CloudMAC Frame Prioritization

QoS and routing of IEEE802.11 frames in a Opendaylight controlled network

CloudMAC Ram Prioritering
Prioritering and dirigering av IEEE802.11 ramar i ett OpenDaylight kontrollerat nätverk

Joakim Carlsson

Faculty of Health Science and Technology
Computer Science
30 hp
Andreas Kassler
Donald F Ross
150604

Abstract

Wireless networks are common in large organisations that can cover multiple floors and buildings. Wireless networks become expensive as they grow and more control and coordination is needed to operate and management them. This thesis describes how CloudMAC, a software defined networking solution (SDN), were implemented in OpenDaylight Hydrogen, a SDN controller. CloudMAC reduces complexity in large wireless local area networks. CloudMAC splits access points (AP) into, a physical (accesses the wireless medium) and a logical (handles the processing of data) part. These two part are then placed in different locations in a wired network. The parts are connected by making tunnels through the network. Some of the communications in wireless networks are time sensitive. Such time sensitive communication is easily disturbed during congestion. To improve CloudMAC, quality of service (QoS) was implemented. QoS was used both in the wired network and in accessing the wireless medium. Evaluations shows how to evaluate queues utilization and performance.

This thesis is submitted in partial fulfilment of the requirements for the Master's degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Joakim Carlsson

Approved, 2015-06-10

Supervisor: Andreas Kassler

Examiner: Donald F Ross

Abbreviations

AC	Access Category
AD-SAL	API Driven SAL
AIFS	Arbitration Interframe Space
API	Application Abstraction Layer
ATIM	Announcement Traffic Indication Message
BSS	Basic Service Set
CF	Contention Free
CFP	Contention Free Period
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CW	Contention Window
DCF	Distributed Coordination Function
DIFS	DCF interframe space
DLS	Direct Link Setup
DSE	Dependent Station Enablement
EDCA	HCF contention-based channel access
FIFO	First In First Out
FTP	File Transfer Protocol
HC	hybrid coordinator
HCCA	HCF controlled channel access
HCF	hybrid coordination functions
HT	High Throughput
HTB	Hierarchy Token Bucket
HTTP	Hypertext Transfer Protocol
HWMP	Hybrid Wireless Mesh Protocol
MD-SAL	Model Driven SAL
ODL	OpenDaylight
OSGI	Open Service Gateway Initiative
OVS	Open vSwitch
PCF	Point Coordination Function
PIFS	PCF Interframe Space
QoS	Quality of Service
RTT	Round Trip Time
SA	Source Association
SAL	Service Abstraction Layer
SDN	Software Defined Networking
SIFS	Short interframe space
TDLS	Tunnelled Direct Link Setup
TXOP	Transmission Opportunity
VAP	Virtual Access Point
WFA	Wi-Fi Alliance
WNM	Wireless Network Management
WTP	Wireless Termination Point

QMF	QoS management frame

Table of contents

Table of Figures

Table of tables

1 Introduction

Wireless networks are common in large organisations that can cover multiple floors and buildings. Wireless networks become expensive as they grow and more control and coordination is needed to operate and management them. This is often realised by making access points (AP) complicated and expensive. When network complexity increases, it becomes more time consuming and thereby costly to maintain them. It also becomes far easier to misconfigure such networks. They grow more rigid. A solution to complex networks are the concept of programmable networks called software defined networking [1] (SDN). In SDN, network devices give up on managing their routing, a centralised controller takes over the responsibility of controlling routing. CloudMAC [2] is a SDN solution for decreasing wireless local area networks. A decrease in complexity is achieved by decoupling the physical access to the wireless medium from the logical processing. The physical access to the medium is done by a wireless termination point (WTP). The logical processing is handled by a virtual access point (VAP). OpenDaylight [3] is a SDN controller and is a Linux Foundation collaborative open source project [4].

1.1 Project Goals

This project had four goals:

1. Evaluate OpenDaylight by implementing CloudMAC in OpenDaylight. The evaluation was to see if CloudMAC could be implemented in OpenDaylight.
2. Implement QoS in CloudMAC and evaluate performance gains of QoS.
3. Enable CloudMAC to automatically detect WTPs and VAPS. This is to make it easier to setup CloudMAC networks.
4. Make the CloudMAC OpenDaylight implementation configurable.

An illustration of where the project goals belong in the network are shown in Figure 1.

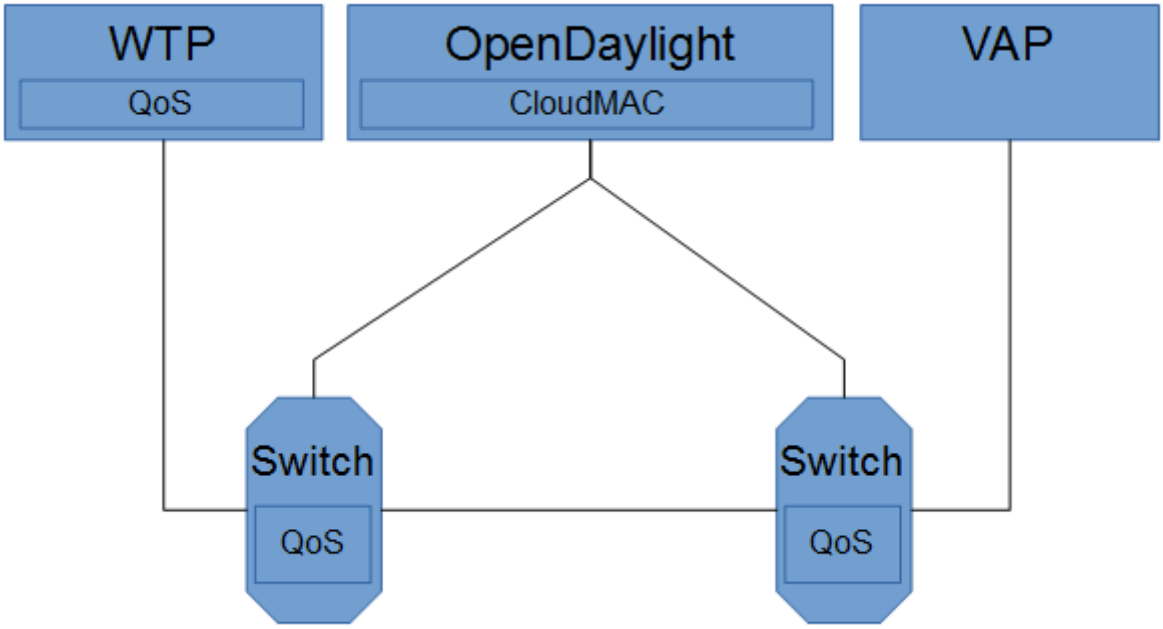

Figure 1 - Illustration of where the project goals belong in the network

1.2 Dissertation Layout

Chapter 2 will in detail go through the inner workings of CloudMAC, how an AP can be split into a WTP and a VAP, and how QoS can be implemented in switches and WTPs. All relevant technologies will also be explained in chapter 2.

Chapter 3 will go through the design of the project. An explanation for how the system was designed to automatically in detect WTPs and VAPs. Explanations for how the roles of a traditional AP is split into a WTP and VAP. The design of CloudMAC will be described as ten use cases:

1. The case of a WTP being started and connected to the CloudMAC network
2. The case of a VAP being connected to the CloudMAC network, both with and without the presence of a WTP broadcasting
3. The case of a WTP being disconnected
4. The case of a VAP being disconnected
5. The case of a station connecting
6. The case of a station trying to connect and fails to connect
7. The case of keeping connected stations connected
8. The case of a station disappearing
9. The case of handling traffic not associated with the CloudMAC network
10. The case of handling the crash of a switch

Chapter 4 will go through how the designs in chapter 3 are implemented. The WTPs and VAPs require modified drivers and the modifications and how they were implemented will be described. How CloudMAC was implemented will briefly be described.

Chapter 5 will go through three evaluations: WTP QoS, switch QoS, and connection. WTP QoS evaluation tests how well the QoS implementations hold up. Switch QoS evaluation checks that there are benefits to using QoS in switches. Connection evaluation is performed to see if there were any problems with connecting to CloudMAC.

Chapter 6 is the final chapter and will go through the conclusions and results from the evaluations, and discuss shortcomings of the project along with possible future work. A judgement on how well CloudMAC could be implemented in OpenDaylight will be given.

1.3 The results of the project

All four goals were reached. CloudMAC was successfully implemented as an OpenDaylight module, although with room for optimization. The module is configurable with a myriad of settings available to tweak the module. The detection and tracking of WTPs and VAPs works very well and makes it trivial to change the topology of networks controlled by the CloudMAC module. QoS in both switches and WTPs were worthwhile and showed significant improvements for CloudMAC.

2 Background

This chapter will describe the technologies and standards that are needed in order to understand the project. It will start with the concepts of SDN and OpenFlow, then move on to describe OpenDaylight along with some standards for wireless local area networks. It concludes by describing the technologies and tools used during the project. In Figure 2 all the concepts and where they belong in the project.

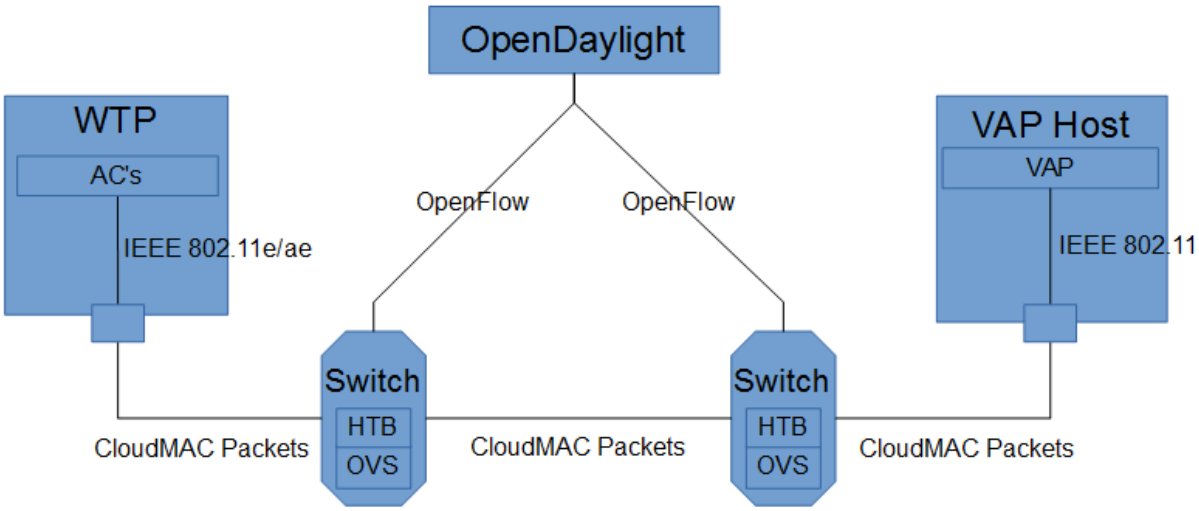

Figure 2 - Illustration of CloudMAC components

2.1 Software Defined Networking

Computer networks are heterogeneous and their size ranges from single switches to large networks with a multitude of switches and devices of varying models and manufacturers. Often the tools available are lacking. This makes it a difficult task to properly configure and maintain dynamic or even static networks. As such introducing new services can become a long and costly process and might result in the network becoming unusable if incorrectly configured. A solution to this is the concept of programmable networks called software defined networking (SDN). In programmable networks individual networking devices relinquish the right to control their own routing and instead rely on a centralised entity to instruct them how to route. With this concept, deploying new services only requires that the services are implemented in the centralised entity. Maintaining networks also becomes easier as the centralised entity is all that is needed to configure the network. In SDN the centralised entity is implemented as a software controller. There are some issues with the concept. If the controller crashes or is disconnected from the network then the entire network will become unusable. There is also the disadvantage that routing will be delayed as network devices need to transmit a copy of packets that do not match any instructions to the controller.

The controller then needs to evaluate how the traffic that the packet belongs to should be treated. Finally the controller needs to instruct network devices how to handle that traffic.

2.2 OpenFlow

In order to implement SDN networks, there needs to be a way for the network devices and the SDN controller to communicate. A common standardised protocol is preferred so that network devices can communicate regardless of model and manufacturer. OpenFlow [5] is such a protocol which allows network devices to send packets that they cannot match to the SDN controller and the SDN controller can instruct network devices how they should handle routing of packets. Instructions in OpenFlow consists of three groups of parameters; match fields, actions, and attributes. Match fields are used to match fields in packets, actions are how the device should handle packets that have been matched, and attributes are values that dictate how an instruction should be processed while matching. OpenFlow have several match fields, actions, and parameters many of which were not used for the project, only those that were used are explained. For a complete list please see [5]. Four match fields were used and those were IN_PORT, DL_SRC, DL_DST, and DL_TYPE which are explained in Table 1. The actions used were ENQUEUE, OUTPUT, DROP, and CONTROLLER which are explained in Table 2. Two attributes were used PRIORITY and HARD_TIMEOUT which are explained in Table 3. Below is an example of an OpenFlow instruction may look like:

IN_PORT=0, PRIORITY=2000, DL_TYPE=0x1337, DL_SRC=0A:0B:00:00:00:01, DL_DST=AE:BF:00:00:00:01, ACTIONS=ENQUEUE:3:1

This rule will match packets sent from a device with an MAC address of 0A:0B:00:00:00:01 to a device with an MAC address of AE:BF:00:00:00:01 with Ethernet type is 0x1337. The match will take place before any other rules with a priority lower than 2000. The packets that are matched will be queued on the second queue for port 3.

Name	Syntax	Description
Ingress Port	IN_PORT=port	Matches the port from which the packet arrived.
Ethernet Source	DL_SRC=mac	Matches the MAC address of the sender.
Ethernet Destination	DL_DST=mac	Matches the MAC address of the receiver.
Ethernet Type	DL_TYPE=type	Matches the Ethernet type.

Table 1 – List of OpenFlow Match Fields, their syntax, and descriptions.

Name	Syntax	Description
Enqueue	ENQUEUE:port:queue	Enqueues the packet in a specified queue for a port.
Output	OUTPUT:port	Enqueues the packet in the default queue for a port.
Drop	DROP	Drops the packet.
Controller	CONTROLLER	Sends the packet to the controller of the switch.

Table 2 – List of OpenFlow Actions, their syntax, and descriptions

Name	Syntax	Description
Priority	PRIORITY=priority	The priority of the rule. Rules are matched by descending order.
Hard Timeout	HARD_TIMEOUT=time	How long in seconds that the rule should exist.

Table 3 – List of OpenFlow Attributes, their syntax, and description.

2.3 OpenDaylight

OpenDaylight is a SDN controller and is a Linux Foundation collaborative open source project. OpenDaylight is designed to enable scalable SDN. As a SDN controller it is important that it stays running. Having to shut it down when adding and or removing functionality and services would not be viable in many networks. In order to eliminate the need for downtime, OpenDaylight uses a modular approach and implements functionality and services through modules. Modules are organised into bundles. OpenDaylight has two types of modules API-Driven [6] (AD-SAL) and Model-Driven [7] (MD-SAL) where SAL stand for service abstraction layer. Modules may be started, stopped, installed, or uninstalled during runtime. AD-SAL modules register for interfaces and receive instances implementing those interfaces from the controller. Those instances are then used to query network components. MD-SAL uses a more generic approach using a model where all network components are represented as a tree. To query a network component, a path through that tree to the component is needed. It is also possible to register to interfaces and receive instances of those interfaces. Interfaces can be seen as a common language that modules use to speak to each other.

2.4 IEEE802.11

IEEE802.11 [8] is a standard published by the Institute of Electrical and Electronics Engineers (IEEE) for wireless local area networks. It defines one medium access control (MAC) and multiple physical layer (PHY) specifications. It is the de facto standard for local area wireless networks. Since it is a collection of specifications it supports a myriad of data rates. For IEEE802.11n [8] it ranges from 1.0 Mbps to 600 Mbps [9]. While for a newer specification IEEE802.11ac [10] extends that range to 6.93 Gbps [10]. A very brief description of the transmission process is presented in section 2.4.1. In IEEE802.11 networks are made up of an AP and a collection of stations, this is knowns as a basic service set (BSS). The medium used by IEEE802.11 is split into several channels, the number of channels and which frequencies that are used depends on which amendments [11] that are used. Available [11] frequencies depends on which country the network is in. IEEE 802.11 is extended through the use of amendments to the original standard. Amendments may add

2.4.1 *Transmission*

The basic access methods [8] used by IEEE802.11 are a distributed coordination function [12] (DCF) known as carrier sense multiple access with collision avoidance [8] (CSMA/CA) and a point coordination function [8] (PCF). For the DCF scheme all stations that want to transmit have to go through the following process before transmitting (Figure 3):

(1) First the station has to wait for the medium to turn idle.
(2) Wait for the medium to remain idle for a period of time called short interframe space [13] (SIFS) or alternatively DCF interframe space [14] (DIFS). SIFS is used for control frames or acknowledgement frames. DIFS is used for data and management frames.
(3) The station have to wait an additional period known as a backoff. The backoff is derived from a contention window (CW). The value used for the backoff is calculated by randomizing a value between the minimum and maximum specified by the CW. If the medium turns busy during this period the countdown is stopped and resumed after the medium turns idle and a SIFS or DIFS has transpired.
(4) The station is allowed to transmit. If a collision occurs, the entire process is restarted and the minimum of the contention window is increased.

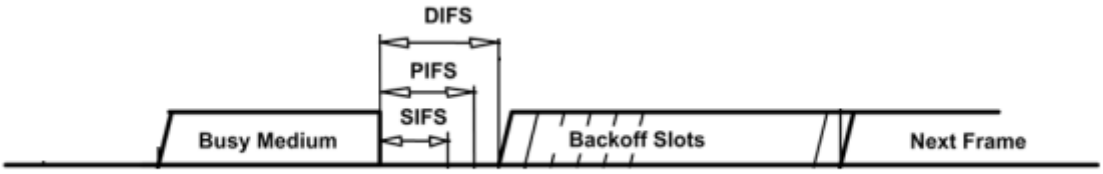

Figure 3 – Illustration of how different interframe spacing's relate to each other.[1]

PCF is for infrastructure networks and works by having a polling coordinator (PC) in the AP which determines which station has the right to transmit. The PC polls the stations in the BSS to provide them an opportunity to transmit. PCF is optional part of the IEEE802.11 standard. The process is:

(1) Wait for a poll from the PC.
(2) Transmit.

The IEEE802.11 does not define how different BSS should efficiently coordinate. PCF uses an interframe space called point interframe space (PIFS) that is smaller than DIFS. In order to not lock out DCF access from the medium there are periods where PCF will intentionally not grant medium access, this is illustrated in Figure 4.

These mechanisms for transmitting IEEE802.11 frames are flexible and can accommodate multiple BSS on the same channel though doing so will degrade performance. The more stations that want to transmit the worse throughput will be achieved as more collisions will occur.

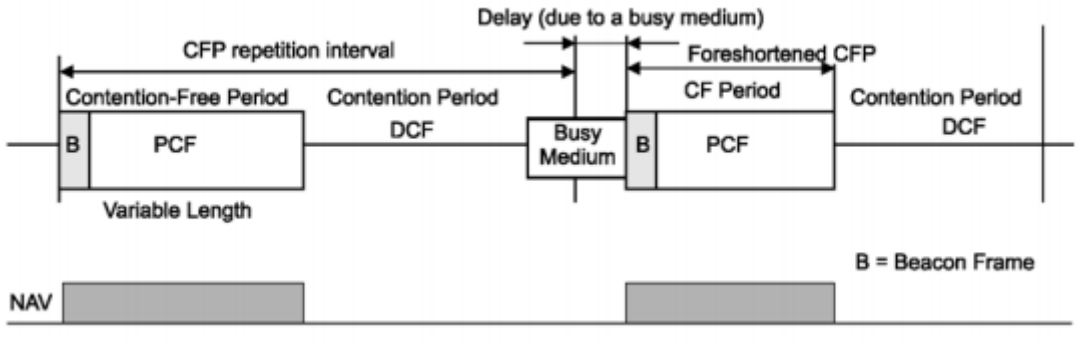

Figure 4 – Illustration of what happens during PCF and DCF.[2]

[1] Figure is modified, original figure is from "IEEE Std 802.11™-2012 (Revision of IEEE Std 802.11-2007)" figure 9.3.
[2] Figure is from "IEEE Std 802.11™-2012 (Revision of IEEE Std 802.11-2007)" figure 9.15.

