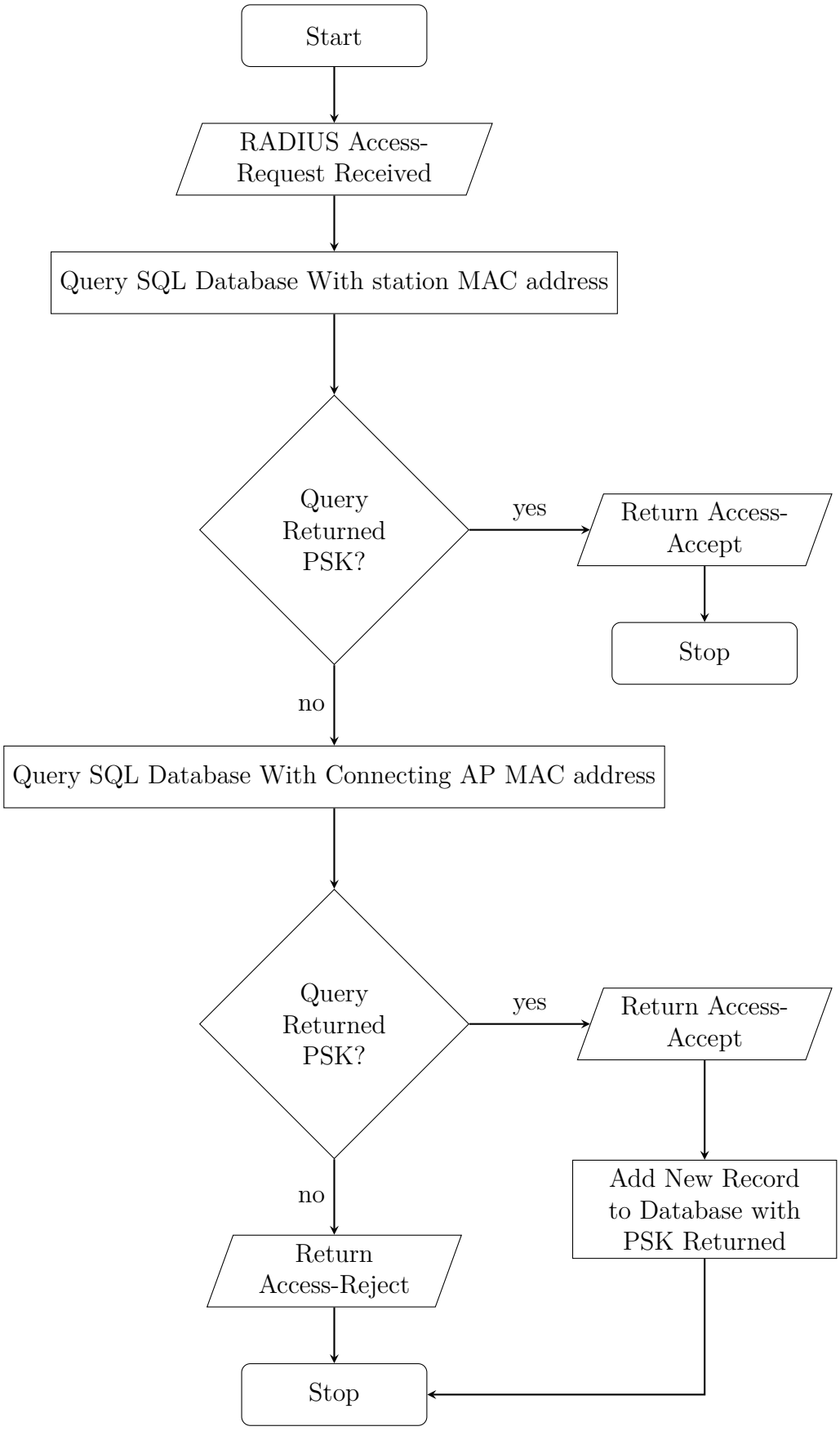

Figure 2.4: Custom RADIUS Authentication Server Flowchart

2.3 Conclusion

This chapter has outlined the requirements for an authentication method to be suitable for this project. It has also investigated different authentication methods to see if they meet the requirements of the project. None of the preexisting authentication methods investigated met the requirements for this project but IPSK with a Default PSK does meet the requirements and so this was chosen for use with this project.

Chapter 3

Investigation: Access Point Configuration and Management

This chapter outlines the requirements that need to be met for an AP configuration and management method to be suitable for this project. It discusses the options that were looked at for this project and selects the most suitable option.

Most AP vendors also have their own AP configuration and management solutions, such as Aruba Network's Mobility Controllers [18] and Cisco's Wireless LAN Controllers [35]. These vendor AP configuration and management solutions are, however, usually proprietary and aren't publicly documented sufficiently to be able to be usefully examined in detail or considered for this project.

3.1 Requirements

The requirements of an AP configuration method to be suitable for this project are outlined below.

- The configuration and management of APs must be easily scalable to make it practical to use with a large number of APs.
- The configuration and management of APs must support the method of authentication selected in chapter 2.
- The configuration and management of APs must suitable for use in a real world deployment.

- The software on the AP must be modifiable to allow custom features to be added.

3.2 Options

There are two broad approaches of AP configuration and management: centralised controller based methods and decentralised controllerless methods. They are differentiated by their either centralised or decentralised management, control and data planes.

3.2.1 Centralised Controller Based Access Point Configuration and Management

The centralised controller AP configuration and management methods discussed in this section use centralised control and data planes. This means they both have a central controller which all network traffic goes through and which handles all configuration. Having a central controller which all network traffic goes through has the advantage of making implementing the network segmentation for this project simpler. However, it has the disadvantage that the central controller also has to be able to handle all network traffic, which can make scaling difficult [6].

CAPWAP

Control And Provisioning of Wireless Access Points (CAPWAP) is a protocol outlined in the Internet Engineering Task Force Requesting For Comment (RFC) 5415 [8] that allows for the centralised control of wireless networks. RFC 5416 [7] outlines the usage of CAPWAP with IEEE 802.11 Wireless Local Area Networks (WLANs). CAPWAP provides centralised control and data planes. The centralised control plane, however, doesn't support all the features required of an APs control plane and so some of the control plane must be implemented outside of CAPWAP. The central controller in a CAPWAP deployment is called an Access Controller (AC) and the APs are called Wireless Termination Points (WTPs).

A CAPWAP deployment consists of an AC that manages a collection of WTPs, the WTPs act as remote radio frequency interfaces that are controlled by the AC. The CAPWAP protocol is designed to be a vendor agnostic protocol for

communication between ACs and WTPs, it provides an alternative to the configuration options usually provided for 802.11 WLAN APs which is often proprietary, manual and static.

CAPWAP provides two types of messages that can be sent between the AC and WTPs, CAPWAP Control messages and CAPWAP Data messages. CAPWAP Control messages consist of seven different message types that are used by WTPs and ACs to provide a control channel between the WTPs and ACs. CAPWAP Data messages are used to encapsulate wireless frames that have been received by a WTP to send it to the AC and wireless frames that are being sent from the AC to a WTP. The data messages are sent in CAPWAP protocol Data Payload packets, these can either be sent over a User Datagram Protocol (UDP) connection or an encrypted Datagram Transport Layer Security (DTLS) [5] connection. The control messages must be sent using an encrypted DTLS connection. There are two different ways CAPWAP can operate, in split MAC mode and local MAC mode. In split MAC mode, the 802.11 wireless frames are directly encapsulated in a CAPWAP protocol Data Payload packet. In local MAC mode, the WTP performs an 802.11 integration function which converts the frames to 802.3 frames, these are then encapsulated in a CAPWAP protocol Data Payload packet.

How the IEEE 802.11 WLAN functionality is split between the WTPs and AC depends on whether they are running in split MAC or local MAC mode. When running in split MAC mode, the real-time 802.11 tasks, such as the generation of beacon packets and responding to probe requests, are handled on the WTP. The distribution and integration services allow for the delivery of MAC service data units, which contain layer 3 and above in the Open Systems Interconnection (OSI) model, between devices on the 802.11 LAN and non-802.11 LANs respectively. Association, disassociation and re-association of devices to the 802.11 LAN, 802.11 encryption and decryption and the handling of all other 802.11 MAC management frames is done by the AC.

When running in local MAC mode, the distribution service can run on either the WTP or AC. Association, disassociation and re-association of devices is handled by the WTP. To keep the AC aware of mobility events, the AC is forwarded 802.11 association request frames, and it can reply with a failed association response frame. This will cause the WTP to send a disassociation frame to the device. All other IEEE 802.11 WLAN functionality is handled

by the WTP.

CAPWAP stores the maximum number of WTPs that a single AC supports on a 16 bit number in the AC descriptor CAPWAP protocol message, this means CAPWAP would theoretically be limited to 65 535 WTPs per AC. However, in a real world deployment the AC would actually be limited by its performance. This because of work load put on the AC by CAPWAPs centralised control and data planes. This could be mitigated by using CAPWAP in Local MAC mode which would allow an AC to be able to handle more WTPs and traffic on the network but wouldn't get around the fundamental issues with trying to scale a network with centralised control and data planes. Because of this, CAPWAP does not meet the requirement that it be easily scalable to make it practical to use with a large number of APs.

CAPWAP is limited as it only supports the IEEE 802.11-2007 specification and not any newer standards such as 802.11n, 802.11k and 802.11r. It does not support IPSK and so doesn't meet the requirement to be compatible with the authentication method chosen in chapter 2. This lack of support for newer protocols has reduced CAPWAPs adoption but despite this it is still used is real world deployments and so it meets that requirement. The lack of support for newer protocols has, however, prevented there from being well developed open-source implementations, which means CAPWAP doesn't meet the requirement to be modifiable to allow custom features to be added.

