PFE

N° d'ordre : PFE-INFO-2014-21

Projet de Fin d'Études

Élève ingénieur de l'INSA de Rennes
Spécialité INFO
Année universitaire 2013-2014

présenté par
Jean GUEGANT

Software Defined Networking applied to home networks

Lieu du Projet de Fin d'Études
Technicolor, Rennes, France

Tuteur du Projet de Fin d'Études
Stéphane Onno

Correspondant pédagogique INSA Rennes
Jean-Louis Pazat

PFE soutenu le 19/06/2014

Acknowledgments

First and foremost, I would like to express my deep gratitude to Stéphane Onno, my internship supervisor, for his patient guidance, enthusiastic encouragement and useful critiques. This introduction to the research and development process was very pleasant and interesting.

I would like to thank the three former INSA students Clémentine Maurice, Nicolas Le Scouarnec and Fabien André, for their advices and the fun times we had together. I wish, also, to thank the other interns for their warm welcome and cheerfulness.

Finally, I would like to extend my gratitude toward Technicolor and the whole team for giving me the chance and opportunity to work in this organization in order to complete an amazing internship.

Contents

List of Figures

Introduction

The present document describes the assignments handled, the working environment, the successes and the issues solved during my final year internship within the French company Technicolor. This internship was accomplished inside the *User Profiling and Data Mining* Technical Area (TA) of Technicolor Research and Development (R&D) from February 2014 to July 2014.

Software Defined Network (SDN) is a new promising technology that will shift the future of networking from protocols to applications. It aims to provide network abstractions and network application layer interfaces for "network programmers" to write network applications from higher abstraction languages. Today's home networks are quite complex to configure and with the advent of Home Automation and Internet Of Things, numerous heterogeneous devices will be inter-connected at home. Questions arise about providing comprehensible means to fulfill the fine-grain monitoring and control of home devices, services and related bandwidth inside home networks.

The goal of the internship was to explore SDN for Home networks and to develop a proof of concept including network applications using various technologies like Python, Java, Linux/UNIX environment, Android SDK...

My internship, after a phase of few months of researches, started to focus on a particular use case of home network: the seamless consumption of your multiple Internet access with SDN. By using the OpenFlow capabilities of your local network equipment, all your Internet connections at home (DSL, 3G, 4G...) can be aggregated and use in a smart and efficient way in order to enhance the overall Quality Of Experience (QOE) of your family.

The remainder of the report is organized as follows. Chapter 1 presents Technicolor and the *Media Computing Lab*. Chapter 2 develops the concept of Software Defined Network and the state of the art. Chapter 3 focuses on the OpenFlow specification. Chapter 4 indicates the usages of this specification in home network and the possible directions for future work. This chapter is followed by a conclusion of the internship.

Chapter 1

Internship context

1.1 Technicolor SA

My internship took place within the research entity of Technicolor, Rennes France, the largest technology center of Technicolor SA Group. Technicolor SA, formerly Thomson Inc and Thomson Multimedia, is a worldwide technology leader in the media and entertainment sector. The company mainly focuses on media and entertainment industries as well as providing services and products for the communication. These products and services encapsulate the entire movie *production* and *consumption* chain, from the shooting of the latest blockbuster to the subtitling of a recent TV series.

Figure 1.1 – Technicolor SA logo.

Founded in 1893 as Thomson SA, this small company is now a multinational corporation scattered in no more than 28 countries. In 2013, Technicolor had approximatively 14,000 employees, a revenue of €3.5 billion and owned a portfolio of more than 40,000 patents. Most of its intellectual properties are derived from more than 5,600 inventions.

Amongst all its activities, Technicolor is particularly renowned:

- For being the participant in more than 70 standard bodies and industry associations. For example: the advanced AV compression technologies (MPEG2, MPEG4...), the digital decoders, the digital TV & video standards...

- For claiming to control MP3 audio codec licensing of the Layer 3 patents in many countries, including the United States, Japan, Canada and EU countries (along with the Fraunhofer Institute). Technicolor has been actively enforcing these patents in the past years.
- For reaching the third place as a producer of gateways and set-top boxes for over a decade. Technicolor is operating in countries all around the globe and is delivering 20 million devices each year.
- Finally, for being the world's largest manufacturer of Blu-ray and DVD products. Technicolor is offering its services to independent content owners, game and software publishers, and even Hollywood studios.

Technicolor is partitioned in three main business units:

Connected Home This business unit targets the media content delivery through modern telecommunication products. Their products range from softwares, like QEO, a communication framework that interconnects devices and applications from all bands and ecosystems, to hardwares like gateways, modems and set-top boxes. This unit generated a revenue of €1.244 billion in 2012.

Entertainment Services This unit aims to be at the forefront of content creation, imaging, and the global distribution of media content. Their services are at disposal for the post-production, special effects (SFX, SPFX, or simply FX), media storage... This unit generated a revenue of €1.730 billion in 2012.

Technology This unit incorporates both Research & Innovation (R&I) and Intellectual Property and Licensing activities (IP&L). All but five percent of the company IP assets result from in-house research and development by the Research & Innovation subunit. Their innovations cover fields like advanced audio and video codecs, content discovery, color gamut expansion and much more.

The IP&L subunit ensures the acknowledgment and the preservation of the intellectual properties originated by the R&I subunit. It mainly involves the assistance for patenting process and the infringement analysis. This unit generated a revenue of €0.515 billion in 2012.

1.2 User Profiling and Data Mining

My internship's team *User Profiling and Data Mining* (a technical area of *Media Computing Lab*) is located in the research centers of Rennes and Paris. Technicolor owns two more research centers located in Palo Alto and Hanover. Technicolor employs up to 300 researchers for research, worldwide, that publishes more than 100 papers every year.

Figure 1.2 – Technicolor Rennes new research center inaugurated in November 2012 ($18,000m^2$, including two datacenters hosting 200 servers)

The guideline of research of Technicolor Rennes are:

- The improvement of video content by enriching it with interaction, visual and sound effects.
- The study of viewer's behavior when facing the content and the impact of new technologies such as 3D in daily-life.
- Ease the access to content, by analyzing the user habits and generating custom recommendations from the analysis.
- Improvement of digital technologies for cinema professionals by providing efficient toolkits for manipulating media content.

The *Media Computing Lab* and more precisely the *User Profiling and Data Mining* team tries to discover patterns in large data sets that can be collected from all media products. The acquired knowledge can, thereafter, be used to enhance the user experience.

Whilst the Software Defined Networks concept is not closely related to data management and user understanding, the research on this subject took root from a project that revolves around the analysis of data sets coming from a huge Internet Service Provider. Some of the statistics accumulated from the ISP's users were too coarse grained for a deep analysis. On the other, home networks lack of flexibility when it comes to add new inspection features or more generally any new trait. The Software Defined Networks, and its main standard OpenFlow can solve most of these drawbacks. I carried out my internship toward this research direction.

Chapter 2

Software Defined Networks (SDN)

The Software Defined Networks [1] paradigm originates from a work done in the University of California, Berkeley and in the Stanford University in 2008. It aims to ease the management of any network by abstracting its core components and their low-level functionalities. The purpose of this chapter is twofold. The first section presents an overview of this new paradigm. The second section explores its current implementations.

2.1 A new approach to computer networking

2.1.1 Abstraction and virtualization of your network

The evolution of computers Nowadays computer science like software engineering, hardware designing or database management rely on well established basic principles. For decades, engineers and researchers keep on top of their mind that the current technologies should be easy to manage, must have a great interoperability and could evolve soon or later. The best way to achieve this flexibility is to abstract each components and try to separate your architecture into layers.

For instance, a general-purpose computer should be separated into the following parts:

1. A micro-processor and the core components (GPU, motherboard, user devices...).
2. One or more operating systems.
3. Many applications running on top.

