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

February 15, 2018

Resumo

Abstract

...

Contents

List of Figures

List of Tables

Abbreviations & Acronyms

SDN	Software Defined Networks

Introduction

1.1 Context

1.2 Motivation

1.3 Goals

Software Defined Networking

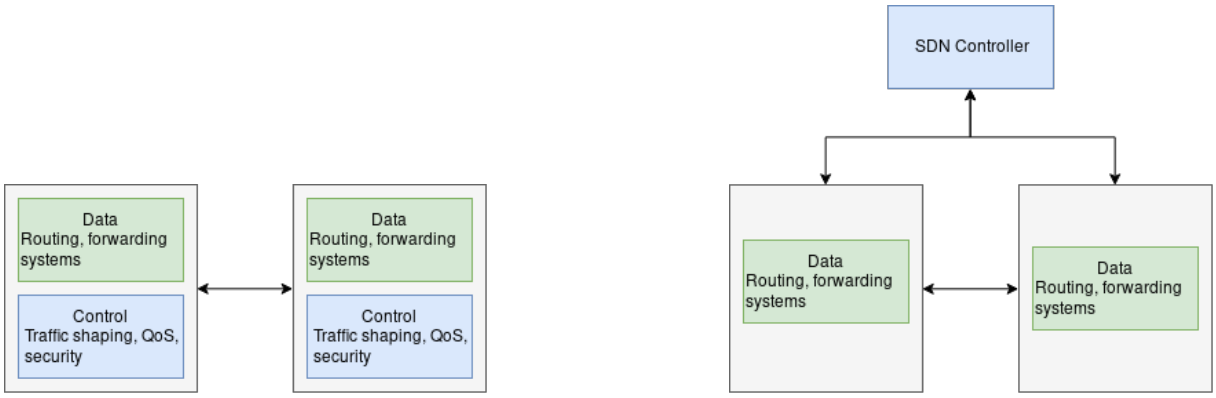
Computer networking is a vital part of the services that are offered today, and as such, the performance in technology backing these services is central to the quality of these services. As the service providers move their data centers to cloud computing environments, enabling several improvements in the predictability, quality of service and ease of use of their services, new technologies are 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**, which, as described by the Open Networking Foundation, *In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications* [**?**]. The two main contributions of this architecture is:

- **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, ...

This new paradigm introduces programmability in the configuration and management of networks, by consolidating the control of network devices to a single central controller, achieving separation of the control and the data plane, and supporting a more dynamic and flexible infrastructure. Another important development following the development of SDN is the concept of Network Function Virtualization. This concept allows to remove *middleboxes*[1], by replacing these with generic software applications.

By moving network infrastructure to SDN models, the difficulty of managing a network is greatly reduced, since the logical centralization of the control layer exposes the global view of the network, simplifying management tasks. Furthermore, this also removes the challenge of configure each networking device individually, turning network operation and management into setting high level policies in the

[1] Computer networking device that does some operations on traffic, excepting packet forwarding. Examples include caches, IDS's, NAT's, ..



(a) Traditional networking architecture (b) SDN architecture

Figure 2.1: Traditional vs SDN network architecture

controllers, and letting the protocols that handle connection between the devices and controllers set the actual rules.

Software-Defined Networking is defined as being composed of two layers: **Northbound Interfaces**, which are composed of Application Programming Interfaces (API) for communication between applications and the controller, enabling network services like routing, security, visualization and management; and the **Southbound Interfaces** which main role is to connect the network devices between the controllers and network via protocols like OpenFlow (see section 2.1), or P4 [2].

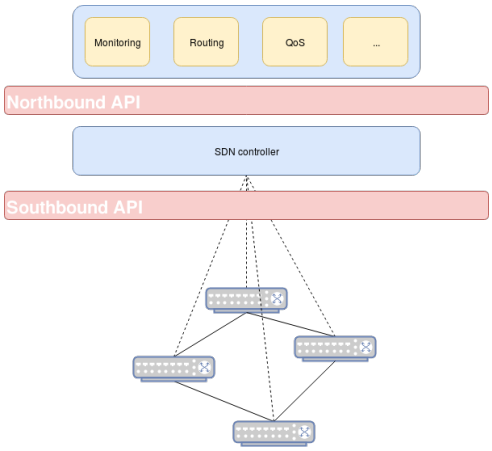


Figure 2.2: SDN Interfaces division

Understanding SDN platforms is then composed of understanding the operation of both interfaces, and defining requirements for their operation, which are listed below [**?**]. These requirements are general principles for networks, but the addition of the SDN controller introduces another point of failure, that could be damaging to the entire network.

[2] https://p4.org/

1. High Performance
2. High Availability
3. Fault Tolerance
4. Monitoring
5. Programmability
6. Modularity
7. Security

2.1 OpenFlow

With the growth of the networking infrastructure of the past few decades, the need for an environment 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.

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, 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 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 referred to as *Controller-to-Switch* messages. Other relevant message types are the *Asynchronous*, that enable the notification of some event that occurred, 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 cryptographically secure, to prevent spoofing of this information. As such, the channel is tipically transported over TLS.

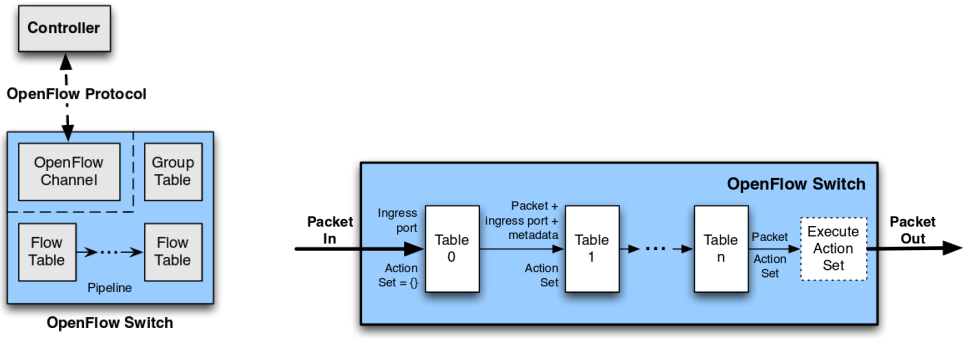


Figure 2.3: Images describing OpenFlow components. On the left, an overview to the entire system, and on the right a view at the table structure of the OpenFlow Switch

