UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

PEDRO HELENO ISOLANI

Interactive Monitoring, Visualization, and Configuration of OpenFlow-Based SDN

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Juergen Rochol

Porto Alegre
September 2015

CIP – CATALOGING-IN-PUBLICATION

Isolani, Pedro Heleno

Interactive Monitoring, Visualization, and Configuration of OpenFlow-Based SDN / Pedro Heleno Isolani. – Porto Alegre: PPGC da UFRGS, 2015.

95 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2015. Advisor: Juergen Rochol.

1. Software-defined networking. 2. Monitoring. 3. Visualization. 4. Configuration. I. Rochol, Juergen. II. Título.

ACKNOWLEDGMENTS

Firstly, I would like to thank God for giving me the strength and faith to reach my objectives with great determination and wisdom. I thank my family, specially my mother Arlete Isolani for all comprehension and support expended during the execution of this work. Also, a special thanks to my girlfriend, Beatriz Cristhine Day, who was always very comprehensive and supportive of me, no matter the circumstances, trying to help and advise me to keep doing a good work.

I would like to thank my advisor Prof. Juergen Rochol and my co-advisor Prof. Lisandro Zambenedetti Granville for the granted opportunity and all expended attention to my study. Also, I would like to thank you for all the reviews in my articles and dissertation text. Thank you for all the meetings that we had to address professional or personal matters. Also, thank you for the advice you gave me during hard times and in moments of high-impact decisions of my life.

I would like to thank Prof. Cristiano Bonato Both who provided me with the support to arrive at UFRGS and who always helped, guided and taught me during the courses and other commitments of the master's course.

Thanks to my friend Juliano Araujo Wickboldt for all the support, dedication and commitment on the writing and development of my articles amongst other works. Thank you for working with me on Saturdays, Sundays and nights if necessary, in order not to miss a deadline. Special thanks for helping me finish my experiment script while I took a nap for a few hours. Thank you also for advising me and helping me make several important decisions in my life.

Thanks to all my friends that somehow helped me to successfully complete this work. Special thanks to my friends from the lab: Lucas (Bondan), Juliano, Anderson, Luís, Eduardo (Germano), Giovani (Naldo), Vinicius (Funai), Marcelo (Marotta), Maicon, Ricardo (San), Wanderson (Goiano), Christian (Batman), Matheus (Ganso), Leonardo (Belo), Luíne (Browa), Augusto (Guto) and other friends from the computer networks group.

Thanks a lot to the friends that I had the opportunity to know and live with. For the zudos Ricardo Grunitzki (Rick), Sérgio Montazzolli Silva (Serjão), Matthew Bruce Vallis (Matt), Alex Fillies, Anderson Rocha Tavares (Andinho), Eric Gutierres and Tomas Anderberg, thank you all for the friendship and for all the good and happy moments that we had in Porto Alegre.

AGRADECIMENTOS

Eu gostaria de agradecer primeiramente a Deus por me dar força e fé para seguir em busca dos meus objetivos com muita determinação e sabedoria. Agradeço a minha família, em especial a minha mãe Arlete Isolani e por toda compreensão e apoio dispendido durante a execução deste trabalho. Um agradecimento especial também para minha namorada Beatriz Cristhine Day que sempre foi muito compreensiva, me apoiou, não importando as circunstâncias, procurando me ajudar e aconselhar a continuar e fazer um bom trabalho.

Eu gostaria de agradecer ao meu orientador Prof. Juergen Rochol e ao meu co-orientador Prof. Lisandro Zambenedetti Granville pela oportunidade concedida e toda atenção dispendida com o meu mestrado. Também gostaria de agradecer todas as revisões realizadas nos meus artigos e também no texto da dissertação. Agradeço também por todas as reuniões feitas para resolver assuntos profissionais ou pessoais. Além disso, obrigado pelos conselhos nos momentos difícies e decisões de grande impacto na minha vida.

Agradeço também ao Prof. Cristiano Bonato Both que me forneceu suporte ao chegar na UFRGS e sempre se dispôs a guiar-me e a ensinar-me durante as disciplinas e aos demais compromissos do mestrado.

Agradeço ao meu amigo Juliano Araujo Wickboldt por todo auxílio, dedicação e comprometimento na redação dos meus artigos, desenvolvimento da ferramenta, entre outros trabalhos. Agradeço por ter trabalhado comigo aos sábados, domingos e se necessário até noites a dentro para não perder um *deadline*. Agradeço especialmente por ter me ajudado a terminar de programar meus scripts de experimento enquanto eu dormia por algumas horas. Obrigado também por me aconselhar e me ajudar a tomar várias das decisões importantes na minha vida.

Obrigado a todos os meus amigos que de alguma forma me ajudaram a concluir com êxito esse trabalho. Agradeço especialmente aos colegas do meu laboratório: Lucas (Bondan), Juliano, Anderson, Luís, Eduardo (Germano), Giovani (Naldo), Vinicius (Funai), Marcelo (Marotta), Maicon, Ricardo (San), Wanderson (Goiano), Christian (Batman), Matheus (Ganso), Leonardo (Belo), Luíne (Browa), Augusto (Guto) e os demais colegas do grupo de redes.

Agadeço também aos amigos que tive oportunidade de conhecer e conviver. Aos zudos Ricardo Grunitzki (Rick), Sérgio Montazzolli Silva (Serjão), Matthew Bruce Vallis (Matt), Alex Fillies, Anderson Rocha Tavares (Andinho), Eric Gutierres e Tomas Anderberg agradeço por toda amizade e pelos muitos momentos de alegria em Porto Alegre.

“Measurements are not to provide numbers but insight”.

— INGRID BUCHER.

ABSTRACT

Software-Defined Networking (SDN) is an emerging paradigm that arguably facilitates network innovation and simplifies network management. SDN enables these features based on four fundamental principles: (*i*) network control and forwarding planes are clearly decoupled, (*ii*) forwarding decisions are flow-based instead of destination-based, (*iii*) the network forwarding logic is abstracted from a hardware to a programmable software layer, and (*iv*) an element, called controller, is introduced to coordinate network-wide forwarding decisions. Nowadays, much has been discussed about using SDN principles to improve network management – where SDN is taken as a management tool –, instead of discussing which are the new management challenges that this network paradigm introduces. In the context of SDN, management activities, such as monitoring, visualization, and configuration can be considerably different from traditional networks, thus deserving proper attention. For example, an SDN controller can be customized by network administrators according to their needs. Such customizations might pose an impact on resource consumption and traffic forwarding performance, which is difficult to assess because traditional network management solutions were not designed to cope with the context of SDN. As a consequence, an SDN-tailored management solution must be able to help the administrator to understand and control how the SDN controller behavior affects the network. Considering this context, we initially performed an analysis of control traffic in SDN aiming to better understand the impact of the communication between the controller and forwarding devices. Afterwards, we propose an interactive approach to SDN management through monitoring, visualization, and configuration that includes the administrator in the management loop, where SDN-specific metrics are monitored, processed, and displayed in interactive visualizations. Thus, the administrator is able to make decisions and configure/reconfigure SDN-related parameters according to his/her needs. To show the feasibility of our approach a prototype has been developed, called *SDN Interactive Manager*. The results obtained with this prototype show that our approach can help the administrator to better understand the impact of configuring SDN-related parameters on the overall network performance.

Keywords: Software-defined networking. Monitoring. Visualization. Configuration.

Visualização, Monitoração e Configuração Interativa de Redes Definidas por Software baseadas em OpenFlow

RESUMO

Redes Definidas por Software (*Software-Defined Networking* – SDN) é um paradigma emergente que sem dúvida facilita a inovação e simplifica o gerenciamento da rede. SDN provém esses recursos baseado em quatro princípios fundamentais: (*i*) os planos de controle e encaminhamento da rede são claramente desacoplados, (*ii*) as decisões de encaminhamento são baseadas em fluxo ao invés de baseadas em destino, (*iii*) a lógica de encaminhamento é abstraída do hardware para uma camada de software e (*iv*) um elemento, chamado controlador, é introduzido para coordenar as decisões de encaminhamento. Atualmente muito se tem discutido acerca do uso de SDN em benefício do gerenciamento de redes – onde SDN é considerado uma ferramenta de gerenciamento –, ao invés de se discutir quais são os novos desafios de gerenciamento que esse paradigma introduz. No contexto de SDN, atividades de gerenciamento como monitoramento, visualização e configuração podem ser consideravelmente diferentes das mesmas realizadas em redes tradicionais, merecendo a devida atenção. Por exemplo, um controlador SDN pode ser customizado por administradores de rede de acordo com suas necessidades. Essas customizações podem impactar em consumo de recursos e desempenho no encaminhamento de tráfego. Tal impacto é difícil de se avaliar porque solucões de gerenciamento de redes tradicionais nao foram projetadas para lidar com o contexto de SDN. Como consequencia, uma solução de gerenciamento de SDN deve ser capaz de ajudar o administrador a entender e controlar como o comportamento do controlador SDN afeta a rede. Considerando esse contexto, nós inicialmente desenvolvemos uma análise do tráfego de controle em SDN visando melhor entender o impacto da comunicação entre controlador e dispositivos de encaminhamento. Em seguida, nós propomos uma abordagem interativa para gerenciamento de SDN através do monitoramento, visualização e configuração da rede incluindo o administrador em um ciclo de atividades de gerenciamento, onde metricas específicas de SDN são monitoradas, processadas e mostradas em visualizações interativas. Assim, o administrador da rede é capaz de configurar/reconfigurar parâmetros de SDN de acordo com seu/sua necessidade. Para demonstrar a viabilidade da nossa abordagem, nós desenvolvemos um protótipo chamado *SDN Interactive Manager*. Os resultados obtidos através do protótipo apresentaram que a nossa abordagem é capaz de auxiliar o administrador a melhor entender o impacto da configuração de parâmetros relativos a SDN no desempenho da rede como um todo.

Palavras-chave: Redes Definidas por Software. Monitoramento. Visualização. Configuração.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

API	*Application Programming Interface*
ARP	*Address Resolution Protocol*
CMI	*Common Information Model*
CR	*Cognitive Radio*
CRUD	*Create, Read, Update and Delete*
CSS	*Cascading Style Sheets*
DMTF	*Distributed Management Task Force*
DPID	*Datapath ID*
DIFANE	*DIstributed Flow Architecture for Networks Enterprises*
FR	*Functional Requirements*
GUI	*Graphical User Interface*
HHH	*Hierarchical Heavy Hitters*
HTML	*HyperText Markup Language*
IaaS	*Infrastructure as a service*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
MAC	*Media Access Control*
MI	*Management Interface*
MVT	*Model-View-Template*
NFV	*Network Functions Virtualization*
OMNI	*OpenFlow Management Infrastructure*
REST	*Representation State Transfer*
TCP	*Transmission Control Protocol*
ToS	*Type of Service*
UDP	*User Datagram Protocol*
REST	*Representational State Transfer*

SDN	*Software-Defined Networking*
SNMP	*Simple Network Management Protocol*
SPOF	*Single Point of Failure*
SSL	*Secure Socket Layer*
URL	*Uniform Resource Locator*
VLAN	*Virtual Local Area Network*

CONTENTS

1 INTRODUCTION

Software-Defined Networking (SDN) is an emerging paradigm that enables network innovation based on four fundamental principles: (*i*) network control and forwarding planes are clearly decoupled, (*ii*) forwarding decisions are flow-based instead of destination-based, (*iii*) the network forwarding logic is abstracted from a hardware to a programmable software layer, and (*iv*) an element, called controller, is introduced to coordinate network-wide forwarding decisions (ROTHENBERG et al., 2014). SDN is grabbing the attention of both academia and industry since it allows the easy creation of new abstractions in networking, simplifying management, and facilitating network innovation (FOUNDATION, 2012). In this sense, SDN reduces or even eliminates some traditional network management problems, such as enabling network configuration in a high-level language or providing support for enhanced network diagnosis and troubleshooting (KIM; FEAMSTER, 2013).

Nowadays, much has been discussed about using SDN principles to improve the network management – where SDN is taken as a management tool –, instead of discussing which are the new management challenges that this network paradigm introduces (KIM; FEAMSTER, 2013). For example, SDN principles can improve network management tasks due to the fact of it provides a logically centralized view, but it can also introduce a *Single Point of Failure* (SPOF) or, in case of more than one controller, can be extremely complicated to guarantee isolation of network applications and maintain different and conflicting service requirements. Although SDN can solve some traditional management problems, it also creates new ones that have not been addressed yet (WICKBOLDT et al., 2015). Management aspects such as network security, resilience, performance, and failover can be significantly different from traditional networks and need to be managed properly.

Monitoring, visualization, and configuration are fundamental management activities to understand and control the network behavior (SALVADOR; GRANVILLE, 2008) (BARBOSA; GRANVILLE, 2010) (GUIMARAES et al., 2014) (BONDAN et al., 2014) (MACHADO et al., 2014). In the context of SDN, these activities can also be considerably different than in traditional networks, thus deserving proper attention (WICKBOLDT et al., 2015). For example, the behavior of an SDN controller can be customized by network administrators according to their needs. These controller customizations might impact in terms of resource consumption and traffic forwarding performance. Such impact is difficult to assess because traditional network management solutions were not designed to cope with the context of SDN. As a consequence, an SDN-tailored management solution must be able to help the administrator to understand and control how the SDN controller behavior affects the network.

The state-of-the-art in SDN has addressed monitoring using the OpenFlow protocol (MCKEOWN et al., 2008) – currently the most relevant SDN implementation – for different purposes (JOSE; YU; REXFORD, 2011) (TOOTOONCHIAN; GHOBADI; GANJALI, 2010) (ZHANG, 2013) (YU et al., 2013) (CHOWDHURY et al., 2014). Most of these investigations are focused

on proposing anomaly detection systems or mechanisms to establish a balance between control channel overhead and accuracy of the collected information. These investigations tend to employ monitoring information to automatically adapt the network to specific conditions (*e.g.*, switches lowest resource usage as possible). However, to the best of our knowledge, no solution is available to integrate monitoring information with interactive visualization and configuration tools for SDN. We argue that with such a solution the administrator could better understand and interact with the network, significantly improving everyday tasks of SDN management.

An OpenFlow-based architecture introduces a simple way to develop and maintain communication in SDN because of its centralized logic of controlling forwarding devices. However, this simplicity may allegedly impose a high cost on the network controller and create bottlenecks on the control channel (YEGANEH; GANJALI, 2014). Recent solutions such as Devoflow (CURTIS et al., 2011) and DIFANE (YU et al., 2010) attempted to alleviate these bottlenecks by distributing the control logic of OpenFlow. Nevertheless, there is no detailed study about in which situations such bottlenecks appear and whether they can or cannot be mitigated or even avoided by simple configuration, *i.e.*, without the need to develop a specific distributed controller. We argue that is required a detailed study about the overhead imposed by the communication protocol used to understand and then prevent bottlenecks on the control channel.

In this dissertation we initially (*i*) perform an analysis of the control traffic in SDN to understand the overall impact of OpenFlow control messages on the control channel. Our analysis focused on explaining the overhead of these messages in terms of resource consumption and network performance resulting from the communication between the controller and forwarding devices. Based on this analysis, we (*ii*) propose an interactive approach to SDN management through monitoring, visualization, and configuration management activities. Our goal is to include the administrator in the management loop, where SDN-specific metrics are monitored, processed, and displayed in interactive visualizations. Thus, the administrator is able to make decisions by configuring/reconfiguring SDN-related parameters according to his/her needs. Our main contribution is to integrate these three management activities allowing the administrator to easily understand and control the network.

