Software Defined Networking

Luís Sá Couto, Carolina Fernandes, Rafael Belchior

Abstract—The advent of computing through techniques like virtualization, containerization or cloud services has created a gap between computing and networking. In fact, current networking techniques are not capable of complying with the rising performance and monitoring requirements of today's technology landscape. To fill in this gap, software defined networks appeared. This new approach makes networks programmable and, thus, more suitable for both end users and enterprises' requirements. One of the many things that such networks enable is a fat tree topology for data center networks, which allows for increased bandwidth at a minimal cost, enabling high scalability requirements. To illustrate how these new approach to networking enables such a topology, we provide an illustrative data center network scenario that uses an SDN controller, Open Daylight, and Mininet to emulate the network.

Index Terms—Software Defined Networking, OpenFlow, Open Daylight, Mininet, Fat Tree Topology.

1 Introduction

THE operation and control of networks did not parallel the increasing levels of automation in computing that containerization and virtualization brought about. It seems clear that both end-users and enterprises, have dramatically increased their needs for fast, innovative and highly adaptable networking resources.

Classic networking techniques depend on very complex routing protocols that perform local control actions. This inability to provide a global perspective makes it unfeasible to comply with the rising operation and control requirements that the rapidly changing technology landscape presents.

In addition to the lack of a global control view, most of the networking software that is used is vendor specific and highly coupled to its host hardware, making it very hard for researchers to solve current networking conundrums like the changing traffic patterns and the increase of handled data.

The advent of cloud services and virtualization enabled the softwarization of computing. This allowed for, between many things, vendor independence and rapid innovation capability. Such advances created an unignorable gap between the level of automation of computing and that of networking.

In fact, the highly manual networking techniques that are used today remit to times where computers did not even have an operating system. Hence, a new approach to manage the interconnection of devices seems inevitable.

It is with the intent to fulfill the aforementioned gap that Software Defined Networks (SDN's) emerge. This new architecture for networks leverages simple techniques and technologies to provide a clear abstraction between the "what" and the "how" of a network. With this abstraction, one can, not only have a global view of the network, but also separate data from control to, in a simple manner, comply to nowadays' requirements.

The rest of this paper will focus on: first, elaborating the limitations that current networking exhibits; second, set the scene for SDN's; third, study how a SDN can be leveraged to implement a data center network scenario; and fourth, a step by step guide to implement that scenario.

- *Luís Sá Couto, nr. 79078,*
  E-mail: student1@tecnico.ulisboa.pt,
- *Carolina Fernandes, nr. 79023,*
  E-mail: carolina.c.pereira@tecnico.ulisboa.pt,
- *Rafael Belchior, nr. 98765,*
  E-mail: student3@tecnico.ulisboa.pt,
  Instituto Superior Técnico, Universidade de Lisboa.

Manuscript received January 6, 2018.

(1.0) Excellent	WORK OBJECTIVES						DOCUMENT QUALITY						
(0.8) Very Good	Goals×2	Skills×4	Accur.×4	Solut.×6	Recom.×2	SCORE	Struct ×.5	Abstr×.25	Concl×.25	Ortog ×.5	Gram ×.25	Form ×.25	SCORE
(0.6) Good													
(0.4) Fair													
(0.2) Weak													

2 The Limitations of Current Networking

Current networking technologies and architectures are incapable to fulfill the needs of today's applications. The widespread vertical hierarchy architecture [1] was suitable for the classic client server interaction [1]. However, virtualization and cloud services changed the way software is built [2]. Nowadays, distribution is the key to achieve the scalability, the fault tolerance and the real time responses that current users need. This distribution allied to Virtual Machine (VM) migration [3] provides a very complex and unpredictable traffic pattern, even for the simplest of the systems. These rising requirements pose ponderous challenges to an highly obsolete networking architecture, uncovering its limitations. Such limitations can be described across three main issues that will be covered throughout the following subsections.

2.1 A Complex Set of Protocols

Protocols are defined individually in the sense that each tries to solve a specific problem and none has a wide overview of the interconnection between them. Very complex routers with thousands of lines of code implement these protocols. Although these routers and protocols achieve their reliability and functional purposes, they are characterized by a great amount of complexity. Furthermore, networks built in this manner are inherently static.

The absence of dynamic capabilities of current networks is in clear contrast with the flexibility that computing exhibits. In fact, the different types of traffic and needs that applications and users impose are, at the time, being dealt with manually by network providers.