2.4.2 *Frame Structure*

In IEEE802.11 data is grouped into frames, the structure of the frames are seen in Table 4. Not all fields will be described, only those that have some relevance to the project, for a full description of all fields see [8]. The relevant fields are frame control (section 2.4.2.1), the address field (section 2.4.2.2), and retry (section 2.4.2.3).

Bytes	2	2	6	6	6	2	6	2	4	0-7951	4
Name	Frame Control	Duration/ID	Address 1	Address 2	Address 3	Sequence Control	Address 4	QoS Control	HT Control	Frame Body	FCS

Table 4 - IEEE802.11 Frame Format

2.4.2.1 Frame Control Field

The frame control field contains eleven fields split over sixteen bits. The relevant fields are type, subtype, to DS, and from DS. The structure can be seen in Table 5.

	Least Significant Bit						Most Significant Bit				
Bit #	0,1	2,3	4,-,7	8	9	10	11	12	13	14	15
Name	Protocol Version	Type	Subtype	To DS	From DS	More Fragments	Retry	Power Management	More Data	Protected Frame	Order

Table 5 – IEEE802.11 Frame Control Field Format

Type: There are three types of frames management, control, and data. Management frames manage the connection. Control frames control the transfer of data. Data frames contain data. A subset of management frames can be seen in Table 6.

Name	Description
Beacon	Announces a BSS presence.
Association Request	Asks a BSS to allocate resources and provides information about itself.
Association Response	Response indicating that if the request was accepted or rejected
Deassociation Request	Tells a BSS that it can release resources associated with that station.
Authentication	Used to begin secure communication.
Deauthentication	Used to end secure communication.
Probe Request	Asks all or a specific nearby BSS to respond with information about itself.
Probe Response	Response with information about the capabilities of a BSS.

Table 6 – Relevant IEEE802.11 management frames

Subtype: Identifies which purpose the frame has.

To DS, From DS: Is used to figure out the meaning of the address fields see section 2.4.2.2.

2.4.2.2 To DS & From DS & Address 1, 2, 3, 4

The meaning of the address fields are dependent on the values of the To DS and From DS. DS stands for distributed system and is a set of interconnected BSS. If to DS is set to one that means that the frame is being sent into a DS. If from DS is set it means that it is sent from a DS. If both are set it means that the frame is being passed along inside a DS. Their meaning is explained in Table 7 and their mapping in Table 8.

Name	Description
Source	The MAC address of the original sender.
Destination	The MAC address of the final destination.
Receiver	The MAC address of an AC that is part of a DS and that is the destination of the frame.
Transmitter	The MAC address of an AC that is part of a DS and that transmitted the frame.

Table 7 - Meaning of the IEEE802.11 address field designations

To DS	From DS	Address 1	Address 2	Address 3	Address 4
0	0	Destination	Source	BSSID	Not present
0	1	Destination	BSSID	Source	Not present
1	0	BSSID	Source	Destination	Not present
1	1	Receiver	Transmitter	Destination	Source

Table 8 – IEEE802.11 Address field interpretation

2.4.2.3 Retry

Many frames require acknowledgement of reception. If no such acknowledgement was received this field is set to one and the frame is transmitted again. A value of zero means that no retry was needed.

2.4.3 *IEEE802.11e*

IEEE802.11 have a limitation on the level of QoS it can provide. The only QoS present is that acknowledgement and control frames are treated better than other frames. This limited QoS means that stations are allowed to send response frames within a short interframe space (SIFS) after the last bit of a frame was received. Management and data frames are treated equally even though several management frames are time sensitive (see Table 10 for some examples). If a channel is busy they may be delayed to the point where a station cannot respond before the response is obsolete. How sensitive the frames are depends on the driver implementation. Not all data frames are handled equally, some applications perform better if the delay experienced is low while others perform well no matter what the delay. Examples of applications that require low delay are voice-chat, video-chat, and online gaming. Examples of an application that have a higher tolerance for delay is FTP [15]. IEEE802.11e [8] is an amendment to IEEE802.11 [8] that adds QoS through the introduction new hybrid coordination functions [8] (HCF), HCF contention-based channel access (EDCA) and HCF controlled channel access (HCCA).

EDCA [8] defines four access categories (AC) (see Table 9) and introduces arbitration interframe space (AIFS) in favour of DIFS. The ACs act semi-independently, almost like separate stations, each with its own CW and AIFS. The process of transmitting is changed so that in step 2 of section 2.4.1 AIFS is used instead of DIFS and step 4 where the concept of a transmission opportunity (TXOP) is introduced. TXOP is a time frame in which an AC is allowed to transmit frames.

HCCA is for infrastructure networks and works by having a hybrid coordinator (HC) in the AP which determines which station has the right to transmit. The HC polls the stations in the BSS giving them the opportunity to transmit. The process is:

(1) Wait for a poll from the HC.
(2) Transmit during a TXOP.

PCF uses an interframe space called interframe space that is smaller than DIFS. In order not to lock out DCF access from the medium there are periods where HCCA will intentionally not grant medium access. During this time stations may operate in accordance with EDCA.

Highest Priority			**Lowest Priority**
Voice	Video	Best Effort	Background

Table 9 – QoS access category priority

Frame Name
Association Request
Association Response
Probe Request (addressed to a specific BSS)
Authentication
Deauthentication

Table 10 - Sample of time sensitive IEEE802.11 frames

2.4.4 *IEEE802.11ae*

IEEE802.11ae [16] is an amendment to IEEE802.11 that defines policies for management frame prioritization and a method for communicating the prioritization policy used. The policies define under which AC a management frame should be queued. Polices map subtypes to ACs. The motivation for this separation is that not all management frames are equally important or time sensitive. If all management frames are queued under the voice AC it can interfere with other traffic. Policies are announced through beacon, association response, reassociation response, and probe response frames. The sharing of policies is so that all stations in a BSS can operate under the same conditions in terms of QoS.

2.5 CloudMAC

CloudMAC [2] makes it possible to use simple access points even in large complex wireless networks. This is achieved by moving the processing of wireless frames to a virtual access point (section 2.5.3) located elsewhere in the network. It is a cloud solution. In CloudMAC access points (AC) are split in two, a wireless termination point (WTP) and a virtual access point (VAP). WTPs handle the reception, transmission of IEEE802.11 frames, and acknowledgement of frame reception. VAPs handle the processing of IEEE802.11 frames and generate responses. Most switches can only route Ethernet packets, as such IEEE802.11 frames cannot be directly routed without prepending an Ethernet or equivalent header. In order to get the IEEE802.11 frames from a WTP to a VAP a CloudMAC header is prepended. The CloudMAC header contains fields extracted from the IEEE802.11 frame. For specifics of the CloudMAC header read section 2.5.2. Figure 5 shows an illustration of two clients that are connected to two VAPs through a WTP. The CloudMAC controller has set up two tunnels labelled a and b which are separated and the clients cannot directly communicate with each other. The lines labelled c represents control channels between the controller and the switches. This section will describe how these parts operate.

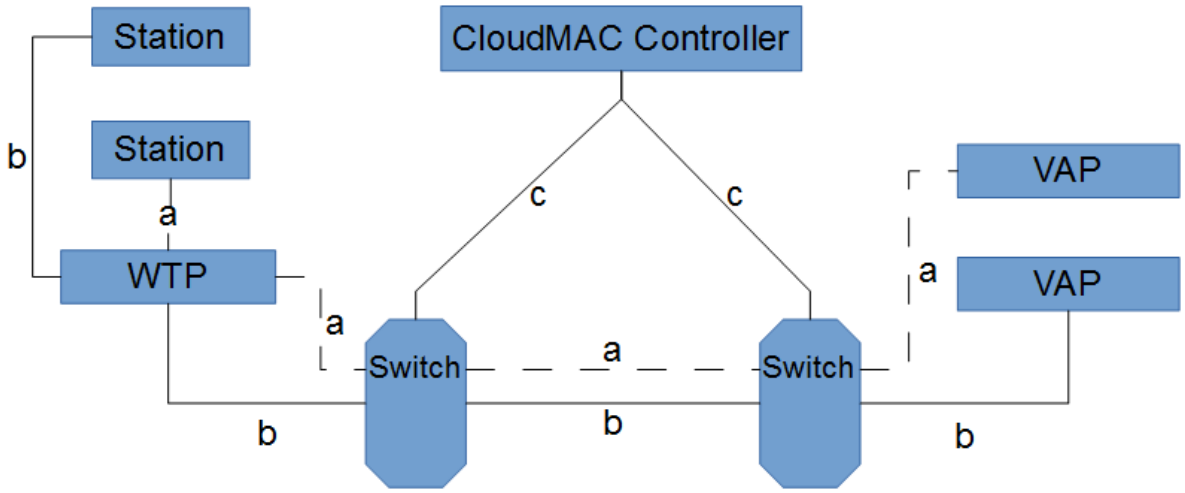

Figure 5 - Illustration of CloudMAC

2.5.1 *Radiotap*

Radiotap [17] is a header format for storing captured wireless IEEE 802.11 [8] frames and associated information about the state of the reception. Radiotap has a variable length header. To know which fields are present the present field has to be examined. There are four default fields that are always present;

(1) Version – The version of Radiotap, the current version is zero.
(2) Padding – Unused in version zero, is used to align the rest of the fields to natural word boundaries.
(3) Length – The entire length of the Radiotap header.
(4) Present – A bitmask that indicates which extra fields are present.

Table 11 shows a Radiotap header with no extra fields present while Table 12 shows a Radiotap header with rate and antenna signal present;

- Rate – The rate that the IEEE802.11 frame was received at. That can also be the speed at which the IEEE802.11 frame should be transmitted.
- Antenna Signal – the antenna signal of the transmission can mean either the signal strength the IEEE802.11 frame was received at or the strength it should be transmitted at.

Version	Padding	Length	Present
1 byte	1 byte	2 bytes	4 bytes

Table 11 - Radiotap header with no extra fields present

Version	Padding	Length	Present	Rate	Antenna Signal
1 byte	1 byte	2 bytes	4 bytes	1 byte	1 byte

Table 12 - Radiotap header with rate and antenna signal present

2.5.2 *CloudMAC Packet Format*

A CloudMAC packet starts with a CloudMAC header which has two different forms. Which is present depends on whether the packet is inbound or outbound. After this header follows a Radiotap header and body.

Inbound: If the packet is heading towards a Virtual Access Point (VAP), it looks the same as an ordinary Ethernet header as can be seen in Table 13.

Destination	Source	Ethernet Type
6 Byte	6 Byte	2 Byte

Table 13 - CloudMAC inbound packet format

Outbound: If the packet is heading towards a wireless termination point (WTP) the structure is as seen in Table 14. Signal is the signal strength the WTP should transmit at. Rate is the rate to transmit at and VAP identification is the truncated MAC address of a VAP where the first two bytes have been removed.

Destination	Signal	Rate	VAP Identification	Ethernet Type
6 Byte	1 Byte	1 byte	4 Byte	2 Byte

Table 14 - CloudMAC outbound packet format

2.5.3 *Virtual Access Point*

For the project the virtual access points are instances of hostapd. Hostapd [18] is a user space daemon access point server. Outgoing IEEE802.11 frames are turned into CloudMAC packets and IEEE802.11 frames are extracted from incoming CloudMAC frames.

2.5.4 *Wireless Termination Point*

A WTP is a device that captures IEEE802.11 frames and encapsulates them into CloudMAC packets and extracts IEEE802.11 frames from received CloudMAC frames and transmits them. When IEEE802.11 frames are transmitted, the WTP looks at the signal and rate in the CloudMAC header and performs the transmission according to signal and rate. Due to the acute time sensitivity of IEEE802.11 acknowledgement frames, the WTP can also be configured to acknowledge frames instead of VAPs. Acknowledgement frames are sent as a verification that a frame was received. The verification is expected to be transmitted within one SIFS. If no acknowledgement is received within one SIFS then the frame will be transmitted. SIFS are in the range of tenths of microseconds and are much smaller than the time required to route and process IEEE802.11 frames.

2.6 Linux Hierarchy Token Bucket

Hierarchy token bucket (HTB) [19] [20] allows, as the name suggests, the creation of a hierarchy of classes which gives fine grained control over traffic. The hierarchy can be thought of as multiple trees of classes. Each class can be configure semi-independently of other classes and bandwidth can be borrowed from other classes in the same tree. Classes all have their own queueing scheme with the default being first in first out (FIFO). To create HTB instances a tool called traffic control (TC) is used. Table 15 lists all options available when creating a HTB instance. Table 16 lists all the options when creating classes. The process of creating and setting up queues will be described below.

Name	Syntax	Description
Parent	Parent MAJOR:MINOR or root	This specifies where the HTB instance should be placed. If a major and minor ID is provided then it is attached to a class. If root is provided then it will be attached to the root of the interface.
Handle	Handle MAJOR:	Specifies a handle that can be used to reference an HTB instance when adding classes.
Default	Default MINOR	This specifies which class unmatched traffic should be sent to.

Table 15 - List of HTB TC parameters, syntax, and descriptions

Name	Syntax	Description
Parent	Parent MAJOR:MINOR	Specifies where to attach a class.
Class ID	Classid MAJOR:MINOR	Specifies the handle that can be used when child classes are added to the class.
Priority	Prio PRIORIY	Specifies the priority of a class. Classes are handled in an ascending order.
Rate	Rate BITRATE	The guaranteed throughput of the class in bits per second.
Ceiling	Ceil BITRATE	The maximum throughput the class can achieve by borrowing from other classes.
Burst	Burst BYTES	The number of bytes the class may transmit while borrowing bandwidth.
CBurst	Cburst BYTES	The number of bytes the class may transmit at the maximum speed of the interface.

Table 16 - List of HTB class parameters, syntax, and descriptions

Suppose that we want to create three different queues for interface ETH0.2. All three queues should be given 30 Mbps and queue 1 should not be allowed to borrow from the other two but they can borrow from queue 1. Queue 2 should be given priority over queue 3 when borrowing bandwidth. By default all packets should be queued under queue 1. The setup for this can be seen in Figure 6.

```
(1) tc qdisc add dev eth0.2 root handle 1: htb default 1
(2) tc class add dev eth0.2 parent 1: classid 1:1 htb rate 90000kbps ceil 90000kbps
(3) tc class add dev eth0.2 parent 1:1 classid 1:1 htb rate 30000kbps ceil 30000kbps
(4) tc class add dev eth0.2 parent 1:1 classid 1:2 htb prio 1 rate 30000kbps ceil 90000kbps
(5) tc class add dev eth0.2 parent 1:1 classid 1:3 htb prio 2 rate 30000kbps ceil 90000kbps
```

Figure 6 - Example of HTB tree creation

(1) Creates an HTB instance and places it at the root of interface ETH0.2, gives it a handle, and tells it to place all traffic by default in queue 1.

(2) Creates a new tree below root HTB instance and configure it to give a guaranteed throughput of 90 Mbps and no borrowing allowed (it would not borrow in any case seeing how it is the root of a tree.).

(3) Creates queue 1, attaches it to the root of the tree, and configures it so that it has a guaranteed throughput of 30 Mbps and is not allowed to borrow.

(4) Creates queue 2, attaches it to the root of the tree, configures it so that it has a guaranteed throughout of 30 Mbps, can borrow up to 60 Mbps from other classes, and has priority 1.

(5) Creates queue 3, attaches it to the root of the tree, configures it so it has a guaranteed throughout of 30 Mbps, can borrow up to 60 Mbps from other classes, and has priority 2.

These steps result in the tree in Figure 7. This setup allows queue 1 to use up to 30 Mbps, queue 2 to use up to 90 Mbps, and queue 3 to use up to 90 Mbps. The amount they can use will depend on the usage of the other queues.

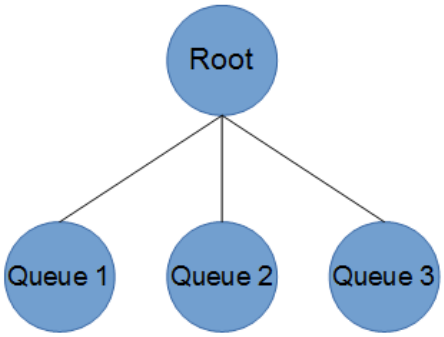

Figure 7 - Illustration of a HTB tree with a root class and three leaf classes

2.7 Open vSwitch

Open vSwitch [21] (OVS) is an easily configured software multilayer network switch platform. OVS supports multiple network management protocols and can be connected to SDN controllers. OVS supports the creation of queues for the purpose of QoS. OVS allow you to set up virtual switches allowing physical switches to be split up into multiple isolated or interconnected virtual switches. Built-in tools allow easy creation and inspection of flow rules.

2.7.1 *OVS-VSCTL*

OVS-VSCTL [22] is a program that provides an easy to use high level interface to its underlying database. This program can be used to set up and configure virtual switches. To create a virtual switch with three ports one would go through the process in Figure 8.

```
(1) OVS-VSCTL add-br br0
(2) OVS-VSCTL add-port eth1
(3) OVS-VSCTL add-port eth2
(4) OVS-VSCTL add-port eth3
```

Figure 8 - Example of virtual switch creation

(1) Creates a virtual switch named br0. The switch does not yet have any ports attached to it.

(2) Attaches interface eth1 to br0 as a port.

(3) Attaches interface eth2 to br0 as a port.

(4) Attaches interface eth3 to br0 as a port.

If QoS is desired then a QoS instance needs to be created and attached to a port. The same QoS instance may be added to multiple ports. When creating QoS instances or a queue instance, an ID is returned that uniquely identifies that instance. Keeping track of these IDs when creating several QoS with several queues are not mandatory. OVS-VSCTL allows multiple commands to be chained together by separating commands on a single line with " -- ". When commands are chained it is possible to name them and the name can be used in place of IDs. To name a command start the command with "—id=@name". The names disappear after all commands have finished. Commands can be split up over several lines if "\" is inserted before line breaks. To enable QoS for the previous example run the command in Figure 9.

```
(1) ovs-vsctl -- set port eth1 qos=@newqos \
(2)           -- --id=@newqos create qos type=linux-htb \
                   queues:0=@q1 queues:1=@q2 other-config:max-rate=10000000 \
(3)           -- --id=@q1 create queue other-config:priority=1 \
(4)           -- --id=@q2 create queue other-config:priority=7
```

Figure 9 – Example of QoS creation in OVS

(1) The QoS named “newqos” will be attached to port eth1 when it is created.

(2) A QoS instance named “newqos” is created. The queues “q1” and “q2” will be attached once created. IT is configured to allow a bandwidth useage up to 100 Mbps.

(3) A queue instance named “q1” is created and given a priority of 1.

(4) A queue instance named “q2” is created and given a priority of 7.

For a full list of commands please see [22].

2.7.2 *OVS-OFCTL*

OVS-OFCTL [23] is a program to manage OVS or any OpenFlow switch. It makes it easy to control and monitor routing of switches. In order to create a new flow rule OVS_OFCTL can be used as follows: “ovs-ofctl add-flow br0 ‘in_port=0, actions=drop’” which would create a rule to drop all traffic entering the switch named “br0” from port 0. To list all flow rules along with statistics of the rules of a switch “ovs-ofctl dump-flows br0” can be used. To delete all flows of a switch “ovs-ofctl del-flows br0” can be used.

For a full list of commands please see [23].

2.8 Chapter Summary

This chapter gave a brief description of the project and the tools and technologies used during the project. It begins by introducing the concept of SDN which touches OpenFlow and OpenDaylight followed by IEEE802.11 and how CloudMAC relates to IEEE802.11 and ends with a quick overview of HTB and Open vSwitch.

3 Design

A simple and easily configurable system was the one key design goal of the project, in striving for a system that will automatically detect new VAPs, and WTPs, and track their presence in case of crashes. Another goal was to be able to accommodate time sensitive IEEE802.11 frames. If the network is congested, two things can happen that can negatively affect those frames, first by the buffers in the network might be too long and queuing time can be increased, secondly packets might be dropped. If packets are dropped then the continuous communication between the stations and VAPs will break down resulting in a disconnection. The same can happen if the time it takes for frames to go between a station and a VAP becomes too high then stations and VAPs will conclude that the frame was lost. The solution to this is QoS and will be described section 3.2.

