Software-Defined Networking

THOMAS PARADIS

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2014

TRITA-ICT-EX-2014:5

Software-Defined Networking

Thomas Paradis

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

20 January 2014

Examiner: Professor Jim Dowling

Abstract

Software Defined Networks (SDN) is a paradigm in which routing decisions are taken by a control layer. In contrast to conventional network structures, the control plane and forwarding plane are separated and communicate through standard protocols like OpenFlow. Historically, network management was based on a layered approach, each one isolated from the others. SDN proposes a radically different approach by bringing together the management of all these layers into a single controller.

It is therefore easy to get a unified management policy despite the complexity of current networks requirements while ensuring performance through the use of dedicated devices for the forwarding plane. Such an upheaval can meet the current challenges of managing an increasingly dynamic network imposed by the development of cloud computing or the increased mobility of everyday devices.

Many solutions have emerged, but all do not satisfy the same issues and are not necessarily usable in a real environment. The purpose of this thesis is to study and report on existing solutions and technologies as well as conceive a demonstration prototype to present the benefits of this approach. This project also focuses on an analysis of risks posed by these technologies and the possible solutions.

Acknowledgments

I would like to express my deep gratitude to M. Jean-Marc Lambert, my internship supervisor at Gemalto, for his patient guidance, enthusiastic encouragement and useful critiques of this work during those six months. I would also like to extend my thanks to the members of the Gemalto Meudon Cloud Computing team for having kindly integrated me within their team and all other Gemalto employees for their warm welcome and their sympathy during my master thesis at Gemalto.

Special thanks to Jim Dowling for having accepted to examine my thesis.

Finally, I wish to thank my family for their support and encouragement throughout my study.

Contents

List of Figures

List of Tables

Introduction

The management of networks have to face new challenges [36], either in network policies (rights management, Access Control Lists (ACL) or isolation between tenants), security (new threats) or even in traffic management (Quality of Service). In the same time, networks are becoming more dynamics due to Cloud Computing, BYOD policies or new middleboxes (Firewalls, IDS) needed by the network administrators.

However, networks lag behind and use still old layered technologies with vendor-specific interfaces. Thus, it is complicated and error-prone [73] to manage the network. Based on the idea of a centralized management [11, 10], Software Define Networking (SDN) addresses this problem by decoupling the data plane and the control plane [37].

The principle of SDN is simple: a control layer will solve current network requirements as specified by the network administrator using recent and efficient technologies and will transmit basic computed rules through the data plane layer which can be, for example, simple switches with a standard interface like the OpenFlow protocol [51]. This protocol allows the control layer to specify rules (from OSI Layer 2 to 4: Ethernet, IP, TCP/UDP...) about how the switch will have to handle packets. Therefore, the control plane will be able to use high level resources (like databases or LDAP), work on standard servers (or virtual machines) and be easily modifiable, ensuring flexibility and scalability while performances are guaranteed by the use of dedicated hardware for the switches [42].

As a consequence, SDN address current expectations by providing a flexible and programmable network [37, 30] which can receive information from switches, users, administrators and also security devices and use them in order to update the current policy according to defined rules which can now use sophisticated functions. Currently, SDN and OpenFlow start to become widespread, not only in research works but also in industry for production deployment [28].

This idea is complementary with the growing importance of virtualization. Since everything is virtualized, it makes sense to try to virtualize the network appliances and bring an ease of management. This is the purpose of Network Functions Virtualization (NFV). It is designed to speed up innovation since tests and deployment would be easier and increase openness since such appliances would be software designed. NFV is complementary with SDN since existing virtualized resources can be used for both the control plane (CPU, storage) and the data plane (virtual switches). Moreover by reusing existing hardware and separating requirements, substantial savings can be made [28].

Thus, this thesis project is focused on the analysis of the current state of SDN and its pro and cons for current networks. After focusing on its implications, consequences, security risks and potential improvements, I will mock-up a simple controller which can demonstrate some advantages of SDN.

Goals

The goals of this master thesis can be described as follows:

1. Analysis of the benefits of SDN compared to a traditional network
2. Study of the risks associated with this technology
3. Development of a demonstration prototype to expose the benefits of SDN in some use cases.

Structure of this thesis

This report is organized as follows:

- Chapter 1 presents the background of Software Defined networking (SDN) in current networks, its advantages and some noteworthy SDN controllers.
- Chapter 2 consists of a risk analysis and some use case in which SDN can be used to improve network security.
- Chapter 3 describes the requirements expected for my demonstration prototype.
- Chapter 4 covers the technical choices and the details of the implementation of my controller.
- Chapter 5 discusses of the results provided by my demonstration prototype.
- Chapter 6 suggests some future work following this thesis.

Chapter 1

Software Defined networking

By decoupling the control plane and the data plane (which is responsible for forwarding packets), Software Defined networking (SDN) [68] allows us to abstract the underlying infrastructure and program the network. In this chapter, after specifying the current needs, I will present the concepts of SDN, its benefits and the various current implementation.

1.1 Principle

This section will be dedicated to presenting the needs which led to the SDN principles and the advantages of this technology.

1.1.1 Current state of networks

1.1.1.1 A need for programmable network

In the early days of network engineering, we would just have to bother with simple Ethernet or IP rules and the network was static, but today, network requirements are becoming more complex [36, 73] since networks are becoming more and more dynamic and new concepts to dissociate control and data are developed [11, 10] to address these limitations.

Indeed, networks have to be responsive and elastic since many computers are connected, moved or disconnected due to BYOD policies, Wireless networks and on-demand service of Cloud computing for example. Moreover the associated complicated policies, either in security (e.g. isolation between tenants, firewall rules, access lists and rights management) or traffic management (Quality of Service, load balancing) have to be set up accordingly. Networks are also becoming bigger and different networks should coexist (different security levels in enterprises, multiple tenant in cloud computing) and have customizable network policies.

In the same time, more high-level and network devices (switches, routers and middleboxes) are needed and these devices have to be configured separately and often come from different vendors (and therefore use vendor-specific commands). Finally, network engineers have to face more and more complexity in routing [36, 7] and flows management since circulating flows change and evolve quickly but also

since new threats have to be taken care of. In order to do that, they can be helped by various middleboxes (IDS, Firewalls, NAT) which have to be place adequately in this ever-changing network.

Therefore it is difficult to manage networks and enforce new policies efficiently since each device operates on a given network-level protocol and needs to be configured appropriately with each other. So, it is not easy to add a rule because there is no standardized way to do it but also because it may conflict with one of the many devices on the way [73]. In addition, the rules were originally intended to be static and therefore the network is static and poorly adapted to current needs and technologies.

1.1.1.2 Importance of virtualization

Another important factor is the growing importance of virtualization which is based on the principle of decoupling a service from the physical layer on which that service operates. Today, everything is virtualized: not only servers but also storage and applications. The advantages are clear: easy replication, ease of management and standardization of resources — overcome the hardware specificities to provide an abstraction —. However, networks still lag behind, but solutions begin to exist (e.g. virtual switches).

They are based on the idea that network equipment should be able to understand simple standardized rules, follow orders and run everywhere. Just like server virtualization reproduces virtual CPU and RAM, network virtualization software reproduce logical switches, routers (Layer 2 to Layer 4 in the OSI model) or even firewalls (Layer 4 to Layer 7). Thus, they can run on commodity hardware which is already available in datacenters or, in the case of virtual switches, directly into the hypervisors next to the virtual machines they will be connected to.

1.1.2 SDN: decoupling the control plane and the data plane

Based on the idea we can easily control switches, Software-defined networking (SDN) introduces the idea of decoupling data plane and control plane (figure 1.1). The control plane is the system which makes decisions where the traffic should be sent whereas the data plane is the system which takes care of forwarding the traffic.

The role of the data plane is simple: given a packet, it should transmit it to the next hop following some simple — and if possible, generic — rules (from layer 2 to layer 4 in OSI model: Ethernet, IP, TCP and UDP...). Thus, we can ensure performances — a critical point in networks since they have to deal quickly with huge amount of data without important delays or packets losses — by using specialized hardware like FPGA [42] or ASICs (application-specific integrated circuit). Section 1.2 will discuss how transmit these simple rules to these devices.

The control plane can now only take care of transmitting these low level rules which are computed from unified and complex high level policies. By doing do, the control plan can use high level resources like advanced rights management with databases or LDAP, and use existing resources by working on standard servers or virtual machines. Some examples of controllers are detailed in Section 1.4).

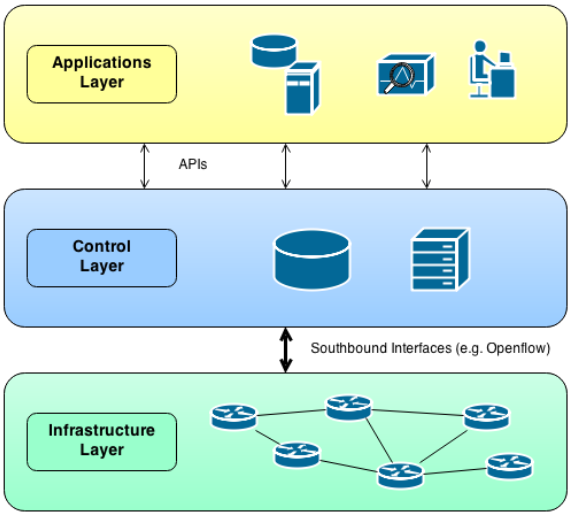


Figure 1.1: Software-defined networking: A 3 layers architecture

The controller can also adjust these rules according to dynamic needs, new policies, network events or alerts. In order to be informed of such events, the controller will expose an API, detailed in Section 1.3.

1.1.3 Advantages

1.1.3.1 Programmable Networks

With SDN, it is simpler and less error-prone to change network policies since we just have to change a high level policy and not multiple rules in diverse network equipment.

Moreover, reaction to a network modification (a new inter-connection or a new device for example) or a network event (like an attack or a suspicion of intrusion) is easy and efficient since we can easily alter the whole network to take this change in order. Moreover, this can be done as close as possible to the source and avoid a useless load in our internal network. Before SDN, we would have to manually alter rules which can lead to errors or slow reaction times.

Finally, the centralization of the logic in such a fully customized controller with global knowledge and high computing power simplify development of more sophisticated functions.

This ability of programming the network is the key element of SDN.

1.1.3.2 Flexibility

SDN also brings a high flexibility in network management. It becomes easy to re-route traffic, inspect particular flows, test new policies or discover unexpected flows for example since the controller will adapt the low-level rules to specified requirements. Thus, SDN is a great tool for testing new routing algorithm or protocol and for simplifying network security as well as malware detection.

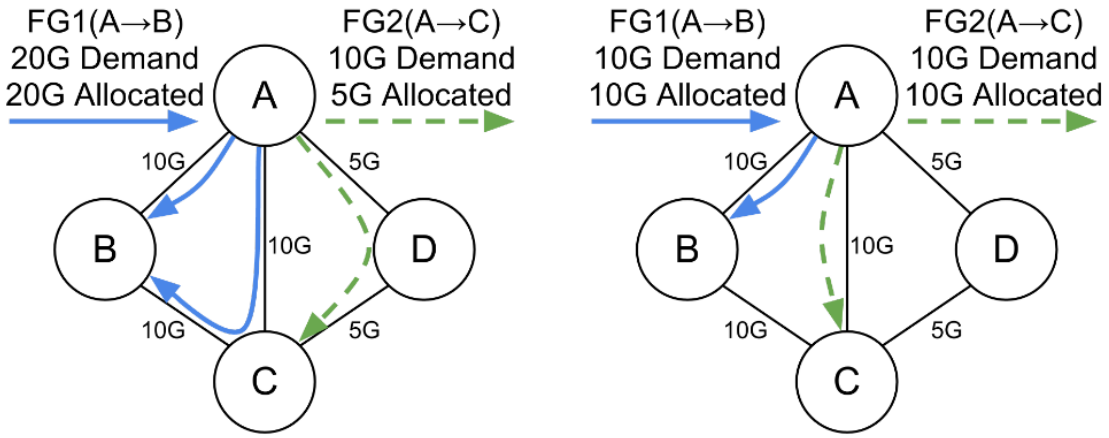


Figure 1.2: Google's B4 traffic engineering. After defining a Flow Group which is a group of applications (for scalability) described by a tuple: (source site, destination site, QoS), the flows are routed accordingly to the flow group bandwidth demands and relative priority between them. The image is taken from B4: Experience with a globally-deployed software defined WAN [28].

1.1.3.3 Unified policy

With its controller, SDN also guarantees a unified and up-to-date network wide policy: since the controller is responsible for adding computed rules in the switches, there are no risks that a network operator forgot a switch or install incoherent rules between devices. Indeed, the operator will just specify a new rule and the controller will adapt the configuration to send coherent rules in every relevant device. Moreover, if a device is moved, the old rules can be removed and the new ones computed and dynamically added. Thus, it is easy to deploy a unified policy but also to add intelligent automation in the network.

1.1.3.4 Routing

SDN can also be used to manage routing information in a centralize way [20, 7, 71] by delegating routing and using an interface for the controller. Some solutions are currently under study: RouteFlow [43, 60] or the IETF workgroup on i2rs [25] for example.

Manage routing through SDN can be used to finely manage the use of links and thus improve performance by increasing the average use of each link without totally saturate the link since it would be counterproductive. Google with its B4 deployment [28] and its traffic engineering over SDN (figure 1.2) was able to get a link utilization 2 to 3 times more efficient than standard techniques by analyzing application needs and adapt packet routing consequently.