2.2 Network Devices

Networking devices are a central part of network operation, performing routing, switching, management operations, and span the different layers of the OSI model. The investment in dedicated hardware to perform management functions can potentially be replaced by SDN controllers, offloading the QoS policies and traffic engineering (TE) functions from hardware to software. As such, in this section we focus on the devices that are responsible for the operation on layers 2 and 3 of the standardized networking stack, switches and routers. A networking switch is a device that connects multiple devices on a computer network, using hardware addresses (MAC) to forward data inside the network, by mapping each port with a certain MAC address, while a router is responsible of forwarding packets between different computer networks. These devices run at different layers of the networking stack, the former operating

at the data-link, or layer 2, and the latter operating at layer 3, or network. This is not a clear separation however with multilayer devices, where switches also provide routing capabilities. Typical vendors for these solutions include Cisco, Juniper, but the rise of whitebox switches that have support for deployments in SDN environments enables network operators to avoid vendor lock-in, and take advantage of the open nature of these devices. Commonly associated with whitebox switches is the support for the OpenFlow protocol, making these an essential part of the SDN infrastructures.

The performance [3] of networking devices is central to the proper operation of networks, especially in deployments in Data Centers, where the interfaces must be able to support 100 Gbps links and further, while also maintaining the programmability that is expected of SDN based infrastructures. This performance is linked with the hardware that is chosen to serve as the base for the devices [**?**]:

1. **General Purpose Processors** provide the greatest flexibility of all the solutions, while providing the worst results in performance, due to the general purpose design of the hardware and the optimizations present in the other architectures.
2. **Field-Programmable gate arrays (FPGA)** are a platform that enable the configuration of the devices via hardware design tools, maintaining the programmability of the GPPs, while also allowing for designing the devices around the tasks that they perform, including optimizations for switching/ routing. A notable example of platforms based in these systems is NetFPGA [4], an open source hardware platform designed for research, and supporting up to 100G operation.
3. **Application-specific integrated circuits (ASIC)** are integrated circuits that are customized for one particular application, removing the programmability, but also providing the greatest performance of the former options.

These architectures generally allow to design SDN architectures around general purpose hardware, contributing to the flexibility of this paradigm, even considering the proprietary nature of the ASICs, which can be bundled with SDKs for developing other applications on top of these.

Considering OF enabled hardware switches, the processing of incoming packets is done as by matching a (up to) 15 field tuple [**?**] to several flow tables, that have rules sent from the controller. In these cases, the possibility of bottlenecks are due to several factors, including the latency of the installation of new flow rules, and the memory limitation on the hardware. Solutions to the memory limitations in OF switches include DevoFlow [**?**], which utilizes wildcard rules to reduce the number of flow entries that are installed on the devices, while also aggregating traffic, which simplifies detection and management of the elephant flows, due to reduced control plane load; and SmartTime [**?**] manages the timeouts for the rules on the switch, reducing this in the presence of microflows, and increasing the timeout in the case of the occurrence of longer lived flows, which improves memory utilization and reduces the load on the controller.

[3] Defined as the throughput and latency of the network node
[4] https://netfpga.org/site/#/

Virtualised environments also allowed the development of Software Switches, due to the highly dynamic nature of virtual environments, where Virtual Machines (VMs) can move between physical compute nodes and frequent network topology changes. Furthermore, standard Linux bridges cannot handle the multi-server deployments [5] used in virtualised environments. Open vSwitch is a soft switch that can join traditional switches on these platforms, replacing them where these couldn't be deployed. OVS switches are also compliant with the OpenFlow protocol, which clearly shows the flexibility that can be achieved by combining all the networking devices with one management protocol.

2.3 SDN Controllers

Responsible for

2.3.1 Performance Overview

2.3.2 Existing Platforms

2.3.2.1 Floodlight

2.3.2.2 OpenDaylight

[5] https://github.com/openvswitch/ovs/blob/master/Documentation/intro/why-ovs.rst

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 in figure 3.1.

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

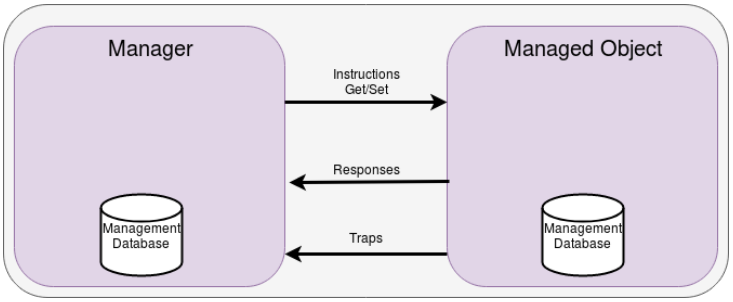


Figure 3.1: Architectural components of SNMP

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 **??**), 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.

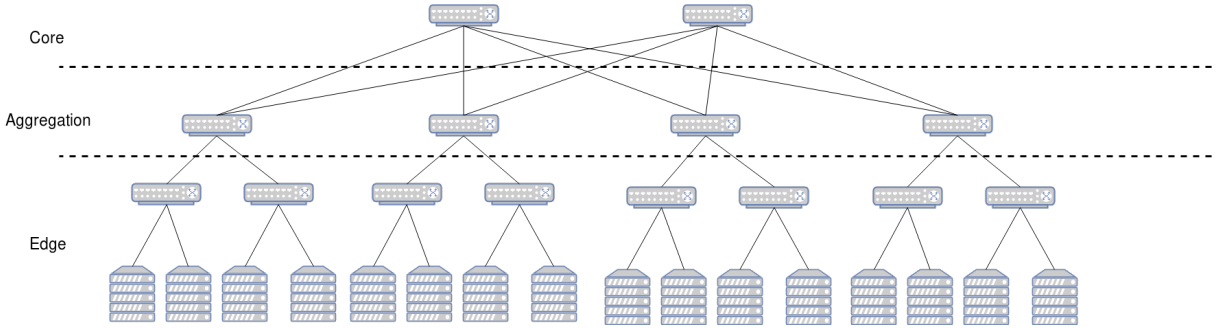


Figure 3.2: Visual representation of the fat tree topology commonly used in data centers

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 sequences of packets sent from a source to a certain destination (host, anycast or multicast domain)

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.

3.5 Statistics

3.5.1 OpenFlow

After exploring the requirements for network management, and the way the SDN model can support developing better systems, we now focus on the possibilites for obtaining this information from the networking devices. The OpenFlow protocol maintains a set of counters for each flow entry, port and group statistics, and this information can be queried to obtain a general view on the status of each OF switch. By sending specific controller-to-switch messages, the switch will return a set of the maintained statistics, which can then be parsed and analysed further.

