# IMPLEMENTING A TRAFFIC ENGINEERING SERVICE IN SDN

Athanasios Xirofotos

University of Thessaly

Department of Electrical and Computer Engineering

VOLOS 2017, GREECE

# **Title of Thesis**: Implementing a traffic engineering service in SDN

(Υλοποίηση υπηρεσίας κυκλοφορίας με χρήση δικτύωσης ορισμένης από λογισμικό (SDN))

Author: Xirofotos Athanasios / Ξηροφώτος Αθανάσιος

Supervisors:

Athanasios Korakis , assistant Professor

Antonios Argyriou, assistant Professor

University of Thessaly

Department of Electrical and Computer Engineering

Volos 2017, Greece

*Dedicated to my Family and Friends*

## *Words of Appreciation*

*First and foremost, I would like to thank my supervisor Thanasis Korakis as well as Kostas Choumas for giving me the chance to cooperate with them. I also would like to thank them for the overall support and their guidance throughout this work. An extra special thanks to K.Choumas for helping me out with academic difficulties I had faced during my military obligations.*

*A huge thanks to all of my friends for the unwavering and unconditional support they have shown me throughout all these years we are together.*

*Lastly, I would like to thank my family for the constant support.*

## Περίληψη

Τα SDN είναι μια νέα ανερχόμενη αρχιτεκτονική δικτύων υπολογιστών. Σε ένα δίκτυο υπολογιστών έχουμε τις έννοιες του data plane και του control plane. Σήμερα, η διεπαφή μεταξύ του control plane και του data plane είναι κλειστή και βρίσκεται στο εσωτερικό δρομολογητών και μεταγωγέων με αποτέλεσμα να μην μπορεί κάποιος να αλλάξει εύκολα τα πρωτόκολλα δρομολόγησης που χρησιμοποιούνται σε ένα δίκτυο υπολογιστών. Η βασική ιδέα της αρχιτεκτονικής SDN είναι η αποσύνδεση του control plane από το data plane και η δημιουργία μιας ανοιχτής διεπαφής μεταξύ τους. Το control plane τρέχει εξωτερικά από τους δρομολογητές πάνω από ένα λεγόμενο network operating system (NOS), το οποίο διαχειρίζεται τους πίνακες προώθησης των δρομολογητών και μεταγωγέων ενός δικτύου. Με αυτή την προσέγγιση γίνεται πολύ πιο εύκολο να εφαρμόσει κανείς καινοτόμες τεχνικές δρομολόγησης και διαχείρισης της κίνησης / κυκλοφορίας . Επίσης προσφέρουνε απλούστερη διαχείρηση του δικτύου καθώς και καλύτερη χρήση των πόρων και των συσκευών του δικτύου. Παρά τις ευκολίες υπάρχουν και πολλοί περιορισμοί ένας εκ των οποίων και πολύ σημαντικός είναι η ενεργοβόρα και υψηλού κόστους μνήμη που φέρουν οι συσκευές ενός SDN. Γι αυτό και η μνήμη των συσκευών ενός SDN είναι πολύ περιορισμένη. Για να αντιμετωπιστεί αυτή η δυσκολία αναπτύσσονται νέες πιο έξυπνες και πιο αποδοτικές τεχνικές/αλγόριθμοι χρήσης της μνήμης. Σκοπός αυτής της διπλωματικής εργασίας είναι να υλοποιήσει και να παρουσιάσει μια υπηρεσία-μηχανισμό που θα στοχεύει στην αποδοτικότερη χρήση της μνήμης αυτών των συσκευών.

# Abstract

Software Defined Networking (SDN) is an emerging networking paradigm that separates the network control plane from the data forwarding plane with the promise to dramatically improve network resource utilization, simplify network management, reduce operating cost, and promote innovation and evolution. Although one of the biggest limitations in OpenFlow-driven SDN is the expensive and power demanding memory. Due to this the network devices and commodity switches suffer from limited amount of memory. To overcome this limitation, sophisticated techniques and memory-efficient algorithms have to be developed. The purpose of this thesis is to implement a traffic engineering service to deal with this problem by focusing on the reduction of flow table size.

*Keywords: SDN, OpenFlow, Traffic Engineering*

# Table of Contents

# List of Figures

# List of Tables & Graphs

# 1 Introduction

Software-defined Networking (SDN) is an emerging networking paradigm with a great potential to foster innovation through programmable networks. SDN has gained a lot of popularity, attention and adaptation from network operators, equipment vendors and over-the-top application service providers. SDN networks are characterized by the separation of the control and data planes wherein a logically centralized controller performs routing decisions on behalf of forwarding elements. This separation of control and data plane exposes the capabilities of network devices and thus it provides great potentials and high flexibility in managing and deploying network services. However, this flexibility comes at the cost of placing significant stress on switch state size because it requires installation of flow rules in a limited capacity switch memory. Increasing the memory size to accommodate flow rules for these large number of flows is not a viable solution since the specific memory is costly and power hungry.

In this thesis we develop a traffic engineering service to overcome the limitations stated above by dynamically multiplexing some flows into fewer whenever this is a viable option.

## 1.1 Organization

The rest of this dissertation is organized as follows. The first part deals with background information. More specifically we present a comprehensive overview of Software Defined Networks. In particular, Chapter 2 presents both traditional and SDN architectures and presents the critical role of the OpenFlow standards in SDN. In addition, it describes the function of OpenFlow controllers and OpenFlow switches and depicts the problem this thesis deals with by analyzing more technical details.

Implementation details provided in Chapter 3. Specifically we analyze the multiplexing technique as well as the frame tagging. We explain the algorithm behind the module and present the tool selection and the equipment we used for the experiments.

The results of our proposed scheme as well as the evaluation of the results are discussed in Chapter 4.

Chapter 5 presents some related works. Finally, we summarize and conclude this dissertation in Chapter 6, and propose some potential directions to follow up this research in the future.

# 2

# Background & Motivation

## 2.1 Software Defined Networks

The Internet as we know it today, shows limitations due to the widespread and rapid expansion, thus limiting the scope for developing and implementing innovations. The "software-defined" networks (Software-Defined Networks) are the future of computer networking. This architecture provides researchers an easier method to test new technologies and protocols. One of the most important things is that SDN can be a main substrate for Cloud Computer networking (Cloud Computing) [1].

To begin with Software-Defined Networking (SDN) concept was first time introduced in 2010 [2] as the new networking paradigm which aims to ease the control and the management of a computer network environment.

SDN can be explained as an architectural principle where the networks control and the management are centralized and decoupled from data plane, thus making the network programmable. Traditionally, the data and the control planes in the Ethernet networking devices (and most of the communication principles) have been tied together. This means, the prevailing operating system and its features with the provided hardware are implemented in a single device. Therefore, network devices, such as switches, routers, firewalls, etc., are built with the intelligence of handling traffic relative to the adjacent devices. This makes the intelligence distributed and scattered in the network. In addition, most of the network devices are Command Line Interface (CLI) based and configuration is done separately per device, making configuration slow and prone to errors. This prevents the networking industry of responding quickly to feature requests or innovate new management abilities.

The data plane has a sensible layering model which is known by the name Open Systems Interconnection model (OSI model) [3]. It is well known model in the networking industries and academies. It is standardized by the International Organization for Standardization (ISO). The OSI model enables network applications and services to isolate the data operations to a single layer and provide interfaces between layers. This has enforced developers to develop and improve the operations without concerning the other layers. This type of layering models are used in many other fields (e.g. operating system) and it has provided simplicity to understand the overall view and the interactions. As a result, we can witness increase in the development and research in these fields. As it seems, similar layering model is essentially needed for networks control and management plane, which was not available. This creates the need to invent a new networking architecture, SDN.

The basic idea of the SDN architecture is the decoupling of the control plane from the data plane and the creation of an open interface between them. The control plane runs outside of the routers over a so-called network operating system (NOS), which manages the forwarding tables of the routers and switches of a network. This approach is much easier to implement innovative routing and traffic management techniques since a new routing protocol can be implemented very quickly, simply by using new software over the NOS, without requiring changes to routers and switches. The SDN architecture has approached much interest from industry in the last 2-3 years, already supported by many companies producing routers and switches, such as Cisco and Juniper, and is already used in some networks, such as Google inter-data-center network [4].

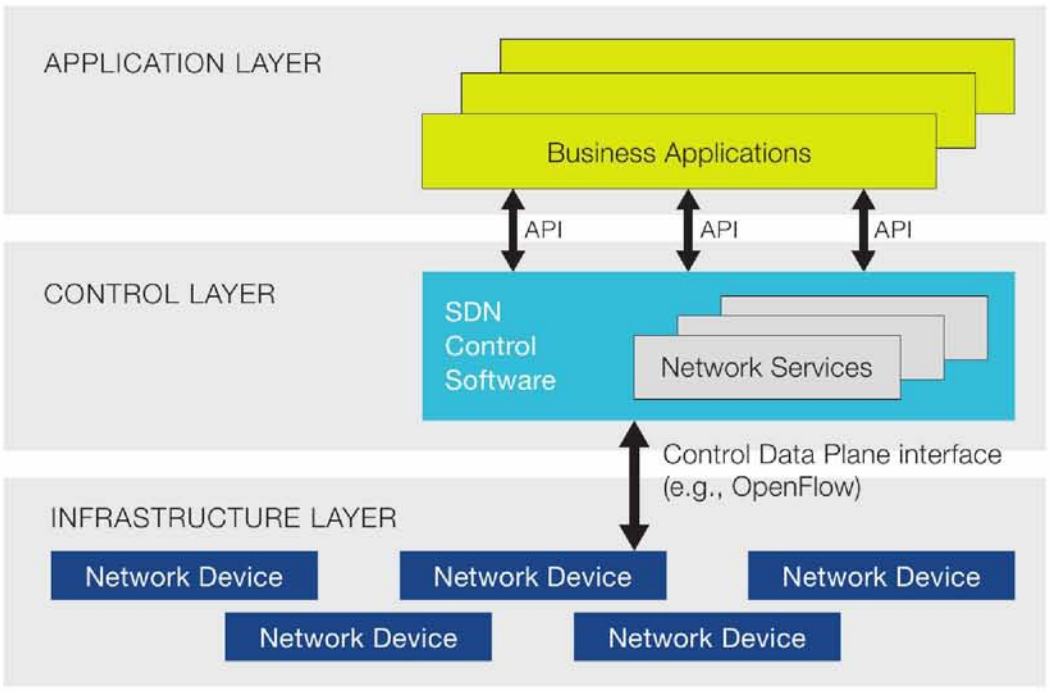


Figure 1: The SDN stack.