1.1.3.5 Cloud management

SDN also allows an easy management of a cloud platform. Indeed, in such environment, separating the control plane with which customers and systems

interact via its API and the forwarding plane is needed and the dynamicity brought by SDN addresses cloud specific problems like scalability, adaptation, instantiations or movements of virtual machines and isolation [3].

For example, OpenStack Neutron (previously Quantum) [52] already provides "networking as a service" and is planned to be integrated with an SDN controller, OpenDaylight [48] — Neutron will use the OpenDaylight northbound interface to manage the network — to work in an SDN environment. Another project, FlowN [17] provides a framework to isolate tenant while providing for each one the ability to program the network: each tenant will be able to run its controller application — allowing the user to personalize it if needed or just use the default controller which will provide all-to-all connectivity (similar to Amazon EC2) — and manage its network. To do it, FlowN intercepts each network request and distribute to the right tenant's controller based on the ingress port of the packet.

In order to provide network isolation for thousands of tenants and servers, solutions like NVGRE (Microsoft: tunneling layer 2 packets over L3), VXLAN [35] (a Cisco technology supported by Open vSwitch: encapsulating Ethernet frames into UDP packets), Nicira NVP (L2 over IP managed by a control plane) or Amazon EC2 (IP over IP managed by a control plane) address this problem but a common pattern is that we cannot scale without control plane [27].

1.1.3.6 Hardware simplification

SDN tends to use standard and basic technologies to control network equipment, while the computing power is only required at the controller. Thus, current switches/routers do not need to provide many features and capabilities deductions. They must just quickly enforce the orders specified by the controller.

These features can be performed by specialized equipment only offering a vendor-agnostic interface. Since all switches will have the same purposes, there will be less hardware distinction since they will offer fewer proprietary features (which will go into the controller). Thus network equipment will become low-cost commodities [28] offering standard interfaces. With such hardware, it would also be simple to add new devices — since they are not specialized —, connect them to the network and let the controller manage them accordingly to the defined policy. Thus, the network will easily become scalable as soon as the controller is scalable.

1.1.3.7 Hybrid deployment

SDN is now a mature technology which is used in real networks [28] with great success and important performances gains. Google with its B4 deployment chose to use custom cheap OpenFlow [37, 50] switches.

Since SDN assumes a full deployment of SDN switches, it cannot be deployed easily in all enterprise networks — which can still have legacy switches and may not upgrade all their hardware equipment in one time — but a transitional development is possible [32] in order to take advantage of SDN possibilities.

Unlike typical deployment patterns — the SDN network is placed alongside the existing network or the network is separated into slices that will be shared between

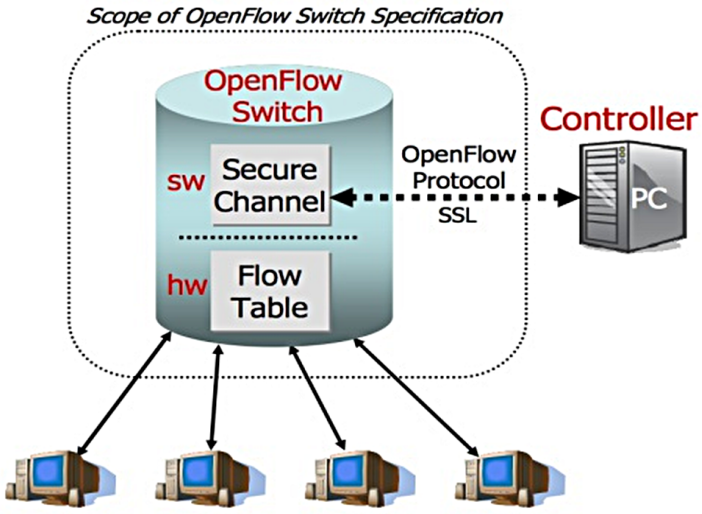


Figure 1.3: The OpenFlow switch. The image is taken from OpenFlow: enabling innovation in campus networks [37]

the legacy network and new SDN networks [37] —, Panopticon [32] provides a way to deploy SDN in the network by making the two technologies cohabit: it will install SDN switches on the data paths and even if a packet could not only use SDN switches, it will at least be transmitted by one. The counterpart is that it is difficult to filter a packet according to its origin, we cannot act at the source and we cannot apply global policies.

1.2 Southern Plugins

As seen in section 1.1.2, the controller may rely on different plugins to communicate his orders to the forwarding plane. These rules must be simple so that the switches (physical, virtual and possibly other network devices) can understand and deal with it in an effective manner.

1.2.1 OpenFlow

The most commonly used technology is OpenFlow [37, 51]. This is an open source standard communications interface conceived to control the network.

It is based on an Ethernet switch with an internal flow table and a protocol to manage this table. The basic structure is shown in figure 1.3.

1.2.1.1 The OpenFlow Switch

An OpenFlow switch is composed of a flow table (described below) which contains information about how to process a packet, a Secure Channel that connects the switch to the controller through a TLS/SSL connection and the OpenFlow protocol which provide an open and standard way for a controller to communicate

with a switch. The Secure Channel can be used to transmit modifications of the flow table from the controller to the switch but also packets and flows information from the switch to the controller. Thus, the controller can analyze a packet which has not been matched by the flow table and create an adequate rule for this flow or analyze current traffic on the network.

1.2.1.2 The flow table

There is below a simple and quick overview of OpenFlow capabilities. More detailed information on the features provided by the different OpenFlow versions is available in Appendix A.

An entry in the flow table is composed of multiple fields [50, 51]:

- The match fields: the ingress port and a packet header which describe the packet: it may use information from OSI Layers 2 to 4 (Ethernet, IP and TCP/UDP) as well as wildcards if needed in order to provide adaptability [16]. The available fields depend on the OpenFlow version and are detailed in Appendix A.
- The flow priority: the match with the higher priority will be used in order to determine the action.
- Counters which will store statistics on the flow (how many packets/bytes already matched this flow). They can be used to detect, amongst other things, useless flows or abnormal traffic. There are also counters related to physical ports, table and queue.
- The timeouts of the flow that defines when the flow is removed from the table. There are two types: the *idle timeout* defines the acceptable time-frame of inactivity of the flow (after this period, the flow will be removed from the table) and the *hard timeout* which defines the maximum lifetime of the flow, whether or not it is still used.
- The action:
  - Forward this packet to a given port — or duplicate it and forward it to multiple ports — or flood it.
  - Encapsulate the packet in an OpenFlow packet and forward it to the controller for further analysis and possibly that a corresponding flow will be added to the switch by the controller to handle the next similar packets.
  - Drop the packet.
  - Enqueue the packet which will forward a packet through a queue. It is used to provide basic quality of service.
  - Modify packet fields. It can be used to replace VLAN ID, MAC and IP addresses for example.
- A cookie (An opaque controller-issued identifier)

1.2.2 Other plugins

According to the goals of the controller and the features it provides, it may be more relevant to communicate in a different way with the switches or other network devices. It can for example use a different communication protocol — like a vendor specific protocol — to reuse existing hardware which does not support OpenFlow.

Thus, a controller can use a proprietary switch interface or use other technologies such as BGP. In the latter case, it will no longer manage flows, but will operate at a higher level like global physical and virtual topologies and may simplify current operations [60, 30].

1.3 Northern APIs

Currently, there are no standards: each controller may expose its own API corresponding to its specificities. This is mainly due to the fact that it is a relatively new technology and particularly that each application has different needs, and an overall API could not provide sufficient granularity.

Theses API are used to communicate with the applications managing the business logic and high level algorithms. These applications can use the controller to gather network information, use them to perform analytics, and then use the controller to orchestrate the new rules throughout the network. They can also be used by legacy security applications, like Snort [66] for example, in order to signal threats to the controller.

1.4 Overview of some SDN controllers

Many controllers have been developed and offer various possibilities. Make an exhaustive list has little interest, but I will try to introduce some noteworthy controllers. They can be classified briefly into two groups. The first group is mainly controllers providing an OpenFlow interface and a few layers of abstraction. The second group will contain more complete solutions.

1.4.1 NOX

Initially developed at Nicira, NOX [47, 23] is the first OpenFlow controller. It is open-source and mainly written in C++ and has been used by many researcher, so a majority of SDN or OpenFlow applications in papers are based on NOX. It provides a developer-friendly environment in which it is easy to modify built-in modules or implement new ones. It is currently on the decline: there were no major changes since mid-2012 [45, 46].

1.4.2 POX

POX is NOX's younger sibling [57]. It is an open-source controller written in Python, and like NOX, provides a framework for developing and testing an

OpenFlow controller. Due to the use of Python, it is easy to implement and deploy new features, but POX performances — although it can be improved by using tools such as PyPy — are significantly lower [65, 18] than those of other controllers and it is therefore not appropriate to enterprise deployment.

1.4.3 Beacon

Beacon [5, 18] is a Java controller known for its stability. It has been created in 2010 and is still maintained and has been used in several research projects. Another advantage is the use of OSGi to manage independently and restart the different bundles. Due to its performance [19, 18], it is a viable solution for use in real conditions. This controller has also been used in other projects like Floodlight or OpenDaylight.

1.4.4 Floodlight

Floodlight [21] is an open-source Java-based OpenFlow Controller forked from the Beacon controller and supported by BigSwitch Networks. It is easy to set-up and shows also great performances. Currently, this is still the most mature OpenFlow controller. With all its features, Floodlight is more a complete solution than a simple controller.

1.4.5 OpenDaylight

OpenDaylight [48] is an industry-supported Linux Foundation project. It is an open source framework to facilitate access to Software-Defined Networking (SDN) and Network Functions Virtualization (NFV). As Floodlight, it can also be considered as a complete solution. The first version, Hydrogen, should be released in December 2013. Although originally supported by many renowned companies in that area, some of these latest are moving away from this project: BigSwitch continues to develop Floodlight [6] and Juniper introduced its own Open Source controller Contrail [14].

Chapter 2

SDN and network security

In this chapter, I will detail the effects of SDN on network security. At first, I will present a risk analysis of using a centralized network controller as well as using a standard protocol to manage the network. Then I will present potential improvements brought by the use of SDN compared to traditional networks.

2.1 SDN inherent problems

As shown in section 1.1.3, the use of SDN solve some problems in network management but adding a centralized management (through the controller) adds new problems.

In this section, I will assume the controller works with OpenFlow (or at a similar level) and act responsively to unknown packets. However, many considerations remain valid if the controller automatically configures the network and all packets are proactively matched with rules in the switches.

2.1.1 A less robust network

SDN, by its specifications, can lead to potential problems. Indeed, the management is gathered in a single point, the different layers constituting the network are merged into one big layer and the diversity of protocols decreases.

2.1.1.1 An ideal target

Since SDN brings a centralized management through the controller, the controller is more is more likely to be attacked. These attacks will not only be more numerous but also further targeted.

Moreover, indirect attacks can be set up if a user station is compromised in order to attack the controller, or add some rules on behalf of the user.

2.1.1.2 Less isolation in network

Another crucial aspect of the changes brought by SDN is reducing the isolation between the OSI Model layers. The basic principle of SDN is to centralize the

management to make it more coherent and simple but, historically, security in networks was improved by the number of layers since layered network architecture can improve robustness by limiting layer interactions [2]. With SDN, this is no longer the case anymore — even if many current solutions also manage multiple layers through the same device —, although it depends on the southbound interfaces used[1].

2.1.1.3 Less diversity in network equipment

In addition to these changes, there has been a standardization of protocols. Thus, as OpenFlow becomes the new standard, a flaw in the protocol or a fault in the implementation can be dramatic since it would affect most networks equipment. Previously, this risk could be mitigated by using different devices coming from different vendors, each with their specificity, making the simultaneous exploitation of different flaws on all these devices very difficult.

In addition, all these network equipment must certify that it is absolutely impossible for a user to directly modify the rules in the devices. OpenFlow solves this problem by using an SSL/TLS channel but, for example, do not specify that the certificates must be checked on the switch as well as on the controller: the controller have to ensure the switch is not a malicious device, but even if the communication between the switch and the controller should be completely isolated, the switch has to ensure the controller identify.

2.1.2 The controller: a critical decision point

Moreover, since we have a centralized management, anyone with access to the controller may control the entire network, thus a bug or vulnerability in the controller would have dramatic consequences and have to be avoided at all costs. Therefore, packet processing, northbound interface utilization and flow rules determination should be completely secure and robust.

2.1.2.1 Southbound interfaces

The verification of data coming from southbound interfaces is particularly critical. Indeed, the information reported by these interfaces corresponds to the network status, statistics and unknown packets. These can therefore be created — by forging packets or generating some traffic — by an user to exploit vulnerabilities in the analysis of the packet but also to provide false information to the controller or flood it through useless information. In addition, the controller must pay particular attention to the identification of switches to prevent a malicious device masquerading as a switch in the network.

[1] A OpenFlow controller will manage OSI layers 2 to 4 whereas BGP will focus on layer 3, for example

2.1.2.2 Northbound interfaces

The controller northbound interface must also be secure and well-protected. We can consider two cases: only a few administrators can access it and add new rules, or everyone should be able to access it, authenticate him or herself and add new rules, for example declaring a new flow for its application which will be accepted if he or she has the required authorizations. An example of such a policy is described in section 2.2.1.

The northbound API should also be well-structured and carefully check flows modifications in order to guarantee that high level flow rules remain consistent.

2.1.2.3 Rules consistency

Finally, the controller has more responsibilities when managing the rules. Indeed, it must only add well-defined rules on switches since it becomes responsible for network isolation (e.g. VLANs) and security. We have to ensure a rule cannot lead to potential exploitation through specially forged packets for example.