Sending the port statistics message returns an array with the measured counters for each port. These counters include information like the amount of received and transmitted bytes/ packets, errors and dropped packets, and the duration that the port is alive.

The next important message is the OFMP_FLOW message, since this allows for getting the individual flow statistics, and obtain the information about each flow entry, including the time that it has been set on the switch, the number of packets/bytes in the flow, and the match fields. Also worth noting are the aggregate flow messages which describe how many packets are in the total flow entries, and also the number of flow entries that exist.

Also relevant is the information that is retrieved using the group statistics, as they allow to monitor the number of flows that direct to the group, and again the packet/ byte count that are matched with this group.

The information provided from these messages allows for a comprehensive view of the state of each switch, and a network management system (NMS) should utilize this information to achieve an understanding of the state of the network.

3.5.2 sFlow

One problem arises, however, when periodic requests generate too much information, and the control channel is overflowed with messages of port statistics, which is a possible scenario when the flow tables start getting too large. As such, a different alternative is to sample a small amount of packets from the switch, send the packet headers to the controller. One approach to this method is *sFlow*, a standard for collection, analysis and storage of network flows and traffic, for each device and its interfaces. sFlow is implemented using embedded agents on switches and routers, which compile interface and flow samples and exports them to the sFlow collector via datagrams.

Due to the problems that arise with continuously collecting traffic data, packet sampling has emerged as a valid solution to this problem, by collecting every n-th packet randomly. The simplicity of the technique allows for reducing the complexity of sFlow agents, and having the sampling operation being done in hardware, resulting in the collection of the samples being done at the same speed of the channel it is monitoring. This reduces the losses that are inherent to the sampling process, which leads to biased analysis of the traffic [**?**].

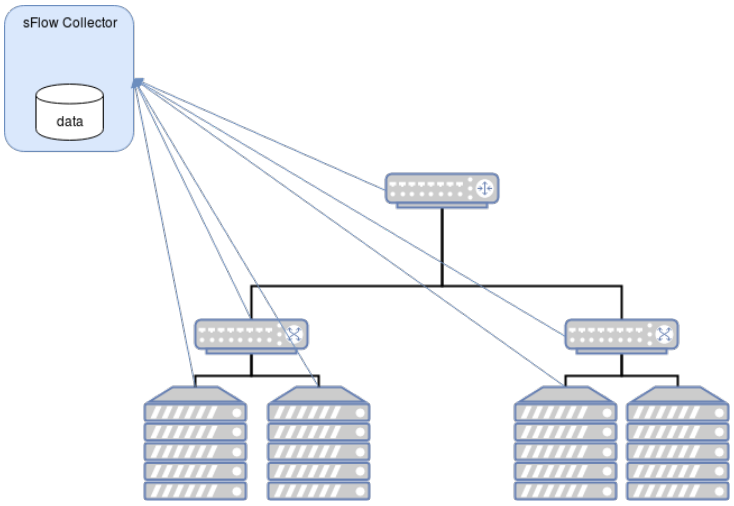


Figure 3.3: Architectural components of sFlow

Figure 3.3 shows the basic architecture that composes the sFlow system. One advantage of this system is the number of systems that incorporate sFlow agents [2], allowing for a detailed analysis of flows, and enabling flexibility for scalability in the network. By utilizing a sFlow collector that can accurately collect and process the datagrams incoming from the Agents, this protocol can be used to control most of the central aspects in network management, like troubleshooting network problems; controlling congestion on the network; or even analysing the possible security threats internal and external to the network.

[2]Complete list of compliant devices: http://www.sflow.org/products/network.php

BISDN

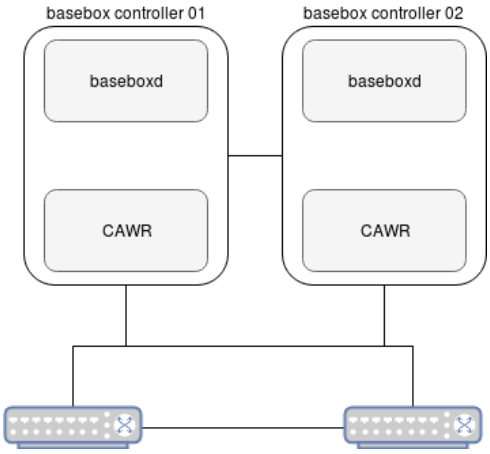


Figure 4.1: Basebox architecture

As the SDN market grows larger and larger in the networking world, new applications and products are developed and improved. Seeing the prevalence of closed source and proprietary solutions for this market, a need for open products that enable further growth and innovation in cloud DCNs is evident. The main gain in moving from vertically integrated solutions, is the decrease of costs involved, as cheaper solutions can be found in whitebox [1] switches and open sourced networking applications. With this motivation, BISDN developed Basebox, a Linux-powered solution to integrate switches and SDN controllers, allowing for data center operators to configure and manage networks using linux commands, removing the need for having to manage several devices with different interfaces and workflows, and adding the capability of running standard networking applications on top of the controllers and switches. Basebox also includes the possiblity of running in a failover scenario, by introducing a backup controller for the network, and the possibility of creating a giant switch abstraction, by adding another controller, CAWR, and having this manage all the southbound switches.

In this chapter we focus on this product, on the first two sections some characteristics of the developed product are described, then we focus on the development of a management API, presenting the required

[1] whitebox switches are

technologies that were implemented, and then finally display the results that were obtained in this part of the thesis.

4.1 Existing product

4.1.1 baseboxd

4.1.2 CAWR

Statistical Detection

As a result of the large scale of current data centers, maintaining control over these networks proves a difficult task. Networks operators must then adapt to the current situation by improving the monitoring infrastructures to allow faster response to problems. Building a feature complete management API for a SDN controller means that information obtained from the port and flow statistics previously implemented should return information about the global state of the network, so that root-cause analysis of the source of network issues can be done faster and easier, which reflect on better service and lower costs for network operators.

Network behaviour analysis is defined by the constant monitoring of a network, so that events that compromise the "healthy" state of the network can be removed or mitigated. These include not only cases where the anomalies are caused with malicious intent, like the case of DDoS attacks, but also failure of network devices or changes in user behaviour [**?**]. These systems are equipped with alarm capabilities, so that system administrators can quickly respond to changes, possibly even giving some information about the source of the problem. However, the automation of these monitoring processes means that the possible existence of false alarms reduces the operators capabilities to act on actual failures.