Regarding our control channel analysis, we quantify the load imposed by OpenFlow 1.0 control messages – this OpenFlow version is currently the most adopted SDN implementation – and then we focused on messages that occur more frequently in our particular scenario of study. Afterwards, we also point out configuration parameters that can influence on this load and may be set to reduce the chance of bottlenecks in the control channel, especially on the controller. Next, to emphasize the feasibility of our proposed approach, we developed a prototype called *SDN Interactive Manager* as an SDN-devoted management system that performs monitoring, provides interactive visualizations, and enables network parameters configurations. Our prototype was developed to allow the administrator to interact with visualizations through a *Graphical User Interface* (GUI) and enables to configure/reconfigure SDN-related parameters

according to his/her needs. Our objectives in building the prototype are the following:

- Perform SDN monitoring with configurable polling intervals specifically focusing on inspecting resource usage and control channel load metrics.
- Present monitoring information obtained in interactive visualizations that emphasize these metrics.
- Support the administrator interaction by configuring and reconfiguring SDN-related parameters.

To evaluate our approach, we used the Floodlight[1] controller and a campus network scenario emulated over Mininet (LANTZ; HELLER; MCKEOWN, 2010). Our control channel analysis results present the impact of the SDN-related parameter *rule idle timeout* configuration and the polling interval can affect both resource usage and control channel load (both directions). Furthermore, our prototype *SDN Interactive Manager* shows that, by interacting with his visualizations on the provided GUI, the administrator is able to adjust SDN-related parameters on the network, improving the controller configuration, and thus reducing the resource consumption (number of idle flows installed on switches) and control channel usage (amount of messages transmitted on the control channel) when it is possible.

The remainder of this dissertation is organized as follows. In Chapter 2, we provide a brief description of the central background concepts associated with our approach and related work. In Chapter 3, we present an analysis of control traffic in OpenFlow-based SDN. In Chapter 4, we present our approach in detail and our developed prototype. In Chapter 5, we present the evaluation of our proposed approach with the developed prototype. Finally, in Chapter 6, we conclude this dissertation presenting final remarks and the perspective for future work.

[1] www.projectfloodlight.org/floodlight/

2 BACKGROUND AND RELATED WORK

This chapter presents a brief overview of the main concepts required by our control channel analysis and proposed approach. Section 2.1 depicts the SDN concepts and architecture. Subsection 2.2 presents an overview of OpenFlow specification and versions. Subsection 2.3 describes well-known controller behaviors in the literature that are used on both control channel analysis and proposed approach evaluation. Next, Section 2.4 depicts management approaches present in the literature that address SDN monitoring, visualization, and configuration activities.

2.1 SDN Concepts and Architecture

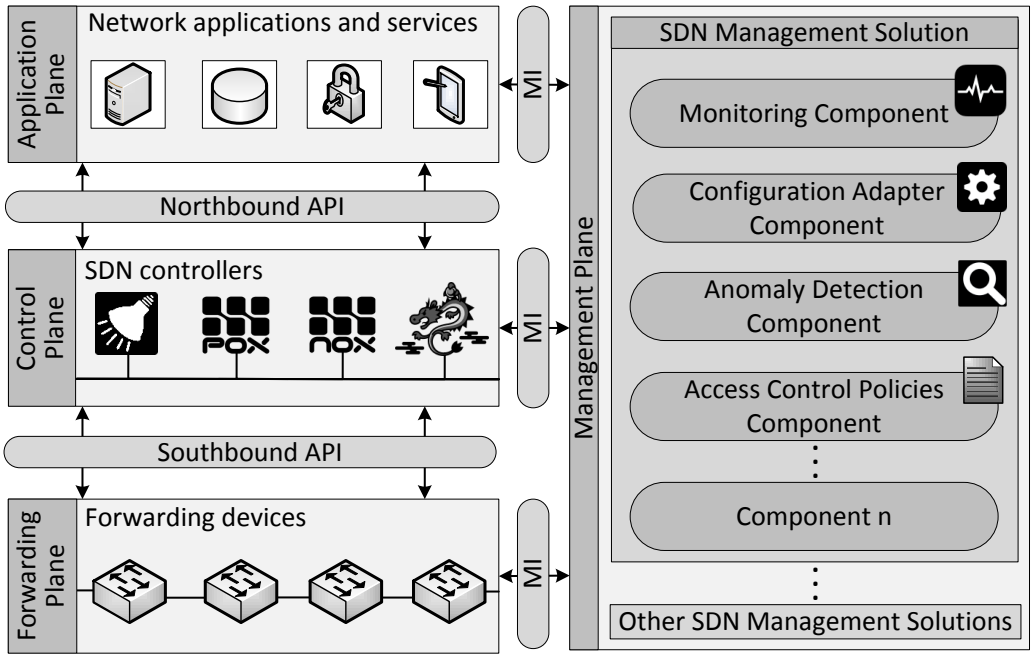
Software-Defined Networking (SDN) is an emerging paradigm that originally refers to a network architecture where forwarding decisions are clearly decoupled from the network control logic. In other words, the network control logic is removed from forwarding devices – that become simple packet forwarding devices – to a centralized element, called controller. Basically, the SDN architecture was designed to enable network innovation based on four fundamental principles: (*i*) network *control* and *forwarding* planes are clearly decoupled; (*ii*) forwarding decisions are flow-based instead of destination-based, where a flow is defined as set of packet header fields acting as a match to filter and perform actions with all matched packets; (*iii*) the network forwarding logic is abstracted from hardware to a programmable software layer; and (*iv*) an element, called SDN controller or *Network Operation System* (NOS), is introduced to coordinate network-wide forwarding decisions (KREUTZ et al., 2015).

In the context of SDN, much has been discussed about SDN and network management by itself, mostly from the perspective of where SDN is taken as a management tool (KIM; FEAMSTER, 2013). Several approaches use SDN to deal with management activities because it simplifies or even solves some traditional management activities. For example, because the fact of forwarding devices need to be registered or discovered by the network, in SDN, all forwarding devices establish a communication path with the controller (*i.e.*, between *forwarding* and *control* planes) and the network discovery management activity – a traditional management activity – is intrinsically solved. However, SDN also creates new management challenges that are not yet discovered or widely addressed in the literature (*e.g.*, controller placement and resilience).

Overall, SDN introduces an architecture with four planes: *management*, *application*, *control*, and *forwarding* (FOUNDATION, 2014). All these planes communicate with each other through interfaces. For example, the *management* plane uses a set of *Management Interfaces* (MI) to exchange information and to control elements in other planes. In addition, an interface called *northbound* API establishes bidirectional communication between *application* and *control* planes, while the *southbound* API does the same for *control* and *forwarding* planes. Ideally, all these interfaces should be standardized to allow easy replacement of devices and technologies. In practice, the OpenFlow protocol is the current *de facto* standard *southbound* API. All

other interfaces are undergoing discussion and development. Figure 2.1 depicts SDN planes with *Southbound* and *Northbound* APIs, MIs and management solutions.

Figure 2.1 – SDN Architecture.



Source: FOUNDATION (2014).

Conceptually, each of the four planes has a set of specific functions that need to be fulfilled, as following explained:

- ***Management Plane:*** is situated on the side of each of other planes and is able to coordinate them individually through MIs. Contains one or more SDN management solutions responsible for managing elements in other SDN planes (*e.g.*, monitoring device status, configuring and allocating resources, and enforcing access control policies).
- ***Application Plane:*** is situated on top-left of the architecture. Contains one or more applications that can serve several different purposes (*e.g.*, firewall, circuit establisher, and load balancer). Each of these applications are granted with access to a set of resources by one or more SDN controllers.
- ***Control Plane:*** is situated between the *application* and *forwarding* planes. Contains one or more controllers that coordinate network devices (*e.g.*, *Floodlight*, NOX, POX, and Ryu). At least an SDN controller needs to execute the requests coming from the application plane. Commonly, these controllers also include internal logic to handle network events and make traffic forwarding decisions.
- ***Forwarding Plane:*** is situated on the bottom and comprises a set of forwarding elements with transmission capacity and traffic processing resources. The notion of keeping forwarding elements simple and making decisions at higher software layers is central in SDN.

Due to the introduction of these four principles and the flexible architecture, SDN is grabbing the attention of both academia and standardization bodies (*e.g.*, ONF, IETF) and industry (*e.g.*, Google, Cisco, NEC, Juniper), since it allows the easily creation of new abstractions in networking, simplifying management, and facilitating network innovation (FOUNDATION, 2014). In this sense, SDN reduces or even eliminates some traditional network management problems, such as enabling network configuration in a high-level language or providing support for enhanced network diagnosis and troubleshooting (KIM; FEAMSTER, 2013).

2.2 OpenFlow Brief Background

OpenFlow is a trademark of Stanford University that defines an open protocol with a standardized interface to manage networks through a logically centralized controller (MCKEOWN et al., 2008). In OpenFlow, traffic forwarding decisions are handled flow-based, instead of destination-based (SDN principle number (*ii*)). To establish flow paths over a network, an OpenFlow controller adds and removes forwarding rules in network switches to coordinate the data traffic. An OpenFlow forwarding rule is composed of a *match* of packet header fields, *e.g.*, source and destination IP addresses at least, and a set of *actions* to be performed, *e.g.*, forward or drop a packet. The OpenFlow specification also defines that an OpenFlow switch is composed of (*i*) a flow table to store forwarding rules, (*ii*) a secure communication channel to establish a connection from forwarding devices with the controller, and (*iii*) the OpenFlow protocol itself (PFAFF et al., 2009).

All detailed requirements that an OpenFlow switch should follow are defined by the documentation of OpenFlow Switch Specification (PFAFF et al., 2009). Moreover, there are two variations of an OpenFlow switch can assume: the *dedicated OpenFlow* and the *OpenFlow-enabled* switch. The *dedicated OpenFlow* switch is a dumb forwarding element that is controlled by one or more external control elements. On the other hand, some commercial forwarding devices have beyond their own proprietary software – to control the forwarding logic – also a flow table, a secure channel, and the OpenFlow protocol, becoming an *OpenFlow-enabled* switch (another variation of OpenFlow switch).

The OpenFlow version 1.0 is currently the most adopted by network hardware vendors and controllers developers (KREUTZ et al., 2015). The specification defines three message types: (*i*) *Controller-to-switch*, (*ii*) *Asynchronous*, and (*iii*) *Symmetric* (PFAFF et al., 2009). *Controller-to-switch* messages are initiated by the controller and are used to manage or inspect the state of OpenFlow switches. *Asynchronous* messages are initiated by the switch and are used to send notification of network events and changes to the controller. *Symmetric* messages can be initiated by either the controller or switches and are mainly used for network bootstrap, latency measurement, and to keep alive the control channel. Each of these message types are composed of multiple sub-types that are used for specific network coordination actions. Table 2.1 presents all OpenFlow 1.0 message types with their respective sub-types and descriptions.

Table 2.1 – Control messages defined by the OpenFlow 1.0 specification.

Message type	**Sub-type**	**Description**
Controller-to-switch	*Features*	Obtain features and capabilities about the switches.
	Configuration	Set query configuration parameters in switches.
	Modify-State	Manage the state of the switches.
	Read-State	Retrieve statistics about switch tables, ports, flows, and queues.
	Send-Packet	Send packets to a specific switch port.
	Barrier	Ensure message dependencies and receive notifications.
Asynchronous	*Packet-In*	When a packet not match a flow entry or a matched flow does entry action is "send to the controller".
	Flow-Removed	When a flow entry expires in the switch flow table.
	Port-Status	Send port configuration state changes.
	Error	Notify problems to the controller.
Symmetric	*Hello*	Exchanged between switch and controller upon connection startup.
	Echo	Sent by both controller and switch to establish connectivity.
	Vendor	Functionality to store a stagging area for other OpenFlow revisions.

Source: PFAFF et al. (2009).

Some of the main message sub-types defined by the OpenFlow specification are: *Modify-State* and *Flow-Removed* to install and remove rules on forwarding devices; *Send-Packet* to send a packet to a specific switch port; and *Packet-In*, to notify the controller when a switch receives a packet that does not match with a forwarding rule entry in the switch flow table (Table 2.1). Another message sub-type that is used by monitoring solutions to retrieve statistics from switches is *Read-State*. There are two variations of *Read-State* message sub-type: *request* and *reply*. The *request* messages are sent from the controller to switches to request for statistics counters and the *reply* messages are sent from the switches to the controller delivering the requested statistics counters. As a result, the OpenFlow specification enables both the configuration of forwarding devices and monitoring traffic statistics.

To proper use OpenFlow in an SDN-based network it is fundamental to understand the operation of these messages, as well as the controller behavior implementation. The OpenFlow specification does not state how messages should be used to proper manage an OpenFlow-based network without considerably affect network performance. It is on the account of the administrator to understand the OpenFlow specification and the controller's behavior, and then decide how OpenFlow can be employed to accomplish everyday management tasks.

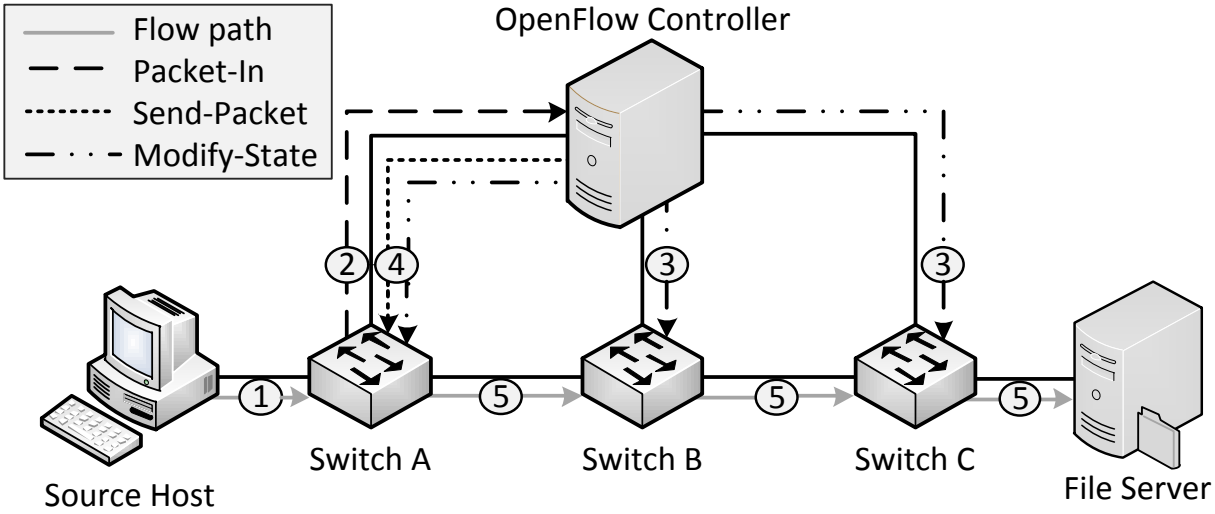
2.3 Controller Behavior

OpenFlow messages are used to coordinate forwarding devices in different ways, depending on the controller behavior implementation. Three well known controller behaviors that affect the installation of forwarding rules and, consequently, the network operation are: *Hub*, *Switch*, and *Forwarding* behavior (KLEIN; JARSCHEL, 2013). The *Hub* behavior implementation dictates that every packet that does not match a flow entry is flooded to all switch interfaces (obviously inefficient). The *Switch* behavior implementation dictates that when the controller

receives a *Packet-In* the controller installs a rule and sends the packet to the switch that generates this Packet-In. Last, the *Forwarding* behavior implementation needs only one *Packet-In* to install rules on the entire flow path. This behavior is the most sophisticated implementation of those mentioned, presenting a lower control overhead imposed by OpenFlow messages.

An example of how the *Forwarding* behavior coordinates rule installation between *Source Host* and the destination *File Server* is depicted in Figure 2.2. First, when *Source Host* sends a data packet to *Switch A* (1), this switch checks whether there is a forwarding rule entry matching this packet's header fields in the flow table. If the header fields match with an entry, the corresponding actions should be applied to this data packet, *e.g.,* forward or drop. However, if no match exists, by default, *Switch A* generates a *Packet-In* message to the controller (2). Upon receiving the *Packet-In*, the controller calculates the flow path and sends a set of *Modify-State* messages to install forwarding rules in all switches in the path between source and destination, except to *Switch A* (3). Afterwards, the controller sends *Send-Packet* and *Modify-State* messages to *Switch A* that originated the *Packet-In* (4). Finally, once all forwarding rules are installed on switches, the data packet is sent through the flow path (5) to the destination.