2.1.3 The controller: a single point of failure

Alongside all these important needs (many checks on packets, rules and orders), these risks (high attacks frequency and targeted attacks) and therefore a high performances requirement under a heavy workload, the controller remains a single point of failure. Indeed, if it falls or start malfunctioning, the good operation of the network will be directly affected. Thus, it is essential to use a distributed system to manage this controller.

Early versions of the OpenFlow protocol [50] only specified a single controller for the switches, so if the OpenFlow proxy controller like Flowvisor [61] who handled the redirections of switches requests to a real controller — or if there was only one controller — failed or started malfunctioning, the network could be paralyzed or degraded.

Now, the latest versions [51] allow to specify multiple controllers and their role (Master, Equal, Slave), thereby simplifying the management of a distributed controller since requests can be transmitted to another controller directly by the switch[2].

However, heavy work [8, 29] is still required to replicate information and ensure consistency and performance in this distributed system. Indeed, the controller will have to address common problems in such systems as network partitioning and component failures — due to hardware, software problems or an heavy load which can lead to byzantine failure — while ensuring scalability, consistency and availability.

[2]At the switch level, in the control level architecture, the switch's controller may be a load balancer for example

2.1.4 Latency in the network

When the unknown packets are sent to the controller for analysis and then to add the corresponding flows rules in the switch, an incompressible delay is added during the transmission of the first packet due to the round-trip to the controller and the processing time required by the controller.

Assuming this delay is low (a reasonable assumption if the round-trip delay is low and the controller well designed and quick to respond [70]), it is minor compared to the time required to establish a TCP session [9], but it can be annoying for UDP or ICMP packets for example — at least for the first packets, before the switch receive a matching rule for this flow —. A solution would be to proactively setup flows for devices that need to handle many different packets without any delay.

2.1.5 Number of flows

To overcome its obvious performance problems, SDN delegates forwarding to specialized equipment based on the notion of flows. However, the maximum number of flows, although high, is not unlimited and may possibly be reached if packets are specially forged to avoid matching the existing rules.

However, given the capabilities of current switches — the switches can re-use their existing forwarding information base (FIB) to manage Flow tables — and the fact that OpenFlow v1.1 allows to specify several flow-tables — and hence allows multiple medium cheap FIB rather than a single, huge and expensive FIB —, the processing capacity of *specialized* OpenFlow switches is not a problem, but if the controller wants to keep track of these flows, it can become problematic since it does not have the same capabilities and is supposed to aggregate flows for multiple switches.

Indeed, the number of packet it receives will become even more important, and packets may be especially forged to avoid matching an existing flow and flood the controller under a huge number of requests (and thereby subject it to a high load) or useless information. This problem can be mitigated by increasing the number of controllers and load balancing. To protect against this type of attack, these packets must be handled directly on the switch, either by dropping all packets when a certain number of rules is reached, or by installing valid rules which will match a high number of packets [40]. The concern is how define such rules without losing SDN advantages like flow analysis or fine control of flows in the network.

Solutions like DevoFlow [40] address this problem by limiting the number of requests to the controller by devolving control to the switch (they define two new mechanisms: rule cloning and local actions) and using wildcards as much as possible.

AVANT-GUARD [63] proposes a complex mechanism to mitigate SYN Flood directly on switches without doing any request to the controller — and avoid flooding the controller with a lot of possibly useless connections — by using SYN cookies and storing initiated connections on the switches, but even if they present a hardware implementation, there is no comparison with existing and dedicated specialized hardware solutions to clean traffic from the Internet at line rate. Moreover, this only addresses the SYN flood attack and there are many other threats. Thus we

may think that cleaning traffic may be out of scope for SDN switches and that SDN does not address to replace all existing dedicated hardware, just manage network devices in an easier way.

2.2 SDN to improve network security

As briefly presented in section 1.1.3, SDN bring multiple advantages in network management. This section will present some other use cases that SDN can realize.

2.2.1 A multi-level policies system

This section will be dedicated to present a system to manage a multi-level policies system where administrators will setup network global rules and the user will ask for specific permissions.

2.2.1.1 Design

With SDN, it is possible, for example, to deploy multiple network policies. The network administrators will define basic and simple rules and user rights. The users can also connect to the controller and ask for authorizing a flow as long as they remain within the scope specified by an administrator. FortNOX [56] is an extension of NOX which provides a system of hierarchical rules for NOX applications and a conflict analyzer.

Thus, we can improve network security by limiting the number of default rules and still stay productive because the user can declare itself its needs to the controller that will immediately update the network if necessary. The user is not required to wait for the changes to be taken into account by the relevant network equipment — since the controller will push the changes — while network administrators can still keep a close control on these rules. SANE [11] presents a system to hide user services from unauthorized other users for example.

This hierarchical system of rules — which will be converted by the controller in basic rules to network equipment — can also be used to quickly remove permissions in order to react to an attack without jeopardizing critical services which will use higher priorities rules for example.

2.2.1.2 Using cryptography in network management

Since we are operating with a programmable network, many opportunities that were previously impossible or very complicated now become available. Previously, the rules should be inserted on the switches and the necessary checks made beforehand. Now, rules can be specified in a programmable controller, scalable and with a higher computing power than a switch for a more modest cost.

When using a multi-level policy, the rights management can be greatly simplified by using a cryptographic system. Besides aspects concerning authentication — a strong authentication is recommended given the importance of the controller and

the fact that the controller may be easily adapted to such a solution since it is a simple software —, we can also consider a system where rules will be signed to prove their validity (FortNOX briefly explores this idea [56]). This allows us to ensure that the rules have not been altered and came from a certified source (and thus enforcing right management) but also to delegate rules and use systems like OAuth for managing exceptional rights.

Moreover, secure elements like Smart cards or Hardware Security Module (HSM) could be used to protect cryptographic materials, even if the controller — or the switch — becomes compromised.

2.2.2 Custom tags for identifying flows

A nice feature would be to tag the packets of an OpenFlow flow, for example to attest the validity of the flow. A router which encounter an unknown packet could transfer it the controller (or analyze it itself) and, based on the information added in the packet, the controller could ensure the validity of the packet and check that this flow is authorized before choosing to forward or not the packet and install an adequate rule in the switch.

2.2.2.1 Realization with OpenFlow

An analysis on the switch would require — if we want to maintain the performance and not to develop a new switch — to be able to match existing fields. OpenFlow allows matching multiple existing fields, but such a system would require operating at the IP level. Indeed, acting at the Ethernet layer, would impose to keep the Ethernet header all the way, using MPLS would require the user to place the tags himself which is complex, unwise and therefore far from the original idea.[3]

However, it is not possible to tune the IP packet according to our needs, while retaining the ability to match the packet via OpenFlow. Indeed, OpenFlow allows only (for now) to match fixed fields for performance reasons. For example, the options field of the IPv4 datagram could possibly suit us, but it is not possible for the current OpenFlow switches to process it. Similarly, the only matchable IPv6 field (other than addresses) is the flow label [1] which is not suitable for our idea.

Without designing and producing a physical compatible switch, it is possible to adapt a virtual switch (like Open vSwitch [33, 55] for example) which would be able to handle a custom extension of OpenFlow or a specific protocol and parse the options field of IP packets for example. However, it will be at the expense of performance (the option field has variable size) and deployment since the switches will need to work on VMs. So it brings little benefit since this solution will not be widely deployed and yet requires major changes for a limited gain because the analysis of this field must be fast and efficient due to needed processing speed.

[3] It would also create problems when it will flow from the LAN

2.2.2.2 Controller based implementation

An alternative would be to use these fields in the first packet of the flow. This packet would then be transmitted to the controller, which could benefit from this information whether to allow or drop the flow. This is a similar approach (though slightly more complex as it forces the user to modify the IP packet) to the approach consisting of the user requesting new permissions to the controller. The second approach also has the great advantage of simplifying the application and authentication (which, in the packet approach, must be contained in a specific field and therefore less subject to various changes or elaborated processes).

2.2.3 Middleboxes placement and isolation

SDN has the advantage of being programmable. Thus, it is possible to simply specify concepts or rules that are then computed by the controller and sent to the underlying architecture. These adaptations can be complex, for example when considering the separation of thousands of tenants without relying on VLANs (which are limited at 4094[4]), but the principle of SDN is to rely on the controller to generate these rules and limiting the number of errors.

Similarly, this abstraction of the network allows placing middleboxes (such as Firewall, IDS / IPS, or tools for traffic inspection or analysis) in a strategic location of a network and specifying to the controller which flows have to be sent to these middleboxes. Currently, network equipment must be placed in the data path, with SDN, the flows can be automatically routed — at the expense of an increase in traffic, but by wisely placing the middlebox, only a few flows will be concerned by this increase, and the overhead can be low or null — without worrying about the placement of the middlebox or how to route the concerned flows.

It is also possible to use the computing power available at the controller to try to optimize the management of these flows [62] and report potential improvements or irrelevant configurations. In addition, if the network is dynamic, moving a device (laptop, VM in a cloud environment) is no longer a problem: the rules will be automatically recalculated and the flow will still go through the devices as specified by the network administrator.

2.2.4 Security management and reaction

SDN, because of its dynamic and programmable aspect allows dynamic reactions to threats detected on the network. For example, if a machine is suspected of being infected, it is easy to isolate it completely, but also to have a more refined response like redirect its traffic for further analysis or clean it before actually sending it over the network. This makes possible to keep the productivity high while improving security. Due to the programmable aspect of the network, any alternative such as isolate it from critical services and external network is also possible.

Another scenario (an implementation of this scenario using the FRESCO framework is detailed in [64]) is to change the external network aspect when detecting

[4] 0x000 and 0xFFF are reserved values

suspect behaviors. For example, when security elements detect a scan, the potential attacker is reported to the controller that will redirect the related requests to a honey pot voluntarily vulnerable or providing false information. If the attacker attempts to exploit one of these vulnerabilities, it will be reported to the controller which can blacklist this attacker. If the attacker does not seize any opportunity, it may be treated as a normal visitor that triggered a false positive. Some frameworks to deploy such solutions exists [64, 4] but they are still very specific to one use case and one controller.

2.2.5 Conclusion

Such opportunities are attractive, but to use them at full potential, they have to operate on an edge-network using SDN, which can — given the fact that SDN is still young — be dangerous. Indeed, the threats mentioned in section 2.1 can quickly become critical when facing the Internet. Currently, SDN should be still reserved for the well-known networks [22, 28].

Chapter 3

Prototype's specifications

In this chapter, I will describe the expected features of my demonstration prototype.

SDN offers many possibilities. However, given the allocated time, it is impossible to implement all of them while keeping some generality in the realization. So choices had to be made on the features presented by my controller and they were motivated by these criteria:

- Be able to demonstrate the simplicity of management brought by the decoupling control plane and the data plane.
- The benefits provided by SDN compared to a traditional network, especially the management of switches and network topology.
- The potential network security improvements due to a better flow management which is transparent for the user.

3.1 Use cases

3.1.1 Move rules management at switch level

The prototype demonstration will be able to manage the rules nearest to the source.

For example, packets intended to be dropped will have to be dropped on the switch nearest to the issuer and will not circulate through the network to be dropped on a firewall, thus, unnecessary traffic is avoided. In doing so, we also take advantage of the processing capabilities of a hardware switch, thus reducing the load on other devices.

In addition, handling packets on their way out can be used to manage access rights while avoiding going back to a firewall if the machines must communicate on the same LAN. This is particularly advantageous in a cloud computing environment where VMs from different owners — or to ensure sub-networks management into a tenant's network — can coexist on the same hypervisor and communicate through a virtual switch.

3.1.2 Network Discovery

The controller must also be able to discover the network (switches and their relationship to each other, but also the hosts) and can build a topology easily usable.

This will allow us to easily identify hosts, without necessarily worrying about their physical position. The specified policies can then be applied automatically regardless of the physical location, or according to the physical position — if a fixed machine is detected at an abnormal location, an alert will be raised and actions will be taken for example — in the case of machines not designed to move.

Another advantage is the calculation of paths which will be taken by the packets to meet all specified criteria (middleboxes, custom destination, QoS) for a given flow. Finally, a quality of service (QoS) can be easily implemented if the administrator prefers to use some links over others.

3.1.3 Allow flow inspection or redirection

In addition, the controller will allow the network administrator to easily analyze some flows defined according to basic rules.

It will therefore be able to duplicate and route flows to its destination but also to a defined — by the presence of a device or simply via the port of a switch — location of the network. To do it, it may duplicate the flow and separately dispatch it to its destinations, or attempt to route the flow to pass through all the intermediate steps. This inspection will try to overload the network as little as possible.

Similarly, the controller must be able to redirect a flow, that is to say not deliver it to its destination but transmit it instead to a given device. Thus, this middlebox may be able to clean the traffic — e.g. get rid of malicious requests in inbound traffic or dangerous packets in outbound traffic —, modify it to meet a security expectation — for example, encrypting plain-text traffic before that it is sent over the Internet — or verify its validity at an application level before allowing forwarding.

3.2 Technical requirements

By its intrinsic nature of a prototype, the deployment and the demonstration should be easily achievable in a limited environment. Demonstration on a simulator running on a PC or a few PCs and switches would be desirable.

However, even if a test in real condition is not essential, the prototype should be able to run on dedicated hardware — which can support OpenFlow for example — without major modifications. In this case, performances, although desirable, are not the first priority.

Finally, the goal is to demonstrate the possibilities offered by SDN and thus security, performances and redundancy are secondary, although a theoretical study can be helpful.

Chapter 4

Design and implementation of the prototype

This chapter will be dedicated to the description of the technical choices and of the design of my demonstration prototype. Then I will go through the different modules which constitute my controller.

4.1 Technological choices

4.1.1 Choice of the controller

Given the amount of work involved, it is necessary to take an existing controller to avoid wasting time. It should be easy to use, to modify and adaptable to our needs.