As seen from the Figure 1, SDN architecture is divided into three layers: application layer, control layer, and infrastructure layer. This architecture and arrangement of control and management, provides the possibility to centralize the state of the network and the intelligence into one part of the network. This, enhances the property of network programmability, the network industry can start to innovate and enable differentiation in the developing process. Furthermore, programmability accelerates creativity and introduction of new network features and services. With centralization, SDN simplifies provisioning while optimizing performance and granularity of the policy management. Therefore, SDN can make networks become more scalable, flexible and proactive. SDN architecture stack abstracts and decouples hardware from software, control plane from forwarding plane, and physical from logical configuration.

**Infrastructure layer** is the layer where all the hardware exists and are connected physically. On these hardware devices runs a software which provides a control data plane interface (Southbound API) which is used to communicate with the upper level: Control layer.

**Control layer** is the most important layer in the architecture. There is a controller which talks to all the network devices in the infrastructure and keeps track of the topology. While exchanging information of the network state with upper layer applications (through Northbound API), the controller translates their commands to the network devices to have respective and desired network behavior.

**Application layer** is the layer where all the features, services and policies are defined. Applications request the information of network devices and the topology in order to act upon it. These applications can create features end-to-end and make big picture decisions according to the changes in the network. When the network topology, feature, or policy requirements changes, applications have the control to change dynamically the network behavior from one single point.

Between these layers, there are Application Programming Interfaces (APIs) which provide the essential communication tools between the layers. The Northbound API is provided by the controller and the applications have to manage their communication to the controller through it. Many SDN controllers were settling down with REST API [5]. Other popular API's are C++, JAVA and Python. The Southbound API is the communication between the controller and the network devices.

There are a lot of cases that Software Defined Networks can find application, showing the great capability SDN have. Some of them are:

- Network Access Control
- Sets the appropriate privileges for any user or device of the network. It controls how someone accessing the networks, including access control limits, and the incorporation of service chains as well as appropriate quality of service (QoS).

- Network Virtualization
- Virtual network on top of a physical network, allowing a large number of multi-tenant networks to run over a physical network, spanning multiple racks in the datacenter or locations if necessary, including fine-grained controls and isolation as well as insertion of acceleration or security services.

- Service Velocity
  - Virtual edge, Distributed app testing environments, application development workflows.

- Application Enhancement
  - Specific SDN application, reserved bandwidth for application needs, geo-distributed applications, intelligent network responses to application needs.

So, SDN brings new challenges in networking technology and in this thesis the focus is set on traffic engineering. A programmable network provides full control of a network. Thus, bringing more capabilities to handle traffic in the network, whether in a local area network (LAN) or in the core. Similarly, new unexplored challenges can be unveiled or discovered. Nevertheless, SDN has gained rapidly its reputation and is the biggest hype word in networking business.

## 2.2 OpenFlow Protocol

OpenFlow is the first open standard communication interface defined between the control plane and the data plane in order to enable the implementation of a flexible SDN architecture. OpenFlow provides direct access and manipulation of the data plane of virtual or physical network devices, such as switches and routers. This means that OpenFlow is a communication protocol which gives access to the forwarding plane of a network switch or router through the network. This allows network packet forwarding to be defined by software.

OpenFlow began to be developed in 2007, being a collaboration between the commercial and academic worlds. Initially developed by Stanford University and California University in Berkley, the standardization is being conducted by the Open Network Foundation (ONF). ONF is an organization dedicated to the promotion and adoption of SDN through open standards development. OpenFlow is a follow up on previous projects on programmable networks, namely Ethane[6] , GENI [7] and SANE [8] which were one of the first projects to decouple control and data plane. OpenFlow shortly started to become more popular and as an open standard, it developed quickly to support more and more functionalities. This project has been developed using the latest version of OpenFlow the 1.3.

### 2.2.1 OpenFlow Architecture

As stated before, OpenFlow is based on the separation between the data plane and the control plane and executes a flow-based control.

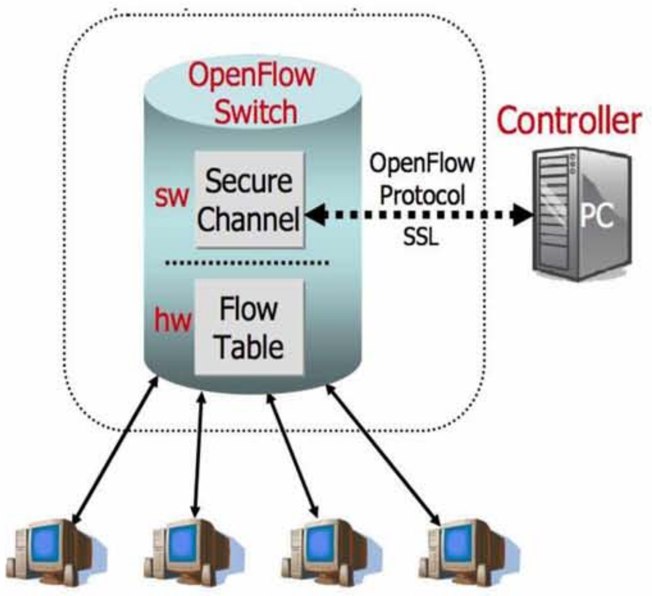
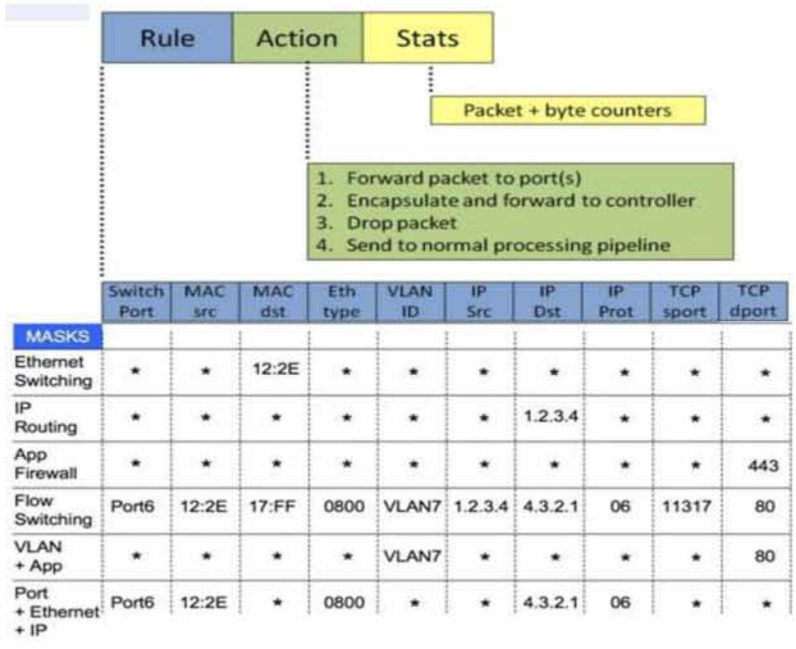


Figure 2: The Openflow Architecture

This flow is defined by the information contained in the packet, from layer 1 to layer 4.OpenFlow defines the messaging protocol and also the semantics for changing switch states. OpenFlow networks consist of an OpenFlow Controller, OpenFlow switches (devices) and the OpenFlow Protocol, as shown in figure 2.

Communication between the switch and the Controller is done through a secure, Transport Layer Security (TLS) / Secure Sockets Layer (SSL) based, channel. Both the Controller and the switch interface implement the OpenFlow Protocol [9].

Packet forwarding is executed in the OpenFlow switch based on the flow table entries, where forwarding and routing decisions are defined the Controller. When a switch receives a packet that does not have a matching flow table entry, it sends the packet to the Controller. The Controller can then dispose of the packet or add the packet to an entry in the switch flow table [9].


Rule
Action
Stats
Packet + byte counters
1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline

| MASKS | Switch Port | MAC src | MAC dst | Eth type | VLAN ID | IP Src | IP Dst | IP Prot | TCP sport | TCP dport |
|---|---|---|---|---|---|---|---|---|---|---|
| Ethernet Switching | * | * | 12:2E | * | * | * | * | * | * | * |
| IP Routing | * | * | * | * | * | * | 1.2.3.4 | * | * | * |
| App Firewall | * | * | * | * | * | * | * | * | * | 443 |
| Flow Switching | Port6 | 12:2E | 17:FF | 0800 | VLAN7 | 1.2.3.4 | 4.3.2.1 | 06 | 11317 | 80 |
| VLAN + App | * | * | * | * | VLAN7 | * | * | * | * | 80 |
| Port + Ethernet + IP | Port6 | 12:2E | * | 0800 | * | * | 4.3.2.1 | 06 | * | * |

Figure 3: Flow table entry

As seen in figure 3 each entry in the flow table of an OpenFlow switch is divided in three parts: Rule, Action and Statistics.

- Rule: a header to match with the frames of the flows. There are several supported Ethernet headers in OpenFlow specification [9] but as OpenFlow is made to be extensible, custom headers can be additionally defined. The switch merely performs a bit mask match. Therefore, OpenFlow switch is open for innovative non-IP traffic.