The CloudMAC controller is designed as an AD-SAL OpenDaylight module. How it handles unmatched packets, routing of CloudMAC traffic, and automatic detection of WTPs and VAPs is described in section 3.6.

The CloudMAC daemon needs to handle on demand acknowledgment of IEEE802.11 frames and announce the WTPs presence in the network to the CloudMAC module. How this is designed is described in section 3.3.

Figure 10 shows where all the components belong, how they relate to each other, and lists their main responsibilities.

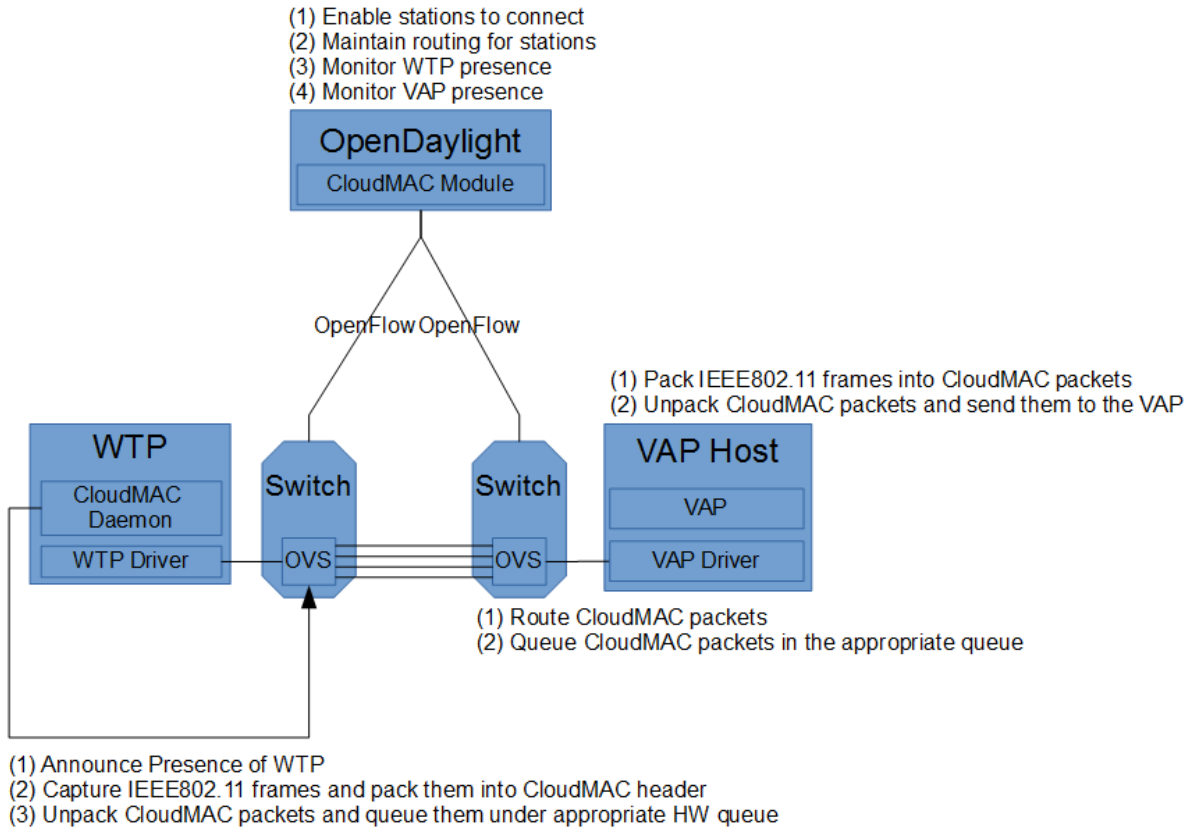

Figure 10 - Illustration of CloudMAC network components and their roles

3.1 OpenFlow

In order to tunnel the CloudMAC packets from WTPs to VAPs there needs to be rules that route the packets from one port to another in the switches. This means there is a need to match which port a packet arrived on (*INGRESS_PORT* match field) and which Ethernet type it is (*DL_TYPE* match field). In addition to just this routing there needs to be isolation between connected stations for security reasons such as shared folders. To achieve this the Ethernet source (*DL_SRC* match field) and destination (*DL_DST* match field) are used to filter what is matched. In order to queue packets on a specific port and queue the *ENQUEUE* action is needed. In order to periodically sample the traffic going through the tunnels the CONTROLLER action is needed. Sending a copy of every packet to the controller would needlessly put stress on the controller. Therefor two rules are used, they are designed such that the first one is active (the rules *PRIORITY* attribute is higher) at first but timeout (*HARD_TIMEOUT* attribute) faster than the second. Only the second rule sends copies to the controller. Some traffic that WTP captures, belongs to unrelated BSS and is dropped on the port where it was received, this means the *DROP* action is needed.

3.2 Switch QoS Queues

To enable QoS in the network switches, queues were created through Open vSwitch by a script. Four queues for each network port in order to set up four quality levels. The queueing scheme was Linux-HTB. HTB was configured so that all four queues share a common bandwidth of 300 Mbps while having differing priority. The configuration of HTB was done through OVS-VSCTL. The following command was run for each port of the switches the script in Figure 11.

```
ovs-vsctl -- set port eth1 qos=@newqos \
                  -- --id=@newqos create qos type=linux-htb queues:0=@q1 queues:1=@q2
queues:1=@q3 queues:1=@q4 other-config:max-rate=300000000 \
                  -- --id=@q1 create queue other-config:priority=1 \
                  -- --id=@q2 create queue other-config:priority=2 \
                  -- --id=@q3 create queue other-config:priority=3 \
                  -- --id=@q4 create queue other-config:priority=7
```

Figure 11 - Example creating four queues in OVS

This script sets up four queues with differing priorities, limits the throughput to 300 Mbps, and sets up the order of the queues. Table 17 shows an easier to read version where the queues are listed and together with the QoS class and priority. The rationale for having background as queue index 0 is so that flow rules that do not specify a queue will queue under the background queue.

Queue Index	QoS Class	Priority
0	Background	7
1	Voice	1
2	Video	2
3	Best Effort	3

Table 17 - Switch Queue QoS Classes

3.3 CloudMAC Daemon

The daemon runs on all WTPs and has four main features:

(1) Continuous announcements of its presence to the network.
(2) Continuous announcements of which IP it can be configured on.
(3) On demand start acknowledging IEEE802.11 frames.
(4) On demand stop acknowledging IEEE802.11 frames.

The need for the CloudMAC daemon to periodically announce the presence of a WTP is so that the CloudMAC module can automatically detect WTPs and track them. The packets will not match any rules in the switches and will be sent to the OpenDaylight controller. In order to provide the CloudMAC module with a way of configuring the daemon it listens on TCP port 1999 for ASCII commands and this IP address need to be announced. For acknowledgement of IEEE802.11 frames one command is needed one for starting and one for stopping. In order to start the following command "*lease mac duration*" needs to be issued. The commands are issued over telnet by the CloudMAC module when it detects a connecting station. Both commands require a MAC address to be specified, but the starting command also has a parameter for a duration. The duration is so that in the event of a crash or restart of the CloudMAC module the daemon will after a short while reset itself. This means that the CloudMAC module continuously needs to extend the lease. The need for WTPs to acknowledge IEEE802.11 frames instead of the VAPs comes from the fact that the delay introduced by the network means that clients will needlessly retransmit frames believing that they never arrived. Apart from the waste of bandwidth, this lead in testing to clients deciding to disconnect due to a high number of retransmissions.

3.4 VAP Driver

The VAP driver modifications are quite simple overall, for outgoing IEEE802.11 frames it adds the CloudMAC header and depending on the frame type and subtype maps frames to one of the four CloudMAC Ethernet types. The mapping can be seen in Table 18. The mapping is chosen so that frames that are time sensitive are given priority over those that are less sensitive. As such management frames are given the highest priority with the exception for probe responses which was deemed slightly less important as per the default policy introduced in IEEE 802.11ae [16]. For incoming CloudMAC packets it simply removes the CloudMAC header. All control frames are mapped to the video AC (Ethernet type 0x1338). All data frames are mapped to the background AC (Ethernet type 0x1336). There is one exception data frames with the sub type data are mapped to the best effort AC (Ethernet type 0x1337). For management frames the policy shown in Table 18 was used, it is a very simplified version of the default QoS management frame (QMF) policy in [16]. Most of the sub types are action frames [24] [25]. Action frames are used to trigger a desired action in a stations or APs. The rationale for simplifying the policy was that most of the management frame sub types were not used. Also a more simple policy is easier to implement. As such the mapping of unused frame sub types were inconsequential and were mapped to the voice AC. Table 18 shows the AC, Ethernet type, and switch queue index mapping for all IEEE 802.11 frames in IEEE 802.11ae. The switch queue mappings are as follows: Background (index 0), Voice (index 1), Video (index 2), and Best Effort (index 3).

Management Frame Sub Type	AC Mapping	Ethernet Type Mapping	Switch Queue Index Mapping
(Re)Association Request/Response	Voice	0x1339	1
Probe Request (individually addressed	Voice	0x1339	1
Probe Request (group addressed)	Voice	0x1339	1
Probe Response	Best Effort	0x1337	3
Timing Advertisement [26]	Voice	0x1339	1
Beacon, ATIM, Disassociation, Authentication, Deauthentication	Voice	0x1339	1
Spectrum management [27]	Voice	0x1339	1
Spectrum management-channel switch announcement	Voice	0x1339	1

QoS	Voice	0x1339	1
Direct Link Setup [8] (DLS)	Voice	0x1339	1
Block Ack	Voice	0x1339	1
Public	Voice	0x1339	1
Public-DSE [28] deenablement, extended channel switch announcement	Voice	0x1339	1
Public-measurement pilot [29]	Voice	0x1339	1
Public-TDLS [30] Discovery Response	Voice	0x1339	1
Radio measurement [31]	Voice	0x1339	1
Fast BSS Transition [32]	Voice	0x1339	1
HT	Voice	0x1339	1
SA Query [33]	Voice	0x1339	1
Protected Dual of Public Action	Voice	0x1339	1
Protected Dual of Public Action—extended channel switch announcemen	Voice	0x1339	1
wireless network management (WNM)	Voice	0x1339	1
Unprotected WNM	Voice	0x1339	1
Mesh [34] Action [35]-HWMP Mesh Path Selection	Voice	0x1339	1
Mesh Action-Congestion Control	Voice	0x1339	1
Mesh Action	Voice	0x1339	1
Multihop [36] Action	Voice	0x1339	1
Self Protected [37]	Voice	0x1339	1
Reserved (used by Wi-Fi Alliance (WFA))	Voice	0x1339	1
Vendor-specific Protected	Voice	0x1339	1
Vendor-specific	Voice	0x1339	1

Table 18 – The IEEE 802.11 QoS management frame (QMF) policy used in the project

3.5 WTP Driver

The WTP driver modifications are quite simple overall. For captured (incoming) IEEE802.11 frames it adds the CloudMAC header and depending on the frame type and subtype, maps frames to one of the four CloudMAC Ethernet types; 0x1336, 0x1337, 0x1338, and 0x1339. The mapping is the same as for the VAP driver. For outgoing CloudMAC packets the Ethernet type is examined and mapped to one of four different hardware queues before being transmitted. The mapping can be seen in Table 19. In order to make it easier to detect and track VAPs, captured beacon frames are dropped.

Ethernet Type	Hardware Queue	AC
0x1336	3	Background
0x1337	2	Best Effort
0x1338	1	Video
0x1339	0	Voice

Table 19 – Ethernet to hardware queue and AC mapping

3.6 OpenDaylight CloudMAC Module

The CloudMAC module is designed to accommodate multiple use cases described below. First all cases will be listed with a brief description and then discussed in detail with step by step instructions. After the use cases a description of all the parameters of the CloudMAC module that can be configurable.

Use Case 1 – tracking WTPs: This case describes what should happen when a WTP is started and connected to the network.

Use Case 2 – tracking VAPs: This case describe how a newly connected VAP is handled and the difference of what happens if there is a WTP not broadcasting any beacon frames and if there is not.

Use Case 3 – WTP Disappears: This case describes what happens if a WTP for some reason stops sending announce packets. If no station is connected through the WTP then only steps one and two apply.

Use Case 4 – VAP Disappears: This case describes what happens when a VAP disappears.

Use Case 5.1 – Station Connects: This case describes what happens when a station tries to connect and succeeds.

Use Case 5.2 – Station Tries to Connect: This case describes what happens when a station tries to connect and fails.

Use Case 6 – Station Remains Connected: This case describes what happens in order to keep stations connected. Steps labelled #.a and #.b take place in parallel.

Use Case 7 – Station Disconnects: This case describes what happens when a station disconnects or crashes.

Use Case 8 – Cross Traffic: This case describes how traffic generated by stations not associated with CloudMAC should be treated.

Use Case 9 – Switch Crash: This case describes what happens before, during, and after a switch crashes while there is an active VAP allocation going through it.

3.6.1 *Case 1 – Tracking WTPs*

When a WTP is connected to the network it will continuously generate announce packets. These packets will not be matched by any flow rules and will be forwarded to the controller for evaluation. If the WTP is not known by the controller it will be added to a list of known WTPs otherwise it will be noted that the WTP is still active. This process can be described in the following steps:

(1) WTP:1 sends a announce packet to switch:1.
(2) Switch:1 does not have any flow rules that match the announce packet.
(3) Switch:1 sends the announce packet to the controller.
(4) WTP:1 is not known and the controller adds WTP:1 to a list of known WTPs.
(5) WTP:1 sends an additional announce packets. WTPs will periodically generate announce packets.
(6) Switch:1 does not have any flow rules that match the announce packet.
(7) Switch:1 sends the announce packet to the controller.
(8) The controller resets the expiration of the WTP:1.
(9) Go to step 5.

A schematic illustration of the process can be seen in Figure 12.

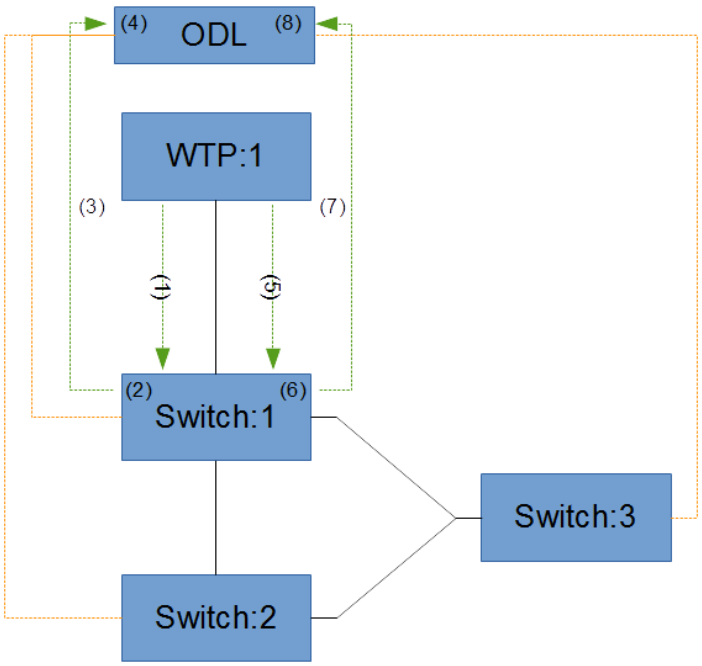

Figure 12 – Use Case 1: Tracking WTPs

3.6.2 *Case 2 – Tracking VAPs*

VAPs will continuously generate beacon frames. These frames will at first not match any rules and will be forwarded to the controller. If the VAP is not known it will be added to a list of known VAPs otherwise it will be noted that the VAP is still active. Beacon frames will temporarily be routed to WTPs in order to announce the presence of CloudMAC. WTPs should only transmit beacon frames for one VAP at a time. When the temporary routing end, the process will begin anew. The process can be described as two different sets of steps depending on whether WTP is transmitting beacon frames.

3.6.2.1 Free WTP available

This is the steps that shows what happens when there is a WTP that does not transmit any beacon frames.

(1) VAP:1 sends beacon frames to switch:2.
(2) Switch:2 does not have any flow rules that match the beacon frame.
(3) Switch:2 sends the beacon frame to the controller.
(4) VAP:1 is not known and the controller adds VAP:1 to a list of known VAPs.
(5) A free WTP is available.
(6) Controller issues routing command to switch:2.
(7) Controller issues routing command to switch:2.
(8) VAP:1 sends additional beacon frames to switch:2.
(9) Switch:2 routes the beacon frame to switch:1.
(10) Switch:1 routes the beacon frame to WTP:1.
(11) WTP:1 transmits the beacon frames.
(12) Routing for switch:1 times out.
(13) Routing for switch:2 times out.
(14) VAP:1 sends additional beacon frames to switch:2.
(15) switch:2 does not have any flow rules that match the beacon frame.
(16) switch:2 sends the beacon frame to the controller.
(17) controller resets the expiration of VAP:1.
(18) Go to step 5 if a free WTP is available otherwise 5 of section 3.6.2.2.

A schematic illustration of the process can be seen in Figure 13.

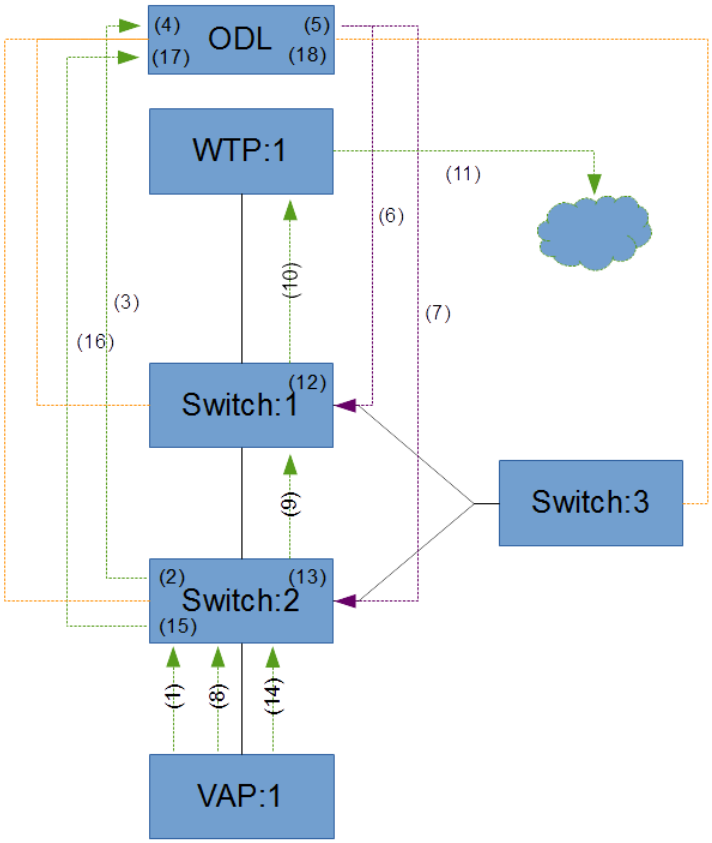

Figure 13 - Use Case 2.1: Tracking VAPs

3.6.2.2 No free WTP

These are the steps that show what happens when there is no WTP available to transmit beacon frames. A WTP is considered available if it not transmitting beacon frames.

(1) VAP:1 sends a beacon frame to switch:2.
(2) Switch:2 does not have any flow rules that match the beacon frame.
(3) Switch:2 sends the beacon frame to the controller.
(4) VAP:1 is not known and the controller adds VAP:1 to a list of known VAPs.
(5) No free WTP is available.
(6) Controller issues blocking rule to switch:2.
(7) VAP:1 sends beacon frames to switch:2.
(8) Switch:2 drops the beacon frames.
(9) Block rule in switch:2 times out.
(10) Switch:2 receives a beacon frame.
(11) Switch:2 does not have any flow rules that match the beacon frame.
(12) The beacon frames are sent to the controller.
(13) Controller resets the expiration of the VAP.
(14) Go to step 5 of section 3.6.2.1 if a free WTP is available otherwise 5.

A schematic illustration of the process can be seen in Figure 14.

