FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

API design and implementation of a management interface for SDN whitebox switches

Rubens Jesus Alves Figueiredo

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Ana Cristina Costa Aguiar

December 24, 2017

Resumo

Abstract

...

Contents

List of Figures

List of Tables

Abreviaturas e Símbolos

SDN	Software Defined Networks

Part I

Introduction and Background

Introduction

1.1 Context

1.2 Motivation

1.3 Goals

Software Defined Networking

Computer networking is a vital part of the services that are offered today, and as such, the performance in technology backing these services is central to the quality of these services. As the service providers reorganize their data centers in the cloud computing domain, enabling several improvements in the predictability, quality of service and ease of use of their services. New technologies are then required to make sure that their services are adapted to the fast changing landscape of networking services. One of the most notable innovations in this field is called Software Defined Networking, because its architecture allows for two essential features

- **Separation of network planes** SDN allows for the separation of the network control plane from the data forwarding plane by having network "intelligence" present in the network controllers, and having them control the forwarding elements that live in the Data Plane
- **Centralization of network management functions** By isolating the management on a separate plane, there is possibility of developing a single controller that can regulate the entire network, having unrestricted access to every element present in the network, simplifying management, monitoring, application of QoS policies, flow optimization, ...

In this chapter we explore the essential characteristics of SDN, the technologies that provide the back end for the development of this technology, and current implementations of the most popular SDN controllers, so that we can see the features that should be present while developing a management interface for SDN controllers.

2.1 OpenStack

2.2 OpenFlow

As the growth of the networking infrastructure of the past few decades became evident, the need for an enviroment that allows for experimentation and testing of different protocols and equipment became evident. If networking research would depend on the previously existing methods, then new ways of creating and developing protocols would become increasingly hard to implement and develop. As such, there was need for a framework that could enable testing of new ideas on close to realistic settings. So, on February 2011 the version 1.1 of OpenFlow was released, and this proposal quickly became the standard for networking in a Software Defined Network. Since 2011, this protocol has suffered some revisions, and the latest version supported is version 1.5.1. Since this framework has evolved quite a bit, this section focuses on the versions 1.3, which are the versions that are used in development of this dissertation.

Several reasons led to the quick standardization of this protocol, which are related not only to the initial requirements of the platform, like the capability of supporting high-performance and low-cost implementations, and the capability of ensuring separation between production and testing traffic, but also the extensibility that the open source development model provides, removing the limitations that closed or commercial solutions give the network researchers.

The big advantage of OpenFlow is that it is, from the data forwarding plane point of view, easy to process. Since the control decisions are made by the controller, which lives in a separate plane, all the switch needs to do is correctly match the incoming packets, and forward them according to the rules established by the controller. The components that are part of this system and enable this functionality are:

- **FlowTables** This element describes the main component of the switching capabilities of the OpenFlow switch. Inside the switch there are several flow tables that can be used to match incoming packets, and process them in the rules that are specified by the controller. These rules can contain actions that affect the path of the packets, and these actions usually include forwarding to a port, packet modification, among others. Classification is done via matching one or more field present in the packet, for example the switch input port, the MAC and IP addresses, IP protocol, basically all information required to correctly process the incoming packet. The required actions for an OpenFlow switch are the capability of forwarding to a set of output ports, allowing the packet to move across the network; to send them to the controller, in the case of a miss of match; and finally the ability to drop packets, which is useful for DDoS mitigation, or more security concerns.

- **OpenFlow Protocol** Through the establishment of the OpenFlow Protocol between the switch and the controller, there is the definintion of several messages that allow for the control of the switch. This protocol enables capabilities such as adding, deleting and updating flow mods in the switch, that are refered to as *Controller-to-Switch* messages. Other relevant message types are the *Asynchronous*, that enable the notification of some event that ocurred, this type includes the Packet-In message, that is a type of message that is sent to the controller when a certain packet has no match in the flow tables present in the switch; and the *Synchronous* message that enable functionality such as the Hello message, that is used to start the connection between the switch and the controller.

- **Secure Channel** OpenFlow defines the channel that is between the switch and the controller as a secure communications channel. As the messages that are sent to the switch are critical for the correct operation of the system, as indicated in the previous point, the channel should be criptographically secure, to prevent spoofing of this information. As such, the channel is tipically transported over TLS.