- Action: as a rule is matched with traffic, the action which should be performed for it has to be defined. The actions are also open for extensions, but some basic actions are already provided in the specification. Such as, forwarding to one or more ports, forward to the controller, drop the frame, and modify frame fields. The only requirement for adding customized actions is that the data path must have flexibility while providing high performance and low cost.

- Statistics: Each and every time when a flow rule is matched, the switch has to update the frame counters, which indicates the popularity of a specific flow. There are counters for every table, each flow, all the ports and every queue. Also a timer of last activity and initial set of the flow are maintained.

### 2.2.2 Controller

OpenFlow Controller an independent software application running in a dedicated server which is responsible for managing OpenFlow switches. In other words, this Controller is responsible for everything happening in the network. The Controller can add, remove or update the flow table entries statically or dynamically, using the OpenFlow Protocol.

As Figure 4 shows SDN Controllers can simplify network management, handling all communications between applications and devices to effectively manage and modify network flows to meet changing needs. When the network control plane is implemented in software, rather than firmware, administrators can manage network traffic more dynamically and at a more granular level. An SDN Controller relays information to the switches/routers (via southbound APIs) and the applications and business logic (via northbound APIs). In particular, OpenFlow Controllers create a central control point to oversee a variety of OpenFlow-enabled network components. The OpenFlow protocol is designed to increase flexibility by eliminating proprietary protocols from hardware vendors.

Flow tables are a database that stores all flow entries associated with an action, so the switch can apply that action to a certain flow.

Every functions of the control plane and management are executed by the Controller. The Controller configures every device, maintains topology information and monitors the state of the whole network.

The OpenFlow Controller can have a reactive behavior or a proactive one.

- Reactive flow instantiation – When a new flow comes into the switch, the OpenFlow agent SW on the switch, does a lookup in the flow tables, either in a search ASIC if in hardware or a software flow table in the case of a vSwitch. If no match for the flow is found, the switch creates an OFP packet-in packet and fires it off to the controller for instructions. Reactive mode reacts to traffic, consults the OpenFlow controller and creates a rule in the flow table based on the instruction. The problem with reactive is today's hardware has laughable amounts of CPU in it.

- *Proactive flow instantiation* – Rather than reacting to a packet, an OpenFlow controller could populate the flow tables ahead of time for all traffic matches that could come into the switch. Think of a typical routing table today. You have longest prefix matching that will match the most granular route to a destination prefix in a prefix tree lookup. By pre-defining all of your flows and actions ahead of time in the switches flow tables, the packet-in event never occurs. The result is all packets are forwarded at line rate, if the flow table is in TCAM, by merely doing a flow lookup in the switches flow tables. That is the same hardware that populates its forwarding tables today from "routing by rumor" in today's routing protocols and "flood and spray" layer2 learning standards. Proactive OpenFlow flow tables eliminates any latency induced by consulting a controller on every flow.

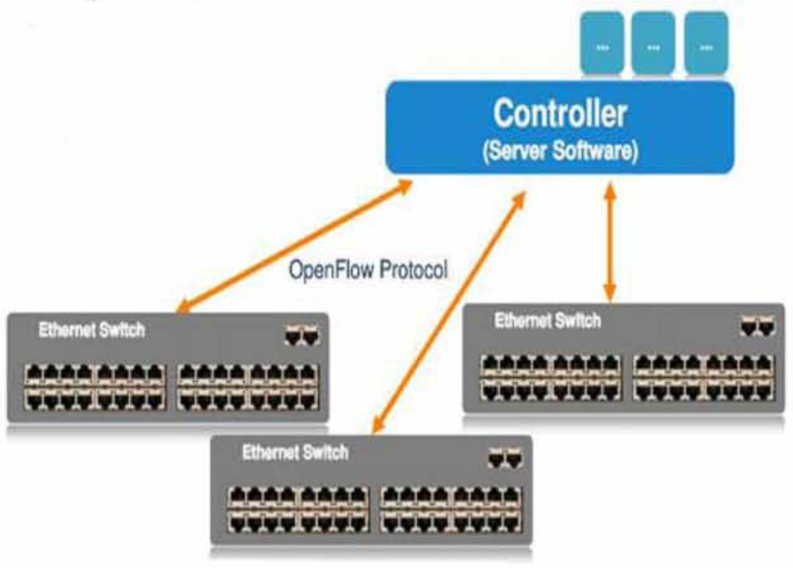


Figure 4: SDN Controller

There are different types of OpenFlow Controller software, each one has its own identity. Besides software performance, the programming language that each software implements is different.

- **NOX** [10] - is an open-source development platform for application control in SDN networks based in C++. It provides an OpenFlow 1.0 API and a fast and asynchronous I/O. NOX's primary targets are Linux distributions, it supports multithreading and the platform includes examples, such as Topology discovery, learning switch and network-wide switch.
- **POX** [11] - consists in a NOX implementation written in Python, allowing rapid development and prototyping of network control software components. POX has as characteristics, the reuse of component samples from a selected path, topology discovery, runs in any operating system and supports the same visualization tools as NOX. It

can be said that POX is a good platform for people starting out in programmable networks, it is also a good platform for research, academic applications and network prototyping. While NOX is good for configuring big system networks, or for when Controller performance has to be fast. A disadvantage of POX is it does not support multithreading, it is used to explore and distribute prototypes, to run SDN, for virtualizing the network, designing Controllers and programming models.

- **Floodlight** [12] - is enterprise-class and apache-licensed, based in Java, but for people who do not like writing in Java, these people have the option of programming in Jython. Floodlight derived from Beacon, originally developed by David Erickson, it is now supported by the programmer team, including engineers, of Big Switch Networks. It supports multithreading, was developed to work with an increasing number of network devices that support OpenFlow and deals with a mix of OpenFlow and non-OpenFlow networks - able to manage multiple islands of OpenFlow hardware switches.

- **ONOS** [13] - was developed in 2014, The ONOS (Open Network Operating System) project is an open source community hosted by The Linux Foundation. The goal of the project is to create a software-defined networking (SDN) operating system for communications service providers that is designed for scalability, high performance and high availability.

- **Trema** [14] - is a framework that includes everything necessary to create an OpenFlow Controller, it was developed by NEC and is based on the Ruby platform or C. This platform was only tested in a UNIX environment, only supporting GNU/Linux and version 1.0 of OpenFlow, plus the latest Ruby version still does not support the OpenFlow Controller library. The Trema framework can emulate an OpenFlow based network and end-hosts and provides tests for the Controller. Contains a plugin for Wireshark, allowing it to monitor data-flows through functional modules.

- **OpenDaylight** (ODL) [15] is an open-source SDN controller project maintained as part of the Linux foundation. The ODL project was started in 2013 and is widely supported by industry members and researchers, with the goal to make SDN more transparent and to act as basis for Network Function Virtualization (NFV). The controller is written in Java with the main development decisions voted on by an elected Technical Steering Committee. One of the main non-commercial use cases for ODL is providing network services for the OpenStack cloud platform. It is a highly available, modular, extensible, scalable and multi-protocol controller infrastructure built for SDN deployments on modern heterogeneous multi-vendor networks. OpenDaylight provides a model-driven service abstraction platform that allows users to write apps that easily work across a wide variety of hardware and south-bound protocols.

### 2.2.3 OpenFlow Switch

The OpenFlow switch is basically an Ethernet switch that supports the OpenFlow Protocol. OpenFlow is based on switching devices with one or more flow tables, a group table and an OpenFlow Channel to an external Controller, that is, a standard interface to add or remove flow entries, as can be seen in Figure 5.

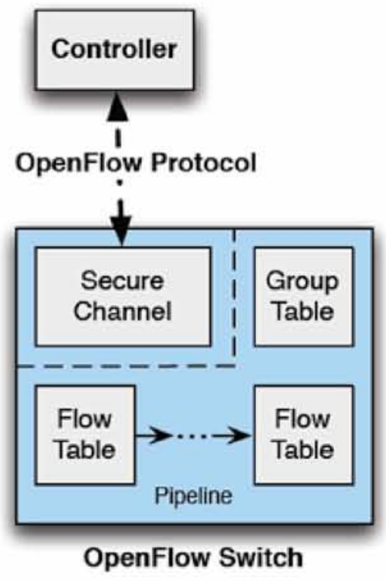


Figure 5: The OpenFlow Switch

Each device maintains a flow table that contains a set of flow entries. Each consists of match fields, counters and a set of instructions to apply on the matching packets.