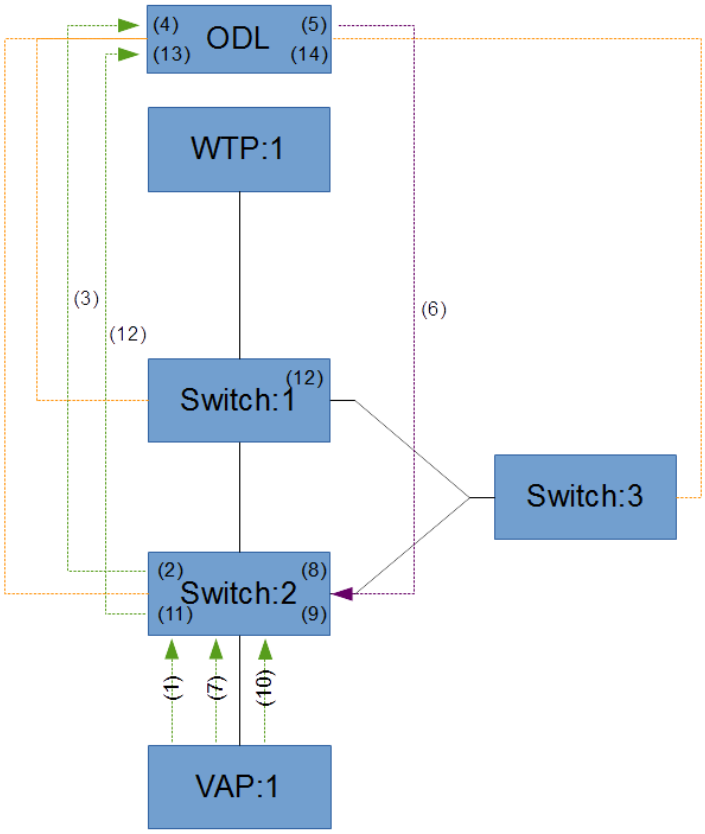

Figure 14 - Use Case 2.2: Tracking VAPs

3.6.3 *Case 3 – WTP Disappears*

If a WTP crashes or is disconnected from the network, the network will detect the disappearance and remove the WTP from the list of known WTPs. This happens when there has been a configurable interval of time without any announce packets received by the controller. The routing rules of switches will timeout and disappear as well and any stations will need to wait for these timeout to occur before reconnecting. This process can be described in the following steps:

(1) WTP:1 stops generating announce packets.
(2) VAP:1 sends frames to switch:2.
(3) Switch:2 routes frames to switch:1.
(4) Switch:1 routes frames to the port on which WTP:1 was connected to. The frames are never transmitted due to the Ethernet cable being disconnected or because of a crash of WTP:1.
(5) Station:1 sends frames to WTP:1.
(6) Station:1 detects that it can no longer communicate with VAP:1.
(7) VAP:1 detects that it can no longer communicate with station:1.
(8) Routing for switch:1 times out.
(9) Routing for switch:2 times out.
(10) VAP allocation times out.
(11) Acknowledgement lease times out.
(12) WTP:1 expires and is removed from the list of known WTPs.

A schematic illustration of the process can be seen in Figure 15.

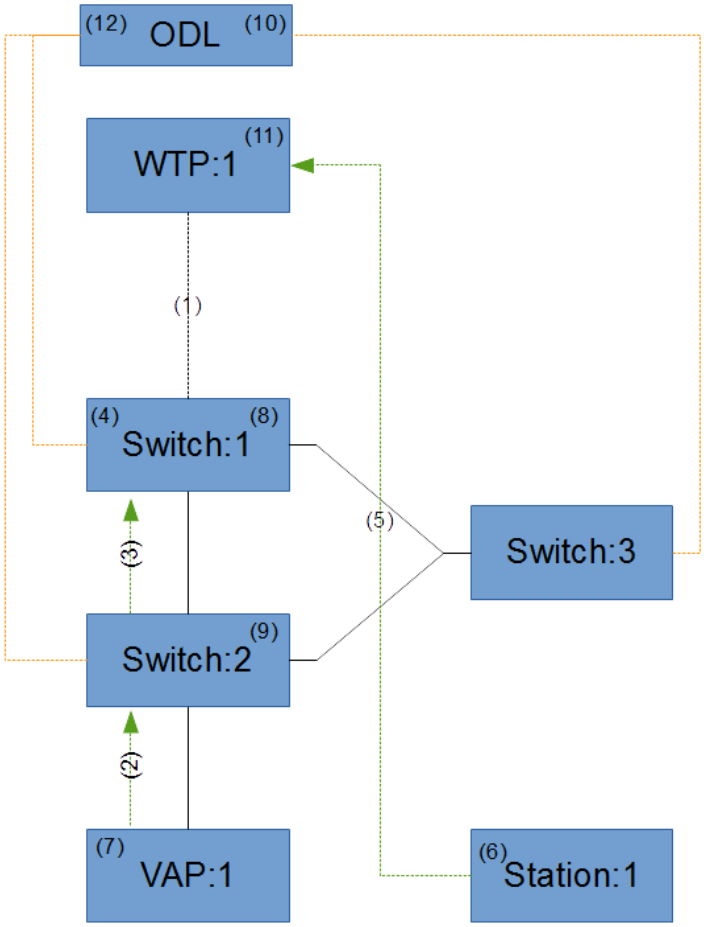

Figure 15 - Use Case 3: WTP Disappears

3.6.4 *Case 4 – VAP Disappears*

If a VAP crashes or is disconnected from the network eventually the network will detect the disappearance and will remove the VAP from the list of known VAPs. This happens when there has been a configurable interval of time in which no beacon frames were received by the controller. The routing rules of switches will timeout and disappear as well, and any stations will have to wait for these timeout to occur before reconnecting. This process can be described in the following steps

(1) The VAP stops generating beacon frames.
(2) Station:1 sends frames to WTP:1.
(3) WTP:1 sends frames to switch:1.
(4) Switch:1 routes frames to switch:2.
(5) Switch:2 routes frames to the port that VAP:1 was connected to. What happens here is that for some reason the frames never reach VAP:1 or if they do no response comes back from VAP:1.
(6) Station:1 detects that it can no longer communicate with VAP:1.
(7) Routing for switch:1 times out.
(8) Routing for switch:2 times out.
(9) Acknowledgement lease times out.
(10) The VAP expires and is removed from the list of known VAPs.

A schematic illustration of the process can be seen in Figure 16.

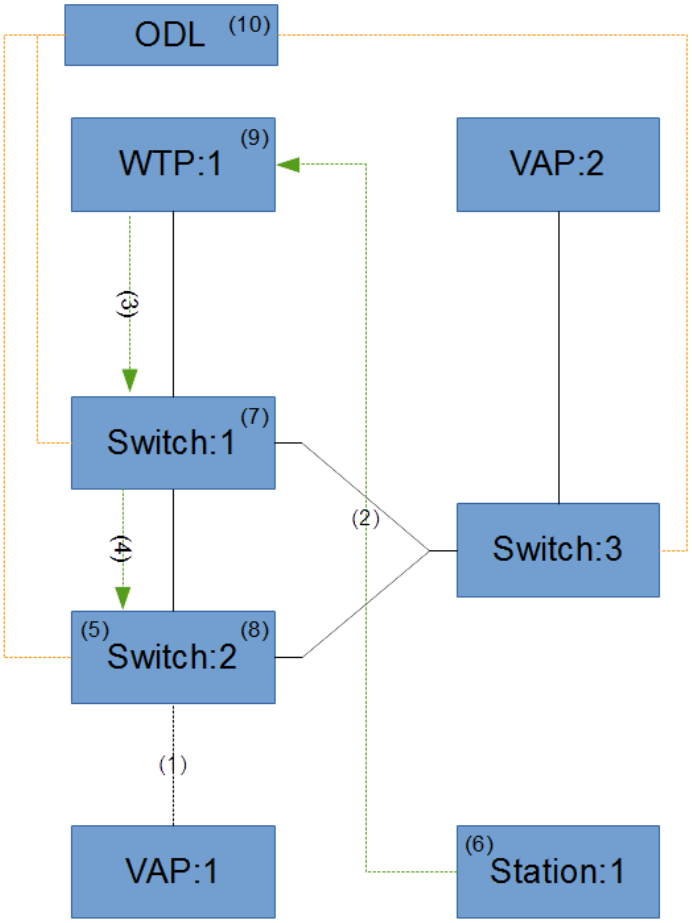

Figure 16 - Use Case 4: VAP Disappears

3.6.5 *Case 5.1 – Station connects*

When a station connects it will generate traffic that will be captured by a WTP and forwarded to the controller since no flow rules will match it. The controller will evaluate the traffic and if it is deemed a connection attempt it will see if there is any VAPs available. If there is no VAP available then the traffic from the station will be temporarily blocked. If there is a VAP available then the controller will allocate the VAP for the station, set up routing rules so that the traffic from the station can reach the VAP, and lease acknowledging from the WTP.

3.6.5.1 VAP Available

These are the steps that show what happens when there is a VAP available.

(1) Station:1 sends probe request.
(2) WTP:1 sends the probe request to switch:1.
(3) Switch:1 does not have any flow rules match traffic from station:1.
(4) Switch:1 sends the probe request to the controller.
(5) Controller identifies the probe request as a connection attempt and selects a free VAP (VAP:1).
(6) The controller issues routing command to switch:1.
(7) The controller issues routing command to switch:2.
(8) The controller leases acknowledgement from WTP:1.
(9) Station:1 continues to send frames.
(10) WTP:1 sends frames into switch:1.
(11) Switch:1 routes frames to switch:2.
(12) Switch:2 routes frames to VAP:1.
(13) VAP:1 sends response frames into switch:2.
(14) Switch:2 routes frames to switch:1.
(15) Switch:1 routes frames to WTP:1.
(16) WTP:1 transmits frames to station:1.
(17) Connection complete.

A schematic illustration of the process can be seen in Figure 17.

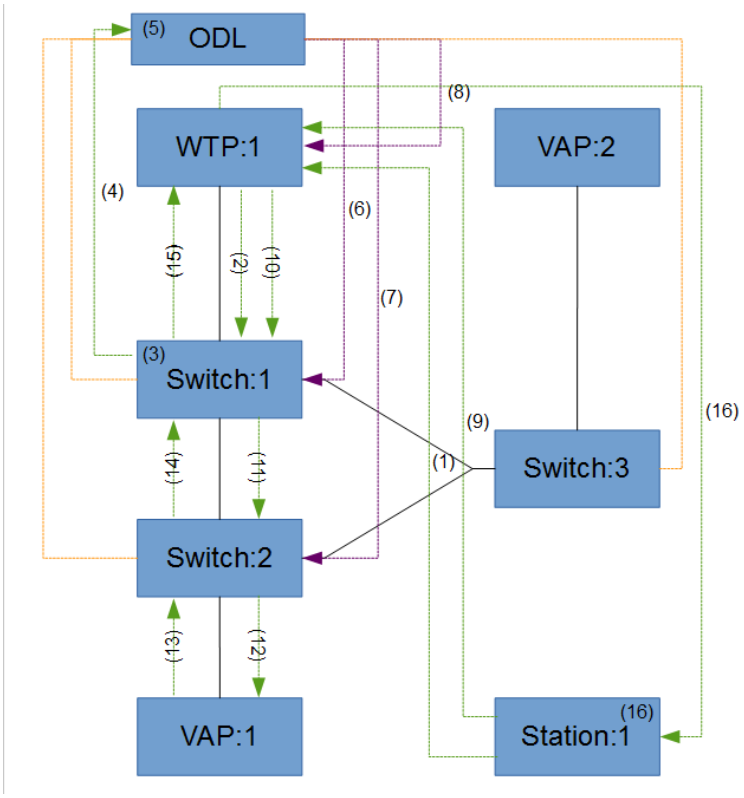

Figure 17 - Use Case 5.1: Station Connects

3.6.5.2 No VAP available

These are the steps that show what happens when there is no VAP available.

(1) Station:1 sends probe request.

(2) WTP:1 sends the probe request to switch:1.

(3) Switch:1 does not have any flow rules that match traffic from station:1.

(4) Switch:1 sends the probe request to the controller.

(5) The controller identifies the probe request as a connection attempt but cant find any free VAPs.

(6) The controller issues blocking command to switch:1.

(7) Station:1 continues to send probe requests.

(8) WTP:1 sends the probe requests to switch:1.

(9) Switch:1 drops the probe requests.

(10) Connection fails.

A schematic illustration of the process can be seen in Figure 18.

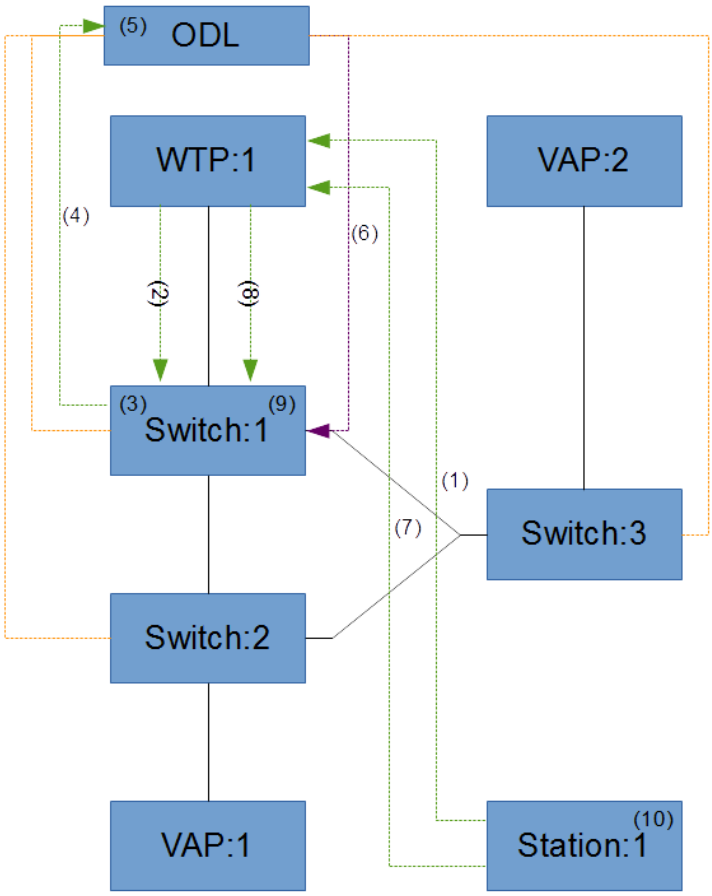

Figure 18 - Use Case 5.2: Station Tries to Connect

3.6.6 *Case 6 – Station Remains Connected*

Since the routing rules are temporary the controller needs to continuously recreate them but should only do so if it can detect that they are used. In order to detect if they are used the controller create two set of rules for the switch closest to the station. The first set of rules simply routes the traffic to the next switch but the second set also sends a copy to the controller which will recreate the rules and extend the acknowledge lease from the WTP once it receives the copy. The process can be described in the following steps:

(1) Station:1 sends a frame.

(2) WTP:1 sends the frame to switch:1.

(3.a) Switch:1 routes the frame to switch:2.

(3.b) Routing for station:1 is about to time out.

(4.a) Switch:2 routes the frame to VAP:1.

(4.b) Secondary flow rules activate and sends a copy to the controller.

(5.a) VAP:1 responds with a frame.

(5.b) The controller refreshes the allocation of VAP:1.

(6.a) Switch:2 routes the frame to switch:1.

(6.b) The controller issues routing command to switch:1.

(7.a) Switch:1 routes the frame to WTP:1.

(7.b) The controller issues routing command to switch:2.

(8.a) WTP:1 transmits the frame to station:1.

(8.b) The controller leases acknowledgement from WTP:1.

A schematic illustration of the process can be seen in Figure 19.

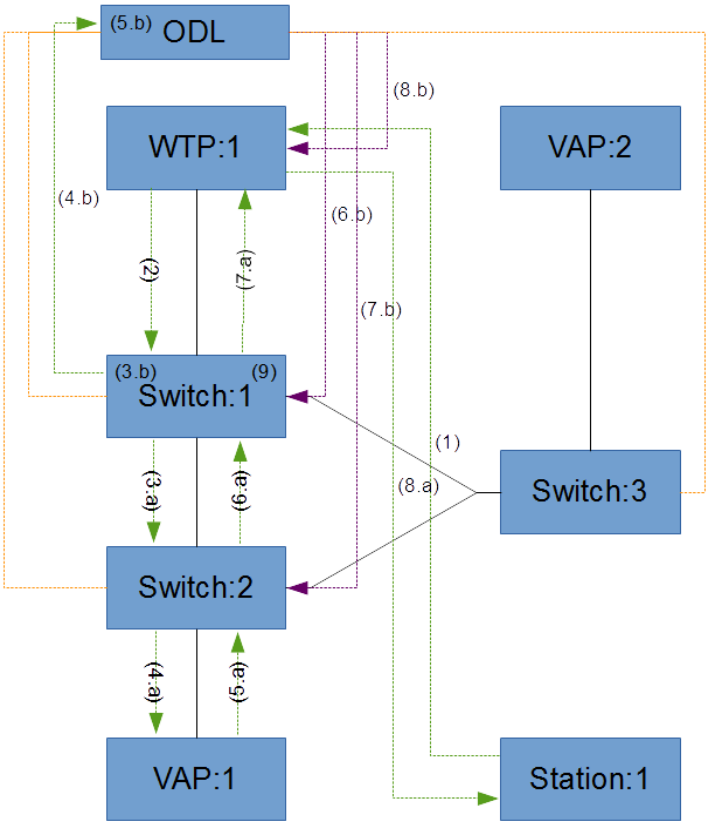

Figure 19 - Use Case 6: Station Remains Connected

3.6.7 *Case 7 – Station Disconnects*

When a station disconnects then it will stop generating traffic directed to the VAP it was connected to. When traffic from the station stops traversing the network for a configurable amount of time then all rules will timeout and all allocation be freed up. The process can be described in the following steps:

(1) Station:1 stops sending frames.
(2) Routing for switch:1 is about to time out.
(3) Routing for switch:2 is about to time out.
(4) Secondary flow rules activate but there are no packets to copy and send to the controller.
(5) Routing for switch:1 times out.
(6) Routing for switch:2 times out.
(7) Acknowledgement lease times out.
(8) Allocation for VAP:1 times out.

A schematic illustration of the process can be seen in Figure 20.

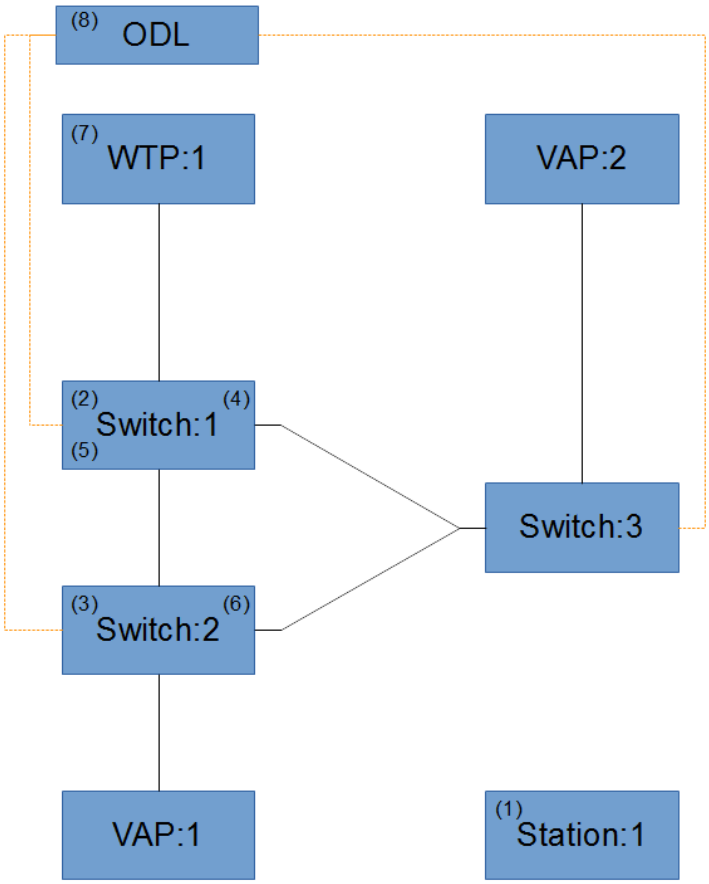

Figure 20 – Use Case 7: Station Disconnects

3.6.8 *Case 8 – Cross traffic*

Since there will be other BSS not associated with CloudMAC present that traffic should be blocked by a configurable amount of time to reduce the workload for the controller. The process can be described in the following steps:

(1) Station:1 sends frames.
(2) WTP:1 sends the frames to switch:1.
(3) Switch:1 does not have any flow rules that match the traffic.
(4) Switch:1 sends the probe request to the controller.
(5) The Controller determines that the traffic is not a connection attempt and does not recognize station:1.
(6) The Controller issues blocking command to switch:1.
(7) Blocking command times out.
(8) Go to step 1.