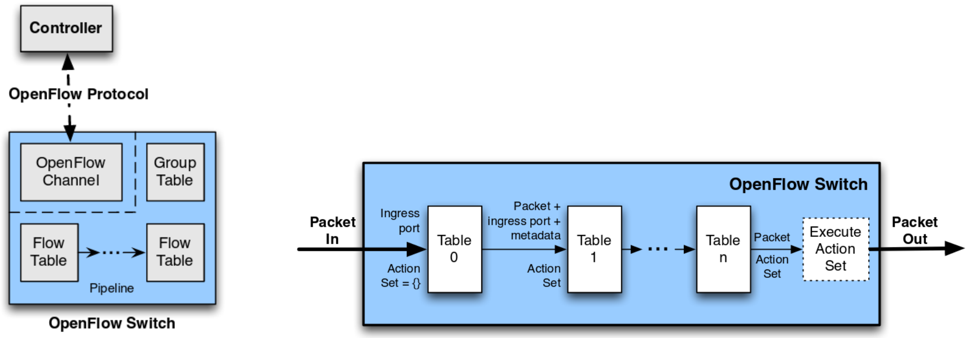

Figure 2.1: Images describing OpenFlow components. On the left, an overview to the entire system, and on the right a view at the table structure of the OpenFlow Switch

2.2.1 Open vSwitch

Although the OpenFlow protocol itself is well developed and well documented, there still needs to be support from switch applications to support this protocol. Although hardware switches are usually very expensive, and

2.2.1.1 Open vSwitch

2.3 SDN Controllers

2.3.1 Floodlight

2.3.2 OpenDaylight

2.4 SDN Northbound

Network Management

This chapter focuses on the management of networks, where we explore what is most necessary to obtain a comprehensive understanding of the network; formalize the statistics that can be reported via OpenFlow; see some research that has been done in the management of SDN applications, including what existing controllers provide us; and finally explore the way that DDoS detection and mitigation is usually implemented.

3.1 Introduction

As networks grow larger and more complex, systems must be put in place that allow for closely monitoring the resources that make up the network, while also allowing for a certain freedom for the possible constant change of the network. As such, typical vendor solutions don't really fit into this ever changing landscape, since they present very solid and vertically integrated solutions. The SDN paradigm, however, is able to solve this issue, since it enables for the centralized control of the underlying networks, which provides visibility and even control over the network, simplifying network diagnosis or troubleshooting.

Although SDN is a promising paradigm in terms of networking management, it also introduces some points of failure that are non existing, or not as impactful in current networking deployments. This is related, for example, to the centralization of the controller, which makes it susceptible to Denial-of-Service attacks or even the possibility of some malicious attacker that could possibly exploit the privileged view that the SDN controller has.

The topic of network management is very extensive, due to the several components that make up today's networks, and the vast amount of information that they provide. It can be summed up as the operation and maintenance of network infrastructure so that the service it provides is not only "healthy", but also is operated at a level that keeps costs down for service providers.

3.2 Requirements for management systems

As the complexity of the networks, and network devices that compose them grow bigger and bigger, the management systems should accommodate for the their necessities. As such, the basic groups of requirements for management functions are that defined in the ITU-T X 700 Recommendation [**?**] are:

- **Fault management** is the capability for detection, isolation and correction of abnormal operation in the system
- **Accounting management** provides ways to monitor the system resource utilization, and using this data to generate information about the costs that the operation of a certain resource will incur. This allows for better optimizing the network utilization of network, as it provides insights on how to plan the evolution of the network
- **Configuration management** is related to the maintenance and updates of hardware and software in the network, and the general setup of devices that allow to start, maintain and terminate services
- **Performance management** relates to monitor systems for the traffic utilization, response time, performance and logging histories. This allows to maintain Service Level Agreements (SLA) from the service provider and the client, providing better services even in cases of unusual traffic.
- **Security management** enables setting up security policies in terms of access control to resources, private information protection, among others.

A network management system usually consists of a centralized station, and management agents running on the network devices. Using management protocols, the agents can report to the station information about the its operational status, which includes information ranging from CPU load to bandwith usage. Typically this information can be retrieved by the controller polling the agents, or the agents sending information on their own, usually to inform status changes. Using this information, the network operator can get insight on the performance or possible errors of the devices that are monitored. In the next section, we explore one of the most popular management protocols, SNMP.

3.3 SNMP

The Simple Network Management Protocol is an IETF defined protocol that allows for the interconnection of networking devices, and provide a structured way to retrieve relevant information about these devices. As the name suggests, SNMP allows for a simplified approach to network monitoring, since it reduces the complexity of the functions that the management agent needs to comply with, which bring several advantages, like reducing the costs for development of management tools; provides a way to monitor, independently from different hardware providers the resources; and also supports freedom in extending the protocol in order to include other aspects of network operation. [**?**]

The architectural model of SNMP can be described as the following components:

The management database is one of the most important components of this system, because it serves as a reference to the entities that are managed in the SNMP protocol. The formal name for this database is the MIB - Management Information Base [**?**], and its composed of a collection of objects.

