TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Elektrotechnik und Informationstechnik
Professur Kommunikationsnetze

Masterarbeit

Design and implementation of an SDN based authentication and separation mechanism for WiFi users

Nishant Ravi

Chemnitz, April 26, 2017

Autor:	**Nishant Ravi**
Email:	**nisr@hrz.tu-chemnitz.de**
Matrikelnummer:	**355151**
Prüfer:	**Prof. Dr.-Ing. Thomas Bauschert**
Betreuer:	**Dipl.-Ing. Florian Schlegel**
Ausgabedatum:	Feb 21, 2017
Abgabedatum:	April 26, 2017

Abstract

The ever-increasing use of data services on mobile devices, places increased demands on existing networks. Especially in busy areas, such as shopping centers, office buildings or in event centers, the existing network coverage by UMTS and LTE is no longer sufficient. It is, therefore, obvious to direct some traffic through other radio standards. In this case, WLAN is particularly suitable because those frequencies are free to use without any license restrictions and since most mobile devices have long since supported this. However, an uncontrolled number of WLAN access points can interfere with each other. It is, therefore, desirable to install only one set of access points at these locations and manage them centrally. The research project BIC-IRAP (Business Indoor Coverage Integrated Radio Access Points) is a project aimed at providing a seamless coupling between LTE and WLAN.

The separation of data traffic is an important aspect when using shared hardware. No direct data exchange between the networks of different mobile radio providers should be possible. Likewise, the networks of different businesses or companies should be kept strictly separate from each other. Classic VLANs would be used for this purpose. Within the scope of the BIC-IRAP project, however, there were considerations to control parts of the network using SDN. Therefore, the goal of this master thesis is to operate an access point (AP) on an OpenFlow controlled switch. Users can be authenticated against a RADIUS server. The AP should supply at least two separate networks. If possible, the separation of data traffic should already take place in the AP. Optionally the AP should provide Hotspot 2.0 functionality.

The conceptualization and implementation must be documented in detail. The optional components are carried out in consultation with the supervisor. The successful completion of the work is a test set-up. The achievable performance characteristics must be recorded.

Contents

List of Figures

1 Introduction

The penetration of mobile internet users has increased many fold from a few thousands to millions over a short span of time. Due to the increasing demand for data among subscribers, mobile operators are pushed to go beyond boundaries to provide efficient and reliable data service to their customers. Although, the existing network services such as UMTS and LTE can handle larger data capacity, their coverage is not always sufficient in crowded places such as office buildings, convention centers, shopping malls etc. There is an urgent need to find a solution on how to offload the mobile data traffic over to other radio standards.

In such case, WLAN is an existing radio standard that has already been deployed in large numbers and has been supported by millions of devices lately. One unique advantage of using WLAN over other radio standards would be its license free usage of its radio frequency for commercial use. This WLAN standard, when deployed in a controlled manner can support data traffic routed from the mobile services. The IEEE 802.11 WLAN has already been widely used for commercial enterprises ranging from office networks, shopping malls to educational institutions etc. The deployment rages from a few dozens to hundreds of access points (APs), which serve many users through multitude of devices ranging from mobile devices, laptops to printers and other connected hardware. These networks also provide varied set of services that includes authentication, authorization and accounting (AAA), dynamic channel reconfiguration, interference management, security such as intrusion detection and prevention and also providing quality of service.

These enterprise WLAN AP's are usually centrally managed through a controller. The task now is to find a solution to seamlessly direct traffic between LTE and WLAN. The research project BIC-IRAP (Business Indoor Coverage Integrated Radio Access Point) is currently aimed at providing a solution for the seamless coupling between LTE and WLAN.

The growing adoption of Software Defined Networking in the recent years has given rise to providing unique solutions without depending too much on hardware. The advantage of using SDN is that, it separates the network control plane from the physical network topology and uses software control flow to define how traffic is forwarded in the network. For example, the routing table and the flow control of a switch can be easily controlled remotely through a software controller. The characteristic features of SDN is possible due to the use of protocols such as OpenFlow, a standardized protocol that is used by many open source controllers to manipulate the flow tables of network switches. This provides more flexibility to programmatically control the behavior of network switches by building network applications that talk to the network controller. Any OpenFlow enabled switch from any vendor provides a common interface to be manipulated via a controller, thus providing flexibility and simplified network management.

1.1 Contribution

This thesis provides a novel approach towards separating the data traffic between the different network providers within an access point. This is made possible through the simple, yet effective use of OpenFlow protocol that enables the development of different enterprise WLAN services as applications such as, using software defined network controllers. The performance benefits achieved though this system is possible without any changes to the existing 802.11 client. The proposed system is compatible with the existing enterprise WLAN security protocols like WPA2 enterprise.

1.2 Results

The expected outcome of this thesis is to demonstrate a prototype system that runs an Access Point (AP) on an OpenFLow controlled switch. The AP also provides enterprise grade authentication system using WPA2 enterprise alongside a RADIUS server, and host two separate networks.

1.3 Research Context[1]

The research described in this thesis was done based on the BIC-IRAP project which is focused on combining the strengths of LTE and WLAN seamlessly. Through the integration of small and micro cells of LTE with WLAN in the BIC-IRAP system, the two radio technologies are available through a single dynamically configurable hardware configuration.

1.4 Thesis Structure

This thesis report is organized as follows, Chapter 2 describes the background for this thesis. Chapter 3 describes in detail about SDN and the different types of controllers that are in use today. Chapter 4 talks about the control and authentication mechanism such as the protocols and technologies used. Chapter 5 explains about the environment required to build the system such as the tools and software's. Chapter 6 describes in detail how the application is developed, from conceptualization to coding in python. Chapter 7 shows how the system is being implemented. Chapter 8 presents the results obtained from the system after series of testing and enhancement. Chapter 9 concludes the thesis and describes further improvements and drawbacks of this method.

2 Background

This chapter discusses about the background to understand this thesis, it includes an introduction to software defined networking, explains 802.11 protocol, and a brief introduction on Hotspot 2.0 and BIC-IRAP project.

2.1 Software Defined Networking [2]

SDN is nothing but the physical separation of the network control plane from the forwarding plane. The control plane consists of all the logic (or instructions) that the switch requires for correctly setting up a forwarding plane.

Traditionally, the vendor has the control over the instructions necessary for signaling since the devices run a proprietary firmware within the switch. This makes the devices non-interoperable with other vendors, thus hampering flexibility. Though most of these switches provide SNMP based management solution via Command Line Interface (CLI), they still do not allow the introduction of custom control plane function or protocol into the switch. This makes experimenting with new protocols cumbersome. Software Defined Networking aims to alleviate these problems by making the switched forwarding plane to be easily accessible remotely and modifiable using the OpenFlow protocol. Any third-party software can take advantage of this open protocol to manage and orchestrate an entire network.

SDN architecture generally has three components or groups of functionality as shown in the figure 2.1.

- **Application Layer:** Consists of programs that communicate the behaviors and needed resources with the SDN controller via the application protocol interfaces (API's). It can also build an abstracted view of the network by collecting information from the controller.
- **Control Layer:** This logical layer functions as a relay that sends the instructions or resources sent by the application layer to the networking components.

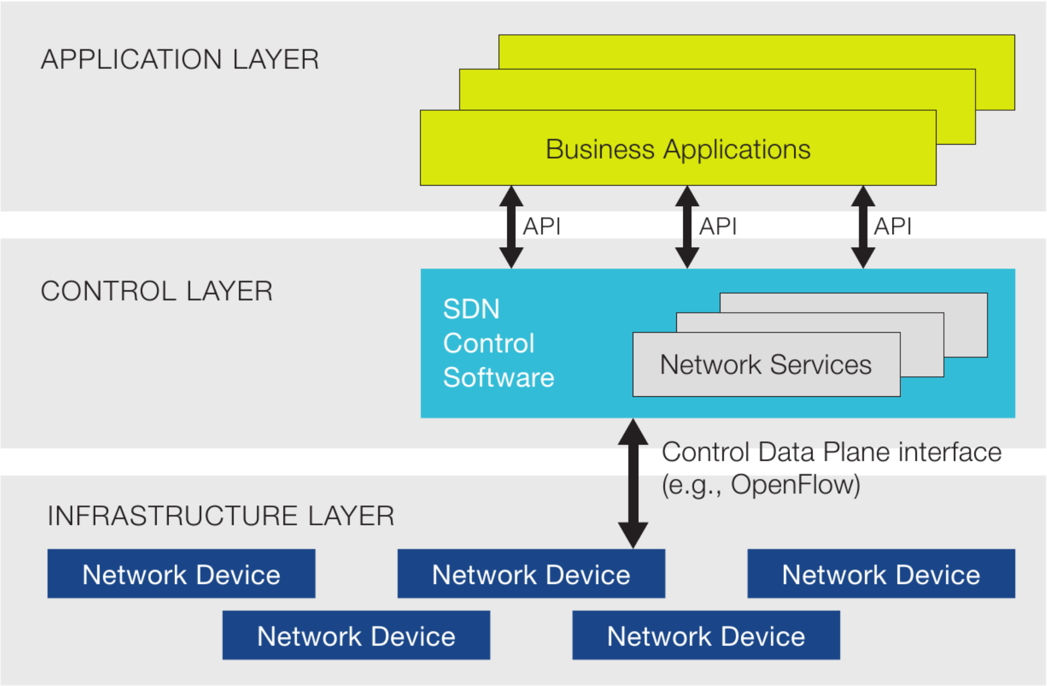


Figure 2.1: SDN architecture diagram[17]

- **Infrastructure Layer:** This holds the SDN networking devices that control the forwarding and data processing capabilities of the network including the function to forward and process the data paths.

2.2 IEEE 802.11 MAC [3]

The IEEE 802.11 Media Access Control Layer (MAC) [3] defines the protocol for stations to establish connections with each other and transmit data frames. Medium access in 802.11 is performed by a distributed coordination function (DCF), which uses carrier sense multiple access with collision avoidance (CSMA/CA) to enable random medium access among all contending stations (STAs). Hence, it reduces the amount of collisions. Logically, the MAC is divided into two parts, an upper MAC, and a lower MAC. The upper MAC handles management frames, which include probe, authentication, association requests and their corresponding responses. The lower MAC handles control frames, which includes acknowledgement (ACK) frames, along with request-to-send (RTS) and clear-to-send (CTS) frames. The frames handled by the lower MAC have real-time constraints. For instance, ACK frame timeouts are within the order of micro-seconds. For this reason, control frames are handled and generated within hardware. Management frames, however, have softer time constraints, and can be handled in software locally (as is the case in Linux systems that use hostapd [34]), or remotely (as is the case when using a centralized WLAN controller [35]).

An 802.11 based wireless interface can operate under the following operating modes: STA (client), access point (AP), mesh, ad-hoc and item Monitor mode. The most common mode of operation is the infrastructure mode (which includes enterprise WLAN environments). In this mode of operation, clients connect to the AP using a series of message exchanges in a process called “association”. The decision on which AP to associate with is left entirely to the client. Clients learn about APs either passively through beacon frames that are periodically broadcasted by the access points, or actively by performing a probe scan.

In a probe scan, clients first send out probe request frames over all channels. APs that receive these frames and are willing to accept a connection from a client respond with a probe response frame. All APs from which the client receives probe responses are candidates for the client to associate with. Next, the client sends an authentication frame, and waits for an authentication response from the AP. This is followed by the client sending an association request, and receiving an association response from the AP. If the network is operating in open authentication mode, the client is considered to be associated at this point, and can now transmit data frames to be forwarded by the AP. If the AP is configured to use WPA, WPA2, or WPA2 Enterprise, the corresponding 802.1X [34] handshake is performed after the association phase before clients can forward data frames through the AP.

2.3 Hotspot 2.0 [4]

It is a new wireless network standard that is designed to make connections to public Wi-Fi hotspots more easy and secure. They are already supported on many mobile devices running some of the popular operating systems such as Windows 10, Mac OS 10.9 or newer, Android 6.0 or newer, and iOS 7 or newer.

The main purpose of Hotspot 2.0 is to provide seamless mobility like cellular style “roaming” for Wi-Fi networks. The device will automatically connect to the available networks based on the networks partners on the home networks while roaming globally. This is made possible using the latest 802.11u [36] protocol designed for the same purpose. The organization WIFI Alliance also call this as Passpoint [37].

3 Software Defined Networking

The Open Network foundation, a non-profit organization, has been undertaking research for the past couple of years in designing and standardizing open network components such as OpenFlow, SDN etc. The Open Network foundation claims that, after the components were rolled out to a variety of network devices and software's from different vendors. It has been delivering substantial benefits to both enterprises and carriers such as: [38]

- **Directly Programable:**Network directly programmable because the control functions are decoupled from forwarding functions, which enable the network to be programmatically configured by proprietary or open source automation tools.
- **Centralized Management:** Network intelligence is logically centralized in SDN controller software that maintains a global view of the network, which appears to applications and policy engines as a single, logical switch.
- **Reduce CapEx:** Software Defined Networking potentially limits the need to purchase purpose-built, ASIC-based networking hardware, and instead supports pay-as-you-grow models
- **Reduce OpEX:** SDN enables algorithmic control of the network of network elements (such as hardware or software switches/routers that are increasingly programmable, making it easier to design, deploy, manage, and scale networks. The ability to automate provisioning and orchestration optimizes service availability and reliability by reducing overall management time and the chance for human error.
- **Deliver Agility and Flexibility:** Software Defined Networking helps organizations rapidly deploy new applications, services, and infrastructure to quickly meet changing business goals and objectives.
- **Enable Innovation:** SDN enables organizations to create new types of applications, services, and business models that can offer new revenue streams and more value from the network.

3.1 Existing SDN Controllers

For this Master thesis, a few available SDN controllers are first studied for its functionality that can be manipulated for data path segregation. A brief overview on each controller is discussed in the following sections.

3.1.1 Ryu Controller [5]

Ryu is a component-based software defined networking framework. It provides software components with well-defined Application Protocol Interfaces (APIs) that make it easy to create new network management and control applications. The component that is of particular interest for this master thesis is the switching hub using OpenFlow.

Switching hubs have a variety of functions like learning the MAC address of the host connected to a port and retaining it in the MAC address table. When receiving packets, the packets that are addressed to a host which is already learned previously is transfered directly to the port connected to the host. Also, when the received packets are addressed to an unknown host, then the switch floods these packets to all ports.

The main reason to choose RYU over other controllers is due to its customizability and easy to create core applications using Python. RYU allows users to modify core functions or use these functions to create custom applications that suits specific needs, in this case, it was used to create a switching application that can segregate users within the OpenVswitch, instead of being controlled each time by the controller.

The software components provided by RYU with well-defined Application Programming Interface (API's) as shown in the figure 3.1, makes it easy for developers to create custom network management or control applications. The existing components can be quickly and easily be modified or can develop a custom component so that the underlying network can meet the changing demands of the application. RYU is designed to increase the agility of network by being more easily manageable and adapt how traffic is handled.

RYU Controller is supported by the telecommunications company Nippon Telephone and Telegraph (NTT) of Japan and has a strong open source community that maintain and manage the code which is hosted on GitHub. OpenStack, an open source cloud operating system that provides Information as a service (IaaS) [39] also supports deployment of RYU as network controller in its cloud operating systems.

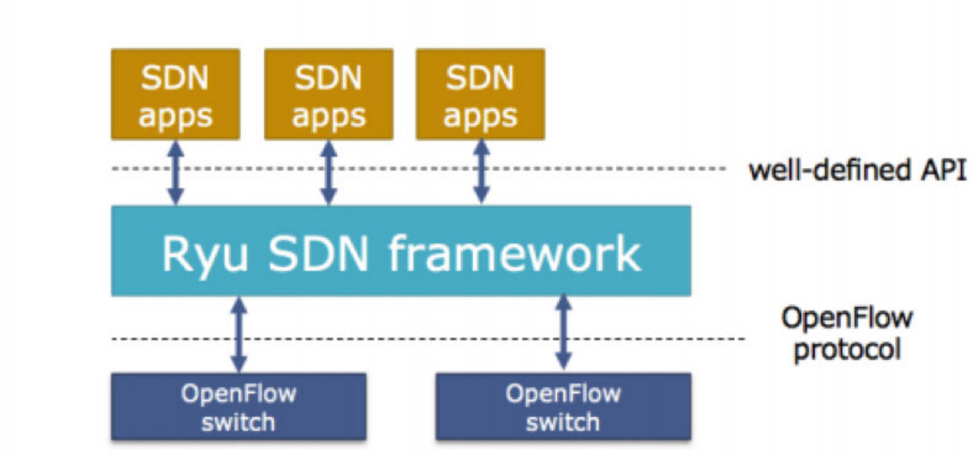


Figure 3.1: RYU SDN Controller Framework [18]

3.1.2 Floodlight Controller [6]

It is yet another open source SDN controller similar to RYU. The benefit of using this controller is its ability to easily develop applications using Java, which is widely used for high level programming by developers and to adapt the software as per requirement. Floodlight offers Representational state transfer application program interfaces (REST APIs) which help developers to easily program interfaces with the product.

Floodlight is used to run as the network backend for OpenStack. When used with the Neutron plugin with OpenStack, the Floodlight controller functions as a network-as-a-service model with the help of REST API offered by Floodlight. The diagram 3.2 shows the architecture of Floodlight controller.

The architecture consist of three tiers. The Application tier consist of aplications that work with the controller such as OpenStack, circuit pusher etc. The Control Plane tier is the core of the controller where Floodlight resides, it manages the applications and the switches using OpenFlow. The Northbound APIs also known as REST APIs are used for efficient management of communication between the Floodlight controller and the services and applications running on the network. The Data Plane tier consist of switches such as the Hypervisor (Virtual Switch used by virtual machines) and physical switches. The Indigo Data Plane interface is a software developed by Floodlight that make switch hardware OpenFlow compatible.

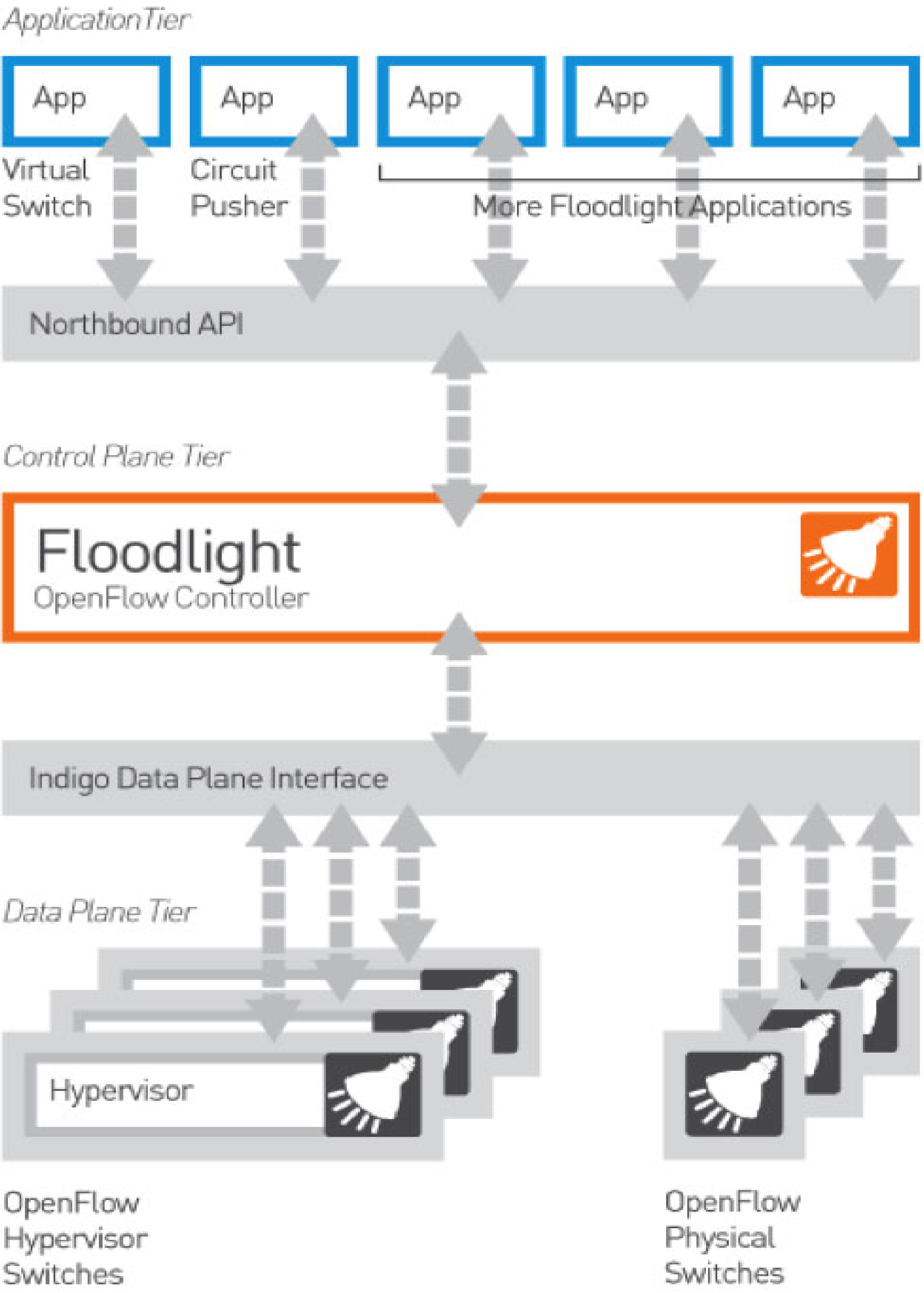
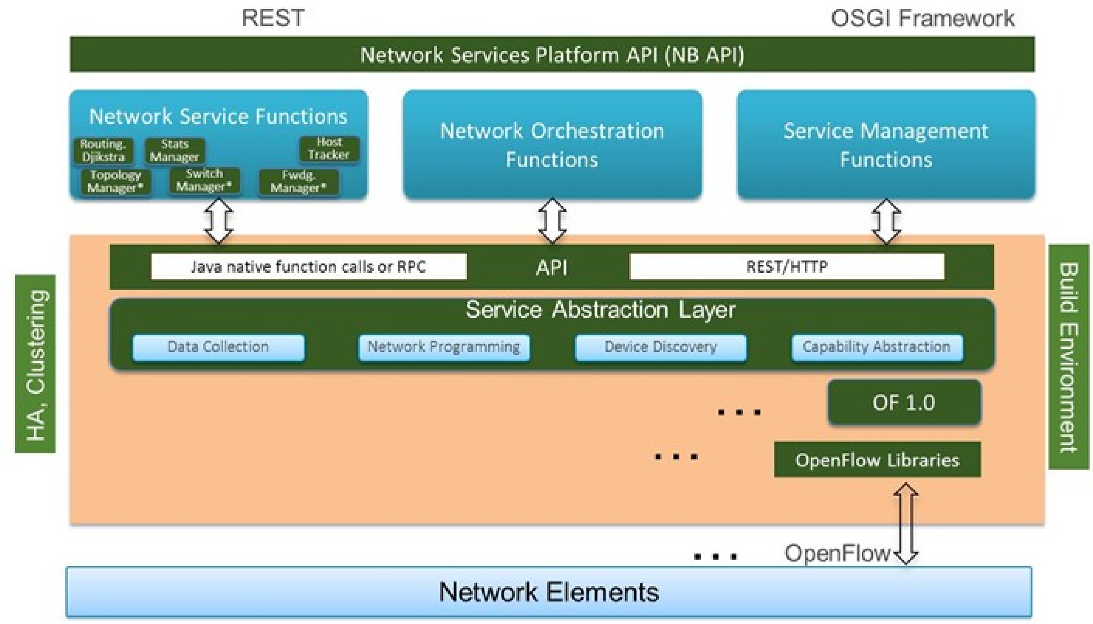