Figure 2.2 – OpenFlow controller *Forwarding* behavior example.



Source: by author (2015).

A few extra points need to be clarified regarding the *Forwarding* behavior. First, the example in Figure 2.2 assumes that the location of the destination *File Server* is known by the controller in advance. In practice, before every host originates traffic, the controller will generally need to discover these locations by flooding the network. Also, the controller does not guarantee that rule installation messages, *i.e.*, *Modify-State*, arrive in switches properly ordered, since these messages can be sent in parallel. Thus, if *Modify-State* messages do not arrive in the switch before the data packet, extra *Packet-In* messages will be generated and sent to the controller. In addition, the controller behavior does not orchestrate how *Read-State* messages are used by either the controller or forwarding devices. As such, a monitoring solution is still required alongside the OpenFlow controller to fill this gap and help in managing OpenFlow-based

networks.

In addition to the forwarding rule installation, the behavior implementation also dictates which OpenFlow header fields are installed as wildcard fields (*i.e.*, field is a match, regardless of value). The *Forwarding* behavior installs a forwarding rule specifying only the ingress port, and the Ethernet source and destination MAC addresses. The rest of fields that compose the entire match are set as wildcards (represented by '*'). These header fields are: the Ethernet type, *Virtual Local Area Network* (VLAN) ID, VLAN Priority, *Internet Protocol* (IP) source and destination addresses, IP protocol and Type of Service (ToS), and *Transmission Control Protocol* (TCP) or *User Datagram Protocol* (UDP) source and destination ports. All header fields and their granularity based on *Forwarding* behavior are presented on Table 2.2.

Table 2.2 – OpenFlow 1.0 header fields installation based on *Forwarding* behavior.

Ingress port	**Ethernet src**	**Ethernet dst**	**Ethernet type**	**VLAN ID**	**VLAN priority**	**IP src**	**IP dst**	**IP protocol**	**IP ToS bits**	**TCP/UDP src port**	**TCP/UDP dst port**
Specific value	Specific value	Specific value	*	*	*	*	*	*	*	*	*

Source: by author (2015).

It is important to emphasize that the forwarding rule installation granularity also affects the resource usage and control channel load metrics. To store a forwarding rule with wildcard fields, the switch generates matches on the flow table according to his internal functionality. Given that the switches having expensive and limited *Ternary Content-Addressable Memories* (TCAM), it is necessary to carefully manage the amount of rules installed to avoid network devices running out of resources. Moreover, *Read-State-reply* messages are also affected because the length of these monitoring messages can grow with the amount of specific rules installed.

2.4 Related Work

This Section describes recent studies that deal with overheads at the control channel such as bottlenecks in OpenFlow-based SDN and present management approaches that address aspects of SDN monitoring, visualization, and configuration to provide relevant information to the administrator. In Subsection 2.4.1 several approaches that deal with bottlenecks at the control channel are presented. In Subsection 2.4.2, we describe monitoring, visualization, and configuration approaches in other network contexts. In Subsection 2.4.3, SDN management approaches that deal with monitoring, visualization, and configuration are described in detail. In Subsection 2.4.4, these approaches are discussed and compared considering the trade-off between control channel load and resource consumption.

2.4.1 Control Channel Bottlenecks

Because the fact of SDN centralize the logic of forwarding devices on one or more controllers, some approaches argue that control messages can create bottlenecks between device-controller. Recent solutions, such as *Devoflow* presented by CURTIS et al. (2011), attempt to alleviate these bottlenecks by distributing the control logic of OpenFlow. *Devoflow* avoids these bottlenecks through mechanisms to control forwarding devices and statistics collection. To better control the network without invoking the controller decision, *Devoflow* uses a rule cloning mechanism to deal with micro-flows with fewer rules. Moreover, the switch also has local actions that do not require controller decisions as well. For statistics collection, the solution has triggers, sampling, and approximate counters that also help to reduce the imposed overhead of monitoring messages at the control channel.

Another solution that attempts to avoid bottlenecks in the control channel is the *DIstributed Flow Architecture for Networks Enterprises* (DIFANE), presented by YU et al. (2010). The authors argue that these bottlenecks occur because the flow-based networking rely too heavily on centralized controller that installs rules reactively, based on the first packet of each flow. To solve this problem, they proposed a solution that decentralizes the control logic by keeping it on SDN forwarding plane. DIFANE relegates the controller to the simple task of partitioning rules over the switches, called authority switches, that are responsible for a network slice. In this context, a network slice is a part of the network that is controlled by an authority switch. These authority switches receive packets (e.g., Packet-In messages), – that were previously sent to the controller – and reply with the proper rule installation message (*e.g.*, *Modify-State* and *Send-Packet* messages).

2.4.2 Monitoring, Visualization, and Configuration in Different Contexts

Some recent studies have addressed monitoring, visualization, and configuration management activities in other network contexts. SALVADOR; GRANVILLE (2008) proposed a study for *Simple Network Management Protocol* (SNMP) traffic using visualization techniques. As a proof of concept, the authors also prototyped a software called *Management Traffic Analyzer* that has been used to visualize SNMP traces. BARBOSA; GRANVILLE (2010) also analyzed SNMP traffic through information visualization. However, the authors proposed interactive visualization techniques and an insight-based approach that aim to better understand the protocol.

In the same context of traffic visualization, GUIMARAES et al. (2014) proposed a solution named *mtAnalyzerV2* that improves the collaboration among network operators in SNMP traffic measurements and analysis. The authors explore visualization techniques and a set of user-friendly capabilities to archive collaboration among network operators. Moreover, from a different perspective and more similar to our approach, BONDAN et al. (2014) addressed monitoring, visualization, and configuration management activities by using SNMP for monitoring

and thus, enabling the network administrator to continuously learn how to better configure the spectrum sensing function in *Cognitive Radio* (CR) context.

As a conclusion, monitoring, visualization, and configuration are fundamental management activities to understand and control the network behavior. All proposals presented intend to provide a better understanding of the network through these activities. Therefore, in the next Subsection, we present some recent SDN solutions that deal with these management activities that are more related to our proposed approach.

2.4.3 SDN Monitoring, Visualization, and Configuration

Currently, many investigations consider using monitoring information obtained with the OpenFlow protocol for different purposes. Zhang (2013) proposed the use of monitoring messages to develop algorithms for anomaly detection systems based on methods for statistics information collection. They propose OpenWatch that performs adaptive zooming in aggregation of flows to be measured, *i.e.*, an algorithm that dynamically change the granularity of measurement. This proposed solution mainly aims at establishing a balance between the monitoring overhead – imposed by monitoring messages – and the anomaly detection system accuracy.

Jose *et al*. (2011) employ these aforementioned monitoring messages to propose mechanisms for online measurement of large traffic aggregates, which can be used for both anomaly detection and traffic engineering. The authors explore a measurement framework that matches packets according to a small collection of rules and updates traffic counters based on the highest priority match. Moreover, they proposed a measurement model based on OpenFlow switches that archived high accuracy and a low traffic overhead to monitoring rules in *Hierarchical Heavy Hitters* (HHH) problem.

Yu *et al*. (2013) proposed FlowSense, which aims to keep the lowest control channel overhead and the highest information accuracy as possible. FlowSense, however, uses only OpenFlow *Asynchronous* messages to collect statistics information. The statistics collection are made only based on *Packet-In*/*Flow-Removed* to receive statistics about flows. Although FlowSense introduces zero cost in the monitoring process, the statistics information may be inaccurate in the case where OpenFlow messages received by the controller are too sparse. To separate the SDN monitoring, YU; JOSE; MIAO (2013) proposed OpenSketch that provides a clear separation between monitoring and control planes. The monitoring plane extracts the statistics while the control plane has the monitoring libraries that self configure to supply different monitoring tasks programmed by the controller.

Chowdhury *et al*. addressed SDN monitoring from a different perspective. The authors proposed a framework, called Payless (CHOWDHURY et al., 2014), able to deal with monitoring considering the polling frequency and data granularity. This framework is able to adjust the polling frequency to balance the control channel overhead, imposed by monitoring messages, and accuracy of monitored information. Payless relies on OpenTM (TOOTOONCHIAN;

GHOBADI; GANJALI, 2010) to select only important switches to be monitored, also aiming to reduce the overhead imposed in the control channel. The authors employed a modularized architecture with an algorithm to set the polling frequency for monitoring a set of selected switches. Therefore, it is possible to extract just the relevant information while keeping the control channel overhead low.

Another OpenFlow-based monitoring framework is PaFloMon, proposed by ARGYROPOULOS et al. (2012). PaFloMon aims to perform a user-aware passive monitoring using a slice-based perspective. In the context of SDN, a network slice is a logical topology build on top of the physical topology that allows many controllers to share the same physical infrastructure. Moreover, this framework also provides per-slice monitoring plane isolation and separates the monitored data into different repositories that represent each slice. In addition, this framework allows to aggregate these monitored information along different repositories.

The aforementioned proposals are focused on the use of monitoring information to automate tasks, such as reducing control traffic overhead and protecting the network. None of previous investigations aim to employ monitoring information to neither help the administrator understanding the network behavior nor interact with it. However, there are some solutions such as the *Floodlight*[1] controller, the OpenDaylight[2] platform, and *OpenFlow MaNangement Infraesctructure* (OMNI) (MATTOS et al., 2011) that deal with SDN physical and logical topology visualization and configuration in addition to manage the network. Moreover, these solutions also allow the administrator to view the network topology and enable to set some network configurations, but they do not address control channel statistics to enable the network administrator to consider this aspect in their configurations.

The *Floodlight* controller is an Open Source software for building SDN with 1.0 and 1.1 versions of the OpenFlow protocol. It is a Java-based controller, modular, that has a *Graphical User Interface* (GUI) to interact with network visualizations and flow configurations. Through *Floodlight Representational State Transfer* (REST) *Application Programming Interface* (API) it is possible to request network information allowing to view the physical topology, nodes and edges that represent switches or hosts, and links of the network. In addition to network topology visualization, this controller also provides switch vendor information and data traffic counters such as per switch number of packets, bytes transmitted and overall flows.

OpenDaylight is an open platform for network programmability to enable SDN and Network Functions Virtualization (NFV). This platform – as the *Floodlight* controller – has a Web interface that allow the administrator to view the physical topology and enables to set some network configurations on the controller. Using the OpenDaylight Web interface it is possible to set static flows into switches flow tables, change switches information and operation mode (*e.g.*, reactive or proactive forwarding), view network flows information in details, in addition to ports, interfaces and data traffic counters also. *Proactive* forwarding could be used to install

[1] http://www.projectfloodlight.org/floodlight/
[2] http://www.opendaylight.org/

flow tables entries for all possible matches eliminating any latency induced by consulting a controller. On the other hand, the *reactive* forwarding is when a switch forwards every packet that does not match any entry to the controller.

OMNI provides a framework in addition to visualize the network physical topology, nodes and links information. The OMNI framework also enables the administrator to view the logical topology. Moreover, it provides a Web interface that allows the administrator to execute flow migrations at the controller. However, *Floodlight*, *OpenDaylight* and OMNI solutions do not address metrics such as control traffic statistics and forwarding devices resource consumption rates as well. These metrics can help to identify bottlenecks at the control channel and may help the administrator to better understand the controller behavior implementation.

2.4.4 Discussion and Comparison of SDN approaches

As cited before, the aforementioned SDN proposals are focused on automated tasks and do not include the administrator decision on the network management. These presented solutions address SDN monitoring for other purposes (*e.g.*, an anomaly detection system or traffic engineering) and not to provide visualizations or configurations so the network administrator can improve the network configuration or solve network problems. Moreover, these presented solutions do not include statistics information about the control channel given that is an important rate to be observed in a management solution also. In addition, many authors argue that bottlenecks at the controller control channel occur because SDN approaches use a single controller, but they do not mention in what proportion or when they can occur.

It is important to emphasize that *OpenDaylight*, OMNI and *Floodlight* presented solutions that address SDN visualization. All of them present the physical topology view and only OMNI present the logical topology view. However, none of them has features to count the control traffic messages and help the administrator to improve SDN-related parameters configuration considering control channel load and resource consumption. SDN configuration also has been addressed by some of these approaches, but to the best of our knowledge, there are no solutions that leverage OpenFlow monitoring messages to create network visualizations and address the interactive configuration of SDN-related parameters.

Just the Floodlight controller supports monitoring, visualization and configuration management activities. But, the default implementation of *Floodlight* controller does not perform all of these activities by itself. Applications that perform this SDN monitoring, visualization and configuration activities are needed to integrate and show to the network administrator. *Floodlight* only provides interfaces to other solutions request for informations through its REST API and GUI. Table 2.3 depicts each of presented solutions that deal with SDN monitoring, visualization, and configuration aspects.

Table 2.3 – Comparison of SDN proposals for monitoring, visualization, and configuration.

Name	Monitoring	Visualization	Configuration
Devoflow (CURTIS et al., 2011)	Performs sampling and statistics triggering based on thresholds for flow counters and by using approximate counters.	Not addressed.	Not addressed.
OpenWatch (ZHANG, 2013)	An algorithm that performs statistics collection with an adaptive zooming in aggregation of flows for an anomaly detection system.	Not addressed.	Not addressed.
Measurement model (JOSE; YU; REXFORD, 2011)	Measurement model that deals with SDN monitoring on online traffic aggregations with high accuracy and low overhead.	Not addressed.	Not addressed.
FlowSense (YU et al., 2013)	An algorithm that uses OpenFlow *Asynchronous* messages to retrieve statistics about flows with zero measurement cost.	Not addressed.	Not addressed.
Payless (CHOWDHURY et al., 2014)	Performs monitoring using an algorithm that dictates the polling frequency and store this data in a database.	Not addressed.	Uses OpenTM to select the monitored switches, an algorithm to automatically configure the polling frequency, and has an interface to choose options to store the monitored data.
OpenTM (TOOTOONCHIAN; GHOBADI; GANJALI, 2010)	Proposes a low-overhead accurate traffic matrix estimator to measure the switches and keep track of flow statistics.	Not addressed.	Not addressed.
PaFloMon (ARGYROPOULOS et al., 2012)	Performs user-aware passive monitoring and per-slice monitoring plane isolation.	Not addressed.	Not addressed.
Floodlight Controller	Supports monitoring of its memory usage, performance and data traffic counters.	Through a GUI it allows to view the physical topology with nodes and links, flows and data traffic counters.	Supports to load different applications and modules to configure the network.
OpenDaylight Platform	Not addressed.	Allows to view the physical topology with nodes and links, flows and data traffic counters.	Not addressed.
OMNI (MATTOS et al., 2011)	Not addressed.	Allows to view the physical and logical topology with nodes and links, flows and data traffic counters.	Perform static flow pusher and flow migration.

Source: by author (2015).

3 CONTROL CHANNEL ANALYSIS

This chapter presents an analysis of the control channel load imposed by OpenFlow 1.0 control messages. Section 3.1 presents the motivation and the methodology in addition to present the user traffic profile developed for the analysis in Subsection 3.2. Section 3.3 presents a preliminary study performed on a simplified topology to better understand the impact of OpenFlow 1.0 messages on the control channel based on forwarding rules installation and monitoring process. Section 3.4 depicts the analytical modeling made based on *Forwarding* behavior. Section 3.5 explains and discusses the results obtained on a campus network for messages related to install forwarding rules and collect statistics mainly.