2.2 Inability to Scale

Data centers are growing as flexible computing resources become widespread [4]. So, with current networking techniques, the number of devices to support this growth will burgeon too. Since configuration is primarily manual, it is infeasible to keep up with connectivity needs of the millions of virtual servers that might produce petabytes of data.

Besides managing increasing numbers of servers and bandwidth, current architectures make it hard to divide multi-tenant networks between tenants without safety issues [5]. Furthermore, it is not possible to effectively manage resources with, for instance, adapting bandwidth.

2.3 Closed Equipment

Both end-users and enterprises have fast changing needs that arise from consistent technological developments. However, since networking software is vendor-specific and is highly coupled with vendor-specific hardware, these suppliers can become a bottleneck, dictating the rules of the research on the area. Furthermore, the fact that it is usually closed software appears as an obstacle to innovation, making it hard for new relevant features to reach the market.

3 Software Defined Networking

The movement behind SDN's conveys that networking devices should be programmable just as other computing platforms are. SDN provides a decoupling between the control plane and the data plane where forwarding happens. By doing so, it allows for network engineers to work over a logical abstraction of the network. Furthermore, control is centralized in an SDN controller that manages all the switches.

To effectively capture the intuition behind this technology, one can look at an SDN as a three layer architecture with discernable levels of abstraction, just like today's computers [6]. Figure 1 provides an introductory overview of an SDN.

At the lower level of abstraction, we have the hardware, which, in this case, is composed by very simple and, thus, very fast switches. In fact, by centralizing control, these devices become just forwarding machines with no need for very complex code to implement very complex protocols. At this layer, devices have their flow tables populated by instructions that come from the controllers.

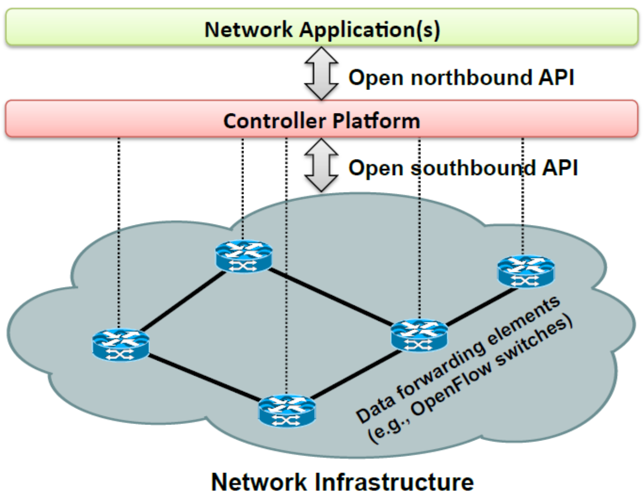

Figure 1. An overview of the architecture of a SDN's.

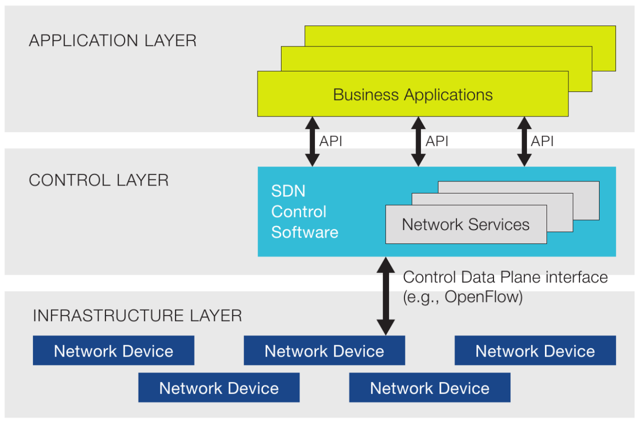

Figure 2. A detailed representation of the architecture of SDN's.

Controllers are on the middle layer and send instructions to the switches through a southbound Application Programming Interface (API) which is usually implemented with OpenFlow [7]. Furthermore, these controllers provide a northbound API, usually a Representational State Transfer (REST) API, that allows for network applications to run. With this outline, one can see that this control layer functions as the operating system of the whole network, providing a logical abstraction of the hardware below and enabling independent applications to run software that leverages the abstracted hardware.