Figure 3.2: Floodlight Controller architecture [19]



* limited to functionality that is possible via Open Flow 1.0

Figure 3.3: OpenDaylight Architecture Framework [20]

3.1.3 OpenDaylight

OpenDaylight controller is based on JVM, similar to Floodlight, which was a derivative of OpenDaylight that can be deployed on any systems that supports Java. OpenDaylight controller uses the following tools as its framework:

- **Maven:** OpenDaylight uses Maven, which uses Project Object Model to script the dependencies between the bundles for easier build automation.
- **OSGi:** It works as the back-end for OpenDaylight as it loads bundles dynamically and packages JAR files and binding them together for exchange of information.
- **JAVA interfaces:** They are used for event listening, specifications, and forming patterns.
- **REST APIs:** These are the northbound APIs that manage the topology, flow program, host tracking, static routing and so on.

The figure 3.3 shows the framework of OpenDaylight with the above tools mentioned.

3.2 Applications of SDN

Many research efforts have been conducted until now in writing SDN applications. Jose et. al. [40] propose using commodity OpenFlow enabled switches for traffic measurement. The authors propose a framework where a collection of rules are installed on OpenFlow switches, and having a controller track the corresponding flow match counters. The controller can then draw inferences from the counters and dynamically tune the rules as required in order to identify different traffic aggregates.

Resonance [41] is another application that uses programmable switches to enforce access control in the network. The authors try to prove that today's enterprise networks rely on different combinations of middle boxes, intrusion detection systems, and network configurations in order to enforce access control policies, whilst placing a burden on end-hosts in the system to remain patched and secure. The proposed system uses the SDN approach comprising of programmable switches and a controller, which together implement a network monitoring framework, a policy specification framework, and the ability to trigger specific actions at the switch level.

OpenSAFE [42] is a framework that enables network monitoring using OpenFlow. It addresses the problem of routing traffic for network analysis in a reliable manner without affecting normal traffic.

Hedera [43] is an adaptive flow scheduling system for data center networks. The premise for Hedera is that existing IP multipathing techniques used in data centers usually rely on per-flow static hashing, which can lead to under-utilisation of some network paths over time due to hash collisions. The system works by detecting large flows at the edge switches of a data center, and using placement algorithms to find good paths for the flows in the network. Experiments performed using simulations indicate significant improvements over static load balancing techniques.

In the paper *OpenFlow based server load balancing gone wild* [44], the authors address the problem of server load balancing using OpenFlow switches. The number of flow entries that can be saved on an OpenFlow switch is much less than the number of unique flows that a switch might need to handle in data center workloads. Thus, micro flow management using per-flow rules is not practical for performing flow distribution between different servers using a switch. The authors take advantage of OpenFlow's wildcard based rules capability, and propose algorithms to compute concise wildcard rules that achieve a specific distribution of traffic.

These are some of the applications that have been written for SDN controllers but none of them address the challenge to dynamically redirect packets in real time, based on different clients and their credentials used for authentication. This thesis proposed to build one such application that can segregate packets coming from different clients in such a way that there is no possible connection between multiple clients associated within the same access point.

3.3 Open vSwitch [7]

It is a production quality multilayer virtual switch, designed to enable massive automation through programmatic extension. It also supports standard management interfaces and protocols such as NetFlow, sFlow, CLI, port mirroring, VLANs, LACP etc. In addition to this, it is also designed to support distribution across multiple physical servers similar to VMWare's vNetwork distributed vswitch. Open vSwitch was developed by the Linux foundation and is licensed under Apache 2.0.

The virtual switch is a software layer that resides in a server that is hosting virtual machines. VM's and also containers such as Docker have logical and virtual Ethernet ports. These logical ports connect to a virtual switch. The diagram 3.4 shows the features of an Open vSwitch. [45]

From the management and control perspective, Open vSwitch leverages on the OpenFlow and the Open vSwitch Database (OVSDB) management protocol, which means it can work as both a soft switch running within the hypervisor and as the control stack for switching operations on the physical switches. OVS is also used in SDNs deployed in data centers where it connects all the virtual machines (VMs) within a hypervisor instance on a server. It is the ingress point in overlay networks running on top of the physical networks in the data center and it is the first point of entry for all the VM's sending traffic to the network. In data center SDN deployments, using OVS for virtual networking is considered the core element since its main use case is a multi-tenant network virtualization. In some service chaining use cases, OVS is sometimes used to direct traffic between network functions.

Open vSwitch is designed in such a way that it is meant to be managed and controlled by third-party controllers and managers. OVS can also directly work with OpenStack using a plugin or directly from an SDN controller, such as OpenDaylight. It is also possible to deploy OVS on all servers in an environment and let it operate with the MAC learning functionality.

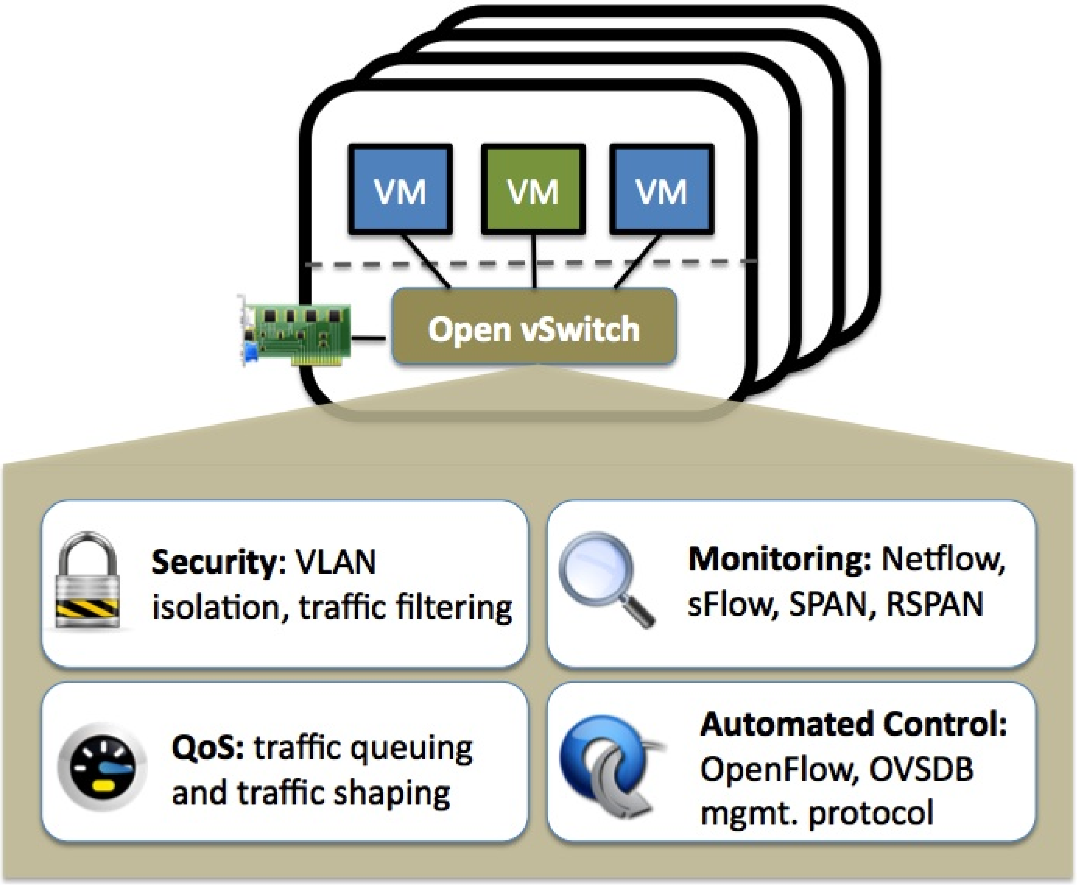

VM
VM
VM
Open vSwitch
Security: VLAN isolation, traffic filtering
Monitoring: Netflow, sFlow, SPAN, RSPAN
QoS: traffic queuing and traffic shaping
Automated Control: OpenFlow, OVSDB mgmt. protocol

Figure 3.4: Open vSwitch features [21]

4 Control and Authentication Mechanism

In this chapter, the softwares and protocols used for providing authentication and for controlling the access point is discussed. A brief introduction to OpenWrt and the protocols, OpenFlow and RADIUS is described in the flowing topics.

4.1 OpenWrt [8]

OpenWrt is a Linux distribution that works like any other Linux distro designed for embedded devices. OpenWrt offers bult-in package manager that allows installing packages from its repository or manually build custom firmware file using custom-built package. The packages can contain many supported applications like an SSH server, VPN, traffic-shaping application, enterprise wireless solutions, BitTorrent client and even a hotspot manager.

OpenWrt is designed for power users who require customizability over the stock firmware provided by the manufacturer. There are many custom firmwares such as DD-WRT available for most popular routers. For this thesis, OpenWrt is chosen because of its flexibility and stability than most custom firmwares that are available for many hardware vendors.

OpenWrt has many features that can be mentioned but are out of scope for this thesis. A run-down version of the most relevant features are listed below that are related to this thesis.

- **SSH server for terminal access:** Provides SSH server, which allows to connect directly to the routers terminal via SSH and remote configuration is also possible when the router is configured to connect with the internet.
- **Capture and Analyse network Traffic:** Tcpdump tool is included in the build to analyze the packets that are traversing through the router. The tool can also be used to create packet logs that can be opened in packet analyzer tools such as Wireshark.
- **GUI Interface:** OpenWrt also includes a GUI interface for managing most of the routers configuration, the built in one is named as LuCi.

- **OpenvSwitch:** The virtual switch is also available as a package on OpenWrt repository that is used in this thesis to be installed in the router for creating a virtual switch within the router that can also work along with the physical switches present.
- **Wireless Utilities:** OpenWrt provides many different packages for managing wireless sockets in the router. For this thesis, Hostapd package is chosen because of its fully featured support for a wide range of authentication mechanisms such as IEEE 802.1x/WPA/EAP/RADIUS with EAP protocol. It can be configured in the file located at /etc/hostapd.conf in the routers folder.
- **Freeradius:** The open source RADIUS server is also available as a package for the OpenWrt build but is not used in the firmware for this thesis because of memory unavailability in the TP Link WDR-4300 router.
- **MySQL:** The is a fully featured MySql server. It is also available as a package that can be installed on the router but again could not be used in this thesis due to memory restrictions in the router.

The figure 4.1 shows the SSH interface of OpenWrt terminal and the figure 4.2 shows the LuCi web interface.

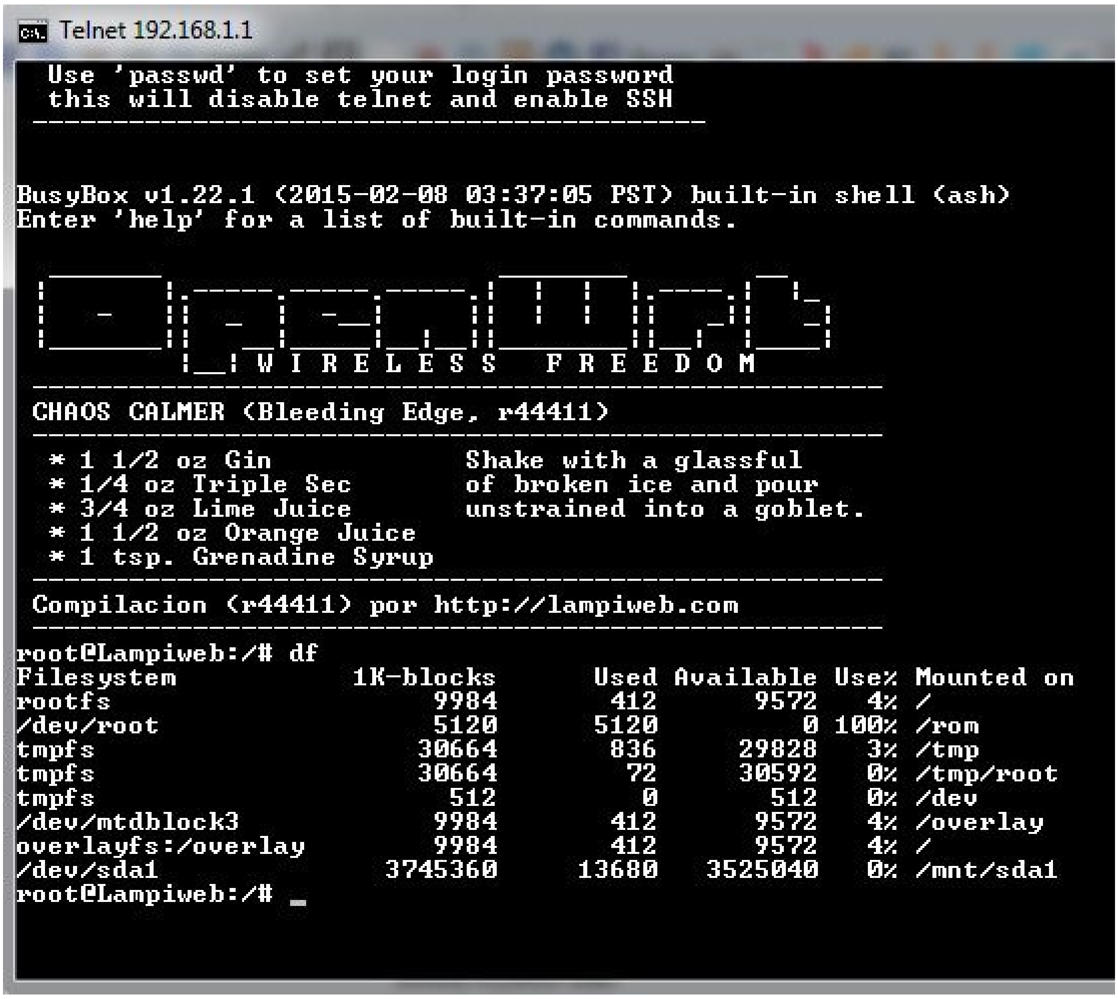
4.2 Protocols

For this thesis, protocols such as OpenFlow, RADIUS and 802.1x security are used extensively and are discussed in detail below, explaining their use cases and features.

4.2.1 OpenFlow [9]

It is a standard communication interface defined between the control and forwarding layers of the SDN architecture, allowing direct access for manipulating the forwarding plane of the network devices such as switches and routers, both physical and virtual (hypervisor based).

OpenFlow, along with SDN technologies have helped IT to manage and address the high-bandwidth, and dynamic nature of today's applications. It also has helped adapt the network to ever-changing business needs, and significantly reduce the complexity in maintenance and operations. The successful deployment [46] of OpenFlow in large organizations such as Google [47], Géant of Switzerland,ToulX France etc. shows what can be achieved with OpenFlow and the extent to which its features can be utilized.

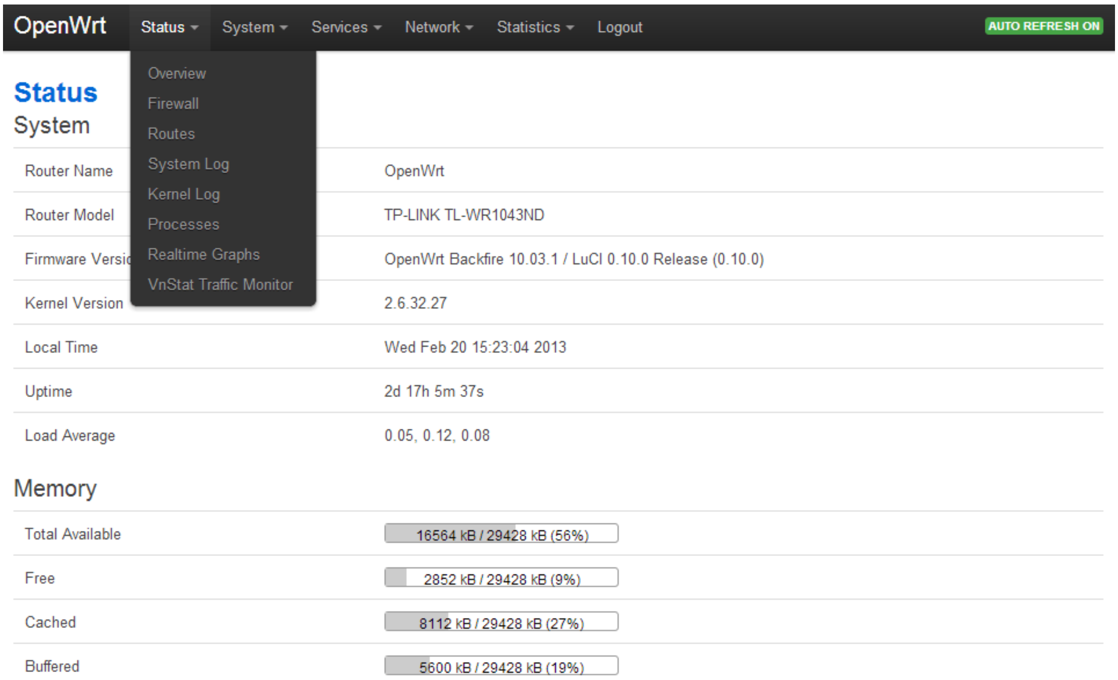


```
  Use 'passwd' to set your login password
  this will disable telnet and enable SSH
 ------------------------------------------


BusyBox v1.22.1 (2015-02-08 03:37:05 PST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 CHAOS CALMER (Bleeding Edge, r44411)
 -----------------------------------------------------
  * 1 1/2 oz Gin            Shake with a glassful
  * 1/4 oz Triple Sec       of broken ice and pour
  * 3/4 oz Lime Juice       unstrained into a goblet.
  * 1 1/2 oz Orange Juice
  * 1 tsp. Grenadine Syrup
 -----------------------------------------------------
 Compilacion (r44411) por http://lampiweb.com
 -----------------------------------------------------
root@Lampiweb:/# df
Filesystem           1K-blocks      Used Available Use% Mounted on
rootfs                    9984       412      9572   4% /
/dev/root                 5120      5120         0 100% /rom
tmpfs                    30664       836     29828   3% /tmp
tmpfs                    30664        72     30592   0% /tmp/root
tmpfs                      512         0       512   0% /dev
/dev/mtdblock3            9984       412      9572   4% /overlay
overlayfs:/overlay        9984       412      9572   4% /
/dev/sda1              3745360     13680   3525040   0% /mnt/sda1
root@Lampiweb:/# _
```

Figure 4.1: OpenWrt Terminal View [22]



Status

System

Router Name	OpenWrt
Router Model	TP-LINK TL-WR1043ND
Firmware Versi	OpenWrt Backfire 10.03.1 / LuCI 0.10.0 Release (0.10.0)
Kernel Version	2.6.32.27
Local Time	Wed Feb 20 15:23:04 2013
Uptime	2d 17h 5m 37s
Load Average	0.05, 0.12, 0.08

Memory

Total Available	16564 kB / 29428 kB (56%)
Free	2852 kB / 29428 kB (9%)
Cached	8112 kB / 29428 kB (27%)
Buffered	5600 kB / 29428 kB (19%)

Figure 4.2: OpenWrt GUI Interface [23]

How does OpenFlow Work? [48]

In a traditional switch or a router, the packet forwarding and high level routing decisions occur on the same device. In an OpenFlow switch, the routing and forwarding functions are separated. The data path portion is still on the switch and the routing decisions are handled by a separate controller, typically it's a standard server. The OpenFlow switch and controller communicate using the OpenFlow protocol, which defines messages such as packet-in, packet-out, modify-forwarding path and get stats.