CloudMAC

CloudMAC [11] provides a novel architecture for enterprise or carrier grade WLAN deployments. CloudMAC splits the functionality of a traditional AP between a WTP and a Virtual AP (VAP) connecting them together with an OpenFlow switch. CloudMAC provides a very centralised method of AP configuration and management. This is because the management, control and data planes are all contained in the VAPs.

Each VAP is a Virtual Machine (VM) with one or more virtual WLAN cards. This is shown in figure 3.1. The WLAN cards are implemented with a custom driver that appears to the operating system as a normal WLAN card installed on the system. Because the virtual WLAN cards appear as normal WLAN cards, standard AP software such as hostapd can be run on the VAPs. WTPs are APs that allow for the forwarding of raw 802.11 MAC frames between the

wired and wireless interfaces. The virtual WLAN cards generate 802.11 MAC frames for transmission as it would in a non virtualised environment. These frames then have an additional control header added to them which contains information about the modulation and coding scheme and transmission power. The frame can optionally be encrypted and are then sent through a layer 2 tunnel and OpenFlow switch. The layer 2 tunnels are used to tunnel 802.11 MAC frames between the VAPs and WTPs. When a WTP receives a MAC frame from the VAP, it reads the required information from the control header and then transmits the frame through its WLAN interface. The matching of VAPs and WTPs is done by the OpenFlow switch, this allows for a central OpenFlow controller to manage the pairing of VAPs to WTPs and for multiple VAPs to be bound to a single WTP. The OpenFlow controller can also provide other centralised control such as blocking devices from being able to connect to the network or to rate limit specific devices. When the virtual WLAN card receives a configuration command from an application running on the VAP a configuration command packet is generated and sent to the WTPs. This packet is received by a control application running on each WTP which applies the configuration.

CloudMAC has the same problems as other centralised AP configuration and management solutions. It is difficult for deployments with centralised control and data planes to be scaled up because of the work load this puts on the central controller. To mitigate this, many VAPs could be deployed accross multiple VM hosts. This would also be problematic as CloudMAC doesn't provide a solution for a centralised management plane which would be required to make a deployment with many VAPs practical. Because of this, CloudMAC doesn't meet the requirement to be scalable and practical to use with a large number of APs. It provides flexibility as to what runs on the VAPs meaning it can support virtually any authentication method including IPSK. This means it meets the requirement to be compatible with the authentication method chosen in chapter 2. Almost any software can be run on the VAPs, including custom software, this means that CloudMAC meets the requirement be modifiable to allow custom features to be added. CloudMAC has never been developed beyond a prototype stage and so doesn't meet the requirement to be suitable for a real world deployment.

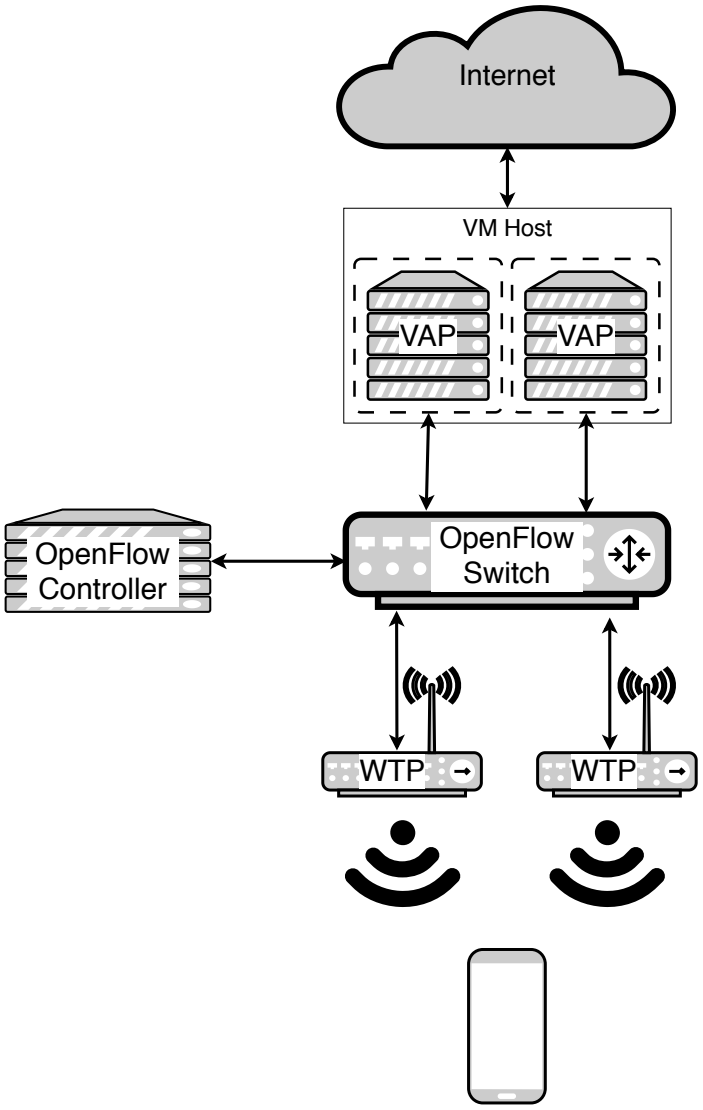

Figure 3.1: The CloudMAC Architecture

3.2.2 Decentralised Controllerless Access Point Configuration and Management

The decentralised controllerless AP configuration and management methods discussed in this section use decentralised control and data planes. This means each device has its own control and data plane that is configured through the management plane. The management plane can either be centralised or decentralised. Having each AP have its own control and data plane means that the number of APs can be scaled up without having to worry about performance limitations of a central controller. Large scale deployments of APs with decentralised control and data planes can be impractical to manage without some form of centralised control, this can be solved through the use of a centralised management plane[6].

OpenWrt with the UCI System

OpenWrt [26] uses the OpenWrt Unified Configuration Interface (UCI) system [25] for configuration. It provides a decentralised management, control and data plane. As with configuring many APs, the UCI system is used through the APs shell, usually access via Secure Shell (SSH).

There are two ways the UCI system can be used to configure OpenWrt. The first is through direct modification of files in the /etc/config directory and the second is through the UCI command line utility. The files in the /etc/-config directory each broadly relate to different parts of the system. For example /etc/config/dhcp configures the Dynamic Host Configuration Protocol (DHCP) [1] server running on the system and /etc/config/wireless configures wireless networks being broadcast by the system. Any changes made in the file are applied when the affected programs are started using the initialisation scripts in /etc/init.d/. APs can also be configured using the UCI command line utility. It also works by modifying the files in the /etc/config directory but is a method of configuring APs that is more suitable for scripting. Configurations are referenced to be read or modified through the UCI data model. The UCI data model consists of the elements listed below:

- The config element references the file in /etc/config that is being modified, for example wireless or dhcp.
- The sections element references a collection of configurations usually

about a single thing, for example a single Wi-Fi network.

- Different sections can be separated into types, for example interface sections can have the types lan, wan, loopback and wan6.
- Each section has options, these options reference specific configurations, for example the Internet Protocol (IP) address of an interface or the SSID of a Wi-Fi network.
- Each option has a value, this would be the actual IP address of an interface or the string that is the SSID of a Wi-Fi network.

The commands for an example configuration change of changing the port the uHTTPd Web Server is listening on to 8080 using the UCI command line tool is shown below [25].

```
uci set uhttpd.main.listen_http='8080'
uci commit uhttpd
/etc/init.d/uhttpd restart
```

When changing the configuration using the UCI command line tool, the command must be specified, such as setting a value, getting a value or deleting a value and then the target that is being modified must be specified using the UCI data model. For configuration changes to be saved, the commit command must be used, this saves the changes to the relevant file in the /etc/config directory and then as with manually modifying the files, the relevant programs must be restarted using their initialisation scripts in /etc/init.d/. Another common way of configuring OpenWrt APs is through the LuCI web interface. The interface makes use of the UCI command line utility to provide a simpler configuration method.

The decentralised control and data planes provided by OpenWrt and the UCI system mean that it doesn't have performance issues when used in large deployments like can happen with centralised control and data planes. The lack of a centralised management plane can be an issue though. The method of AP configuration provided by OpenWrt and the UCI system is very simple and makes it easy to deploy one or a few APs but by itself it is not very suitable for deploying large numbers of APs. Tools such as Ansible can be used with the OpenWrt UCI system to provide a centralised management plane and make deploying large numbers of APs practical, because of the OpenWrt and the

UCI system meet the requirement of being scalable and practical to use with a large number of APs.

The OpenWrt and the UCI system use hostapd to provide IEEE 802.11 functionality. Hostapd does support IPSK authentication but the UCI system doesn't support configuring hostapd to enable it. This means that the OpenWrt UCI system fails to meet the requirement that it support the method of authentication chosen in chapter 2. However, while testing OpenWrt and the UCI system for this project, the UCI system on an AP was modified so that it always configured hostapd to use IPSK. This was only done for testing purposes but if the UCI system was chosen as the authentication and management system it could be developed further to make it suitable. For example it would need to modified so that the RADIUS server hostapd uses could be configured through the UCI system. If it was developed further it could be made so that it meets the requirement that it support the method of authentication chosen in chapter 2.

The OpenWrt and the UCI system are mature and widely used and so meet the requirement that it be suitable for use in a real world deployment, they are open-source and so meets the requirement to be modifiable to allow custom features to be added.