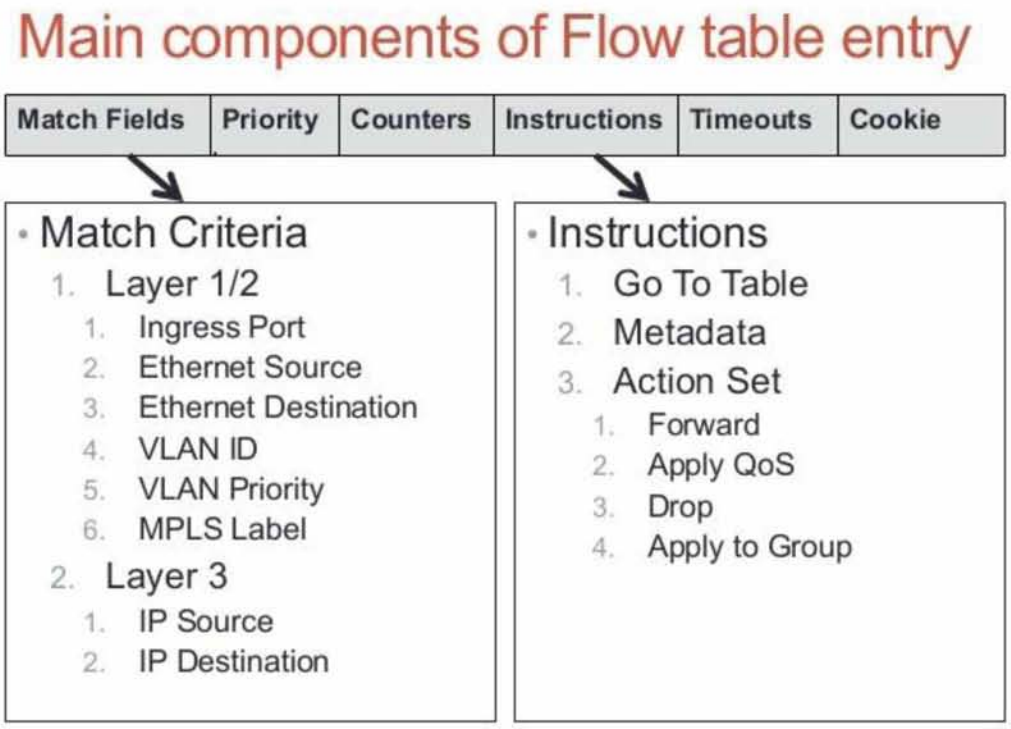


Figure 6: Components of a flow entry in a flow table

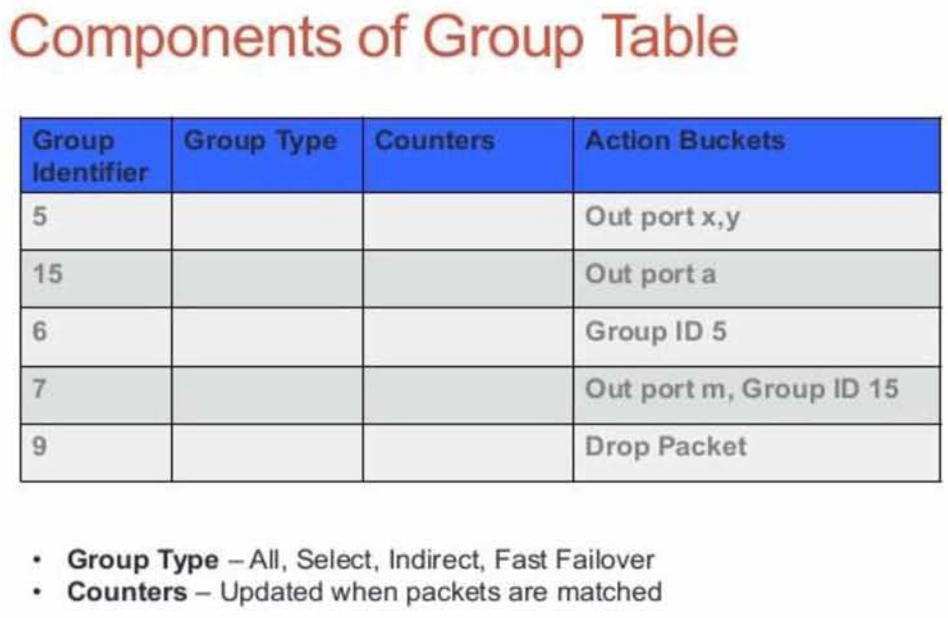
As can be seen in Figure 6 each flow entry contains:

- ***Match fields***: to match against packets. These consist of the ingress port and packet headers, and optionally metadata specified by a previous table.
- ***Priority*** **:** matching precedence of the flow entry.
- ***Counters*** **:** updated when packets are matched.
- ***Instructions*** **:** to modify the action set or pipeline processing.
- ***Timeouts*** **:** maximum amount of time or idle time before flow is expired by the switch.
- ***Cookie*** **:** opaque data value chosen by the Controller. May be used by the Controller to filter flow statistics, flow modification

and flow deletion. Not used when processing packets.

The flow table entry is identified by the Match Field and Data Priority, these two fields identify a single flow entry in the flow table.

The group table consists of group entries. The ability of a flow entry to point to a group allows the representation of additional forwarding methods. As can be seen by figure 7 each entry group is identified by four fields.



## Components of Group Table

| Group Identifier | Group Type | Counters | Action Buckets |
|---|---|---|---|
| 5 | | | Out port x,y |
| 15 | | | Out port a |
| 6 | | | Group ID 5 |
| 7 | | | Out port m, Group ID 15 |
| 9 | | | Drop Packet |

- **Group Type** – All, Select, Indirect, Fast Failover
- **Counters** – Updated when packets are matched

Figure 7: Components of a group entry in the group table

Each group entry consists of:

- ***Group Identifier***: a 32 bit unsigned integer uniquely identifying the group.

- ***Group Type***: to determine group semantics, meaning that a switch does not need to support every group type, it only needs to support those marked as "Required" the other group types the switch may support are "Optional".

  "Required" groups have two types:

  - ❖ **all**: This executes all buckets in a group, with this group being used for broadcast or multicast forwarding, in other words, the packet is cloned for each bucket, then processed by each bucket in the group.

  - ❖ **indirect**: Executes a bucket defined in a determined group. This group only supports one bucket.

  *"Optional"* groups also have two types:

  - ❖ **select:** Executes a bucket in a group. The packets are processed by a single bucket in the group, based on switch-computed selection algorithm.

  - ❖ **fast failover:** Executes the first bucket in real-time. Each bucket action is associated to a specific port and/or to a group that controls this liveness.

- **Counters:** updated when packets are processed by a group.

- **Action Buckets:** an ordered list of action buckets, where each action bucket contains a set of actions to execute and associated parameters.

The **OpenFlow Pipeline process** defines how the packet interacts with flow tables (Figure 8). For this procedure, the device has to have at least one flow table.

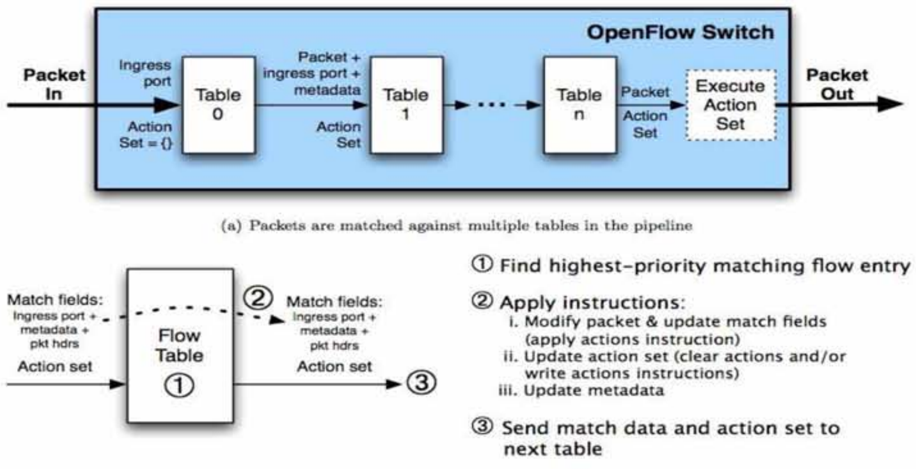

OpenFlow Switch
Packet In
Ingress port
Action Set = {}
Table 0
Packet + ingress port + metadata
Action Set
Table 1
Table n
Packet
Action Set
Execute Action Set
Packet Out
(a) Packets are matched against multiple tables in the pipeline
Match fields: Ingress port + metadata + pkt hdrs
Action set
Flow Table ①
② Match fields: Ingress port + metadata + pkt hdrs
Action set
③
① Find highest-priority matching flow entry
② Apply instructions:
i. Modify packet & update match fields (apply actions instruction)
ii. Update action set (clear actions and/or write actions instructions)
iii. Update metadata
③ Send match data and action set to next table

Figure 8: Flow table processing

Flow tables are sequentially numbered, starting at 0. Processing is always initiated at flow table 0. When a switch receives a packet, the packet's match field is compared with the flow entry match field. The packet may match more than one entry of a flow table. In this case, the chosen flow entry is the one with the highest priority.

When the packet matches a flow entry, the flow table executes the instructions stored in the corresponding flow entry, these instructions may be to send the packet directly to another flow table (Goto instruction), the packet's header, metadata, packet/ match set fields and action set are updated and it is then sent to the flow table indicated by the Goto instruction and the process repeats successively. If the flow entry does not have a Goto instruction, then the pipeline processing terminates and the packet is processed according to the associated actions

When the packet has no match with any flow entry of the flow table, the packet is then disposed if the flow table has no table-miss flow entry. If not the flow table has a table-miss flow entry, then the packet is processed according to the table-miss configurations, it can be disposed using Clear-Actions and sent to the Controller (via packet-in message) using the Controller reserved port. The table-miss processes the non-existing tables, meaning that it specifies how the packet is processed when it has no match with the flow entries. The flow entries are removed if specified by the Controller, or by the switch flow expiry mechanism. This mechanism is based on the state and configuration of the flow entry.

To remove a flow entry from the flow table the Controller sends a delete flow entry message for the corresponding flow table (OFPFC DELETE or OFPFC DELETE STRICT) [9].

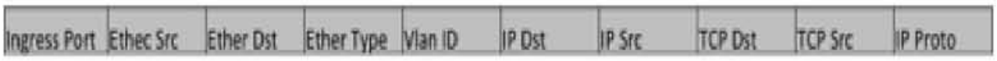
To remove a flow entry by the flow expiry mechanism, each flow entry contains an idle-timeout and a hard-timeout. If the idle-timeout is greater than zero, it means that the switch registers the arrival time of the last match packet, and if in the time specified by the idle-timeout no packet is associated to this flow entry, it is removed.  If the hard-timeout is greater than zero, the switch registers the arrival time of the flow entry and removes it after the specified time, regardless if the flow entry has a lot of packet matching to it [9].