As stated above, there is an application layer running on top of the control layer. These applications work over a global abstraction of the network and can, thus, implement several network services in a very simple and hardware independent manner. In fact, the whole network becomes programmable by these applications. Figure 2, summarizes the described architecture.

To have a fully programmable network brings no small advantages. It completely changes the limitations of current interconnection technologies. A simple overview of how deeply SDN's can fulfill today's computing needs can be structured around four key principles.

1) Network operators can, by writing simple code, configure automatically lots of network devices allowing for scalability.
2) It becomes possible for providers to write innovative code to create new network services without waiting for hardware vendors to provide more features.
3) It becomes very easy to manage security and control in multi-tenant networks. For one can assign an SDN controller to each tenant so they can control their share of the network. Furthermore, this type of separation can also ensure security policies.
4) High bandwidth can be established with new kinds of data center network topologies so the east-west kind of traffic [8] that current servers demand can be dealt with. The next section focus precisely on one of this topologies that is enabled by the implementation of SDN's.

4 Fat Tree Topology

Current data centers are made of servers grouped in racks. Such racks are connected through a data center network, usually organized in a tree like structure of switches [9]. Current network technologies are structured in binary trees, where great amounts of data are exchanged. To cope with this increasing amount of data, such networks must have increasing levels of bandwidth in their connections. Figure 3 shows a typical architecture for one of these data center networks.

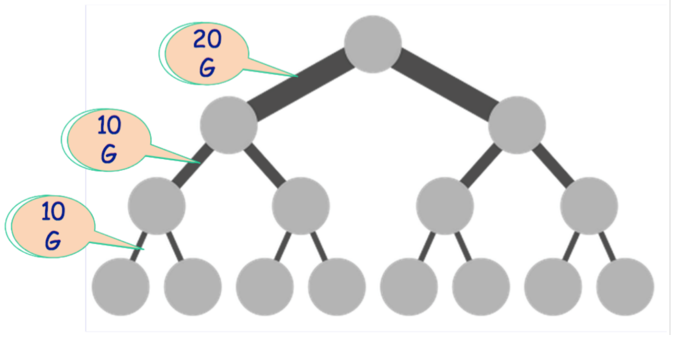

Figure 3. A binary tree topology with increased bandwidth on the upper links [9].

Although increasing the bandwidth of the upper level links can help to deal with the volume of data packets that go through the network, it introduces two main issues [9]. First, large bandwidth cables are expensive and so a trade-off between provision capability and cost emerges. Second, such a topology tends to present a single point of failure [10], since if one of these connections fails some hosts can become unreachable.

To surpass the limitations of the classical three layer topology, a new approach was proposed in [11]. In this topology, every link has the same small (and thus cheap) bandwidth. However, every switch as several connections to both their child and parent nodes. With this approach, several paths exist between every pair of nodes. Thus, there is neither a single point of failure nor increased cost to add bandwidth. Instead, bandwidth is gained by exploiting the several available paths connecting two nodes. Figure 4 exemplifies a topology of this kind.

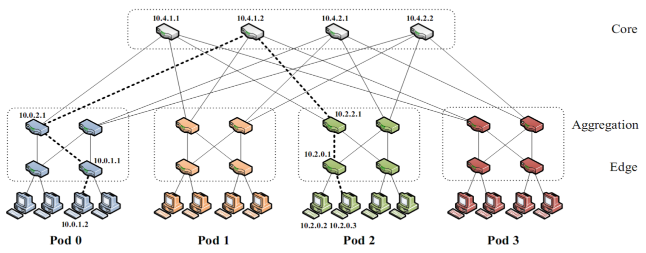

Figure 4. A fat tree topology [9].

The advantages of using a fat tree topology seem clear and have been explored in [9]. Yet, implementing such an architecture with current networking techniques would be a routing nightmare, subject to flooding and blocking [1].

So, SDN's come in as the real enablers of this topology. With centralized routing, defined programmatically at the control level, a routing algorithm of shortest paths can be implemented to effectively exploit the large connectivity of fat trees. In fact, in the original paper [11] a routing algorithm was proposed.

5 Open Daylight, an SDN Controller

With more than 1 billion subscribers, Open Daylight (ODL) [12] is a modular open-source controller for customizing and automating networks of any size and scale, with a big focus on network programmability. ODL is also the core of broader open-source SDN frameworks, such as Open Network Automation Platform (ONAP) [13], OpenStack [14], and Open Platform for Network Functions Virtualization (OPNFV) [15].