For a visual representation, see the Figure 2.1.

Each part provides an open and well-defined interface that can be employed by the upper parts. The enumeration is highly simplified and is way far from being exhaustive. An operating system can, for example, be divided into the user land and the kernel land. The operating system kernel is made of drivers and core services communicating each

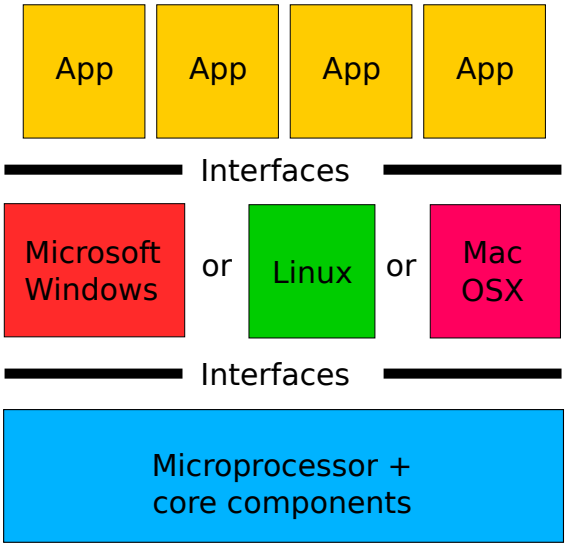

Figure 2.1 – A general-purpose computer architecture.

others , etc. Note that even if one of the components hides its inner working for any reason (proprietary technology, security...), a simple interface can be provided without breaking the compatibility with all the other components.

One further step is to virtualize part of your environment in order to logically dividing the mainframes' resources instead of breaking them down into physically entities. The Java virtual machine (JVM) [2] and the Common Language Runtime (CLR) [3] for Microsoft's .NET framework are typical implementation of a process virtual machine that highly strengthen the portability, the performance, the ability to reuse components and extensibility.

Back in the appearance of computers (around the 60's), the industry proved these basic principles to be wrong. In fact, the main actors were acting in the opposite way. In these times, your workstation was a specialized hardware executing a specialized operating system onto which specialized applications were running on. These vertically integrated, closed and proprietary solutions were barrier to the evolution and restricted computer science to few industries targeting expert users. Thanks to the arrival of the personal computer (PC) and many other improvements, today's solution are horizontally integrated and boost the innovation. It is even possible to create a daily-life device from the hardware to the software, like a modular smart-phone integrated with the android operating system.

Stagnation in the networking area Unlike the majority of the computer science areas, the communication systems are quite stuck for few years, with a high lack of

progress. Whilst the theory behind this technical domain was organized such that it should be adjustable and extensible, it proved to be false in reality, at least in the core parts.

For example, by taking a look at the OSI (or Open System Interconnection), a model that defines a networking framework to implement protocols in seven layers, you can clearly feel the motivation to abstract each components of the network architecture in order to solve the above-cited problems. Firstly, the OSI model is partially violated by few protocols like FTP, or ICMP, that mix-up two different layers. Secondly, the practice of commercial networking is that the standardized external interfaces are narrow and none can decide to change one of the OSI layer without truly impacting the market. In result, the networking domain currently consist in a bag of protocols, difficult to manage and that evolves slowly [10]. One can take a look at the reluctant transition from the protocol IPv4 to its new version: IPv6. The Internet Engineering Task Force committee in charge of this case, started to work on the IPv6 proposal in the beginning of 1992. In May 2014, IPv4 still carries more than 96% of Internet traffic worldwide [4].

From the home gateway, to an university media infrastructure, every single network equipment is bounded to the implementation maintained by the manufacturers. This strategy could be a barrier for new ideas. Even if some open-source initiatives, like the operating system OpenWRT or DD-WRT, permit to create additional features to many routers and switches, the core structure of the networks can't change.

Dedicated to solve these clumsy problems, a research lab focused recently (around 2008) on hiding the lower level functionalities of network core items with an efficient interface. This was the beginning of the Software Defined Networks (SDN). The goal was to reach the same kind of architecture we talked about earlier (a nowadays workstation) for networks:

1. A merchant switching chip with basic features as the hardware base. This layer will solely handle the forwarding of the data according to some really basic rules. This is the equivalent of the CPU.
2. One or many control plane that will do the link between the two other layers. This is the equivalent of the operating systems.
3. Many applications that will do the logic part of your network. This is the equivalent of a classic software application.

By implementing such a stack, a network can be fully programmable. This permits to freely run any kind of experiment on it! It could also simplify the task of the manufacturers who can only focus on the hardware and a bit of boilerplate for enabling the SDN protocols. The network logic, in this configuration, is up to the administrators.

2.1.2 Separating the control plane from the data plane

The Software Defined Networks approach decouples the system into two parts:
- The data plane, that is simply dedicated to forward traffic to the selected destination.
- The control plane, that makes decisions about where traffic is going to sent.

For a visual representation, see the Figure 2.2.

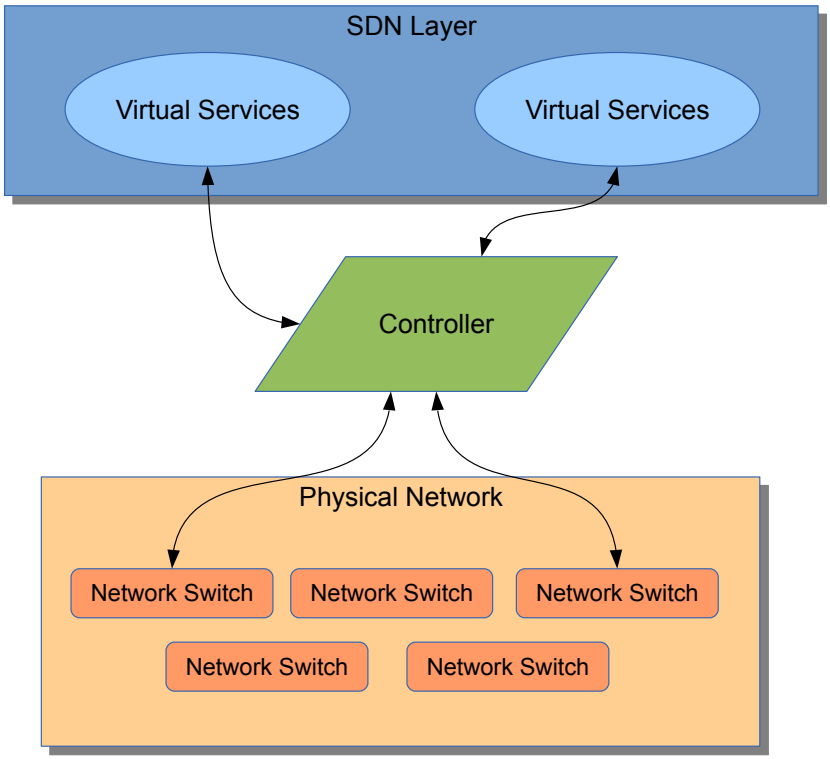

Figure 2.2 – Software defined networks system overview.

The data plane The data plane simply receives a bunch of packets and redirects them according to some predefined rules. The terminology for these packet groups is **flow**. Once a new flow is received, its first packet is compared to the rules stored locally:
- if the packet header matches one of the rule, some actions are taken according to this rule; otherwise
- an exception is thrown and the packet is sent to the control plane that will update the logic of the network by pushing some new rules.

The rules are extremely fine-grained and one can easily creates brand new protocols from them.

Physically, it only consists of **network switches** cleaned from any sophisticated cleverness. A bit of processing treatment is nonetheless necessary for keeping the communication between the data plane and the control plane. These dumb switches are way less expensive than a traditional ones. A vendor can also decide to sell its switches as an hybrid-mode, which keep two stacks: one for SDN and one for the legacy applications. The network administrators can later choose if they wish to include the switch into an SDN ecosystem or use them in a more traditional manner.