OpenSync

OpenSync [24] is open-source software used to provide telemetry, control and networking on APs. OpenSync provides a centralised management plane while the APs each have their own self contained control and data planes. OpenSync operates as a collection of managers each with an individual purpose and also utilises Open vSwitch (OVS) and Open vSwitch Database (OVSDB) to provide a connection to the controller and to store configuration details. The configuration is stored and modified through an instance of ovsdb-server, that is started as part of OpenSync, which implements the OVSDB protocol. Configuration is applied to OpenSync by setting key value pairs in the relevant tables in ovsdb-server. The configuration is then used by the relevant manager. Only two of the managers are required to run: the diagnostics manager and the connection manager, the rest are optional and are only required to implement certain features. The functions of the managers which are used in this project are outlined below.

- Diagnostics Manager - The diagnostics manager is the first of the managers to be started. It is usually started by a init script. This script also starts ovsdb-server before starting the diagnostics manager. When the diagnostics manager has started, it then checks that it can connect to the ovsdb-server instance and then reads the initial configuration from it. The initial configuration that is read from the ovsdb-server instance is stored in a file copied into the APs file system when OpenSync is installed. The contents of that file is loaded into the ovsdb-server instance when it is started. This initial configuration contains information such as the address of the OpenSync controller that is required to finish the configuration of OpenSync. Once the diagnostics manager has checked, it can connect to the ovsdb-server instance, it starts the other OpenSync managers that are configured to be started.
- Connection Manager - The connection manager is responsible for connecting to the OpenSync controller, this connection is then used by the controller to configure OpenSync.
- Wireless Manager - The Wireless Manager reads the Wi-Fi network configuration details from the database and uses it to generate a hostapd configuration file that is then used to configure and run hostapd.
- Network Manager - The network manager handles the configuration of interfaces in OpenSync. It reads the relevant configuration from the database and can do things such as setting IP addresses for interfaces or creating an OVS bridge and adding interfaces to it.
- OpenFlow Manager - The OpenFlow manager is responsible for reading OpenFlow rules from the database and adding them to the OVS bridge. OpenFlow rules can either be configured on OVS instance either through loading them into OVSDB, and the OpenFlow manager loading them into the bridge or by setting an OpenFlow controller for the bridge and it loading OpenFlow rules into the bridge using the OpenFlow protocol.

OpenSync is configured by a central OpenSync controller operating in the management plane. It has no defined limit to how many APs it can configure, this along with OpenSync having decentralised control and data planes means that OpenSync meets the requirement to be easily scalable to make it practical to use with a large number of APs. Like the OpenWrt discussed in section 3.2.2,

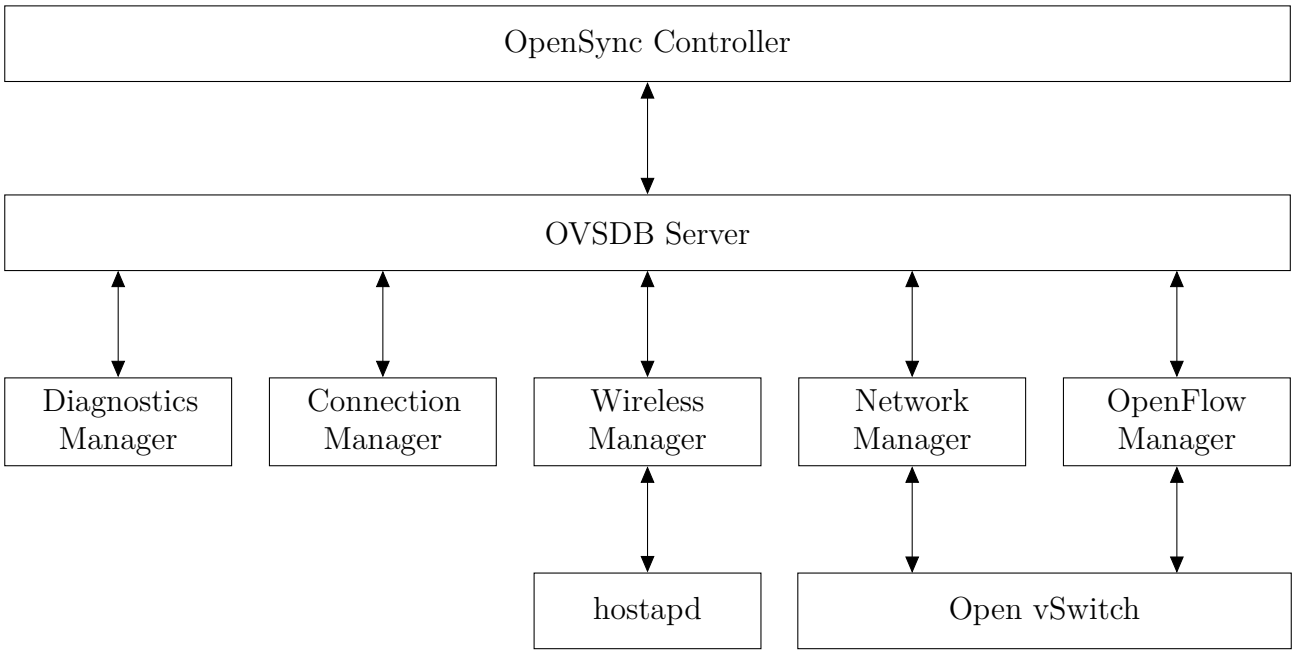

Figure 3.2: OpenSync Architecture

OpenSync doesn't natively support IPSK but it can be modified to do so that it, this would allow it to meet the requirement of supporting the method of authentication chosen in chapter 2. OpenSync is relatively new but it is out of beta and in active development and so should meet the requirement to be suitable for use in a real world deployment. OpenSync is open-source and so meets the requirement to be modifiable to allow custom features to be added.

3.3 Conclusion

Unlike in chapter 2, multiple options investigated in this chapter meet all the requirements. These options are OpenWrt with the OpenWrt UCI system and OpenSync. OpenWrt and the UCI system, however, has a decentralised management plane and so requires each AP to be individually configured and doesn't natively scale. Instead it relies on third party tools such as Ansible to be practical for deployments with large numbers of APs. In contrast to this, OpenSync has a centralised management plane and natively scales very easily. Because of this, OpenSync was chosen to be used for AP configuration and management in this project.

Chapter 4

Implementation

This chapter contains two sections, the first outlines each of the components that is used in this project and any work required to modify them to make them suitable for use, the second outlines how they work together in this project.

4.1 Components

4.1.1 OpenWrt

OpenWrt [26] is a Linux based operating system for embedded devices, it is most commonly used on network devices. The operating system can be configured either using the shell or the LuCI web interface. Packages can be installed using the opkg package manager. OpenWrt was chosen as the operating system to run on the APs as it is one of the most widely used open-source operating systems for wireless routers and APs. It also has support for a wide range of hardware as well as supporting thousands of application packages making it very flexible.

4.1.2 Open vSwitch

Open vSwitch [23] (OVS) is a production quality, multi layer virtual switch which meets the OpenFlow Switch specification and is configured using the OpenFlow protocol and the OVSDB protocol. OVS can be used as a switch in an entirely virtualised environment or to provide switching between a device's physical interfaces or a hybrid of both. OVS has two main components, ovs-switchd is a daemon that, in combination with a data path such as the Linux kernel data path, implements the switch. The other main component is the

ovsdb-server, this speaks the OVSDB protocol and is used by ovs-switchd to retrieve its configuration.

In this project, OVS is used as an OpenFlow switch. OpenFlow switches operate using a series of one or more flow tables and a group table, each of these tables is made up of one or more flow rules. When a packet enters an OpenFlow switch, a lookup is performed on table 0 to find which rule the packet matches. A packet will match the rule in a table that has the highest priority value of all the rules where the rules match fields match all of the values in the packet. The instructions in the matched rule are then actioned on the rule, there is a range of different instruction that can be actioned on a packet including modifying header values in the packet, sending the packet to another table or outputting it out a port. The tables and the rule in tables are configured by an OpenFlow controller speaking the OpenFlow protocol to the switch.

4.1.3 Open vSwitch Database protocol

The Open vSwitch Database (OVSDB) protocol [12] is the protocol used to interact with Open vSwitch databases which can be used both as a general database and also to configure instances of Open vSwtich. The OVSDB protocol is based on JSON-Remote Procedure Call (JSON-RPC) [9] in which interactions with the database are encoded in JavaScript Object Notation (JSON) [14] and sent directly over a Transmission Control Protocol (TCP) connection. There are many different JSON-RPC operations supported by the OVSDB protocol including listing all databases, getting the schema of a database, monitoring changes to the database and performing a list of transactions on the database. These transactions are how data is interacted with in the database. Transaction operations allow for data to be inserted, read, updated, mutated, deleted and for logical assertions to be made against the data.

4.1.4 Hostapd

Hostapd [16] is an open source user space implementation of the IEEE 802.11 standard for WLANs along with many of the authentication methods that are part of that standard such as WPA, WPA2, EAP and importantly for this project it supports IPSK. It also supports IEEE 802.1x, acting as a RA-

DIUS client and acting as an EAP server. Hostapd is configured through a text configuration file that lists the configuration parameters. In this project, hostapd is configured by OpenSync to generate the 802.11 Wi-Fi network and authenticate the Wi-Fi network with IPSK based on WPA2, the details of how WPA2 and IPSK work are explained in section 2.2.2. When performing IPSK authentication hostapd includes the MAC address of the station in the 'User-Name' and the MAC address and SSID of the network being connected to in the 'Called-Station-Id' attribute field. It expects the PSK to be returned in the 'Tunnel-Password' attribute field.