This chapter focuses on the recent research done in order to implement systems that rely on statistical analysis for monitoring the state of the networks and detection of abnormalities.

5.1 Elephant flow detection

Detection of network anomalies is subject to intense research, and as such, several methods were developed, that assume different levels of control over the network and provide different results to different applications. Our goal in this section is then to provide a description of the different types of network issues that can occur, and some proposed solutions for these issues.

The problem of detection of traffic anomalies has been subject to extensive research, and several different approaches have been proposed. These methods are based on different techniques [**?**]:

- Modifications to the applications and services to notify the controller about the state of the traffic on each service. Despite this approach resulting in the most accurate "detection" of network

anomalies, the support for this technique is not extensive, due to the required changes to each service, and it does not account for abrupt changes on the traffic

- By setting hard limits on the transmission capabilities of each port and switch, the controller is ensured of the non existence of flows that could impact network performance. This is a mitigation strategy that does not scale well to very large networks, since it requires the storage of the rules imposed to every port, and can potentially lead to the inefficient use of network resources, reducing flexibly on the DCN's.

- By employing sampling and collection techniques, using mechanisms like sFlow **??**, and building the profiles of the normal state of the network, this method can detect outliers that deviate from the normal state of the network. Utilizing this method reduces impact that continuously polling the network might have, while reducing impact on the packet and byte counts [**?**], but the loss of information inherent to sampling may be a challenge on successful deployment, which should account for optimal sampling strategies and inference from the obtained statistics.

- Periodic polling of the statistics from the switch, and employing statistical analysis methods to determine change points in the state of the network.

5.2 Time series analysis

Understanding processes and their results is a key factor in the success of implementing new features, or analysing existing ones for their efficiency and output. This analysis, important for the different fields in engineering, economics, health, allows obtaining information about the normal and abnormal state of each underlying process, provide forecasts and predictions on short and long term behaviour of the relevant data, classify and cluster information, and more. The act of collecting and processing the data, over a period of time is called *time series analysis*.

Monitoring systems provide a guarantee in quality engineering, since they allow to follow a system and its properties, and notify operators if changes happen that impact the normal status of a relevant parameter. These changes can be occasional or systematic, but should the state of the system deviate from the limits set by the operators, researching the cause of the errors can reveal some errors or malfunctions in the system. *Change detection* is the study of the different parameters of the system, and determining the points where these cause a significant deviation from the normal operation. In network traffic analysis, these methods can be applied to determine when the behaviour of certain flows, that can be monitored over metrics originating from the controller and switches, impact the traffic characteristics. One key part on the application of changepoint detection is the understanding and selection of the proper metrics to monitor, to ensure that these are sensitive to the traffic changes. One other important consideration in applying changepoint detection mechanisms is the reduction of false alarms, that occur when the metrics are too sensitive to traffic changes, and limit the network operators capability of accurately responding to real network issues.

Change detection mechanisms are classified as follow [**?**]:

- One of the most important distinctions in change detection theory is the difference between online and offline detection. Offline, or batch detection methods consider a fixed length of observations, and retrospectively analyse the dataset to determine the time where the change, or changes took place. Online detection, or sequential detection, unlike batch detection which uses all the available observations to detect the changes, including the ones obtained after the change took place, is based on the determination of the change points based on the arrival of the new data, allowing for determining the change as fast as possible. [**?**].
- An important criteria for the classification of changepoint detection algorithms is the knowledge of the distribution that the system has before the change occurs. A Bayesian system considers the
- Finally, the last important distinction is related to the scalability, and relate to the amount of data that needs to be stored to accurately implement detection on new samples. Parametric approaches rely on learning a probability distribution from the monitored variables, and using this learned data to estimate the unknown parameters, after which the training data can be discarded. Non-parametric models however, do not take into consideration the distribution of the monitored variables, and analyse statistical properties instead. The cost of this analysis is that the previous data must be stored to provide better results, but this problem can be mitigated using algorithms using sliding window or moving averages.

5.2.1 Mathematical formulations

A time series can be defined as a stream of observations $X = \{x_1, ...x_i\}$, where x_i is a vector arriving at time i. The time series X can also be described as the sum of the following components: S_t, which refers to the seasonal component of the data; T_t, which defines the trend of the data, and R_t represents the residual values, accounting for non expected variation and noise.

$$y_t = S_t + T_t + R_t$$

Or:

$$y_t = S_t * T_t * R_t$$

In regards to the classification of the trends according to the type of change, they can be classified as:

- **Deterministic** when the trend consistently increase/decrease
- **Stochastic** when the opposite happens

Approximating the time series data to a model allows for generating predictions for the next values, since the relationship between two variables in the data is then known. The Box-Jenkins method defines the steps of building the model as [**?**]:

1. **Identification** Model the data, by reducing the variables to a stationary state, and removing the possible seasonality in the series
2. **Fitting** Estimate the parameters for the model
3. **Checking** Verify if the model accurately fits the available data, returning to the identification step if its not adequate

Linear	$y = m \cdot x + b$
Polynomial	$y = b + c_1 \cdot x + \ldots + c_n \cdot x^n$
Exponential	$y = c \cdot x^b$
Logarithmic	$y = a \cdot ln(x) + b$

Table 5.1: Trend models

Time series data usually present a non stationary behaviour, that are characterized by changes in the mean and variance. Statistical methods require, however, stationarity in the data. The presence of trends and cyclic behaviours is the most common violation of stationary, and table **??** shows the most common trends present in the data, and the parameters that need to be learned from the time series in study. Removing systematic changes like trends and seasonal variation is possible with differencing:

$$\nabla X_t = X_t - X_{t-1}$$

By modelling the time series data with moving average (MA), autoregression (AR) or a mix of the two (ARMA) processes, under the assumption that the underlying process can be modeled by previous historical values, and assuming this model remains true for future measurements, the time series data historical behaviour can be used to generate forecasts for future values in the time series. The one step ahead prediction is referred as $\hat{x}_t$.

Exponential smoothing allows for generating predictions using the historical behaviour, but differs from normal MA methods by applying a set of weights to the data that exponentially decreases over time. Considering the time series X_t, the prediction $\hat{x}_t$ can be obtained by the equation 5.2.1. In this model, the smoothing factor $\alpha (0 < \alpha < 1)$ is obtained empirically, and its value will determine the forgetting factor for the past observations.

$$\hat{x}_1 = x_0$$
$$\hat{x}_t = \alpha x_{t-1} + (1-\alpha) x_0, t > 1$$

Due to its simplicity, this method is not suitable for situations where the data has trends, or seasonal behaviours [**?**]. The solution for this problem is introduced with double exponential smoothing, also know as Holt forecasting, and triple exponential smoothing, also known as Holt-Winters forecasting.