However the many expected (and quite specific) features do not promote the use of a complete solution. In fact, the amount of work on the latter could become significant to suit our needs (while maintaining the existing solution proposed by the architecture). A simple controller to easily control the switches would be more appropriate for this prototype.

The control of the switches will be via OpenFlow because it is a widespread technology — and it will be even more in the future — in this area and it can easily control a switch while ensuring performances and fine enough granularity in management. The OpenFlow protocol is detailed in section 1.2.1.

My choice of a simple and flexible solution supporting OpenFlow fell on POX [57]. This controller written in Python meets the requirements state above. Its big disadvantage is not being able to bear the load required by a real environment, but since this prototype is a prototype demonstration, this is not a problem.

4.1.2 Choice of the environment

4.1.2.1 Mininet

Mininet [38, 31] is a system for simulate a full network (end-hosts, OpenFlow switches and links) on a single Linux kernel. It creates virtual networks using

process-based virtualization and network namespaces [34], thus any program that can normally run on the Linux host can be run inside the end-hosts. It also uses very little resources and big networks — it can scale to hundreds of nodes — can be quickly simulated [31, 24]. It is easy to set-up and can be tuned to fit any needs and its inherent limitations [1] are not a problem for this demonstration.

Thus, Mininet is a perfect tool to test the controller locally and simulate a network easily during a demonstration.

4.1.2.2 Physical demonstration

From the perspective of a physical demonstration, I also looked at how to use physical switches with OpenFlow. For a small simulation the idea of using cheap routers supporting OpenFlow is attractive.

So I retrieved a Wireless router (an old Linksys WRT54G) and have installed a custom firmware, OpenWrt [53] patched to support OpenFlow [49]. The OpenFlow version was determined by my choice of POX as a controller, since POX only support version 1.0 of the OpenFlow protocol.

Some specific adjustment for this model had to be done because of its management of its different LAN ports [2] to run OpenFlow.

After getting a functional OpenFlow switch, I realized performance tests. They have shown that it is possible to obtain a throughput of the order of 10Mbps. This is due to the fact that these routers are not made to handle the OpenFlow protocol and therefore any packet management (analysis of the packet payloads and search for the appropriate rule in the flow table) is performed by the slow CPU of the router. So no hardware acceleration is used and everything has to go through a non-dedicated component, leading to poor performances.

This speed is reasonable for small physical simulations but quickly shows its limits. Similarly, all router transformed in the same way into an OpenFlow router will also show performance limitations — related to the computing power of the router's CPU — because the patch uses the router's CPU. Thus, for real performances, it is necessary to use dedicated OpenFlow switches.

As a simulation poses other practical problems such as getting several PCs and routers and deploying it easily, while Mininet offers greater simplicity and better performances, I choose to realize my work using Mininet.

4.2 General architecture of the controller

This controller is an adaptation of the controller POX. POX is a framework for controlling OpenFlow switches and is designed to operate with a system of modules. Therefore, I have decided to keep some of the default modules (communication with

[1] `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#what-are-mininets-limitations`

[2] `http://wiki.openwrt.org/toh/linksys/wrt54g#internal.architecture.-.wrt54g.v2.v3.wrt54gs.v1.v2`

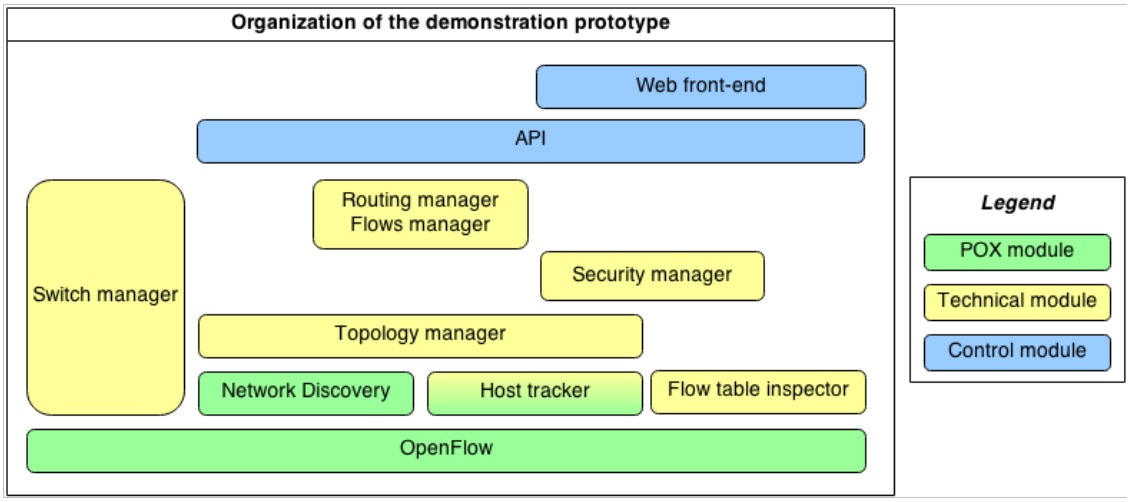


Figure 4.1: Architecture of my controller's modules

the switches, the OpenFlow protocol, and a basic network discovery) and I added my own modules in order to provide the expected features.

4.2.1 Structure of the controller

Figure 4.1 represents the structure of these modules and their relationships with each other. It also shows the origin of each module: a core POX module or a custom module designed for this controller. Custom modules can be divided into two groups: the technical modules which are in charge of handling the network and the control modules which allow rules modifications and information retrieval.

4.2.2 Basic operating principle

When an unknown packet arrives at the controller, the latter analyze and forward it to the relevant modules: the switch manager module, the network discovery module and the Host tracker module. These last two modules use it to get information about the network while the first one will determine how to process the packet and return this information — possibly with a rule to install — to the switch. During the decision process, it will use the other modules: the routing manager, the security manager and the topology manager modules to get relevant information.

When the administrator submit a new rule through the API, it is transmitted to the relevant module that analyze it before taking it into account whether it is compatible with current existing rules. Moreover, insofar as possible, this rule is directly transmitted to the switches for it to be immediately taken into account and that its application occurs as soon as possible.

Regarding the rules installed in the Flow tables of switches, they are relatively simple. In case of a link between switches, it is considered to be safe and packets are routed by simply using the destination address — and in some cases, inbound port

in order to match the link —. For the other ports, any unknown packet is analyzed by the controller, and if authorized, a specific flow rule will be installed on this edge switch to match similar packets.

4.3 Modules description

In this section, I will present each module, its role and its basic working.

4.3.1 Topology modules

Organized into three modules, they are responsible to infer the network topology by analyzing the packets knowingly sent to the controller and the unknown packets — not matched by a rule in the flow table — sent by the switches.

4.3.1.1 Network discovery module

This module — which is a part of the POX controller — can discover the network topology.

It installs rules to redirect LLDP [13] packets to the controller and periodically sends LLDP packets to the switches. Thus, by collecting and analyzing these packets, it can discover the links between switches and build a network topology.

4.3.1.2 Host tracker module

This module is based on an existing POX module. It allows discovering the devices connected to the network as well as their location. To do this, it analyzes the unknown packets — not matched by a rule in the flow table — sent by the switches to extract information on the MAC address and possibly the IP address(es).

With this information, it will signal the appearance of new devices, the move of a device in the network and their disappearance. To ensure its proper functioning, it is supposed to regularly receive packets from each host. This is often the case thanks to traffic between hosts, but to ensure that a host is still running, it will also try to regularly ping it (at the ARP level).

4.3.1.3 Topology module

This module uses the data provided by the previous two modules to establish a network topology and allow other modules to retrieve this information in a simple and concise manner. It also exposes APIs to allow an administrator to retrieve information about devices connected to a switch or the position of a device in the network as well as network links.

4.3.2 Security modules

These modules are primarily responsible for controlling flows circulating on the network.

4.3.2.1 Security Manager Module

This module works as a firewall: from a database of rules — which consultation and modification are exposed through an API — it informs the switch manager module if packets of a flow are allowed or not.

In case of a change of rules in its database, it will immediately propagate the change on all the involved switches. For example, if an address is blacklisted, flows to drop all packets from this address will be installed on all edge switches where the address was recently seen. Thus, blacklisting is effective as soon as the entry is inserted into the database of the security manager.

Since a stream can only be initiated on an edge switch, the unwanted packets are blocked as soon as possible without spamming the entire network. If a blacklisted machine is moved to another switch, as soon as it tries to send packets, the flow will be detected as a forbidden flow and a rule matching the corresponding ban — which may be wider than a specific rule only matching this flow — will be installed on the new switch to drop packets, thus limiting the number of rules and the load of the controller. Finally, if a flow becomes allowed, orders to remove the corresponding dropping rules are sent to all switches which are still having a dropping rule for that flow.

This module allows imposing security policies as close as possible to devices and avoids unnecessary packets circulating in the network or even between two machines connected to the same switch. This is particularly interesting in the context of cloud computing where two machines belonging to two different tenants can coexist on the same hypervisor and can communicate through a virtual switch. By automatically configuring the virtual switch, we can easily and automatically impose isolation policies which will remain valid even if the machines are intended to be moved. Indeed, it is equivalent to a displacement of a device and, as said above, the policy will automatically be reconfigured on the new switch — or the virtual switch in case of an hypervisor — on which the machine has been moved.

4.3.2.2 Flow table inspector

This module operates at a much lower level than the others: it can contact switches and retrieve information about OpenFlow flow tables (as described in section 1.2.1.2).

As they will be served through an API, it is possible to view current rules on any switch of the network and see how the rules in a switch will adapt to various changes depending on the configuration of the controller. This is particularly useful for demonstration purposes.

It can also analyze statistics and discover an abnormal amount of traffic for a flow. Such alerts are reported to the security manager which can optionally alert the administrator or take mitigation measures if necessary. In case of malicious traffic such as a DDoS, it can possibly drop or redirect it to a traffic cleaner to keep only the legitimate traffic for example.

4.3.3 Routing & Flows manager

This module is responsible for routing packets. It relies on the topology to calculate the paths and is used by the switch manager module when the latter receives a packet which has been authorized by the security manager module. It also exposes an API to allow configuration of flows which have to be redirected or inspected.

Redirected flows will not be routed to their normal destination, but will be brought to the address which has been specified during the configuration. This address may be the IP address of a device or a port of a switch. Regarding inspected flow, they will be duplicated at the right time — thus avoiding overloading the network with redundant flow — to be sent to several place of the network: the initial destination and to the devices involved in the inspection of this flow. Thus, a flow can be both inspected and redirected if necessary.

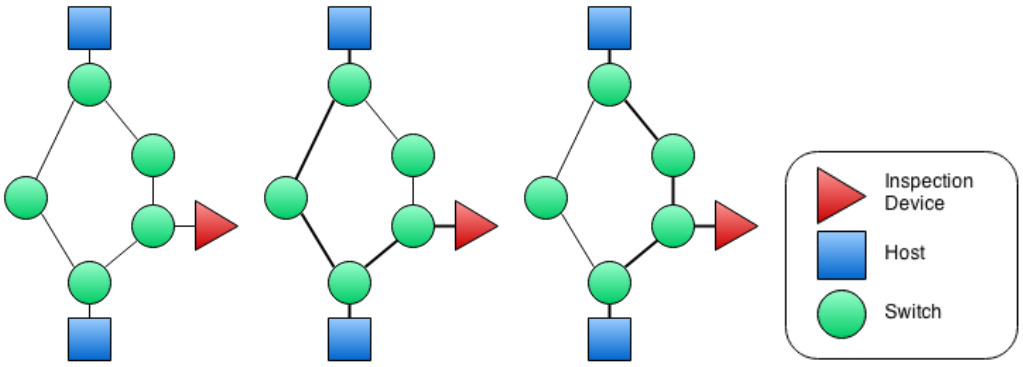
Both redirection and inspection processes have the particularity to not alter packets — especially the source and destination address are not affected by the routing and packets are not encapsulated —: the packets reaching the redirection or inspection port are exactly those emitted by the source. Then, they may be analyzed for different purposes. In addition, as the rules are added automatically and dynamically, if a device moves or a new flow is affected by a redirection or an inspection rule, appropriate rules will be automatically set up — as soon as a relevant packet is sent to the controller because there is no matching rule for this packet into the flow table of the switch — for routing this flow to its destination(s) in an efficient way.

4.3.3.1 Flows inspection

For example, if the traffic must be inspected, a probe can be placed in the network and by specifying the analysis criteria and the location of the probe to the flow manager module, affected packets will automatically be sent to this location. If the probe is on the packets path (or a close path as shown in figure 4.2), packets will be routed directly. If the probe is not on the packets path, they will be duplicated by the switch which will be the branching point of the different paths. This principle can also apply to all security devices analyzing traffic to detect threats such as the Intrusion detection systems (IDS) or the Intrusion prevention systems (IPS).

4.3.3.2 Flows redirection

The traffic can also be accurately redirected — always by using the layers 2 to 4 of the OSI model to specify the matching rules — to pass through security devices before going out. This is, for example, the case of an application firewall or a proxy that can filter a part of the traffic in a complementary way to the firewall module. This feature can also be used to encapsulate sensitive traffic into a tunnel before sending it to a more exposed network.



(a) Base topology (b) Basic routing (c) Paths merging when profitable (d) Legend

Figure 4.2: Path calculation in routing manager

4.3.3.3 Routing

Path calculation is performed via the Dijkstra algorithm. It relies on the topology information and as links can have a greater or lesser weight (specified by the administrator to the topology module), the paths taken by packets may therefore favor some links and avoid others. When there is a need to duplicate packets, the module will try to minimize the network load by favoring the use of a common path as long as it is possible and not too long compared to duplication (figure 4.2 represent this problem).