3.1 Motivation and Methodology of Analysis

OpenFlow-based SDN introduces a simple way to develop and maintain communication networks because of its centralized logic of controlling forwarding devices. However, this simplicity may allegedly impose high cost on the network controller and create bottlenecks at the control channel (YEGANEH; GANJALI, 2014). Recent solutions, such as Devoflow (CURTIS et al., 2011) and DIFANE (YU et al., 2010), attempted to alleviate these bottlenecks by distributing the control logic of OpenFlow. Nevertheless, to the best of our knowledge, no other study has detailed in which situations such bottlenecks appear and whether they can or cannot be mitigated or even avoided by simple configuration, *i.e.*, without the need to develop a specific distributed controller. Therefore, in our analysis, we initially quantified the load of OpenFlow 1.0 control messages that appear most frequently in our specific scenario of study. Afterwards, we also pointed out configuration parameters that can influence this load and may be set to reduce the chance of bottlenecks in the control channel.

Our analysis is divided into two perspectives of resource consumption: (*i*) control channel load related to installation of rules on forwarding devices (*Packet-In*, *Modify-State*, and *Send-Packet*) and request/reply messages for monitoring flow statistics (*Read-State*); and (*ii*) resource usage in terms of forwarding rules, active and idle, installed on network devices. These four sub-types of messages have been selected because, in our scenarios of study, they represent the absolute majority of control traffic (accounting for 97.78% of the number of messages exchanged between the controller and forwarding devices, and 99.76% of the overall control traffic). Furthermore, regarding resource usage, the amount of rules installed in switches' expensive and limited TCAM needs to be carefully managed to avoid network devices running out of resources (CURTIS et al., 2011).

The metrics that we analyzed for control traffic load are: (*i*) bit rate and number of messages per second for monitoring, (*ii*) bit rate and number of messages for rule installation processed on the controller, and (*iii*) bit rate and number of messages for rule installation processed by network devices. The analyzed metrics for resource usage on forwarding devices are: (*i*) the

total number of installed forwarding rules and (*ii*) the amount of those installed forwarding rules which are idle, *i.e.*, counters unchanged between two monitoring intervals.

3.2 User Traffic Profile

The general prevalent traffic profile of users is characterized by everyday Internet surfing and access to the university's virtual learning environment. We modeled user behavior to generate emulated Internet traffic based on several previous studies (SYSTEMS, 2014; XIE et al., 2013; CHENG; LIU; DALE, 2013; KATSAROS; XYLOMENOS; POLYZOS, 2012; 3GPP, 2004). Table 3.1 summarizes all parameters used to emulate user traffic profile during our experimentation. In our experiments we generate only video and Web traffic, in a proportion of 75% to 25% respectively. Given the size of requests, *i.e.*, video streams generate more traffic than Web requests, we selected one user to place video requests for every 6 Web users. We included in our topologies one Web Server and one Video Server to respond to users' requests, so that we were able to control precisely the size of responses. We also assumed that all hosts are active during the whole experiment time and placed a request in average every 30 seconds.

Table 3.1 – Configuration parameters of user traffic profile.

Parameter	**Value**
Web request size	Lognormal Distribution ($\mu = 11.75$, $\sigma = 1.37$) Mean: 324 KBytes, Std. Dev.: 762 KBytes
User reading time	Exponential Distribution ($\lambda = 0.033$) Mean: 30 seconds
Video watch time	180 seconds
Video bit rate	300 kbps
Traffic Mix	Video: 75%, Web: 25%
User Mix	1 video user for every 6 Web users
Monitoring	Polling frequency: 5 seconds
Controller behavior	Floodlight's default *Forwarding Behavior* implementation
Experiment duration	30 min

Source: by author (2015).

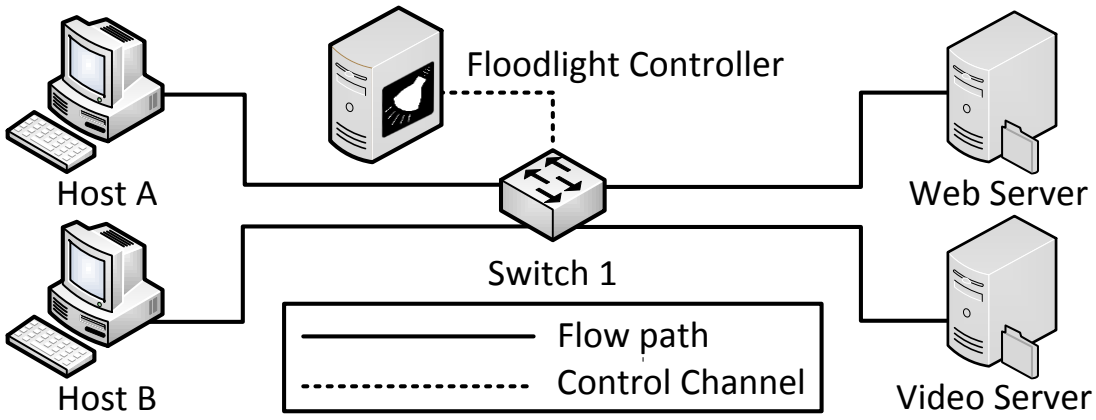
The experimentation workload was emulated on Mininet (LANTZ; HELLER; MCKEOWN, 2010) in one server with the following hardware description: Dell PowerEdge R815 with 4 AMD Opteron Processor 6276 Eight-Core processors and 64GB of RAM.

3.3 Preliminary Study

This section describes a preliminary study performed in a simplified topology with an inducted network traffic and controller behavior in order to evaluate OpenFlow 1.0 control messages overhead at the control channel.

To better identify the impact of every OpenFlow 1.0 control message, we designed a simplified topology with a single switch connecting two hosts, one Video Server and one Web Server. We conducted this experimentation using the *Forwarding* behavior implementation (see Section 2.3) – default controller implementation provided by Floodlight v0.90 – and we used the User Traffic Profile (see Section 3.2) except the User Mix parameter that was set to 50% for both Video and Web servers. This simplified scenario was chosen to easily identify which are the most common messages transmitted during the experimentation period. Figure 3.1 depicts the simplified topology with a single switch, two hosts, one Web Server and one Video Server.

Figure 3.1 – Simplified topology with a single switch connecting two hosts, a Web Server and a Video Server.



Source: by author (2015).

During the 30 minutes of experiment time, the controller has installed several forwarding rules on this single switch to enable hosts A and B to request for content on Video and Web servers. Further, the Floodlight controller also maintained its counters for every OpenFlow 1.0 sub-type of message (see Section 4.3 for details about our implemented extension on the Floodlight controller to support control channel statistics). These counters are accessible through the same *Representational State Transfer* (REST) API provided by Floodlight controller to report switch *Individual-Flow-Statistics*. We performed 300 monitoring requests for switch *Individual-Flow-Statistics* during each of all 30 experiments sized so to achieve a 95% confidence and an error no higher than 2%. When the experiment ends, we counted the amount of each sub-type of message to analyze which are the more frequent in our scenario of study.

Overall, the results shows that *Read-State*, *Modify-State*, *Packet-In*, and *Send-Packet* sub-types of message represent the four largest amounts of control messages generated. There are two different directions that these messages exchanged between the controller and forwarding devices can go: from switches to controller ($S \rightarrow C$) and from the controller to switches ($C \rightarrow S$). Table 3.2 presents all OpenFlow messages sub-types organized by its types (*Controller-to-switch*, *Asynchronous*, and *Symmetric*). Each sub-type of message was placed together with its respective percentage of packets processed and control overload in addition to separate them in *request* and *reply* messages indicating their direction on the control channel.

Table 3.2 – Percentage of packets processed and control channel load of each OpenFlow 1.0 control message transmitted in our preliminary scenario.

Message type	**Sub-type**	**Packets processed percentage**	**Control channel load percentage**	**Direction**
Controller-to-switch	*Features (Request)*	0.00%	0.00%	$C \rightarrow S$
	Features (Reply)	0.21%	1.23%	$S \rightarrow C$
	Configuration (Request)	0.00%	0.00%	$C \rightarrow S$
	Configuration (Reply)	0.21%	0.05%	$S \rightarrow C$
	Modify-State	17.28%	25.01%	$C \rightarrow S$
	Read-State (Request)	36.11%	36.68%	$C \rightarrow S$
	Read-State (Reply)	62.62%	31.31%	$S \rightarrow C$
	Send-Packet	20.74%	34.52%	$C \rightarrow S$
	Barrier (Request)	0.12%	0.02%	$C \rightarrow S$
	Barrier (Reply)	0.42%	0.07%	$S \rightarrow C$
Asynchronous	*Packet-In*	36.13%	67.20%	$S \rightarrow C$
	Flow-Removed	0.00%	0.00%	$S \rightarrow C$
	Port-Status	0.00%	0.00%	$S \rightarrow C$
	Error	0.00%	0.00%	$S \rightarrow C$
Symmetric	*Hello (Request)*	0.24%	0.03%	$C \rightarrow S$
	Hello (Reply)	0.21%	0.04%	$S \rightarrow C$
	Echo (Request)	25.39%	3.69%	$C \rightarrow S$
	Echo (Reply)	0.00%	0.00%	$S \rightarrow C$
	Vendor (Request)	0.12%	0.09%	$C \rightarrow S$
	Vendor (Reply)	0.21%	0.09%	$S \rightarrow C$

Source: by author (2015).

Echo request and *reply* messages – messages sent by both controller and switch to establish connectivity – also have an expressive amount of messages transmitted. However, this percentage is higher because there is a simplified scenario, there are only two hosts requesting for content and the *User Traffic Profile* (see Section 3.2) dictates that an user may delay its requests making the network idle most of the experiment duration. When the network is idle, the *Forwarding* behavior dictates that the controller does not need to send any other packet to switches and vice-versa, except using *Echo* messages to guarantee the connectivity of both. So, we decided to study the impact of other four sub-types of message aforementioned (*Read-State*, *Modify-State*, *Packet-In*, and *Send-Packet* sub-types). In the next section, we present an analytical modeling to understand what is the proportion that these messages are generated.

Because of the fact that *Read-State* messages are invoked by our monitoring mechanism and the *Forwarding* behavior does not use these messages by default, we decided to study the overhead imposed by these messages too. We focused on *Individual-Flow-Statistics* that are used to monitor individual flow counters and to analyze the amount of rules that these messages can retrieve, we performed several rule installations on a single switch and then we monitored those rules. The results obtained show that requests for *Individual-Flow-Statistics* have always the same length (12 bytes). On the other hand, the length of replies grow based on the number of rules installed on the switch. We noted that a *Read-State* can retrieve 682 rules per message. If the amount of rules pass this value, another *Read-State* is generated to reply the reminder.

3.4 Analytical Modeling

In this section, we present an analytical modeling for the impact of the *Forwarding* behavior implementation and our monitoring messages generated on the control channel. We divided our analytical modeling in two steps. First, we analyze OpenFlow messages related to monitoring (*Read-State*) that depends on the number of switches requested and the quantity of forwarding rules installed on them. Second, we analyze OpenFlow rules installation process (*Packet-In*, *Modify-State*, and *Send-Packet*) which depend on the *Forwarding* behavior implementation – used by default on Floodlight controller – that we used in our entire study.

Regarding *Read-State* messages, there are two variations to be considered in our analysis: (*i*) messages sent from the controller to the switches (*Read-State* request), requesting for statistics, and (*ii*) messages sent from switches to the controller, replying the statistics collected to the controller (*Read-State* reply). Messages sent from the controller to the switches are called by Floodlight controller as *Stats-Request* and messages that reply the collected statistics to controller are called *Stats-Reply*. Based on OpenFlow switch specification 1.0 (PFAFF et al., 2009), it is possible to forecast the overhead imposed by the size of *Stats-Request* and *Stats-Reply* messages analyzing the switches characteristics (*e.g.*, number of port and tables) and the quantity and granularity of forwarding rules installed.

All *Read-State* messages have a fixed header size with 12 bytes and a message body with a variable size. The variable size depends on the type of statistics requested and data that are collected in the reply message. *Stats-Request* messages have an expected size due to the type of statistics requested. *Individual-Flow-Request*, *Aggregate-Flow-Request*, *Port-Request*, and *Queue-Request* message requests require to include an additional information in the message body. This information size corresponds to 44, 44, 8, and 8 bytes respectively. *Description-Request* and *Table-Request* do not require additional information in the message body, therefore, do not include additional size to the message body.

On the other hand, *Stats-Reply* messages vary depending on both type of statistics and quantity of information requested. For example, messages that retrieve port information (*Port-Request*) include an additional message body of 104 bytes for each switch port that each contains. Assuming that a switch does not change the number of ports – this is possible on virtual switches –, a switch with 24 ports always generates *Stats-Reply* messages with 2058 bytes. However, for *Individual-Flow-Statistics* messages for instance, this forecasting is significantly more complex, since the quantity of installed rules on switches and their granularity depends on both controller behavior implementation and user traffic generated on the network.

The OpenFlow specification 1.0 (PFAFF et al., 2009) dictates that a *Read-State* message can reply counters with a length representable on a 16 bits *Unsigned Integer*. This means that the counter of these messages retrieved can be represented in 65536 bytes. Consequently, if we discount the *Read-State* fixed header size (12 bytes) of this value and divide the rest (65524 bytes) by the length of each forwarding rule installed (96 bytes), the result is the same

of the previous experiment made in Section 3.3. Each *Read-State* messages can reply at most 682 forwarding rules about *Individual-Flow-Statistics* and one request for these statistics can generate more than one reply message, depending on the number of exceeding rules installed.

Assuming that each flow is a logical connection – normally bidirectional and unicast – between two endpoints/hosts of the network that pass through several devices, each flow path device needs at least two rules installed to establish the end-to-end connection (forwarding rules to from both directions). Since the size of *Stats-Request* messages are simple to calculate, the monitoring overhead imposed by *Stats-Reply* messages ($SRMO$) are more complicated. To make the explanation more didactic, we decided to divide the overall monitoring overhead in two parts, considering a fixed and a variable equation. Equation 3.2 represents the fixed monitoring overhead, Equation 3.3 represents the variable monitoring overhead, and Equation 3.1 represents the sum of the two previous. The equations are affected by the quantity of switches in the flow path and forwarding rules installed at the moment.

$$SRMO = SRMO_F + SRMO_V \tag{3.1}$$

$$SRMO_F = N_{switches} * (H_{OF} + H_{StatReq}) \tag{3.2}$$

$$SRMO_V = \sum_{f \in F} \left(\sum_{s \in Path_f} (Body_{IFS} * 2) \right) \tag{3.3}$$

The fixed part of the monitoring overhead (Equation 3.2) corresponds to the OpenFlow default header (H_{OF}) plus the specific header for *Stats Reply* messages (H_{SR}), totaling 12 bytes. This value is also multiplied by the quantity of requested switches in the network, once each switch is responsible to reply individually about their forwarding rules statistics (having forwarding rules installed or not). However, this fixed part also grows if the number of forwarding rules installed exceed the length of a *Stats-Reply* message. A *Stats-Reply* message can retrieve only 682 rules per message, accounting a length of 65536 bytes. If this value is exceeded, another *Stats-Reply* is generated for each 682 rules to retrieve all the information. Then, another packet with a 12 bytes header (H_{OF}) is generated on the channel. On the other hand, the variable part of the monitoring overhead depends on the quantity of active flows (set F) and the quantity of requested devices that these flows pass through (set $Path_f$). Therefore, for each active flow f and switch s in the flow path the *Individual-Flow-Stats* ($Body_{IFS}$) size is calculated by adding data for these flows on both directions (source to destination and destination to source). The $Body_{IFS}$ size is 88 bytes plus 8 bytes per action totaling 96 bytes considering that the flows are unicast and have always just one action associated.