The control plane Also named **controller**, is the "*brain*" of the network. It manages one or more data plane units by giving some orders packaged into rules. The network coherence is supplied by some **Virtual Services** running seamlessly on top of the controller. These virtual services are most of the time provided by some applications using a public API implemented in the controller framework. This architecture highly reduces the resource charges and reinforces the flexibility, since all the hard-core mechanisms are now centralized. All the network services can take profit of an ease of communication never reached in the past: the networks instantly get more cohesion. Note that the control plane can be exported to a cloud solution for reducing the running costs and maintenance costs (elastic resource allocation, load balancing, improved security...). The control plane can be closer to the data plane if increasing the latency is mandatory.

The two planes communicate using a secure channel called **control path** or **control channel**. One SDN switch can communicate with one or more controller. In this case, a consensus is established between the control planes, assuming you wish to keep a good orchestration of the network. Last but not least, in case of a failure or a misconfiguration of the communication between the two planes, a backup mode must be prepared with a minimalist setup.

An example of flow rules installation could be for a new TCP connection from an internal device and in destination to an external entity of the local network. On the first TCP packet reception (a SYN packet), no rule has been established, the packet is then forwarded to the control plane. In return, an application on the control plane may generates rules that will be applied to the switch, like:

- Some rules for forwarding on the right physical port (the up-link).
- Some rules that handle the state of the TCP connection.
- Some rules that perform the NAT mechanism.

During the whole life cycle of the TCP connection, no other packets will be send to the controller, the overhead of this method is therefore quite negligible. Note that the controller could have also pro-actively pushed these rules.

2.2 The state of the art

2.2.1 Actors

At first solely used by a couple of researchers, SDN quickly influenced many big designers, manufacturers, and sellers of networking equipment. Amongst them: Brocade, Cisco, Juniper, Huawei, and many more. These companies furnish some SDN enabled equipments as well as contributing to the community through various open projects. SDN has been designed to be non-intrusive to their intellectual properties and discards any reluctance towards this technology.

It also gathered third-party users, like: Microsoft, Amazon, Google and Facebook. These multi-nationals control pools of processing, storage, and networking resources throughout data centers. SDN eases the operation of massive networks: first, by separating hardware from software, they can choose hardware according to the required features while being able to innovate on the software side. Second, it provides logically centralized control that will be more deterministic, more efficient and more fault-tolerant. Third, it helps to separate monitoring, management and operation for individual boxes.

For the same reasons, one application of SDN is for infrastructures as a service (IaaS) like Amazon Elastic Compute Cloud [5]. The famous, free and open-source software cloud computing platform OpenStack offers SDN capabilities to their architecture. Same goes for the distributed virtual multilayer switch project OpenVSwitch [7] tightly coupled to OpenStack.

The concept of software-defined networking has captured the attention of network engineers and the trade press, but only few examples of a live SDN implementation exist for small projects and small infrastructures. Nonetheless, the concept accumulates a plethora of experimentations ranging from usage with smart phones, optimization of Netflix streams, enhancing cellular networks or finding applications to home network (like this internship). Recent projects like OpenDaylight [6] seem to be a glimpse of hope for a worldwide spreading of these technologies in the upcoming years.

2.2.2 Network Function Virtualization

Parallel to the SDN creation, a new hot-topic in networking called Network Function Virtualization (NFV) appeared. For instance, this concept aims, for Internet Service Provider (ISP), to relocate the core components of their infrastructure into the cloud. For example: families won't have their gateway located in their houses, but somewhere hidden in the ISP cloud. A simple switch (or wifi access) is considered as enough for plugging yourself at home. Although some people may claim it wrong, NFV and SDN could perfectly coexist on the same network.

You may take look at a brief comparison of NFV and SDN in the following table [8]:

Category	SDN	NFV
Reason for being	Split up control and data, centralization of control and programmability of network	Relocation of the network from dedicated equipments to generic servers
Target location	Local network, Campus, data center / cloud	Service provider network
Target devices	Switches and servers	Switches and servers
Initial Applications	Cloud orchestration and networking	Switches, routers, firewalls, gateways, content delivery network
New Protocols	OpenFlow	Not yet
Formalization	Open Networking Forum (ONF)	ETSI NFV Working Group

2.2.3 OpenFlow

The OpenFlow Switching specification was also created in 2008 to evangelize and support the SDN movement *openflowhitepaper*. This is the only current implementation for the concept of SDN[1]. This specification has been developed during various open meetings and the committee recently released the version 1.4 (other renowned versions are 1.1, 1.2 and 1.3). We will dive into the inner working of the version 1.0 and 1.3 into the following chapter *openflowspec*.

Few switches support OpenFlow in the catalog of a vast number of vendors including Alcatel-Lucent, Big Switch Networks, Brocade Communications, HP, Arista Networks, NoviFlow, Cisco, Dell Force10, Extreme Networks, IBM, Juniper Networks, Digisol, Larch Networks, Hewlett-Packard, NEC, and MikroTik. For example, the Hewlett-Packard Company works on the OpenFlow protocol for over five years now, and in February 2012 made its big play, announcing the introduction of a portfolio of OpenFlow-enabled switches, covering 16 models in total and supporting more than 10 million installed ports.

It is important to remember that SDN is not OpenFlow, and vice versa. It is one of the main block of a software-defined network and a very important one.

2.2.4 SDN applications

SDN was designed to be as powerful as a legacy network. It features:
- Routing, switching, load balancing.
- Security (Firewall, DPI, Dynamic Access Control, DOS detection).

1. To the best of the author knowledge.

- Bandwidth management and QOE.
- Traffic engineering (Intra Domain, optimization, scheduling).

But more than just managing a normal network, SDN is particularly efficient when it comes to gain new conceptualizations of your network like virtualization. The two following examples show some applications unmanageable without SDN.

Network Hypervisor [11] Let's suppose that you are a sysadmin of a big university campus. Your mission consists to produce a stable and secure network for two kind of people:

- Students, that always try to do some tricky jokes on the network.
- Teachers, that are far more serious but need a fair share of the network's resources for their experiments.

On top of this, there is a huge demand for improving the streaming services without impacting the comfort of the other services.

In the blink of an eye, you will attempt to partition your network into some virtual local network areas which are mutually isolated for teachers and students. This concept is referred as virtual LAN or VLAN. In most cases, your rules are locally applied on the switches by selecting the physical ports or devices (by MAC, layer 2 in the OSI model) that will be part of the VLAN. You may also define a Quality of Experience (QoE) policy that will preserve the quality of each services. Although this strategy is used by the vast majority of today's sysadmins, it still lacks of features for fine-grained policies.

SDN introduces the notion of **Network Hypervisor**, also designated as **Network Slicing** or **Network Virtualization**. Once again, this concept has been borrowed from the computer engineering, since a network is now programmable with SDN. As easily as computers can execute a hypervisor that creates and runs operating system, an SDN Hypervisor controller can be the base for many other guest SDN controllers. Network resources (switch, bandwidth, latency...) are adequately shared and seamlessly by the hypervisor. You may also desire to have different representations of your network for each controller (physical representation like switches and devices, or even logic representation).

In your campus case, you create rich "slices" of your network for each different kind of people (students and teachers). These slices can be configured by any combination of switch ports (layer 1 in the OSI model), src/dst ethernet address or type (layer 2), or even src/dst IP address or type (layer 3). For the streaming flows, you create another slice but based on a differentiation on the layer 4: src/dst TCP/UDP port or ICMP code/type. Notice, that you can now differentiate a flow at any layer of the OSI model.