4.3.3.3.1 Proactive routing

Once the path is obtained, it will be used to proactively install rules within switches and avoid delays and unnecessary steps. Indeed, without this, a rule adapted to the current flow and the destination for the current packet would be returned to the switch, which would set up his table and forward the packet. Then the next switch would receive an unknown packet, transmit to the controller which would have to check the validity of the packet again and repeat the same steps. In addition to being heavy, it's complicated because we cannot easily get the original ingress port from which the packet originally came. So after determined the path or paths to be used by the packet, the rules specific to this flow are proactively installed in the concerned switches. Then the rule is installed on the initial switch and the packet is transmitted through the network without ever being sent to the controller, guaranteeing quick forwarding for this packet and the other packets of the flow.

4.3.3.4 Immediate rules application with security rules

Unlike the firewall module, it is more complicated to apply a new policy immediately when adding a new inspection or redirection rule.

Indeed, OpenFlow 1.0 [50] can only be used with a single flow table. Thus, if

only one rule can match a packet — the matching rule with the highest priority —, in order to duplicate a packet, it is necessary to install a rule forwarding the packet to two or more output ports and thus duplicating the packet. Adding a forwarding or inspection rule therefore impose to store all the rules set up in the switches in order to know the other ports the packet has to be forwarded to. In addition, adding an inspection rule also need to store flow paths calculation — since they cannot be the most straightforward for optimization reasons and depend of the topology and the position of inspection devices at the time of calculation — to be able to update the schema. Finally, certain rules should be applied to all switches in the edge (if there is no information about the source, for example). It is very heavy and can lead to many unnecessary rules and an overload of switches. Thus, the implemented solution is — since the flow is assumed harmless if the application is not done immediately but with a little delay, otherwise it would have been blocked by the firewall — to leave the existing flows expire due to the OpenFlow timeouts and install the new policy when renewing the rule. The delay of response is therefore tied up to the timeout delay of the rules if an existing rule already exists.

OpenFlow 1.3 [51] solves this problem, as it is possible to get a packet through several flow tables, each one adding an action and those actions being applied to the end. In this case, we can imagine a table dedicated to bringing the packet to its destination or do nothing, and another table to duplicate the packet for routing it to an inspection device if necessary. So, adding an inspection rule would just be the same as update this table or use other tables.

4.3.4 Switch manager module

Veritable orchestrator, its role is to inform the switch on the action to be taken when processing a packet which has not been matched by the flow table.

When a switch sends an unmatched packet to the controller, this packet will be distributed to several modules — like the topology modules which need it to infer information about the network — but it is the switch manager module that will be responsible for responding to the switch. This response will consist of the identifier of the packet in the switch buffer as well as an action to perform and possibly a new rule to add in its flow table. This rule is traditionally a rule matching packets belonging to the same flow (or possibly wider) as the concerned packet to avoid further controller calls in the near future.

The switch controller module will first ensure that the packet is allowed to travel on the network. If this is not the case, it will drop it and install on the switch a rule to drop all packets matched by the firewall rule that leads to drop this packet. This prevents installing many rules on the switch and the switch doing unnecessary queries to the controller.

4.3.4.1 Location protection

If the packet relates to a MAC address that is not authorized to be connected from this place on the network, a flow may be installed to drop all traffic from that

port — if this port is dedicated to host a single device — or just all traffic from this MAC address.

4.3.4.2 ARP management and networks isolation

If this is an ARP packet and ARP traffic analysis is enabled, it will check that the pair (MAC address, IP) is authorized and add a rule to allow this pair in the future. If it's a forbidden couple, rules will be installed to allow only valid couples for this port and drop the rest of the ARP traffic. As authorized couples correspond to rules with a higher priority, authorized ARP packets will be transmitted. However, unauthorized ARP packets will be dropped by a generic rule matching all ARP traffic on that port. Packets from other switches will not be affected by these rules because they are supposed reliable since they were previously authorized upon their entries into the network. By doing so, we can limit IP address spoofing and other attacks on ARP protocol as *ARP spoofing*. Of course, this does not protect against MAC spoofing, but for that, we can check that MAC address is allowed to connect from a defined location.

The controller may then proceed in two ways:

- Flood ARP packets on all ports — currently the loops are avoided by a native POX module (the `spanning_tree` module, even if its functioning has not much to do with the protocol of the same name) which will deactivate the problematic ports during network discovery — but we can also only flood it on some ports according to the ingress port in order to allow multiple tenants to coexist on the same VLAN while providing isolation for cloud computing purposes for example.

- Use the information available in the topology module and if it is recent enough, forge a packet to answer directly the request. This feature can also be used to announce false gateways for example.

On one hand, forge an ARP response has the advantage of speeding up the process — since packets and different rules do not need to be installed and transmitted across the network — and it allows to easily check the ARP traffic of a machine. However, this adds to the load on the controller and this prevents to monitor the ARP responses to detect changes in the network and potentially a change in the topology. On the other hand, flooding a packet induces more traffic — even if it is often not a problem since it does not represent a huge part in the traffic — but special attention should be paid to the selection of ports that will be flooded as they may relate to different tenants or other networks which have to be isolated.

To achieve this separation, VLAN can be considered if they are managed by the controller. Indeed, they are limited in number and use them globally on the whole networks would be limiting — VXLAN [35] was developed to address this problem by encapsulating Ethernet frames into UDP packets — but if VLAN are used to manage isolation at a switch level and VLAN management on the whole network is done by the controller — a VLAN on one switch may have nothing to do with the

same VLAN on another switch —, this solution would scale and the complexity will be managed by the controller. Thus, it would be easy to flood the network with ARP packets since the association of the VLAN and the switch would correspond to a single network.

Other solutions are presented in section 1.1.3.5.

4.3.4.3 Routing and generic rules

If the packet is allowed by the firewall, it will be transmitted. The packet will be analyzed and its destination will be computed using the routing manager. As shown in the description of the routing and flows manager module (section 4.3.3), the routing manager module will analyze the redirection and inspection rules applicable to this flow, compute a suitable path, install proactively the rules on the downstream switches to reduce unnecessary queries to the controller by these switches and finally send the next destination(s) to the switch manager module. If these ports are not likely to cause a loop — the input port is the same as one of the output ports, most likely due to an error in the topology —, a rule is designed to represent accurately the flow and installed on the switch.

By installing a detailed rule on the switch, we ensure that another flow from the host — and potentially with different policies (security, redirection or routing) — is not matched by this rule. Thus, each flow will be assigned the tailored policy concerning itself. However, to avoid a proliferation of rules on other switches, we only apply this granularity on the edge switch; the others will have generic and simple routing rules either for traffic between hosts or routing packets to inspection devices.

However, to ensure that inspected or redirected flows are properly forwarded to their destinations and do not match generic routing rules for inter-hosts routing as they travel through the core of the network, additional changes are needed to route inspected packets to inspection devices. A specific rule corresponding to the inspection rule and specific to each flow is installed with a higher priority in all switches where the flow must be duplicated. Indeed, since OpenFlow 1.0 does not support multiple flow tables, duplication should be performed by a single table, and thus a single rule. OpenFlow 1.3 addresses this problem by using multiple flow tables.

Finally, to allow updates of the rules, to refresh data of topology modules and to remove rules that no longer serve, these rules — with some exceptions — have a limited lifetime (which depends on their type) implemented with the timeouts provided by OpenFlow. The longer this timeout is, the less the controller will be called upon but the insurance to have rules exactly corresponding to the given instructions sent to the controller will be lower. In addition, information obtained by the analysis of unknown packets sent to the controller may become old and therefore obsolete.

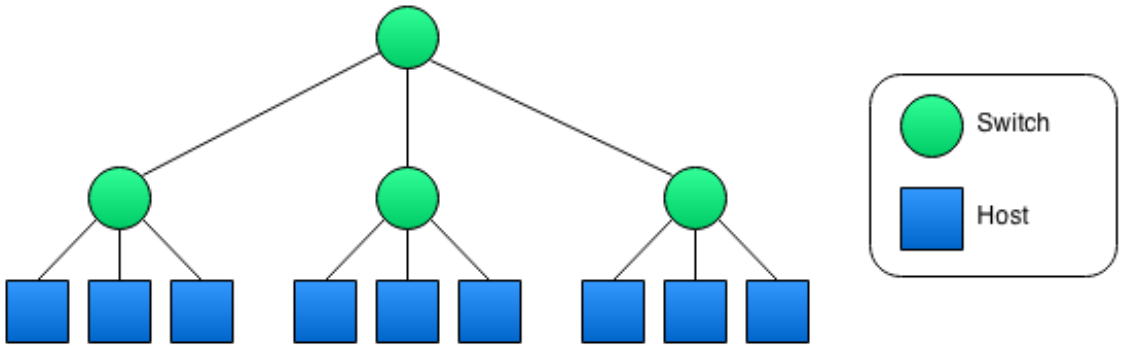


Figure 4.3: Tree topology with a fan-out (number of child of each switch) of 3 and a depth (number of switches layers) of 2

4.3.4.4 Advantage of generic rules over detailed rules

The benefits are important since the number of rules will tremendously decrease for medium or big topologies. For example, with a tree topology (a representation of a tree topology is visible in figure 4.3) each host establishes a connection with each other — we suppose there are not inspecting or redirecting flows —, we have for the core switches (not directly connected to an host):

$$\begin{aligned} N_{rules} &= N_{routing\ rules} \\ &= N_{final\ connected\ hosts} \\ &= fanout^{depth} \end{aligned}$$

And for the edge switches:

$$\begin{aligned} N_{rules} &= N_{hosts\ on\ the\ switch} 2 N_{outgoing\ connections\ per\ host} + N_{routing\ rules} \\ &= 2fanout(fanout^{depth} - 1) + fanout \\ &= fanout^{depth} - fanout \end{aligned}$$

Figure 4.4 presents results for a tree topology with a *fanout* of 5 and a *depth* of 3 (so $\sum_{i=0}^{depth-1} fanout^i = 31$ switches, $fanout^{depth} = 125$ hosts and thus 124 connections per hosts). As we can see, on the edge switches, the number of rules is pretty much the same than the switches with no generic rules since the installed rules have an exact match in order to filter the flows and do not allow other flows to pass through these rules, hence ensuring security and control.

However, for the other switches, which are not connected directly to the hosts but have an important traffic since they represent the core of the network, generic rules greatly reduces the number of rules. This has no influence on security since the flows are still checked by the edge switches, but with this system, the system can scale — and avoiding saturating the flow table of a switch too rapidly — since it will scale linearly with the number of hosts and not with the number of connections.

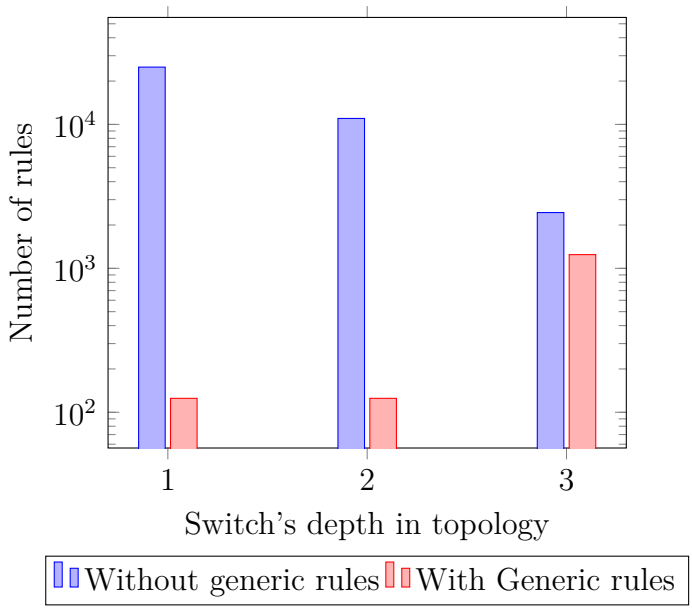


Figure 4.4: Influence of generic rules for a topology in tree (*fanout*=5, *depth*=3)

4.3.5 Webserver modules

This part is devoted to the description of the API and the webserver which allow control at several levels.

4.3.5.1 Control API module

As noted above in the description of the different modules, each module can expose its API to allow various changes in its configuration or information retrieval. This module is in fact a generic module that is responsible for launching a web server and exposes the REST API of each module, prefixed by their namespaces. Each module can easily expose its API while authentication and rights control of each user accessing this API can be done globally using this module.

4.3.5.2 Front-end webserver module

However, the API calls although convenient to interface different services are not practical for the end user and especially for demonstration purposes. Indeed, regardless the fact that it is not practical to make quick API calls, the user will have to know the structure of the API and the different parameters for each endpoint. To present this API in a user-friendly way, this module provides a wrapper as a web user interface.

Thus, the user can see the formatted information, some enhancements if needed and possibly in a graphical way: for example, the topology is now a graph to quickly view different devices and information about them. Another example is the presentation of the switches flow tables that allow the user to track the path of a packet from switch to switch based on an initial rule. The functioning is pretty basic: these are basic AJAX API calls which are then formatted on the client-side.

4.4 Conclusion

By using a modular structure, the demonstration prototype allows to individually improve features that have been made possible by SDN. This design also allows to add and easily test new features which therefore is particularly suitable for a test prototype.

Chapter 5

Validation of the prototype

In this chapter, I will validate that my demonstration prototype respects the specifications described in chapter 3. I'll start by validating the correct operation of my controller then I will focus on performance.