A schematic illustration of the process can be seen in Figure 21.

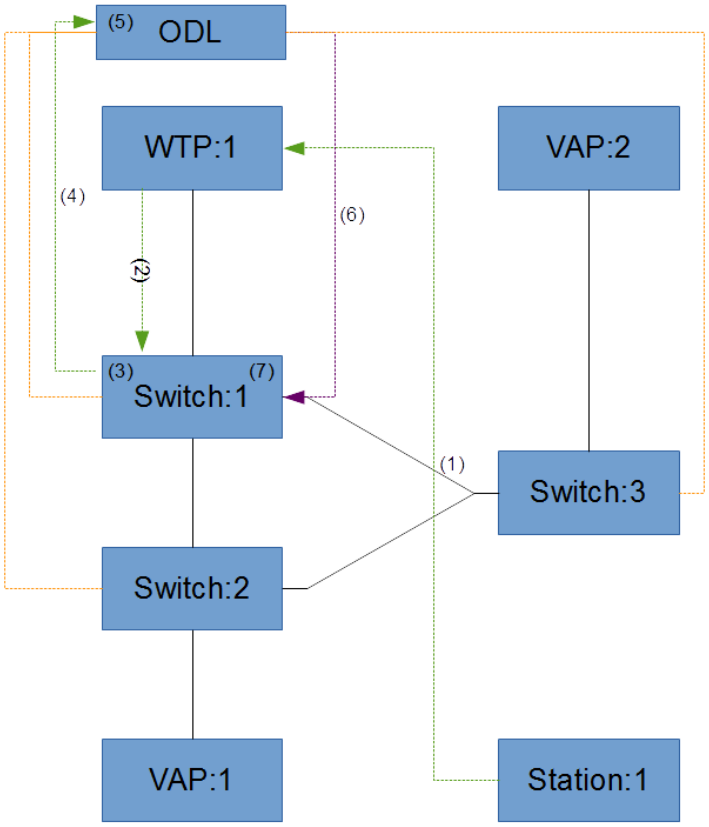

Figure 21 - Use Case 8: Cross Traffic

3.6.9 *Case 9 – Switch Crash*

If a switch restarts then the routing should be recreated, since the switch that restarted will not have any rules it will forward traffic that previously went through it and the controller will be able to identify the traffic and create the rules. The process can be described in the following steps:

(1) Station:1 sends frames.

(2) WTP:1 sends frames to switch:1.

(3) Switch:1 routes the frames to switch:2.

(4) Switch:2 routes the frames to VAP:1.

(5) VAP:1 responds with frames.

(6) Switch:2 routes the frames to switch:1.

(7) Switch:1 routes the frames to WTP:1.

(8) WTP:1 transmits the frames to station:1.

(9) Switch:1 restarts.

(10) Station:1 sends a frame.

(11) WTP:1 sends frames to switch:1.

(12) Switch:1 does not have any flow rules that match the frames.

(13) Switch:1 sends the probe request to the controller.

(14) The controller detects that the traffic comes from a known VAP allocation. VAP allocation is refreshed.

(15) The controller issues routing command to switch:1.

(16) The controller issues routing command to switch:2.

(17) The controller leases acknowledgement from WTP:1.

(18) Station:1 sends frames.

(19) WTP:1 sends frames to switch:1.

(20) Switch:1 routes the frames to switch:2.

(21) Switch:2 routes the frames to VAP:1.

(22) VAP:1 responds with frames.

(23) Switch:2 routes the frames to switch:1.

(24) Switch:1 routes the frames to WTP:1.

(25) WTP:1 transmits the frames to station:1.

A schematic illustration of the process can be seen in Figure 22.

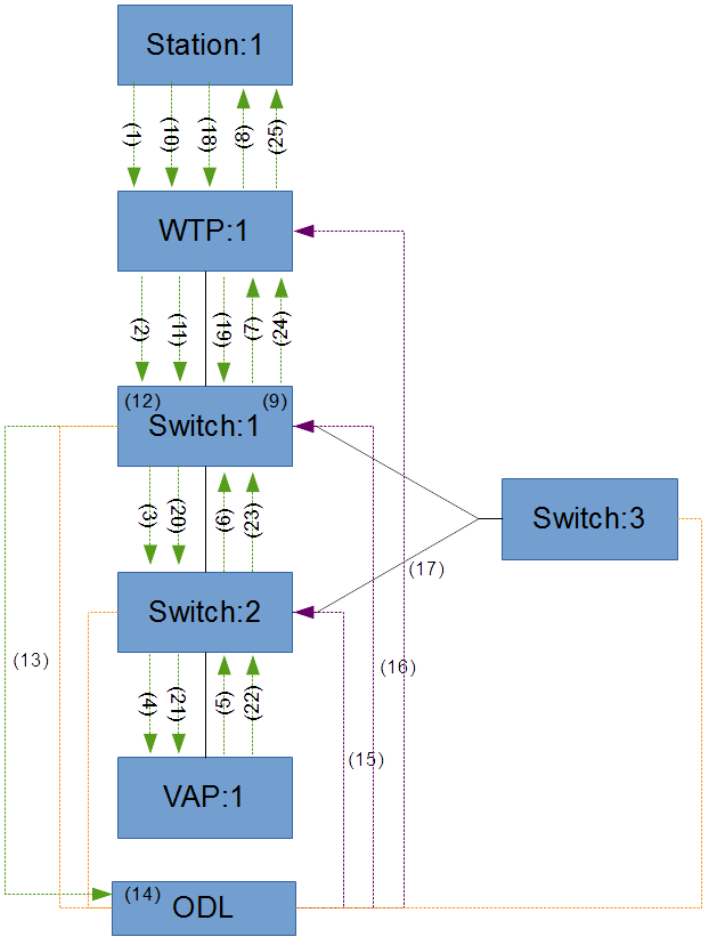

Figure 22 - Use Case 9: Switch Crash

3.7 Chapter Summary

This chapter described the design of the CloudMAC module and how the VAPs and WTPs work together to make the system automatic and how QoS is provided. It also briefly touches on the design of how the CloudMAC packets are created and routed between WTPs and VAPs.

4 Implementation

In order to route IEEE802.11 frames the CloudMAC header needs to be prepended to Radiotap encapsulated IEEE802.11 frames. This is done in two separate modified versions of an old version of backports. Backports is a collection of wireless drivers for a myriad of devices and chipsets. The two drivers of interest to this project are ATH9K and HWSIM. This chapter will go through how this process is achieved in the drivers in addition to how the OpenDaylight CloudMAC module is implemented.

4.1 WTP Driver

This driver is based on ATH9K from Compat Drivers v3.8-1-u [38] which is an old version of backports. The modifications have mainly been made to three functions **ieee80211_set_wmm_default**, **ieee80211_monitor_start_xmit**, and **ieee80211_rx_monitor**. The modifications are extended from previous projects that were done by Jonathan Vestin, Christian Lindeström, and Peter Dely.

4.1.1 *ieee80211_set_wmm_default*

ieee80211_set_wmm_default can be found in net/mac80211/util.c. This function sets up and configures QoS. The modifications activate QoS by overriding enable_qos. This results in the ACs being configured as in Table 20, except that all ACs are set to the same values as background.

AC	CW Min	CW Max	TXOP	AIFS
Voice	3	7	47	2
Video	7	15	94	2
Best Effort	15	1023	0	3
Background	15	1023	0	7

Table 20 - AC QoS settings

4.1.2 *ieee80211_monitor_start_xmit*

ieee80211_monitor_start_xmit can be found in net/mac80211/tx.c. This function is invoked when the driver wants to queue IEEE802.11 frames in the HW queues. This function expects packets to begin with a Radiotap header which it will extract information from, and later use when the IEEE802.11 frame is transmitted. CloudMAC packets do not start with a Radiotap header but they do contain one. So to make it possible to transmit CloudMAC packets they need to be identified, which is done by checking the Ethernet type of packets. If CloudMAC packets are identified the CloudMAC header needs to be stripped, revealing the underling Radiotap header. Depending on which of the four Ethernet types the CloudMAC packet has, its queue mapping will be set the appropriate HW queue. The mapping can be seen in Table 18. Before the header is stripped the rate and signal

strength is extracted and later used to override the values from the Radiotap header. This needs to be done before the function tries to extract the Radiotap header.

4.1.3 *ieee80211_rx_monitor*

ieee80211_rx_monitor can be found in net/mac80211/rx.c. This function is invoked after an IEEE802.11 frame has been received and needs to be processed so that it can be forwarded to a monitor interface. Originally this function prepended a Radiotap header before forwarding the frame to the monitoring interface. Since Radiotap cannot be routed by OpenFlow we need to prepend the CloudMAC header after the Radiotap header has been added. When adding the CloudMAC header the source and destination are extracted from the IEEE802.11 frame and rate and signal from the Radiotap header. CloudMAC has four Ethernet types. Which one that should be used for a frame is identified by checking the frame type and subtype and mapped as seen in Table 18. Beacon frames are problematic and make it harder to detect VAPs so we drop all captured beacon frames. They can be detected by checking the frame type and subtype.

4.2 VAP Driver

This driver is based on HWSIM from Compat Wireless 2010-10-19 which is an old version of backports. The modifications are mainly done in two functions **mac80211_hwsim_monitor_rx**, **init_mac80211_hwsim**, and by the addition of **cloudmac_dev_xmit**. These modifications are extended from previous projects that was done by Jonathan Vestin, Christian Lindeström, and Peter Dely.

4.2.1 *init_mac80211_hwsim*

init_mac80211_hwsim can be found in drivers/net/wireless/mac80211_hwsim.c. This function initializes the virtual wireless interfaces and sets up monitoring interfaces. These original monitoring interfaces are referred to as hw_sim. In order to be able to view the IEEE 802.11 frames both with and without the CloudMAC header, two monitoring interfaces per virtual wireless interface are needed. For each monitoring interface a name and memory need to be allocated. Now the monitoring interfaces need to be configured so the kernel knows which events the interfaces implement. The events handled are when the maximum transmission unit changes, when the MAC address of the interface is set, when an address need to be validated, and when a packet is received and need to be transmitted. There is now an additional monitoring interface to use which will be referred to as cloud_sim.

4.2.2 *mac80211_hwsim_monitor_rx*

mac80211_hwsim_monitor_rx can be found in drivers/net/wireless/mac80211_hwsim.c. This function is invoked after an IEEE802.11 frame has been received and needs processing. It processes IEEE802.11 frames and prepends a Radiotap header and forwards them to a monitoring interface. Since it was requirement to be able to examine traffic and package IEEE 802.11 frames into CloudMAC packets, a second monitoring interface is used. Since Radiotap cannot be routed by OpenFlow we need to prepend the CloudMAC header after the Radiotap header has been added. When adding the CloudMAC header the source and destination is extracted from the IEEE802.11 frame while rate and signal was extracted from the Radiotap header. CloudMAC has four Ethernet types. Which one that should be used for a frame is identified by checking the frame type and subtype and mapped as seen in Table 18.

4.2.3 *cloudmac_dev_xmit*

cloudmac_dev_xmit can be found in drivers/net/wireless/mac80211_hwsim.c. This function is called when a cloud_sim interface receives a packet. It is expected that all packets that are received are CloudMAC packets. When a packet is received the CloudMAC header is stripped revealing the Radiotap header beneath. The packets are then replayed onto all hw_sim interfaces.

4.3 CloudMAC Module Components

In this chapter a high level overview of the components that make up the CloudMAC module will be provided along with a description about why AD-SAL was chosen over MD-SAL and Hydrogen over Helium. All the components can be seen in Figure 23.

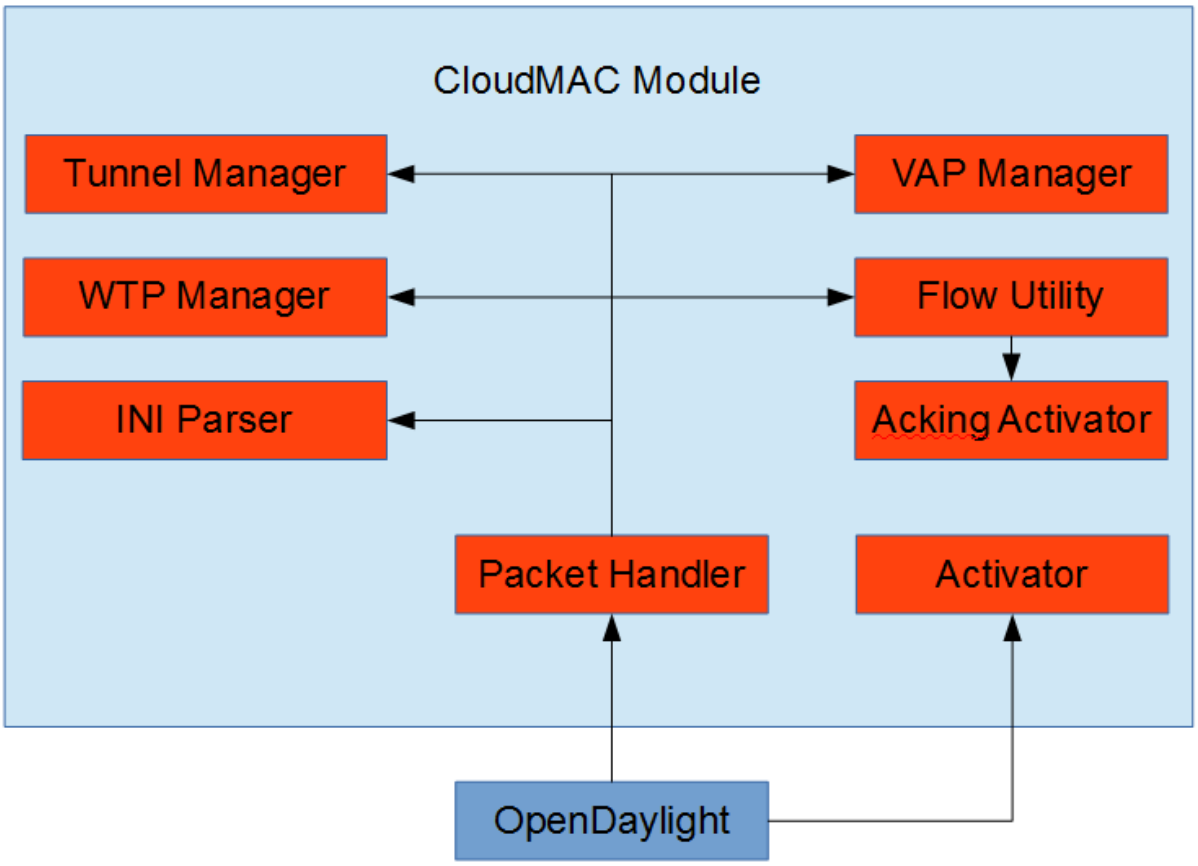

Figure 23- Illustration of how CloudMAC components and their relation to one another

4.3.1 *INI Parser*

This component parses INI files and allows the module to read configuration settings from files. An INI file contains key value pairs grouped into sections. INI files were chosen as they are simple and the CloudMAC module only needed key-value pairs for its configuration.

4.3.2 *Acking Activator*

This component connects to the CloudMAC daemon of WTPs over TCP and issues commands in order to activate acknowledgement of IEEE802.11 frames for VAPs. The command that is issued is "*lease mac duration*". This activates acknowledgement of IEEE802.11 frames for a VAP identified by a MAC address. This component allows the duration of how long the acknowledgement will continue to be specified.

4.3.3 *Tunnel Manager*

This component tracks the mapping of connected stations to VAPs and WTPs. The mapping is temporary and needs to be refreshed periodically. It is refreshed by the packet handler component when it samples packets and detects traffic belonging to a tunnel.

4.3.4 *Flow Utility*

This component is used to make the process of creating flow rules and activate acknowledgements of frames easier. It has functions to create tunnels from one port to another and to block packets on a port for CloudMAC traffic. It uses the acking activator component to activate acknowledgements for frames. The interface IRouting is used when creating tunnels to find a path from one port to another in the network. The paths are used in conjunction with the IFlowProgrammerService service to send routing instructions to switches.

4.3.5 *VAP Manager*

This component monitors the beacon activity of known VAPs so that the module knows when old VAPs go offline and when new ones come online. It also tracks which VAPs are in use by connected stations. Both the VAPs and their allocations will time out if no activity is reported by the packet handler component.

4.3.6 *WTP Manager*

This component monitors the announcement activity of known WTPs so that the module knows when WTPs go offline, come online, or change IP. It also tracks which WTPs that are broadcasting beacon frames. WTPs will time out if no activity is reported by the packet handler component. The broadcasting of beacon frames is temporary and only meant to alert stations to the presence of CloudMAC.

4.3.7 *Packet Handler*

This is the core component of the module that binds the other components together. It examines incoming packets and identifies whether it is a connection attempt, sampled packet, beacon frame, or a WTP announcement packet.

If there is a connection attempt then the packet handler component will check if there is a VAP available and whether that is the case it will set up a tunnel from the WTP that the packet originated from, and to the chosen VAP. In the case of a connection attempt that targets a specific VAP that VAP will be chosen if it is available. If a station tries to connect to a VAP that is already in use that traffic is temporarily blocked. If a station tries to change VAP then that request is ignored. If a connection

attempt is to an unknown VAP then that traffic is temporarily blocked. Traffic in these cases are blocked to reduce the workload of the module.

The module samples packets periodically for each tunnel. It does so in order to monitor whether tunnels are still in use or if VAPs can be freed up for use by other stations. If a tunnel is still in use the packet handler will use the flow utility and tunnel manager components to refresh the tunnel.

Beacon frames are a sign that a VAP is present somewhere in the network. When beacon frames are detected they are checked if the VAP is known since earlier. If that is the case the VAP manager component is notified. Otherwise the VAP manager component is notified of the presence of a new VAP. If there is a WTP that is not already broadcasting beacon frames then the beacon frames are routed to such a WTP.

If a WTP announce packet is received, it is checked to see whether or not the WTP is known since earlier. If that is the case then the WTP manager component is notified. Otherwise the WTP manager component is notified of the presence of a new WTP.

The packet handler component uses the INI parser component to read from a file whose relative path are "./settings/cloudmac.ini" and loads settings from it. All configurable values are expected to be under a section named "cloudmac". It is expected that that file exists, but it does not need to contain anything beyond the a section named "cloudmac" If a setting is not found in the INI file the CloudMAC module will use default values for those settings.

In order to listen unmatched packets the component subscribes to packet in events from the IListenDataPacket interface. To decode packets from packet in events the DataPacketService service is used.

4.3.8 *Activator*

This is the entry point for all AD-SAL modules. This class has a function getImplementations() that provides OpenDaylight with a list of classes that OpenDaylight should create instances of. OpenDaylight knows which classes to create instances of. OpenDaylight configures the instances througha function called configureInstance(). In the configureInstance() function we register the PacketHandler class to receive instances of IListenDataPacket, DataPacketService, IRouting, and IFlowProgrammerService. These interfaces along with descriptions can be found in Table 21.

Interface	Description
IListenDataPacket	This interface provide packet-in events.
DataPacketService	Is used to encode, decode, and send packets.
IRouting	Used to find a path through the network.
IFlowProgrammerService	Used to create flow rules on switches.

Table 21 - List of interfaces used by the AD-SAL CloudMAC module with descriptions.