Regarding the forwarding rules installation process, the *Forwarding* behavior uses *Packet-In*, *Send-Packet*, and *Modify-State* messages to coordinate all flows in the network. They are mostly invoked when a switch receive a message that the packet header does not match with any

entry in the flow table. In this case, the default switch action is to encapsulate the message and send it in a *Packet-In* message to the controller. After that, the controller sends *Modify-State* messages to the entire flow path and one *Send-Packet* message to the switch that generated the *Packet-In* (See the detailed explanation in Section 2.3). Obviously, the controller is only able to operate this way when the host is known. On the other hand, *i.e.*, when the host is unknown, messages are flooded until the controller finds the source and destination hosts.

Packet-In messages are sent from switches to the controller always when a packet header does not match any entry in a flow table to forward this packet. These messages are composed by a header with 20 bytes plus the entire received packet. Generally, *Address Resolution Protocol* (ARP) or *Transmission Control Protocol-SYN* (TCP-SYN) are the most common types of messages that are sent with *Packet-In* messages, just because in most cases, these messages are the first messages of new flows. For example, in this case these messages will add 42 and 72 bytes on *Packet-In* size respectively. However, because a forwarding rule has an idle and a *hard timeout* which determines when a forwarding rule expires without the controller intervention, *Packet-In* messages can be generated by other types of messages also. Thus, *Packet-In* are generated more frequently when idle and *hard timeouts* are short, on the other hand, when they are bigger, the probability of a *Packet-In* arrive to the controller is more sporadic.

Send-Packet and *Modify-State* messages are commonly used as an answer for a *Packet-In* messages received from switches. Equation 3.4 represents the overhead imposed by these messages to forwarding rule installation (FO) generated by an OpenFlow network.

$$FO = \begin{cases} N_{switches} * (H_{OF} + H_{SP} + M_{Original}) & if\ unknown\ host \\ \left(\sum_{s \in Path_f} (H_{OF} + H_{MS}) \right) + (H_{OF} + H_{SP} + M_{Original}) & if\ known\ host \end{cases} \tag{3.4}$$

FO is calculated differently in two occasions: (*i*) when the controller receives a *Packet-In* message to install a forwarding rule in a switch and the position of the destination host of the flow is unknown by the controller and (*ii*) when it is known by the controller. In the first occasion, the controller will flood *Send-Packet* messages ($H_{OF} + H_{SP} + M_{Original}$) to all switch ($N_{switches}$) to find the destination host. In the second occasion, when the position of the destination host of the flow is known, the controller sends several *Modify-State* messages ($H_{OF}+H_{MS}$) for each switch s over the flow path $Path_f$ and one *Send-Packet* and *Modify-State* ($H_{OF}+H_{SP}+M_{Original}$) for the switch that generated the *Packet-In*. Generally, the occasion (*i*) occurs once to establish the going traffic and the occasion (*ii*) occurs for the subsequent traffic because the host is already known by the controller.

It is important to emphasize that the overhead imposed by transport protocols such as TCP, *User Datagram Protocol* (UDP), and *Secure Socket Layer* (SSL) for example, are not included in the control traffic for controller neither switches. Moreover, *Packet-In* messages are not

included in FO – as well as *Stats-Request* are not included in $SRMO$ also – once that represent traffic on different directions in the control channel. However, to estimate the overhead of *Packet-In* messages is simple. In the occasion *i* are generated one *Packet-In* message from every switch that receive the flood, on the other hand, in the occasion *ii*, only the first switch of the flow path generates a *Packet-In* to the controller.

3.5 Experimental Evaluation

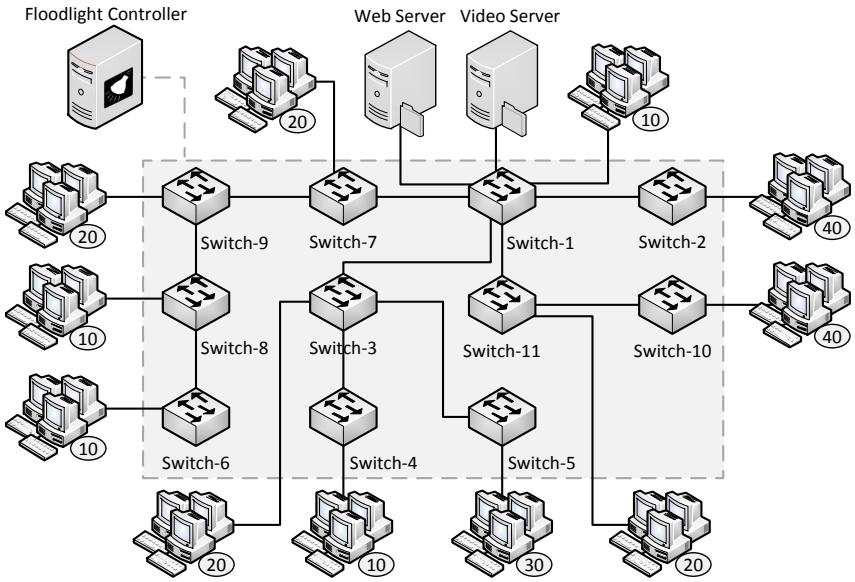
In this section, we provide an analysis of the control channel traffic load of OpenFlow 1.0 messages in a campus topology, emphasizing those messages that occur most frequently in our previous scenario of study. First, in Subsection 3.5.1 is detailed the scenario with the topology and workload chosen to perform our control channel analysis. Next, in Subsection 3.5.2 is presented and discussed the experimental results obtained in order to identify the impact of those messages related to control channel load and resource usage metrics.

3.5.1 Case-Study: A Campus Network

To conduct our control channel analysis and also to evaluate our approach to SDN management (later discussed in Chapter 4), we have chosen to use a campus network scenario inspired in our own university premises. This type of network infrastructure is well suitable to benefit from features of SDN with OpenFlow, which has actually been originally conceived for this type of scenario (MCKEOWN et al., 2008). The OpenFlow 1.0 was chosen because of the fact that this version is considered the most relevant SDN implementation nowadays. Our emulated campus network consists in 11 OpenFlow switches connecting 230 hosts from laboratories and administration offices forming the topology. Each switch of the network is connected through a dedicated channel to a remote controller. More specifically, a centralized Floodlight v0.90 controller is set to coordinate network-wide forwarding decisions.

To generate traffic in this experimental evaluation we used the User Traffic Profile (see Section 3.2) characterized by everyday Internet surfing and access to the university's virtual learning environment. This model dictates the user behavior to generate emulated Internet traffic based on several previous studies (SYSTEMS, 2014) (XIE et al., 2013) (CHENG; LIU; DALE, 2013) (KATSAROS; XYLOMENOS; POLYZOS, 2012) (3GPP, 2004). In this experiment we generate only video and Web traffic, in a proportion of 75% to 25% respectively. Given the size of requests, *i.e.*, video streams generate more traffic than Web requests, we selected one user to place video requests for every 6 Web users. We include in the campus topology (shown in Figure 3.2) one Web Server and one Video Server to respond to users' requests, so that we are able to control precisely the size of responses (in a real campus network these requests would normally go through a gateway or proxy). We also assume that all 230 hosts are active during the whole experiment time and place a request in average every 30 seconds.

Figure 3.2 – Campus network topology.



Source: by author (2015).

The experimentation workload was emulated on Mininet in one Dell PowerEdge R815 with 4 AMD Opteron Processor 6276 Eight-Core processors and 64GB of RAM server.

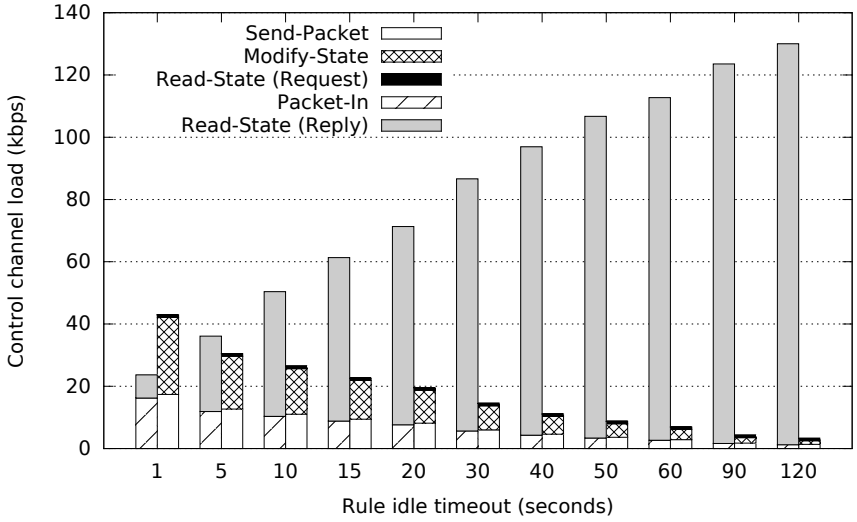
3.5.2 Experimental Results & Discussion

To understand the variations of control traffic, we varied one factor present in any OpenFlow-based network, which is the *idle timeout* configuration of forwarding rules. This factor indicates when an entry of the flow table of an OpenFlow switch, *i.e.*, a forwarding rule, is removed due to lack of activity. The default *idle timeout* (in seconds) is configurable in the Floodlight controller and is applied for every new forwarding rule installed. Figures 3.3, 3.4, and 3.5 show how the *idle timeout* configuration affects both the control channel load and resource consumption due to the frequency of rules installation process. Moreover, referring to analyze *Read-State* messages, we fixed the polling frequency in 5 seconds to understand how the variation in size (not the frequency) of these messages impacts control traffic, specially related to their size.

Figure 3.3 shows how the control traffic load (in Kbps) varies as the *idle timeout* increases. The height of each bar shows the total control traffic in both directions, *i.e.*, from the controller to the network (*Send-Packet*, *Modify-State*, and *Read-State (Request)*) and from the network to the controller (*Packet-In* and *Read-State (Reply)*). It is clear to notice that the average traffic of *Read-State requests* remains constant, while *Read-State reply* traffic increases significantly. This happens because *Read-State replies* contain forwarding rule statistics, thus as the *idle timeout* increases so does the chance of a given forwarding rule being installed at a switch in each monitoring poll. However, *Packet-In*, *Send-Packet*, and *Modify-State* traffic decreases in average as the *idle timeout* increases. This happens because all these messages will appear

mostly if users remain inactive for a period longer than the configured *idle timeout*, which is fairly unlikely to happen for long *idle timeout* values.

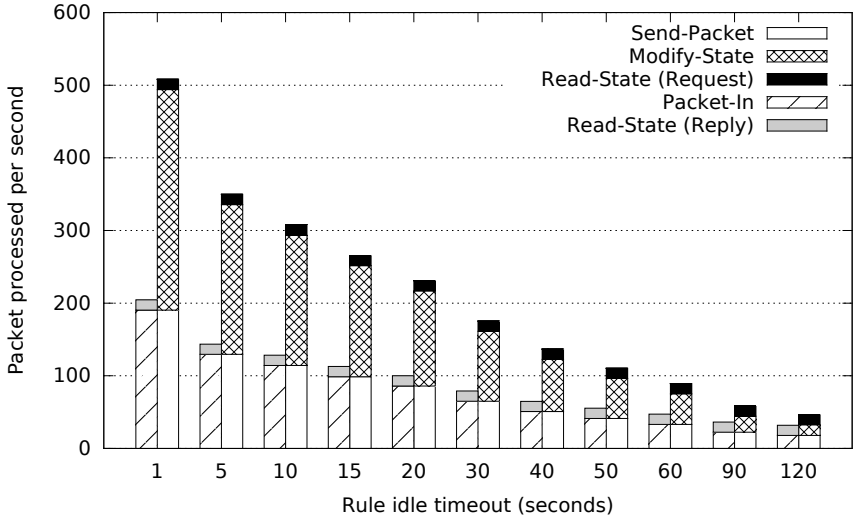
Figure 3.3 – Control channel load vs. *idle timeout* configuration.



Source: by author (2015).

Figure 3.4 shows the number of packets processed per second by both the controller and network devices. Similar to Figure 3.3, this chart also shows in the total bar height an accumulated value, *i.e.*, the number of messages flowing in each direction. Mainly, the results in Figure 3.4 show that when the rule *idle timeout* configuration is set to low values, the average of *Packet-In* generated to the controller direction and *Send-Packet*/*Modify-State* messages sent to network devices both increase. Most importantly, both network and controller have a limit of packets that can be processed before reaching the warning operation mode. Thus, the administrator needs to watch over this configuration to avoid bottlenecks at the control channel.

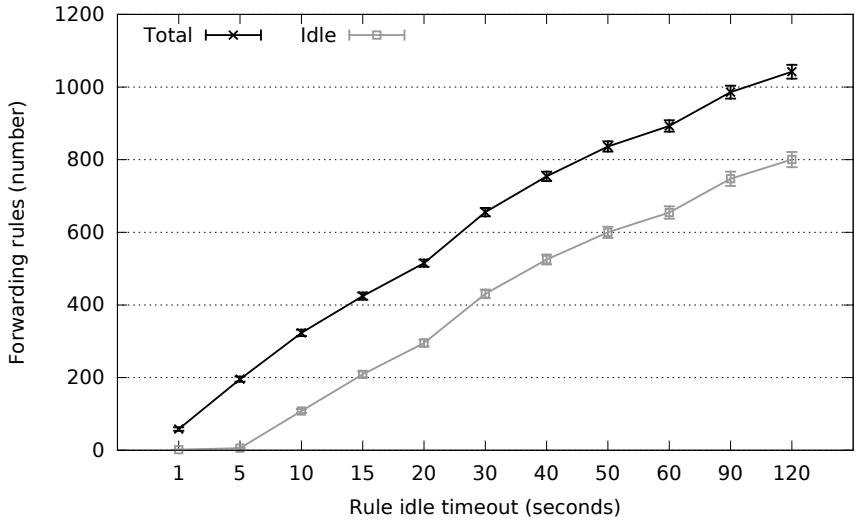
Figure 3.4 – Packets processed per second vs. *idle timeout* configuration



Source: by author (2015).

Figure 3.5 shows the total number of forwarding rules and which of those are idle for a given *idle timeout* configuration. A forwarding rule is idle when its counters do not change between two monitoring polls. The results show that increasing the *idle timeout* value causes a larger number of idle rules installed in switches. This occurs because the user traffic profile of most users generates sparse and short-lived requests (*e.g.*, Web requests). Considering our scenario, when the *idle timeout* is set to 120 seconds, for example, the average rules installed in the network is about 1042, being 77% of these rules inactive. Given that the switches' TCAM are an expensive resource to be wasted and the number of rules that can be stored in these memories is limited, it is important to control this configuration closely to avoid running out of resources. Thus, for this point of view, the best configuration is to define 1 second to rules *idle timeout* that maintain almost zero idle rules installed on switches and less control traffic due to less rules to be monitored. On the other hand, setting the *idle timeout* configuration as 1 second also generates more packets processed per second on the controller's direction due to rule installation process is frequent (*Packet-In*, *Modify-State*, and *Send-Packet* messages).

Figure 3.5 – Total/idle rules vs. *idle timeout* configuration



Source: by author (2015).

In summary, this analysis shows how a single factor, *i.e.*, the idle timeout, configured at the controller can significantly affect the control channel traffic and resource consumption of an OpenFlow-based network. Moreover, all experiment samples were sized so to achieve a 95% confidence and an error no higher than 2%. We also suppressed error bars from Figures 3.3 and 3.5 for the sake of visualization. Furthermore, it is important to mention that we discovered a hard limit to collect statistics with *Read-State* messages on the controller implementation (in our experiment this limit was 682 rules per *Read-State Reply* message). If a *Read-State Request* is sent to a switch with more than 682 rules installed, the switch generates all necessary *Read-State reply* messages to retrieve all information from switches. Thus, when a switch has more installed rules than a *Read-State* message supports, more packets are generated in order to reply all flow statistics about this switch on the control channel.