5.1 Criteria validation

The first validation of the prototype was to check that it met the criteria set out in the specifications. That is to say that it was a valid demonstration prototype. Then I ensured its proper functioning, first with a rather simplistic topology and some hosts to check not only that the hosts managed to communicate with each other but also that the various blockages, inspections and redirections were functional. Finally, a further study was carried out by increasing the number of flows, switches and hosts and finally by choosing several types of topology and by simulating hosts movements and network problems.

5.2 Performances analysis

As expected and foreseeable, performances are not particularly good when there are many flows to install. However, once the flow is installed, the throughput is correct.

5.2.1 Flow setting time

To get an idea of the time required for the controller to establish the flows, I set up a topology with a variable number of hosts and switches. To devote this test exclusively to the time required to install a rule, I took a tree topology to favor relatively long paths between hosts. Then I made the hosts ping each other sequentially.

To establish a flow between the hosts, it is necessary that the ARP packets are authorized and broadcast. Once the host retrieved the remote ARP address, it will send the ICMP request and the controller will install the corresponding flow, then

when the remote host will send the response, another corresponding flow will be installed.

5.2.1.1 Flow number

So we have 2 flows per switch and hosts pair: a flow for the ICMP request and another for the ICMP response. An improvement of the controller to avoid a too large number of rules in the switches is to install a specific rule on the edge switch and a generic rule thereafter. However, at the moment, the controller will not store the generic rules already sent to the switches and will therefore re-emit one for each new flow. The packet containing the generic rule is accordingly issued to each flow and is well received by the switch which will update its flow table or not (in the case of a previous packet was already sent). Thus, we can consider that the processing time is the same with or without this controller optimization.

The number of installed rules therefore includes all packets sent to add a rule, whether it has been taken into account or not. As a flow rule must be installed on each switch between two hosts, for a tree topology with a depth *depth* and a fan-out *fanout*, we have:

$$\begin{aligned} N_{rules} &= \sum_{(h,d)\in\{hosts\}^2} 2N_{switches\ on\ path\ from\ h\ to\ d} \\ &= 2N_{hosts} \sum_{d=1}^{depth} N_{concerned\ hosts} N_{switches\ on\ path} \\ &= 2fanout^{depth} \sum_{d=1}^{depth} ((fanout-1)fanout^{d-1})(2d-1) \\ &= 2(fanout-1)fanout^{depth-1} \sum_{d=1}^{depth} fanout^{d}(2d-1) \end{aligned}$$

5.2.1.2 ARP

In addition to the ICMP flows described above, the hosts must also fill their ARP table and thus add requests and additional flows. To ignore this phenomenon, ARP entries can be statically added before the experiment on all hosts.

5.2.1.3 Average time to install a rule

Ping each host sequentially takes time, so to take it into account, two measurements are made. The first one is made just after the initialization of the network: the controller will then discover the hosts and install the appropriate flows and the second just after. As flows have not yet expired, the packets will use these flows and the controller is not involved.

We can therefore analyze the overhead added by the request done to the controller. This is then divided by the number of rules to obtain the average time needed to receive an unknown packet, handle it and install adequate rules in all switches on the path.

Topology	Topology		Statistics		
	Depth	Fan-out	Hosts	Switches	Rules
Tree 3,3	3	3	27	13	5940
Tree 3,4	3	4	64	21	35712
Tree 4,3	4	3	81	40	79056

(a) Presentation of test topologies

Topology	With ARP			Static ARP		
	First	Installed	Av. Time	First	Installed	Av. Time
Tree 3,3	15.7 s	2.04 s	2.31 ms	11.9 s	1.93 s	1.67 ms
Tree 3,4	114 s	20.5 s	2.63 ms	69.3 s	16.5 s	1.48 ms
Tree 4,3	246 s	64.5 s	2.30 ms	160 s	42.5 s	1.48 ms

(b) Time needed to establish rules depending on the topology

Table 5.1: Measures of time needed by the controller to add a rule in a switch.

5.2.1.4 Results Interpretation

A summary of the results obtained is shown in the table 5.1. The procedure used and all results are available in the Appendix, section B.1.

However, this result must be qualified. Indeed, when performing this test, I have noticed that the controller process used only very little CPU but the process corresponding to virtual machine in which Mininet was running was using all its allocated resources. The performance bottleneck can therefore rather be due to the network test environment and the resources need by all the different virtual switches running on a single virtual machine rather than the controller. The two measures method involving or not the controller gives a more accurate idea of the time required to install a rule — since the overhead brought by the virtual switches will be the same —, but this time also includes the addition of the rules in the virtual switches and therefore significant calls for the Mininet environment.

Analysis of the results remains valid quantitatively, but we may hope better performances of the controller when using dedicated hardware or if Mininet is no longer the limiting factor (by running it directly in the host for example).

Four observations can be made about these results:

- The less the topology is complex, the more the results will be affected by big differences in measures due to the saturation of the Mininet environment.

- The more the fan-out increases, the more complex the topology becomes (number of hosts or links) and therefore the controller takes longer to manage this additional complexity, which results in an increase in the average time to install a rule.

- The more the depth increases, the more complex the topology becomes (far more switches so far more hosts and links), but despite the fact that the average path length between two hosts increases, the average time needed to establish

a rule is sensibly the same. It is because the rules are installed all along the way, thus even if the processing time of a packet is longer than the average time decreases or stays the same since we have installed more rules.

- The overhead added by the ARP traffic seems impressive, but given the fact that Mininet seems to be the limiting factor, it is reasonable to assume that this overhead is mainly due to the number of ARP packets circulating in the network rather than an overload on the controller side.

5.2.2 Throughput between hosts

Then I focused on estimating the influence of my controller on the throughput between the hosts. For this, I used a tree topology with a depth of 5 and a fan-out of 2 to get paths between hosts of different lengths while maintaining a relatively small topology.

Then I measured the TCP flow bandwidth between the hosts using `iperf` [26], but the results were not conclusive because Mininet was using all its resources and no longer offered consistent results. The results are available in Appendix B.2. However, these results are less relevant because the installation of a flow is fast (as shown in 5.2.1) and only induce a delay for the establishment of the flow since only the first packet will be affected (see section 2.1.4).

5.3 Conclusion

This controller therefore meets the criteria for the demonstration and the prototype is also quite efficient since the virtualized network test environment saturates faster. Potential improvements are listed in chapter 6.

Chapter 6

Future work

This chapter will be devoted to the possible improvements of my demonstration prototype but also to some new and noteworthy technologies which are complementary to Software Defined Network.

6.1 Possible improvements on the controller

This section will be dedicated to the future work that can be done on my controller.

6.1.1 Routing

A first improvement for this controller is to improve the routing of packets.

Currently, the topology module provides the needed information (links between switches, location of the different devices) and when a packet has to be routed, paths are calculated (using Dijkstra's algorithm) then merged to avoid excessive load (as described in section 4.3.3 and shown in figure 4.2).

This presents several problems for use in large scale: topology must be known globally (or at least between the concerned devices), the path computation is long, inefficient and do not reuse the previously computed information. A cache system — emptied when there are changes in topology or in the links preferences — reduces the number of calls to the Dijkstra's algorithm, but is not a long term solution.

In the particular case of this controller, routing is used for packets between two devices, but this can be viewed as a special case of the much more complex routing on which is based Internet. Indeed, this is a well-known problem, routers constituting the Internet are already having to face it and therefore, solutions already exist: OSPF [41, 12], IS-IS [54] and BGP [59]. For example, OSPF (defined for IPv4 in [41] and for IPv6 in [12]) address this problem: routers broadcast known networks and path length. Using this information, routers will run Djikstra locally in order to select which path will be use. Thus, the load is minimized, and the database distributed.

A similar technique could be used here, while taking advantage of the fact that the controller has more information — and unlike the cases described above, the

information can easily be certified valid — thanks to network discovery, information provided by the administrator (like preferred links to use) and that this information can be centralized.

In our special case, we also can improve the algorithm using the Floyd-Warshall algorithm for example.

6.1.2 Flood and loop-free topology

Currently, the controller allows packets flooding, and to prevent that loops in the topology may cause a broadcast radiation, it will disable redundant links as soon as it discovers one during the topology discovery.

One solution would be to implement RSTP [72] in order to recover quickly in the event of a failure. As seen in the paragraph addressing network isolation (in section 4.3.3), we can also choose to limit the broadcast to a series of ports without disabling links. These redundancies can thus be used to achieve link aggregation and therefore increase throughput between switches.

6.1.3 Flow statistics' analysis

Another improvement is a better statistics management. They are currently retrieved from switches regularly, but a too quick retrieval will induce an overload at the controller and an unnecessary bandwidth between the switch and the controller. However, a period of retrieval too big would not allow the controller to react quickly enough to a sudden increase in traffic, making the statistics monitoring unproductive.

DevoFlow [40, 15] deals with this problem and proposes two solutions: analyze, for example with sFlow[1], headers of uniformly chosen packets or use triggers mechanisms with OpenFlow counters in order to push reports to the controller. This alternative may be interesting since Mogul and Condon argue [39] that switch-local CPUs are more relevant to manage OpenFlow counters than switch ASICs.

6.1.4 Redundancy and distributed system

A big problem of the controller is that although it has become the most critical network element — the controller is necessary to ensure that packets are transmitted on the network —, there is no redundancy. In addition, the current system is not very scalable. We can separate the different modules and assign a switch manager per group of switches in order to reduce the load but these measures consist only in separating the different functions rather than conceive a true distributed completely scalable system.

Solutions like Onix [29], Maestro [8] or HyperFlow [69] address this problem by providing scalability while keeping network control logic into a single virtual controller. Maestro [8] focuses on exploiting multi-threading possibilities to improve performances. HyperFlow [69] implements a publish/subscribe system over the

[1] `http://www.sflow.org/organization/index.php`

distributed file system WheelFS to propagate useful controller events and an updated local cache of the state of the controller. Onix [29] provides a platform to deploy a complete network control plane as a distributed system which has been used in B4 [28], the private WAN of Google.

6.1.5 Controller protection

This section will only be focused on managing permissions and authorization. The inherent problems of SDN were discussed in section 2.1 and the load of the controller in case of it receives a lots of packets — in a malicious way or simply because of high traffic — are respectively handled by more generic flows to drop packets (see 2.1.5) which depend on the chosen policy and a distributed management of controller's load discussed above in section 6.1.4.

Currently, the modification interface exposed by the controller is not protected. Adding a simple rights management — a user can only change some options — is easy but not very interesting due to the dynamic nature of the controller. For example, if malicious traffic is detected and security measures are taken or if an administrator wants to block a flow, we want the controller to block the flow whether or not a user has legitimately applied for authorizing a sub-flow. Thus, we need a system of hierarchical and secure rules as described in section 2.2.1 for example.

6.2 SDN Future

With Software Defined Networks allowing a separation of control and forwarding functions and a centralization of control, the ability to program the network through well-defined interfaces came up. Thus many technologies can take advantage of it or use it as a complementary solution. In this section, I will present some of these solutions.

6.2.1 Network Functions Virtualization (NFV)

Network Functions Virtualization (NFV) [44] is a complementary technology to SDN and is based on the fact that many network appliances are currently running on dedicated proprietary hardware and thus present many problems:

- The needed space and power to deploy it.
- Innovation and short hardware lifecycle lead to a rapid end of life of hardware appliances, thus increasing costs and restricting innovation.
- Complexity for deploying and operating these appliances.

In the same time, standard IT virtualization is growing in every domain: server, storage and networks. Thus, running network functions on software on top of existing hardware is highly desirable. With NFV, scalability and simplicity would be improved since any network equipment could be moved to, or instantiated in, various

locations in the network. There would be no need to install a new equipment, since — as virtual machines can be moved, instantiated or duplicated everywhere — it would use existing hardware. Moreover, scalability would be simpler because if there is an increase in network needs or a failure, relevant network equipment could be instantiated accordingly — and since it can be installed everywhere, optimize the location of this appliance and reduce overhead is easy — for the desired period of time.

It would also be easier to create and test new solutions — since it just software applications that need to allocate some existing resources —, thus encouraging openness and increasing rapidity of Time to Market. Finally, this will be cost-saving since equipment costs will be reduced, the use of existent resources will be optimized and energy consumption could be reduced by switching off servers during off-peak hours.

With SDN bringing a centralized network management, NFV perfectly fits in the way since a global controller would easily scale the network according to network operators and customers' needs by instantiating relevant network appliances. Moreover, NFV can provide an infrastructure upon which the SDN software can run since an objective of SDN is to run on commodity servers and switches.

However some technical challenges have to be addressed, particularly achieving high performance on virtualized network appliances, be fully automated and coexist with legacy platforms.

6.2.2 Application Defined Networking (ADN)

Application Defined Networking (ADN) is based on SDN and on the principle to let applications specify their requirements and let the network satisfy them. Requirements can be about Firewalling, Quality of Service and Load Balancing or needed Bandwidth for example. This is a direct consequence of cloud computing needs.

6.2.3 Software Defined Perimeter (SDP)

Software Defined Perimeter (SDP) [67] is a security framework to protect applications from network based attacks.

Traditional security measures consists of deploying security measures to block incoming connections while filtering outgoing connections but such measures are becoming obsolete due to network dynamicity and new security flaws providing untrusted access inside the network. SDN addresses this problem by giving the ability to adapt and change network security perimeters according to users' needs while SDP is meant to securely ensure identity verification and then provide access to an application using the controller. Thus, an application infrastructure could not be detected from the outside, mitigating common attacks like network scanning, denial of service (DoS), application vulnerability exploits like SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF) or man-in-the-middle attacks

for example.

The identity of a host would be checked by an SDP controller, and then this host would be authorized to connect to a SDP protected host. SDN through network management is particularly suitable for deploying such a system, since the controller is responsible for controlling access to the network. SDP would provide secure channels and identity verification.