Each slice is associated to an SDN controller running on top of this hypervisor. SDN will enforce an isolation between each slice, i.e., one slice cannot access another's traffic. Some services can be accomplished in these controllers through some applications. For example, every slice needs the ability to route the traffic and each one gets a particular behavior:

- The students' network is now watched by a security application, in case a malicious user tries to do a joke.
- The teachers' network is enhanced by a load-balancer for each servers.
- The streaming network is able to cache some chunks of the videos watched by users.

The hypervisor can also decide to allocate more resources, like bandwidth, to the streaming traffic.

The Figure 2.3 exposes a scheme for this new campus architecture.

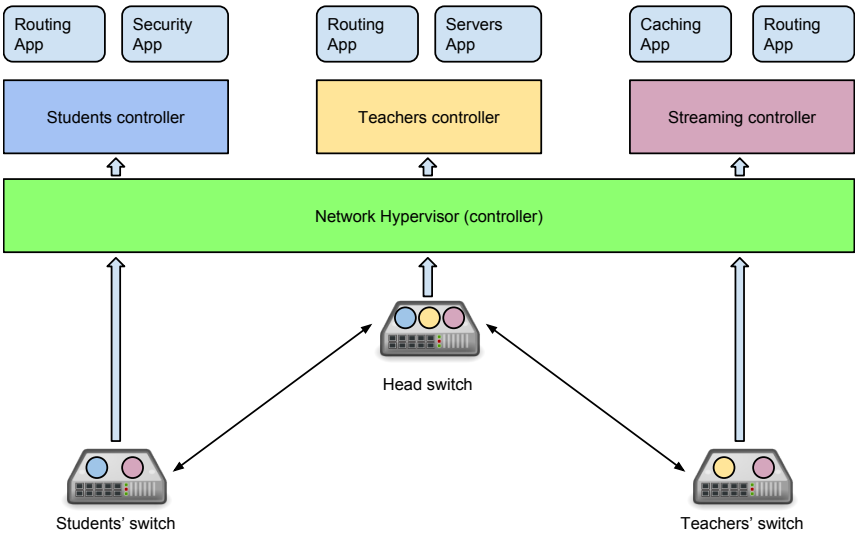

Figure 2.3 – Network Hypervisor over SDN.

The dream by some Internet Service Providers and tier 1 networks is to slice their architecture for each kind of traffic. For example, they could decide to allocate 10M/bits for Youtube in a shape of a slice. The end user could, then, enjoy a 10M/bits pipeline arriving at home with the guaranty that nothing interacts with it. The ISP could even offer the control of this slice to YouTube, for optimization, since the virtual networks would be fully encapsulated by the hypervisor.

Network Fabric [12] When managing a huge data center, one of the most cumbersome task is to replace or move your hardware. The architecture you designed is a well-oiled machine, and you will suddenly insert a new entity in it. SDN can help you to leverage the impact of this evolution at the layer 2 of the OSI model.

You will simply abstract the identifier of the equipments in a data center environment. This identifier is probably a MAC address at the layer 2. All you need to do is to consider your hardware by a **virtual** MAC address. Using some SDN rules pushed onto the switches, you can translate the real MAC address of any of your hardware into a virtual MAC address. Now, if one of the server in charge of your Domain Name System (DNS) is disrupted, you can easily change it with one from the new generation without fearing to impact the layer 2 policies. A straightforward modification of the MAC translation tables should be enough. You replace the old real MAC address, by the one written on the new spare server.

This solution brings many favorable factors:

- You modify your configuration in one and only one place, the Network Fabric application hosted on a remote controller. Adding a new server in your network is now a "plug-and-play" mechanism.
- The MAC addresses you may catch while sniffing the traffic might be now meaningful. For example, all the addresses for the web services could start by the prefix 42:42:42:42:42:xx.
- You can also swap two devices without any physical intervention or even configure a hardware that own two virtual addresses for two different services.

The Figure 2.4 is a visual representation of your Network Fabric environment.

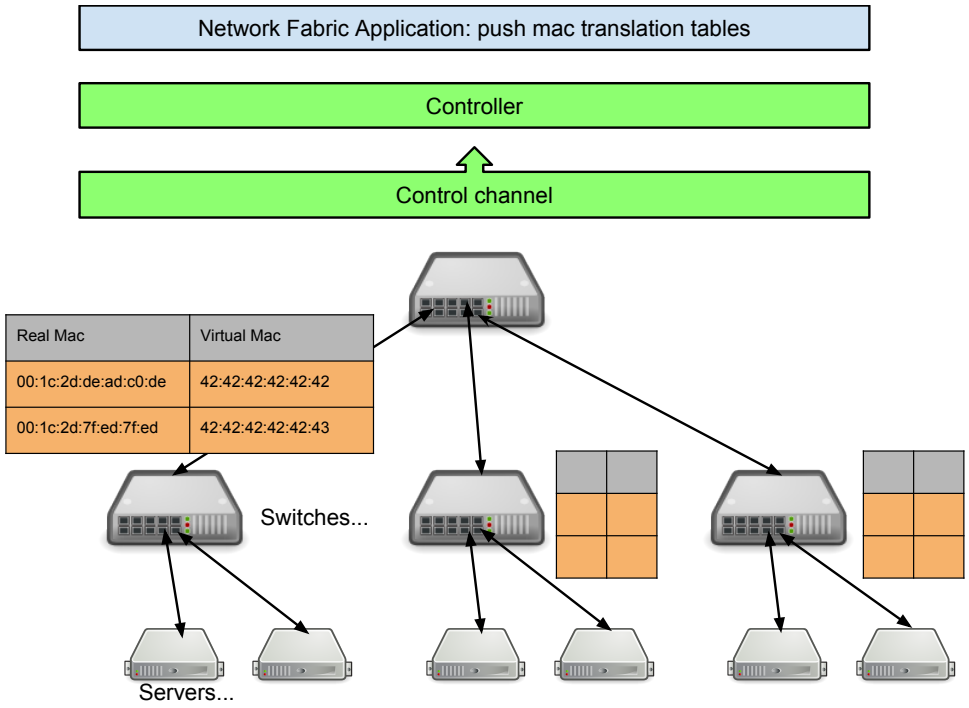

Figure 2.4 – A fault-tolerant Network Fabric applied on the layer 2 of a data center.

Chapter 3

Digging into the OpenFlow mechanisms

Being the main implementation of the SDN concept, a brief look up at the Figure 2.2, "an overview of SDN", could be useful to understand the current chapter. This chapter is a gentle introduction to the OpenFlow protocol, if you wish to deeply improve your knowledge, a glance at the specification might be attractive [9].

3.1 The OpenFlow standard

3.1.1 Architecture overview

An OpenFlow switch consists of one or more **flow tables** and some extra tables, which perform packet lookups and forwarding, and an **OpenFlow channel** linked to an external **controller**. Once a packet arrives in one of the port (either virtual or physical) of the switch, it will initiate its processing into the **OpenFlow pipeline**. The packet is inspected and compared within a couple of tables connected in series. According to its path, the packet will be later sent through one or more port of the switch, or in destination to the controller. Likewise, the packet could be simply discarded during the processing steps.

The switch communicates with the controller and the controller manages the switch via the OpenFlow protocol. Openflow provides security by encrypting communications and checking identities between the data plane and the controller. Using the OpenFlow protocol, the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets) and pro-actively. When the pipeline redirects a packet to the controller, a **Packet-In** event is emitted. This event contains also the identifier of the data plane that forwarded the packet. Correspondingly, a **Packet-Out** event is thrown if the controller decides that a packet should be forged into the data-plane. Linked to this event, a port number specifies the location of the egress.