These methods introduce further components to dampen the effects of cyclic behaviours in the data. Double exponential smoothing is defined by [**?**]:

$$\begin{aligned}\hat{x}_t &= L_{t-1} + T_{t-1}\\ L_t &= \alpha x_t + (1-\alpha)(L_{t-1} + T_{t-1})\\ T_t &= \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}\end{aligned}$$

And triple exponential smoothing is defined by:

$$\begin{aligned}\hat{x}_t &= L_{t-1} + T_{t-1} + I_{t-1}\\ L_t &= \alpha(x_t - I_{t-s}) + (1-\alpha)(L_{t-1} + T_{t-1})\\ T_t &= \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}\\ I_t &= \gamma(x_t - L_t) + (1-\gamma)I_{t-s}\end{aligned}$$

The components these methods introduce account for the cyclic behaviours in the data: L_t accounts for the baseline behaviour of the data, which is calculated on the simple method; T_t smooths the trend with the β parameter; and S_t accounts for the seasonal components with the γ parameter. As with α these parameters are defined mostly by previous experience, but a common optimization is the minimization of the square sum of prediction errors, defined by:

$$SSE = \sum_{t=1}^{T} (x_t - \hat{x}_t)^2 = \sum_{t=1}^{T} \varepsilon_t^2$$

5.2.2 Change Detection

A relevant indicator for the validity of the generated forecasts is the forecast error, that is calculated via the difference between the real measurements and the predicted value. Furthermore, by assuming a distribution for the forecast error, and a certain significance level, it is possible to validate the generated forecasts, and detect values that do not fit the model, accusing a possible variation in the parameters of the model. As such, the prediction error is able to be employed in change detection algorithms. This prediction error can be defined as:

$$\varepsilon_t = x_t - \hat{x}_t$$

Hypothesis testing is used to perform a test of an assumption about two random variables. This hypothesis states that a certain relationship between the two variables exists with a certain significance value, and this relationship can be a relation in the means, the distribution of the observations, etc. The first step is set **null hypothesis**, which is the desired assumption to test, and is referred to as H_0, which is

then tested against the **alternative hypothesis**, H_1, which is considered true if H_0 is rejected. The validity of H_0 is based on a comparison between the two data sets, according to a certain threshold, called the significance level. This significance level also defines the probability of wrongly rejecting or accepting a hypothesis, which are defined as following errors: **Type I error**, happening when the null hypothesis is rejected, but the performed test is true, and the **Type II errors**, occurring when the opposite happens.

Originating from quality engineering, **control charts** are common ways of following the output of a certain process, with the aim of reducing variability associated to manufacturing processes. Control charts allow to evaluate the possible sources of variation, and classify the output of the process, based on the mean or variation of the sampled process, as **in control** or **out of control**, depending on the causes of variation. The process is considered in control when the parameters of the monitored variable, like μ_0 or σ^2 are inside the predefined **control limits**, that are usually set as requirements for the process. The wide range of control charts allow for a flexibility in choosing the right one that fits each application. Common charts used are those proposed by Shewhart, where the measurements from the process in study provide a statistic, like the mean, range, etc. Plotting these parameters allows for drawing the center line at H_0, and the control limits are defined by a multiple of the standard deviation. This control chart allows for actions to the process be performed not only when the points are shown to be out of control, but also when there is a sequence of values above or below the center line, or a upward or downward trend is shown in the control charts.

In the context of change detection, the hypothesis test relies on H_0 stating that there is no change in the parameters of the sample, like the mean or the variation, and the second hypothesis H_1 stating the contrary. A relevant metric for designing change detection models for application under network traffic is the *false alarm rate*, due to the impact that incorrectly identifying the normal state of the network as abnormal can have on the operators capability of addressing real abnormalities. The previously presented statistical difference of errors present in hypothesis testing are not relevant from the network operators point of view [**?**], which implies that the design of the change detection mechanisms should reduce the false alarm rate. In section 5.2.3 some mechanisms for these optimizations are explored.

The CUSUM (**cu**mulative **su**m) control chart provides a test based on *stopping rules*, where the alarms are raised when the parameter of the distribution, θ_t exceeds certain thresholds. In the parametric case, the CUSUM algorithm for detection of a change at t_0 from the observation x_i is based on the log likelihood ratio defined by [**?**]

$$S_t = \sum_{i=1}^{k} s_i = \sum_{i=1}^{k} \ln \frac{P_{\theta_1}(x_i)}{P_{\theta_0}(x_i)}.$$

The previous equation is related to the negative drift of S_t under normal conditions, and the positive drift after a change is detected. The alarm is raised when a test statistic g_t is larger than a threshold h, and can be obtained by

$$g_t = S_t - \min_{1 \leq i \leq t} Si \geq h$$

In the case when $P_\theta(x)$ is not known, the log likelihood parameter cannot be calculated, a non-parametric approach must be used, as mentioned in [**?**].

5.2.3 Performance Evaluation

The effectiveness of a statistical approach for change detection can be seen as the relationship between false and real alarms, since the trade-off with online detection is the number of falsely raised alarms, and the number of accurately reported changes. As mentioned in section 5.2.2 the possible wrong decisions are the type I and II errors, which are usually represented with error, or confusion matrices, seen in table **??**.

	H_0	H_1
H_0	TP	FP
H_1	FN	TN

Table 5.2: Generalized confusion matrix for hypothesis testing

The indicators for the performance of the change point identification are:

$$Accuracy: \frac{TP+TN}{TP+TN+FP+FN} = \frac{TP+TN}{N_{alarms}}$$
$$Sensitivity: \frac{TP}{TP+FN}$$
$$Precision: \frac{TP}{TP+FP}$$

The algorithm performance is defined by:

MTFA (Mean Time Between False Alarms) MTD (Mean Time to Detection) ARL (Average Run Length)

5.3 Existing methods

Monitoring SDN Switches

6.1 Problem

Developing solutions for use in mission critical environments requires the deep understanding and analysis of the requirements present, so that random occurrences and bugs do not happen in clients. As such, the problem posed is the construction of a management API and visualization tool that allows network operators to manage their infrastructure. The development of this thesis was done under the supervision of BISDN, and the final solution must integrate their already existing environments, and provide a way to manage them, by exposing an iterative way to display the topology, set VLAN's, and display the port statistics.