OpenFlow Specification [49]

The protocol can be split into 4 components as shown in the figure 4.3 namely: message layer, state machine, system interface and configuration.

- **Message Layer:**
  It is the core of the protocol stack. It also supports the ability to construct, copy, compare, manipulate and print the messages. The message layer defines the valid structure and semantics for all messages.

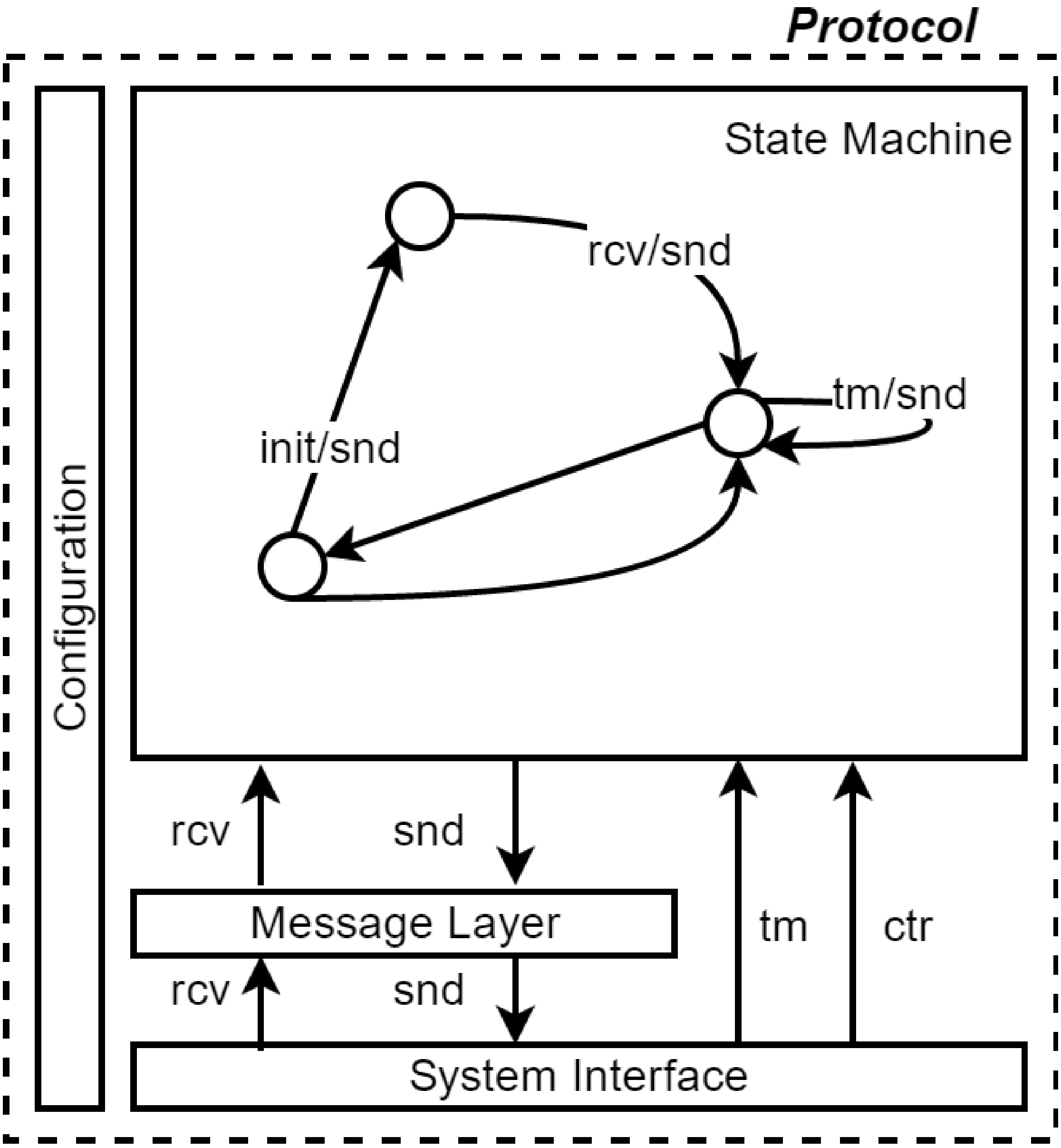


Figure 4.3: OpenFlow Protocol [25]

- **State Machine:**
  It defines the core level behaviour of the protocol. It is typically used to describe the actions such as: flow control, negotiations, delivery, capability discovery etc.
- **System Interface:**
  It typically defines how the protocol interacts with the other protocols in the outside world. The system interface identifies the necessary and optional interfaces along with its intended use such as TLS and TCP as transport channels.
- **Configuration:**
  Almost every protocol has its own configuration or initial values. It can cover anything from buffer size, reply intervals to X.509 certificates.
- **Data Model:**
  Each switch maintains the attributes of each OpenFlow abstration in a relational data model. The attributes either describe its configuration state, or some set of current statistics or the abstraction capability.

OpenFlow Switch [50]

An OpenFlow switch is made up of two components namely the switch agent and the data plane. The switch agent takes care of the communication between two or more controllers and also with the data plane using the requisite internal protocol. The switch agent translates the commands into low-level instructions to send to the data plane and the data plane information is translated to OpenFlow messages that are forwarded to the controller. The data plane takes care of the packet manipulation and forwarding and sometimes sends packets to the switch agent for further handling based on its configuration. The figure 4.4 shows the functionality of the switch as explained above.

OpenFlow Switch Agent [50]

The figure 4.5 shows how the switch agent works, its components are explained as follows.

- **OpenFlow protocol:** This instance is on the switch side
- **Core Logic:** Switch management, command execution to the data plane and manage the data plane offload etc.

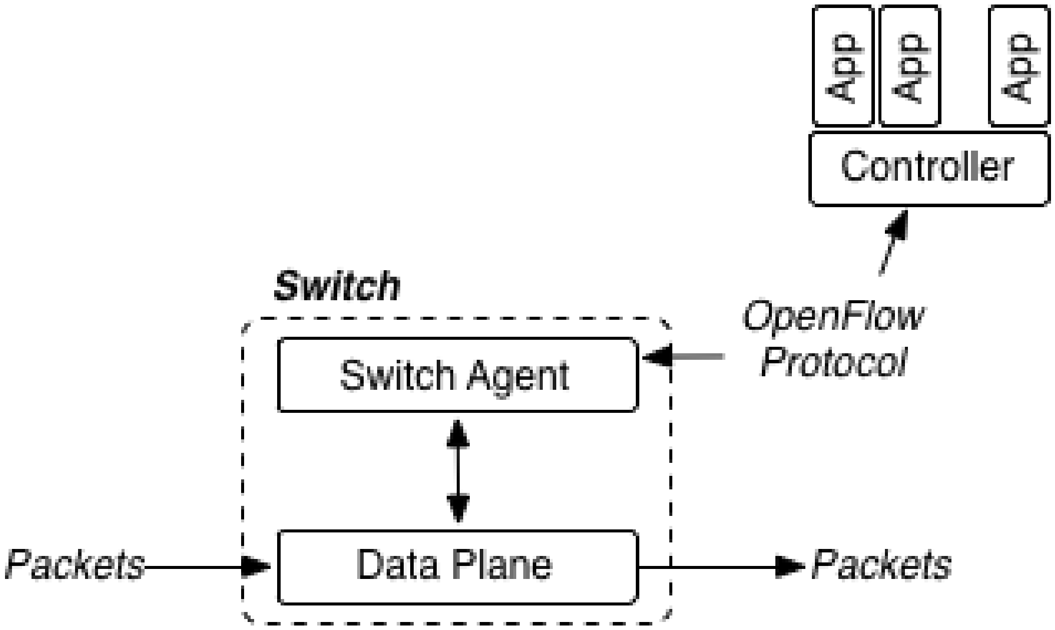


Figure 4.4: OpenFlow Switch Anatomy [26]

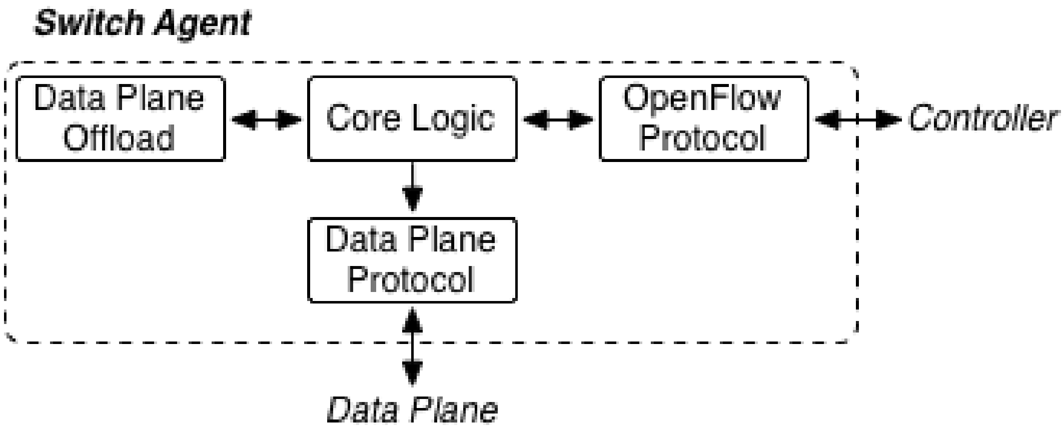


Figure 4.5: OpenFlow Switch Agent [27]

- **Data Plane Offload:** Some functionality present in the OpenFlow will be offloaded by the control plane which is not provided in the existing data plane implementation.
- **Data Plane protocol:** This protocol is internal which is mostly used for configuring the data plane state.

Data Plane [50]

The data plane consists of the ports, flow tables, flows, classifiers and actions. Packets traverse through the system on ports. When each packet arrives, it is matched with the flows in the flow table using classifiers. The flows contain the set of actions that are applied to each packet that matches.

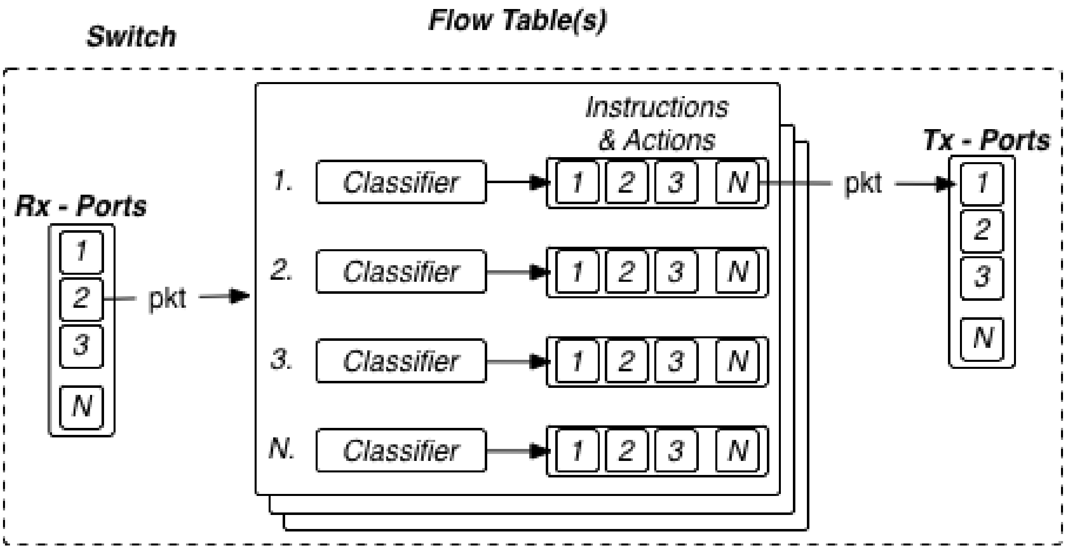


Figure 4.6: OpenFlow Data Plane Schematic [28]

Data Plane - Packet Lifecycle [50]

Each packet is processed in the following sequence as explained below.

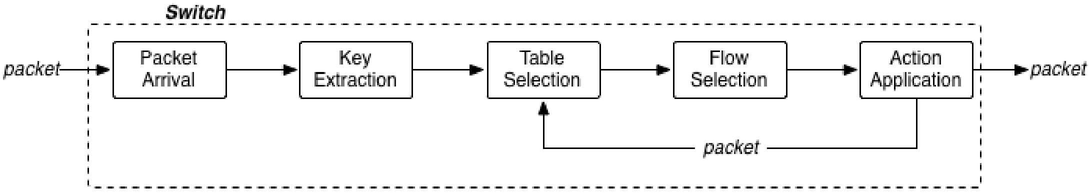


Figure 4.7: Packet Lifecycle [29]

- **Step 1: Packet Arrival**
  Packets arrive in either a physical or virtual port, it is necessary to make note of the arrival port for source-based processing later.
- **Step 2: Key Extraction**
  When each packet arrive on the port, a small meta data is built called the key. This key contains information about the packet such as header values, buffered packet, arrival port, arrival time etc.
- **Step 3: Table Selection**
  When a packet goes through the pipeline, the packet is matched with the first table by default and if multiple tables exist then subsequent tables will be selected through hit or miss actions.
- **Step 4: Flow Selection**
  The Key extracted in the initial step is used for selecting the flow from the table. The first flow where the classifier subsumes the key become the selected flow.
- **Step 5: Application Selection**
  Each flow contains a set of actions, which is applied to the packet when a flow is matched. The actions can modify the state of the packet or change how the packet is treated.

4.2.2 RADIUS [10]

RADIUS stands for Remote Authentication Dial-In User Service, which is an access control server for authentication and accounting protocol. RADIUS is an AAA protocol used for network access applications.

What is AAA Protocol? [51]

AAA stands for Authentication, Authorization and Accounting. *Authentication* is the validation of the user requesting access to a service.It is normally done by providing some credentials such as username and password. *Authorization* provides specific services based on the user's authentication such as physical location restrictions, multiple login access restrictions etc. *Accounting* keeps track of all the users and their network resource consumption is provided by the accounting service, it's like a log for every user who gained access to the network. Typical information includes user identity, nature of service delivered etc. This information may probably be used for billing, management purposes.

Key Features of RADIUS: [10]

The RADIUS works like a *Client / Server* model. The network server acts as the client of RADIUS, which passes the user information to designated RADIUS server. The responsibilities of the radius server include receiving connection requests, authenticating users, providing all the configuration details necessary for the client to deliver service to the user.

RADIUS also provides security by encrypting the communication between the client and the RADIUS server. So, any users credentials sent over the network is encrypted. In addition, the client and the RADIUS server transactions are authenticated over a shared secret which is never sent over the network. The RADIUS server supports several method's for a user to authenticate such as PPP, PAP or CHAP etc.

The RADIUS is also an extensible protocol where all transactions are of variable length Attribute-value-length 3 tuples. Supports addition of new attribute values without disturbing the existing implementation of the protocol.

RADIUS Components [52]

The following components are part of the RADIUS infrastructure.

- Access Clients:
  These are the devices such as a mobile phone, laptop, desktop computer etc. that is requesting to gain access to the network
- Access Servers (RADIUS clients):
  They are network access servers such as access points, 802.1x capable switches etc. These devices communicate with the network access servers using the RADIUS protocol. They are the devices to which the clients are associated to for access to the network.

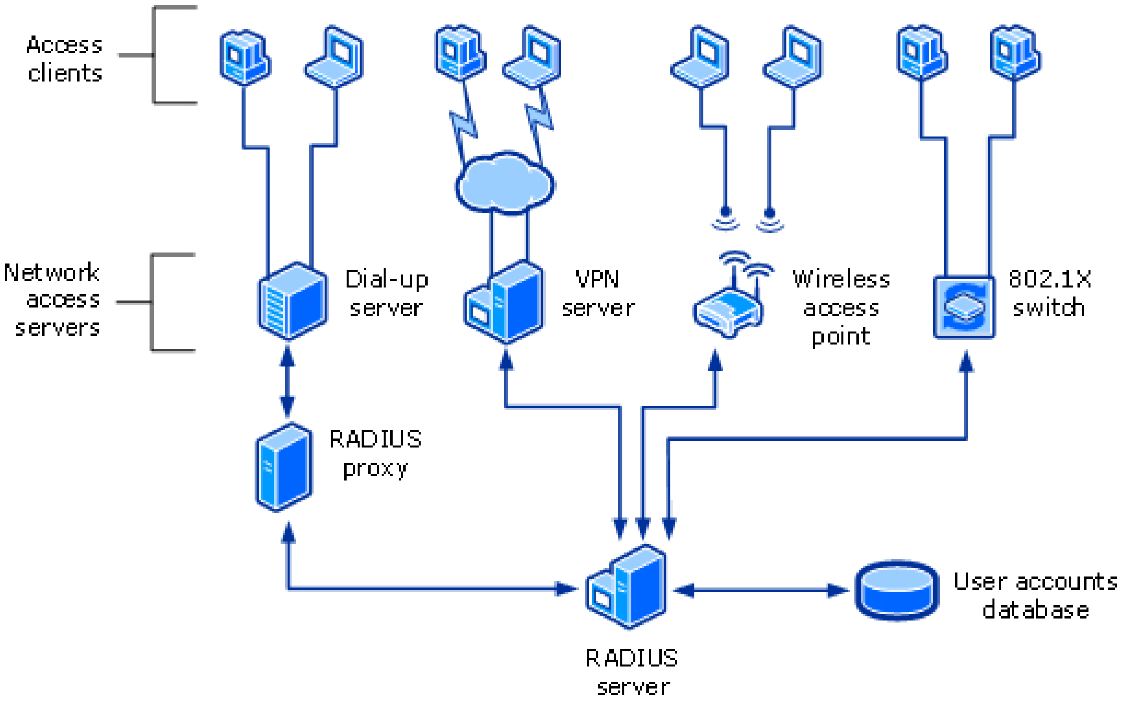


Figure 4.8: RADIUS Components [30]

- RADIUS servers:
  It is the network access server that manages the users and the services associated to each user or client.
- RADIUS proxies
  They are similar to the network access server except that, they do not process the AAA information received from the RADIUS clients. They simply forward the information to the RADIUS server or to another RADIUS proxy based on the information in the packet.
- User account databases (Active Directory, any database such as MySQL):
  It is the database to which the RADIUS server refers to (When configured to work with database) for authenticating an user requesting network access. The database also contains other accounting information such as the session details for each user, the services active for each client.

The components are showing in the figure 4.8.

RADIUS Operation [52]

RADIUS messages are sent as UDP messages using the port 1812 for authentication and port 1813 for accounting messages. Some network access servers (NAS) use 1645 and 1646 for authentication and accounting respectively.

A RADIUS data format looks as shown below where the fields are transmitted from left to right.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                         Authenticator                         |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Attributes ...                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The code field as shown in the data frame above uses one octet which help identify the type of RADIUS packet.

- **Access-Request:** Sent by the RADIUS client requesting authentication and authorization for a connection.
- **Access-Accept:** It's the response from the RADIUS server to the client stating that the connection was authenticated and authorized.
- **Access-Reject:** It's a response from the RADIUS server to the client that the connection attempt failed in authentication.
- **Access-Challenge:** Sometimes the RADIUS server requires more information from the client and sends a challenge as a response the Access-Request message.
- **Accounting-Request:** Sent by the RADIUS client to specify accounting information for an accepted connection.
- **Accounting-Response:** The RADIUS server sends the acknowledgement for the successful receipt and processing of the Accounting-Request message.

RADIUS Authentication Mechanism

RADIUS uses the following message codes when communicating between the RADIUS client and server as shown in the figure below.