4.1.5 RADIUS

Remote Authentication Dial In User Service (RADIUS) [2] is a protocol used to provide authentication, authorisation and accounting on a network. RADIUS operates with a client-server model in which requests are sent between a client, known as a Network Access Server (NAS) and a RADIUS server in UDP packets. There are four types of request Access-Request, Access-Accept, Access-Reject and Access-Challenge. In normal operation a NAS will receive a connection from a device and, if configured to do so for that type of connection, the NAS will send an Access-Request to the RADIUS server. The Access-Request contains attribute fields which contain information about the connection to the NAS like the username or password submitted, the port connected to or the service being requested from the NAS. Request attributes can also contain information about the NAS itself, such as the IP address of the NAS or a NAS identifier string. Once a RADIUS server has received an Access-Request from a NAS it consults a database to try identify the matching user to that request. The RADIUS server can then reply with any of the three other request types. If the RADIUS server is able to identify the user in the database, and the requests attributes match and meet the requirements for that user, the RADIUS server replies with an Access-Accept response. The Access-Accept response can also be used to send configuration information for that configuration to the NAS. For example, the IP address for a device to be configured with. If the attributes of the Access-Request do not match the values expected or all required attributes are not present, the RADIUS server replies with an Access-Reject response. Alternatively, if all the attributes match, the RADIUS server can respond with an Access-Challenge response to request additional information from the user for authentication.

The Access-Challenge can contain a text message which the NAS can display to the user. The user's response to the Access-Challenge, whether they are shown a text message or not, is then sent from the NAS to the RADIUS server in an Access-Request message. Like the original Access-Request message, the RADIUS server consults a database to confirm if the Access-Request message meets the requirements for that user and can either send an Access-Accept, Access-Reject another Access-Challenge message.

Authentication of RADIUS requests is provided through the use of a shared secret which is known by both the NAS and RADIUS server but never sent across the network. To authenticate Access-Request messages, a randomly generated 16 octet number is set at the Authenticator, and this Authenticator value is concatenated with the shared secret then put through a one-way MD5 hash. The hashed value is then exclusive ored (XORed) with the password entered by the user and the result of that is stored in the User-Password attribute. To authenticate response messages (Access-Accept, Access-Reject and Access-Challenge), the Authenticator field is set to the output of a one-way MD5 hash of the response code, that represents what type of message it is, the identifier that is copied from the Access-Request, the length field, the Authenticator field from the Access-Request, the message attributes and the shared secret.

When hostapd is configured to use a RADIUS authentication server to retrieve the IPSK PSK, hostapd acts as the NAS. When a device connects, hostapd sends an Access-Request to the RADIUS server with the MAC address of the connecting device in the 'User-Name' and 'User-Password' attribute fields. If the authentication is successful the RADIUS server should reply with an Access-Accept message with the WPA-Personal password the connecting device is expected to use in the 'Tunnel-Password' attribute field. Unlike the 'User-Password' attribute field which is confirming that both the NAS and RADIUS server have the same values, the 'Tunnel-Password' attribute field has to send a value which is only known to the RADIUS server to the NAS. This means the 'Tunnel-Password' attribute field has to be encrypted using a different method to the 'User-Password' as the 'User-Password' encryption method only allows for both devices to check they have the same value, but not for one the decode the value if it does not know it. The 'Tunnel-Password' field is encrypted by first generating an intermediate value by concatenating and then MD5 hashing the shared secret, the Authenticator field from the Access-

request message and a randomly generated salt field that is also included in the Access-Accept message. The intermediate value is then XORed with the first 16 octets of the plain text password, the output of which is the first 16 octets of the encrypted password. A new intermediate value is then generated by MD5 hashing the shared secret, and the last 16 octets of encrypted password and this new intermediate value is then XORed with the next 16 octets of plain text password. This continues until the entire clear text password has been encrypted. This process is shown below where S represents the shared secret, R represents the request Authenticator from the Access-Request message, A represents the salt value, P represents the plain text password and the encrypted password is stored in C.

```
b(1) = MD5(S + R + A)     c(1) = p(1) xor b(1)    C = c(1)
b(2) = MD5(S + c(1))      c(2) = p(2) xor b(2)    C = C + c(2)
             .                         .
             .                         .
b(i) = MD5(S + c(i-1))    c(i) = p(i) xor b(i)    C = C + c(i)
```

4.1.6 PostgreSQL

PostgreSQL [28] is a SQL compliant open-source relational database management system. It supports both relational, SQL based, and non-relational, JSON based, querying. PostgreSQL natively supports a wide variety of data types as well as user definable data types. It uses transaction processing and is ACID (atomicity, consistency, isolation, durability) compliant, this helps to ensure the validly of data in the event of a error or incident such as a power cut or device failure.

PostgreSQL is used in this project to store the MAC addresses and passwords of devices that have connected to the Wi-Fi network as well as the default passwords for each AP that devices are expected to use when connecting to those APs.

4.1.7 YAML

YAML [31] is a Unicode based data serialisation language that is designed to be easily human-readable. It shares many native data types with Python and many other programming languages, making it easy to read or load data

to or from pragmatically. It is used widely for configuration files and data transmission among other uses.

YAML is used in this project to provide a configuration system that is both easily human-readable and easily readable in Python for the OpenSync controller. An example OpenSync controller configuration file is provided in appendix B.

4.1.8 OpenSync

As discussed in chapter 3, OpenSync [24] was chosen for configuration and management of the APs in this project.

OpenSync doesn't provide pre-built packages for OpenWrt, instead it provides the source code for the OpenSync core and example source code implementations for the hardware and OS target layers. An example target layer implementation is provided for OpenWrt, and an example hardware target layer is provided for armvirt which is intended to be used for virtual machines. This hardware target layer was modified to match the hardware used for testing in this project. The OpenSync core and various target layer source code is then used to build an OpenSync package. It wasn't until building the OpenSync package that the incomplete nature of OpenSync and its documentation became apparent. One undocumented requirement and two issues with dependencies in the build process were discovered.

The undocumented requirement was that OpenSync could not be built on Ubuntu version 16.04, and instead required the newer Ubuntu version 18.04. The two issues with dependencies were that libpcap was required to be declared as a dependency in the Makefile, and the other was that kconfiglib was required for the build process but not installed in the docker container that the package is built in. Once these three issues had been resolved, OpenSync was able to be built and installed.

When testing OpenSync, it was discovered that the function that manages the radio interfaces was only a stub function and control of the device's radios was not implemented in OpenSync. To overcome this, the stub function was implemented so that it generates a hostapd configuration file and then hostapd is started using this file. It was also discovered that the OpenSync OpenFlow manager did not add interfaces to an Open vSwtich bridge when it was configured to do so, this was also fixed. One other addition was made to OpenSync,

the Diagnostic manager was modified to load the host name of the AP into an OVSDB table so that it could be read by the OpenSync controller to identify the device and allow per device configuration.

4.1.9 OpenSync Controller

The OpenSync controller is a custom made Python program for this project. The controller handles two important functions, the configuration of APs and the authentication of devices connecting to the Wi-Fi network. It is designed to be a centralised controller that manages many APs across a network. The controller takes two arguments when it is started, a YAML based configuration file and a port for it to listen for OVSDB protocol connections.

Once the configuration file has been loaded, a radius_server thread is started. This is an implementation of the custom RADIUS server, used for IPSK with a Default PSK authentication and described in section 2.2.2, built using the pyrad RADIUS client/server Python package. The details used to authenticate RADIUS requests with are stored in a PostgreSQL database, this database is communicated with using the psycopg2 Python package. Once the custom RADIUS server has been started, the controller opens a server socket listening on the port specified in the argument given when the controller was started. Every time an AP connects to the server socket, a new thread is created to configure that AP, this allows for the controller to configure multiple APs at once. When OpenSync is started, it loads the host-name of the device it is running on into the OVSDB server. When the controller receives a connection, it retrieves that host-name and uses it to identify what configuration should be applied to the AP. AP configurations can be stored in the configuration file under either a host-name or the name 'default'. Once the controller has retrieved a connecting devices host-name, it searches the configurations from the configuration file, and if it finds a configuration with a name that matches the host-name, it applies that configuration to the device. If it does not find a matching named configuration, it will check if there is a configuration name 'default', if that exists, it will apply that configuration. If there is neither a configuration matching the host-name nor a configuration called 'default', then the AP is not configured. Once the AP has connected and the appropriate configuration has been found the configuration is applied to the AP using the OVSDB protocol. The AP is then registered in the controller's RADIUS PostgreSQL database with the PSK from the configuration file to be used when

first connecting devices through that AP.

When making the controller, there were three main options considered for implementing the OVSDB protocol. These were using the Ryu OVSDB Manager library [38], calling the ovsdb-client shell command [27] or implementing the protocol in the controller using the Python network socket and JSON libraries. Each of these different methods of implementing the protocol were tested and it was decided to implement the protocol using the Python network socket and JSON libraries. This was because OVSDB is a simple protocol and that meant that the abstraction provided by the other options did not simplify the communication significantly and also implementing the protocol in the controller gave greater control over the communication to the APs. Whilst testing the Ryu OVSDB Manager library, a bug was discovered that caused the manager to crash when a connection was made to an OVSDB server. In order to be able to test the library fully, this bug was fixed and the patch was contributed to the Ryu project [34].

4.2 Operation

When the controller is operating, there are two main events that can happen: the configuration of a new AP and the authentication of a device connecting to the Wi-Fi network being broadcast by one of the APs. This section explains in detail how each of those operations works.