4.3.9 *OpenDaylight Module Discussion*

Hydrogen was chosen over Helium as it provided better support for AD-SAL modules while still supporting what was required for the project. AD-SAL support in Helium was incomplete and with no sign of being completed before the end of the project. Most notably is that the compatibility module that is supposed to handle conversions of AD-SAL flow creation to MD-SAL flow creation tries to map OpenFlow 1.0 to OpenFlow 1.3 one to one. Since the enqueue action was replaced by a combination of set-queue and output actions in OpenFlow 1.3 the compatibility module throws an exception since it cannot map it one to one. Another aspect was that Helium could not create controller actions. It generated OpenFlow messages just with the exception that when the OpenFlow messages were analysed with tcpdump it was discovered that the controller action was never present in the instruction list. If the only action specified in the code was a controller action then the list was empty resulting in the switches defaulting to drop actions. If multiple actions were specified in the code the instruction list contained all but the controller actions.

4.4 CloudMAC Module Configuration

The CloudMAC module allows a multitude of values to be configured. Table 22 shows values along with brief descriptions of what purpose they serve. The default values and units used for each value is listed.

Setting Name	*Default Value*	*Unit*	*Description*
flowDuration	120	Seconds	How long the rules that make up the tunnels should exist before needing to be refreshed.
blockFlowDuration	2	Seconds	How long unwanted traffic is blocked.
graceDuration	10	Seconds	The amount of time the controller will wait for packets before a tunnel times out. This waiting occurs at the end of the flow duration not after.
accessPointExpiration	160000	Milliseconds	How long the controller will wait without receiving beacon frames from a VAP before the VAP is considered gone.
tunnelExpiration	120000	Milliseconds	The amount of time the controller will wait for packets before a tunnel is considered expired.
terminationPointExpiration	120000	Milliseconds	The amount of time the controller will wait for packets before a WTP is considered expired.
beaconFlowDuration	10000	Milliseconds	How long unused VAPs beacon frames may be routed to WTPs.
ethernetTypes	4918:4919:4920:4921		Simply a list of the Ethernet types that are part of CloudMAC.
queueIndices	0:3:2:1		A mapping of Ethernet types to queue indices. The first in the list is mapped to the first in the Ethernet types list.
blockPriority	1200		The OpenFlow priority to use for flow rules that block/drop traffic.
tunnelPriority	1100		The OpenFlow priority to use when setting up the tunnels from WTPs to VAPs.
beaconPriority	1000		The OpenFlow priority to use when routing VAP beacon frames to WTPs.
terminationPointConfigPort	1999		The port which should be used to connect to WTPs in order to configure the CloudMAC daemon.

Table 22 - CloudMAC module configuration options

4.5 Chapter Summary

This chapter has gone through which extensions were needed for the WTP and VAP drivers, and how the components of the CloudMAC module interact. It also explains the rational for choosing the Hydrogen version of OpenDaylight over Helium and a small comparison between the two. It ends with a description of all configurable values for the CloudMAC module.

5 Evaluation

This chapter will present several evaluations that each focus on different parts of the system. The evaluation are:

(1) Hardware Queue
(2) Switch Queue
(3) Connection

The hardware queue evaluation tests whether there was any difference in performance between the four ACs. Both with and without congestion. The switch queue evaluation tests the whether the QoS in the switches had any impact during congestion. The connection evaluation tests how the system performs when a station tries to connect.

During several of the evaluations netperf-wrapper was used: “Netperf-wrapper is a wrapper around netperf and similar tools to run predefined tests and aggregate and plot the results. It defines several tests that can be run against one or more hosts, primarily targeted at testing for the presence of bufferbloat under various conditions.” [39]

5.1 Hardware Queue Evaluation

This evaluation was of the hardware queues in the WTPs and the mapping of Ethernet types to the hardware queues. Improving the QoS for CloudMAC when the wireless medium is congested was one of the main goals of the project. As such it was a requirement to evaluate whether there was any improvement.

5.1.1 *Setup*

The setup was as follows. One WTP connected directly to a VAP and a station connected through the WTP. Beside this setup there was another station connected to an AP. The VAP and AP was configured to operate on channel 3 at 54Mbps. The layout can be seen in Figure 24 where the dashed lines represent wireless connections while continuous lines represent wired. The VAP was running on a Linux machine running Ubuntu [40]. Station α was running Ubuntu Linux with D-Link DWA-140 [41] (hardware version B2. Firmware version 1.13(E)). Station β was running Ubuntu Linux with Intel PRO/Wireless 4965 [42] (driver: iwl4965, version: v3.13.0-52-generic).

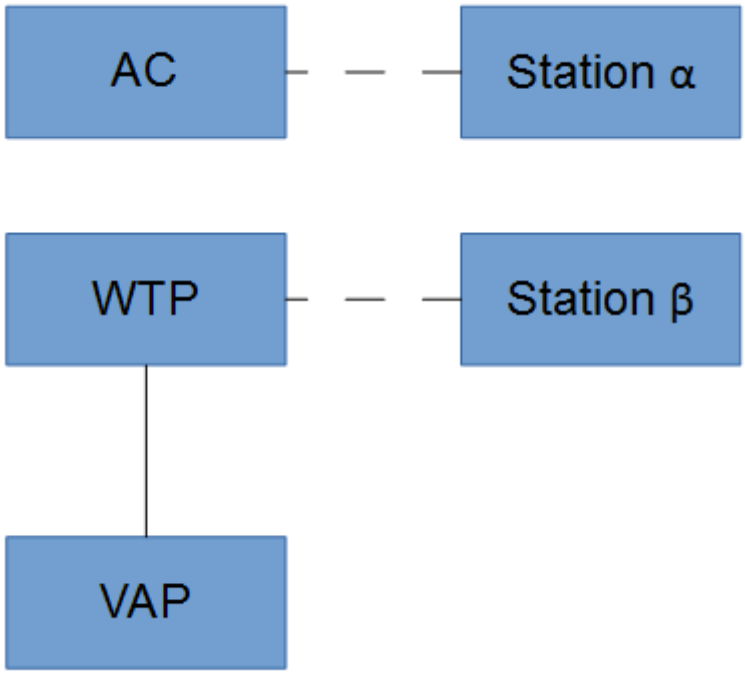

Figure 24 - Hardware Setup for the WTP hardware queue and Ethernet mapping evaluation.

5.1.2 *Testing*

First it was verified that the WTP was indeed actually utilizing the four different hardware queues. This was verified by enabling the debug file system for the WTP driver and activated by adding the lines in Figure 25 to the configuration file.

```
export CONFIG_CFG80211_DEBUGFS=y
export CONFIG_MAC80211_DEBUGFS=y
export CONFIG_ATH_DEBUG=y
export CONFIG_ATH9K_DEBUG=y
export CONFIG_ATH9K_DEBUGFS=y
```

Figure 25 - ATH9K driver debug file system configuration

After the debug file system was enabled the VAP driver was modified to map all IEEE802.11 frames to each of the four Ethernet types one at a time. For each mapping it was checked that only that particular queue was used. It was done by inspecting the /sys/kernel/debug/ieee80211/phy0/ath9k/queues file. The results were that all the queues were utilized as expected. An example of the output can be seen in Figure 26.

```
(VO): qnum: 0 qdepth: 0 ampdu-depth: 0 pending: 0 stopped: 0
(VI): qnum: 1 qdepth: 0 ampdu-depth: 0 pending: 0 stopped: 0
(BE): qnum: 2 qdepth: 0 ampdu-depth: 0 pending: 0 stopped: 0
(BK): qnum: 3 qdepth: 7 ampdu-depth: 0 pending: 7 stopped: 0
```

Figure 26 - Example output of /sys/kernel/debug/ieee80211/phy0/ath9k/queues

After the utilization and Ethernet type to HW queue was tested the round trip time (RTT) was tested using ping traffic generated by netperf-wrapper ping test. First baseline RTT values were measured for each AC. The result can be seen in Figure 27 (Voice AC), Figure 28 (Video AC), Figure 29 (Best Effort AC), and Figure 30 (Background AC). These graphs have been summarised into Table 23. The table the percentages of RTTs that were below certain values.

	Voice	Video	Best Effort	Background
10ms	40%	39%	34%	36%
13ms	70%	71%	68%	69%
16ms	96%	98%	95%	95%

Table 23 – HW queue evaluation, baseline for percent of RTTs below certain values.

After the baseline values were measured, the process was repeated with cross traffic between the AP and station α. The cross traffic consisted of two UDP streams, one in each direction, transmitting locally at 54Mbps generated by iperf in order to oversaturate the wireless medium. The results can be seen in Figure 31 (Voice AC), Figure 32 (Video AC), Figure 33 (Best Effort AC), and Figure 34 (Background AC). The graphs have been summarized into Table 24. The table shows the percentages of RTTs that were below certain values. The configuration of the best effort AC is the same in terms of contention windows and AIFS as the cross traffic. All tests mentioned were run for 60 seconds.

	Voice	Video	Best Effort	Background
10ms	3%	4%	0%	0%
13ms	15%	19%	3%	0%
16ms	37%	38%	19%	0%
22ms	75%	70%	44%	0%
30ms	95%	92%	81%	0%

Table 24 – HW queue evaluation, measured percent of RTTs below certain values during congestion.

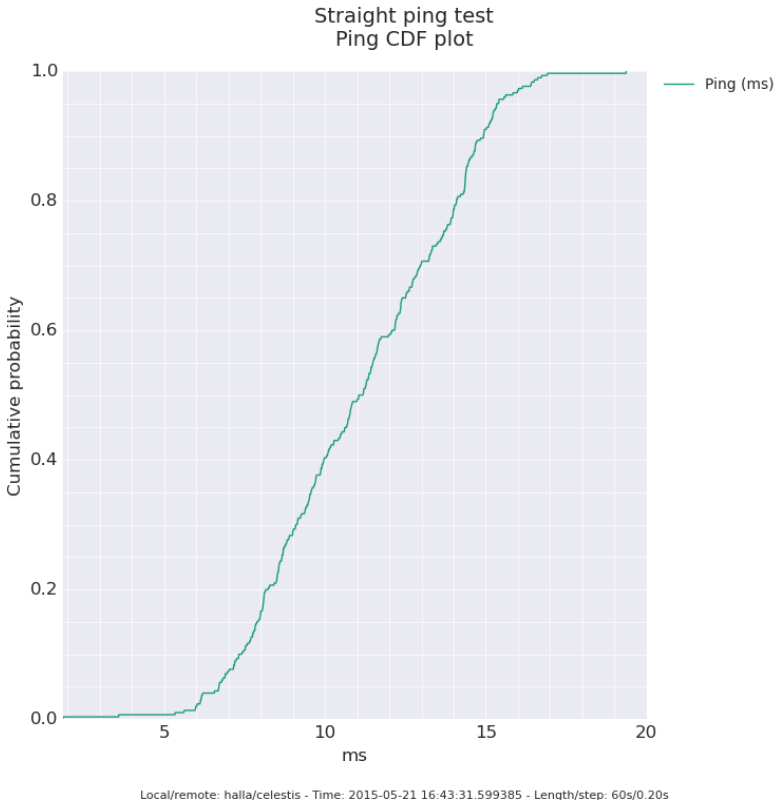

Figure 27 – HW queue evaluation, baseline measurement for the voice AC

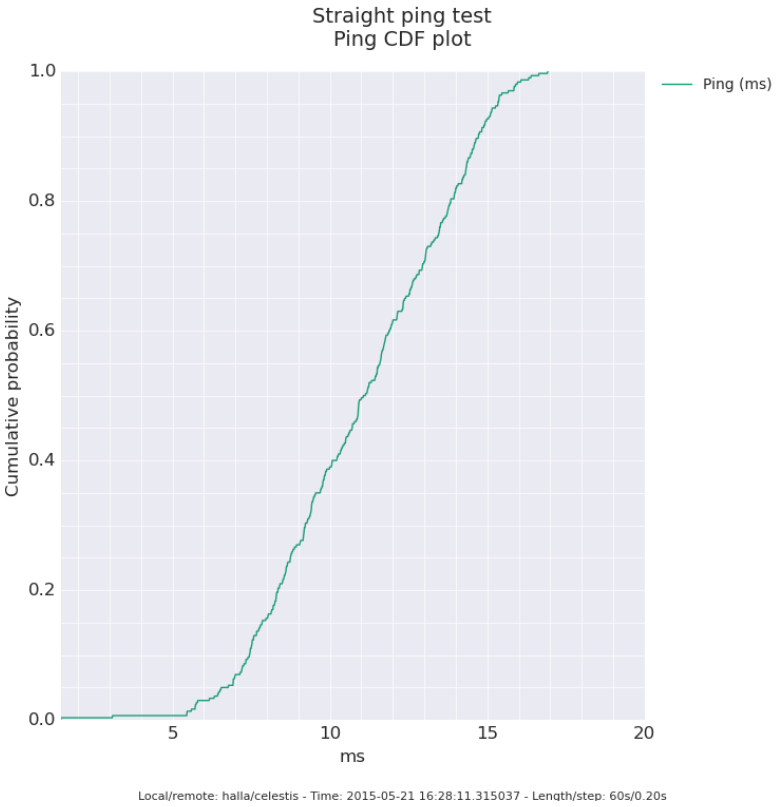

Figure 28 - HW queue evaluation, baseline measurement for the video AC

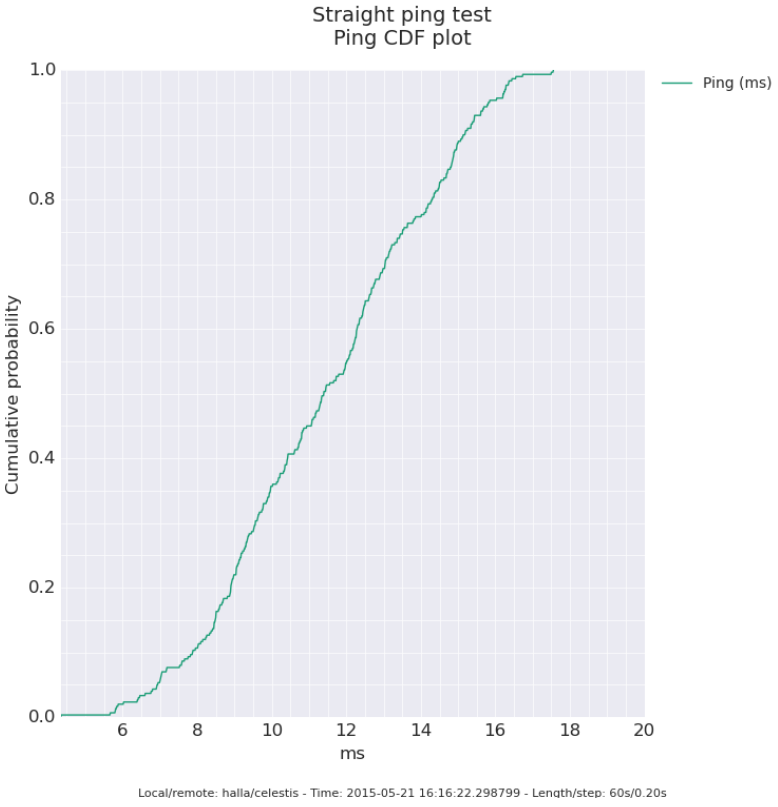

Figure 29 - HW queue evaluation, baseline measurement for the best effort AC

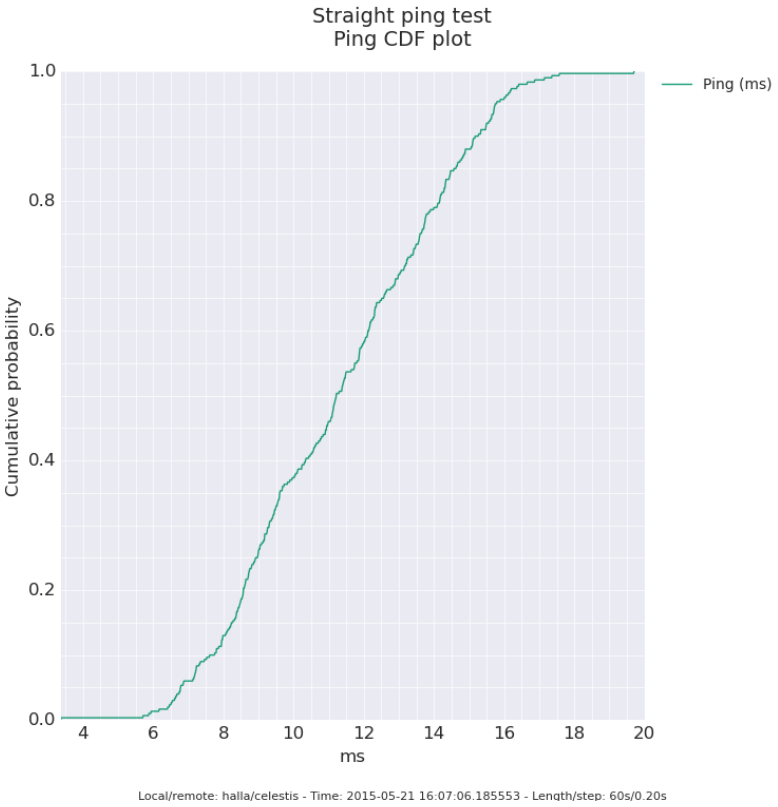

Figure 30 - HW queue evaluation, baseline measurement for the background AC

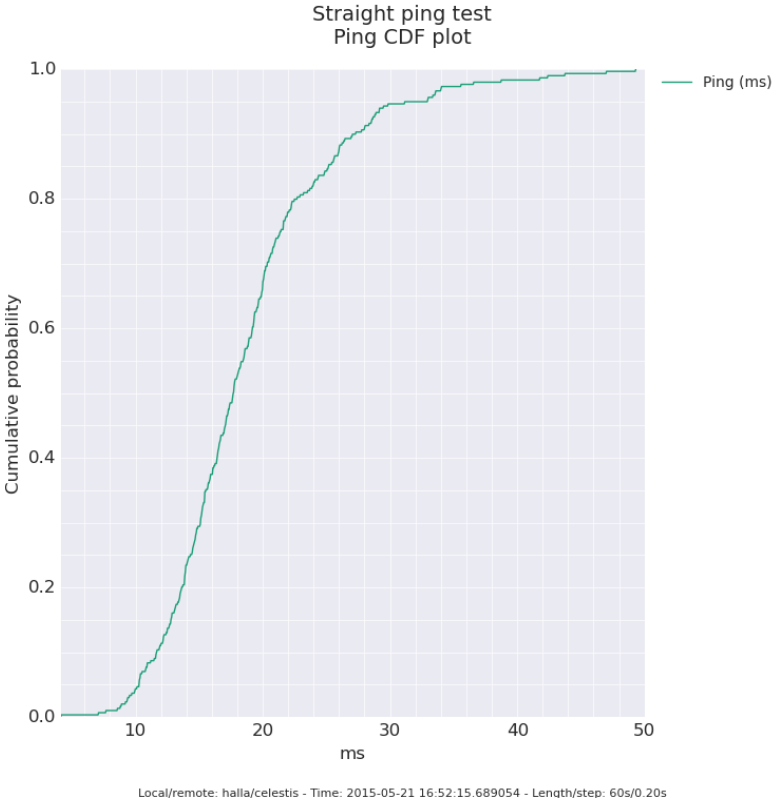

Figure 31 - HW queue evaluation, measurement of RTT for the voice AC during congestion

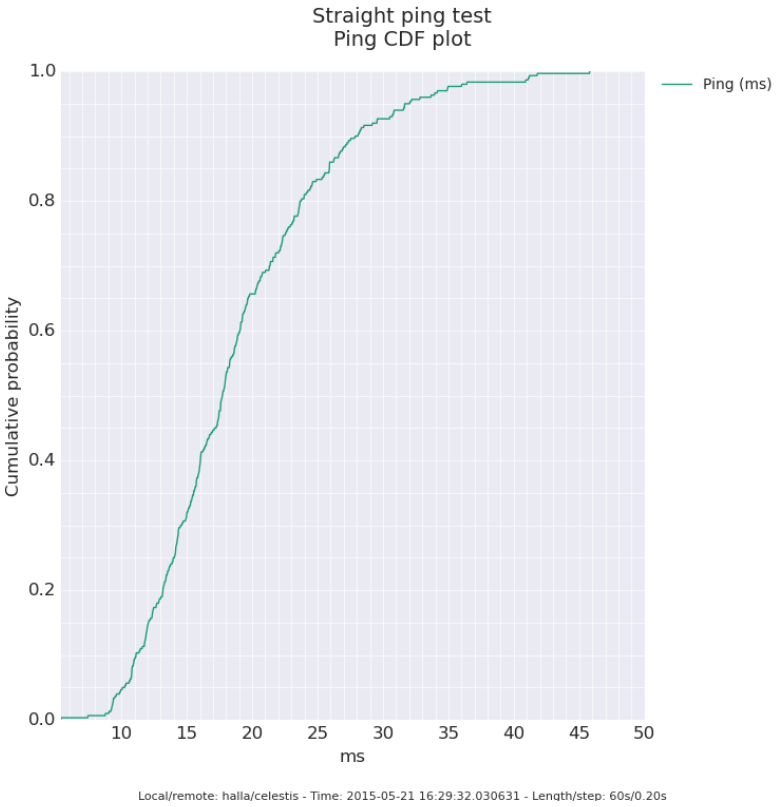

Figure 32 - HW queue evaluation, measurement of RTT for the video AC during congestion