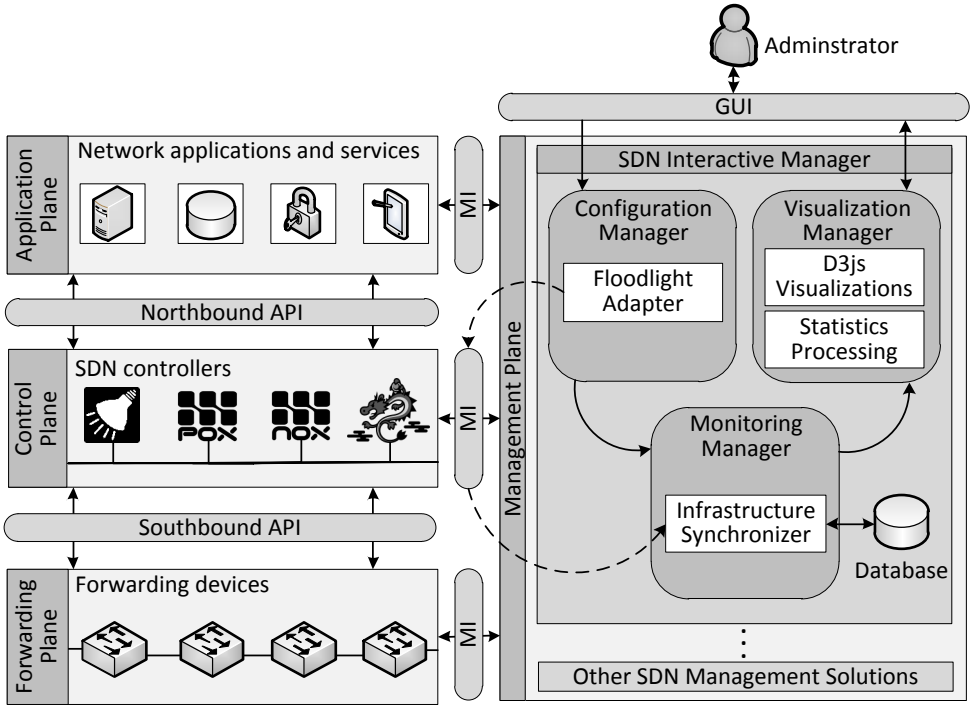
4 SDN INTERACTIVE MANAGER

In this chapter is presented our interactive approach to SDN management through monitoring, visualization, and configuration. First, Section 4.1 details all components and how our approach lines up with the general SDN conceptual architecture. Second, Section 4.2 presents our SDN management requirements that our approach achieved. Next, Section 4.3 presents our developed extensions on the Floodlight controller to support advanced control channel management features. Last, Section 4.4 details our prototype implementation.

4.1 Conceptual Architecture

Our conceptual architecture was designed to archive a management loop that involves the administrator interactions with SDN monitoring, visualization, and configuration components. Basically, we designed an approach in the SDN *Management Plane* – called *SDN Interactive Manager* – that integrates the administrator interactions with SDN monitoring, visualization and configuration management activities. Figure 4.1 shows the SDN architecture, the administrator and the *Graphical User Interface* (GUI) along with the components added to enable management via monitoring, visualization, and configuration. To introduce concepts more didactically, we organize the explanation in two steps. First, we present an overall explanation of our proposed architecture and a brief explanation about the SDN architecture. Second, we explain all components of our approach and how the *Administrator* can interact with them.

Figure 4.1 – Conceptual Architecture



Source: by author (2015).

SDN Interactive Manager includes three main components: *Monitoring Manager*, *Visualization Manager*, and *Configuration Manager*. The *SDN Interactive Manager* sits in the *management* plane of SDN alongside other existing or yet to be developed solutions. Since our approach to SDN management comprises interactive network management, we also depict in the architecture the *Administrator* who interacts primarily with the *Visualization Manager* and the *Configuration Manager* components through a *Graphical User Interface (GUI)*. Therefore, our solution creates a loop of activities by integrating these components with the *Administrator* interactions. Each of these components perform independent tasks described as follows.

Monitoring Manager – This component is responsible for retrieving updated information about the network and storing it in a local *Database*. This is performed mainly through a module called *Infrastructure Synchronizer*, which collects information, such as traffic statistics, network topology and device data, by accessing an MI to one or more controllers situated in the *control* plane. Currently, there is no standard MI, though we envision that such interface could present at least the same functionality as the *northbound* API. Also, customizations in this API could be added to support information relevant to network management (*e.g.*, control traffic counters and resource usage data). Then, the *Infrastructure Synchronizer* module stores both control and data traffic statistics, maintaining a history of the last network state and changes on SDN-related configurations performed by the *Administrator*.

Visualization Manager – This component comprises the *Statistics Processing & Provider* and *Chart Visualizations* modules. With information stored in the *Database*, the *Statistics Processing & Provider* module is able to aggregate data per host, switch, controller, or even the entire network to be used by the *Chart Visualizations* module. Furthermore, the *Statistics Processing & Provider* module is also able to identify which rules are active and which are idle on forwarding devices. To build interactive visualizations that can be analyzed by the *Administrator*, the *Chart Visualizations* module uses a rich library of interface components (*e.g.*, graphs, charts, and diagrams) that enables data updates in real time. Then, based on these visualizations, the *Administrator* can be aware of possible issues or bottlenecks and plan for adjustments and improvements in the network configuration.

Configuration Manager – This component allows the *Administrator* to check and configure SDN-related parameters on network controllers through the MI. The *Floodlight Adapter* module permits setting up the polling interval for monitoring network devices through a friendly GUI. Moreover, through the same GUI, the *Administrator* can trigger the *Floodlight Adapter* module to set the *idle timeout* to be configured globally, per device, or per flow. All SDN-related parameter configurations on the controller are performed using an extension of the *northbound* API implementation that is embedded in MI between *management* and *control* planes. This extension is needed to support SDN-related parameters configuration changes and requests for current configurations.

Our prototype requirements, diagrams, and the development of our conceptual architecture are detailed in the next sections. Our sequence diagrams present only the case of success.

4.2 Prototype Requirements

In this section, we present all *Functional Requirements* (FRs) identified to develop a prototype for SDN management through monitoring, visualization, and configuration. We summarized all relevant requirements identified to manage and control the control traffic generated in OpenFlow-based networks. We chose to emphasize the control traffic because it is practically neglected nowadays, specially using the OpenFlow version 1.0. Therefore, our FRs for SDN management through monitoring, visualization, and configuration are the following presented in Table 4.1.

Table 4.1 – Prototype functional requirements and descriptions

Functional Requirement	**Description**
FR-01	The prototype must be able to perform monitoring on an SDN controller to collect statistics about the control channel
FR-02	The prototype must sync and store statistics about controllers, switches, links, flows, ports, tables, and counters into a database
FR-03	The prototype must identify, count, and aggregate all active and idle flows installed on forwarding devices
FR-04	The prototype must calculate data and control traffic bit rates and packets processed to all switches of the network
FR-05	The prototype must present a physical topology visualization with node information (DPID, data and control channel load rates and level, rules installed/overall, MAC, and IP addresses depending on node type)
FR-06	The prototype must generate visualizations based on the control channel load rates and number of packets processed for the chosen messages (*Packet-In*, *Modify-State*, *Sand-Packet*, and *Read-State*)
FR-07	The prototype must present the total number of active and idle rules installed by controller in interactive visualizations
FR-08	The prototype must enable the administrator to perform SDN-related parameter configurations through an GUI (*rule idle timeout* and *polling interval* configurations)
FR-09	The prototype must update interactive visualizations on each monitoring interval and change its physical topology visualization based on control channel load and resource usage metrics
FR-10	The SDN controller must count, aggregate, and provide all collected statistics about the control channel for OpenFlow 1.0 version

Source: by author (2015).

In addition to presenting all prototype implementation, it is also important to depict all restrictions that we needed to develop on the controller. Because the fact that OpenFlow controllers – by default – did not concern on counting statistics about the control channel, we modified the standard implementation of a SDN controller aiming to enhance the statistics collector by adding the control channel messages counters. For example, the FR-10 was developed on a

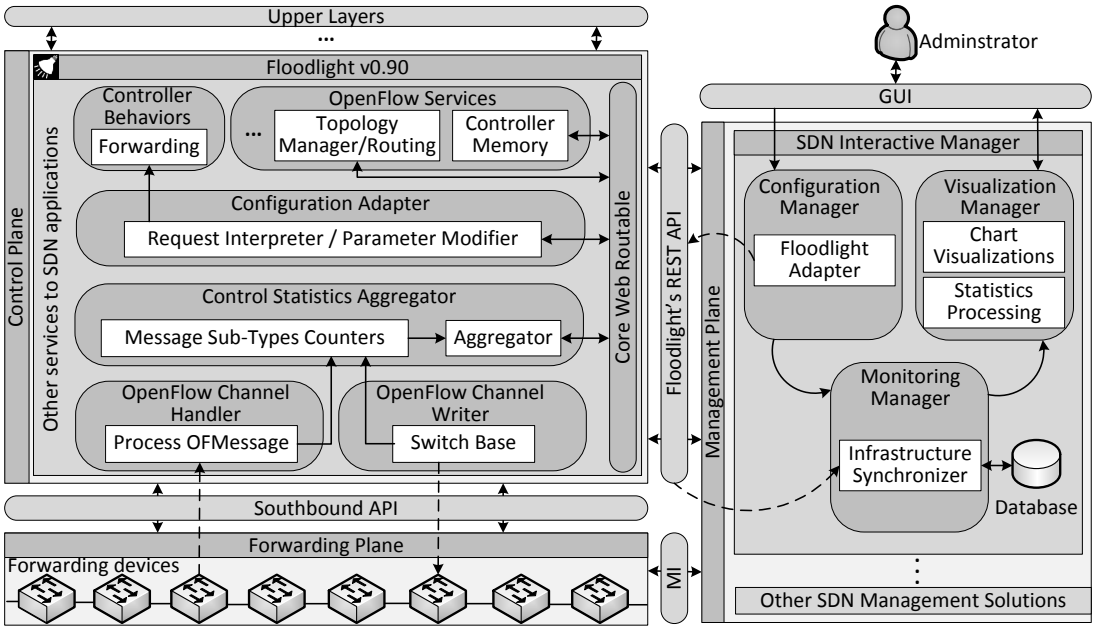
Floodlight v0.90 controller in order to count and reply for statistics about the control channel. In this case, we implemented all counters for OpenFlow 1.0 sub-types of message. Furthermore, we also added this information together with the switch flow information provided by its RESTful API. A more detailed description about this implementation is presented in the next section.

4.3 Controller Modifications

This section presents all modifications made on Floodlight v0.90 to support our advanced management features. First, is presented the Floodlight architecture with some abstractions and new modules added to support control channel statistics. Next, in Subsection 4.3.1 is depicted the *Configuration Adapter* component with its respective relationship with other developed components. Last, in Subsection 4.3.2 is presented the *Control Statistics Aggregator* module developed to handle all control channel counters and provide them to the REST API.

To develop our interactive visualizations, we also needed to request for information that the default controller already provides, such as physical topology information (*e.g.*, details about hosts, switches, links, ports and controller memory). The *OpenFlow Services* component comprehends the *Topology Manager/Routing* and *Controller Memory* modules that allow to request for topology and controller memory usage information respectively. These modules receive a request, process an updated information, and provide it through *Core Web Routable* component. The *Core Web Routable* component offers *Uniform Resource Locators* (URLs) that enable to retrieve/send data from/to these modules. Other OpenFlow services are accessible and can be implemented on Floodlight v0.90, but all used modules are shown in Figure 4.2.

Figure 4.2 – Controller Modifications

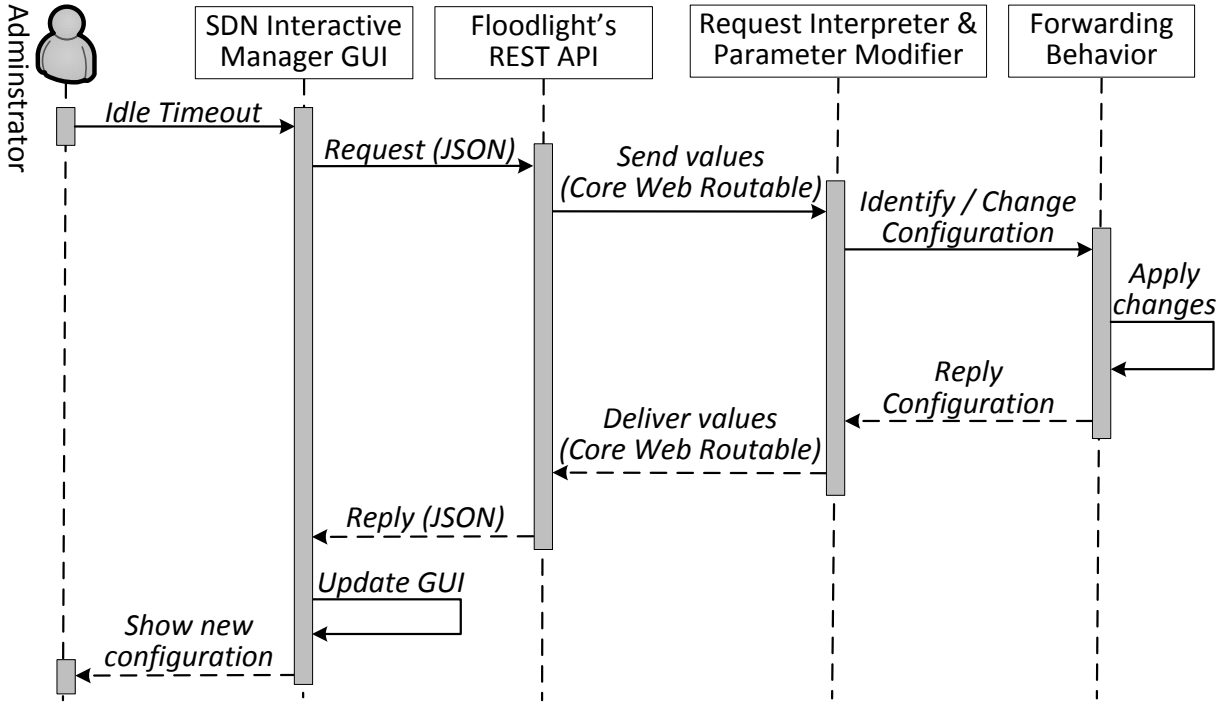


Source: by author (2015).

4.3.1 Configuration Adapter

The *Configuration Adapter* component is responsible for modifying – when requested by the administrator – the *Forwarding* implementation parameters in order to change its behavior. This component is composed by the *Request Interpreter / Parameter Modifier* module that performs SDN-related parameters reception and recognition (sent from *SDN Interactive GUI* through Floodlight's *Core Web Routable* module) and makes SDN-related parameters changes on Forwarding behavior implementation as well as the administrator requested. To better explain this configuration process, a sequence diagram is presented in Figure 4.3 that represents an example of an *idle timeout* parameter configuration made through our developed prototype.