Conclusion

Software Defined Networking, by separating the control plane and the data plane in the networks, deeply redefines the traditional architectural pattern of networks. Based on a centralized controller using multiple protocols to manage the network equipment, this new paradigm allows network administrators to use available and generic resources like commodity servers to run the controller. Since high level applications, resources and technologies like a subtle rights management or cryptography can be used during the decision process, the network becomes programmable and adaptable to current needs like flexibility and mobility of everyday devices, custom network policies or cloud computing. From now on, the network is not a static layout anymore but a dynamic and intelligent layer that will automatically adapt itself to users and administrators requirements along with network changes.

Every network protocol can be used as a management protocol by the controller, but the most widespread is currently OpenFlow which combines a fine, customizable and upgradable management of network packets according to their different network layers with a guarantee of current performance by using a protocol that can be adapted on many network devices, whether they are hardware-based or virtual, thereby removing any advanced function of network equipment and reducing costs as well as complexity. OpenFlow consolidates the separation of the control plane and the forwarding plane of SDN by integrating the concept of controller that can easily update the rules in the switch and which can also be consulted by the switch when it does not know how to handle a packet. Therefore, the network can react and effectively adapt to the different events occurring in the network while leaving considerable latitude to the controller in determining what part should be dynamic and what part should be static.

Many controllers have emerged, offering various features, proving that it was possible to obtain good performances as well as quantifiable gains in case of real deployment. Some security considerations and technical challenges as running a fully distributed and scalable controller are not yet fully resolved but since most of the notable controllers are intrinsically dynamic and extensible, it is perfectly possible to adapt them to meet a specific need or improve security by coupling them with existing technologies.

Bibliography

[1] S. Amante et al. *IPv6 Flow Label Specification*. RFC 6437 (Proposed Standard). Internet Engineering Task Force, Nov. 2011. URL: http://www.ietf.org/rfc/rfc6437.txt.

[2] Mitchell Ashley. "Layered Network Security: A best-practices approach". In: *White paper. StillSecure* (Jan. 2003). URL: www.stillsecure.com/sites/default/files/documents/StillSecure_LayeredSecurity.pdf (visited on 12/18/2013).

[3] Siamak Azodolmolky, Philipp Wieder, and Ramin Yahyapour. "SDN-based cloud computing networking". In: *Transparent Optical Networks (ICTON), 2013 15th International Conference on*. IEEE. 2013, pp. 1–4.

[4] Jeffrey R Ballard, Ian Rae, and Aditya Akella. "Extensible and scalable network monitoring using OpenSAFE". In: *Proc. INM/WREN* (2010).

[5] *Beacon webpage*. URL: https://openflow.stanford.edu/display/Beacon/Home (visited on 12/18/2013).

[6] *Big Switch Steps Down from OpenDaylight Platinum Status*. Big Switch Networks. URL: http://www.bigswitch.com/blog/2013/06/05/big-switch-steps-down-from-opendaylight-platinum-status (visited on 10/03/2013).

[7] Matthew Caesar et al. "Design and implementation of a routing control platform". In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, pp. 15–28.

[8] Zheng Cai, Alan L Cox, and TS Eugene Ng Maestro. *A system for scalable OpenFlow control*. Tech. rep. Technical Report TR10-08, Rice University, 2010.

[9] Neal Cardwell, Stefan Savage, and Thomas Anderson. "Modeling TCP latency". In: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 3. IEEE. 2000, pp. 1742–1751.

[10] Martin Casado et al. "Ethane: Taking control of the enterprise". In: *ACM SIGCOMM Computer Communication Review*. Vol. 37. 4. ACM. 2007, pp. 1–12.

[11] Martin Casado et al. "SANE: A protection architecture for enterprise networks". In: *USENIX Security Symposium*. 2006.

[12] R. Coltun et al. *OSPF for IPv6*. RFC 5340 (Proposed Standard). Updated by RFCs 6845, 6860. Internet Engineering Task Force, July 2008. URL: http://www.ietf.org/rfc/rfc5340.txt.

[13] LAN/MAN Standards Committee. *IEEE Standard for Local and metropolitan area networks – Station and Media Access Control Connectivity Discovery*. IEEE, 2009. URL: http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf (visited on 12/18/2013).

[14] *Contrail*. Juniper Networks. URL: http://www.juniper.net/us/en/products-services/sdn/contrail/ (visited on 12/18/2013).

[15] Andrew R Curtis et al. "DevoFlow: Scaling flow management for high-performance networks". In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 254–265.

[16] Saurav Das, Guru Parulkar, and Nick McKeown. "Why OpenFlow/SDN can succeed where GMPLS failed". In: *European Conference and Exhibition on Optical Communication*. Optical Society of America. 2012.

[17] Dmitry Drutskoy, Eric Keller, and Jennifer Rexford. *Scalable network virtualization in software-defined networks*. 2013.

[18] David Erickson. "The Beacon OpenFlow Controller". In: *HotSDN*. ACM. 2013.

[19] David Erickson et al. "Optimizing a virtualized data center". In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 478–479.

[20] Nick Feamster et al. "The case for separating routing from routers". In: *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*. ACM. 2004, pp. 5–12.

[21] *Floodlight webpage*. URL: http://www.projectfloodlight.org/floodlight/ (visited on 12/18/2013).

[22] Google. *Inter-Datacenter WAN with centralized TE using SDN and OpenFlow*. 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/customer-case-studies/cs-googlesdn.pdf (visited on 12/18/2013).

[23] Natasha Gude et al. "NOX: towards an operating system for networks". In: *ACM SIGCOMM Computer Communication Review* 38.3 (2008), pp. 105–110.

[24] Nikhil Handigol et al. "Reproducible network experiments using container-based emulation". In: *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM. 2012, pp. 253–264.

[25] *Interface to the Routing System (i2rs) Workgroup*. Internet Engineering Task Force. URL: http://datatracker.ietf.org/wg/i2rs/ (visited on 12/18/2013).

[26] *Iperf project*. URL: http://sourceforge.net/projects/iperf/ (visited on 12/18/2013).

[27] Ivan Pepelnjak. *Cloud Networking - From Theory to Practice*. ipSpace.net / NIL Data Communications 2012. URL: https://ripe64.ripe.net/presentations/20-Cloud_Networking_%C3%A2%C2%80%C2%93_From_Theory_to_Practice_(RIPE).pdf (visited on 12/18/2013).

[28] Sushant Jain et al. "B4: Experience with a globally-deployed software defined WAN". In: *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM. 2013, pp. 3–14.

[29] Teemu Koponen et al. "Onix: A Distributed Control Platform for Large-scale Production Networks." In: *OSDI*. Vol. 10. 2010, pp. 1–6.

[30] Vasileios Kotronis, Xenofontas Dimitropoulos, and Bernhard Ager. "Outsourcing the routing control logic: better internet routing based on sdn principles". In: *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM. 2012, pp. 55–60.

[31] Bob Lantz, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM. 2010, p. 19.

[32] Dan Levin et al. "Toward Transitional SDN Deployment in Enterprise Networks". In: *Open Networking Summit*. 2013.

[33] MultiMedia LLC. *Open vSwitch: An Open Virtual Switch*. 2013. URL: http://openvswitch.org/ (visited on 12/18/2013).

[34] *LXC - Linux Containers*. URL: http://linuxcontainers.org/ (visited on 12/18/2013).

[35] M Mahalingam et al. *VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. Internet Engineering Task Force. URL: http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-06 (visited on 12/18/2013).

[36] David A Maltz et al. "Routing design in operational networks: A look from the inside". In: *ACM SIGCOMM Computer Communication Review*. Vol. 34. 4. ACM. 2004, pp. 27–40.

[37] Nick McKeown et al. "OpenFlow: enabling innovation in campus networks". In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[38] *Mininet website*. URL: http://mininet.org/ (visited on 12/18/2013).

[39] Jeffrey C Mogul and Paul Congdon. "Hey, you darned counters!: get off my ASIC!" In: *Proceedings of the first workshop on Hot topics in software defined networks*. ACM. 2012, pp. 25–30.

[40] Jeffrey C Mogul et al. "Devoflow: Cost-effective flow management for high performance enterprise networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM. 2010, p. 1.

[41] J. Moy. *OSPF Version 2*. RFC 2328 (INTERNET STANDARD). Updated by RFCs 5709, 6549, 6845, 6860. Internet Engineering Task Force, Apr. 1998. URL: http://www.ietf.org/rfc/rfc2328.txt.

[42] Jad Naous et al. "Implementing an OpenFlow switch on the NetFPGA platform". In: *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM. 2008, pp. 1–9.

[43] Marcelo R Nascimento et al. "Virtual routers as a service: the routeflow approach leveraging software-defined networks". In: *Proceedings of the 6th International Conference on Future Internet Technologies*. ACM. 2011, pp. 34–37.

[44] *Network Functions Virtualization*. ETSI. URL: http://portal.etsi.org/NFV/NFV_White_Paper.pdf (visited on 12/18/2013).

[45] *NOX Github repository*. URL: https://github.com/noxrepo/nox (visited on 12/18/2013).

[46] *NOX Mailing list*. URL: http://news.gmane.org/gmane.network.nox.devel (visited on 12/10/2013).

[47] *NOX website*. URL: http://www.noxrepo.org/ (visited on 12/18/2013).

[48] *OpenDaylight website*. URL: http://www.opendaylight.org/ (visited on 12/18/2013).

[49] *openFlow 1.0.0 patch for OpenWrt*. URL: git://gitosis.stanford.edu/openflow-openwrt (visited on 11/25/2013).

[50] *Openflow switch specification: Version 1.0.0*. Open Networking Foundation. Dec. 2009. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf (visited on 12/18/2013).

[51] *Openflow switch specification: Version 1.3.3*. Open Networking Foundation. June 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.3.pdf (visited on 12/18/2013).

[52] *OpenStack Neutron project*. OpenStack. URL: https://wiki.openstack.org/wiki/Neutron (visited on 12/18/2013).

[53] *OpenWrt website*. URL: https://openwrt.org/ (visited on 12/18/2013).

[54] D. Oran. *OSI IS-IS Intra-domain Routing Protocol*. RFC 1142 (Informational). Internet Engineering Task Force, Feb. 1990. URL: http://www.ietf.org/rfc/rfc1142.txt.

[55] Ben Pfaff et al. "Extending Networking into the Virtualization Layer." In: *Hotnets*. 2009.

[56] Philip Porras et al. "A security enforcement kernel for OpenFlow networks". In: *Proceedings of the first workshop on Hot topics in software defined networks*. ACM. 2012, pp. 121–126.

[57] *POX webpage*. URL: http://www.noxrepo.org/pox/about-pox/ (visited on 12/18/2013).

[58] *PyPy website*. URL: http://pypy.org/ (visited on 12/18/2013).

[59] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271 (Draft Standard). Updated by RFCs 6286, 6608, 6793. Internet Engineering Task Force, Jan. 2006. URL: http://www.ietf.org/rfc/rfc4271.txt.

[60] Christian Esteve Rothenberg et al. "Revisiting routing control platforms with the eyes and muscles of software-defined networking". In: *Proceedings of the first workshop on Hot topics in software defined networks*. ACM. 2012, pp. 13–18.

[61] Rob Sherwood et al. "Flowvisor: A network virtualization layer". In: *OpenFlow Switch Consortium, Tech. Rep* (2009).

[62] Seungwon Shin and Guofei Gu. "CloudWatcher: Network Security Monitoring Using OpenFlow in Dynamic Cloud Networks (or: How to Provide Security Monitoring as a Service in Clouds?)" In: *Proceedings of the 7th Workshop on Secure Network Protocols (NPSec'12), co-located with IEEE ICNP'12*. Oct. 2012.

[63] Seungwon Shin et al. "AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks". In: *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS'13)*. Nov. 2013.

[64] Seungwon Shin et al. "FRESCO: Modular Composable Security Services for Software-Defined Networks". In: *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS'13)*. Feb. 2013.

[65] Ruslan L Smeliansky and Alexander V Shalimov. *Advanced Study of SDN/OpenFlow controllers*. Jan. 2013.

[66] *Snort*. URL: http://www.snort.org/ (visited on 12/18/2013).

[67] *Software Defined Perimeter*. Cloud Security Alliance. URL: https://cloudsecurityalliance.org/download/software-defined-perimeter/ (visited on 12/18/2013).

[68] *Software-Defined Networking: The New Norm for Networks*. Open Networking Foundation. Apr. 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf (visited on 12/18/2013).

[69] Amin Tootoonchian and Yashar Ganjali. "HyperFlow: A distributed control plane for OpenFlow". In: *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association. 2010, pp. 3–3.

[70] Amin Tootoonchian et al. "On controller performance in software-defined networks". In: *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*. 2012.

[71] J Van der Merwe et al. “Dynamic connectivity management with an intelligent route service control point”. In: *Proceedings of the 2006 SIGCOMM workshop on Internet network management*. ACM. 2006, pp. 29–34.

[72] Wald Wojdak. “Rapid Spanning Tree Protocol: A new solution from an old technology”. In: *Reprinted from CompactPCI Systems* (2003).

[73] Avishai Wool. “A quantitative study of firewall configuration errors”. In: *Computer* 37.6 (2004), pp. 62–67.

Appendix A

OpenFlow specification

A description of the basic structure of an OpenFlow component and of the OpenFlow protocol is available in the section 1.2.1. In this appendix, I will briefly present the features provided by OpenFlow. More details are available in the OpenFlow specifications [50, 51].