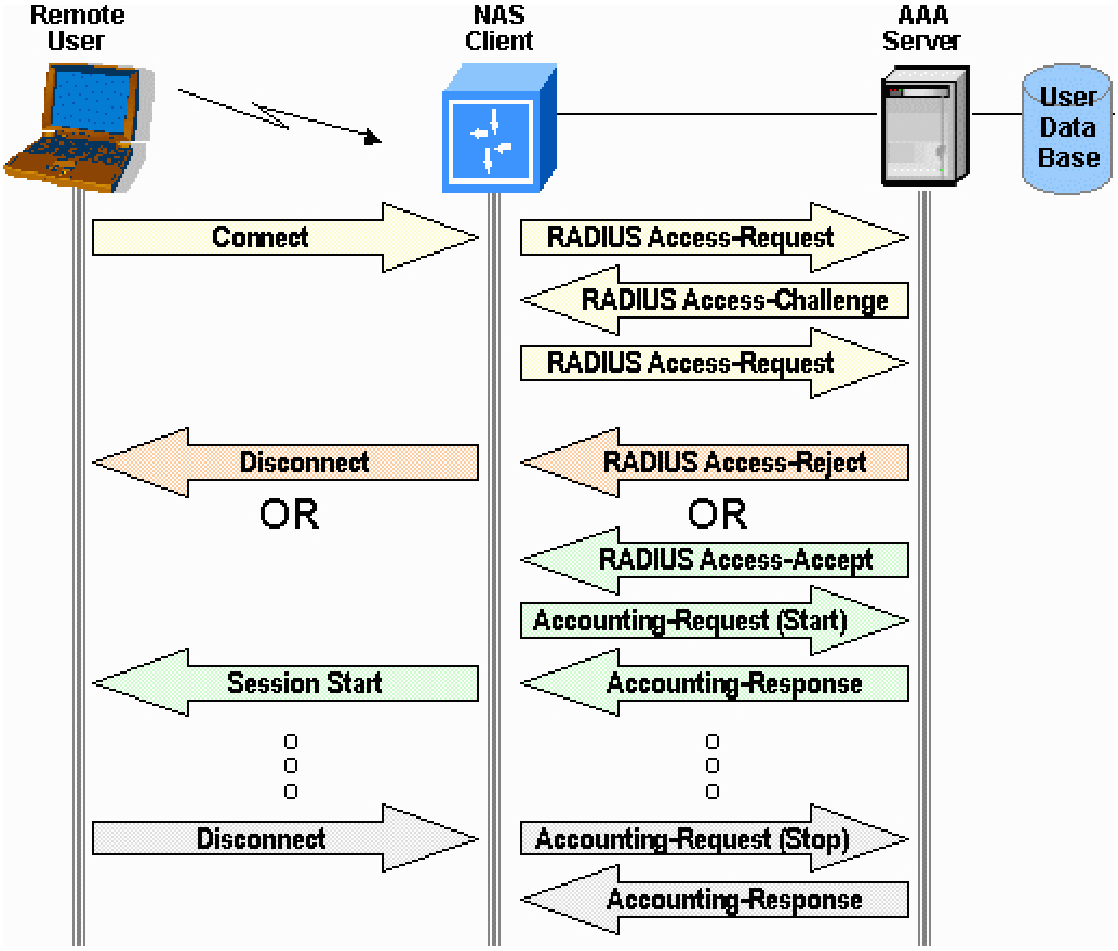


Figure 4.9: RADIUS Architecture [31]

- **STEP 1:**When a user makes a connection request, the NAS client sends an Access-Request message to the AAA server, in this case a RADIUS server.
- **STEP 2:**The RADIUS sever responds by sending an Access-Challenge message requesting more information from the user.
- **STEP 3:**The client responds with an Access-Request message with the requested information back to the RADIUS server. The response is typically a username and password information in the form of PPP, PAP or CHAP authentication mechanisms.

- **STEP 4:**The RADIUS server once after validating the received information sends back an Access-Accept information. If not validated, then it sends a Access-Reject message back to the client.
- **STEP 5:**Upon successfully establishing a connection, the client sends an Accounting-Request message to request to start accounting the user.
- **STEP 6:**The RADIUS server responds by sending an Accounting-Response message after successfully starting an accounting session for the connection. Thus, concludes the connection process of the user to the network.

4.2.3 WLAN 802.1x Security [11]

Wi-Fi or Wireless Local Area Networks (WLAN's) have become increasingly more popular in the recent years. The wireless standard IEEE 802.11 has become the most widely adopted standards for wireless broadband internet access. The security considerations however are more complicated in the wireless environment compared to the wired ones. IEEE 802.11 has defined the following two basic security mechanisms for secure access to wireless network.

- Entity authentication including shared key and open-system.
- Wired Equivalent Privacy (WEP)

Both these mechanisms are proven to be severely vulnerable. To enhance the security in wireless networks, 802.11i standard was proposed. This 802.11i standard defines encryption and authentication improvements in addition to introducing protocols for key management and establishment. 802.11i also incorporates the IEEE 802.1x standard as its authentication enhancement. The IEEE 802.1x is a port based network access control used for authenticating and authorizing devices connected by various LAN's.

The IEEE 802.1x standard is based upon the Extensible Authentication Protocol (EAP), and can use a number of authentication mechanisms which is beyond the scope of the IEEE 802.1x standard. Many authentication mechanisms such as EAP/MD5(port based) , TLS, TTLS, and PEAP can be used. The IEEE 802.1x uses EAP over LAN (EAPoL) for encapsulating EAP messages between the authenticator and the supplicant.

There are three main components in the IEEE 802.1x system namely, the supplicant, authenticator and the authentication server. In case of WLAN, the supplicant is usually the mobile device or node, the Access Point (AP) serves as the authenticator and the

RADIUS server as the authentication server. The Port Access Entity (PAE) authenticator relays all the messages between the authentication server and the supplicant. 802.1x is used in this place to enforce the specific authentication mechanism.

802.1x Authentication Process

Authentication methods of 802.1x include PEAP, MD5 etc. Each method has its own authentication process. The following figure shows the basic EAP based authentication process in Eduroam networks.

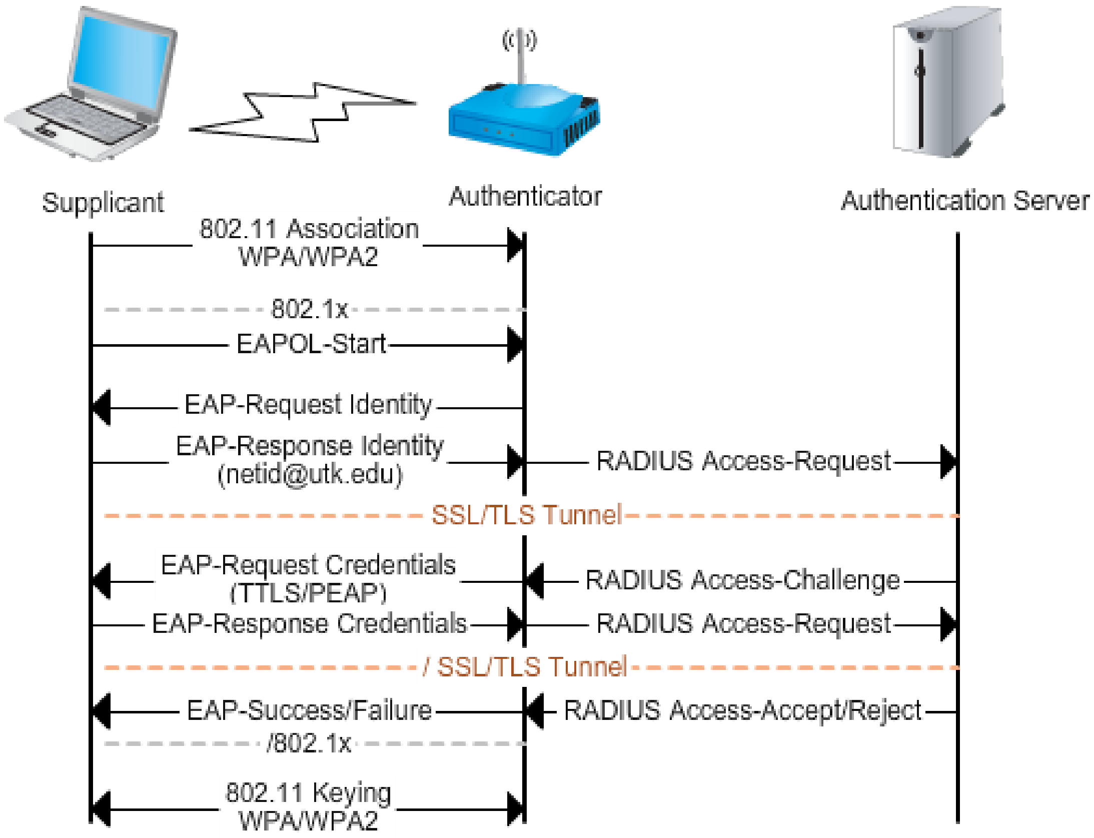


Figure 4.10: IEEE 802.1x WLAN authentication process [32]

- **Step 1:**
  After the association of the supplicant with the authenticator or AP via WPA or WPA2 enterprise, the 802.1x application initiates the EAPOL start message with the authenticator.

- **Step 2:**
  The authenticator responds by send a EAP-Request identity message from the supplicant.

- **Step 3:**
  Once receiving the username as a EAP response from the supplicant, the authenticator then initiates a RADIUS Access-request message with the authentication server.

- **Step 4:**
  The authenticator receives the RADIUS access challenge from the authentication server and forwards that via a secure SSL/TLS tunnel to the supplicant by encapsulating the EAP-Request credentials message with TTLS/PEAP.

- **Step 5:**
  The supplicant responds with a EAP-Response Credentials message which is typically a username and password to the authenticator via the same secure tunnel, the authenticator forwards the received information as a RADIUS Access-Request message with encapsulation to the authentication server.

- **Step 6:**
  The authentication server responds with a RADIUS Access-Accept/Reject message to the authenticator. The Authenticator sends this information as a EAP Success/Failure message to the supplicant.

- **Step 7:**
  The supplicant is now fully associated with the network.

5 Build Environment

This chapter discusses on how the development environment is set up starting with the RYU controller from Git repository and OpenWrt to build firmware for the TP-Link WDR4300 test router.

5.1 RYU in Python virtual environment [12]

Python by default stores all its packages in a global location accessible from any were within the system. Though this may sound advantageous, like many other programming language, uses its own way to store, download and retrieve its packages. Python uses same site packages directory to install 3rd-party packages and different versions of python also reside in the same location.

Python couldn't differentiate between different versions and thus creates dependency issues. To resolve this problem, a virtual environment is used for setting up an isolated location for Python projects. In a virtual environment, each project can have its own dependencies. There is also no limit to the number of environments that can be created since they are just directories containing scripts.

5.1.1 Installation and Access [12] [13]

To install the virtual environment in Linux, the following python package manager (PIP) commands are used in the terminal.

1. Installing the virtual environment :

```
    $ pip install virtualenv

```

2. Create a directory in virtualenv for python packages:

```
    $ virtualenv ryu-virtualenv

```

3. In the newly created environment, there is an activate shell script to change the *path* to the /bin directory in the virtualenv:

```
    $ source bin/activate

```

4. To install RYU in this virtual environment:

```
    $ pip install ryu

```

5. Building RYU applications requires the RYU repo from Git which can be downloaded from the command:

```
    $ git clone  git://github.com/osrg/ryu.git

```

6. Once the installation is complete and the repo downloaded, *ryu-manager* command is used to run the RYU Python applications.

5.2 OpenWrt Build System [14]

OpenWrt supports building custom firmware to any supported hardware. For this thesis, TP-Link WDR4300 is chosen because of its compatibility with OpenWrt, a larger RAM and ROM for adding more packages and functionality, and suppport for multiple SSIDs which is necessary for this thesis in order to simulate two different network.

5.2.1 Hardware Prerequisites

The following requisites must be met to generate an installable firmware on a supported hardware.

- At least 3-4 Gb of hard disk space for OpenWrt build system, source packages and its feeds, and to generate firmware files.
- At least 3-4 GB of RAM to build OpenWrt.

5.2.2 Installation steps on GNU/Linux

1. Install Git to conveniently manage and download repository such as OpenWrt and RYU, build tools for cross compilation process:

```
    $ sudo apt-get install update
    $ sudo apt-get install git-core build-essential libssl-dev
   libncurses5-dev unzip gawk zlib1g-dev

```

2. Clone the Git repository on local machine:

```
    $ git clone https://github.com/openwrt/openwrt.git

```

3. Install available all available feeds for OpenWrt:

```
    $ cd openwrt
    $ ./scripts/feeds update -a
    $ ./scripts/feeds install -a

```

4. To check for any missing packages the following command is used for a GUI applicaton popup in terminal:

```
    $ make menuconfig

```

5. The chapter on implementation discusses in detail on building custom OpenWrt firmware with custom packages such as OpenvSwitch.

6 Designing the Application

This chapter delves into the design objectives of the application, the architectural frame work of the RYU controller and converting the design into a Python code.

6.1 The Design Objectives

The finished application should be compatible with the RYU controller and be able to handle packets in real time. The timing diagram6.1 shows how the RADIUS control flow should happen in the application for authentication.

6.1.1 RADIUS Procedure

1. The mobile client first initiates a radius authentication request with the Access Point.
2. The packet is verified and authenticated by the RADIUS request and response messages.
3. If the authentication is a success, the connection is setup with the access point with WPA2 enterprise keying.
4. Once the client is authenticated, it makes a DHCP request with the access point.

The flowchart 6.4 shows the authentication procedure of the RADIUS server. It also shows the step by step actions that take place in authenticating a client.

6.1.2 RYU Control Procedure

The timing diagram of RYU 6.3 shows how the RYU is supposed to listen to the MAC address of the client and parse the packet to assing the destination port id for the client.

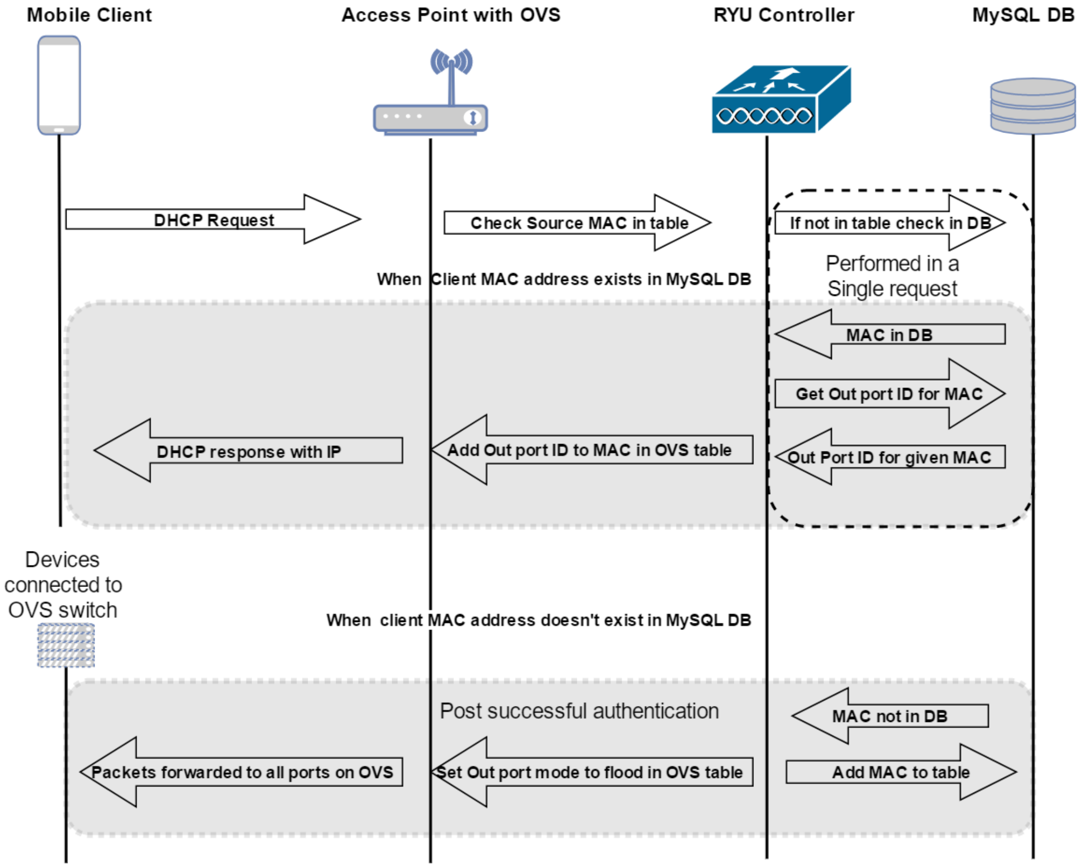

Mobile Client
Access Point with OVS
RYU Controller
MySQL DB
DHCP Request
Check Source MAC in table
If not in table check in DB
Performed in a Single request
When Client MAC address exists in MySQL DB
MAC in DB
Get Out port ID for MAC
DHCP response with IP
Add Out port ID to MAC in OVS table
Out Port ID for given MAC
Devices connected to OVS switch
When client MAC address doesn't exist in MySQL DB
Post successful authentication
MAC not in DB
Packets forwarded to all ports on OVS
Set Out port mode to flood in OVS table
Add MAC to table

Figure 6.1: RADIUS Authentication Procedure - Timing Diagram

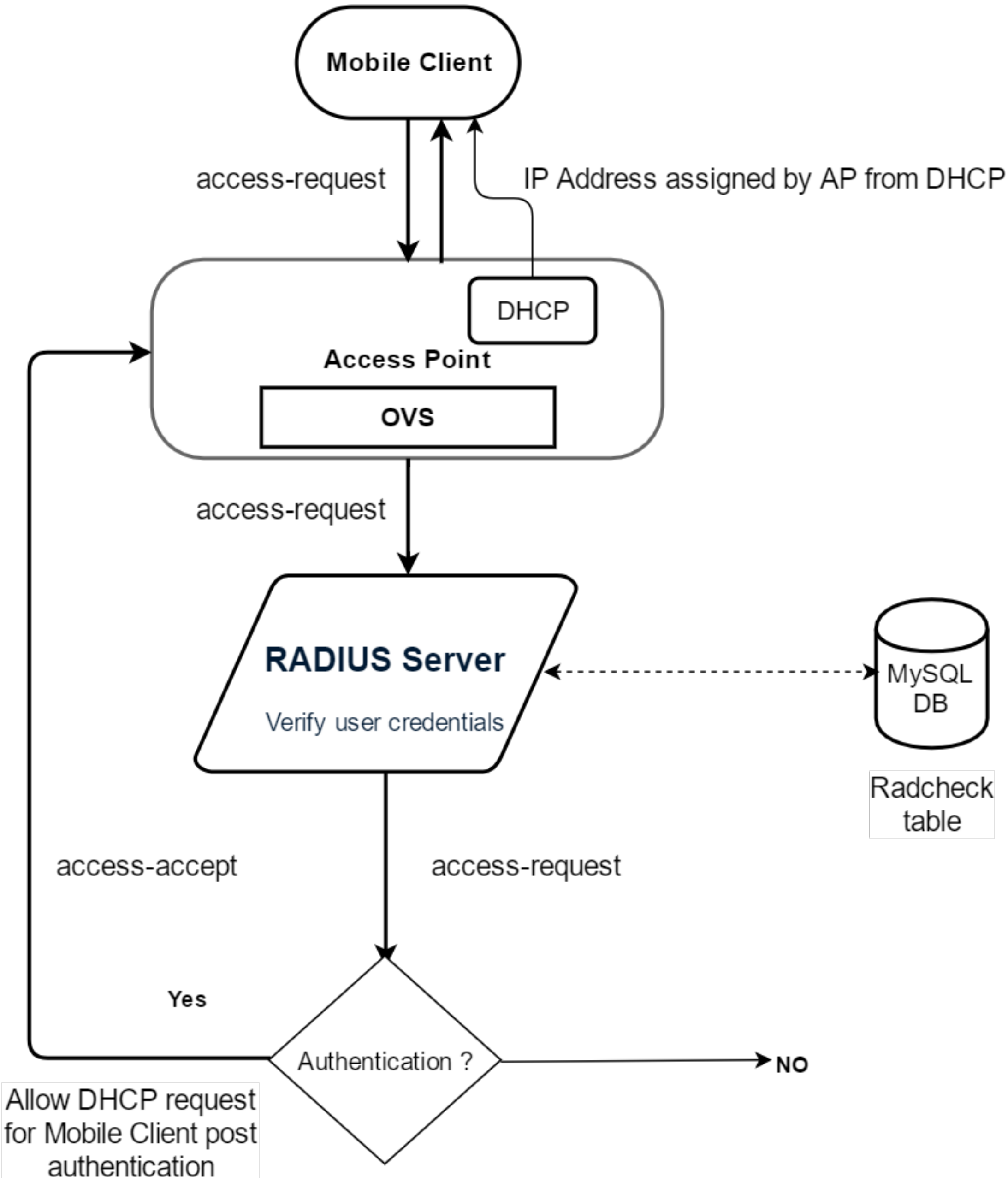


Figure 6.2: RADIUS Authentication Procedure - Flowchart

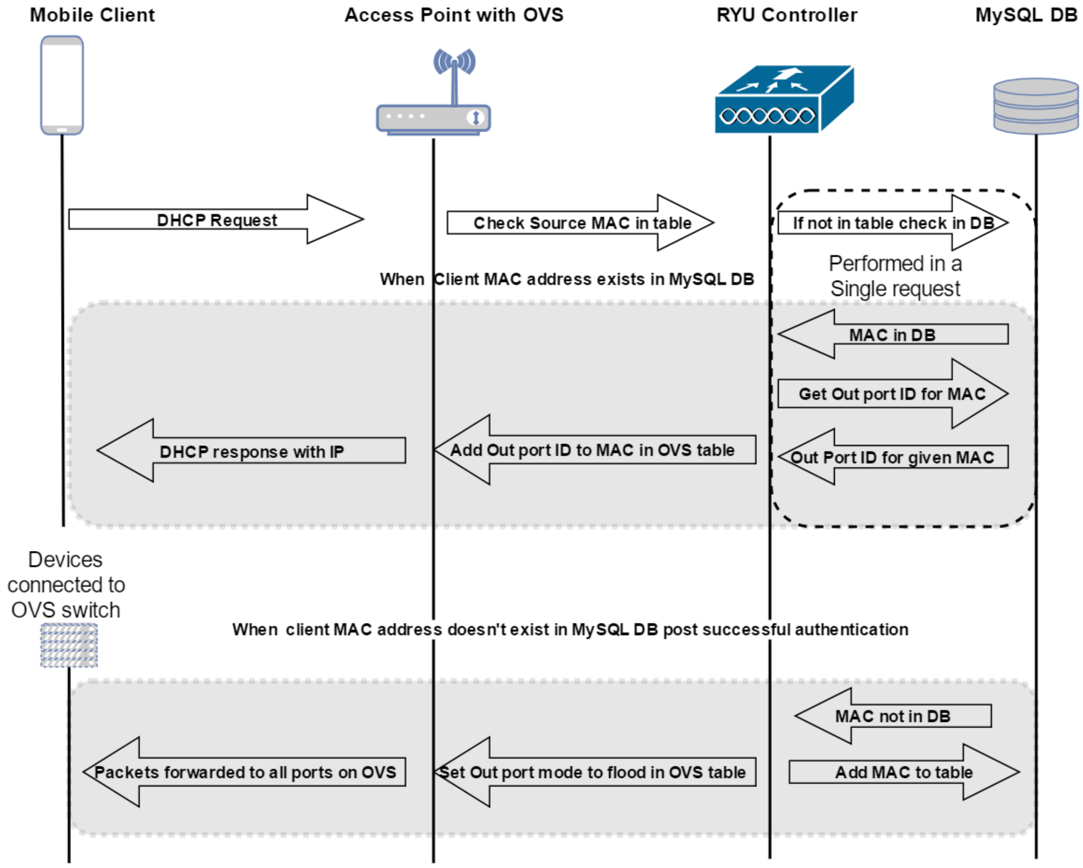

Mobile Client
Access Point with OVS
RYU Controller
MySQL DB
DHCP Request
Check Source MAC in table
If not in table check in DB
Performed in a Single request
When Client MAC address exists in MySQL DB
MAC in DB
Get Out port ID for MAC
DHCP response with IP
Add Out port ID to MAC in OVS table
Out Port ID for given MAC
Devices connected to OVS switch
When client MAC address doesn't exist in MySQL DB post successful authentication
MAC not in DB
Packets forwarded to all ports on OVS
Set Out port mode to flood in OVS table
Add MAC to table

Figure 6.3: RYU Control Procedure - Timing Diagram

When the client initiates a DHCP request to the access point, the RYU controller listens to the clients MAC address and checks if it exists in the OVS flow table. If it exists, then apply the corresponding rule associated for the MAC address. If it doesn't exist, then check for the MAC address in the MySQL database. In the first scenario, if the MAC address exists in the MySQL DB, the out port ID associated with the MAC address is retrieved. The retrieved out port id is assigned to the MAC address and an entry is created in the OVS flow table and completes the process with a DHCP response.