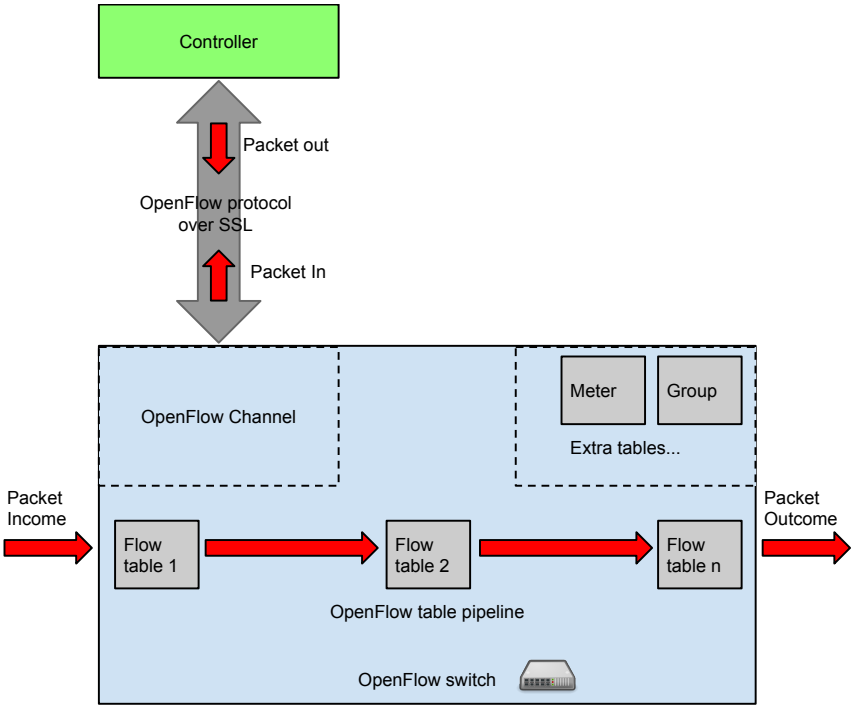

Figure 3.1 – The OpenFlow protocol overview.

3.1.2 Forwarding tables

Disclaimer: according to the OpenFlow version picked, the following section could be inaccurate or deprecated. This section is based on the version 1.3 of OpenFlow which is the current most used standard.

The OpenFlow pipeline is intrinsically consisting of tables that choose the treatment of packets. Starting from the **main table** with the identifier 0, the packet hops onto tables until a forwarding decision is taken. On each tables, the packet header is sequentially checked against the **flow entries** sorted by priority. The pipeline can carry, between each steps, infos for the packet with metadata. If the packet **matches** one entry, the **instructions** associated to this entry are executed. If the instructions contain a **Goto**, the packet can be yield to another table, otherwise the previously stacked **actions** are achieved. The packet can match one and and only one entry in each table. In case that no entry is found, the packet will be simply abandoned and no further treatment will be done. A smart move is to put on each table a **flow miss entry** that will simply feed the controller with all the packets that failed their filtering step. This flow miss entry installation can be easily accomplished with a flow consisting in a wild-card match and an instruction that outputs the packet onto the controller port.

One a side note, a packet can't jump on the same table twice. More precisely, a flow entry with a Goto action can only send a packet onto a table that is in possession of an identifier greater than its own. This restriction avoids never ending loops, but diminishes the OpenFlow's expressiveness: conditional loops can't be simulated.

As a background process, the pipeline will update its internal state. For example, each entry possesses a couple of counters that are increased each time a packet matches. The controller can poll the pipeline state on demand or can command its reset. If any flow entry exceeded one if its **timeout counter**, the flow is dropped and a notification is send to the controller with the flow cookie.

A full activity diagram on the figure 3.3 resumes the pipeline fundamentals:

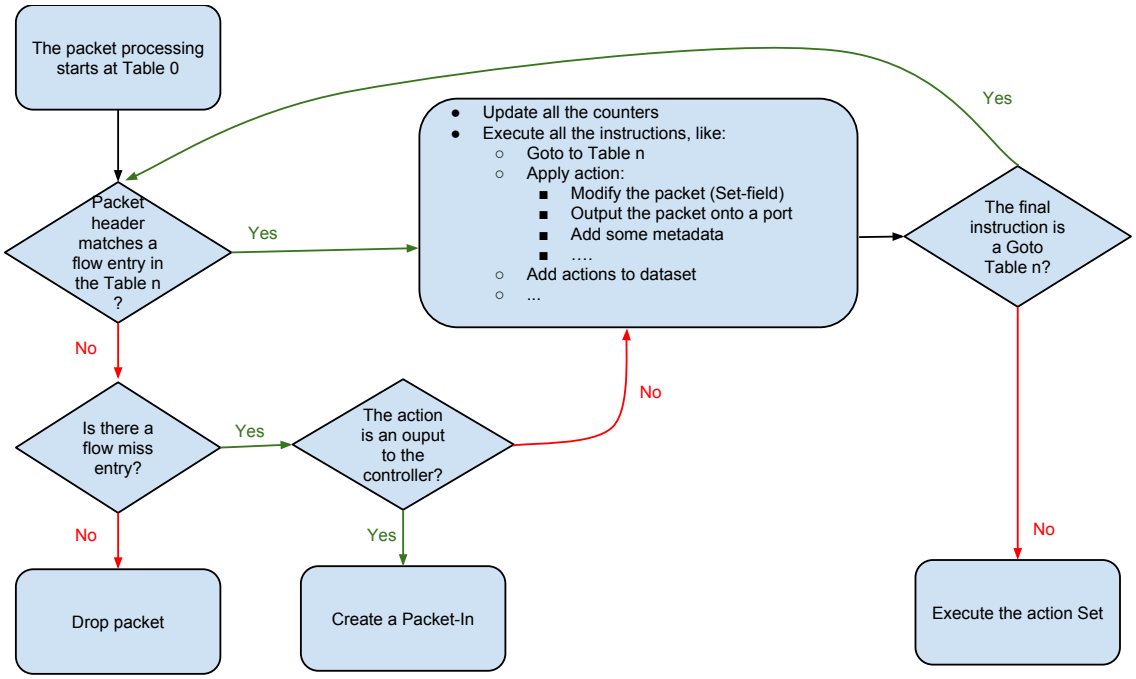

Figure 3.2 – The OpenFlow pipeline activity diagram.

The following listings will explain the primitives structures of the OpenFlow pipeline and can be use as a reference for figuring out the mechanisms of the following example (*A basic example: MAC learning*). Amongst all this structures, some features are required by OpenFlow and some are optionals but highly recommended by the committee.

Logical ports It designates the ports on which a packet can be send through. These ports mainly correspond to physical ports or interfaces (like a wifi interface). However, few

constants are reserved for some special purposes:
- ALL, the packet is send to all the standard ports excluding the ingress port. If the ingress port is equivalent to one of the egress port, your network will suffer from loop effects.
- CONTROLLER, the packet is send thorough the communication channel established between the switch and the controller. This is the equivalent as a Packet-In notification.
- TABLE, the packet will restart its passage into the OpenFlow pipeline.
- IN_PORT, forces the packet to be resend trough the ingress port. This violate the security that avoid network loops.
- ANY, all the standard ports will be used excluding the ingress port.
- NORMAL, it transfers the packet to the normal stack of the switch. This port is not required by the OpenFlow specification but highly suggested if your equipment is in hybrid-mode.
- FLOOD, it uses the normal stack to broadcast the packet on all the ports. Once again, this port is not required by the OpenFlow specification but highly suggested if your equipment is in hybrid-mode.