A.1 Packet processing

OpenFlow 1.0 only uses one flow table but OpenFlow 1.3 introduces the possibility of using multiple flow tables, each one with its own counters and its set of actions. The Packet flow through the processing pipeline is shown in figure A.1. OpenFlow 1.3 also introduces two new tables, each one with its own counters:

- The Group table which permits to apply multiple set of actions on flows.
- The Meter Table which permits to apply Quality of Service operations such as rate-limiting, and can be combined with per-port queues.

An exhaustive list of available counters is shown in Table A.1

Table A.1: OpenFlow 1.3 counters

Counter	**Bits**	
Per Flow Table		
Reference Count (active entries)	32	Required
Packet Lookups	64	Optional
Packet Matches	64	Optional
Per Flow Entry		
Received Packets	64	Optional
Received Bytes	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional

Table A.1: (continued)

Counter	**Bits**	
Per Port		
Received Packets	64	Required
Transmitted Packets	64	Required
Received Bytes	64	Optional
Transmitted Bytes	64	Optional
Receive Drops	64	Optional
Transmit Drops	64	Optional
Receive Errors	64	Optional
Transmit Errors	64	Optional
Receive Frame Alignment Errors	64	Optional
Receive Overrun Errors	64	Optional
Receive CRC Errors	64	Optional
Collisions	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Queue		
Transmit Packets	64	Required
Transmit Bytes	64	Optional
Transmit Overrun Errors	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Group		
Reference Count (flow entries)	32	Optional
Packet Count	64	Optional
Byte Count	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Group Bucket		
Packet Count	64	Optional
Byte Count	64	Optional
Per Meter		
Flow Count	32	Optional
Input Packet Count	64	Optional
Input Byte Count	64	Optional
Duration (seconds)	32	Required
Duration (nanoseconds)	32	Optional
Per Meter Band		
In Band Packet Count	64	Optional
In Band Byte Count	64	Optional

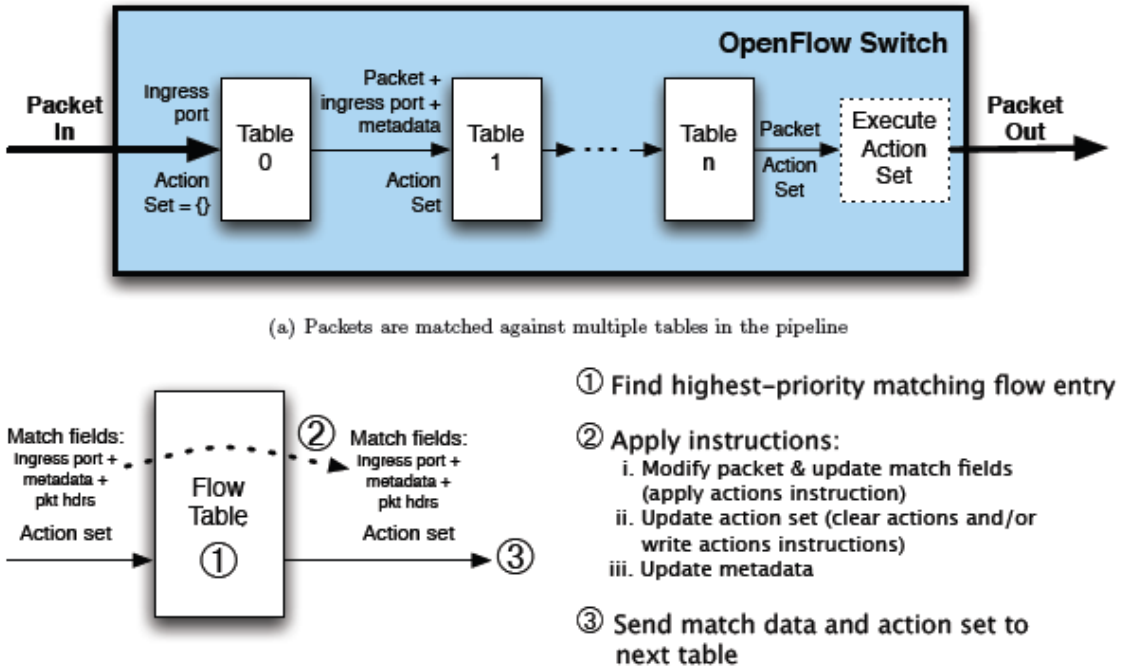


Figure A.1: OpenFlow 1.3 packet processing. The image is taken from the OpenFlow 1.3 specifications [51]

A.2 Flow entry

OpenFlow allows matching a packet according to many criteria. The criteria specified by OpenFlow 1.3 are listed below. The criteria supported by OpenFlow 1.0 are *emphasized*.

- Port
  - Ingress Port (can be logical ports defined by non OpenFlow methods like link aggregation groups, tunnels or loopback interfaces).
  - *Physical Input Port*
- Metadata passed between tables
- Packet Header Fields
  - *Ethernet source address*
  - *Ethernet destination address*
  - *Ethernet frame type*
  - *VLAN Id*
  - *VLAN priority*
  - *IPv4 source address*
  - *IPv4 destination address*
  - *IP protocol*
  - *IP DSCP (6 bits in ToS field)*

  - IP ECN (2 bits in ToS field)
  - *TCP source port*
  - *TCP destination port*
  - *UDP source port*
  - *UDP destination port*
  - SCTP source port
  - SCTP destination port
  - *ICMP Type*
  - *ICMP Code*
  - ARP Opcode
  - ARP source IPv4 address
  - ARP destination IPv4 address
  - ARP source hardware address
  - ARP destination hardware address
  - IPv6 source address
  - IPv6 destination address
  - IPv6 Flow Label
  - ICMPv6 Type
  - ICMPv6 Code
  - Target address for ND
  - Source link-layer for ND
  - Target link-layer for ND
  - MPLS Label
  - MPLS TC
  - MPLS BoS bit
  - PBB I-SID
  - Logical Port Metadata
  - IPv6 Extension Header pseudo-field

For each flow entry, multiple instructions — or actions for OpenFlow 1.0 — can be specified:

- TTL Adjustments
  - Set TTL (MPLS TTL, IP TTL)
  - Decrement TTL (MPLS TTL, IP TTL)
  - Copy TTL inwards: apply copy TTL inward actions to the packet
  - Copy TTL outwards: apply copy TTL outwards action to the packet
- Push/Pop Tags
  - push-VLAN: Push a new VLAN header onto the packet
  - pop-VLAN: Pop the outer-most VLAN header from the packet
  - push-MPLS: Push a new MPLS shim header onto the packet
  - pop-MPLS: Pop the outer-most MPLS tag or shim header from the packet
  - push-PBB: Push a new PBB service instance header (I-TAG TCI) onto the packet

  - pop-PBB: Pop the outer-most PBB service instance header (I-TAG TCI) from the packet

- Set Fields: apply all set-field actions to the packet. All fields specified in the match fields can be modified.
- QoS: apply all QoS actions, such as set queue to the packet
- Group: if a group action is specified, apply the actions of the relevant group bucket(s) in the order specified by this list
- Output: Forward the packet to a port among:
  - ALL
  - CONTROLLER
  - LOCAL (Represents the switch's local networking stack and its management stack)
  - TABLE
  - IN_PORT
  - NORMAL (Represents the traditional non-OpenFlow pipeline of the switch)
  - FLOOD (Represents flooding using the normal pipeline of the switch)
- Drop

Appendix B

Prototype tests results

B.1 Flow setting time

This test was done running my controller on Windows 7 (i5-3380M @ 2.90GHz, 8GB) with PyPy [58]. The Mininet environment was running on Ubuntu 12.10 (3.5.0-40-generic x86_64 GNU/Linux) in a Virtual Machine with VirtualBox (1 vCPU, 2GB).

For each configuration (different topologies and with or without static ARP), I repeated these operations three times:

1. Set up the Mininet environment.
2. Start the controller and wait some seconds in order for the controller to detect links between switches and build a basic network topology.
3. Running a `time pingall` in the Mininet shell to measure the global time for establish all flows when using the controller.
4. Running 2 other consecutive `time pingall` in the Mininet shell to measure the time when using established flows in switches (the flows timeout was set accordingly).
5. Close the controller and shutdown the Mininet environment.

The results available in the report in table 5.1 are an average of the 3 (or 6) measures; the full results are available in table B.1.

Topology	Topology		Statistics		
	Depth	Fan-out	Hosts	Switches	Rules
Tree 3,3	3	3	27	13	5940
Tree 3,4	3	4	64	21	35712
Tree 4,3	4	3	81	40	79056

(a) Presentation of test topologies

		1st run	2nd run	3rd run
With ARP	With controller	15.4	15.9	15.9
	No controller	2.17 2.16	2.18 1.41	2.16 2.16
Static ARP	With controller	12.1	11.8	11.7
	No controller	2.35 1.25	2.18 2.30	1.27 2.23

(b) Results in seconds for topology: Tree 3.3

		1st run	2nd run	3rd run
With ARP	With controller	111.5	119.6	111.8
	No controller	20.7 19.2	22.8 19.4	20.6 20.2
Static ARP	With controller	68.3	70.0	69.5
	No controller	16.4 16.0	17.5 17.0	16.4 15.8

(c) Results in seconds for topology: Tree 3.4

		1st run	2nd run	3rd run
With ARP	With controller	239.1	250.9	249.0
	No controller	61.5 60.5	66.4 64.0	70.1 64.4
Static ARP	With controller	158.6	159.7	160.7
	No controller	40.8 40.7	41.6 44.9	44.9 42.0

(d) Results in seconds for topology: Tree 4.3

Table B.1: Measures of time needed by the controller to add rules in the switch

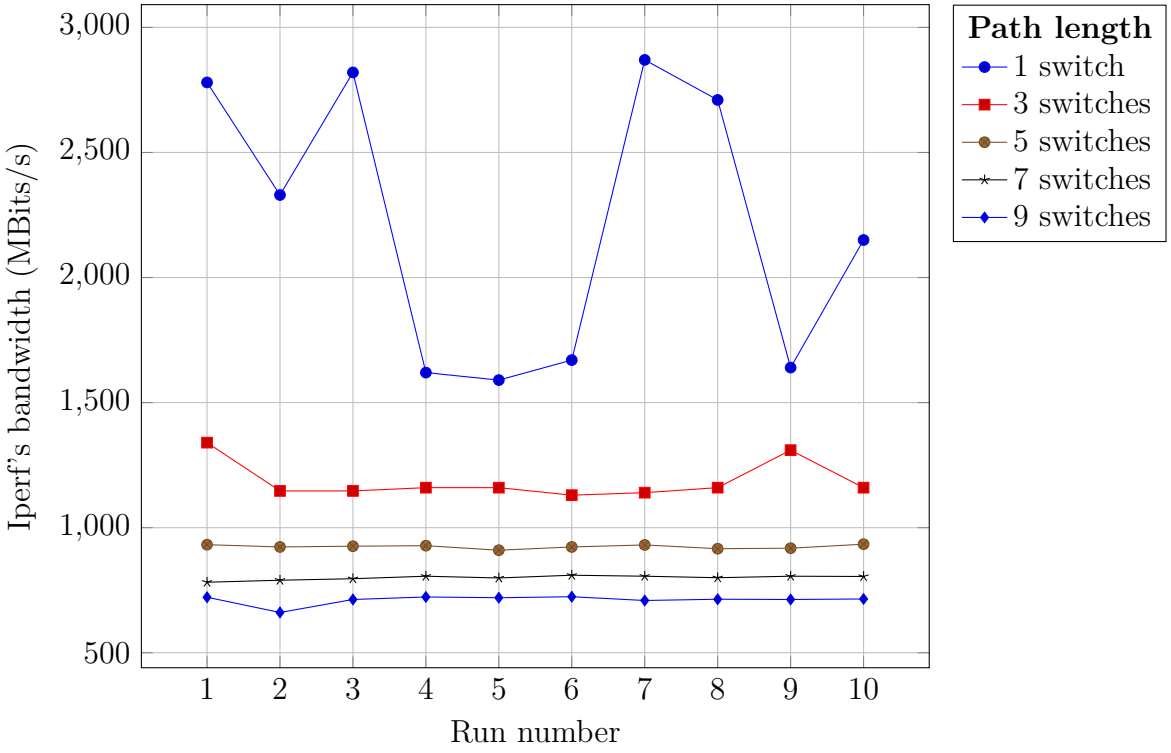


Figure B.1: TCP bandwidth between 2 hosts

B.2 Throughput between hosts

This test was done running my controller on Windows 7 (i5-3380M @ 2.90GHz, 8GB) with PyPy [58]. The Mininet environment (2.1.0/commit 5da93762) was running on Ubuntu 12.10 (3.5.0-40-generic x86_64 GNU/Linux) in a Virtual Machine with VirtualBox (1 vCPU, 2GB).

Its purpose was to evaluate the influence of the controller on the throughput between the hosts. After set up the topology (a tree topology with a fan-out of 2 and a depth of 5), I started `iperf` [26] on one host and successively run the client on different host in order to take into account the path length. For accurate measurements, I realized this test 10 times, and bandwidth is measured for 10 seconds.

It may be noted that the results are not conclusive because it is impossible to determine the influence of the controller on the establishment of the flow (1st run) since the results greatly vary. This is due to the fact that Mininet uses the Linux scheduler and cannot guarantee a consistency in results, especially for one switch.

The reduction of bandwidth depending on the number of switches on the path is because the more switches there are, the more the computational power is divided between all concerned switches.

Given these results, I did not pursue in greater detail the measures since it would evaluate Mininet performances rather than the controller ones and as shown in section 5.2.2, the induced slowdown should be negligible for large flows.

TRITA-ICT-EX-2014:5

www.kth.se