4.2.1 Access Point Configuration

For an AP to be configured, the OpenSync controller must be running and accessible on the management network as the AP. OpenSync is started on an AP using its init script, this script starts ovsdb-server and the OpenSync Diagnostics Manager (DM). When ovsdb-server is started, it loads the database file that contains the preloaded values into the database, this loads the address of the OVSDB manager into the database and ovsdb-server then connects to that manager. In this project, the OpenSync controller acts as the OVSDB manager.

To configure the AP, the controller first loads the configuration for all the interfaces that are going to be managed by OpenSync on the device. These interfaces are all added to the OVS bridge that is created by OpenSync when it

is started. The OpenSync controller can either set an OpenFlow controller for the OVS bridge which can dynamically add and remove flows and provide all the other features that come with using an OpenFlow controller. Alternatively, flows can be applied statically by the OpenSync controller by loading them into the OVSDB, these rules are then applied to the OVS bridge and cannot be changed without reconfiguring the AP.

Once the interfaces have been configured, the wireless interfaces have further configuration loaded into the ovsdb-server instance by the controller and then applied. This configuration is made up of configurations specific to the wireless interfaces such as the frequency band and protocol to use. The next configuration applied is the details of the Wi-Fi network to be generated by the AP, this consists of the networks SSID and the authentication method to use. This configuration, along with the wireless interface configuration, is then used to create a hostapd configuration file, and an instance of hostapd is started.

The final step the controller takes when configuring a device is to register that device in the PostgreSQL database with the password for that device from the configuration file. This PSK is the PSK devices are expected to use when connecting to this AP if they are not already registered in the database themselves.

4.2.2 Device Authentication

This project supports three different types of Wi-Fi authentication. These are an open network with no authentication, WPA-Personal and IPSK with a Default PSK authentication method designed in this thesis. How these methods of authentication operate are explained in section 2.2.

Chapter 5

Testing

The testing of this project, to ensure it performed as expected and met its goals, has been done through a manual process where everything was set up and used as if it was deployed in a real world environment. This method of testing was chosen as opposed to automated unit and integration testing for two main reasons. The first is that during the time this project has been developed, development for OpenSync 1.4, which this project is based upon, has stopped, and to update the project so that it uses OpenSync 2 or later would require significant work, this is explained further in section 6.4.3. Because an important piece of software this project relies upon to work is no longer receiving updates, it would not be suitable to be deployed in a real world environment. As this project is not going to be deployed in a real world environment, automated testing is less important than it would be if it were to be deployed and have continued development. The second reason manual testing was chosen over automated testing was due to the nature of this project. Developing an automated testing framework which runs OpenSync on OpenWrt and can simulate devices connecting and disconnecting to the Wi-Fi would be time consuming and excessive for a project that is not going to be deployed in a real world environment. The manual testing that has been performed covered all scenarios that could reasonably be expected to occur.

5.1 Testbed

The testbed for this project is shown in figure 5.1. In this testbed there is two APs both running OpenSync and both configured using the OpenSync controller running on a server. There are two different networks which the

APs are connected to. The management network is used for communication between the controller and APs for AP configuration and RADIUS authentication requests. The management network is also used to access anything running on the AP such as the SSH server. Every interface, other than the one that is used to connect to the management network, is added to the OVS bridge running on the AP. This includes both wired and wireless interfaces. One of the wired interfaces that is connected to the OVS bridge is connected to the other network which has Internet access. This other network being connected to the same OVS bridge as the wireless interfaces means that any devices connecting to the Wi-Fi can connect through the OVS bridge to the Internet.

The two APs used in this testbed are GL-AR750S wireless travel routers from GL-iNet and the Wi-Fi device used for testing is an Apple iPod Touch 5th generation. The switches used to connect the APs to the two networks are both unmanaged gigabit Ethernet switches. The server which runs the OpenSync controller is running Debian version 9.13. The OpenSync controller was configured with the configuration file in appendix B with the encryption type changed to the relevant configuration depending on the test being performed.

5.2 Tests Performed

Open Wi-Fi Network Test

Using the OpenSync controller, an AP running OpenSync was configured to broadcast an unsecured Wi-Fi network and a device was successfully connected to it. This tested that the OpenSync controller is able to configure OpenSync and that OpenSync is able to generate a valid hostapd configuration file and then call hostapd. The output of the OpenSync controller when doing this is show in figure 5.2.

WPA-Personal Secured Wi-Fi Network Test

Using the OpenSync controller, an AP running OpenSync was configured to broadcast a Wi-Fi network secured with standard WPA-Personal, and a device was successfully connected to it. This tested almost the exact same things as the previous test but this time tests that OpenSync is able to generate a valid configuration file for hostapd that uses WPA-Personal.

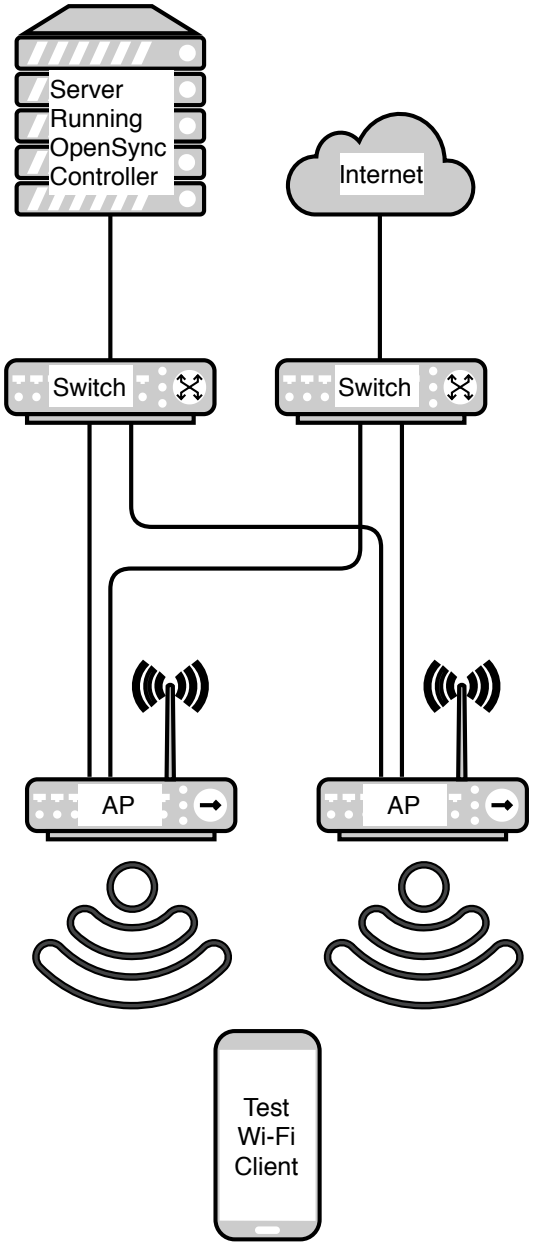

Figure 5.1: Testbed Used For Testing

OpenFlow Rule Applied by Faucet SDN Controller Test

Using the OpenSync controller, OpenSync, running on an AP, was configured to set an instance of the Faucet Software Defined Networking (SDN) controller as the controller for the Open vSwitch instance started by OpenSync. The Faucet SDN controller instance was then used to configure OpenFlow rules on the AP, and testing was performed to ensure these rules had been successfully applied. This tested that the controller is able to configure a custom OpenFlow controller in OpenSync and that OpenSync then correctly sets that controller in OVS.

OpenFlow Rule Applied by OpenSync Test

Some OpenFlow rules were configured in and applied by the OpenSync controller, as opposed to using an external OpenFlow controller. Testing was then performed to ensure these rules had been successfully applied. This tested that the controller was able to configure OpenFlow rules in OpenSync and that OpenSync was able to apply these rules to the OVS bridge.

IPSK with a Default PSK Authentication Test

An AP was configured to broadcast a Wi-Fi network secured with IPSK with a default PSK authentication developed in this thesis, and a device that had not been connected before was successfully connected to the Wi-Fi network using the correct password. This tested that OpenSync is able to generate a hostapd file to enable WPA-PSK-RADIUS, used for IPSK, in hostapd. It also tested that the custom RADIUS server is able to receive authentication requests from hostapd, make SQL queries to the PostgreSQL database, follow the process to authenticate requests outlined in figure 2.4 and respond to requests.

The contents of the PostgreSQL table used to store device MAC addresses and passwords is shown before and after the test device was authenticated to the Wi-Fi network in figure 5.3. The top output shows the table before the IPod has been added, it shows the entry for the AP and the password devices are expected to use when connecting for the first time through that AP. The username field is in the format given by hostapd for the AP details in the 'Called-Station-Id' RADIUS attribute field. The bottom output shows the same table once the IPod has been connected and added to the database. Its username is in the format given by hostapd in the 'User-Name' field. A screenshot of the IPod, that was used at the test device to connect to the

Wi-Fi, is show in figure 5.4.

IPSK with a Default PSK Authentication and an Incorrect PSK Test

An AP was configured to broadcast a Wi-Fi network secured with IPSK with a default PSK authentication developed in this thesis. A device that had not been connected before then tried to connect using a different password from what was configured for unknown devices to use when connecting through that AP and was unable to connect to the Wi-Fi network using the incorrect password as expected. This tested that the authentication is working correctly in hostapd and it is only authenticating devices when the password matches the password returned by the RADIUS request. It also tested that the custom RADIUS server is correctly checking the password devices are expected to use on an AP if the device isn't already in the database.

IPSK with a Default PSK Authentication Reconnecting Test

An AP was configured to broadcast a Wi-Fi network secured with IPSK with a default PSK authentication developed in this thesis. A device that had previously been connected, and therefore the details of the device was in the device database, was successfully connected to the Wi-Fi network using the correct password. The device was also not added to the database a second time. This tested that the custom RADIUS server was correctly checking if a device was in the database and returning the password for the device in the database instead of adding it a second time.