Matches These structures are used in order to filter packets by their patterns. A packet match can have prerequisites and can be solely encoded by a type-length-value (TLV). Some matches even allow to specify a mask, in order to easily check the state of individual bits regardless of the other bits. A non-exhaustive list of these matches is:
- IN_PORT: The numerical port onto which the packet arrived. This might be either a physical port or a logical port defined by the administrators.
- IN_PHY_PORT: The physical port onto which the packet arrived.
- ETH_DEST: The destination Ethernet MAC address.
- ETH_SRC: The source Ethernet MAC address.
- ETH_TYPE: The Ethernet type of the packet (IPv4, ARP...).
- IPV4_DST: The IPv4 destination address. This match can use a subnet mask or an arbitrary bitmask. This match requires an ETH_TYPE match with a value of 0x800 (IPv4).
- IPV4_SRC: The IPv4 source address. This match can use a subnet mask or an arbitrary bitmask. This match requires an ETH_TYPE match with a value of 0x800 (IPv4).
- Many more...

Flow entry This root primitive is composed of:
- Match fields, used to check whether this flow entry must apply its instructions for the given packet.
- Instructions, the pipeline commands that will be applied.
- Priority, that gives the precedence of this entry.
- Cookie, a value choose by the controller. This value is transmitted on every notification thrown by the switch toward the controller. You may use a cookie as an unique

identifier in order to filter flow manipulations.
– Timeout counters, that express the life time of the flow according to various policies: the idle time, the amount of time the flow spent on this table...

Instructions These instructions are embedded into the flow entries to specify the changes applied to the pipeline processing, the action sets or the packet. The required or recommended instructions are:
– Goto, redirects the packet to a following table. The table identifier is encapsulated into this instruction.
– Meter, makes the packet jump into the meter table stipulated by the identifier.
– Apply-action, directly applies some actions to the packet without changing the current Action-Set.
– Write-actions, adds new actions into the set. The set is executed when the packet reaches the end of the pipeline.
– Clear-action, clean-up the set from all the actions, immediately.
– Write-Metadata, attaches new infos to the current processing. This informations can be later accessed during the following steps.

Actions Either packed into a set or executed directly, actions are the core results of the pipeline. Actions are applied by an order specified below. The following list only contains the most useful actions:

1. Decrement TTL, it decreases the time to live field of the packet.
2. Set, it modifies some fields of the packet. You can roughly adjust the same field as the ones you can match on.
3. QOS, it handles the quality of service assured by the pipeline (meters, queues...).
4. Output, it forwards the packet on a specified logical port.

The OpenFlow pipeline provides few more tools to ease the administration of your network:

Group table This table stores some actions wrapped altogether. These groups facilitate the broadcast and multicast forwarding. These groups are used to prepare new output strategies like a fast failover or a port indirection.

Meter table OpenFlow furnishes a table to indicate simple QoS operations, such as port queues or rate-limiting. Each entry of this table is assigned a way to process the packet (token bucket strategy, maximum bandwidth...), few counters, as well as an identifier. A flow table entry can be seamlessly linked to a meter table entry.

Synchronization mechanisms Finally, being inherently an asynchronous protocol, OpenFlow must exposes some synchronization instruments. You can ensure the ordering of your flow modifications with some barrier messages. The ordering can preserve the safety of the whole system and could boost its performance.

3.2 A basic example: MAC learning

In a network forwarding equipment like switch or a hub, the MAC learning is the algorithm that associates a MAC address to a physical port. This MAC to Port dictionary is mandatory to forward a packet stream in destination to a local device. Since an OpenFlow-enabled switch is stripped of any intelligence, it does not even ensure this service. As a sysadmin, the first application you will push on top of your controller must be a Mac learning application. Here is a basic implementation:

The application installs and configures two tables: the main table that will dispatch the traffic between the mac learning table (local traffic) and the rest (external traffic), and the MAC learning table. The main table setup step is achieved by:

1. Creating an entry that matches the local traffic and will transfer the processing of these kind packets to the MAC learning table. The match's layout must be: ETH_TYPE = IP, IPv4_SRC in the LAN and IPv4_DST in the LAN. This rule must have the highest priority, the local traffic being the target.
2. Installing an entry that forwards every packet, with a wildcard match, to the MAC learning table. When a packet is received and no other entry took the precedence, this rule, with the lowest priority, will make the MAC learning checks there is anything to do.

The MAC learning table setup consists in pushing a wildcard entry that will send back all the traffic to the controller. If this rule is reached, it means that no association has been made for the two local devices trying to communicate and their ports. Some new rules will be pushed into this table later on.

During the runtime of the application, the MAC learning mechanism will update its internal context. The association rules are produce in this manner:

1. A packet, following the initiation of a communication by a local device, arrives into the MAC learning switch table. Since no rules has been pushed in the past, the packet will be sent to the controller.
2. The controller updates its dictionary that associates a MAC to a port.
3. If the controller already know the port of the destination device it will:
   (a) Send the packet into the right destination port.

(b) Build two rules that handle the new path between the two devices:
- A rule that forwards all the traffic from the source device to the port of the destination device.
- The inversed rule, that will be in charge of the traffic that come in return from the destination device.

Otherwise the packet is broadcasted into all the logical ports, expect the ingress one, using the output action and the ALL special port.

4. The controller updates its flow entries when they timeout or when a device plugged itself into a new port.

A diagram that roughly explains the internals of a Mac learning application:

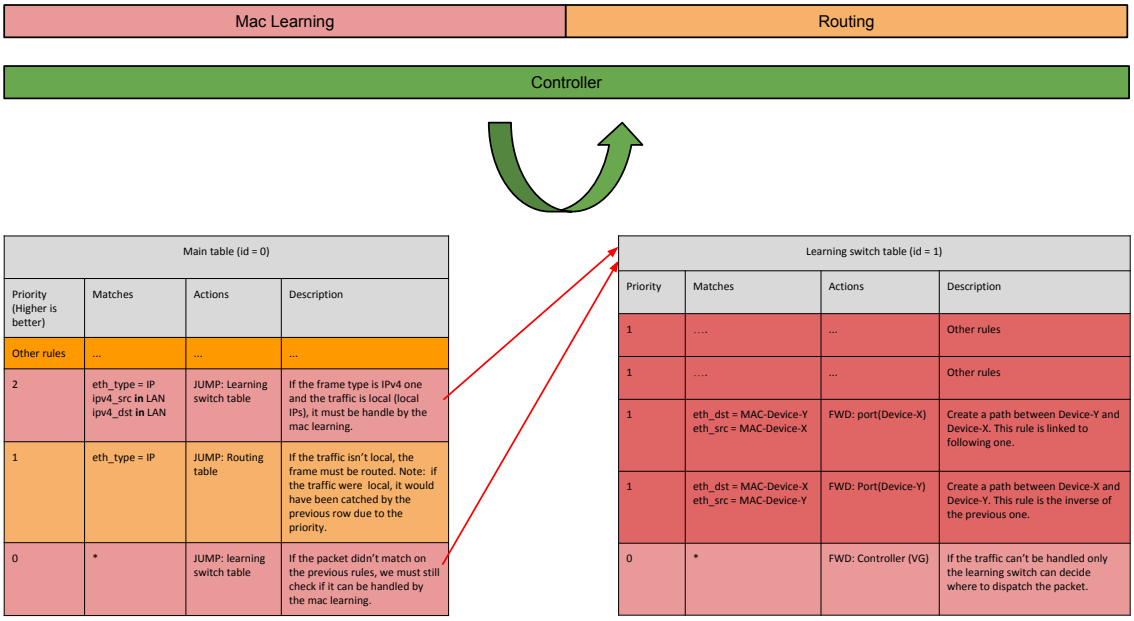

Figure 3.3 – The flow entries necessary to the MAC learning mechanism.

3.3 Available technologies

A great part of this internship revolves around studying and identifying the OpenFlow technologies available nowadays and compliant to the home use-case. An OpenFlow architecture must encompass two core components: the controller and the switch implementation.