In the second scenario after the authentication of the client is successful, if the MAC address does not exist in MySQL DB. The MAC address is added to the table and the out port id is set to flood mode in the OVS flow table in the access point. The process is completed with a DHCP response by providing an IP address to the client. In case of an attack, the system becomes vulnerable only when the fake MAC address used by the client matches with the user credential used for authentication. In such cases a firewall is used to further protect the system.

The flowchart 6.4 will provide an overview of the control flow that would take place in the RYU controller and the OVS. Starting from learning the MAC address to assigning the out port id to the MAC.

From the flowchart 6.4, it is clear that the initial DHCP request made by the mobile client to the AP is flooded to all ports in the AP. The RYU controller captures this request and retrieves the MAC address of the client. The packet passes through two conditions, the first condition is to check whether the client MAC address is already in the table. If the MAC address exists, then the packet passes to the second condition. The second condition is where the MAC address is checked if it is associated to a corresponding out port ID. In case, the out port ID is found. Then the packet is converted to a unicast with the assigned out port through which all the packets coming from the client will pass through to its destination in future. If the packet fails in the first condition, then the packet is dropped else if it fails in the second condition. The packet is flooded to all the ports in the OVS and the MAC address is learnt to avoid flooding in future.

Due to the current OpenFlow version (v1.3) used by the OVS, it is limited to assigning only one out port per user instead of multiple ports. This feature will be available in future versions of OpenFlow when supported by OVS.

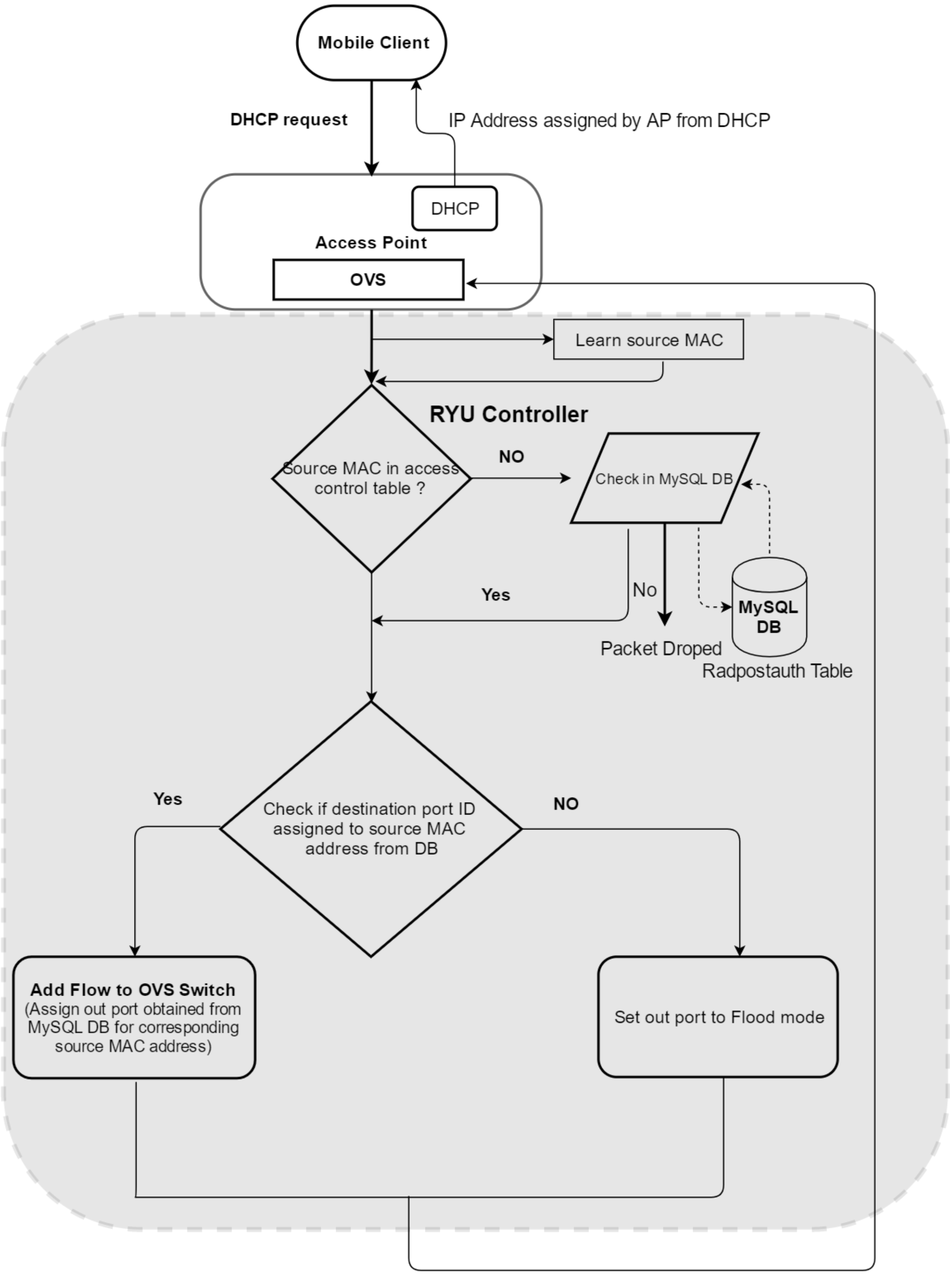

Mobile Client
DHCP request
IP Address assigned by AP from DHCP
DHCP
Access Point
OVS
Learn source MAC
RYU Controller
Source MAC in access control table ?
NO
Check in MySQL DB
Yes
No
MySQL DB
Packet Droped
Radpostauth Table
Yes
Check if destination port ID assigned to source MAC address from DB
NO
Add Flow to OVS Switch (Assign out port obtained from MySQL DB for corresponding source MAC address)
Set out port to Flood mode

Figure 6.4: RYU Control Procedure - Flowchart

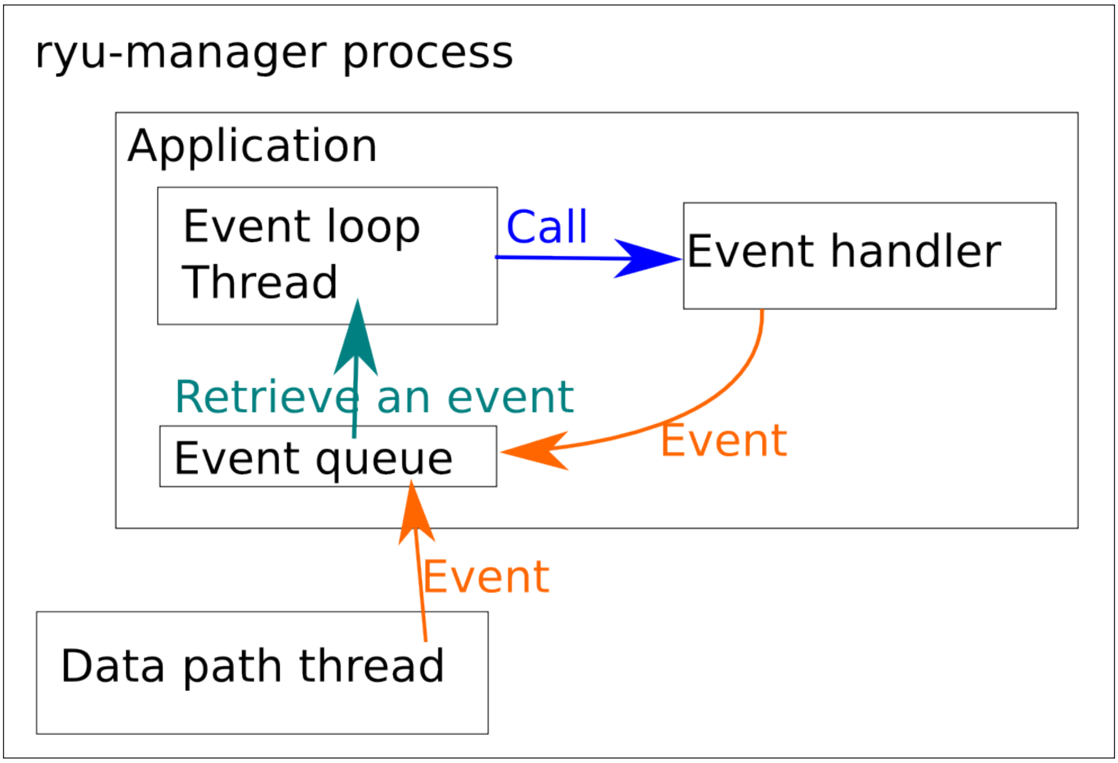


Figure 6.5: RYU Manager Event Process [33]

6.2 RYU Manager Process [15]

For designing the application, the RYU manager process is considered. This is explained using the diagram 6.5. To build the application in python, the following classes are mainly used which are explained below.

- Application is the user logic that explains how the application should behave. It is a class that inherits the *ryu.base.app_manager.RyuApp*.
- Events are class objects that are used in communication between applications. It inherits the class *ryu.controller.event.EventBase*.
- Event queues are the single queues that each application has for receiving events.
- RYU uses eventlets to run in multi-threaded environment. These threads are non-preemptive.
- Event Loops: A thread that is created for each application runs an event loop. When there is an event in the queue, this loop will load the event and call the corresponding handler.

- Event Handlers: They are user defined handlers designed to handle when a specific type of event occurs. They reside in the event loop of an application. Event handlers can be defined by decorating the application class method with the *ryu.controller.handler.set_ev_cls* decorator.

6.3 Python Coding

The code is built using an existing RYU MAC learning application and is modified for user segregation.

The *set_ev_class* sets the event handler for packet-in event and passing it on to the method *_packet_in_handler* to parse the incoming packet. The packet is parsed by the method and details about the packet are extracted such as the in-port, datapath, payload etc.

```
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # If you hit this you might want to increase
        # the "miss_send_length" of your switch
        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes", ev.msg.msg_len, ev.msg.total_len)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        udp_payload = pkt.get_protocols(udp.udp)
```

Once the source and destination MAC address is retrieved from the packet, the source MAC address is then checked in the MySQL DB as shown in the figure above. The statement *cursor.execute* also retrieves the port id from the database for the corresponding MAC if it is found as shown in the code snippet below.

```
    #creating a mysql connection to database

    connection = MySQLdb.connect(host = "192.168.1.169", user = "freerad", passwd = "pass", db = "radius")
    cursor = connection.cursor ()

    #cursor.execute ("select CallingStationId from radpostauth order by id desc LIMIT 1")
    #data = cursor.fetchall ()
    cursor.execute ("SELECT portid FROM radcheck WHERE username IN (SELECT user FROM radpostauth WHERE CallingStationId = %s AND id = (SELECT MAX(id) from radpostauth) )", src)
    outport_for_src = cursor.fetchone ()
    self.logger.info ("outport_for_src tuple is %s", outport_for_src)
    #portid = outport_for_src[0]
    self.logger.info ("Data is %s", src)
    cursor.close()
    connection.close ()
    # Mysql verification end
```

In this step, the incoming port id (*in_port*) is stored in the *self.mac_to_port* array. The first *if* condition then checks if the destination MAC address is in the array *self.mac_to_port*, if it exists then the second condition checks if the out port retrieved form the database is not empty or null. Then the third condition checks if the retrieved out port id from the database is the same as the one retrieved from the packet coming from the client, then the port id in the array *self.mac_to_port* is assigned to the *out_port* variable.

```
        self.mac_to_port[dpid][src] = in_port

    self.logger.info ("DST is %s", dst)
        if dst in self.mac_to_port[dpid]:
        test = self.mac_to_port[dpid][dst]
        self.logger.info ("self.mac_to_port[dpid][dst] value %s", test)
        self.logger.info ("Out_Port before condition %s", outport_for_src)
        if outport_for_src != None and all(outport_for_src):
            if int(outport_for_src[0]) == self.mac_to_port[dpid][dst]:
                out_port = self.mac_to_port[dpid][dst]
                self.logger.info ("Out_Port is the same as in database table %s", out_port)
            else:
                self.logger.info ("Out_Port is not allowed, dropping packet")
                return
        else:
            #except (TypeError,UnboundLocalError):
            out_port = self.mac_to_port[dpid][dst]
            self.logger.info ("Out_Port not defined in database, using learned port")
```

The flowing code snipped explains the scenario when there is no out-port id is mentioned in the database for the corresponding MAC address.

```
    else:
```

```
    #except (TypeError,UnboundLocalError):

        out_port = ofproto.OFPP_FLOOD
    #out_port = 4
        self.logger.info ("Out_Port is Flooded %s", out_port)

self.logger.info ("Above actions Out_Port %s", out_port)
    actions = [parser.OFPActionOutput(out_port)]
self.logger.info ("Actions is %s", actions)
```

The variable *out_port* is assigned a FLOOD mode and the parser *parser.OFPActionOutput* is called to parse the *out_port* mode set in the previous conditions.

A new flow is then assigned to the OVS switch with the obtained *out_port* conditions. The code snipped below shows the usual procedure to add the flow by using *add_flow* method providing values such as datapath, match conditions, and actions like setting the out port and buffer id if it exists. The packet is updated with the new information and sent out using the *send_msg* method that sends it to the OVS switch and a new flow entry is created in the OVS flow table.

```
         # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
        self.logger.info ("Out_Port not flooded adding flow ")
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
            #match = parser.OFPMatch(in_port=in_port, eth_dst='a0:f3:
c1:77:d8:36')
            # verify if we have a valid buffer_id, if yes avoid to
send both
            # flow_mod & packet_out
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, match, actions, msg.
buffer_id)
                return
            else:
                self.add_flow(datapath, 1, match, actions)
        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data
        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.
buffer_id, in_port=in_port, actions=actions, data=data)
        datapath.send_msg(out)
```

7 Implementation

This chapter will discuss about the procedures involved such as flashing the router with OpenWrt, building the OpenWrt firmware with OVS modules, configuring OVS, setting up RADIUS server in a virtual machine and creating a MySQL database with custom table for access to out port id etc.

7.1 Building and Flashing Custom OpenWrt Firmware

To build the custom firmware for the OpenWrt, we use the *make menuconfig* command in terminal from the directory where the OpenWrt repository has been cloned. The following steps will explain which modules to choose from the Menu and to compile the custom build.

1. Navigate to the directory <*buildroot_dir*> in terminal and then enter the command *make menuconfig*.
2. In the menu that appears, select the target system by using the arrows and enter key to navigate the menu.
3. In the target system, choose **Atheros AR7xxx/AR9xxx**.
4. Now, select target profile and choose **TP-Link WDR4300** from the list.
5. In the main configuration menu, select the **Luci** menu and enable luci web interface.
6. Now, going back to the main menu, choose the Network option in the list.
7. Within the Network menu, select the sub menu **Open Vswitch** as shown in figure7.1 and enable all options using space key <*>.
8. In the Network menu, first de-select the option **Wpad mini** and then select **Hostapd** as shown in figure7.2 with full features. This provides the necessary enterprise 802.1x authentication features.

9. In the main configuration menu, navigate to **Utilities** option and select editors and choose either **vi** or **nano** as the text editor of choice.

10. Exit the menu and select save to save the configuration.

11. Back in the terminal, enter the command *make world*. This will compile the changes made in the configuration and build the firmware for the selected hardware profile, in this case TP-Link WDR4300.

12. The freshly built images are available in the root directory of OpenWrt *<build-root_dir>/bin/ar71xxx*.

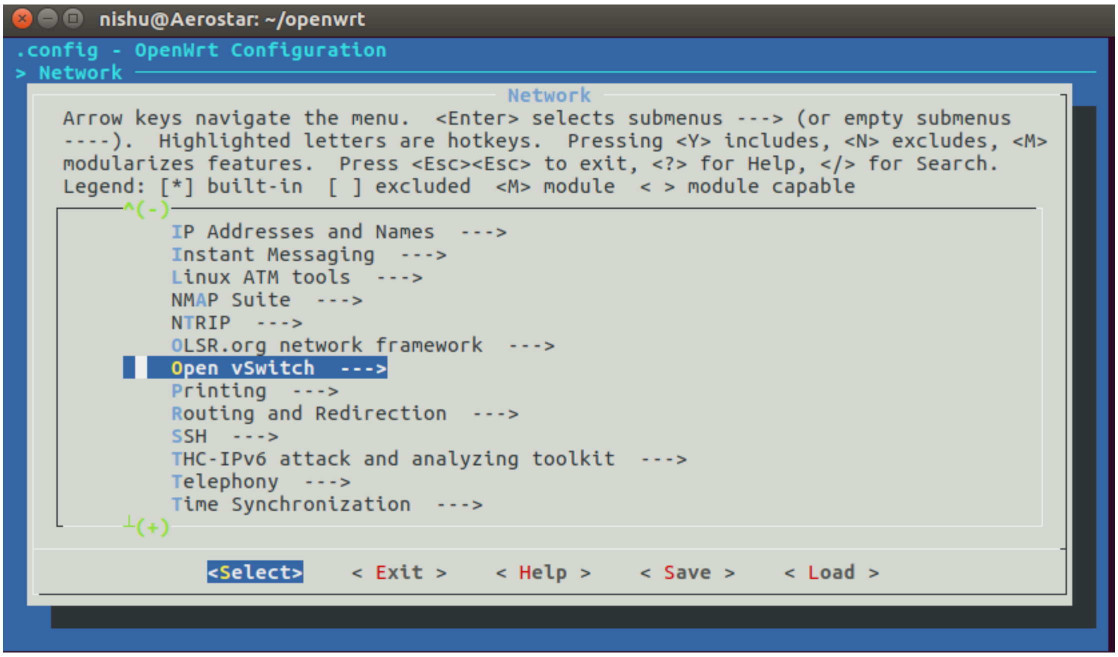
13. Connect the router to the computer via ethernet and login to the OEM interface using the ip 192.168.0.1 with user/pass as admin/admin.

14. Select the option firmware upgrade in the device settings.

15. In the OpenWrt bin directory, select the image *openwrt-ar71xx-generic-tl-wdr4300-v1-squashfs-factory.bin* and rename it to *wdr4300v1_en_3_14_3_up_boot(150518).bin*.