Roaming Between APs with IPSK with a Default PSK Authentication Test

Two APs were configured to broadcast a Wi-Fi network secured with IPSK with a default PSK authentication developed in this thesis. A device was connected to one AP and was then disconnected and connected to the other AP. These two APs were configured with different passwords required to be used when first connecting devices but, because the MAC address and password for the device was registered in the database from when it connected to the first AP, it was able to connect to the second AP with the same password used to connect to the first AP. This tested that the custom RADIUS server correctly looks up devices based on their MAC addresses and returns the correct password even if they are connecting through a different AP.

Roaming Between APs with IPSK with a Default PSK Authentication and an Incorrect PSK Test

Two APs were configured to broadcast a Wi-Fi network secured with IPSK with a default PSK authentication developed in this thesis. A device was connected to one AP and was then disconnected and it was attempted to be connected to the other AP with the default password for the other AP. These two APs were configured with different passwords required to be used when first connecting devices but, because the MAC address and password for the device was registered in the database from when it connected to the first AP, it wasn't able to connect to the second AP with the default password for the second AP. This tested that the custom RADIUS server correctly looks up devices based on their MAC addresses and returns the correct password even if they are connecting through a different AP. It also tested that hostapd won't authenticate the device if the device used by the password doesn't match the password returned in the RADIUS request.

```
Connection from sidewalk-ap-1 at ('114.134.11.156', 50620)
configuring  sidewalk-ap-1  with default config
configuring interfaces
configuring WLANs
configuring WLAN interfaces
enabling WLANs
checking AP regestered in postgresql db
device added to db with User-Name =  \
    E4-95-6E-4A-72-67:testSSID1 and password =  somePassword
```

Figure 5.2: Output from OpenSync controller configuring an AP

```
radius=# select * from radcheck;
          username           |   password
-----------------------------+--------------
 E4-95-6E-4A-72-67:testSSID1 | somePassword
(1 row)

radius=# select * from radcheck;
          username           |   password
-----------------------------+--------------
 E4-95-6E-4A-72-67:testSSID1 | somePassword
 30074d64839e                | somePassword
(2 rows)
```

Figure 5.3: The contents of the table storing authentication data before and after a device has been connected to the Wi-Fi network

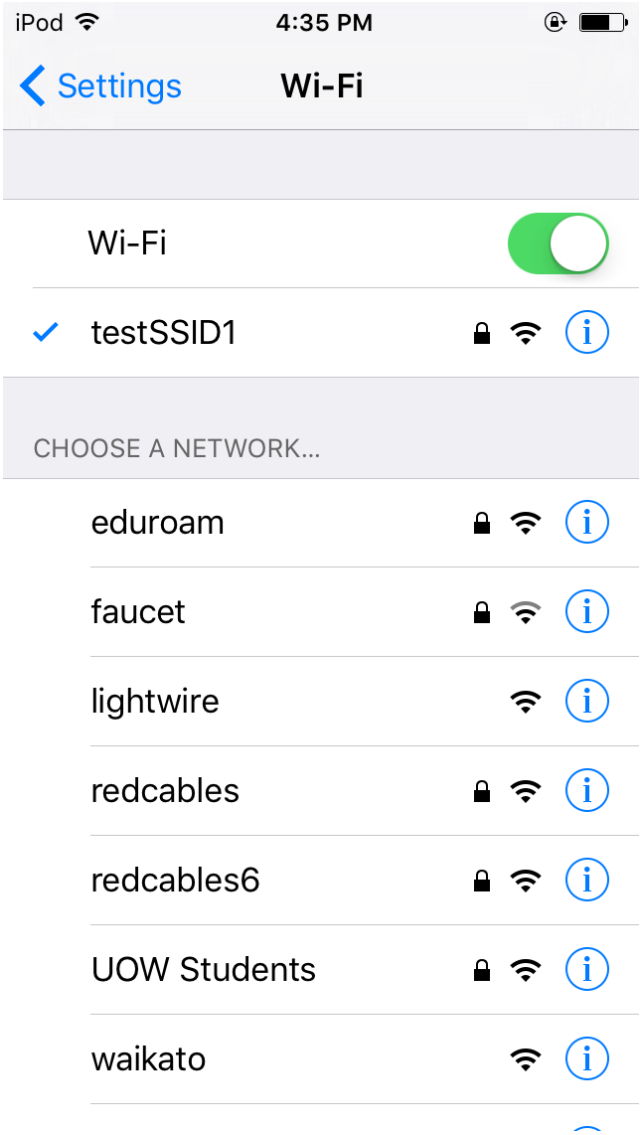

Figure 5.4: A screenshot of an IPod touch connected to a Wi-Fi network broadcast by one of the APs

Chapter 6

Discussion

This section discusses where this project is today, how this project met the goals outlined in sections 2.1 and 3.1, the challenges encountered meeting those goals and the potential future work that could be done.

6.1 Goal Evaluation

The goal of this project is to create a wide area 802.11 Wi-Fi network which many users can connect their devices to but is segmented so that each user has their own virtual layer 2 network. It also aims to provide a method of AP configuration and management suitable for the large number of APs that would be required for this network to cover a wide area.

This project has met the goals of providing a suitable method of user authentication and identification and AP configuration and management. The one part of the goal that this project does not provide is a method of network segmentation so that each user has their own virtual layer 2 network. The user authentication and AP configuration and management are, however, designed so that no changes are required to either the software running on the APs or the controller to implement the network segmentation. This is because PostgreSQL database, populated by the custom RADIUS server when it authenticates devices, provides all the information required to identify device owners to be able to segment the network correctly. The ability to configure the OVS instance running on the APs to use any OpenFlow controller means that OpenFlow could be used to apply any rules required to segment the network correctly.

The requirements and goals for this project were outlined in sections 2.1 and 3.1 and the details of the testing of this project is outlined in sections 5.2. Each of the specific requirements and how they are met is covered below.

Using a Secure and Widely Supported Authentication Method

The authentication in this project is based on IPSK which appears to any device connecting to be WPA-Personal. This means it is secure and supported by virtually all Wi-Fi enabled devices.

Providing a Method to Identify Device Owners

When devices are first authenticated, their MAC address is stored in a database along with the PSK that the device is expected to use. That can be used to identify the device owner to segment the network correctly.

Using a Simple to Use Authentication Method That Doesn't Require Centralised On-Boarding

The authentication using IPSK means that authenticating devices is as simple for the user as could reasonably be expected. Apart from the users having to ensure they are in their home or apartment when first connecting to the Wi-Fi there is no difference to connecting to any other Wi-Fi network that is secured with WPA-Personal.

The Authentication Must Support Seamless Roaming and Reconnecting to APs

Storing device MAC addresses and passwords in a central server means that devices can roam and reconnect to any AP in the network with no extra configuration required by the user.

The Configuration and Management of APs must be Easily Scalable

OpenSync automatically connects to the controller and pulls down the configuration meaning, once OpenSync is installed, it's just as easy to configure one AP as it is to configure one hundred APs.

The Configuration and Management of APs must Support the Authentication Method Chosen

OpenSync has been modified in this project so that it can be used to enable IPSK authentication in hostapd and therefore supports the authentication method chosen.

The Configuration and Management of APs must be Suitable for a Real World Deployment

The AP Configuration and Management is based around OpenSync which is still under active development and receiving updates. Although, as explained further in section 6.4.3, this project as been built using OpenSync 1.4 and so it would have to be updated to use the newer OpenSync 2.

The software on the AP must be modifiable to allow custom features to be added

The APs run OpenWrt and are configured using OpenSync, both of these pieces of software are open-source and can be modified.

6.2 Challenges

The main area challenges were encountered in the project was improving the software used so that it was suitable and met the requirements to be used in this project. The majority of the issues encountered were in OpenSync.

There were three main issues with OpenSync, poor quality documentation, issues with the build process and some features not being implemented. The main area in which the poor quality documentation was an issue was in building the OpenSync controller or as the documentation calls it 'the cloud.' The OpenSync documentation describes OpenSync as cloud agnostic, it says OVSDB is used for inter-process communication between the cloud and OpenSync managers, and it provides documentation for the tables in OVSDB. That is all of the documentation provided about the behaviour of the OpenSync controller, and no example implementation is provided. This made developing the OpenSync controller difficult as the required behaviour of the controller instead had to be figured out by reading OpenSync's code and through guess and check methods. This was made worse by some unintuitive requirements around the order the configuration must be sent to OVSDB.

The other main issues encountered with OpenSync were to do with building OpenSync and adding missing features. This has been discussed further in section 4.1.8.

6.3 Contribution

There are two main contributions that have been made in this thesis: the novel authentication method and development of OpenSync for OpenWrt to make it functional and of an OpenSync controller. The novel authentication method, based around IPSK and the custom RADIUS server, provides a combination of features that aren't available with other authentication methods. It provides the ability to control the access of individual users and identify their devices, like WPA-Enterprise, but also has the ubiquitous support and ease of use of WPA-Personal. This combination of features and widespread support is not available with other authentication methods. The other main contribution of this thesis was to further develop OpenSync to make it usable and to develop an OpenSync controller to use with it. OpenSync has been developed from a state in which it was missing key features that made it effectively nonfunctional to being able to configure all of the important features of an AP. Along with this a functional controller was developed to manage OpenSync.