SDN controllers like ODL act as the network's operating system providing the bridge between the hardware switches and the networking services applications. ODL is an open source SDN controller, that fulfills its purpose through two APIs.

On the one hand, a southbound API leverages the OpenFlow protocol to communicate with the to the network's switches in order to manage their flow tables.

On the other hand, a northbound API receives REST instructions from applications that implement the business logic. In practice, an eXtensible Markup Language (XML) request like the one in figure 5 is sent through an Hypertext Transfer Protocol (HTTP) *PUT* operation [16] to the ODL controller. Moreover, other operations like the removal of flows can be also achieved through a combination of XML requests through HTTP.

Furthermore, a controller of this sort contains a collection of modules that perform different network tasks, such as gathering statistics or run specific algorithms to implement basic network services. Such modules can be installed through ODL features.

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="urn:opendaylight:flow:inventory">
   <priority>PRIORITY</priority>
   <flow-name>NAME</flow-name>
   <match>
      <in-port>INPORT</in-port>
      <ipv4-destination>DESTINATION_IP</ipv4-destination>
      <ipv4-source>SOURCE_IP</ipv4-source>
      <ethernet-match>FOR_MAC_ADDRESS</ethernet-match>
   </match>
   <id>FLOW_ID</id>
   <table_id>FLOW_TABLE_ID</table_id>
   <instructions>
      <instruction>
         <apply-actions>
            <action>
               <output-action>
                  <output-node-connector>OUTPORT</output-node-connector>
               </output-action>
            </action>
         </apply-actions>
      </instruction>
   </instructions>
</flow>
```

Figure 5. A template of an XML to add a flow.

Figure 6 presents a basic overview of the controller's architecture.

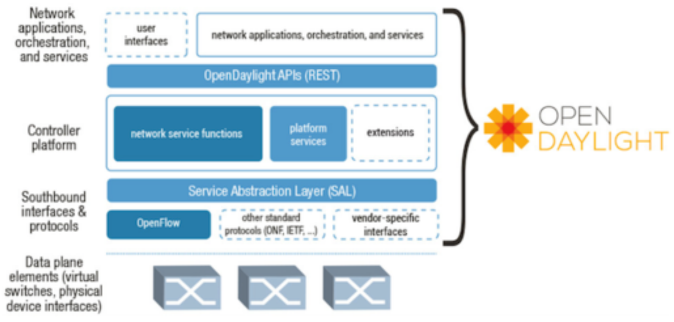
Network applications, orchestration, and services
user interfaces
network applications, orchestration, and services
OpenDaylight APIs (REST)
Controller platform
network service functions
platform services
extensions
Southbound interfaces & protocols
Service Abstraction Layer (SAL)
OpenFlow
other standard protocols (ONF, IETF, ...)
vendor-specific interfaces
Data plane elements (virtual switches, physical device interfaces)
OPEN DAYLIGHT

Figure 6. ODL arquitecture.

To complete our overview of the ODL controller, we provide a shortlist of its main use cases [17]:

1) Automated service delivery provides basic services that may be controlled by the end-user or the service provider.
2) Agile service delivery on a cloud infrastructure.
3) Network Resources Optimization, which aims to dynamically optimize the network based on load and state.
4) Administration of the network and/or multiple controllers.

6 A Data Center Network Scene

For the purposes of this project, a scenario of a data center network was chosen. The goal was to implement a fat tree topology with a SDN technology where a flow manager application runs on top of an ODL controller to manage the network's flows. The aforementioned topology can be seen in figure 7.

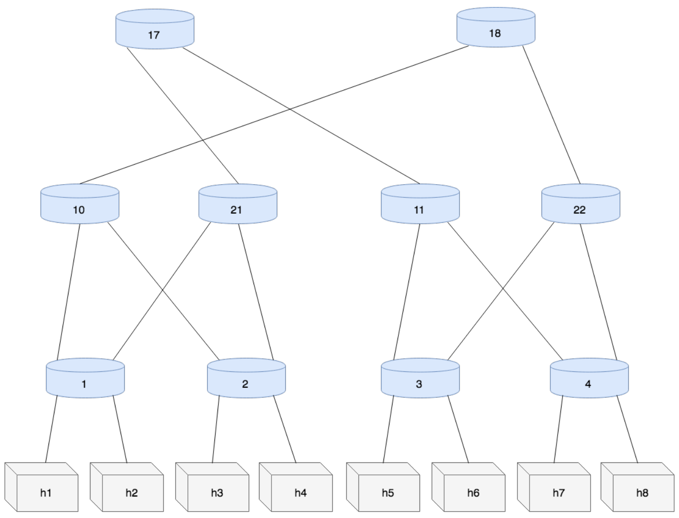

Figure 7. A simple example of a fat tree data center network.