OpenFlow switches are divided in two categories, Commercial switches and Software switches.

Commercial switches are physical switches that come with hardware that supports OpenFlow. Several vendors offer commercial switches that support OpenFlow, such as HP [16] ,Pica8 [17] and NEC [18] .

Software switches are software that supports OpenFlow and can be installed in general purpose hardware. There are also several software switches available, such as Open-WRT [19] and OpenVSwitch [20].

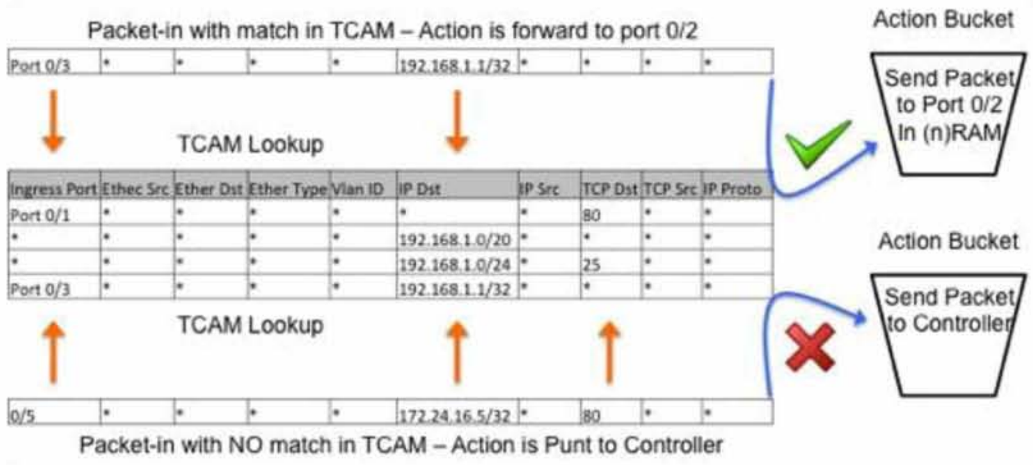
## 2.3 The Problem



| Ingress Port | Ethec Src | Ether Dst | Ether Type | Vlan ID | IP Dst | IP Src | TCP Dst | TCP Src | IP Proto |
|---|---|---|---|---|---|---|---|---|---|

Figure 9: the 12-tuple of fields of every packet used for flow matching

OpenFlow switches in any OpenFlow network consist of Ternary Content Addressable Memory (TCAM) memory. TCAM memory is a special type of memory that enables matching on the headers of received data packets <u>during one clock cycle, regardless of the number of entries in memory</u>. As stated before an OpenFlow network is composed of a controller and a group of OpenFlow switches. The controller and the OpenFlow switches communicate through a control plane on the network in order to maintain flow entries in the switches. The OpenFlow switches transfer data packets on the data plane of the network based on the flow entries. A flow entry includes definitions of the flows that are referred to as the matching fields. Figure 9 shows the matching fields include the ingress port number and the header fields from layers 2-4 that are specified in the OpenFlow switch specification [9]. Wildcards are allowed for any of the matching fields. An OpenFlow switch searches the flow entries that needed to be matched in the header fields of each data packet that is received (Figure 10). TCAM can have multiple matches and determine a best match. TCAM enables searching during on clock cycle, regardless of the number of the entries in the TCAM and regardless of whether wildcards are included or not included in the matching fields.


Packet-in with match in TCAM – Action is forward to port 0/2
Action Bucket
Send Packet to Port 0/2 In (n)RAM
TCAM Lookup
Action Bucket
Send Packet to Controller
TCAM Lookup
Packet-in with NO match in TCAM – Action is Punt to Controller

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| Port 0/3 | * | * | * | * | 192.168.1.1/32 | * | * | * | * |

| Ingress Port | Ethec Src | Ether Dst | Ether Type | Vlan ID | IP Dst | IP Src | TCP Dst | TCP Src | IP Proto |
|---|---|---|---|---|---|---|---|---|---|
| Port 0/1 | * | * | * | * | * | * | 80 | * | * |
| * | * | * | * | * | 192.168.1.0/20 | * | * | * | * |
| * | * | * | * | * | 192.168.1.0/24 | * | 25 | * | * |
| Port 0/3 | * | * | * | * | 192.168.1.1/32 | * | * | * | * |

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 0/5 | * | * | * | * | 172.24.16.5/32 | * | 80 | * | * |

Figure 10: Ternary CAM OpenFlow operation example

In an OpenFlow switch, the required TCAM space is large due to the wide range of header fields that are supported in OpenFlow (Figure 9.The problem with TCAM is that it is power hungry, expensive and takes up quite a bit of silicon space. It is the most expensive component on commodity switches. It has been noted that TCAM is up to 80 times more expensive than static random access memory (SRAM) [21] . Many vendors use a blend of BCAM memory, SRAM, NPUs and software algorithms to perform ternary lookups in order to avoid the usage of the extra-expensive TCAM.

Another big obstacle is that TCAM-based tables were limited to just a few thousand entries. For example the HP5406zl OpenFlow-enabled switch supports approximately 1500 OpenFlow rules [22] due to the TCAM limitations and the NEC PF5820 is capped to only 750 entries [23].

If the table was exceeded, the packets would be handled by the switch's software .This severely limited the scalability and performance of early OpenFlow switches.

Many techniques have been proposed in the research community for reducing the number of OpenFlow rules in the OpenFlow tables by investigating various configurations and characteristics of possible hardware implementations for the flow tables (e.g. types of memory) [24]. Other techniques focusing on decreasing the frequency of the usage of TCAM by implementing smarter memory accessing algorithms or restricting the packet matching on less fields [25].

# 3 Implementation

In this chapter we will discuss and implement a mechanism who will overcome the limitations TCAM introduced. As stated in the previous chapter the most common underlying technology for implementing OpenFlow flow tables is TCAM ,because it allows for fast implementation of matching rules that can support flexible Wildcarding, for any of the packet headers. Since this type of memory is expensive and it consumes significant amounts of power, it becomes costly to build and deploy forwarding devices that can store a large number of OpenFlow rules. This further limits the adoption of OF-based SDN for environments where there is a large volume of traffic flows that must be processed (e.g. forwarded, firewalled, etc.)
In this thesis we develop a service that reduces the number of installed entries in the flow tables of OpenFlow network devices based on dynamically applying aggregation of traffic flows at various points in the network (flow aggregation).

## 3.1 Flow Multiplexing

An aggregate of traffic flows is a collection of traffic flows that are grouped together for common treatment between two points in a network. Packets from all flows in a trunk travel the same path and are subject to the same traffic management policies. The concept of aggregation brings the Layer-2 values of ATM into the layer-3 network. An identifier is needed in order for the controller to be able to manage the aggregate as a single entity. The traffic flows that are to be treated as a single aggregate will then be marked with the identifier and processed accordingly in the network. These aggregates can follow determined paths and be given a consistent quality of service (QoS) treatment. A labelled path can exist between any two points in the network and multiple paths can themselves be aggregated within another label. This hierarchical aggregation simplifies the management of network resource and facilitates engineering of QoS.

Several identification mechanisms are possible such as Multi-Protocol Label Switching (MPLS) labeling, VLAN tagging, and other encapsulation formats. These mechanisms are well known from traditional network architectures and they are well suited also for the aggregation service proposed in this paper.

For real life deployments the choice of the identifier is very important and it usually depends on several aspects such as support from hardware, encapsulation overhead, number of available identifiers, etc.

To encapsulate, identify the aggregates we had to choose between VLAN tagging and MPLS techniques. The choice was VLAN tagging and the reasons are described below.

## 3.2 IEEE 802.1q

The 802.1q standard was created by the IEEE group to address the problem breaking large networks into smaller and manageable ones through the use of VLANs. The 802.1q standard is of course an alternative to Cisco's ISL (inter switch link), and one that all vendors implement on their network equipment to ensure compatibility and seamless integration with the existing network infrastructure [26].

As with all 'open standards' the IEEE 802.1q tagging method is by far the most popular, easy to use and commonly used even in Cisco oriented network installations mainly for compatibility with other equipment and future upgrades that might tend towards different vendors.

In addition to the compatibility issue, there are several more reasons for someone to prefer this method of tagging. These include:

1. **Simple to implement**
2. ***Support of up to 4096 VLANs***
3. **Insertion of a 4-byte VLAN tag with no encapsulation**

Amazingly enough, the 802.1q tagging method supports a whopping 4096 VLANs (as opposed to 1000 VLANs ISL supports), a large amount which is merely impossible to deplete in a local area network or software defined network. And the most amazing aspect of this method is its simplicity in development.

The 4-byte tag we mentioned is inserted within the existing Ethernet frame, right after the Source MAC Address as illustrated in the diagram below:

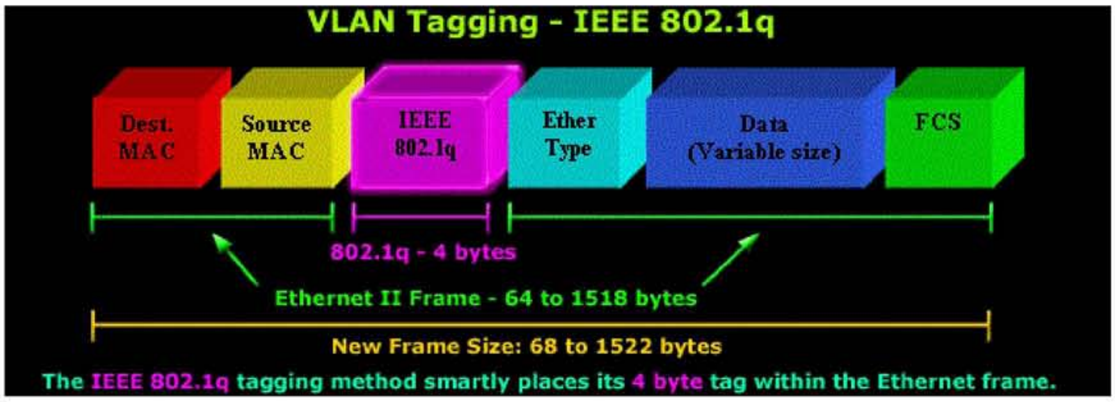

VLAN Tagging - IEEE 802.1q
Dest. MAC
Source MAC
IEEE 802.1q
Ether Type
Data (Variable size)
FCS
802.1q - 4 bytes
Ethernet II Frame - 64 to 1518 bytes
New Frame Size: 68 to 1522 bytes
The IEEE 802.1q tagging method smartly places its 4 byte tag within the Ethernet frame.