Also addressed by this system should be the setting of Quality-of-Service policies, that maintains the levels of accepted behaviour of each device in the network, and identifies and applies some automatic mitigation strategy when the system does not perform according to normal state. By understanding data center traffic characteristics, one of the largest problems are the existence of *elephant flows*, that impact the overall bandwidth of the network. As such, the development of a full management system should also include the definition of a system that receives the incoming port statistics, analyses these, and does some statistical analysis to manage and recover from the impact caused by elephant flows.

6.2 Solution

Following the previously presented requirements, the developed management system is composed of two components: the first is the Graphical User Interface (GUI), which provides an user friendly interface to display topology and statistics, and the second is composed of the policy manager, that enables the system to detect elephant flows and set rules to mitigate these. The next chapters describe the approach taken to the development of each component, in this section we describe in a high level way, the utilized approach.

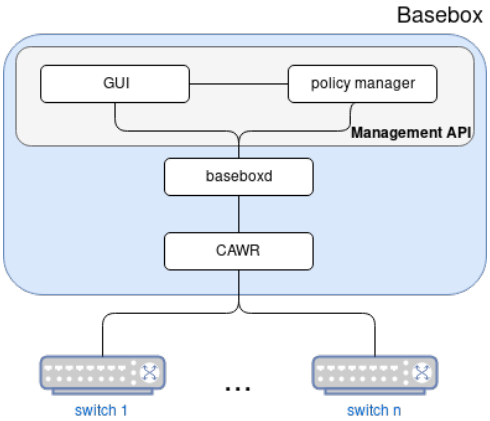


Figure 6.1: Visual description of the proposed system

6.2.1 GUI

The GUI serves as the primary interface for displaying the topology present in the underlying switches and the statistics collected by the OpenFlow protocol. The primary use case for this component is following the changes in the underlying topology, while also allowing the monitoring some aspects of the port statistics, and, as such, the links between switches and the hosts, and the association between the ports and the switches should be displayed. Performance wise, this system must run as fast as possible, and the transmission of data must not interfere with the systems operation. In order to reduce the memory requirements of the platform, and decrease the time that it takes for drawing topology updates, a decision was made to not store state in databases, which would increase the time it takes for topology updates, with queries to store and load data from these databases. This also removes complexity as the system grows, where storing information about links and switches would dramatically increase database size.

Motivated by compatibility with standardized systems, choosing the data model for representing the underlying system required the investigation of similar systems, and the RFCs, or similar standardized documents that exist.

Finally, the web server was designed to be lightweight, while also allowing for flexibility in adding more features and functionality. We chose the Django framework [1] as the backend for our webpage, as this is a current software that allows for fast development of web pages, and is implemented in Python, which means that we are then able to use the vast number of existing libraries for Python. For drawing the topology, the JavaScript library D3js [2] was chosen.

6.2.2 Policy manager

The second component of our system was developed for monitoring the system statistics, and displaying alerts when elephant flows are detected. For this end, we have implemented an algorithm for the detection

[1] https://www.djangoproject.com/
[2] https://d3js.org/

part, and then use the Linux traffic shaping tool *tc* to meter and shape the interfaces. Utilizing this tool has advantages, where we can use the existing Linux kernel libraries to set traffic shaping rules, and utilize baseboxd to convert into OpenFlow rules.

The development of this solution included designing a system that allow us to run different test cases under different situations. The test environment was then based on a virtualised network, *mininet* [3], which is a system that provides realistic environments for testing and research of networks, with support for OpenFlow switches, allowing for testing SDN controllers. However, due to different implementations of the OpenFlow protocol in the hardware switches, and the OpenFlow protocol enabled by Open vSwitch, testing baseboxd and CAWR with mininet proves a difficult task, and several functionalities are not able to be utilized in the same manner. For testing this component, a replacement OpenFlow controller needs to be found, especially one that operates in similar ways as the ones used in real deployments.

The developed system aims to detect the elephant flows before the effect of these is detected in the rest of the network. As we assume the tree topology in data centers, seen in figure 3.2, the testing environment must be designed to monitor the lower layer edge switches, connected directly to the physical hosts. By monitoring the ports on these switches, we detect changes in the reported traffic statistics via a developed Python script.

[3] http://mininet.org/

Management API

Due to the capabilities of Basebox of being a SDN controller used in data centers and a mission critical component for the network operators, it needed management capabilities, so that managing and operating infrastructure becomes an easier task. As such, the original problem presented was to build an interface extending the original work, so that the network statistics and the information of the topology could be easily displayed. There were several steps then necessary to understand the problem, and be able to choose the best approach to this problem. The requirements for the proposed system were:

- Display the topology information reported by CAWR, including the internal switch links, and the LACP discovered bond interfaces on the servers
- Display the port and link statistics for both switches
- Design an alerting system, so that network operators can be informed of changes on the network state
- Provide some diagnostic capabilities

The development of the work was the divided on two parts: the first part would be to implement the API needed to export the port statistics from baseboxd and CAWR, including a Graphical User Interface (GUI); and the second part was to study the alerting system, that would look into the statistics provided by the controllers, and design some rules so that QoS rules could be applied in the final product. This section describes the technologies needed to implement the API for Basebox.

7.1 Data models

Data models are abstract concepts that map the properties of entities and organizes their data, also defining how they relate to each other. To create a switch management interface, the entities we want to model are then the switches themselves, with attributes like the switch name, and the port counters, and the relationships of the data will allow us to display the links and topology of the network. One of the considerations that were taken into account when choosing the data models was the compatibility with standardized data models by the organizational entities like the IETF and the OpenConfig.

The *NETCONF* network configuration model, which we explore further in 7.2.1 also defines a data modelling language known as *YANG* , which is used in this protocol to model its configuration and data,and the remote procedure calls [**?**]. By utilizing models defined in this language, the following condition is met: since this is a specific language for configuration and monitoring of networking devices, the existing data models will be similar to the ones that should be employed in the development of the management API. YANG data model defines the hierarchy of data between a NETCONF client and server with the objective of smooth integration with the existing system's infrastructure.

The systems we aim to model with these requirements are then two: the topology between the servers and switches, and the port statistics for each port one the switch. Since there is no data model that would accurately describe both of them correctly, for the topology we chose the IETF network data model [1], and the OpenConfig interfaces data model [2], which contains the counters for each port.