A controller is connected to all of the switches and manages them using the OpenFlow protocol. Figure 8 presents the schematics of the interaction.

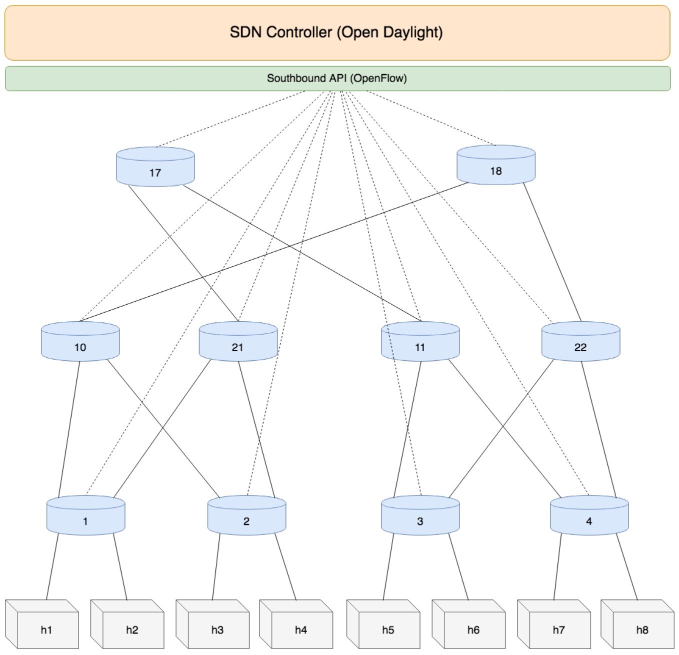

Figure 8. A data center network with an ODL SDN controller that uses OpenFlow to configure the network's switches.

On top of the controller runs a flow manager application that uses the REST based northbound API to perform requests to define end to end flows in the network. After that, ODL will exploit OpenFlow to assign specific forwarding rules to switches in order to fulfill those flows. In figure 9 one can see an overview of the full scenario.

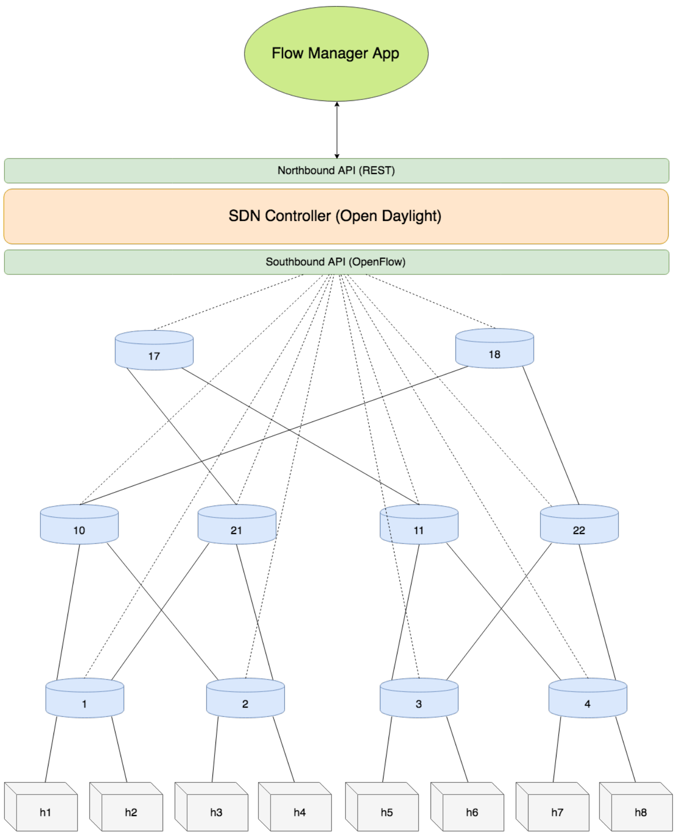

Figure 9. A flow manager application that uses REST to manage the network's flows.

The aforementioned application is implemented in Python and it is able to gather topology information and statistics concerning the network. Based on said information, it is possible to make informed forwarding decisions when populating the switches' flow tables. The next section focuses on how we intend to set up a demonstration scenario.

7 Setting Up the Scenario

In general, five main steps are required to set up a workable simulation of the previously described scenario. All of which will be covered throughout the following five subsections.

7.1 Spawn the Infrastructure

For the purposes of the scenario's demonstration, the infrastructure deployment tool Vagrant [18] can be used. At a glance, a *Vagrantfile* can be configured to create the three main VMs where all happens: first the *mgmt* machine which does not only manage the needed software configurations, but is also the machine where the flow manager application runs; second, the *opendaylight* machine where the controller is launched; and third, the *mininet* machine where a Mininet [19] emulation of the fat tree network is built.

Figure 10 presents a global view over the described infrastructure.

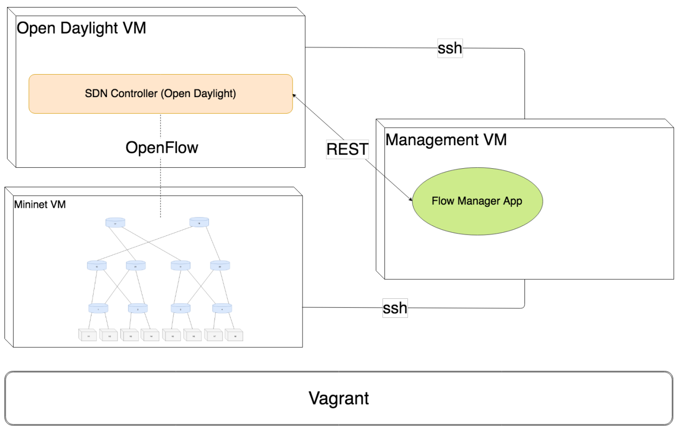

Figure 10. A vagrant infrastructure for the demonstration.

7.2 Setting a Secure Shell (SSH) Trust

To configure the previously described infrastructure, the *mgmt* node will be used as a manager that sends commands to all the other nodes. However, for security reasons, every time a connection between two nodes occurs, the user is prompted for a password.

For practical reasons, we would like to avoid the aforementioned prompts at each connection between the *mgmt* node and the others. To achieve just that, one can establish an SSH trust.

To implement such a trust, we use Ansible [20] with *playbooks* that were provided on the course's lab classes. Such *playbooks* distribute the necessary key files to establish secure but password-less connections.

7.3 Setting Up the SDN Controller

The next step is to install ODL on the *opendaylight* node. To do that, we can handcrafted anoth *playbook* that, on the one hand, installs all the required software (namely JAVA) and, on the other hand, downloads and installs an ODL distribution.

However, just running the said *playbook* does not suffice for the purposes of the demonstration. In fact, a very useful feature of ODL is its graphical interface, which has to be installed by hand. Hence, we had to create an SSH session with the *opendaylight* node to install it.

7.4 Start the Network