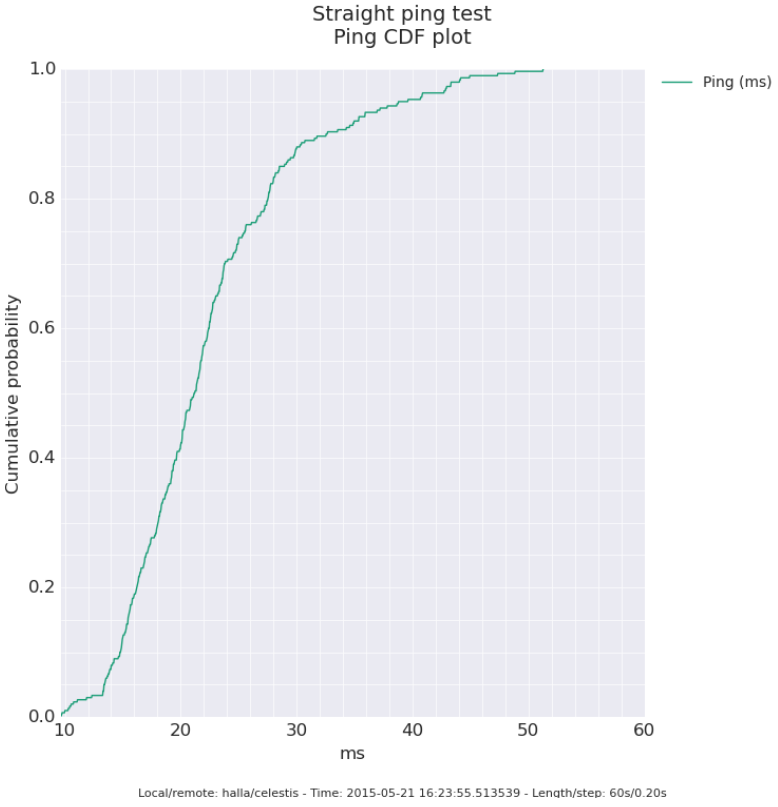

Figure 33 - HW queue evaluation, measurement of RTT for the best effort AC during congestion

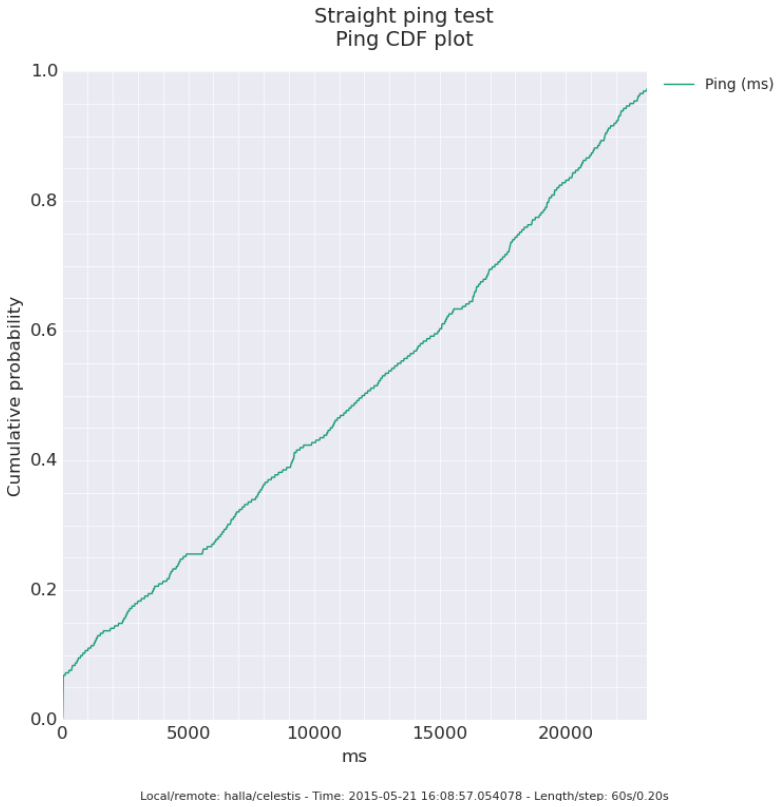

Figure 34 - HW queue evaluation, measurement of RTT for the background AC during congestion

5.1.3 *Conclusion*

The conclusion of this evaluation was that the mapping from the four Ethernet types to the hardware queues did in fact work as intended. The RTT increased as expected during congestion due to the fact that collisions forced stations to retry with larger backoff values. For the RTT values without congestion there were no notable difference. During congestion the background AC gets starved with very large RTTs due to it having a larger AIFS than the cross traffic. The Background AC becomes greatly penalised if it is used in the presence of BSS not using IEEE802.11e. Both the voice and video ACs perform the same as they both have smaller contention windows and AIFS. The number of pings with values lower than 13ms is greatly higher than compared to the background AC. Twice as many pings end up with a RTT of 16ms and 22ms. IEEE802.11e and IEEE802.11ae works very well for reducing the RTT of prioritised IEEE802.11 frames though with some issues if BSS without QoS are percent.

5.2 Switch Queue Evaluation

This evaluation was of the queues in the switches. Improving the QoS for CloudMAC when the wired network is congested was one of the main goals of the project. As such it was a requirement to evaluate whether there was any improvement.

5.2.1 *Setup*

The setup was as follows: one WTP, two PCs running Ubuntu, two TP-LINK witches, and one station running widows with Intel PRO/Wireless 4965 [42] (driver: iwl4965, version: v3.13.0-52-generic).PC β was also running one instance of hostapd [18]. The routing was manually set up so that all traffic from PC α and β were routed to each other with the exception of CloudMAC traffic which was routed between β and the station. The layout is as seen in Figure 35. Continuous lines represent wired connection while dashed are wireless connections. The throughput of ports in the two switches was limited to 100 Mbps.

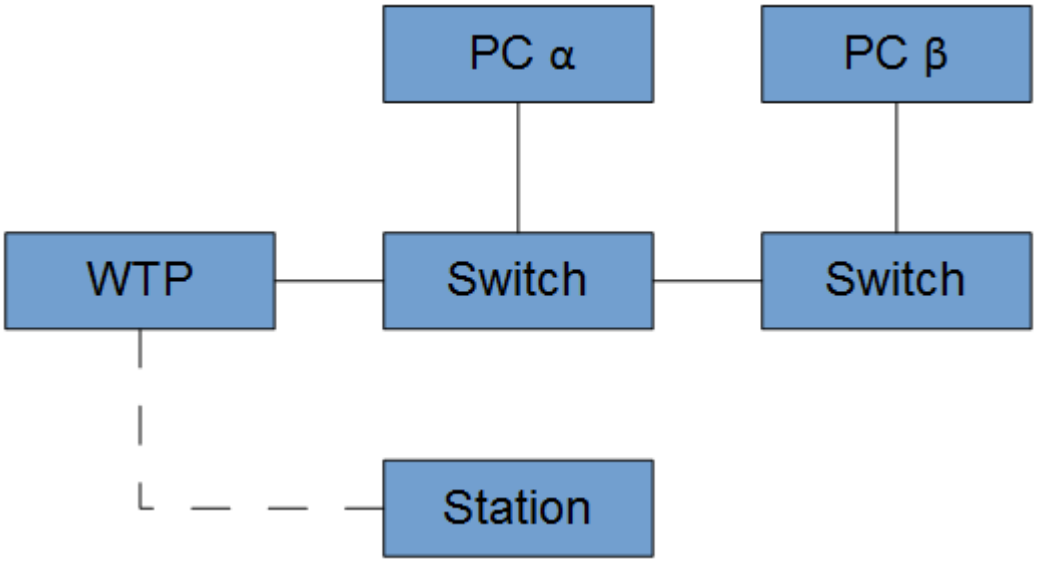

Figure 35 - Hardware Setup for switch queue evaluation.

5.2.2 *Testing*

First the baseline was measured by pinging the station from β with no other traffic using ping traffic generated by netperf-wrapper ping test. The results were that 95% of all packets had a ping of 1.66ms this is shown in Figure 36. After the baseline measurement it was repeated both with and without QoS during congestion. The network was congested by having two UDP streams generated by iperf between PC α and PC β one in each direction. The UDP stream was sent at 100 Mbps. Without QoS all traffic was mapped to the same queues. With QoS, the UDP traffic was queued

under a different queue with lower priority than the CloudMAC traffic. Without QoS, 95% of all pings were under 2.00ms this is presented in Figure 37. With QoS, 95% of all pings were under 1.75ms this is shown in Figure 38. The difference being 0.25ms. All tests were run for 60 seconds.

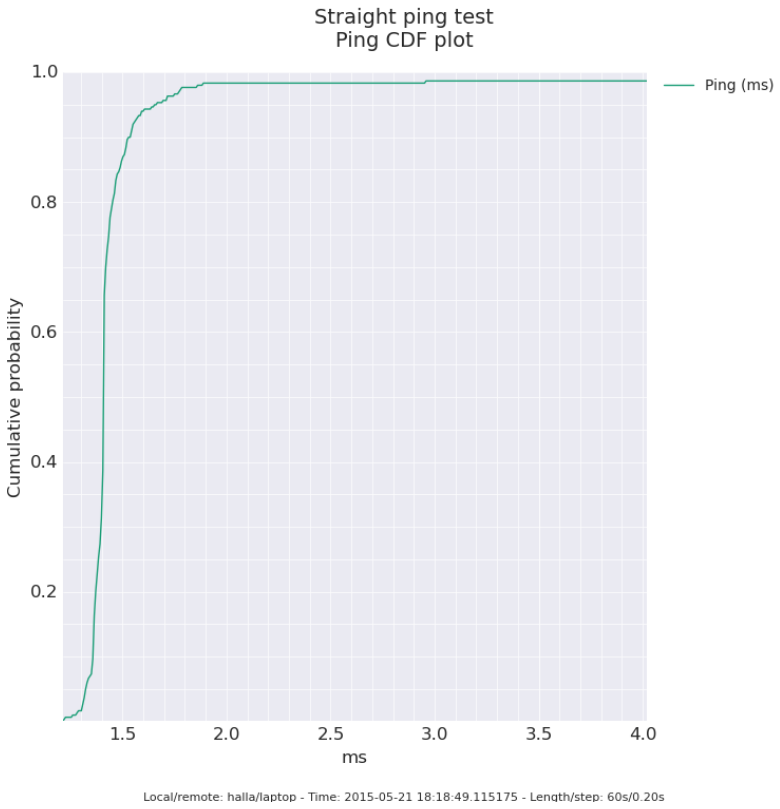

Figure 36 – Switch queue evaluation, baseline measurement of RTT

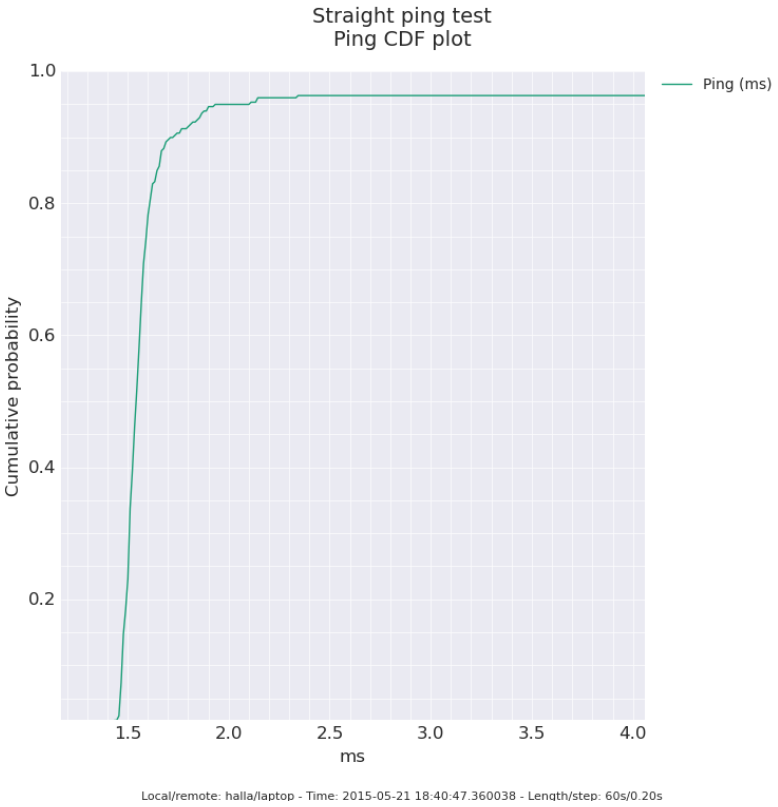

Figure 37 - Switch queue evaluation, measurement of RTT during congestion without QoS

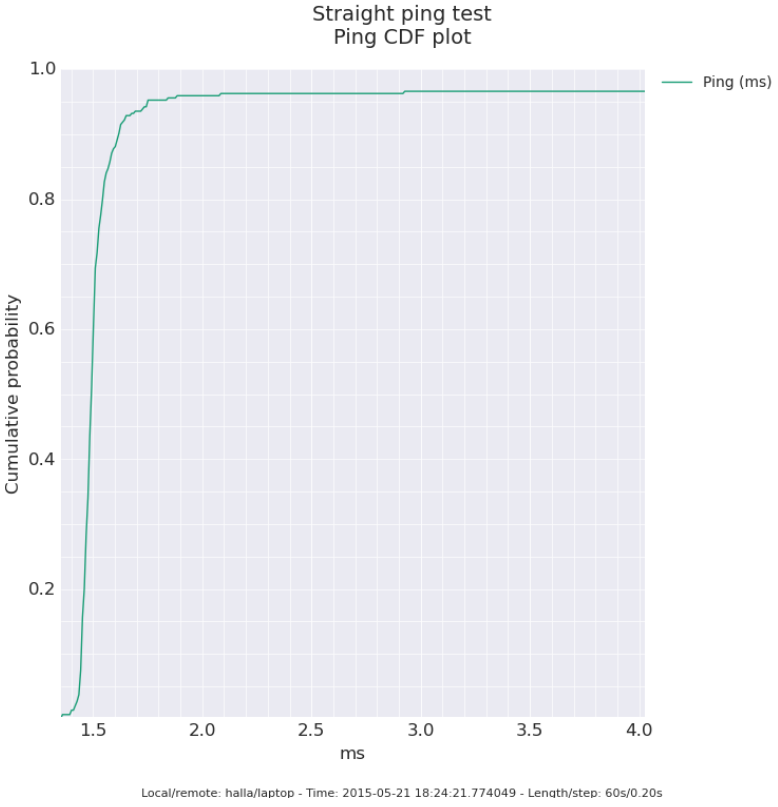

Figure 38 - Switch queue evaluation, measurement of RTT during congestion when using QoS

5.2.3 *Conclusion*

During congestion Without QoS, the RTT increases by 0.34ms (20.4%) while with QoS, it only increases by 0.09ms (5.4%). While this increase without QoS may not seem very large it is per congested switch which would be cumulative and add up over large networks. If QoS is used in switches and that high priority IEEE802.11 frames are given priority in switches, it would increase the stability of CloudMAC and enable CloudMAC to work in larger networks.

5.3 Connection Evaluation

This test were to evaluate how well the CloudMAC system could handle connection attempts. Since part of the project goal was to implement CloudMAC in OpenDaylight, there was a requirement to evaluate how well stations could connect to the system.

5.3.1 *Setup*

The setup of the devices used during this evaluation was:

- Three TP-LINK Open vSwitch enabled switches.
- One device that captured and transmitted 802.11 wireless frames referred to as wireless termination point (WTP). Two in the case of handover.
- One Ubuntu Linux computer running multiple instances of hostapd referred to as virtual access point (VAP).
- One Ubuntu Linux virtual machine running OpenDaylight Hydrogen Base release controller.
- One Ubuntu Linux computer with D-Link DWA-140 [41] (hardware version B2. Firmware version 1.13(E)) referred to as client.
- For the persistence test the client was replaced by another Ubuntu Linux computer with Intel PRO/Wireless 4965 [42] (driver: iwl4965, version: v3.13.0-52-generic

In figure below the blue and green arrows represent two different networks that are isolated from each other. The green (labelled a) network is controlled by the OpenDaylight controller and the blue (labelled b) is there as a separate control network. The green dashed line labelled c represent a wireless connection. The control network is separate to simplify the process of controlling the green network as the green in this state does not need to automatically route normal traffic. It also makes it easier to reset the switches if desired. A representation of the setup can be seen in Figure 39.

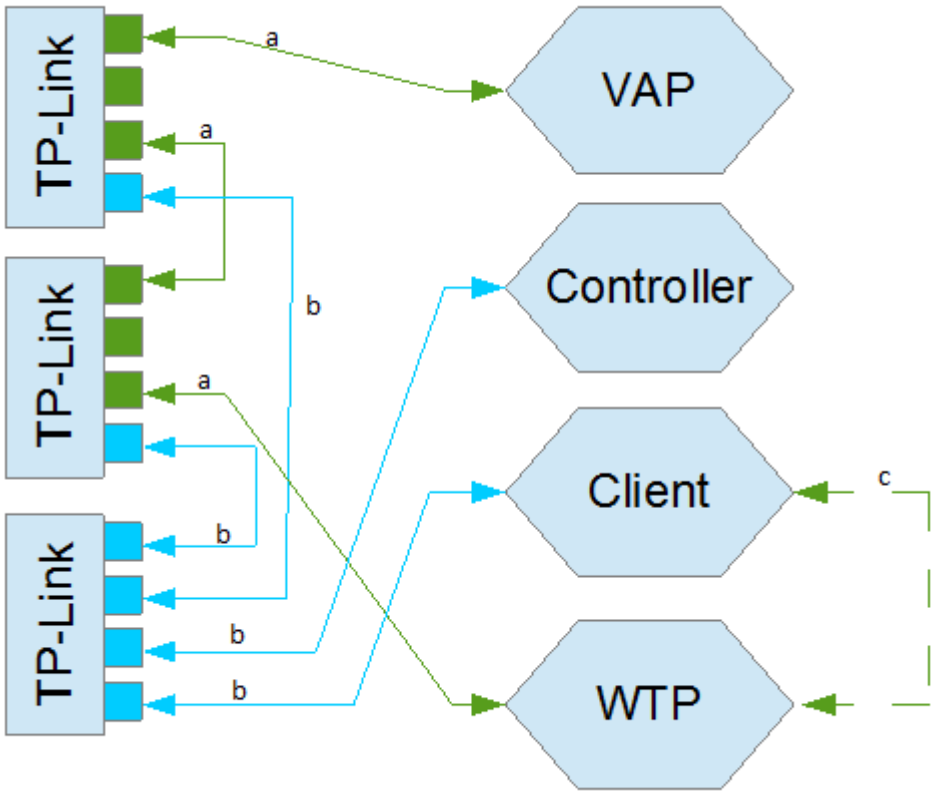

Figure 39 - Hardware Setup for connection evaluation.

5.3.2 *Testing*

The test consisted of attempts to connect to CloudMAC and recording the number of repeated attempts that were needed before a connection could be established. 20 iterations were performed, in 50% of all iterations two attempts were needed and in the other 50% 3 attempts were needed. The need for 2 or 3 attempts is undesirable. The desired outcome were to succeed at the first attempt. To check what were the problem the routing was set up manually and another 20 iterations were performed. The results showed that 100% of all iterations required 1 attempt. The process of connecting is that the switches connected to the WTP sends unmatched packets to the controller. The controller sends the packets to the CloudMAC module which examines the CloudMAC packets and issues routing instructions to the two switches. This process evidently exceeds the time that the station will wait for an association request to yield an association response.