6.4 Future Work

6.4.1 Meeting The Larger End Goal

This project was developed with a larger end goal in mind. This goal is, as explained in section 1.1.1, the creation of a single WLAN which many users can connect to but each user or group of users has their own virtual layer 2 network that all their devices are, on regardless of where they connect to the network. The problems that need to be solved to meet this goal can be split into three main areas: device authentication, AP configuration and the creation of virtual layer 2 networks. The work done in this project provides the solutions to two of these three areas.

The main area of work that is still required is the creation of virtualised layer 2 networks for each user or group of users. There are many different ways this could be done, but despite this, there shouldn't be any changes required to OpenSync or any other software running on the APs. This is because all the switching on the APs is handled by OVS and can be controlled by an external OpenFlow controller, meaning that the OpenFlow rules on the AP can be changed at any time making it very flexible. One way the layer 2 network could be virtualised would be for all traffic from devices on the network to be

sent to one or more central servers which would identify devices and which virtual network they should be on based on the devices MAC address. The servers could keep track of which devices are connected to which AP using the RADIUS authentication requests made by the APs and could then send packets to the relevant AP which would then broadcast it to the device. An alternative method that could be used would be to have an OpenFlow controller that monitors which APs devices are connected to and then modifies the OpenFlow rules on the AP to reflect that. The rules could be used to add VLAN headers to packets based on the MAC address of the device. The APs could then be connected together using OpenFlow switches that would be updated by the same controller and would know which APs have devices from each VLAN connected to them, and would only send packets to the relevant APs. This method of creating a virtual layer 2 network would obviously be limited by the 4094 possible VLAN identifier field values in an IEEE 802.1q header but could be extended to support more virtual networks through the use of the IEEE 802.1ad standard which allows for the stacking of VLAN headers. Both of these proposed methods of extending this project to create virtual layer 2 networks would require further research before they could be implemented but could be a useful starting point for that research.

6.4.2 Additional Features

There is also some work that could be done to improve the device authentication. Currently, users can only register devices by connecting to the Wi-Fi network using their home AP, and there's no way to deregister devices. Improvements to this could be relatively straight forward to implement. One improvement that could be made would be a web portal which users can access and shows them all devices that are registered to them or their group of users and would allow them to deregister any of those devices. The web portal would work by reading and modifying the data in the SQL database that contains all the devices and passwords. The portal could also be used to change the user's password for the Wi-Fi network, this could either just change the password new devices are expected to use when connecting to their AP, this would leave all devices that have already been connected able to use the original password. Alternatively, all of a user's devices could have their expected password updated to the new password, or they could be removed and have to be registered again.

The Web Portal could also be used to register new devices because, while it wouldn't be suitable as a method for all users to have to use to register all devices, it could be useful in some situations. If, for example, in the unlikely but possible situation where a user has a wall mounted smart TV which is closer to the AP in the neighbour's apartment, the TV might try to connect to that AP instead of the one in the user's apartment, this would leave the device unable to be connected to the Wi-Fi by the user. If the user was able to register devices through the Web portal, they could get the MAC address of the TV, enter it in the web portal and it would be added to the SQL database. This would mean the user could connect the TV to the Wi-Fi using their password and even though the TV was connecting to the AP in the neighbouring apartment. This is because it would already be registered and so when the neighbouring AP make a request for the password for the TV to use it would be returned the user's password.

6.4.3 Required Future Work

On the 17th of March 2020 OpenSync 2.0 was released [29], this superseded OpenSync 1.4 that has been used for this project. This was about two months into the development of this project using OpenSync 1.4. It was chosen to continue using OpenSync 1.4 for a range of reasons. OpenSync was chosen because when looking at options for for AP management, the then latest version of OpenSync, OpenSync 1.4 supported OpenWrt. It provided documentation, an OpenSync platform layer, makefiles and automated scripts to build OpenSync 1.4 for OpenWrt. These were provided with OpenSync 2 for other platforms but not for OpenWrt. OpenSync 2 has a similar architecture as OpenSync 1.4 and it would be possible to build OpenSync 2 for OpenWrt however this would require significant development which would be be beyond the scope of this thesis. However if this project were to continue being developed further this work would need to be done as OpenSync 1.4 is no longer receiving updates and so would not be suitable for a real world deployment.

Chapter 7

Conclusion

This thesis has investigated and provided a novel method of authenticating devices to an IEEE 802.11 Wi-Fi network that allows per user authentication and for the ownership of devices to be tracked. It does this while only requiring the connecting device to support WPA-Personal and while being almost completely transparent to the user. It builds upon IPSK authentication and adds some of the features from WPA-Enterprise outlined in section 2.2.2. In particular, it adds the ability for users to easily connect new devices to the network. It also provides a method of AP configuration which is highly scalable and allows for the centralised configuration of an arbitrary number of APs. The AP configuration and management method used provides a compromise between the highly centralised AP configuration methods such as that outlined in the CloudMAC paper detailed in section 3.2.1 and the more decentralised methods such as the OpenWrt UCI system detailed in section 3.2.2.

The solutions are also suitable so that the work on this thesis can be extended further to allow each users devices to be isolated on an individual virtual layer 2 network. If this thesis was to be extended further in this way, an IEEE 802.11 Wi-Fi network that spans a large area could be created that many different people could connect to. On this network, it would appear to each users devices that they are on their own private layer 2 network with no other users devices on it.

References

[1] Ralph Droms. *Dynamic Host Configuration Protocol*. RFC 2131. Mar. 1997. DOI: 10.17487/RFC2131. URL: https://rfc-editor.org/rfc/rfc2131.txt.

[2] Allan Rubens et al. *Remote Authentication Dial In User Service (RADIUS)*. RFC 2865. June 2000. DOI: 10.17487/RFC2865. URL: https://rfc-editor.org/rfc/rfc2865.txt.

[3] "IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements". In: *IEEE Std 802.11i-2004* (2004), pp. 1–190. DOI: 10.1109/IEEESTD.2004.94585.

[4] John Vollbrecht et al. *Extensible Authentication Protocol (EAP)*. RFC 3748. June 2004. DOI: 10.17487/RFC3748. URL: https://rfc-editor.org/rfc/rfc3748.txt.

[5] Eric Rescorla and Nagendra Modadugu. *Datagram Transport Layer Security*. RFC 4347. Apr. 2006. DOI: 10.17487/RFC4347. URL: https://rfc-editor.org/rfc/rfc4347.txt.

[6] V. Soni and R. Mendiratta. "Next-generation wlan architecture for high performance networks". In: *2008 IET International Conference on Wireless, Mobile and Multimedia Networks*. 2008, pp. 125–129. DOI: 10.1049/cp:20080161.

[7] Dorothy Stanley, Michael Montemurro, and Pat R. Calhoun. *Control and Provisioning of Wireless Access Points (CAPWAP) Protocol Binding for IEEE 802.11*. RFC 5416. Mar. 2009. DOI: 10.17487/RFC5416. URL: https://rfc-editor.org/rfc/rfc5416.txt.

[8] Dorothy Stanley, Michael Montemurro, and Pat R. Calhoun. *Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification*. RFC 5415. Mar. 2009. DOI: 10.17487/RFC5415. URL: https://rfc-editor.org/rfc/rfc5415.txt.

[9] JSON-RPC Working Group. *JSON-RPC 2.0 Specification*. Mar. 2010. URL: https://www.jsonrpc.org/specification.

[10] Aerohive Networks. *Aerohive Solution Brief - Aerohive Private PSK*. 2011. URL: https://www.protelesis.com/pdf/aerohive-private-pre-shared-key.pdf.

[11] P. Dely et al. "CloudMAC — An OpenFlow based architecture for 802.11 MAC layer processing in the cloud". In: *2012 IEEE Globecom Workshops*. 2012, pp. 186–191.

[12] Ben Pfaff and Bruce Davie. *The Open vSwitch Database Management Protocol*. RFC 7047. Dec. 2013. DOI: 10.17487/RFC7047. URL: https://rfc-editor.org/rfc/rfc7047.txt.

[13] "IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications". In: *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)* (2016), pp. 1–3534. DOI: 10.1109/IEEESTD.2016.7786995.

[14] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. Dec. 2017. DOI: 10.17487/RFC8259. URL: https://rfc-editor.org/rfc/rfc8259.txt.

[15] *Captive Portal Redirect on Initial Browser HTTPS Request*. May 10, 2019. URL: http://docs.ruckuswireless.com/zonedirector/10.2/GUID-5AE1C931-B393-4431-84CB-2C1E58088440.html (visited on Nov. 3, 2020).

[16] *hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator*. Aug. 8, 2019. URL: https://w1.fi/hostapd/ (visited on Nov. 7, 2020).

[17] *8.5 Identity PSK Feature Deployment Guide - Cisco*. Nov. 8, 2020. URL: https://www.cisco.com/c/en/us/td/docs/wireless/controller/technotes/8-5/b_Identity_PSK_Feature_Deployment_Guide.html (visited on Nov. 8, 2020).

[18] *Aruba Mobility Controller Virtual Appliance — Aruba*. Nov. 7, 2020. URL: https://www.arubanetworks.com/products/wireless/gateways-and-controllers/virtual-appliance/ (visited on Nov. 7, 2020).

[19] aviviano. *Captive portals - Windows drivers — Microsoft Docs*. Nov. 8, 2020. URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/mobilebroadband/captive-portals (visited on Nov. 8, 2020).

[20] *Broadband Portal - OECD*. Oct. 17, 2020. URL: https://www.oecd.org/sti/broadband/broadband-statistics/ (visited on Nov. 7, 2020).

[21] "IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control". In: *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)* (2020), pp. 1–289. DOI: 10.1109/IEEESTD.2020.9018454.