16. Now, in the OEM web interface, select the renamed file and choose upgrade. It will update the router and reboot.

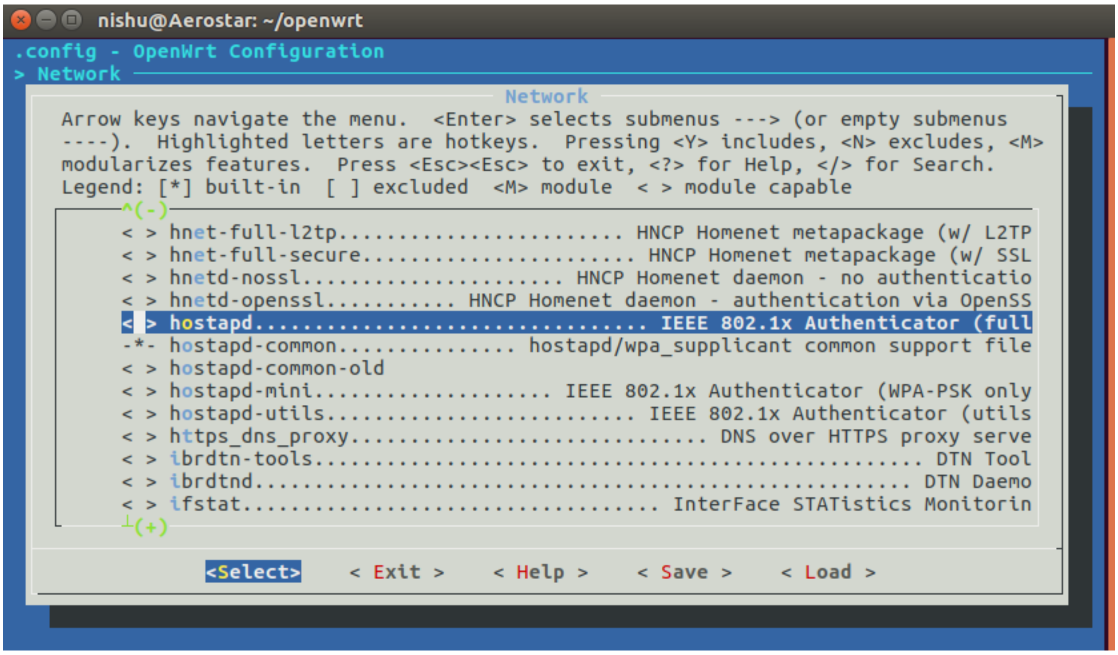
17. OpenWrt has been successfully installed in the router.



```
nishu@Aerostar: ~/openwrt
.config - OpenWrt Configuration
> Network
                                 Network
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable
          ^(-)
                IP Addresses and Names  --->
                Instant Messaging  --->
                Linux ATM tools  --->
                NMAP Suite  --->
                NTRIP  --->
                OLSR.org network framework  --->
                Open vSwitch  --->
                Printing  --->
                Routing and Redirection  --->
                SSH  --->
                THC-IPv6 attack and analyzing toolkit  --->
                Telephony  --->
                Time Synchronization  --->
          ┴(+)

            <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 7.1: Open vSwitch option in OpenWrt build configuration menu



```
nishu@Aerostar: ~/openwrt
.config - OpenWrt Configuration
> Network ──────────────────────────────────────────────────────────────
                                   Network
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable
        ^(-)
        < > hnet-full-l2tp...................... HNCP Homenet metapackage (w/ L2TP
        < > hnet-full-secure...................... HNCP Homenet metapackage (w/ SSL
        < > hnetd-nossl...................... HNCP Homenet daemon - no authenticatio
        < > hnetd-openssl.......... HNCP Homenet daemon - authentication via OpenSS
        < > hostapd................................ IEEE 802.1x Authenticator (full
        -*- hostapd-common............... hostapd/wpa_supplicant common support file
        < > hostapd-common-old
        < > hostapd-mini................... IEEE 802.1x Authenticator (WPA-PSK only
        < > hostapd-utils......................... IEEE 802.1x Authenticator (utils
        < > https_dns_proxy............................ DNS over HTTPS proxy serve
        < > ibrdtn-tools................................................ DTN Tool
        < > ibrdtnd..................................................... DTN Daemo
        < > ifstat................................ InterFace STATistics Monitorin
        ┴(+)

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 7.2: Hostapd option in OpenWrt build configuration menu

7.2 Router Configuration

The router has been freshly updated with the OpenWrt firmware. To configure the router, it is first connected via shell which will be discussed below.

The router is first connected via the shell using the command *sudo ssh 192.168.1.1* as seen in the test layout 8.1. This will open a OpenWrt configuration terminal that look like in the figure 4.1.

The wireless configuration has to be modified to create two ssid's, this can be accessed using the internal editor with the command *nano /etc/config/wireless*. the configuration contains all the information related to the wireless module. In the configuration, there are two device names and only these two needs to be modified. The device configured to use a RADIUS server in the server option and the key, the ssid's are changed to OpenWrt and OpenWrt 5G respectively. The configuration will look like as shown below after modification. The complete configuration can be accessed from the appendix

Wireless configuration

```

config wifi-iface
option device 'radio0'
option mode 'ap'
option ssid 'OpenWrt'
option server '192.168.1.169'
option key 'testing123'
option encryption 'wpa2'
option network 'wifi'

config wifi-iface
option device 'radio1'
option mode 'ap'
option server '192.168.1.169'
option key 'testing123'
option ssid 'OpenWrt 5G'
option encryption 'wpa2'
option network 'wifi'
```

The second step would be to modify the network configuration in order to create two separate networks with bridge *br-lan* for the interface eth0.1 hosting the network *192.168.1.1* and the bridge *br-wifi* for the interfaces eth0.2, wlan0, wlan1 hosting the second network *192.168.3.1* for which the client is assigned a dhcp post authentication. Separating the rest of the ports as individual interfaces allow them to be configured with OVS bridge which can be seen in the configuration below. The full configuration file is added in the appendix of this document Network configuration

```
config interface 'lan'
option type 'bridge'
option ifname 'eth0.1'
option proto 'static'
option ipaddr '192.168.1.1'
option netmask '255.255.255.0'
option ip6assign '60'

config interface 'wifi'
option type 'bridge'
option ifname 'eth0.2'
option proto 'static'
option ipaddr '192.168.3.1'
option netmask '255.255.255.0'
option ip6assign '60'

config interface 'lan2'
option ifname 'eth0.2'

config interface 'lan3'
option ifname 'eth0.3'

config interface 'lan4'
option ifname 'eth0.4'
```

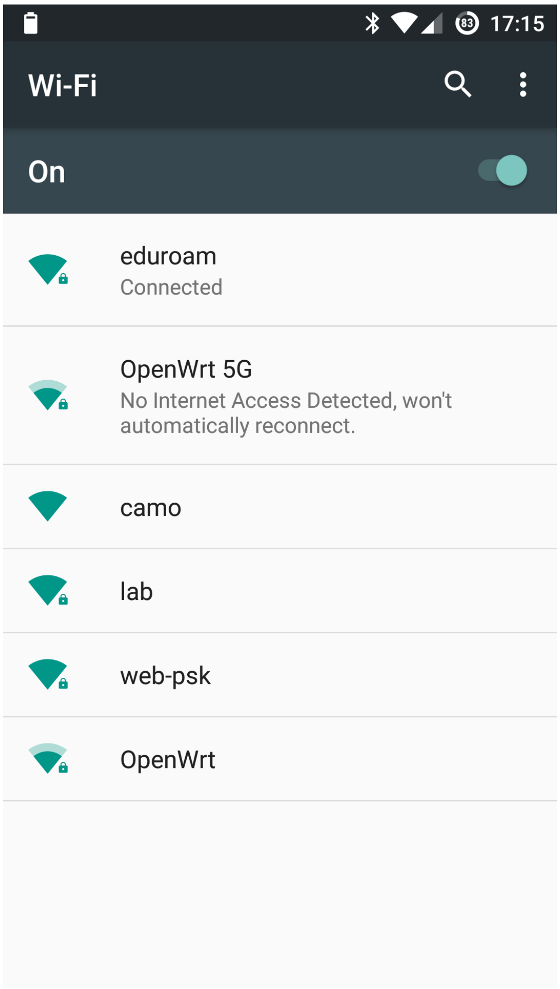
```

config interface 'lan5'
option ifname 'eth0.5'
```

To enable the new network to provide DHCP to the devices that connects to the OVS bridge, the DHCP has to be modified to accomodate the new network. This can be easily access in the editor with the commans *nano /etc/config/dhcp* and the configuration shown below is added to the file and saved.

```
config dhcp 'lan4'
option interface 'lan4'
option ignore '1'
```

Once the configurations are complete, the router is restarted to apply the changes. It is verified by logging into the web interface with the ip *192.168.1.1*. This will open a LuCi web interface in the browser and shows the routers status and reflects the changes that were made with the configuration.



17:15

Wi-Fi

On

eduroam
Connected

OpenWrt 5G
No Internet Access Detected, won't automatically reconnect.

camo

lab

web-psk

OpenWrt

Figure 7.3: SSID's OpenWrt and OpenWrt 5G Available on client device

7.3 Configuring Open vSwitch

The Open vSwitch is configured to connect with the RYU controller and assign ports to be managed by the controller. The configuration is made by using the following commands in the OpenWrt shell terminal.

1. Login to OpenWrt router using shell via the command *sudo ssh 192.168.1.1*.
2. The installation of OVS is checked using the following command *ovs-vsctl show*, if it says there is no such command, then there is no OVS installed. Instead if it shows an empty entry then OVS is installed but not configured.
3. The OVS bridge is created using the command *ovs-vsctl add-br br0*

4. To set the controller for OVS, the command *ovs-vsctl set-controller br0 tcp:192.168.1.207:6633* is used, where 6633 is the standard OpenFlow port.

5. The failsafe mode in a OVS switch tell the switch how to function in case of a connection failure with the controller. There are two modes, **secure** and **stand_alone**. The **secure** mode will not allow any packets to pass through whereas the **stand_alone** mode will function as a normal switch. For this project, the failsafe mode is set to secure to isolate the network using the following command *ovs-vsctl set-fail-mode br0 secure*.

6. The ports that needs to be managed are added to the bridge using the commands

```
        ovs-vsctl add-port br0 eth0.3
        ovs-vsctl add-port br0 eth0.4
        ovs-vsctl add-port br0 eth0.5

```

7. The configuration is verified by the command ovs-vsctl show, this will show the configuration that was made in the OVS.

7.4 MySQL Setup

The following steps explain the procedure to install and configure MySQL server and populate its database to work with Freeradius server on the ubuntu virtual machine.

1. MySQL is downloaded and installed on the Linux PC using the following command in terminal *sudo apt-get install mysql-server*

2. Once the server is installed, a database called radius is created using the following command.

```
    mysql -uroot -p
    CREATE DATABASE radius;
        GRANT ALL ON radius.* TO radius@localhost IDENTIFIED BY "radpass";
    exit

```

3. The schema for **Freeradius** is added to the database using the following command: *mysql -u root -p radius < /etc/freeradius/sql/mysql/schema.sql*

4. To manage the database a **PHPMyadmin** tool is installed and configured to connect with the MySQL database. It provides a web interface to manipulate the complete database.

5. **Radcheck** table contains the user credentials, it was modified to add a port id column where a port id is manually assigned to each user.

7.5 Installing Ubuntu Virtual Machine and Freeradius Server [16]

The initial method to configure the Freeradius and MySQL server within the OpenWrt router failed due to memory restrictions. Therefore, the Freeradius is installed in an Ubuntu virtual machine configured in a Virtual Box environment. The steps involved is discussed as follows.

1. A new virtual machine is created using the latest *Ubuntu 16.10 iso* image in Virtual Box with 2GB RAM and the network is bridged with eth1 which is connected to the internet, initially to download and install Freeradius.

2. Freeradius is downloaded and installed using the following command *sudo apt-get install freeradius*.

3. The radius service is started using the command *freeradius -x* if it throws an error, then error shown is debugged and fixed.

4. Edit the **clients.config** file in */etc/freeradius/* directory and add the following line in the file

```
    client 192.168.1.1{
            secret = testing123
    }

```

5. Now, the Freeradius server is configured to use MySQL database for authentication and accounting. To configure, the file **radiusd.conf** in the directory */etc/freeradius/* and the following steps are taken.

   a) Include **sql.conf** is uncommented.

   b) In the file **sql.conf** which is in the same directory, the database name is added in the line *database = "mysql"*

   c) Under connection info add

```
        server = "localhost"
        login = "radius"
        password = "radpass"
        radius_db = "radius"

```

6. To store the clients MAC address in the DB, the calling-station-id information is added in the schema file **dialup.conf** which resides in */etc/freeradius/sql/mysql/* directory. The information is added in the **radpostauth** table under Authentication Logging Queries section in the file.

```
    postauth_query = "REPLACE INTO ${postauth_table} \
    (user, pass, reply, date, CallingStationId) \
    VALUES ( \
    '%{User-Name}', \
    '%{%{User-Password}:-%{Chap-Password}}', \
    '%{reply:Packet-Type}', '%S', '%{Calling-Station-Id}')"

```

7. Finally, the Freeradius server is tested using the following to check if authentication works properly by using the following command *radtest test radpass 127.0.0.1 0 testing123* where testing123 is the radius key configured in the access point. Access-accept message is received when authentication is successful else, the error can be debugged using the command *sudo freeradius -X* in the terminal

8 Evaluation

To evaluate whether the developed application works as intended, it is tested against the testbed setup as shown in this figure below. This chapter also discusses about the performance of RYU controller and Open vSwitch against the application's performance.

8.1 Testbed Description

The image 8.1 shows how the environment for testing is set up. Each section is explained below in details about the configuration and how each port functions against the devices connected to it.

The Mobile Client is a Samsung Nexus Android phone and the second device used is also an Android device capable of associating with a network via 802.1x. The Access Point is a TP-Link WDR4300 WLAN router capable of dual band wireless access. This is convenient for the project as each band can be associated to a separate SSID (OpenWrt and OpenWrt 5G) that clients can connect with using different credentials.

The physical switches of the access point is shown separately for a better view. The router consists of five Ethernet ports including one WAN port. The WAN port is configured to work as a LAN access port for management access from the Linux PC. The Ethernet ports Eth0.1 and Eth0.2 are connected using the OpenWrt br-lan bridge. The port Eth0.2 is connected to Eth0.3 using a physical loop cable since internal looping was not possible due to constrains in the board design.

The bridge managed by Open vSwitch consist of the ports from Eth0.3 to Eth0.5 and includes a DHCP that provides a separate 192.168.3.1 network and gateway for all the devices connect to these ports. The ports Eth0.4 and Eth0.5 are connected to an IIS server running on a Windows 7 and Windows 10 operating systems respectively on two separate Laptops. Each of these server machines are connected to the OVS managed ports on a separate network, the Windows 7 machine is connected using 192.168.3.126 and the Windows 10 with 192.168.3.136.

The Linux PC on the other hand, consist of three physical network interface cards numbered in order from Eth1 to Eth3. It runs Edubuntu, a flavor of Ubuntu designed

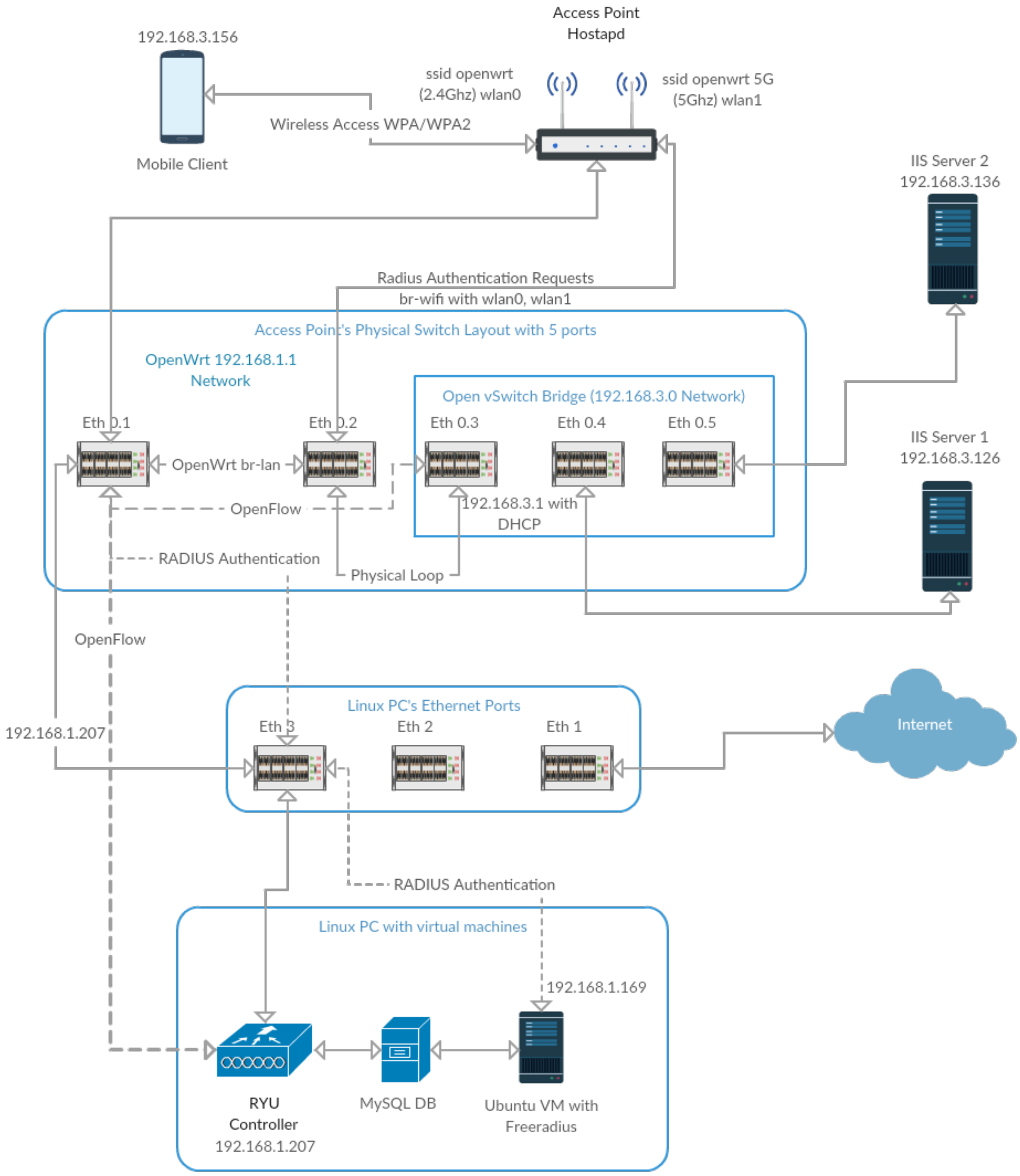


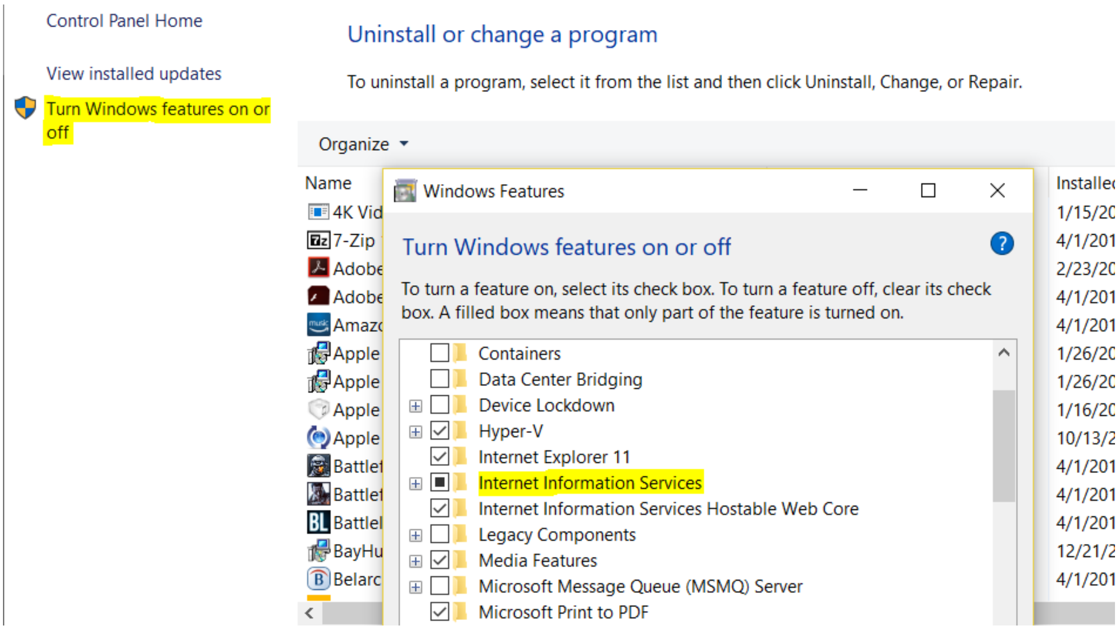
Figure 8.1: Physical Layout of the Test Environment

for educational use. The port Eth1 is connected to the internet and the port Eth3 is connected to the router which is associated with an IP 192.168.1.207. The PC also hosts a Virtual Box Ubuntu VM running Freeradius, the virtual network adapter is bridged with Eth3 port in the PC and has the IP 192.168.1.169. The application MySQL and RYU controller reside in the PC and are associated to the PC's IP.

8.1.1 IIS Setup on Windows

The following steps explains how to enable IIS server and host a test website associated with the machines IP that can be accesses from the mobile client.