It was also checked how stable the connection was, it was checked by connecting with a client and seeing how long the connection would persist. The client continuously sent ping traffic to the VAP, after 7 days the connection was still active.

5.3.3 *Conclusion*

The system was slower than desired at setting up routing rules which resulted in multiple connect attempts being needed but once connections are established they remain stable for long periods of time.

5.4 Chapter Summary

This chapter has presented three evaluations which focused on; WTP HW queue and Ethernet to HW queue mapping, switch queues, and connecting to CloudMAC. Each evaluation presented the setup, test data, and a conclusion about the results.

6 Conclusion

The four goals were:

1. Evaluate OpenDaylight by implementing CloudMAC in OpenDaylight. The evaluation was to see if CloudMAC could be implemented in OpenDaylight.
2. Implement QoS in CloudMAC and evaluate performance gains of QoS.
3. Enable CloudMAC to automatically detect WTPs and VAPS. This is to make it easier to setup CloudMAC networks.
4. Make the CloudMAC OpenDaylight implementation configurable.

CloudMAC could with some minor issues be implemented in OpenDaylight as an AD-SAL module. The minor issues were that Hydrogen [43] an older version of OpenDaylight had to be used instead of the newer Helium [44] version. Helium could not be used since it had a bug that made the OpenFlow controller actions unusable. The OpenFlow controller action was needed to sample packets. Another issue was that there was a considerable delay between a station trying to connect until the routing needed to connect was in place. The exact cause of this was unclear, and should be looked into. So the process of connecting is to attempt to connect, wait a few seconds and make another attempt. Once connected the connection is stable and could last at least a week. The CloudMAC module offers a myriad of settings that can be altered to change how the module operates. These settings allow the module to more easily be optimized, without having to recompile the module. Overall OpenDaylight was quite flexible and can easily be used to implement new services in SDN networks.

There has been two major releases of OpenDaylight thus far; Hydrogen [43] released on the 4th of February 2014 [43] and Helium [44] released on the 29th of September 2014 [44]. Hydrogen was the first release and was released in three variants: base, virtualisation, and service provider. The difference between the variants is that they contain different sets of default bundles. Hydrogen supports OpenFlow 1.0 (section 2.2). The system used in order to enable dynamic loading and unloading of modules during execution is the Open Service Gateway Initiative [45] (OSGI). Helium is the newest release and uses Karaf [46] from the Apache Software Foundation [47] which can be seen as an extension to OSGI. In Helium bundles are also grouped into features to facilitate the turning on and off of certain functionality. Helium supports OpenFlow 1.0 and 1.3. A brief comparison between how a flow rule can be created in AD-SAL and MD-SAL can be seen in Appendix C.

The implementation of QoS for CloudMAC showed significant improvements both in WTPs and OVS switches. In WTPs QoS was implemented by activating IEEE 802.11e and IEEE 802.11ae and mapping CloudMAC packets to one of the four ACs (Voice, Video, Best Effort, and Background). In switches

OVS was used to setup QoS for each port by creating a HTB tree with four queues with. The switch queues were given different priorities. The evaluation showed that CloudMAC would be able to operate with good QoS even during congestion. The evaluation missed one key point though. The HTB queues created by OVS had a queue length that was shorter than what would be expected. The short queues means that the effects of congestion becomes less apparent. With longer queues the difference with and without QoS would increase. The increase comes from packets waiting in queues for longer periods of time.

In order to make CloudMAC more dynamic, the system was created so that WTPs and VAPs would be automatically detected. The detection for WTPs was made possible by having WTPs periodically generate announce packets so that the CloudMAC module could identify which ports WTPs are present on. The presence of a VAP on a port is revealed by its continuous generation of IEEE 802.11 beacon frames. With this aspect the CloudMAC networks topology could easily be changed with no configuration needed.

There was an issue with the WTP were it would run out of memory if too much traffic was sent through it. There was no time available to diagnose the memory issue. The cause of memory running out was suspected to be because of either, the queues being far too long, or that memory was not being freed up fast enough it was no longer needed.

6.1 Future work

Measure the delay between received association request frames (connecting station) and routing rule installation time. The delay made the system slow to respond to connecting stations. Measuring the delay would make it possible to profile which component was causing too much delay.

Evaluate IEEE 802.11e and IEEE802.11ae in an environment where all stations and AP implement both amendments. In this project only the WTP implemented IEEE 802.11e and IEEE 802.11ae. An evaluation in such an environment would give a more clear view of the benefits of QoS in IEEE 802.11.

Evaluate QoS using IEEE 802.11e, IEEE 802.11ae, and switch queue. In this project the switch queues and IEEE 802.11e and IEEE 802.11ae were evaluated independently. Testing both in the same setup would be interesting and should show more improvements.

Implement handover between WTPs. Handover is when one WTP takes over the acknowledgement of IEEE 802.11 frames from another WTP. When handover occurs the tunnel for the station that is causing the handover, would need to be changed to point to the new WTP. The CloudMAC module do not support moving stations.

Wireless offloading is when one WTP starts sending data to another WTP. The offloading can be caused by a congested or limited link for one WTP. Offloading would allow WTPs that do not have congested or limited link to help nearby WTPs that do.

7 References

[1] Open Networking Foundation, “Software-Defined Networking: The New Norm for Networks,” 2012.

[2] J. Vestin, “CloudMAC: Access Point Virtualization for Processing of WLAN Packets in the Cloud,” 2012.

[3] Linux Foundation, “OpenDaylight - An Open Source Community and Meritocracy for Software-Defined Networking,” 2013.

[4] Linux Foundation, “Becoming a Linux Foundation Collaborative Project Working Together to Help Projects Grow”.

[5] OpenFlow, “OpenFlow Switch Specification - Version 1.1.0 Implemented (Wire Protocol 0x02),” Open Networking Foundation, 2011.

[6] OpenDaylight Project, “OpenDaylight Controller:AD-SAL,” 26 2 2014. [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_Controller:AD-SAL. [Accessed 4 6 2015].

[7] OpenDaylight Project, “OpenDaylight Controller:MD-SAL,” 9 2 2015. [Online]. Available: https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL. [Accessed 4 6 2015].

[8] IEEE, “IEEE Std 802.11™-2012 (Revision of IEEE Std 802.11-2007),” IEEE Standards Association, 2012.

[9] “Wi-Fi CERTIFIED™ n: Longer-Range, Faster-Throughput, Multimedia-Grade Wi-Fi® Networks,” 2009.

[10] “IEEE 802.11ac: What does it mean for test?,” 2013.

[11] I. Poole, “Wi-Fi / WLAN Channels, Frequencies, Bands & Bandwidths,” [Online]. Available: http://www.radio-electronics.com/info/wireless/wi-fi/80211-channels-number-frequencies-bandwidth.php. [Accessed 4 6 2015].

[12] S. Choi, J. d. Prado, S. S. N and S. Mangold, “IEEE 802.11e Contention-Based Channel Access (EDCF) Performance Evaluation”.

[13] Wikipedia, “Short Interframe Space,” 15 2 215. [Online]. Available: http://en.wikipedia.org/wiki/Short_Interframe_Space. [Accessed 4 6 2015].

[14] Wikipedia, “DCF Interframe Space,” 26 2 2014. [Online]. Available: http://en.wikipedia.org/wiki/DCF_Interframe_Space. [Accessed 4 6 2015].

[15] “File Transfer Protocol,” 27 5 2015. [Online]. Available: http://en.wikipedia.org/wiki/File_Transfer_Protocol. [Accessed 4 6 2015].

[16] “IEEE Std 802.11ae™-2012 (Amendment to IEEE Std 802.11™-2012),” 2012.

[17] J. Berg, G. Harris and L. Rodriguez, “Radiotap,” [Online]. Available: http://www.radiotap.org/Radiotap. [Accessed 19 5 2015].

[18] “hostapd Linux documentation page,” [Online]. Available: https://wireless.wiki.kernel.org/en/users/documentation/hostapd. [Accessed 4 6 2015].

[19] devik and D. Cohen, “HTB Linux queuing discipline manual - user guide,” 5 5 2002. [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm.

[20] M. Devera and B. Hubert, “tc-htb(8) - Linux man page,” [Online]. Available: http://linux.die.net/man/8/tc-htb. [Accessed 16 5 2015].

[21] “Production Quality, Multilayer Open Virtual Switch,” [Online]. Available: http://openvswitch.org/. [Accessed 4 6 2015].

[22] “ovs-vsctl - utility for querying and configuring ovs-vswitchd,” [Online]. Available: http://openvswitch.org/support/dist-docs/ovs-vsctl.8.txt. [Accessed 17 5 2015].

[23] “ovs-ofctl - administer OpenFlow switches,” [Online]. Available: http://openvswitch.org/support/dist-docs/ovs-ofctl.8.txt. [Accessed 17 05 2015].

[24] NAYARASI, “802.11 Mgmt : Action Frames,” 9 10 2014. [Online]. Available: http://mrncciew.com/2014/10/09/802-11-mgmt-action-frames/. [Accessed 9 6 2015].

[25] P. A. Lambert, “Management Frame Analysis,” 2010.

[26] Wikipedia, “IEEE 802.11p,” 19 4 2015. [Online]. Available: http://en.wikipedia.org/wiki/IEEE_802.11p. [Accessed 9 6 2015].

[27] IEEE, “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications - Amendment 5: Spectrum and Transmit Power Management Extensions in the 5 GHz band in Europe,” IEEE, 2003.

[28] Wikipedia, “IEEE 802.11y-2008,” 1 4 2014. [Online]. Available: http://en.wikipedia.org/wiki/IEEE_802.11y-2008. [Accessed 9 6 2015].

[29] B. G. Lee and S. Choi, “Broadband Wireless Access and Local Networks: Mobile WiMax and WiFi,” Artech House, 2008, pp. 524-525.

[30] WI-Fi Alliance, “Wi-Fi CERTIFIED™ TDLS:Easy-to-use, security-protected direct links to improve performance of Wi-Fi® devices,” WI-Fi Alliance, 2012.

[31] D. Simone, “802.11k makes WLANs measure up,” 29 3 2004. [Online]. Available: http://www.networkworld.com/article/2331683/tech-primers/802-11k-makes-wlans-measure-up.html. [Accessed 9 6 2015].

[32] Wikipedia, “IEEE 802.11r-2008,” 4 6 2015. [Online]. Available: http://en.wikipedia.org/wiki/IEEE_802.11r-2008. [Accessed 9 6 2015].

[33] “Security Procedure for Long Sleeper,” Mentor IEEE, 2013.

[34] Wikipedia, “Wireless mesh network,” 27 4 2015. [Online]. Available: http://en.wikipedia.org/wiki/Wireless_mesh_network. [Accessed 9 6 2015].

[35] Linux Wireless, “IEEE 802.11s,” 26 1 2015. [Online]. Available: https://wireless.wiki.kernel.org/en/developers/documentation/ieee80211/802.11s. [Accessed 9 6 2015].

[36] G. Strutt and J. Kruys, “Multihop Management Frames and other matters,” Mentor IEEE, 2007.

[37] M. Zhao and J. Walker, “IEEE P802.11 Wireless LANs,” IEEE Mentor, 2009.

[38] “wireless.kernel.org – Users,” 29 1 2015. [Online]. Available: https://wireless.wiki.kernel.org/en/users. [Accessed 4 6 2015].

[39] T. Høiland-Jørgensen. [Online]. Available: https://tohojo.github.io/netperf-wrapper.1.html. [Accessed 18 5 2015].

[40] Canonical Ltd, “Ubuntu for desktops,” [Online]. Available: http://www.ubuntu.com/desktop. [Accessed 4 6 2015].

[41] D-Link, “Wireless N USB Adapter,” [Online]. Available: http://us.dlink.com/products/connect/wireless-n-usb-adapter/. [Accessed 4 6 2015].

[42] Intel, “Intel® Wireless WiFi Link 4965AGN,” [Online]. Available: http://www.intel.com/products/wireless/wireless_n/overview.htm. [Accessed 5 6 2015].

[43] OpenDaylight Project, “OPENDAYLIGHT DELIVERS OPEN SOURCE SOFTWARE TO ENABLE SOFTWARE-DEFINED NETWORKING,” 4 2 2014. [Online]. Available: http://www.opendaylight.org/announcements/2014/02/opendaylight-delivers-open-source-software-enable-software-defined-networking.

[44] OpenDaylight Project, “OPENDAYLIGHT PAVES WAY FOR INNOVATION IN SOFTWARE-DEFINED NETWORKING WITH LATEST OPEN SOURCE SOFTWARE RELEASE,” 29 9 2014. [Online]. Available: http://www.opendaylight.org/announcements/2014/09/opendaylight-paves-way-innovation-software-defined-networking-latest-open.

[45] OSGI Alliance, “Agility and Modularity: Two Sides of the Same Coin,” 2014.

[46] Apache Software Foundation, “Apache Karaf Overview,” [Online]. Available: http://karaf.apache.org/manual/latest/overview.html. [Accessed 4 6 2015].

[47] “The Apache Software Foundation: Community-led development since 1999,” [Online]. Available: http://www.apache.org/foundation/. [Accessed 4 6 2015].

Appendix A - Wireless Termination Point Daemon Manual

The purpose of the daemon is to make it possible to configure which MAC addresses a wireless termination point (WTP) acknowledges. The daemon creates thirty two (32) virtual network interfaces and associates a MAC address with each interface in order to acknowledge frames. When an interface is not in use it is disabled. In order to use the daemon connect on TCP port 1999 and issue a command in ASCII, the following command are available.

Syntax	Description
status index INDEX	Gives information about the state of a interface lease. Looks up a lease given an index.
status mac MAC	Gives information about the state of a interface lease. Looks up a lease by given a MAC address.
status	Gives information about the status of all interfaces and their leases.
records	Gives information about the state of all interface leases.
lease MAC TIMEOUT	Creates a lease for a interface for a VAP when given a MAC address and a timeout.
stop MAC	Ends the lease of a interface matching the given MAC address.

Note: Underlined words are parameters and should be replaced with relevant values.

Appendix B - AD-SAL & MD-SAL Comparison

The comparison is of how a flow rule can be created in AD-SAL versus MD-SAL. For MD-SAL there exists two slightly differing ways of creating a flow rule. For AD-SAL a service called IFlowProgrammerService is used (section 0) for MD-SAL a service called SalFlowService (section 0) or a service called DataBrokerService (section 0) can be used.

All three pieces of code creates the same flow rule. The rule consists of matching an Ethernet type and dropping all matched packets. The rule will persist for at most 3600 seconds (hard timeout) if there is constant traffic or whenever there is a period of 60 seconds (idle timeout) where no match is made. As can be seen clearly is that while MD-SAL does things a bit more “fancy” it does so at the cost of added complexity and reduced readability of the code needed to be written. This is a clear theme throughout the MD-SAL approach, and the main reason why I at opted for AD-SAL in instead of MD-SAL.

Appendix B.1 - AD-SAL(IFlowProgrammerService) Code – 10 Lines

```
Match match = new Match();
Flow flow = new Flow();
Drop drop_Action = new Drop();

match.setField(MatchType.DL_TYPE, (short)0x1337);

flow.setActions(new ArrayList<Action>());
flow.addAction(drop_Action);
flow.setMatch(match);
flow.setHardTimeout((short)3600);
flow.setIdleTimeout((short)60);

flowProgrammer.addFlow(connector.getNode(), flow);
```

Appendix B.2 - MD-SAL (SalFlowService) Code – 35 Lines

```
AddFlowInputBuilder flow = new AddFlowInputBuilder();

// Create the list of actions that should be performed when packets are
matched.
List<Action> list = new ArrayList<Action>();
List<Instruction> instructions = new ArrayList<Instruction>();
InstanceIdentifier<Node> node =
notification.getIngress().getValue().firstIdentifierOf(Node.class);
DropActionBuilder drop_action_builder = new DropActionBuilder();
DropActionCaseBuilder drop_action_case_builder = new
DropActionCaseBuilder();
ActionBuilder action_builder = new ActionBuilder();

drop_action_case_builder.setDropAction(drop_action_builder.build());
action_builder.setAction(drop_action_case_builder.build());
action_builder.setOrder(0);
list.add(action_builder.build());

// Create a instruction containing the list of actions to perform on
matched packets.
ApplyActionsBuilder apply_actions_builder = new ApplyActionsBuilder();
ApplyActionsCaseBuilder apply_actions_case_builder = new
ApplyActionsCaseBuilder();
InstructionBuilder instructions_builder = new InstructionBuilder();

apply_actions_builder.setAction(list);
apply_actions_case_builder.setApplyActions(apply_actions_builder.build());
instructions_builder.setOrder(0);
instructions_builder.setInstruction(apply_actions_case_builder.build());
instructions.add(instructions_builder.build());
// Create the Ethernet match.
MatchBuilder match = new MatchBuilder();
EtherType ether_type = new EtherType((long) 0x1337);
EthernetTypeBuilder ether_type_builder = new
EthernetTypeBuilder().setType(ether_type);
EthernetMatchBuilder ether_match_builder = new
EthernetMatchBuilder().setEthernetType(ether_type_builder.build());

match.setEthernetMatch(ether_match_builder.build());

// Finalize the flow rule.
flow.setNode(new NodeRef(node));
flow.setMatch(match.build());
flow.setHardTimeout(3600);
flow.setIdleTimeout(60);
flow.setTableId(notification.getTableId().getValue());
flow.setInstructions(new
InstructionsBuilder().setInstruction(instructions).build());

flowService.addFlow(flow.build());
```

Appendix B.3 - MD-SAL (DataBrokerService) Code – 40 Lines

```
FlowBuilder builder = new FlowBuilder();

// Create the list of actions that should be performed when packets are
matched.
List<Action> list = new ArrayList<Action>();
List<Instruction> instructions = new ArrayList<Instruction>();
DropActionBuilder drop_action_builder = new DropActionBuilder();
DropActionCaseBuilder drop_action_case_builder = new
DropActionCaseBuilder();
ActionBuilder action_builder = new ActionBuilder();

drop_action_case_builder.setDropAction(drop_action_builder.build());
action_builder.setAction(drop_action_case_builder.build());
action_builder.setOrder(0);
list.add(action_builder.build());

// Create a instruction containing the list of actions to perform on
matched packets.
ApplyActionsBuilder apply_actions_builder = new ApplyActionsBuilder();
ApplyActionsCaseBuilder apply_actions_case_builder = new
ApplyActionsCaseBuilder();
InstructionBuilder instructions_builder = new InstructionBuilder();

apply_actions_builder.setAction(list);
apply_actions_case_builder.setApplyActions(apply_actions_builder.build());
instructions_builder.setOrder(0);
instructions_builder.setInstruction(apply_actions_case_builder.build());
instructions.add(instructions_builder.build());
// Create the Ethernet match.
MatchBuilder match = new MatchBuilder();
EtherType ether_type = new EtherType((long) 0x1337);
EthernetTypeBuilder ether_type_builder = new
EthernetTypeBuilder().setType(ether_type);
EthernetMatchBuilder ether_match_builder = new
EthernetMatchBuilder().setEthernetType(ether_type_builder.build());

match.setEthernetMatch(ether_match_builder.build());

// Create the path to where the flow rule will be added.
String id_str = String.valueOf(flowIdInc.getAndDecrement());

// Path to the node in which we want to add a rule.
InstanceIdentifier<Node> node_path =
notification.getIngress().getValue().firstIdentifierOf(Node.class);

// Path to the table where in we want to add the rule.
InstanceIdentifier<Table> table_path =
node_path.augmentation(FlowCapableNode.class).child(Table.class, new
TableKey(notification.getTableId().getValue()));

// Path to where we want to add the flow.
InstanceIdentifier<Flow> flow_path = table_path.child(Flow.class, new
FlowKey(new FlowId(id_str)));
```

```
// Finalize the flow rule.
builder.setInstructions(new
InstructionsBuilder().setInstruction(instructions).build());
builder.setPriority(3000);
builder.setHardTimeout(3600);
builder.setIdleTimeout(60);
builder.setMatch(match.build());
builder.setId(new FlowId(id_str));
builder.setTableId(notification.getTableId().getValue());

// Commit the flow.
DataModificationTransaction addFlowTransaction = data.beginTransaction();
addFlowTransaction.putConfigurationData(flow_path, builder.build());
addFlowTransaction.commit();
```