Each object has a name, syntax and encoding [**?**]. The name of the object, more specifically, the *Object Identifier (OID)*, is a reference to the object itself. This name is usually a list of integers, and

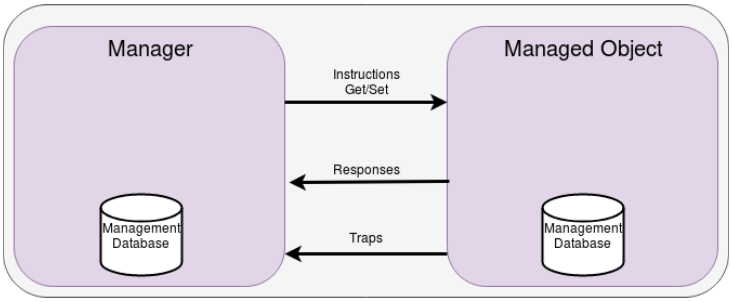

Figure 3.1: Architectural components of SNMP

they serve to build a tree-like hierarchy. This structure allows for the organization of all objects in a logical pattern, as there is a parent node, that contains references to their children, that provide different indexes for different objects. For human readability, there is usually a *Object Descriptor*, to refer to the object type. The syntax defines the type of data structure in the object type; and the encoding describes how the object type is transmitted on the network. In the context of this thesis, an important group is the interfaces group, as this exposes information about the interfaces present in a system. It's OID is the .3.6.1.2.1.2., and contains the number of interfaces in a system, and a table containing the counters related to the interface status, like the received unicast packets, the physical address, among others. The flexibility of the MIB allows for vendors to introduce their own databases into the MIB, while also remaining compatible with the standardized one.

Due to its permanence in the market, the protocol has suffered some large changes since its original design. SNMPv3 now supports important changes to the original one, most notably in the security aspects, introducing strong authentication and encryption capabilities.

3.4 Data Center Networks (DCN)

The rising demand of services like music and video streaming, or mass data storage, brings an increase of demand of compute and storage infrastructures, and the shift to the cloud computing model has led to a proliferation of large data-centers, containing thousands of physical nodes. Related to this growth is the focus on moving not only servers to a virtualised environment, by having one physical host several virtual machines and client applications, but moving also the networking functions to a virtual environment, by replacing the dedicated network hardware with generic compute resources, in a paradigm called *Network Function Virtualisation (NFV)* [**?**]. One of the bigger gains of using NFV, is the possibility of separation of each virtual network (VN), which guarantees better performance isolation and application of Quality of Service (QoS) rules [**?**].

The design of the network architecture is central to the data-center networks, as the placement for physical hosts and virtual machines allows for sharing the resources and create a logical hierarchy of network devices. The study on the design of DCN has resulted in the creation of typical DC topologies, like fat-tree topologies (as seen in 3.2), or others, including de Bruijn server only networks, or BCube switch heavy networks [?]. This approach allows for the traffic characteristics, resource consumption and costs of the networking devices be understood, so that causes for failure of this network are understood and mitigated, and the entire DC can run on the most optimal possible way. The organization in the DCN also allows for traffic in the network being resistant to failure scenarios, since there are multiple paths that can redirect packets to the correct destination, even if a link to a switch fails.

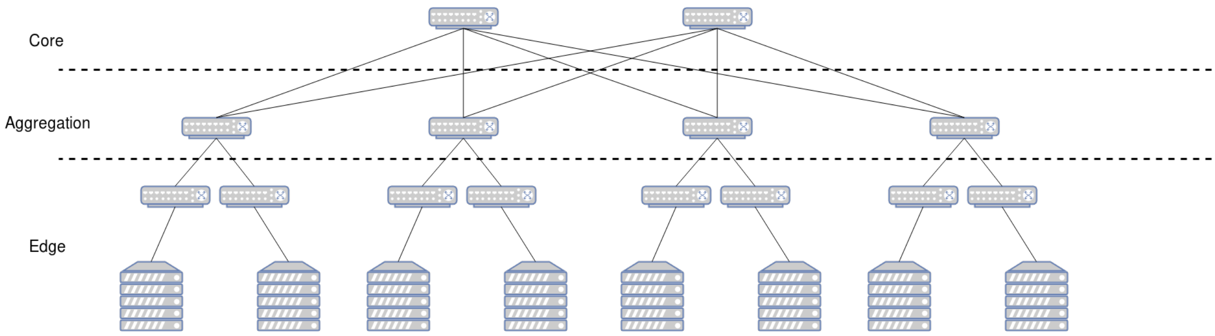

Figure 3.2: Visual representation of the fat tree topology commonly used in data centers