Figure 11: The VLAN frame

## 3.3 Structure of the Mechanism

The idea behind this service is simple. The controller is a pack of applications modules. As figure 12 depicts the core functions of the controller are:

- device and topology discovery and tracking,
- flow management
- device management
- Statistics tracking.

These are all implemented by a set of modules internal to the controller and they comprise the core of the controller. As shown in Figure 12 these modules need to maintain local databases containing the current topology and statistics. The controller tracks the topology by learning of the existence of switches (SDN devices) and end-user devices and tracking the connectivity between them. It maintains a flow cache that mirrors the flow tables on the various switches it controls. The controller locally maintains per-flow statistics that it has gathered from its switches.

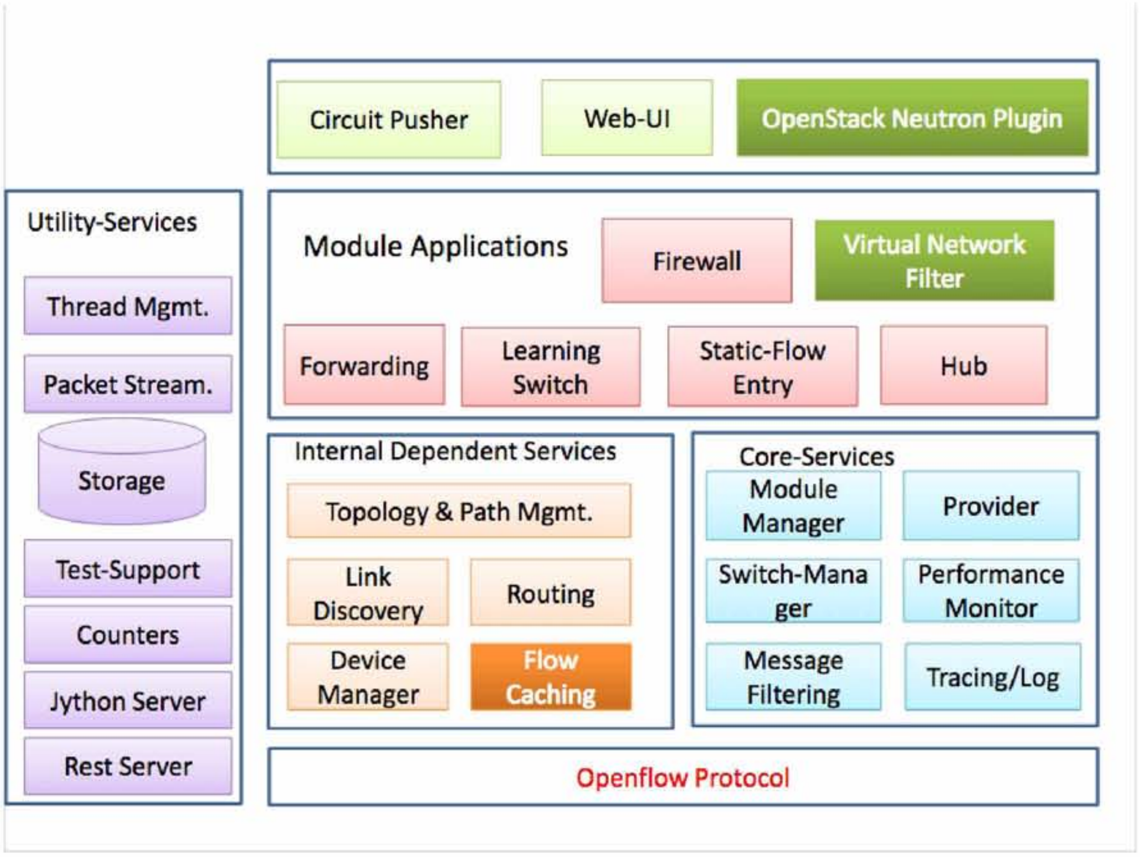


Figure 12: SDN Controller Anatomy

The core modules are like a chain and each module defines which one will follow up. Whenever an action triggers the controller, this action is being processed by each module one by one in order to be handled properly.

The aggregation mechanism is a standalone module for the controller and it has been inserted in that chain described above just before the forwarding module. Whenever an event (OpenFlow message) reaches the controller the module run whether to decide if traffic aggregation is a viable solution or just to do the normal forwarding of the packet. The controller also utilizes the built-in module Topology Manager, which allows for an up-to-date and global view of the network's state. After processing the packet, the controller sends back to the OpenFlow switch the required instructions on how the packet should be processed, by using OpenFlow. Also the controller sends instructions to all the OF switches that are part of the packet's route through the network.

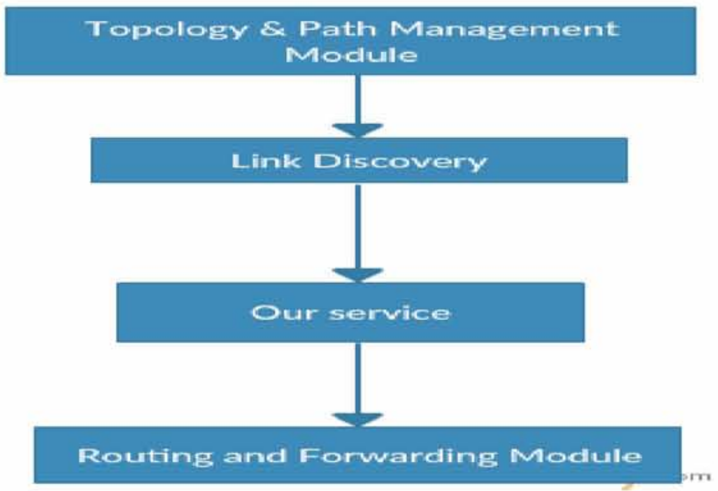

Topology & Path Management Module
Link Discovery
Our service
Routing and Forwarding Module

Figure 13: Mechanism Structure

## 3.4 Algorithm Analysis

As stated before whenever an OpenFlow message is sent to the controller by one of OpenFlow switches, the mechanism is triggered. When the packet arrives into the controller the mechanism extracts the route of the packet, meaning the path the packet will follow. The first step is to check whether or not the packet is an IP packet. If is not an IP packet the algorithm terminates and the controller continues with its standard procedure in order to treat the incoming packet correctly. By examining the source and destination IP of the packet, the controller calculates the route that will follow. The route of each packet is registered and when a new packet arrives, controller compares the new route with all the previous registries (routes). If common route or common path is found then aggregation is a possible option and so on the corresponding flow multiplexes with the existing flows of the common route/path. The new multiplexed flow is tagged with a VLAN id and gets installed on the involved switches/devices replacing the previous ones. If common route or path cannot be found then the algorithm stops and the controller proceeds with its standard procedure. The aggregation process is recursive this means that after every flow merging the new multiplexed flow is compared with the previous registries also. This secures that if common path exists the procedure must be repeated.

## 3.5 Tool Selection

### 3.5.1 SDN Controller Selection

Table 1 below summarizes the different properties of the different OpenFlow Controllers. It can be seen that Trema lacks of documentation which makes the development using Trema a very hard task. POX does not support multithreading which is a huge drawback because imagine all the packets arrive in the controller to be handled sequentially as well as every packets route-matching. Consequently it has the weakest performance. Through table analysis, it can be seen that NOX is a good choice although it has the disadvantage of being written in C++ a programming language which I am not familiar with. Beacon is rejected because the documentation is poor and is outdated. The top 3 nominees are Floodlight, OpenDayLight and ONOS which have a lot in common but I select Floodlight due to the very good documentation.

| | ODL | ONOS | NOX | POX | Beacon | Floodlight | Trema |
|---|---|---|---|---|---|---|---|
| Programming Language | Java | Java | C++ | Python | Java | Java | C or Ruby |
| Compatibility | Linux distributions | Linux distributions | Linux distributions | Linux, Mac OS and Windows | All Platforms, from high end multi-core Linux servers to Android phones | Linux, Mac OS and Windows | Linux distributions |
| Documentation | Poor | Medium | Good | Medium | Medium | Very Good | Poor |
| License | Eclipse Public License 1.0 | Apache 2.0 License | OpenFlow v1.3 license | OpenFlow v1.3 license | GPL v2 license and FOSS License Exception v1.0 | Apache 2.0 License | GPL v2 license, the last version of the Ruby don't support the OpenFlow libraries |
| Open Source | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Multithread | Yes | Yes | Yes | No | No | Yes | Yes |
| GUI | Build in Gui(DLUX) | Web Gui | NoX Gui (Only for monitoring) | Pox Des (only for monitoring) | Web UI | Avior, Build in Gui | No |

Table 1: OpenFlow Controllers features.