Almost all the following solutions rely onto the C OpenFlow library, that simply integrated the communication protocol and the constants necessaries to design one side of the architecture.

3.3.1 Controllers

The controllers experimented during this internship were:

Pyretic [13] This controller is an open-sourced Network Operating System (NOS). Pyretic is both a programmer-friendly domain-specific language embedded in Python and the runtime system that implements programs written in the Pyretic language on network switches.

This product is highly portable, fast enough and gets the fanciest flow manipulation using its specific language to express rules. We used it for our first experiments. Sadly, the pyretic team still didn't took in charge one of the latest version of OpenFlow. The 1.1 OpenFlow version restricted us too much. Furthermore, the framework and its runtime was not flexible enough for our purposes. We, then, opted for the two other controller to continue our researches.

Opendaylight [6] OpenDaylight is an open platform for network programmability to enable SDN and create a solid foundation for NFV for networks at any size and scale. OpenDaylight software is a combination of components including a fully pluggable controller, interfaces, protocol plug-ins and applications.

This project, using a platform entirely in Java with the OSGI plugin framework, is carried by the big actors in the market as well as the Linux foundation. Highly modular and abstracted, this is most likely the project everyone will follow for the next years. Whilst OpenDaylight is titanic (1M of code lines) and we used it for few operations, it revealed to be unsuitable for our needs. First, the project is a bit too young and unstable, and forces us to catch up with every single modifications. Second, this product is too massive for our workforce and for the devices we wish to use in home networks.

Ryu [14] Ryu is a component-based software defined networking framework. Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc.

Our team, as a result to the previous dissatisfactions, decided to base all the projects onto this framework. Ryu is portable, flexible, lightweight, up-to-date and smooth the development process. This substitution of OpenDaylight to Ryu also revealed the possibility to include the controller into a thin hardware like a Raspberry Pi.

3.3.2 Switches

Our researches focus on studying if the SDN movement could spread into the families' electronic devices. None can afford the real hardware-enabled OpenFlow switches, sell by the big telecoms brands, for a home usage. These equipments aren't even designed for a daily usage (huge shape, heavy frames...). In order to place ourselves into the best testing environment, we ordered some cheap routers like the **TL-MR3420** by TP-Link. These routers can be flashed with a new embedded operating system, based on Linux, called OpenWRT. With OpenWRT, a full control of the router is gained and new features can be installed like an OpenFlow software that emulates an OpenFlow behavior.

For a rapid development process and for easing the debugging period, we needed the ability to test our changes directly on our PCs. We could later deploy the new code onto our real environment for further tests.

Following, the three technologies experimented during this internship:

ofsoftswitch (CPQD version) [15] ofsoftswitch is an OpenFlow 1.3 compatible user-space software switch implementation. This version of the project maintained by CPQD is based on the Ericsson TrafficLab 1.1 softswitch implementation. This piece of software is really small and quite portable in an unix environment. We use its current implementation onto our TL-MR3420 routers.

Open vSwitch [7] Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols including OpenFlow. It has also been integrated into many virtual management systems including OpenStack, openQRM, OpenNebula and oVirt.

Open vSwitch is highly reliable and portable. Some ports even exist for the Android smartphone platform, creating a profusion of stimulating use cases for our researches. Although, this virtual switch is more polished than ofsoftswitch, its memory footprint is too excessive for the TL-MR3420. We still used Open vSwitch in our desktop assessments.

Mininet [16] Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single command. We used this carefully crafted Python toolkit to simulate a full network environment onto our computers.

Chapter 4

Our researches' domain: the home networks

With the Internet of Things (IoT), our local networks will tend to be more and more complexes. For example, the number of connected electronics devices in residential home networks increases drastically and the current average in 2013 is about seven devices per household. Our researches focused on solving the incoming problems using the OpenFlow specification and more generally the SDN concept.

4.1 Our experiments

This section will explain in few sentences the experiments we concluded during the 6 months of this internship. This is the result of a deep digging job into documentations, of frequent team brainstorming and personal investigations. In order to avoid plagiarism, a Technicolor researcher must also keep in touch with the competitors' inventions and the new theories bring by public research centers.

4.1.1 Evaluating the technologies

Before diving into the core of our researches' domain, we wanted to ensure that each technology, that we picked, would be suitable. Home networks are, for instance, way more restricted in resources than an Internet backbone. A user from a household will react differently than a technician operating during his working time.

We tested close to half of the OpenFlow controller available by simply implementing a basic MAC learning application (see the section *A basic example: MAC learning*). As for the others, we conscientiously inspected their white papers and their tips.

We also gathered some expertise for the surrounding technologies: like OpenWRT, Linux, Android, Java, Python and many more.

4.1.2 LAN aggregation

Let's imagine that you wish to aggregate two local subnetwork from two different houses. For instance, a grand-ma could intent to print a document using the printer located into the accommodation of her grand-son without doing any trip. Using the traditional technologies, you would immediately jump onto a VPN connectivity like OpenVPN or any equivalent. Whilst these technologies are well proven, they are not straightforward for the common user.

We discovered that the OpenFlow control channel, also called the control path, can be hijacked to transmit information between two controlled switches. The trick simply relies on the Packet-In and Packet-Out events of the OpenFlow protocol.

This solution might seem a bit bulky, but only requires two dumb electronics devices running OpenFlow. Moreover, the Internet Service Provider can just enable this aggregation application remotely without disturbing the users. We fully demonstrated the feasibility of this idea by passing some ping probes from two LANs, without taking in consideration the minor inconsistencies that can arise when two local networks merge.

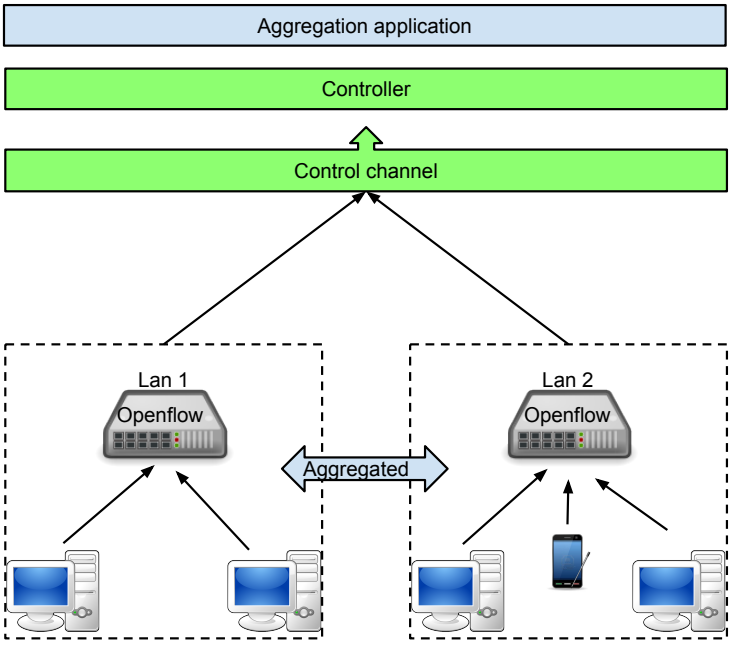

Figure 4.1 – Aggregating two home networks with OpenFlow.

4.1.3 Remote Controller

From the previous experiment we settled to meticulously try all the controller configurations, including the remote one. An Internet Service Provider, that adopt OpenFlow, will most likely install its controller infrastructure into the cloud. Exporting the intelligence of your network is not without any drawbacks. We had to certify that the impacts would be only minors.

In order to verify, we linked a router hosted into Technicolor with a controller running on an Amazon EC2 instance (a web service that provides resizable compute capacity in the cloud). We observed an increase of the latency for the first packets on local communications, but that could not be felt from the user point of view.