3.4.1 DCN Traffic

So that solutions for network management in data-centers can be developed, first there needs to an understanding of the traffic characteristics and resource allocation, and utilize this information to shape the DC fabric. The several studies proposed in traffic engineering for traditional networks do need to be revised in DCN's, since metrics like propagation delay, can be negligible, due to the physical proximity of nodes in DC's [?]. However, studies on DCN's have proven difficult, since many data-center operators do not wish to publish information about their applications and services. Also contributing to this fact is the impossibility of separating the different classes of data centers, since deployments across campus, private and cloud data-centers serve different purposes and have different applications.

By collecting data from different types of DC's, several studies have been made about the traffic characteristics [?, ?, ?]:

- The placement of VMs and servers effects the bandwidth and link capacity, due to the variety of applications that can be running on the servers at any time, and this non-uniform placement of VMs contributes to higher amounts of traffic originating from the same rack
- The majority of flows [1] are described as being small in size, and short in duration, which are usually described as *mice* flows. The counterpart to these are the *elephant* flows, which occupy a very large share of the bandwidth, and degrade application performance, due to choking effect to the latency-sensitive mice flows. Applications are tied to the type of traffic they generate, where online gaming, VoIP and multimedia broadcasting usually originate mice flows, where the large

[1] flows are usually defined as

data transfers and file-sharing originate in elephant flows. However, these flows which account for 80% of total traffic only occur less than 10% of total flows [?].

- In a normal situation, link utilization is low in the layers apart from the core switches. In addition to this discovery, losses are associated with spikes in traffic, instead being related to high utilization of the link, which is one of the effects of the previously mentioned elephant flows.

3.4.2 Limitations

3.4.3 DCN and SDN

The centralized view that SDN controllers maintain over the networks allows for it to keep the information about the flows currently present in the network. As such, in the SDN paradigm, there is possibility to flexibly control the path that the packets take in the network, and improve performance of the network at a large scale. By joining the information available on DCN and SDN, the requirements for traffic engineering (TE) in SDN, from the perspective of flow control are flow management, fault tolerance and traffic analysis [?]. This set of four requirements set the base for properly monitoring a DCN from the perspective of the SDN paradigm.

The next section are taken from [?].

3.4.3.1 Flow management

Flow management refers to the capability that the controller has to set rules for packet forwarding, and maintain the low overhead that is associated with registering a new flow mod, and also limiting the amount of flow entries, as hardware switches usually have a set amount of flow entries that it can support. Further in this document, we explore the effect that the amount of flow entries has on OVS, where increasing the amount of flow entries also increases the packet loss between two hosts.

If we consider the fat-tree topology, then one obvious result is the fact that if one controller is responsible for the management of the entire underlying topology, then one possible results is the creation of one bottleneck when the rules need to be deployed to a node. When the switch receives a new packet, and there are no rules to properly forward this packet, then the packet is redirected to the controller, on the form of a PACKET_IN message, and after processing this packet a new flow mod is sent to the switch. The problem with this scenario lies in the delay that it takes between the reception of the packet, and the installation of the new flow entry, which can be a contributing factor in packet losses in the data plane. This is an attack vector that is also explored in Distributed Denial of Service (DDoS) attacks for SDN platforms, as in an extreme scenario, the spoofed packet addresses will not have matches on the tables, which then result on overflowing the controller [?].

A solution for this issue is then related to decreasing the number of messages sent to the controller, by introducing some load balancing concepts. One of these concepts is related to the way that we can install the flow entries on the switch. The information present in the packets serve to generate the flow-match entries that are deployed on the table. To reduce the number of interactions between the controller and

OF switches, then we can reduce the number of match fields present in the flow mods, which reduces the number of flow entries on the switch and the controller messages. Another solution is related to distribute the controller among the network, but keeping them connected via a separate channel.

3.4.3.2 Fault tolerance

Although the switches are connected in a way that are able to mitigate link, or other switch failures, in the case of faults occurring there needs to be the possibility of creation of new forwarding rules. An even bigger concern lies in the case when the controller fails, which will pose a larger problem in the network. For the case of node failure, fast recovery means that the OF controller can reactively react on link failures, by signaling the switches to forward packets toward new locations; or proactively, by setting the rules prior to the occurrence of the failure. In the case that the failure is short lived, then the controller is also responsible of resetting the paths to the optimal state.