### 3.5.2 Monitoring Tools

Network Monitoring has been an integral part of operating a network. It not only provides the visibility into how well the network performs, but also is an important tool when it comes to troubleshoot problems. Given the distributed nature of switching or routing devices in the network, a typical network monitoring tool usually queries a set of devices via some network management protocol, such as SNMP, and correlates results into reports or graphs.

sFlow [27] and OpenFlow provide complementary functions that together offer exciting opportunities for delivering breakthrough data center and cloud networking performance. The OpenFlow protocol allows controller software running on a server to configure the hardware forwarding tables in a network of switches. The sFlow standard specifies instrumentation in the forwarding table hardware that provides real-time, network-wide visibility into traffic flowing across the network. In addition, sFlow also provides real-time visibility into the performance of servers. Combined, sFlow and OpenFlow can be used to construct feedback control systems that optimize performance, automatically adapting the network to meet changing demands.

For the reasons described above sFlow is the tool I use for monitoring the flows and latencies in my experiments.

### 3.5.3 Network Simulators

To deploy a complete test bed containing multiple networked computers, routers and data link to conduct the experiments we need network simulators. Table 2 shows the comparison of two very popular network simulators. It is obvious that Mininet wins the battle in almost every category.

| | Mininet | ns-3 |
|---|---|---|
| Compatibility with real Controllers | Yes | No |
| OpenFlow Specification | all versions | 0.8.9 |
| Mode | Emulation | Simulation |
| Scalability | High (by Multiple processes) | High (by single process) |
| Performance and Result Correctness | Good Performance | No STP |
| Documentation | Yes | Poor |
| GUI Support | Yes, Observation only | Yes, Observation Only |

Table 2: comparison of Mininet, ns-3.

## 3.6 Experiments and Equipment

In order to assess the performance of this thesis mechanism an experiment must be set up. The selected controller (Floodlight), with and without the aggregation module loaded, run in a set of topologies in order to compare and evaluate the results. In order to have deterministic and low-cost environments to test, a virtual testbed was created that can run on a single computer and does not require additional effort to be maintained and operated. The selected network simulator is Mininet. The performance metric we monitor is the total number of OpenFlow flow entries as well as the latency on every network device involved.

### 3.6.1 The Equipment

For the experiment the Floodlight Controller was installed on a PC with the following specifications:

- Cpu : Core i5-4440S 2.8GHz [28]
- Ram : 8 Giga, 1666MHz
- Hdd : 1 TB, 7200 Rpm
- Nic : 2 x Intel PRO/1000 PT dual port 1 Gbps PCI-Express [29]
- OS: Linux Ubuntu Server 16.10

Mininet was installed on a laptop with the following specifications:

- Cpu : Intel Core i3 7100U 2.4 GHz [30]
- Ram : 8 Giga, 1666MHz
- Hdd : 128 GB SSD
- Nic : Broadcom 802.11n Network Adapter [31]
- OS: Linux Ubuntu 16.04 LTS

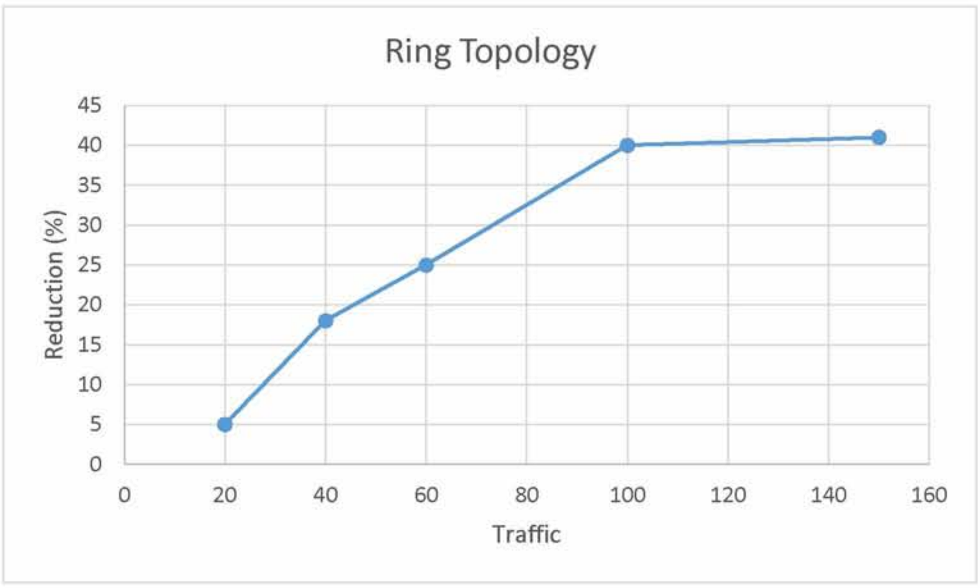
The two devices were connected on the same LAN .The controller and the network simulator were separated in order for the experiment/simulation to be closer to a real world network. All the measurements was given by the chosen tool sFlow and for the sake of validity the experiments was repeated many times.

# 4

# Results

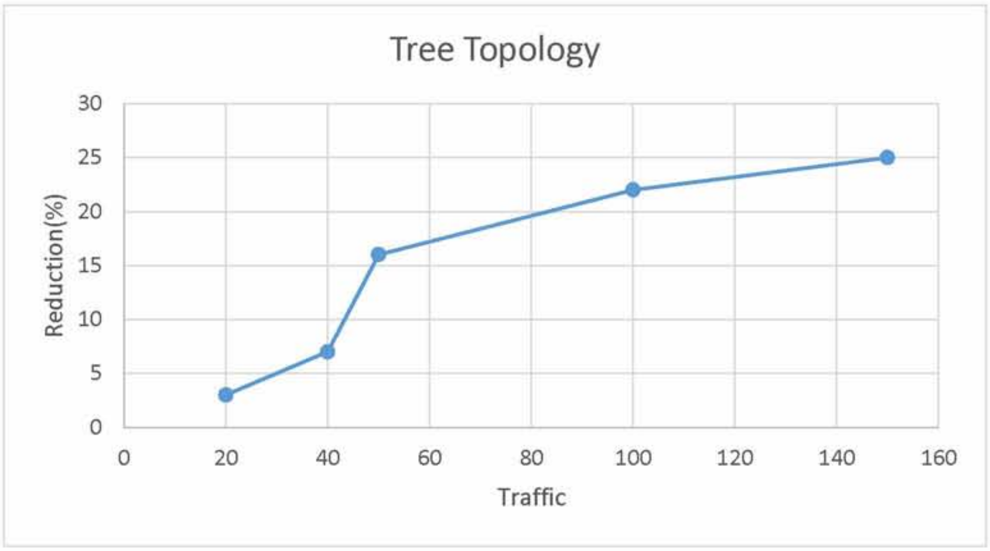
This chapter presents the results that allow a short evaluation on SDN traffic comparing the results taken from network topologies with the traditional Floodlight controller and with the enhanced version of Floodlight. In order to find a correlation between the number of flows, latency and the percentage of reduction experiments must be done in a set of topologies.

The following graph shows the reduction in flows in a ring topology. The percentage of reduction is significant as network traffic escalates (traffic measured in number of flows).



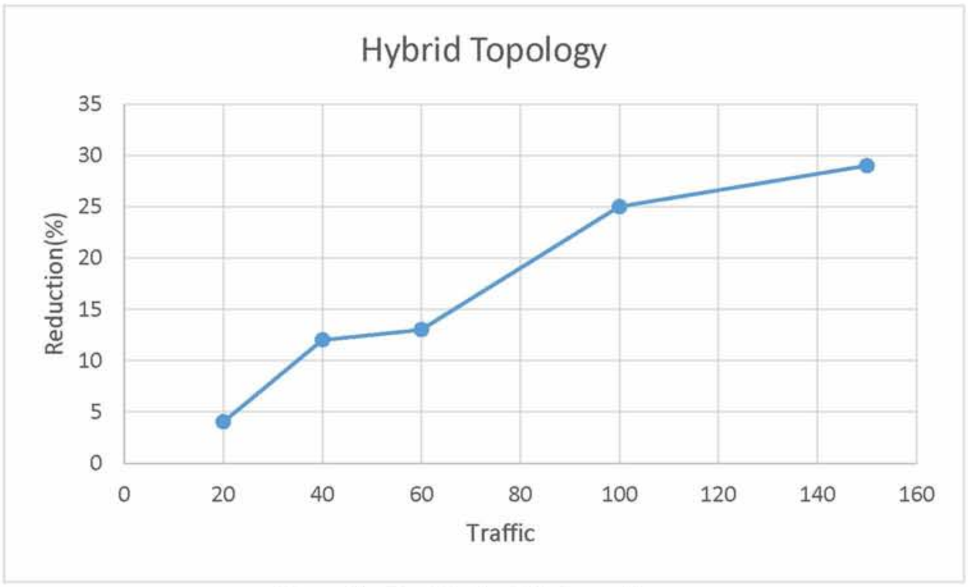
Graph 1: Ring Topology

The next topology we test is the tree topology. The results below depicts a good percentage in flow reduction but not as good as in ring topology.



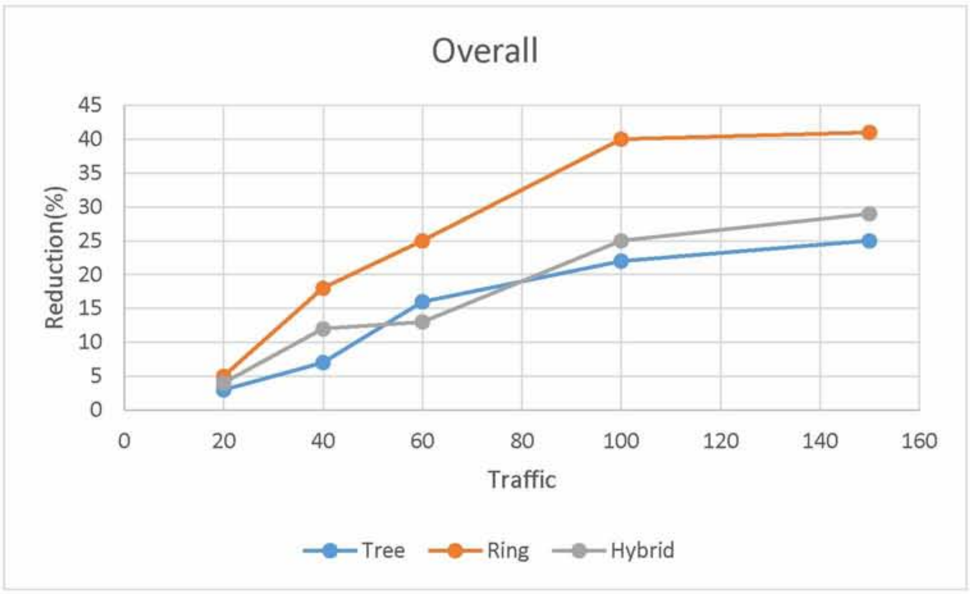
Graph 2: Tree Topology

Next we test the algorithm in a hybrid topology. The results collected from the experiments are shown below but they cannot be used to conclude safely because hybrid topologies have vast formations and alterations and so the results could deviate.