4.1.4 Traffic shaping and QoE

On a local network that supports many end-users, collisions and few bottlenecks may be experienced due to a bad scheduling and shaping of the traffic. The quality of experience (QoE) must be arranged for each devices and for each kind of activities (games, voice over IP, streaming...). These troubles can also arise on the Layer 1-2 of the network, like the wifi air transport.

The meter tables of OpenFlow contain some per-flow rules that enable various simple QoS operations, such as rate-limiting and queues. Thanks to the abstraction provided by OpenFlow, we effortlessly manipulated these rules to create a policy of QoE. We could visually check up the correctness of our work by watching the quality of some online streams.

4.1.5 Multi-WAN routing

At home, a main router may have multiple WAN interfaces. The traffic of the house can be scattered on each of the interfaces. In result, the network capacity and quality should be improved. Unlike "channel bonding" or "link aggregation" which routes individual frames, the Multi-WAN routing separates the telecommunication per session. You can, for example, send all the streaming traffic on one interface and reserve the second one for critical services. Round-robin algorithm can also be employed to schedule each connection.

We benefited from the OpenFlow "programming oriented" idiom for specifying a smart and incremental algorithm that schedule the connections. We imagined complexes scenarios that included many external factors in consideration: like the user choice, a machine learning and data sent by each devices of the network.

4.2 Internship outcomes

The purpose of this section is to detail the experience I gained during this internship as well as review my contributions to Technicolor SA.

This internship is the second I undertook during my studies at the computer science department of INSA Rennes. My first internship had taken place at TFTLabs, a renowned french company in 3D interoperability and collaboration in the manufacturing environment. The goal of this software engineer traineeship was to design and create a 3D viewer, connected to a remote gallery, for the next Microsoft's operating system: Windows 8. TFTLabs manages a complex cloud platform for their online services. I carried out this first internship during the summer following my third year at INSA Rennes.

This introduction, during my first internship, to the network engineering as well as my participation in many extracurricular projects involving this domain, led me to apply for an internship position at Technicolor, Rennes. The "Software Defined Network" research mission permitted to confirm my interest for the networking science. This computer field encompasses many other topics that I am passionate about: like security or distributed systems.

I also enjoyed performing research activities, a discipline I did not encounter much in the past. I was first a bit reluctant and disturbed by this schedule-free tasks, without any real clear target to work on. Despite this, the assessment remains positive and I appreciated the creativity required to seek for revolutionary ideas, find a clever solution after long hours facing a problem, or reaching the cutting edge of technology. SDN has been in turmoil for the last few years, but still catch the attention of the scientific community: for instance, creating a stateful OpenFlow protocol to simplify the networks management.

As regards my contributions to Technicolor, I was able to participate in the design process of some innovations at the state of the art. Likewise, I implemented some critical parts of our software solutions by using some remarkably modern tools.

Amongst all our experimentations, a few permitted us to fulfill two invention disclosures that may later evolve into patents. Moreover, the results of our researches will lead to a publication.

Chapter 5

Conclusion

In this report, after explaining the context of this internship and presented the company within which it took place, Technicolor, as well as its research unit, we studied the state of the art in the field of network engineering and more precisely the Software Defined Networking idiom.

As a reminder, Software Defined Networking (SDN) is a new approach to computer networking that emphasizes the separation between the data plane and the control plane. The data plane is simply in charge of the delivery and the forwarding of the network frames, supervised by the control plane, the entity that implements the logic of the network. Physically, the data plane is assured by some highly simplified switches and the control plane by a processing unit.

After introducing SDN, we presented few of its real applications: Network Slicing with an hypervisor, Network Fabrics for the big data centers. We also depicted its main actors, projects and the related technologies.

The rest of this report was devoted to the main implementation of SDN: OpenFlow. This open standard wraps a communication protocol that mainly grants the power to push, pull, remove, and modify some flow rules onto a specialized switch. These atomic rules filter the packets received on each ports and can apply some specific actions as such dropping the packet, modifying the packet or forwarding it.

OpenFlow enhances your imagination when it comes to design a network architecture. Your network become programmable and you therefore benefit from the same facilities as in software engineering.

Finally, we listed the lessons I learnt from this internship as well as my contributions in this domain within Technicolor SA.

Bibliography

[1] Open networking foundation, *White Papers*. https://www.opennetworking.org/sdn-resources/sdn-library/whitepapers.

[2] *The Java Virtual Machine Specification.* http://docs.oracle.com/javase/specs/jvms/se7/html/

[3] *Common Language Infrastructure (CLI).* http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf

[4] Google. *Google Statistics.* Retrieved in May 2014. https://www.google.com/intl/en/ipv6/statistics.htmltab=ipv6-adoption

[5] *Amazon Elastic Compute Cloud (Amazon EC2).* http://aws.amazon.com/ec2/

[6] *The Opendaylight Project*. http://www.opendaylight.org/project/why-opendaylight

[7] *Open vSwitch features*. http://openvswitch.org/features/

[8] *NFV and SDN: What's the Difference?* http://www.sdncentral.com/technology/nfv-and-sdn-whats-the-difference/2013/03/

[9] *OpenFlow Switch Specification. Version 1.3.0*. June 25, 2012. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[10] *OpenFlow: Enabling Innovation in Campus Networks* http://archive.openflow.org/documents/openflow-wp-latest.pdf

[11] *FlowVisor: A Network Virtualization Layer*, Rob Sherwood et al. Technical report, 2009

[12] PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric http://cseweb.ucsd.edu/ vahdat/papers/portland-sigcomm09.pdf

[13] *The pyretic language*. http://frenetic-lang.org/pyretic/

[14] *The ryu framework*. http://osrg.github.io/ryu/

[15] *OpenFlow 1.3 Software Switch*. https://github.com/CPqD/ofsoftswitch13

[16] Mininet, *An Instant Virtual Network on your Laptop* . http://mininet.org/

Résumé

Ce rapport résume mon projet de fin d'études effectué au sein de Technicolor en tant qu'élève ingénieur au département informatique de l'INSA de Rennes. Au sein du département Recherche et Innovation de cette entreprise, je me suis intéressé à un nouvel idiome dans les réseaux de télécommunications dénommé Software Defined Networking (SDN). Je me suis plus particulièrement concentré sur l'utilisation de ce nouveau concept dans les réseaux locaux.

SDN abstrait les méchanismes des réseaux par une couche programmable, qui permet entre autres l'expérimentation de nouveaux protocoles de communication. Ce concept relativement révolutionnaire a été rapidement adopté par les principaux acteurs des télécommunications, comme par exemple les fournisseurs d'accès ou les centres de traitement de données.
Le but de mon stage consistuait à étudier les applications possibles de SDN pour les réseaux domestiques. L'ensemble des expériences effectués sur ce sujet prouvent sont utilité dans le cadre domestique : les innovations possibles vont de la connection entre deux réseaux distants à l'aggregation d'accès internet.

Abstract

This report summarizes my final internship in Technicolor as an engineering student in the computer science department of the INSA Rennes, France. Within the Research & Innovation entity, I focused on the new network engineering idiom, Software Defined Networking (SDN), applied to the home networks.

SDN abstracts the network architectures into a programmable layer, that enables researchers to run experimental protocols. This new concept has been adopted by the Internet Service Providers, the big data centers and the multinational corporations specialized in Internet-related services.
Technicolor and more precisely my research team was wondering if SDN could be also integrated into a home network ecosystem. In order to answer this question, multiple experiments has been conducted around this topic. Ranging from the intercommunication between two home networks, to some Multi-WAN routers, these experiences definitively proved SDN's capacity to ease the network management in a household.

INSA RENNES

Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coëmes CS 70839 F-35708 Rennes Cedex 7
Tel : 02 23 23 82 00 - Fax : 02 23 23 83 96