The previously described fat tree network, is, for the purposes of this demonstration, emulated by Mininet. To launch it, one just needs to put together a Python script that defines all the nodes and their connections. After that, the emulation tool has a feature that can build an emulated net from such a script.

7.5 Run the Flow Manager Application

Finally, we can produce a Python based flow manager application that contains several useful functions that define the network's flows. In particular, we distinguish the application's capability to build a graph of the network, where each connection's weight is directly proportional to the average traffic passing through it. Using such a structure, we use Dijkstra's algorithm [21] to compute the shortest paths (i.e. paths with minimal cost) between every pair of nodes to define the best possible forwarding rules. After that, we use the REST API to push those end to end flows to switches' tables.

8 Conclusion

Networking techniques did not follow the breakthroughs on computing techniques.

In fact, virtualization, containerization or cloud services either for end user or enterprises created new connectivity requirements that evinced the limitations of an hardware and vendor dependent industry such as the one of networking.

The traffic patterns in today's data centers are changing with large amounts of data being exchanged between the several racks of servers. This creates a need for high bandwidth while keeping reliability and cost effectiveness. Plus, technology is due to keep exploding and, so, scalability is also key.

To fill in this gap, SDN's appeared. This new approach separates the data plane from the logical plane, making networks programmable and, thus, more suitable to computing's rising requirements.

Among other things, SDN's enabled a shift from the classic three layered data center topology to a fat tree topology which allows for increased bandwidth at a minimal cost and high reliability, avoiding a single point of failure.

To illustrate how these technology enabled this new topological approach, we described how to setup an infrastructure that leverages ODL and Mininet to implement a simulation of a data center network of this sort. Furthermore, we described a global viewing flow manager application that uses REST to instruct the controller on how to define the network's forwarding rules.

Acknowledgments

The authors would like to thank Professor Rui Cruz for the ideas, support and encouragement. Without these, such an ambitious project would not have been possible.

References

[1] J. F. Kurose and K. W. Ross, "Computer Networking A Top-Down Approach Featuring the Internet," *Pearson Education India*, vol. 1, p. 712, 2005. [Online]. Available: http://www.amazon.com/dp/B0026579FQ

[2] D. User, "The advantages of using virtualization technology in the enterprise," Jun 2017. [Online]. Available: https://software.intel.com/en-us/articles/the_advantages_of-using-virtualization-technology-in-the-enterprise

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," *NSDI'05 Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, no. Vmm, pp. 273–286, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251203.1251223

[4] . J. . B. P. Judge, "Global forecast bright for the data center construction market," Jan 2015. [Online]. Available: http://www.datacenterdynamics.com/content-tracks/design-build/global-forecast-bright-for-the-data-center-construction_market/93014.fullarticle