7.1.1 Topology

The topology data model maps a collection of nodes, and the relationships between each node, called a link. This also allows for describing the network in a vertical hierarchy, by displaying relationships between several layers, which can then be used to display the entire networking stack, for example displaying the physical links between nodes, their connections at layer 2 and layer 3 of the OSI model, and the virtualised relationships that the elements could have in a cloud deployement. As the development of the product continues, and more features are added, for example, layer 3 routing, then we require a flexible data model that can be extended to support the new capabilities.

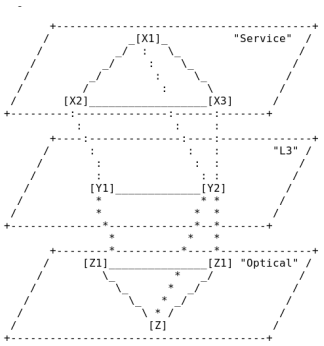


Figure 7.1: Example topology hierarchy achievable with this data model [**?**]

Mapping the data model to the real world data is then adding the two types of information the data model expects: the first one composed of adding the different networks that composed the entire topology, including their nodes and network types; and then using the previous information to build the links between each of the nodes, using the termination points the model exposes. In the implementation of

[1] https://github.com/YangModels/yang/blob/master/experimental/ietf-extracted-YANG-modules/ietf-network%402017-12-13.yang

[2] https://github.com/openconfig/public/tree/1040d11c089c74084c64c234bee3691ec70e8a9f/release/models/interfaces

the management API there was no need to implement underlying networks, but the extensibility this provides will be useful in the future.

Displaying the topology proved useful for CAWR, which provides the big switch topology, since this controller is directly connected to the underlying switches, and can see the links among these networking devices. The connection to the bonded interface on the servers can also be monitored, since these can be configured to use LACP messages to report their status. To display the links between the switches, the information that LLDP provides is used, and if the controller is extended to be able to use LLDP to the servers, the further information can be filled into this data model, and provide a richer view on the status of each server.

```
module: ietf-network-topology
augment /nw:networks/nw:network:
   +--rw link* [link-id]
      +--rw link-id            link-id
      +--rw source
      |  +--rw source-node?   -> ../../../nw:node/node-id
      |  +--rw source-tp?     -> ../../../nw:node[nw:node-id=current()/+
      |                         ../source-node]/termination-point/tp-id
      +--rw destination
      |  +--rw dest-node?    -> ../../../nw:node/node-id
      |  +--rw dest-tp?      -> ../../../nw:node[nw:node-id=current()/+
      |                         ../dest-node]/termination-point/tp-id
      +--rw supporting-link* [network-ref link-ref]
         +--rw network-ref    -> ../../../nw:supporting-network/+
         |                       network-ref
         +--rw link-ref       -> /nw:networks/network+
                                 [nw:network-id=current()/../network-ref]/+
                                 link/link-id
augment /nw:networks/nw:network/nw:node:
   +--rw termination-point* [tp-id]
      +--rw tp-id                          tp-id
      +--rw supporting-termination-point* [network-ref node-ref tp-ref]
         +--rw network-ref    -> ../../../nw:supporting-node/network-ref
         +--rw node-ref       -> ../../../nw:supporting-node/node-ref
         +--rw tp-ref         -> /nw:networks/network[nw:network-id=+
                                 current()/../network-ref]/node+
                                 [nw:node-id=current()/../node-ref]/+
                                 termination-point/tp-id
```

```
module: ietf-network-topology
augment /nw:networks/nw:network:
   +--rw link* [link-id]
      +--rw link-id            link-id
      +--rw source
      |  +--rw source-node?   -> ../../../nw:node/node-id
      |  +--rw source-tp?     -> ../../../nw:node[nw:node-id=current()/+
      |                         ../source-node]/termination-point/tp-id
      +--rw destination
      |  +--rw dest-node?    -> ../../../nw:node/node-id
      |  +--rw dest-tp?      -> ../../../nw:node[nw:node-id=current()/+
      |                         ../dest-node]/termination-point/tp-id
      +--rw supporting-link* [network-ref link-ref]
         +--rw network-ref    -> ../../../nw:supporting-network/+
         |                       network-ref
         +--rw link-ref       -> /nw:networks/network+
                                 [nw:network-id=current()/../network-ref]/+
                                 link/link-id
augment /nw:networks/nw:network/nw:node:
   +--rw termination-point* [tp-id]
      +--rw tp-id                          tp-id
      +--rw supporting-termination-point* [network-ref node-ref tp-ref]
         +--rw network-ref    -> ../../../nw:supporting-node/network-ref
         +--rw node-ref       -> ../../../nw:supporting-node/node-ref
         +--rw tp-ref         -> /nw:networks/network[nw:network-id=+
                                 current()/../network-ref]/node+
                                 [nw:node-id=current()/../node-ref]/+
                                 termination-point/tp-id
```

Figure 7.2: The IETF description for the nodes and links in the draft proposal for network topologies [**?**]

7.1.2 Port statistics

Modelling the port statistics to build a management interface requires first understanding of the OpenFlow statistics. As previously mentioned, OF switches maintain a set of counters, similar to SNMP, that provide information about the state of the ports, group, flow and table stats. The statistics that are exposed from OF are the following:

The chosen data model should then accurately model the fields that we need to expose, and the data type of counters we wish to measure. In this case, the prevalence of other controllers allows to use the same data models present in their implementations. OpenConfig [3] maintains a set of vendor neutral data models, written in YANG, allowing network operators to use standardized models for their networking infrastructure. The entire set of published models can be accessed in their github page [4].

[3] http://www.openconfig.net/

[4] https://github.com/openconfig/public

Table 7.1: OpenFlow port statistics

uint64_t	rx_packets	uint64_t	tx_packets;
uint64_t	rx_bytes;	uint64_t	tx_bytes;
uint64_t	rx_bytes;	uint64_t	tx_dropped;
uint64_t	rx_errors;	uint64_t	tx_errors;
uint64_t	rx_frame_err;	uint64_t	tx_over_err;
uint64_t	rx_crc_err;		
uint64_t	collisions;		
uint32_t	duration_sec;		
uint32_t	duration_nsec;		

7.2 Protocols

None of the controllers had a clear way of obtaining the statistics apart from manually looking in the terminal and following the logs exposed and waiting for the appropriate output. There needs to be then a controllable, to export this information and displaying them in a clear way. The solution was to develop a Graphical User Interface (GUI) for easily displaying the live statistics from the server, however there still was the problem of having to define the API that build the transport channel between baseboxd and CAWR to the GUI server. In this section we describe the two *Remote Procedure Call (RPC)* systems that were researched, and focus on the advantages which led to the final decision of implementing gRPC on Basebox.