Figure 4.3 – Sequence diagram of Floodlight controller configuration process



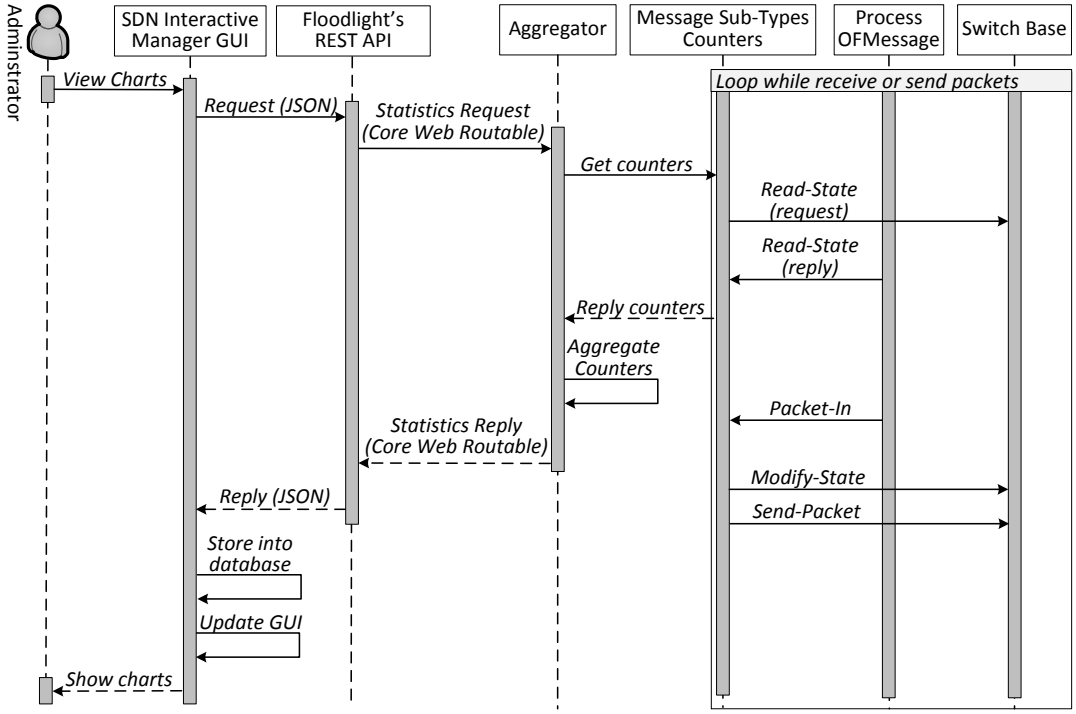
Source: by author (2015).

Basically, this sequence diagram (Figure 4.3) illustrates the administrator sending the *idle timeout* configuration value through *SDN Interactive Manager* GUI in order to alter this value on Floodlight's *Forwarding* behavior. First, the *SDN Interactive Manager* GUI receives this value and sends it in a *Javascript Object Notation* (JSON) to the Floodlight's REST API. Then, this API routes the request – with the *Core Web Routable* module – to reach the *Request Interpreter & Parameter Modifier* module. Next, the *Request Interpreter & Parameter Modifier* module recognizes all parameters to be modified and changes these values on *Forwarding* behavior implementation. Last, the updated values are replied on reverse path until *SDN Interactive Manager* updates its GUI and shows the current configuration to the administrator.

4.3.2 Control Statistics Aggregator

The *Control Statistics Aggregator* component includes *Message Sub-Types Counters* and *Aggregator* modules. The *Message Sub-Types Counters* module comprehends all OpenFlow message types with their respective sub-types in addition to define their characteristics and available memory dedicated to each one. This module is responsible to count the number and length of each message sub-type transmitted over the control channel. To perform this counting, this module communicates with two components: *OpenFlow Channel Handler* (to count messages that came from switches to controller) and *OpenFlow Channel Writter* (to count messages that are sent from controller to switch's direction). The *OpenFlow Channel Handler* uses *Process OFMessage* and *OpenFlow Channel Writer* uses the *Switch Base* module to perform this counting. Last, the *Aggregator* module aggregates all counters per switch and provides it to *Core Web Routable*. An example of control channel statistics retrieving is shown in Figure 4.4.

Figure 4.4 – Sequence diagram of Floodlight controller counting process



Source: by author (2015).

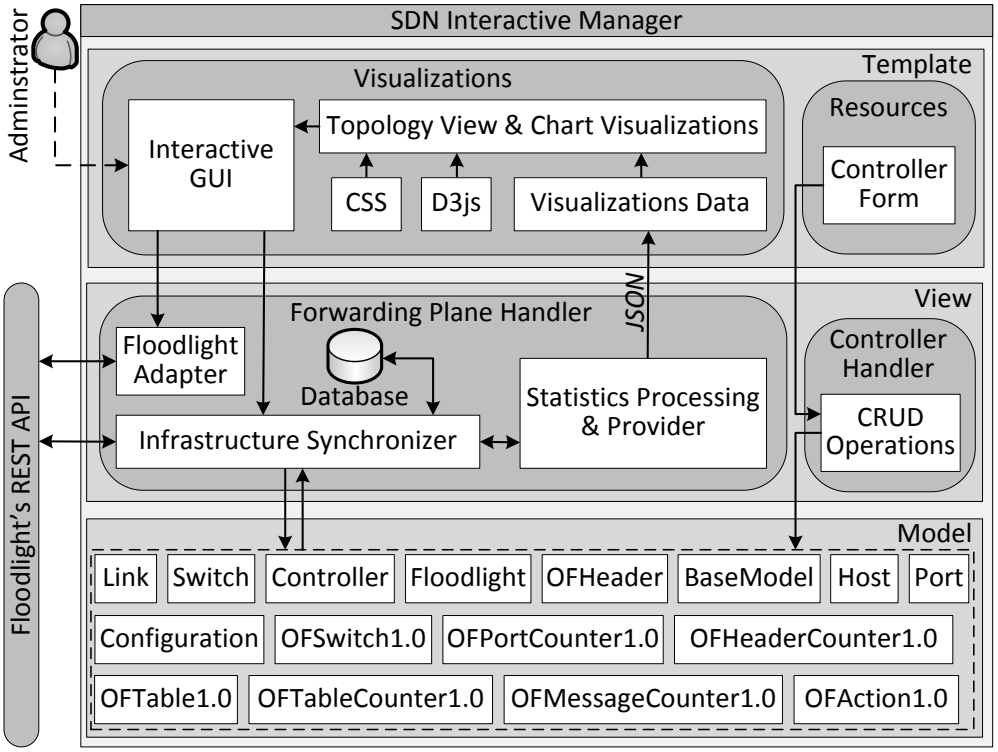
In this sequence diagram (Figure 4.4) is presented the entire process to retrieve control channel statistics through our prototype. First, the administrator accesses the *SDN Interactive Manager GUI* in order to visualize charts about control channel statistics. Our GUI performs several monitoring polls (as previously configured by the administrator) to *Floodlight's REST API* that interacts with *Core Web Routable* module to route this request to the *Aggregator* module. The *Aggregator* module gets all counters for all sub-types of message transmitted at the moment on *Message Sub-Types Counters* module and replies them to *Floodlight's REST API*. The *Message Sub-Types Counters* counts all messages sent to *Switch Base* (from controller to

switches) and received by *Process OFMessage* (from switches to controller). Last, the API returns these counters in a JSON format to *SDN Interactive Manager GUI* that stores into a database and performs updates on all charts showing them (in real time) to the administrator.

4.4 Prototype Implementation

We designed our prototype as a modular application with independent and integrated functionalities for SDN monitoring, visualization, and configuration. The prototype was based on an IaaS (*Infrastructure as a Service*) cloud platform – called *Aurora Cloud Manager* (WICKBOLDT et al., 2014) – following the *Model-View-Template* (MVT) design pattern and Python 2.7[1] with Django 1.6[2] as development framework. The *Database* used to store network devices information as well as traffic statistics is a PosgreSQL[3] database and it has been implemented as Django Models. Each of monitoring, visualization, and configuration functionalities were developed as Views and the GUI as Templates. Moreover, we also integrated our prototype with Floodlight v0.90[4], which is a Java-based SDN controller, supported by a community of developers and engineers of Big Switch Networks. Figure 4.5 depicts our prototype implementation based on MVT design pattern.

Figure 4.5 – Prototype Implementations



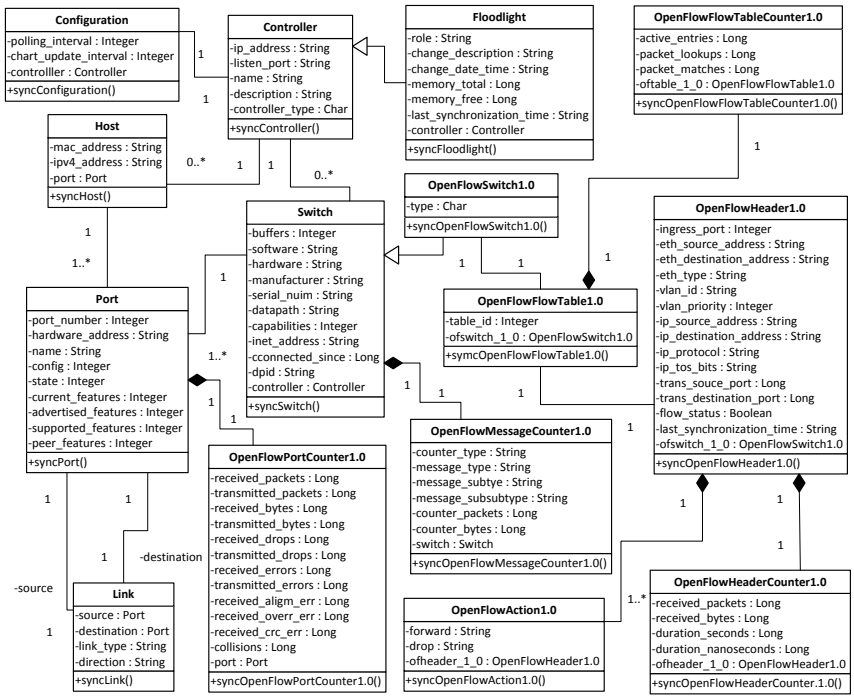
Source: by author (2015).

[1]https://www.python.org/
[2]https://www.djangoproject.com/
[3]http://www.postgresql.org/
[4]http://www.projectfloodlight.org/floodlight/

Figure 4.5 presents all prototype layers, components and modules with their respective interactions with the administrator and Floodlight's REST API. The bottom layer (*Model* layer) comprehends classes as Django models to maintain the necessary characteristics of OpenFlow specification 1.0. On the middle layer (*View* layer) are implemented the *Controller Handler* component that performs *Create, Read, Update and Delete* (CRUD) operations on controllers and the *Forwarding Plane Handler* component that is responsible for configuring SDN-related parameters, syncing, processing, and providing statistics to the upper layer (*Template* layer). On *Template* layer is developed the *Resources* component to coordinate operations with controllers and the *Visualizations* component that comprehends the interactive GUI including graphs using D3.js[5] library, jQuery[6], Bootstrap[7] *Cascading Style Sheets* (CSS), and JSON data necessary to generate interactive visualizations.

Regarding *Model* layer, our Django models are modeled from scratch because the fact of the *Distributed Management Task Force's* (DMTF) *Common Information Model* (CIM) is rather complex and may cause an extra overhead on storing all data into a database[8]. Complex models can introduce higher overhead on our prototype syncing and configuring processes and in this case we preferred to extract just the data that we used on our prototype evaluation. Therefore, we modeled a more simple data scheme to support control messages of OpenFlow-based networks with classes presented below on Figure 4.6.

Figure 4.6 – Class diagram of prototype models



Source: by author (2015).

[5]http://d3js.org/
[6]http://api.jqueryui.com/1.10/
[7]http://getbootstrap.com/
[8]http://dmtf.org/standards/cim/

Our prototype implementation was guided to accomplish the three main components of our conceptual architecture (*Monitoring Manager*, *Visualization Manager*, and *Configuration Manager*) making a loop with these management activities together with the administrator interaction. To better explain prototype functionalities, we divided the explanation presenting a (*i*) brief description about the implementation and the correlation of these components with the conceptual architecture (see Section 4.1) and then we (*ii*) present sequence diagrams used on development process to configure and retrieve statistics information from Floodlight controller.

Regarding the *Monitoring Manager* component, our implementation focused on periodically updating network information into a database. The *Infrastructure Synchronizer* module syncs data from controller (*e.g.*, data and control traffic counters of every rule installed on each switch of the topology) through the RESTful API provided by the Floodlight controller. This data is mapped in Django models to the database and provides them to *Statistics Processing & Provider* module. The *Statistics Processing & Provider* accesses information about the physical topology, including links, switches, and hosts in order to calculate traffic rates, active and idle flows on the network. Then, this information is calculated and serialized to be sent to *Topology View & Chart Visualizations* interface. Moreover, the *Floodlight Adapter* module is responsible to receive the requested configuration from GUI and send it to the Floodlight's REST API. Therefore, when the Floodlight controller finishes the configuration and sends a reply to our prototype, the administrator receives the updated configuration on GUI.

The *Visualization Manager* component is implemented as a Web based application relying mainly on *Visualizations* and *Resources* components. The *Resources* component includes the interface to perform CRUD operations with controllers. The *Visualization* component is composed mainly of an *Interactive GUI* with *HyperText Markup Language* (HTML) pages with CSS and charts made from *Visualizations Data* using D3.js library and jQuery mainly. The major purpose of this component is to provide an interactive way for the administrator to understand the current network status and visualize the impact of his/her configurations in real time. Thus, as the network is periodically monitored by the *Monitoring Manager*, visualizations are updated in the same pace. The integration between these two components allows our prototype to display network visualizations, such as: topology view and chart visualizations (*Topology View & Chart Visualizations*) with intuitive hints on where resource bottlenecks might be occurring, charts of idle and active rules installed on forwarding devices, amount of OpenFlow messages flowing in control channels, and the traffic rate these messages generate.

The implementation of the *Configuration Manager* aims to provide an easy way for the *Administrator* to change the network configurations through the same *Interactive GUI* where visualizations are displayed. For that purpose, we developed a module as an extension of the Floodlight default REST API so that configuration parameters can be sent from our prototype to the controller. Our prototype currently supports the configuration of the rule *idle* and *hard timeouts* parameters on the controller – made through *Floodlight Adapter* module –, but the implementation can be easily extended to support others. The forwarding rule *hard timeout* dic-

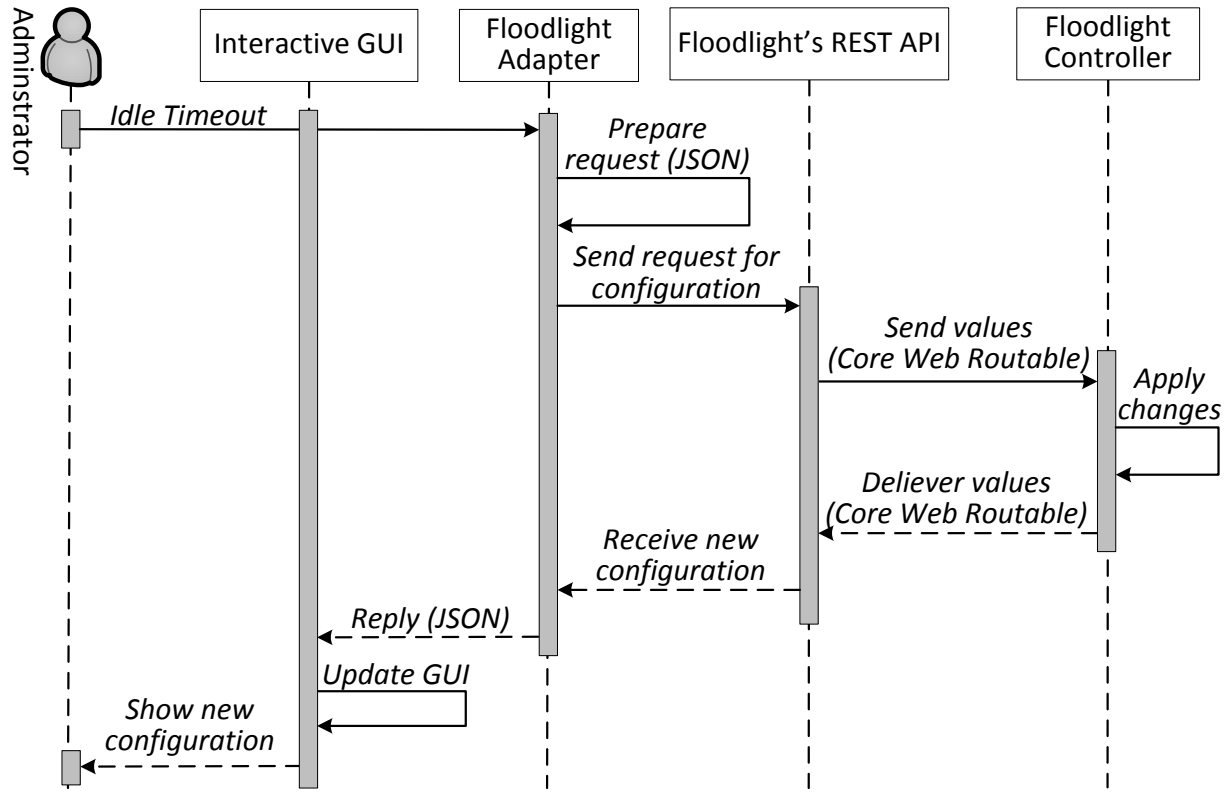
tates how many seconds a rule remains installed on a switch independently of their status (active or idle). Parameters as the *polling interval* and the *chart update interval* can also be set through our prototype. The monitoring *polling interval* affects the frequency of reading information from counters of network devices and the chart *update interval* only impacts on GUI refresh interval. The impact of these parameters on the control channel load and resource consumption, as well as how visualizations reflect their changes are presented in the next chapter.