1. To enable IIS server, Start -> Control Panel -> Programs and Features -> Add Features on or off (on the left pane in the window). See figure 8.2 below.
2. This will enable a pop up menu with several options to enable or disable. In the menu, choose the Internet Information Services and enable the sub options also.
3. Once the installation is complete and after a reboot, the IIS application can be opened from the programs option in Windows or a simple windows search will also show the application.
4. In the IIS window as shown in the figure 8.3, the server is enabled and running by default, it can be checked by typing localhost in a browser.
5. The hosted site is in the directory C:/inetpub/wwroot/, iisstart.htm is the default page that shows when accessed from browser. This would be enough for this project to show if the mobile clients can access these sites.



Control Panel Home

View installed updates

Turn Windows features on or off

Uninstall or change a program

To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.

Organize ▾

Name	Installe
4K Vid	1/15/20
7-Zip	4/1/201
Adobe	2/23/20
Adobe	4/1/201
Amaz	4/1/201
Apple	1/26/20
Apple	1/26/20
Apple	1/16/20
Apple	10/13/2
Battle	4/1/201
Battle	4/1/201
Battle	4/1/201
BayHu	12/21/2
Belarc	4/1/201

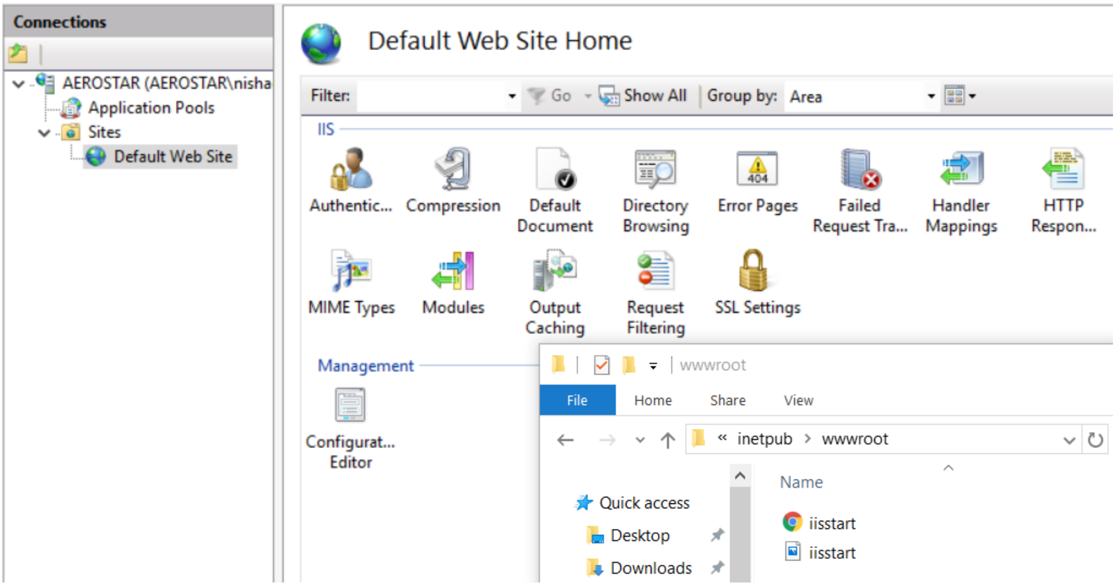
Windows Features

Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- □ Containers
- □ Data Center Bridging
- ⊞ □ Device Lockdown
- ⊞ ☑ Hyper-V
- ☑ Internet Explorer 11
- ⊞ ■ Internet Information Services
- ☑ Internet Information Services Hostable Web Core
- ⊞ □ Legacy Components
- ⊞ ☑ Media Features
- ⊞ □ Microsoft Message Queue (MSMQ) Server
- ☑ Microsoft Print to PDF

Figure 8.2: Enabling Internet Information Services (IIS) in Windows



Connections

- AEROSTAR (AEROSTAR\nisha
  - Application Pools
  - Sites
    - Default Web Site

Default Web Site Home

Filter: ▾ Go ▾ Show All | Group by: Area ▾

IIS

Authentic... Compression Default Document Directory Browsing Error Pages Failed Request Tra... Handler Mappings HTTP Respon...

MIME Types Modules Output Caching Request Filtering SSL Settings

Management

Configurat... Editor

wwwroot

File Home Share View

« inetpub › wwwroot

Quick access
Desktop
Downloads

Name
iisstart
iisstart

Figure 8.3: IIS Server options and root directory

8.1.2 Associating Two Mobile Clients to Access Point

To test the handling capability of the user segregation application, two mobile clients are used as mentioned previously. There are two user ids (test and radius) that are predefined in the MySQL DB which are associated to two out port ids. The users should only be logged in once after the RYU controller is started for the user segregation application to associate the users to its out port's. The user 'test' is logged in via the OpenWrt ssid and the user 'radius' is logged in via the OpenWrt 5G ssid as seen in the figure 7.3.

8.2 RYU Controller Performance

The performance of RYU is discussed in the following steps, starting from running the controller to the response time of handling the associated mobile clients.

The RYU controller is initialized and started using the command *ryu-manager –verbose usr_seg.py > /tmp/log 2>&1*, the verbose output can be viewed in real time by tailing the output from the logfile using the command *tail -f /tmp/log* in a separate terminal. The detailed verbose output of RYU can be accessed in the appendix named RYU Verbose output.

In the OpenWrt shell, the OVS is checked if the controller is active by issuing a command in the shell terminal *ovs-vsctl show*, if it shows 'connection is true' then the controller has access to the OVS switch. The flow table shows the current data path that is stored for each connected device and can be viewed by using the command *ovs-dpctl show*, this will show the ports and its internal port id's starting from 1 to 3 for physical ports Eth0.3 to Eth0.5.

From the testbed layout, it can be seen that the initial RADIUS requests are made by the hostapd thru the 192.168.1.1 network. Once the authentication is complete, the mobile client initiates a DHCP request which is captured by the RYU user segregation application. The application then instructs the OVS to associate the user client to the out port id from the MySQL DB. This can be verified in the data path that is stored in the OVS Flow Table using the command as before *ovs-dpctl show*. From the output, the out port assigned for the client's MAC address can be seen.

The RYU verbose log also contains this information when the user segregation application initiates the instruction to change the flow for the client. The log is documented in the appendix which can be access here RYU log trace

8.3 Open vSwitch performance

The performance of the Open vSwitch is tested during various load scenarios using tools such as Iperfv3 and ping times. The steps to execute the tests and also analyzing the results are discussed in detail below.

Iperf is a testing tool that provies simulated loads for testing device performance in a network. The tool can be configured to generate traffic of varying sizes. It also provides a lot of customizations for each test scenarios. The Man page for Iperf shows all the options that can be used with the tool. For this project, the default settings are used. The Iperf server is running on the IIS server machines and the Iperf client was run on both IIS server machines and the mobile client.

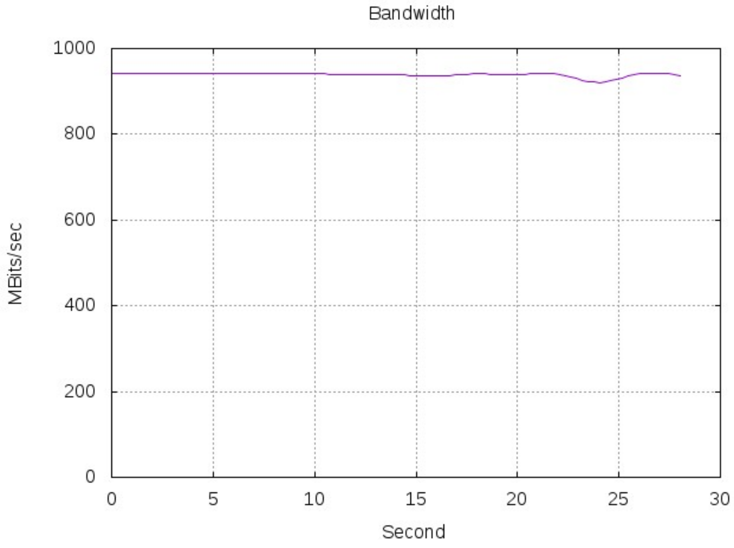
The two scenarios for testing considered here are:

- Scenario 1: Without RYU or OpenWrt to test the standard switch capacity
- Scenario 2: With RYU and Open vSwitch running.

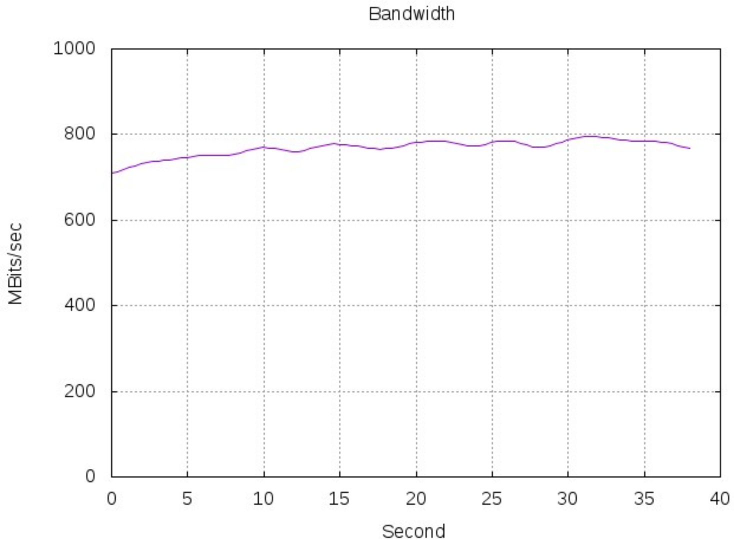
ping tests were also made keeping the same scenarios as above. The results of the Iperf tests were ploted as a graph for a better analysis and each test was made three times.

8.3.1 Iperf result comparison

The graphs below show the results of the Iperf test between the IIS server machines with and without the RYU and OVS running.



(a) Bandwidth without RYU



(b) Bandwidth with RYU

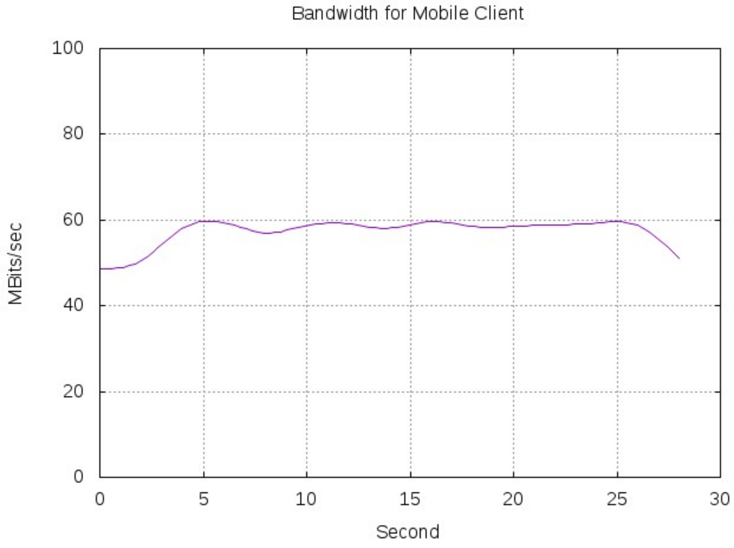
Figure 8.4: Iperf Throughput between two IIS servers

From the comparison, it is evident that there is a slight drop in the throughput when the RYU is running. Each ethernet port on the access point have a maximum bandwidth of 1Gbps, this is maintained when there is no RYU running on the access point. The values from the figure is shown be consistent with the maximum capacity hovering around 900Mbps.

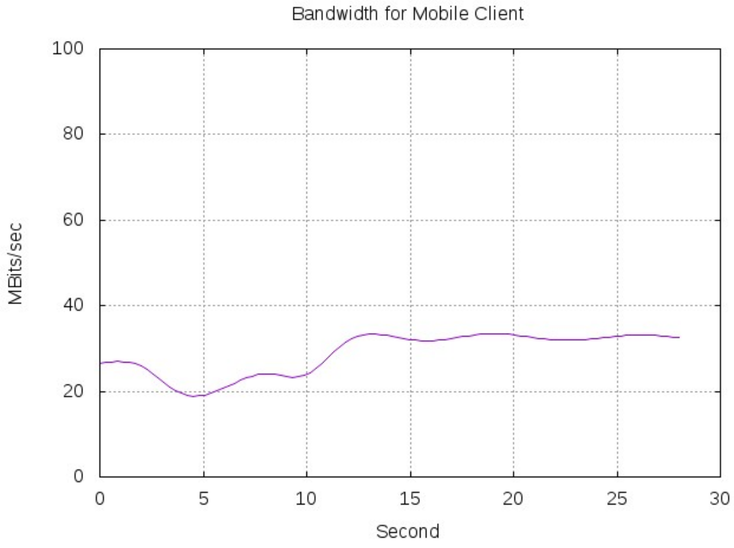
Whereas in the second figure, its clearly visible that the maximum achieved throughput is only available to be around 800Mbps and the waves in the graph represent the inconsistency to maintain target bandwidth.It can be concluded from this test that, running the RYU and OVS on the access point does impact the performance and bandwidth though not significantly as these speeds are still large enough to handle large traffic but with some latency.

In the next scenario, the Iperf tool is run on the mobile client and the IIS server machines though this test has a limitation. The wireless bandwidth is limited to the speeds of 802.11g and 802.11n which are mostly 54Mbps and around 100Mbps respectively. Though this test doesn't accurately represent the capability of the RYU and OVS performance, it does show the capacity of the access point to handle through put on the wireless medium as well.

The following two figures shows the comparison of throughput bandwidth between the two scenarios.



(a) Bandwidth without RYU



(b) Bandwidth throughput with RYU

Figure 8.5: Iperf Throughput between Mobile client and IIS servers

The results are similar to the once above, the one without RYU and OVS seem to perform better when compared to using a RYU and OVS in the access point. As can be seen from the first graph, the bandwidth slowly starts to rise within the first 5 seconds to a maximum speed of 60Mbps which is the capacity of the wireless network. Where as in the second graph, the mobile device was only able to achieve a maximum speed of around 35 Mbps which is far below the full capacity of the wireless bandwidth and it takes more than 10 seconds to actually achieve its maximum capacity.

The initial dips are due to the wireless congestion from various other networks in the tested environment. In spite of that, the achieved bandwidth is still below the maximum capacity of the wireless network.From the above tests, it is evident that on a mobile device the impact of running RYU and OVS on the access point is more significant and can deteriorate when there is congestion in the wireless medium.

8.3.2 Ping tests

Ping tests provides an insight into the latency introduced due to various other activities performed by the network interfaces when a device get connected to the network from the first ping for ARP to subsequent pings.

The ping tests were executed in two methods, one is a normal test and the other is to flood. They were performed in the same scenarios used for the previous tests.

The normal results of the ping test in both the scenarios is shown below.

```
--- 192.168.1.126 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9197ms
rtt min/avg/max/mdev = 0.141/0.291/0.354/0.058 ms
```

Listing 8.1: Ping Test Without OVS

```
--- 192.168.3.126 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9205ms
rtt min/avg/max/mdev = 0.273/0.313/0.343/0.028 ms
```

Listing 8.2: Ping Test With OVS

From the result, it can be noticed that the average time is around 0.291 milliseconds when there is no OVS installed and when the OVS is running, the average is around 0.313 milliseconds. It can be inferred that there is only a slight latency between the two scenarios and that the OVS does impact a little in this regard. Though this will not be a major concern for a few devices but in a large network, this reduction in latency can impact greatly when many number of devices are connected to the access point bringing down the efficiency of the network. The results are similar when tested by ping flood. The results are shown below.

```
--- 192.168.1.126 ping statistics ---
100000 packets transmitted, 100000 received, 0% packet loss, time
    18625ms
```

```
rtt min/avg/max/mdev = 0.061/0.180/51.981/0.450 ms, pipe 4, ipg/ewma 0.186/0.134 ms
```

Listing 8.3: Ping Flood Without OVS

```
--- 192.168.3.126 ping statistics ---
100000 packets transmitted, 100000 received, 0% packet loss, time 21643ms
rtt min/avg/max/mdev = 0.111/0.205/48.572/0.526 ms, pipe 4, ipg/ewma 0.216/0.196 ms
```

Listing 8.4: Ping Flood With OVS

The average round trip time for flood statistics in both the scenarios show that running the OVS only creates a 0.1 millisecond delay but overall for transmitting 100000 packets, it took around 3000 milliseconds more. The results are conclusive as with the previous test and demonstrates the significant increase in latency in a high traffic environment.

9 Conclusion

The software defined networking approach to orchestrating networks is picking up momentum. By using open interfaces and protocols to programmatically control network elements, it is becoming easier to introduce new functionalities to a network, without being constrained by vendor lock-in. In this thesis, we consider the specific case of introducing user segregation to enterprise WLANs. Through the user segregation application, we have demonstrated that by using a set of abstractions, it is possible to segregate users based on their de-vice mac address and host two networks within an Access Point and provide isolation between them.

9.1 Discussion

This thesis explores the idea of hosting flexible enterprise WLAN networks within an single Access Point, wherein the access points being managed by a software defined network based controller and the isolating the users and their networks at the MAC level. We have taken a step forward in achieving user segregation as outlined in the chapter 6 Designing the Application. The ideas described are validated through an implementation of the application described in chapter 7 Implementation. A performance evaluation of the system as described in chapter 8 Evaluation, which demonstrates the practicality of the system and the application within the context of the test environment at the University's communication networks lab. The tests were conducted using mobile devices and PC's in a controlled environment with predefined users to simulate an actual network.

9.2 Technology Demonstrator

The results obtained from the tests conducted were satisfactory. The application performed as designed and could achieve isolation of users between networks and the modification of flow table in the Open vSwitch happened in real time for users with assigned out port ids from the database.

The application was designed to test the capability of the openflow based controller and OpenWrt to host and manage multiple networks within an access point and provide authentication based on RADIUS similar to Hotspot 2.0 to provide seamless mobility for users when roaming on different networks. There were many constrains faced during the designing of this application, the OpenWrt system doesn't support implementation of Hotspot2.0 in its system, memory constraints and the processor speeds also impacted the performance of the rotuer as a result the Freeradius and MySQL has to be installed in a separate machine, instead on the access point itself. One of the major reasons that this application cannot be used in a real environment is a serious lack of security. The application only provides secure connection between the users and their corresponding IIS servers while the other network and its iis server is completely isolated. This can be seen the figure. Whereas the IIS servers can communicate with each outer and also due to ARP request from the servers with the switch, the OVS learns the path to all the connected devices and thus allowing one server to access the mobile device connected in the other network in reverse. This compromises the security when one server is breached, then it can be used to gain access to all the devices connect to the network.

9.3 Future Enhancements

The application and the system can be further enhanced by introducing user groups on each network and make the SDN controller to behave as a firewall and provide access control to these groups thereby enhancing the security and avoiding full network access during security breach. The other possible extension would be to use a different device as an access point which has more memory and computing power. This can allow integrating Freeradius and MySQL within the access point and thereby making the system more independent and controlled only by the SDN controller. This would ultimately simplify installation and reduce deployment costs overall.

Bibliography

[1] What is BIC-IRAP? URL http://www.bic-irap.de/index.php/en. Online; accessed 01-March-2017.

[2] What is SDN?, . URL https://www.opennetworking.org/sdn-resources/sdn-definition. Online; accessed 01-March-2017.

[3] IEEE Standards Association et al. 802.11-2012-ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std*, 802, 2012.

[4] Understanding wi-fi hotspot 2.0 and how to leverage it for your business, by jason guest. http://hotelexecutive.com/business_review/3674/understanding-wi-fi-hotspot-20-and-how-to-leverage-it-for-your-business. (Accessed on 03/02/2017).

[5] Switching hub — ryubook 1.0 documentation, . URL https://osrg.github.io/ryu-book/en/html/switching_hub.html. (Accessed on 03/06/2017).

[6] What is a floodlight controller? - defined. https://www.sdxcentral.com/sdn/definitions/sdn-controllers/open-source-sdn-controllers/what-is-floodlight-controller/, . (Accessed on 03/08/2017).

[7] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *NSDI*, pages 117–130, 2015.

[8] What is openwrt and why should i use it for my router? http://www.makeuseof.com/tag/what-is-openwrt-and-why-should-i-use-it-for-my-router/, . (Accessed on 03/15/2017).

[9] Openflow - open networking foundation. https://www.opennetworking.org/sdn-resources/openflow, . (Accessed on 03/16/2017).

[10] Rfc 2865 - remote authentication dial in user service (radius). https://tools.ietf.org/html/rfc2865, . (Accessed on 03/17/2017).

[11] Jyh-Cheng Chen and Yu-Ping Wang. Extensible authentication protocol (eap) and ieee 802.1x: tutorial and empirical experience. *IEEE Communications Magazine*, 43(12):supl.26–supl.32, Dec 2005. ISSN 0163-6804. doi: 10.1109/MCOM.2005.1561920.

[12] Virtual environments — the hitchhiker's guide to python. http://docs.python-guide.org/en/latest/dev/virtualenvs/, . (Accessed on 03/20/2017).

[13] Ryu sdn framework. https://osrg.github.io/ryu/, . (Accessed on 03/20/2017).

[14] Openwrt build system – installation [openwrt wiki]. https://wiki.openwrt.org/doc/howto/buildroot.exigence, . (Accessed on 03/20/2017).

[15] Architecture — ryubook 1.0 documentation. https://osrg.github.io/ryu-book/en/html/arch.html, . (Accessed on 04/06/2017).