[5] V. D. Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "A survey of network isolation solutions for multi-tenant data centers," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2787–2821, Fourthquarter 2016.

[6] "What is three-tier architecture? - definition from techopedia." [Online]. Available: https://www.techopedia.com/definition/24649/three-tier-architecture

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1355734.1355746

[8] J. M. S. Taylor, "The growth in east-west traffic," Jun 2011. [Online]. Available: https://www.networkworld.com/article/2177684/lan-wan/the-growth-in-east-west-traffic.html

[9] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, p. 63, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1402946.1402967

[10] "What is a single point of failure (spof)? - definition from techopedia." [Online]. Available: https://www.techopedia.com/definition/4351/single-point-of-failure-spof

[11] C. E. Leiserson, "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, 1985.

[12] "Home." [Online]. Available: https://www.opendaylight.org/

[13] "Home." [Online]. Available: https://www.onap.org/

[14] "Home - openstack open source cloud computing software." [Online]. Available: https://www.openstack.org/

[15] "Home." [Online]. Available: https://www.opnfv.org/

[16] "Http request methods." [Online]. Available: https://developer.mozilla.org/pt-PT/docs/Web/HTTP/Methods

[17] "By function." [Online]. Available: https://www.opendaylight.org/use-cases-and-users/by-function

[18] "Vagrant by hashicorp." [Online]. Available: https://www.vagrantup.com/

[19] M. Team, "Mininet." [Online]. Available: http://mininet.org/

[20] R. H. Ansible, "Ansible is simple it automation." [Online]. Available: https://www.ansible.com/

[21] D. Fan and P. Shi, "Improvement of Dijkstra's algorithm and its application in route planning," in *2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery*, 2010, pp. 1901–1904. [Online]. Available: http://ieeexplore.ieee.org/document/5569452/

Luís Sá Couto 22 years old. I am pursuing my Engineering studies at Instituto Superior Técnico (IST) since 2013. I was a teaching assistant of the distributed systems course in the same institution. My research interests are machine learning and neurocomputation and a PhD in the field is my next goal. However, my professional experience in the consulting firm McKinsey & Company awakened my interest in the corporate world. My main hobby is SCUBA diving. Being an assistant instructor as allowed me to work in several diving centers.

Carolina Fernandes 22 years old. I am a master's student in Information Systems and Computer Engineering at Instituto Superior Técnico, with specializations in Enterprise Systems and Information Systems. I am currently developing my thesis work on Future Operations Support Services for telecommunications operators and working as a consultant at PricewaterhouseCoopers. In my free time, I love swimming and playing the violin.

Rafael Belchior 21 years old. I am pursuing my Engineering studies at IST since 2014. I'm the current Coordinator of Grupo de Contacto com Empresas (GCE). My passion for entrepreneurship appeared when I participated in European Innovation Academy, a startup accelerator program. I joined the best of two worlds when I had the opportunity to be a startup team leader volunteer at Web Summit 2017.