Graph 3: Hybrid Topology

Looking at the aggregate chart we see that in ring topology we can achieve the highest reduction in flows this can be based on the fact that in this topology the paths are common for the nodes and there is no path separation and thus packets travel together. In tree and hybrid topologies the performance is quite similar with hybrid being in the lead. Tree has the worst performance because of its multi-path structure.



Graph 4: Overall

Probing the latencies in every experiment there is a latency drop every time the algorithm starts to kick in and that is when packets share a common path and aggregation can happen. This latency drop is about 30-40 ms for every flow we aggregate. The drop can be substantial as the number of flows in the network escalates. Although the fact that we cannot calculate the delay from the extra lines of code of the module, makes our first calculation inaccurate so we couldn’t include it in this analysis.

# 5 Related Work

Until today, there has been some progress on the issue of traffic engineering and aggregation on software defined networks. Some examples are mentioned below.

Many different approaches on the subject like [32] which uses a more algorithmic way to reduce the size of flow tables. The paper presents a technique named Fast flow table aggregation in which the rules are separated into prefix-permutable partitions and then to each partition is applied to a modified prefix fusion and bit merging (merge together rules that differ by a single bit iteratively).

The paper in [33] has a lot in common with this thesis using actually the same concept and approaches the matter similarly. Although they are very close, [33] lacks in implementation details.

Another study approaches the matter of diminishing the flow entries, differently. The paper in [34] suggests a flow table reduction scheme .The main idea of that is that congested switches' flow tables may have common flow rules and characteristics. These common entries are restructured in one new rule substituting the previous registries leaving space for other rules to be installed.

# 6

# Conclusion

Software Defined Networking as a considerably new principle has evolved the way networks operate and has brought unprecedented innovation pace into the field of computer networks. However similar to almost any new concept, it has its own challenges. In this paper we deal with a hot topic in Software Defined Networks which is the resource management. Initially we provide some basic information about the concept of SDN in order for the reader to fully understand the purpose of this thesis. We examine closely the special memory of OpenFlow devices and focus on the problems and limitations it raises. To treat the memory management problem and reduce the number of OpenFlow rules in the devices we implement a mechanism that aggregates traffic wherever it is possible and without changing or downgrading the quality of that network. The experiments in chapter 4 verify the effectiveness and the efficiency of the mechanism by achieving in some topologies, significant flow reduction. Apart from the noteworthy flow reduction during the experiments we also notice a considerable latency drop which is not included in this thesis because we could not verify the accuracy of that drop. Although this latency drop is something to be taken under consideration for future analysis and further investigation.

## 6.1 Future Work

The current implementation is finished although future development must be done. The code needs to be optimized to reduce the delay of the overhead it introduces in order to have accurate measurements of latency drop. The SDN controller efficiently updates the network with consistency in real-time and safety without packet drops, when the synchronization overhead with the switches is low. Not often and for reasons we haven't found yet we have faced a raise in synchronization overhead which impacts the execution time and brings in further delay we must deal with. There are other parts that demand improvement like the failure recovery and resource allocation to avoid some consistency issues during the experiments.

# Bibliography

[1] CACM Staff, "A purpose-built global network," *Commun. ACM*, vol. 59, no. 3, pp. 46–54, Feb. 2016.

[2] G. Ganger, J. Wilkes, W. USENIX Association, G. ACM Special Interest Group in Operating Systems., and A. S. ACM Digital Library., *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*. USENIX Association, 2011.

[3] International Organization for Standardization, "ISO/IEC 7498-1:1994 Information Technology - Open Systems Interconnection - Basic Reference Model: The Basic Model," *International Standard ISO IEC 7498-1*. pp. 1–68, 1996.

[4] A. Vahdat, D. Clark, and J. Rexford, "A Purpose- Built Global Network: Google's Move to SDN," *Commun. ACM*, vol. 59, no. 3, pp. 46–54, 2016.

[5] W. Zhou, L. Li, M. Luo, and W. Chou, "REST API design patterns for SDN northbound API," in *Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014*, 2014, pp. 358–365.

[6] M. Berman *et al.*, "GENI: A federated testbed for innovative network experiments," *Comput. Networks*, vol. 61, pp. 5–23, Mar. 2014.

[7] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," *Sigcomm '07*, pp. 1–12, 2007.

[8] M. Casado *et al.*, "SANE: a protection architecture for enterprise networks," *15th USENIX Secur. Symp.*, pp. 137–151, 2006.

[9] ONF, "OpenFlow Switch Specification Version: 1.2.0," *Current*, vol. 0. pp. 1–312, 2011.

[10] "McCauley, M.: About NOX," 2012. [Online]. Available: http://www.noxrepo.org/nox/about-nox/.

[11] "McCauley, M.: About POX," 2012. [Online]. Available: http://www.noxrepo.org/pox/about-pox/.

[12] "Floodlight Is an Open SDN Controller," 2013. [Online]. Available: http://www.projectfloodlight.org.

[13] "The Open Network Operating System (ONOS) is a software defined networking (SDN) OS," 2014. [Online]. Available: http://onosproject.org/.

[14] "Shimonishi, H.: Trema : Full-stack openflow framework in ruby and c," 2009. .

[15] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of OpenDaylight SDN controller," in *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, 2014, vol. 2015–April, pp. 671–676.

[16] "Hp 3800 switch series (2011)," 2011. [Online]. Available: http://h17007.www1.hp.com/us/en/networking/products/ switches.

[17] “Pica8 (2011),” 2011. [Online]. Available: http://www.pica8.com/open-switching/open-switching-overview.php.

[18] “Watanabe, H.: Nec programmableflow - univerge pf5820. Tech. rep., NEC,” 2012. .

[19] “Yiakoumis, Y.: Pantou : Openflow 1.0 for openwrt,” 2011. [Online]. Available: http://www.openflow.org/wk/index. php/OpenFlow_1.0_for_OpenWRT.

[20] “Open vSwitch: Production quality, multilayer open virtual switch,” 2013. [Online]. Available: http://openvswitch.org/ features/.

[21] J. Liao, “SDN system performance,” 2012. [Online]. Available: http://pica8.org/blogs/?p=201.

[22] “NEC ProgrammableFlow Networking.,” 2012. [Online]. Available: http://www.necam.com/PFlow/.

[23] A. A. R. Curtis, J. C. J. J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “DevoFlow: scaling flow management for high-performance networks,” *Proc. {ACM} {SIGCOMM} 2011 Conf. Appl. Technol. Archit. Protoc. Comput. Commun. Toronto, {ON}, Canada, August 15-19, 2011*, pp. 254–265, 2011.

[24] Z. Guo *et al.*, “JumpFlow: Reducing flow table usage in software-defined networks,” *Comput. Networks*, vol. 92, pp. 300–315, Dec. 2015.

[25] and S. S. Hiroaki Yamanaka, Eiji Kawai, Shuji Ishii, “OpenFlow Networks with Limited L2 Functionality,” Network Testbed Research & Development Promotion Center National Institute of Information and Communications Technology, 2013.

[26] “IEEE 802.1Q.” [Online]. Available: https://en.wikipedia.org/wiki/IEEE_802.1Q.

[27] “sFlow: 2 Software defined networking.” [Online]. Available: http://blog.sflow.com/2012/05/software-defined-networking.html.

[28] Intel, “Intel® Core$^{TM}$ i5-4440S Processor.” [Online]. Available: http://ark.intel.com/products/75040/Intel-Core-i5-4440S-Processor-6M-Cache-up-to-3_30-GHz.

[29] Amazon, “Intel PRO/1000 Pt Dual Port Server Adapter.” [Online]. Available: https://www.amazon.com/Intel-1000-Dual-Server-Adapter/dp/B000BMZHX2.

[30] Intel, “Intel® Core$^{TM}$ i3-7100U Processor.” [Online]. Available: http://ark.intel.com/products/95442/Intel-Core-i3-7100U-Processor-3M-Cache-2_40-GHz-.

[31] Amazon, “Boadcom-BCM94321MC-802-11n-Wireless.” [Online]. Available: https://www.amazon.com/Boadcom-BCM94321MC-802-11n-Wireless-Selected/dp/B00FA3NR3E.

[32] L. M. L. Shouxi Luo, Hongfang Yu, “Fast incremental flow table aggregation in SDN,” in *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, 2014.

[33] N. M. Saurav Das, Yiannis Yiakoumis, Guru Parulkar, “Application-Aware Aggregation and

Traffic Engineering in a Converged Packet-Circuit Network," in *Proceedings of Optical Fiber Communication Conference and Exposition*, 2012.

[34] Y. Z. Bing Leng, Liusheng Huang, Xinglong Wang, Hongli Xu, "A Mechanism for Reducing Flow Tables in Software Defined Network," in *IEEE ICC 2015 - Next Generation Networking Symposium*, 2015.