In the case of controller failover, then the backup controllers should act on this failure, and act as the new master. OF switches should connect to the set of available controllers, which should coordinate the management of the switch amongst themselves. After the switches first connection to the controllers, they should maintain this connection alive, but the controllers have the possibility of changing their roles. The controller roles are as follows:

- OFPCD_ROLE_EQUAL, where the controller has full access to the switch, receiving all incoming messages, and can modify the state of the switch
- OFPCD_ROLE_MASTER, which is a similar status to the previous one, but where the switch ensures that only one switch is connected as the master role
- OFPCD_ROLE_SLAVE is a role that controllers has read-only access to the switch, having no permissions for altering the state of the switch. The only message that controllers registered with this role receive are the port-status messages

As previously mentioned, the way that controllers handle their connection is independent of the OpenFlow connection, and the failover should occur with minimal changes to the underlying flow rules and overhead.

3.4.3.3 Traffic analysis

So that the management tools can correctly display information about the state of the network, status statistics should be continuously collected and analysed. These statistics should provide the information about flows, packets and ports, so that the measured metrics can serve as a baseline for the decisions of the controller to adapt the flow mods to enable the best possible performance. For the statistics collection there are two possible ways of getting the data: by continuously sampling packets from the switches; or applying sampling techniques, and generalizing the information from the sampled data [**?**].

The problem here lies in the collection of the statistics in poses a problem for large scale deployments, where continuously polling the network devices introduces both overhead and very large amounts of data to be parsed, or the data is not enough to detect failures in a short amount of time.

In the next section we present the mechanisms that can be used to store and publish statistics, and explore the meaning of flow, packet and port statistics.

3.5 OpenFlow

Part II

Fundamentals and Related work

Part III

Application and Results

Part IV

Conclusions

References

[1] M. Schwartz and N. Abramson. The Alohanet-surfing for wireless data [History of Communications]. *IEEE Communications Magazine*, 47(12), 2009.

[2] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in SDN-OpenFlow networks. *Computer Networks*, 71:1–30, October 2014.

[3] Hitoshi Masutani, Yoshihiro Nakajima, Takeshi Kinoshita, Tomoya Hibi, Hirokazu Takahashi, Kazuaki Obana, Katsuhiro Shimano, and Masaki Fukui. Requirements and design of flexible NFV network infrastructure node leveraging SDN/OpenFlow. In *Optical Network Design and Modeling, 2014 International Conference on*, pages 258–263. IEEE, 2014.

[4] Hiroaki Hata. A study of requirements for SDN switch platform. In *Intelligent Signal Processing and Communications Systems (ISPACS), 2013 International Symposium on*, pages 79–84. IEEE, 2013.

[5] Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, 2013.

[6] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On Controller Performance in Software-Defined Networks. *Hot-ICE*, 12:1–6, 2012.

[7] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On Controller Performance in Software-Defined Networks. *Hot-ICE*, 12:1–6, 2012.

[8] Md. Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data Center Network Virtualization: A Survey. *IEEE Communications Surveys & Tutorials*, 15(2):909–928, 2013.

[9] Christos Douligeris and Aikaterini Mitrokotsa. DDoS attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643–666, April 2004.

[10] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[11] Keith Kirkpatrick. Software-defined networking. *Communications of the ACM*, 56(9):16, September 2013.

[12] Pankaj Berde, William Snow, Guru Parulkar, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, and Pavlin Radoslavov. ONOS: towards an open, distributed SDN OS. pages 1–6. ACM Press, 2014.

[13] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6. IEEE, 2014.

[14] Robert W. Merriam and Elisabeth Feil. The potential impact of an introduced shrub on native plant diversity and forest regeneration. *Biological Invasions*, 4(4):369–373, 2002.

[15] Abraham Yaar, Adrian Perrig, and Dawn Song. Pi: A path identification mechanism to defend against DDoS attacks. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 93–107. IEEE, 2003.

[16] Jurgen Schonwalder, Martin Bjorklund, and Phil Shafer. Network configuration management using NETCONF and YANG. *IEEE communications magazine*, 48(9), 2010.

[17] Laura Feinstein, Dan Schnackenberg, Ravindra Balupari, and Darrell Kindred. Statistical approaches to DDoS attack detection and response. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 303–314. IEEE, 2003.

[18] Marco Canini, Dejan Kostic, Jennifer Rexford, and Daniele Venzano. Automating the testing of OpenFlow applications. In *Proceedings of the 1st International Workshop on Rigorous Protocol Engineering (WRiPE)*, 2011.

[19] Lucian Popa, Sylvia Ratnasamy, Gianluca Iannaccone, Arvind Krishnamurthy, and Ion Stoica. A Cost Comparison of Datacenter Network Architectures. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 16:1–16:12, New York, NY, USA, 2010. ACM.