[16] guide/sql howto. https://wiki.freeradius.org/guide/SQL-HOWTO#Create_MySQL_Database. (Accessed on 04/08/2017).

[17] Sdn architecture diagram, . URL https://www.sdxcentral.com/wp-content/uploads/2015/03/sdn-architecture.png. Online; accessed 02-March-2017.

[18] Ryu-controller-sdn-framework.jpg (jpeg image, 900 × 495 pixels). URL https://www.sdxcentral.com/wp-content/uploads/2014/09/ryu-controller-sdn-framework.jpg. (Accessed on 03/08/2017).

[19] Floodlight architecture diagram. https://www.sdxcentral.com/wp-content/uploads/2014/09/floodlight-open-sdn-controller-diagram.jpg, . (Accessed on 03/08/2017).

[20] Architectural_framework.jpg (717×435). https://wiki.opendaylight.org/images/b/b1/Architectural_Framework.jpg, . (Accessed on 03/14/2017).

[21] featured-image.jpg (714×594). http://openvswitch.org/assets/featured-image.jpg. (Accessed on 03/14/2017).

[22] openwrt4-49e2cc8.jpg (554×493). http://img110.xooimage.com/files/0/d/4/openwrt4-49e2cc8.jpg, . (Accessed on 03/16/2017).

[23] bootstrap-luci-theme.png (959×580). https://i1.wp.com/advanxer.com/blog/wp-content/uploads/2013/02/bootstrap-luci-theme.png, . (Accessed on 03/16/2017).

[24] 12-8-openflow-diagram.jpg (417×348). https://www.opennetworking.org/images/stories/sdn-resources/openflow/12-8-OpenFlow-Diagram.jpg, . (Accessed on 03/16/2017).

[25] openflow-protocol.png (217×242). http://flowgrammable.org/static/media/uploads/components/protocol.png. (Accessed on 03/16/2017).

[26] switch_anatomy.png (335×209). http://flowgrammable.org/static/media/uploads/components/switch_anatomy.png, . (Accessed on 03/17/2017).

[27] switch_agent_anatomy.png (385×166). http://flowgrammable.org/static/media/uploads/components/switch_agent_anatomy.png, . (Accessed on 03/17/2017).

[28] switch.png (473×250). http://flowgrammable.org/static/media/uploads/components/switch.png, . (Accessed on 03/17/2017).

[29] packet_lifecycle.png (710×138). http://flowgrammable.org/static/media/uploads/components/packet_lifecycle.png, . (Accessed on 03/17/2017).

[30] Radius components (513×318). https://i-technet.sec.s-msft.com/dynimg/IC195130.gif, . (Accessed on 03/17/2017).

[31] Radius operation (560×460). http://www.wi-fiplanet.com/img/tutorial-radius-fig1.gif. (Accessed on 03/17/2017).

[32] 802.1x_over_802.11_with_eap_expansion.png (513×393). https://www.eduroam.us/files/images/admin_guide/technical_overview/802.1x_over_802.11_with_EAP_expansion.png. (Accessed on 03/17/2017).

[33] fig1.png (987×669). https://osrg.github.io/ryu-book/en/html/_images/fig1.png. (Accessed on 04/06/2017).

[34] hostapd: Ieee 802.11 ap, ieee 802.1x/wpa/wpa2/eap/radius authenticator. http://w1.fi/hostapd/. (Accessed on 03/02/2017).

[35] Rfc 5412 - lightweight access point protocol. https://tools.ietf.org/html/rfc5412. (Accessed on 03/02/2017).

[36] Ieee xplore full-text pdf:. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5721908. (Accessed on 03/02/2017).

[37] Wi-fi certified passpoint | wi-fi alliance. http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-passpoint. (Accessed on 03/02/2017).

[38] What's software-defined networking (sdn)? https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/, . (Accessed on 03/06/2017).

[39] What is OpenStack?, . URL http://www.openstack.org/software/. Online; accessed 06-November-2016.

[40] Lavanya Jose, Minlan Yu, and Jennifer Rexford. Online measurement of large traffic aggregates on commodity switches. http://static.usenix.org/events/hotice11/tech/full_papers/Jose.pdf. (Accessed on 03/14/2017).

[41] Ankur Kumar Nayak, Alex Reimers, Nick Feamster, and Russ Clark. Resonance: Dynamic access control for enterprise networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, pages 11–18, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-443-0. doi: 10.1145/1592681.1592684. URL http://doi.acm.org/10.1145/1592681.1592684.

[42] Jeffrey R Ballard, Ian Rae, and Aditya Akella. Extensible and scalable network monitoring using opensafe. In *INM/WREN*, 2010.

[43] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19, 2010.

[44] Richard Wang, Dana Butnariu, Jennifer Rexford, et al. Openflow-based server load balancing gone wild. *Hot-ICE*, 11:12–12, 2011.

[45] What is open vswitch (ovs)? https://www.sdxcentral.com/cloud/open-source/definitions/what-is-open-vswitch/, . (Accessed on 03/14/2017).

[46] Special-report-openflow-and-sdn-state-of-the-union-b.pdf. https://www.opennetworking.org/images/stories/downloads/sdn-resources/special-reports/Special-Report-OpenFlow-and-SDN-State-of-the-Union-B.pdf, . (Accessed on 04/26/2017).

[47] 42948.pdf. http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42948.pdf, . (Accessed on 04/26/2017).

[48] Openflow » what is openflow? http://archive.openflow.org/wp/learnmore/, . (Accessed on 03/16/2017).

[49] Sdn / openflow | flowgrammable. http://flowgrammable.org/sdn/openflow/#tab_protocol, . (Accessed on 03/16/2017).

[50] Sdn / openflow | flowgrammable. http://flowgrammable.org/sdn/openflow/#tab_switch. (Accessed on 03/16/2017).

[51] Aaa and nas. https://www.tutorialspoint.com/radius/aaa_and_nas.htm, . (Accessed on 03/17/2017).

[52] Radius protocol and components. https://technet.microsoft.com/en-us/library/cc726017(v=ws.10).aspx, . (Accessed on 03/17/2017).

Appendix

Appendix A

Router Configuration Content

In this Appendix Router Configuration Content, the reader can find the specifically mentioned configuration for the router.

A.1 Wireless configuration

The Wireless configuration parameters set in the [OpenWrt] are:

```
config wifi-device 'radio0'
option type 'mac80211'
option hwmode '11g'
option path 'platform/ar934x_wmac'
option htmode 'HT20'
option txpower '20'
option country 'DE'
option disabled '0'
option channel '5'

config wifi-iface
option device 'radio0'
option mode 'ap'
option ssid 'OpenWrt'
option server '192.168.1.169'
option key 'testing123'
option network 'wifi'
option encryption 'wpa2'

config wifi-device 'radio1'
option type 'mac80211'
option channel '36'
option hwmode '11a'
option path 'pci0000:00/0000:00:00.0'
option htmode 'HT20'
option txpower '17'
option country 'DE'
```

```

config wifi-iface
option device 'radio1'
option mode 'ap'
option server '192.168.1.169'
option key 'testing123'
option ssid 'OpenWrt 5G'
option encryption 'wpa2'
option network 'wifi'
```

A.2 Network configuration

The Network configuration parameters set in the [OpenWrt] are:

```

config interface 'loopback'
option ifname 'lo'
option proto 'static'
option ipaddr '127.0.0.1'
option netmask '255.0.0.0'

config globals 'globals'
option ula_prefix 'fd04:beb4:615d::/48'


config interface 'lan'
option type 'bridge'
option ifname 'eth0.1'
option proto 'static'
option ipaddr '192.168.1.1'
option netmask '255.255.255.0'
option ip6assign '60'

config interface 'wifi'
option type 'bridge'
option ifname 'eth0.2'
option proto 'static'
option ipaddr '192.168.3.1'
option netmask '255.255.255.0'
option ip6assign '60'

config interface 'lan2'
option ifname 'eth0.2'

config interface 'lan3'
option ifname 'eth0.3'
```

```

config interface 'lan4'
option ifname 'eth0.4'

config interface 'lan5'
option ifname 'eth0.5'

config switch
option name 'switch0'
option reset '1'
option enable_vlan '1'

config switch_vlan
option device 'switch0'
option vlan '1'
option ports '1 0t'
option vid '1'

config switch_vlan
option device 'switch0'
option vlan '2'
option ports '2 0t'
option vid '2'

config switch_vlan
option device 'switch0'
option vlan '3'
option ports '3 0t'
option vid '3'

config switch_vlan
option device 'switch0'
option vlan '4'
option vid '4'
option ports '0t 4'

config switch_vlan
option device 'switch0'
option vlan '5'
option vid '5'
option ports '0t 5'
```

A.3 DHCP configuration

The DHCP configuration parameters set in the [OpenWrt] are:

```

config dnsmasq
option domainneeded '1'
option boguspriv '1'
option filterwin2k '0'
option localise_queries '1'
option rebind_protection '1'
option rebind_localhost '1'
option local '/lan/'
option domain 'lan'
option expandhosts '1'
option nonegcache '0'
option authoritative '1'
option readethers '1'
option leasefile '/tmp/dhcp.leases'
option resolvfile '/tmp/resolv.conf.auto'
option localservice '1'

config dhcp 'lan'
option interface 'lan'
option start '100'
option limit '150'
option leasetime '12h'
option dhcpv6 'server'
option ra 'server'

config dhcp 'wifi'
option interface 'wifi'
option start '100'
option limit '150'
option leasetime '12h'
option dhcpv6 'server'
option ra 'server'

config dhcp 'wan'
option interface 'wan'
option ignore '1'

config odhcpd 'odhcpd'
option maindhcp '0'
option leasefile '/tmp/hosts/odhcpd'
option leasetrigger '/usr/sbin/odhcpd-update'

config dhcp 'lan4'
option interface 'lan4'
option ignore '1'
```

Appendix B

RYU Verbose output

In this Appendix, the logs from the `/tmp/log` file have been provided for different log purpose.

B.1 RYU log trace

```
loading app usr_seg.py
loading app ryu.controller.ofp_handler
instantiating app usr_seg.py of SimpleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK SimpleSwitch13
CONSUMES EventOFPPacketIn
CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
PROVIDES EventOFPPacketIn TO {'SimpleSwitch13': set(['main'])}
PROVIDES EventOFPSwitchFeatures TO {'SimpleSwitch13': set(['config'])}
CONSUMES EventOFPEchoRequest
CONSUMES EventOFPPortStatus
CONSUMES EventOFPEchoReply
CONSUMES EventOFPSwitchFeatures
CONSUMES EventOFPPortDescStatsReply
CONSUMES EventOFPHello
CONSUMES EventOFPErrorMsg
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7f42b40851d0> address:('192.168.1.1', 58461)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7f42b4085a90>
move onto config mode
EVENT ofp_event->SimpleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0xf35c1db1,OFPSwitchFeatures(auxiliary_id=0,capabilities=79,datapath_id=176968783353910,n_buffers=256,n_tables=254)
move onto main mode
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
```

```
packet truncated: only 170 of 342 bytes
Timestamp 2017-02-16 12:49:36.274578
Timestamp 2017-02-16 12:49:36.276005
outport_for_src tuple is None
Data is 00:26:9e:e2:b2:f8
packet in 176968783353910 00:26:9e:e2:b2:f8 ff:ff:ff:ff:ff:ff 2
DST is ff:ff:ff:ff:ff:ff
Out_Port before else flood condition None
Out_Port is Flooded 4294967291
Above actions Out_Port 4294967291
Actions is [OFPActionOutput(len=16,max_len=65509,port=4294967291,type=0)]
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
Timestamp 2017-02-16 12:49:36.485248
Timestamp 2017-02-16 12:49:36.486536
outport_for_src tuple is None
Data is 00:26:9e:e2:b2:f8
packet in 176968783353910 00:26:9e:e2:b2:f8 ff:ff:ff:ff:ff:ff 2
DST is ff:ff:ff:ff:ff:ff
Out_Port before else flood condition None
Out_Port is Flooded 4294967291
Above actions Out_Port 4294967291
Actions is [OFPActionOutput(len=16,max_len=65509,port=4294967291,type=0)]
Timestamp 2017-02-16 12:49:36.490576
Timestamp 2017-02-16 12:49:36.491804
outport_for_src tuple is None
Data is 00:26:9e:e2:b2:f8
packet in 176968783353910 00:26:9e:e2:b2:f8 33:33:ff:a2:39:fe 2
DST is 33:33:ff:a2:39:fe
Out_Port before else flood condition None
Out_Port is Flooded 4294967291
Above actions Out_Port 4294967291
Actions is [OFPActionOutput(len=16,max_len=65509,port=4294967291,type=0)]
Timestamp 2017-02-16 12:49:36.495438
Timestamp 2017-02-16 12:49:36.496494
outport_for_src tuple is None
Data is 00:26:9e:e2:b2:f8
packet in 176968783353910 00:26:9e:e2:b2:f8 33:33:ff:00:07:5c 2
DST is 33:33:ff:00:07:5c
```

```
Out_Port before else flood condition None
Out_Port is Flooded 4294967291
Above actions Out_Port 4294967291
Actions is [OFPActionOutput(len=16,max_len=65509,port=4294967291,type=0)]
Timestamp 2017-02-16 12:49:36.499691
Timestamp 2017-02-16 12:49:36.500575
outport_for_src tuple is None
Data is a0:0b:ba:c9:9e:04
packet in 176968783353910 a0:0b:ba:c9:9e:04 ff:ff:ff:ff:ff:ff 1
DST is ff:ff:ff:ff:ff:ff
Out_Port before else flood condition None
Out_Port is Flooded 4294967291
Above actions Out_Port 4294967291
Actions is [OFPActionOutput(len=16,max_len=65509,port=4294967291,type=0)]
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
packet truncated: only 170 of 350 bytes
Timestamp 2017-02-16 12:52:47.115680
Timestamp 2017-02-16 12:52:47.116894
outport_for_src tuple is ('2',)
Data is a0:0b:ba:c9:9e:04
packet in 176968783353910 a0:0b:ba:c9:9e:04 ff:ff:ff:ff:ff:ff 1
DST is ff:ff:ff:ff:ff:ff
Out_Port before else flood condition ('2',)
Setting Out_Port same as in table, changing flow 2
Above actions Out_Port 2
Actions is [OFPActionOutput(len=16,max_len=65509,port=2,type=0)]
Out_Port not flooded adding flow
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
Data is c8:5b:76:1b:ed:41
packet in 176968783353910 c8:5b:76:1b:ed:41 ff:ff:ff:ff:ff:ff 3
DST is ff:ff:ff:ff:ff:ff
Out_Port before else flood condition None
Out_Port is Flooded 4294967291
Above actions Out_Port 4294967291
Actions is [OFPActionOutput(len=16,max_len=65509,port=4294967291,type=0)]
Timestamp 2017-02-16 12:53:03.276915
Timestamp 2017-02-16 12:53:03.277904
outport_for_src tuple is None
Data is c8:5b:76:1b:ed:41
packet in 176968783353910 c8:5b:76:1b:ed:41 33:33:00:01:00:03 3
DST is 33:33:00:01:00:03
Out_Port before else flood condition None
Out_Port is Flooded 4294967291
Above actions Out_Port 4294967291
Actions is [OFPActionOutput(len=16,max_len=65509,port=4294967291,type=0)]
```

```
Timestamp 2017-02-16 12:53:03.282229
Timestamp 2017-02-16 12:53:03.283229
outport_for_src tuple is None
Data is c8:5b:76:1b:ed:41
packet in 176968783353910 c8:5b:76:1b:ed:41 01:00:5e:00:00:fb 3
DST is 01:00:5e:00:00:fb
Out_Port before else flood condition None
Out_Port is Flooded 4294967291
Above actions Out_Port 4294967291
Actions is [OFPActionOutput(len=16,max_len=65509,port=4294967291,type=0)]
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
Timestamp 2017-02-16 12:54:32.618185
Timestamp 2017-02-16 12:54:32.619362
outport_for_src tuple is ('3',)
Data is c0:ee:fb:20:41:24
packet in 176968783353910 c0:ee:fb:20:41:24 01:00:5e:00:00:16 1
DST is 01:00:5e:00:00:16
Out_Port before else flood condition ('3',)
Setting Out_Port same as in table, changing flow 3
Above actions Out_Port 3
Actions is [OFPActionOutput(len=16,max_len=65509,port=3,type=0)]
Out_Port not flooded adding flow
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
Timestamp 2017-02-16 12:54:36.051777
Timestamp 2017-02-16 12:54:36.052886
outport_for_src tuple is ('3',)
Data is c0:ee:fb:20:41:24
packet in 176968783353910 c0:ee:fb:20:41:24 01:00:5e:00:00:fb 1
DST is 01:00:5e:00:00:fb
Out_Port before else flood condition ('3',)
Setting Out_Port same as in table, changing flow 3
Above actions Out_Port 3
Actions is [OFPActionOutput(len=16,max_len=65509,port=3,type=0)]
Out_Port not flooded adding flow
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
Timestamp 2017-02-16 12:55:09.360397
Timestamp 2017-02-16 12:55:09.361360
```

Listing B.1: The RYU Verbose log

Appendix C

User Segregation Application code

In this Appendix User Segregation Application code, the entire application code written in Python is listed here.

```python
# Copyright (C) 2011 Nippon Telegraph and Telephone Corporation.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.
import MySQLdb
import sys
import datetime
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.lib.packet import udp



class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install table-miss flow entry
        #
        # We specify NO BUFFER to max_len of the output action due to
        # OVS bug. At this moment, if we specify a lesser number, e.g.,
        # 128, OVS will send Packet-In with invalid buffer_id and
        # truncated packet data. In that case, we cannot output packets
        # correctly.  The bug has been fixed in OVS v2.1.0.
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
        ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id, priority=priority, match=match, instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # If you hit this you might want to increase
        # the "miss_send_length" of your switch
        if ev.msg.msg_len < ev.msg.total_len:
            self.logger.debug("packet truncated: only %s of %s bytes", ev.msg.msg_len, ev.msg.total_len)
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        udp_payload = pkt.get_protocols(udp.udp)


        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            # ignore lldp packet
            return
        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})
    #self.logger.info ("Destination is %s", eth.dst)

    #creating a mysql connection to database -last edit 17/11

    connection = MySQLdb.connect(host = "192.168.1.169", user = "freerad", passwd = "pass", db = "radius")
    cursor = connection.cursor ()

    #cursor.execute ("select CallingStationId from radpostauth order by id desc LIMIT 1")
    #data = cursor.fetchall ()
    cursor.execute ("SELECT portid FROM radcheck WHERE username IN (SELECT user FROM radpostauth WHERE CallingStationId = %s AND id = (SELECT MAX(id) from radpostauth) )", src)
    outport_for_src = cursor.fetchone ()
    self.logger.info ("outport_for_src tuple is %s", outport_for_src)
    #portid = outport_for_src[0]
    self.logger.info ("Data is %s", src)
    cursor.close()
    connection.close ()
    # Mysql verification end

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)


        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

    #self.logger.info ("Mac to port in dst is %s ",self.mac_to_port[dpid] )
    self.logger.info ("DST is %s", dst)
        if dst in self.mac_to_port[dpid]:
        test = self.mac_to_port[dpid][dst]
        self.logger.info ("self.mac_to_port[dpid][dst] value %s", test)
        self.logger.info ("Out_Port before condition %s", outport_for_src)
        #try:
        #if portid in self.mac_to_port[dpid][dst]:
        if outport_for_src != None and all(outport_for_src):
            if int(outport_for_src[0]) == self.mac_to_port[dpid][dst]:
                out_port = self.mac_to_port[dpid][dst]
                self.logger.info ("Out_Port is the same as in database table %s", out_port)
            else:
                self.logger.info ("Out_Port is not allowed, dropping packet")
                return

        else:
        #except (TypeError,UnboundLocalError):
            out_port = self.mac_to_port[dpid][dst]
            self.logger.info ("Out_Port not defined in database, using learned port")
            #return
        else:
        #try:
        #if portid > 0 :
        self.logger.info ("Out_Port before else flood condition %s", outport_for_src)
        if outport_for_src != None and all(outport_for_src):
            out_port = int(outport_for_src[0])
            self.logger.info ("Setting Out_Port same as in table, changing flow %s", out_port)
        else:
        #except (TypeError,UnboundLocalError):

            out_port = ofproto.OFPP_FLOOD
        #out_port = 4
            self.logger.info ("Out_Port is Flooded %s", out_port)

    self.logger.info ("Above actions Out_Port %s", out_port)
        actions = [parser.OFPActionOutput(out_port)]
    self.logger.info ("Actions is %s", actions)

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
        self.logger.info ("Out_Port not flooded adding flow ")
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
            # verify if we have a valid buffer_id, if yes avoid to send both
            # flow_mod & packet_out
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, match, actions, msg.buffer_id)
                return
            else:
                self.add_flow(datapath, 1, match, actions)
        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id, in_port=in_port, actions=actions, data=data)
        datapath.send_msg(out)
```

Listing C.1: The User Segregation Mac Learning Application

Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.
Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistungen von folgenden Personen erhalten:

keine

Weitere Personen waren an der Abfassung der vorliegenden Arbeit nicht beteiligt. Die Hilfe eines Promotionsberaters habe ich nicht in Anspruch genommen. Weitere Personen haben von mir keine geldwerten Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Chemnitz, April 26, 2017

Nishant Ravi