[22] *In-Depth Analysis of Changes in World Internet Performance Using the Speedtest Global Index*. Nov. 7, 2020. URL: https://www.speedtest.net/insights/blog/global-index-2019-internet-report/ (visited on Nov. 7, 2020).

[23] *Open vSwitch*. Nov. 7, 2020. URL: http://www.openvswitch.org/ (visited on Nov. 7, 2020).

[24] *OpenSync*. Nov. 7, 2020. URL: https://www.opensync.io/ (visited on Nov. 7, 2020).

[25] *OpenWrt Project: The UCI system*. Sept. 29, 2020. URL: https://openwrt.org/docs/guide-user/base-system/uci (visited on Sept. 29, 2020).

[26] *OpenWrt Project: Welcome to the OpenWrt Project*. Nov. 7, 2020. URL: https://openwrt.org/ (visited on Nov. 7, 2020).

[27] *ovsdb-client*. Aug. 18, 2020. URL: http://www.openvswitch.org/support/dist-docs/ovsdb-client.1.txt (visited on Sept. 25, 2020).

[28] *PostgreSQL: The world's most advanced open source database*. Nov. 7, 2020. URL: https://www.postgresql.org/ (visited on Nov. 7, 2020).

[29] *Release 2.0.0.0 · plume-design/opensync@063e1af*. Oct. 30, 2020. URL: https://github.com/plume-design/opensync/commit/063e1af2502de222669c5e8aa7decf3e0f67a34d (visited on Oct. 30, 2020).

[30] *Security — Wi-Fi Alliance*. Nov. 8, 2020. URL: https://www.wi-fi.org/discover-wi-fi/security (visited on Nov. 8, 2020).

[31] *The Official YAML Web Site*. May 10, 2020. URL: https://yaml.org/ (visited on Nov. 7, 2020).

[32] *Understanding PSK Authentication - TechLibrary - Juniper Networks*. July 16, 2020. URL: https://www.juniper.net/documentation/en_US/junos-space-apps/network-director4.0/topics/concept/wireless-wpa-psk-authentication.html (visited on Nov. 8, 2020).

[33] *Understanding PSK Authentication - TechLibrary - Juniper Networks*. July 16, 2020. URL: https://www.juniper.net/documentation/en_US/junos-space-apps/network-director4.0/topics/concept/wireless-wpa-psk-authentication.html (visited on Nov. 8, 2020).

[34] *updated jsonrpc.Session call to have correct arguments for latest ver... · faucetsdn/ryu@c343376*. Sept. 26, 2020. URL: https://github.com/faucetsdn/ryu/commit/c34337684964029ffd4fc5653633892863942fd6 (visited on Sept. 26, 2020).

[35] *Wireless LAN Controller (WLC) - Cisco*. Nov. 7, 2020. URL: https://www.cisco.com/c/en/us/products/wireless/wireless-lan-controller/index.html (visited on Nov. 7, 2020).

[36] *WPA3: Your next wireless devices should support it — HPE*. Nov. 7, 2020. URL: https://www.hpe.com/us/en/insights/articles/wpa3-your-next-wireless-devices-should-support-it-1910.html (visited on Nov. 8, 2020).

[37] *XBOX One Falls Short, No WPA2-E Support — Aruba Blogs*. Oct. 23, 2020. URL: https://blogs.arubanetworks.com/solutions/xbox-one-falls-short-no-wpa2-e-support/ (visited on Oct. 23, 2020).

[38] *OVSDB Manager library¶*. URL: https://ryu.readthedocs.io/en/latest/library_ovsdb_manager.html.

[39] *Types of Wi-Fi networks that aren't recommended or won't work with Nest products*. URL: https://support.google.com/googlenest/answer/9249905?hl=en.

Appendix A

Source Code

All of the code produced in this thesis is available at

https://github.com/sdyear/opensync-openwrt

Appendix B

Example OpenSync Controller Configuration File

This is an sample configuration file for the OpenSync controller.

```
#configuration for the postgreSQL database
postgresql_db:
  address: '163.7.137.203'
  dbname: radius
  user: radius
  password: password
# a list of different configurations for APs, configs in the list can
# either be titled 'default' or the hostname of the specific AP that
# config is targeting. The controller checks the hostname against the
# list of AP configs and if it matches it will apply that config to
# the AP otherwise it will apply the config called default if that
# exists, otherwise it will apply no config.
access_points:
  default:
    # OpenFlow rules on Open vSwitch running on the AP can either be
    # configured through an OpenFlow controller or through specifying
    # the individual rules as shown below. Only one of these options
    # should be used at a time. If neither is specified Open vSwitch
    # will forward the packets to NORMAL (the non-OpenFlow pipeline).
    # openflow-controller:
    #   address: "tcp:163.7.137.63:6653"
    #   datapath-id: "0x01"
    # The rule field can be left blank to create a rule with a rule
    # which wildcards all match fields (e.g. for a table-miss rule).
    # openflow_rules:
    #   - action: normal
    #     priority: 1
    #     rule:
    #     table: 0
    #     token: rule1
    #   - action: out_pot=2
```

```
#     priority: 2
#     rule: in_port=any,eth_type=0x800
#     table: 0
#     token: rule2
# This section specifies the interfaces to be added to the
# Open vSwtich bridge.
interfaces:
  - if_name: "eth0.2"
    if_type: eth
    enabled: true
    network: false
    of_port: 1
  - if_name: "wlan0"
    if_type: eth
    enabled: true
    network: false
    of_port: 2
# this section is used to specify what interfaces are wireless
# interfaces and to configure what mode they opperate in.
wlan_interfaces:
  - if_name: wlan0
    enabled: true
    freq_band: 5G
    country: NZ
    hw_mode: 11n
#  this section defines the wireless networks which are broadcast
#  by the AP. There are 3 different security options.
# encryption: OPEN
#  this disables WPA and makes a completely open wifi network
# encryption: WPA-PSK
# passphrase: <passphrase
#  This configures standard wpa-psk with a single passphrase as
#  specified
# encryption: WPA-PSK-RADIUS
# auth_server: "163.7.137.63:1812"
# auth_server_shared_secret: secret
# default_password: somePassword
#  This configures Individual PSK based authentication, the
#  details of a RADIUS server must be specified. This RADIUS
#  server is then queried every time a device connects. The
#  default_password feild specifies the password a unknown
#  device is expected to use for that AP.
wifi_networks:
  - ssid: testSSID1
    if_name: wlan0
    enabled: true
    security:
     - encryption: WPA-PSK-RADIUS
     - auth_server: "163.7.137.63:1812"
     - auth_server_shared_secret: secret
     - default_password: somePassword
```

Appendix C

Setup instructions

These instruction cover how to build and set up this project using the example vendor target provided. Instructions on how to create other vendor target layers for other OpenWrt based APs are also provided below. Port 6640 is the Internet Assigned Numbers Authority (IANA) assigned port number for OVSDB management protocol and so is recommended to be used for communication between the APs and controller.

C.1 Build Instructions

First clone the project repository, change directory into the cloned repository and fetch the submodules:

```
git clone https://github.com/sdyear/opensync-openwrt.git
cd opensync-openwrt
git submodule update --init
```

An example target OpenWRT layer is provided in `opensync/vendor/ath79`, this can be modified to support other OpenWRT targets with some relatively small changes.

- Copy vendor `/ath79` to `vendor/<new_target>`
- Modify `build/vendor-arch.mk`, replacing `ATH79` and `ath79` with the new name
- Rename and adjust `vendor/<new_target>/src/lib/target/inc/target_<new_target>.h`

- Modify `vendor/<new_target>/src/lib/target/entity.c` so that the functions return the correct model name, id, etc.

The address of the OpenSync controller can be set either before building the package or by SSHing into AP once OpenSync is running. To set the address before building the package modify the line beginning with `CONTROLLER_ADDR` in `opensync/platform/openwrt/build/openwrt.mk` with the format `CONTROLLER_ADDR="tcp:<ip of controller>:<port>"`. Port 6640 is recomended.

To build the OpenSync Package, change directory to example and then run make `TARGET=<oepnwrt target> SDK_URL=<OpenWrt SDK>`. The sdk should be for the version of OpenWrt you are targeting and must be version 19.07.1 or later. To build OpenSync using the vendor target given in this repository run the commands below.

```
cd example
make TARGET=ATH79 SDK_URL=http://example.com/openwrt-sdk-ath79.tar.xz
```

This will output an .ipk file in the example/out/ directory.

C.2 Installing OpenSync

Installing the custom OpenSync package on OpenWrt requires these packages to also be installed:

- libev
- jansson
- protobuf
- libprotobuf-c
- libmosquitto
- libopenssl
- openvswitch
- libpcap
- hostapd a patched version of hostapd that works with Open vSwitch is required and available at `http://packages.wand.net.nz/openwrt/hostapd/`

To change the address at which OpenSync expects the OpenSync controller to be at while OpenSync is running, use the command below. Port 6640 is recommended.

```
/usr/plume/tools/ovsh u AWLAN_Node \
redirector_addr:="tcp:<IP address of controller>:<port>"
```

C.3 Running the Controller

Before the controller is run, a PostgreSQL server with a database of the type defined in `opensync-controller/schema.sql` must be running for it to connect to. The details of this database are configured in the controllers YAML based configuration file.

The controller is in the opensync-controller directory and is run with the command `./controller <config-file.yml> <port number>` —. Port 6640 is recommended.

The controller configuration file is YAML based. An example configuration file is included in the opensync-controller directory.

The controller is written in python3 and requires the following python packages to be installed:

- pyyaml
- psycopg2
- pyrad
- six

C.4 Running OpenSync

Once OpenSync has been installed on the AP and the controller is running, OpenSync can be started with the command `service opensync start`.