7.2.1 NETCONF

Despite it's dominance on network management products, SNMP features some bad characteristics that pose an obstacle for the widespread use in network configuration, and not only network management, like [**?**]:

- Incompleteness of the devices features
- SNMP access can sometimes crash systems, or return wrong data
- Unavailability of MIB modules, which forces users to use CLI's
- Poor performance
- Security is difficult to handle

The IETF then, in light of this feedback obtained from network operators, started developing a protocol that allowed for the installation, manipulation and deletion of configuration of networking devices called NETCONF, which enables devices to expose a full API to their systems. This protocol, based in client/ server communication and is based in the four layers, as can be seen in the following image:

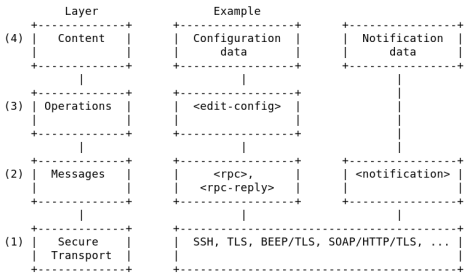


Figure 7.3: NETCONF protocol layers [**?**]

Data models and operations, covered in detail in the previous section, is related to the Content layer on the image, so this will not be covered in this section.

Configuration of a network device can be complex, and managing separate configurations between device startup and normal operation is a difficult task, but there is occasional need for this capability. NETCONF defines the existence of different *datastores* to enable this feature, allowing the network operator to set an initial configuration, used when the device is initialized, and switching to the running datastore when the device is ready to maintain normal operation. This concept of datastores also enables the creation of a candidate datastore, providing the capability of testing configurations on the network device, checking for any possible errors, while making sure that there is no impact on the current configuration of the device. After the changes have been tested and validated, a <commit> operation can be used to deploy the new configuration to the running datastore.

Another useful feature that is described in the NETCONF protocol, is the possibility of using the rollback-on-error capability. When rolling a new change, and if the system is enabled to support this feature, NETCONF can detect errors in the changes done to the configurations, and return the system to the previous state that is error free.

The NETCONF API provides several operations to interact with the managed devices to get system information and push new configurations. The set of supported operations in the base NETCONF protocol can be accessed in https://tools.ietf.org/html/rfc6241#section-7.

In regards to the transport layer, NETCONF is able to run on top of several protocols. However, NETCONF requires that a persistent connection is maintained between devices, and this connection should be reliable, and support transmission failure. In addition, the security should be handled by the transport layer [**?**], providing the guarantee that transactions are done in a cryptographically secure channel, between two authenticated hosts. As a results, typical NETCONF implementations are based on SSH or TLS protocols.

7.2.2 gRPC

The basic idea behind the RPC system is defining services by setting the interaction between remote systems, allowing for directly calling objects on remote systems. Based in the client/server communications pattern, gRPC allows for interactions between different environments, even implemented with different

programming languages, all based on the same data structure. This data structure can be serialized using another modelling language, called *protobuf*, which will define the data, which is defined as messages, and the services that contain the RPC calls between systems. Since this system is based on the HTTP2 transport layer, we are able to use the advantages that this protocol provides us.

Despite the serialization language used in gRPC is based on protocol buffers, unlike YANG, there are some projects [5] that enable the translation between YANG to protofile, which allows us to use the data models we chose, only adding one extra step to convert the files.

Despite both protocols capability of meeting the requirements that were presented to us, the gRPC framework was chosen due to several reasons:

- Both frameworks allow us to use the standardized data models currently proposed by the IETF and OpenConfig
- NETCONF trades information as XML encoded information, for both the edit and get config operations; while gRPC allows to handle information in a way thats native to the language implementation of the client/server
- The integration with the existing system was easier: since gRPC has implementations for the languages that the controllers are developed on (i.e. C++/ Python), this framework was easier to implement than NETCONF, which would have required integration with third party tools, or a longer development cycle to make sure that the developed applications would met the requirements

7.3 Results

[5] https://github.com/openconfig/goyang

Elephant Flow Monitoring

8.1 Testing Environment

The design of a testing environment must allow for the accurate simulation of the traffic conditions on the real DCNs, and should provide the flexibility to understand and change the underlying topologies. There requirements clearly indicate a strong motivation for deploying a testing environment in a virtualised environment, using tools like *mininet*, which provides a miniature network that can be changed as needed. This testing suite provides a strong alternative to deploying these changes in hardware.

Despite the developments previously made to the SDN controllers, utilizing these in combination with the virtualised environment poses a challenge, related to the implementation of the OpenFlow protocol in the hardware and software switches. Hardware switches that were used for testing in the implementation of the GUI have a modified version of the OpenFlow tables structure, OFDPA [1], and the libraries that make up the controller are designed around this. To utilize the controllers, changes to OpenvSwitch would be required, or an alternative would have to be discovered, which would limit functionality of the controllers.

To solve this issue, and to have a stable controller, that works as intended, for this step we adopt a different controller, and focus on the mechanisms that are also present in the hardware controllers. Furthermore, researching other approaches provides ideas that can later be adopted in these. The chosen controller was Floodlight [2], for it is continuously updated, and provides a REST API for obtaining statistics, setting table rules.

These elements compose the testing environment that seen in 8.1. To ease the installation of the utilized applications, we based these applications on VMs and containers, and the installation files can be found on the following page example.com.

[1] OpenFlow Data-Path Abstraction
[2] XXX - insert link here

Figure 8.1: PLACEHOLDER

8.1.1 Performed tests

8.1.1.1

8.2 Algorithm design

8.2.1 Formal definition

$$x_i = \begin{bmatrix} B_{RX} \\ P_{RX} \\ B_{TX} \\ P_{TX} \end{bmatrix}$$

B_{XX} and P_{XX} describe to the port statistics obtained from the controller, the byte and packet counters, respectively, and the indexes describe if the counters account for the transmitted or received data in that port.

A time series y_t can be decomposed in the following parameters: S_t which accounts for the seasonal component of the time series data; T_t accounting for the constant trend in the data, and R_t, which accounts for the residuals. These can be combined to the following equation:

8.3 Testing setup

8.4 Tests

8.5 Results

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.