In next subsections are presented both configuration and syncing (Subsecs 4.4.1 and 4.4.2) processes made through our prototype.

4.4.1 Configuration Process

The controller configuration process initializes with the administrator interacting by requesting a new configuration through the *Interactive GUI*. The administrator can change the *monitoring interval* or *idle timeout* of forwarding rules installation. If the administrator chooses to change the monitoring interval, the process goes from the administrator interaction to the *Interactive GUI* that changes this value and starts to operate with the new one. On the other hand, if the administrator chooses to alter the forwarding rule *idle timeout* configuration, the *Interactive GUI* receives the parameter value and sends it to the controller to change the current value. The entire process is illustrated on Figure 4.3 as a sequence diagram.

Figure 4.7 – Sequence diagram of prototype configuration process



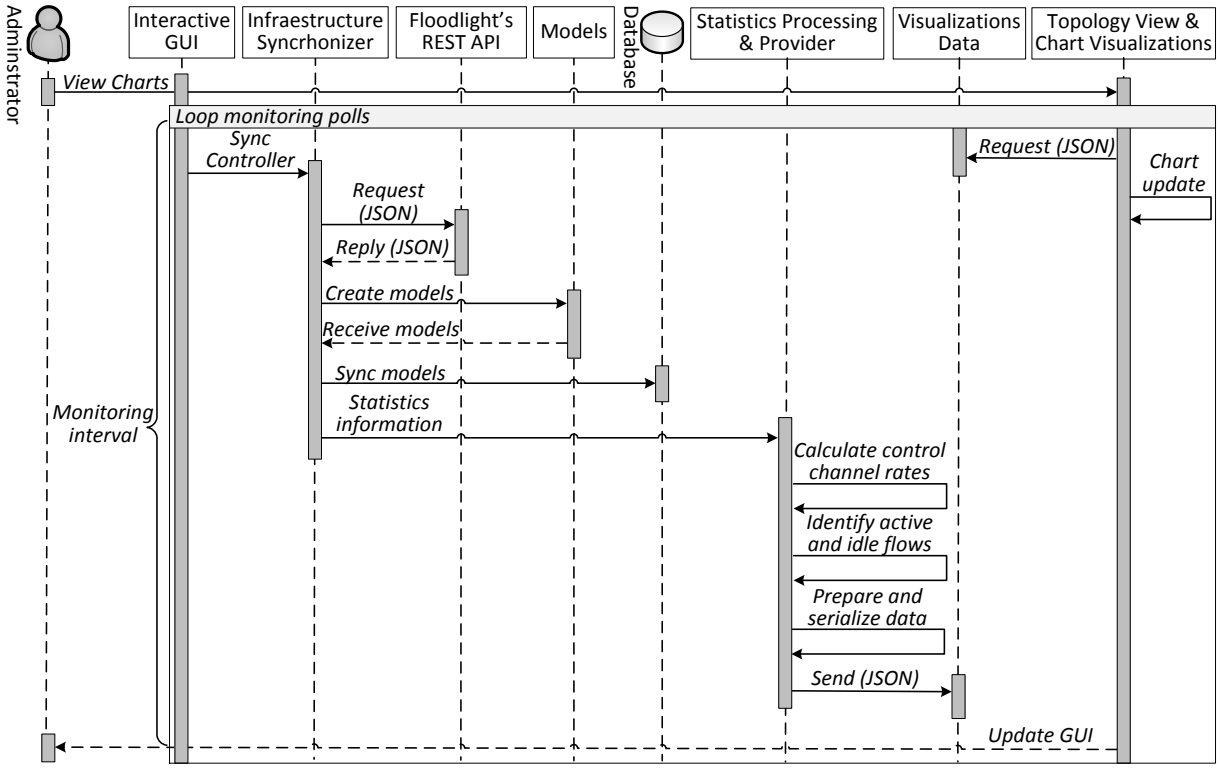
Source: by author (2015).

The administrator sends a request for new configuration through the *Interactive GUI* until reach *Floodlight Adapter* module to serialize the configuration data. Then, a new request is generated to the *Floodlight's REST API* in order to send the new configuration parameters to the controller. This API receives the request and sends these parameters values to inside modules that deal with the rest of configuration process already mentioned on Figure 4.3. When the controller finishes applying all changes requested, the *Floodlight's REST API* sends the updated configuration to the *Floodlight Adapter* module that replies this configuration in a JSON format to the *Interactive GUI*. Last, the *Interactive GUI* is updated to show the newest configuration parameters to the administrator.

4.4.2 Syncing Process

The syncing process is the most sophisticated and costly process of our prototype. The syncing process involves: requesting updated statistics about the control channel, sync all Django models into database, calculate necessary bit rates and packets processed, identify active and idle rules to provide this information to interactive visualizations. In order to better explain this entire process, Figure 4.8 presents step-by-step of the syncing process in a sequence diagram.

Figure 4.8 – Sequence diagram of prototype syncing process



Source: by author (2015).

First, the administrator accesses the *Interactive GUI* to view charts in real time. At this time, the *SDN Interactive Manager* prototype initiates the monitoring poll loop on a controller.

The first action is a request sent by the *Interactive GUI* to sync a selected controller information through *Infrastructure Synchronizer* module. The *Infrastructure Synchronizer* module requests the information about control channel statistics, topology, switches, hosts, links, ports, tables, and controller memory usage to the *Floodlight's REST API*. This API sends a reply with all data necessary to *Infrastructure Synchronizer* module that initiates to sync every Django model into a database. Firstly are created all models and then they are synchronized. Then, the statistics information is sent to the *Statistics Processing & Provider* module.

After the *Statistics Processing & Provider* module receives the statistics information, control channel rates and packets processed are calculated, active and idle flows are identified, and this information calculated is serialized and sent to the *Visualizations Data* interface. This interface is used by the *Topology View & Chart Visualizations* module to build and update Javascript-based graphs made using D3.js library, CSS, Bootstap, and jQuery. Then, graphs are updated and presented to the administrator through the *Interactive GUI* again. In the next chapter is presented the evaluation of our approach made through several experiments performed using our developed prototype.

5 CASE-STUDY, RESULTS AND EVALUATION

In this chapter is presented an evaluation – as a proof of concept – of one case-study in order to simulate administrator interactions with an OpenFlow-based SDN. Based on our prototype interactive visualizations, our objective is to demonstrate the administrator improving SDN-related parameters configurations according to his/her necessity. Section 5.1 presents the evaluation based on interactive charts that present control channel load and resource usage metrics and Section 5.2 presents the same evaluation but using the topology chart visualization.

5.1 Management Through Control Channel Load and Resource Usage Interactive Charts

In this section it is described the evaluation of our approach using our developed prototype through interactive charts. We performed this evaluation using the same campus network of our *Control Channel Analysis* made in Chapter 3. The detailed campus topology and workload was presented in Subsec 3.5.1 and the user traffic profile in Section 3.2. Our goal is to measure control channel load and resource usage considering the administrator interactions over the experiment time span. To understand the impact of changing SDN-related parameters, we simulated some administrator interactions to control the controller behavior presented in Section 2.3. In addition to varying the forwarding rule *idle timeout* configuration, we also change the monitoring polling interval to understand the impact of *Read-State* messages as well. Table 5.1 presents all configurations changes simulated during the evaluation period.

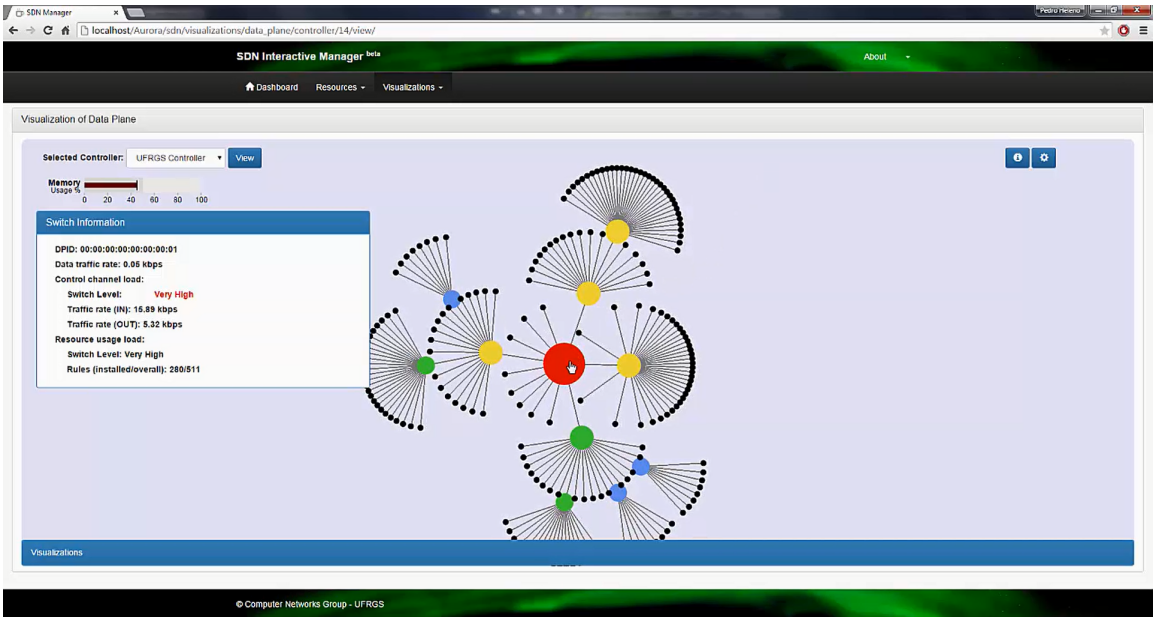
Table 5.1 – Simulated administrator interactions

Parameter	**Value**					
Reconfiguration time (hour:minutes)	11:05	11:12	11:25	11:37	11:47	11:57
Rule idle timeout (seconds)	5	60	30	30	30	30
Monitoring interval (seconds)	5	5	5	40	30	15

Source: by author (2015).

Figure 5.1 depicts the user-friendly Web interface (*SDN Interactive Manager GUI*) developed in order to provide the administrator with interactive visualizations and to allow easy configuration of SDN-related parameters. The visualization of the physical topology allows to alternate between two different perspectives (available on the top-right corner): control traffic and data traffic. Depending on the active perspective, different colors of switches are used to represent different levels of control or data traffic. Regarding resource usage, due to switches TCAM usage vary depending on flow granularity and switch particular specifications, our prototype implementation used the percentage number of flows installed on each switch related to the network overall to determine the level of their sizes. The other two buttons on the top-right corner are related to: view information about nodes used on physical topology visualization and view configuration options available to change through our prototype implementation.

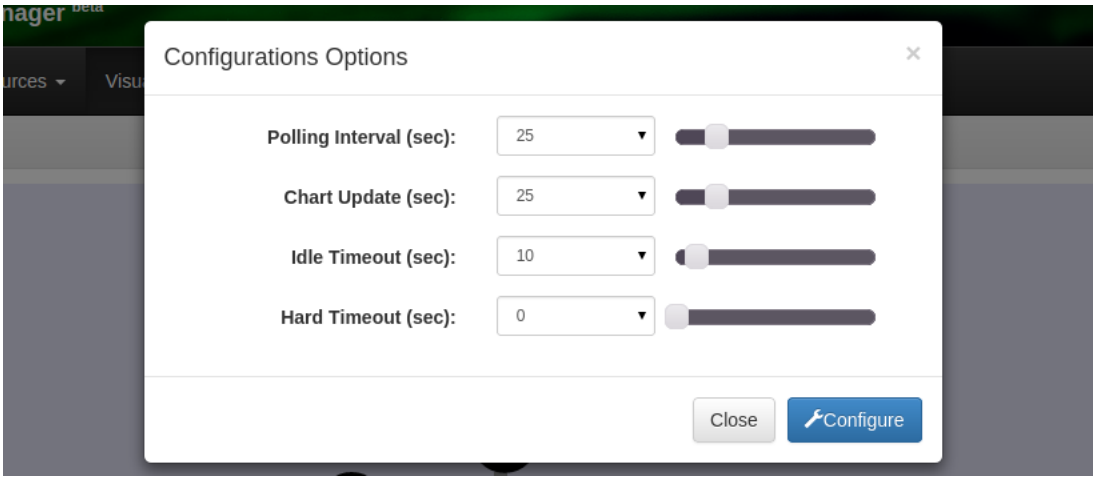
Figure 5.1 – Web interface of the prototype developed.



Source: by author (2015).

Triggering the top-right corner button (configuration change button) of *SDN Interative GUI*, a dialog window is shown with four configuration parameters that can be adjusted: *polling interval* for monitoring, *chart update interval* for charts refreshing, *idle* and *hard timeouts* of forwarding rules (see Figure 5.2). When the administrator changes the *polling interval* for monitoring or the *chart update interval* on the *SDN Interactive GUI*, our prototype adjusts the interval of statistics request, or charts refreshing respectively on *Infrastructure Synchronizer* module in order to set the newer values for these intervals and starts to operate with them. On the other hand, when the administrator changes the rule *idle* or *hard timeouts*, the chosen values are sent to the controller through the RESTful API.

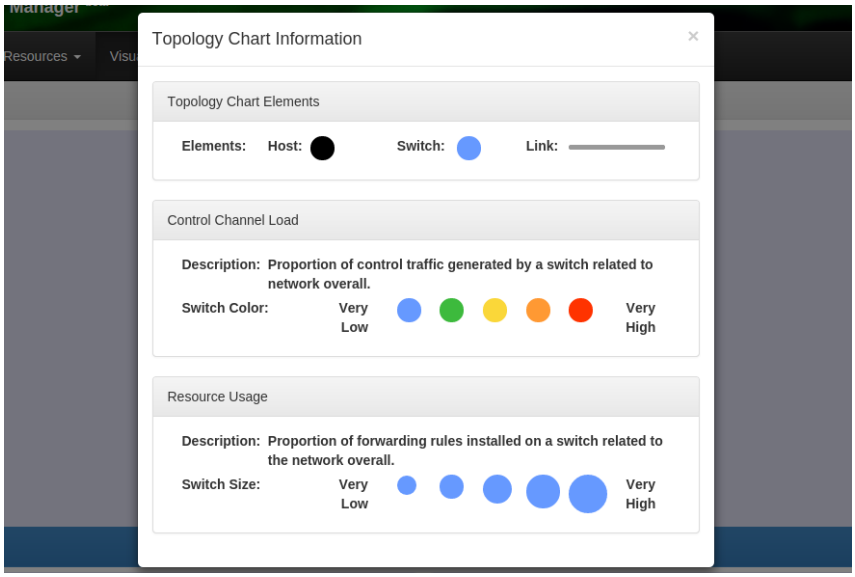
Figure 5.2 – Web interface dialog window to perform configuration changes.



Source: by author (2015).

To explain the different representations of the physical topology visualization such as node color, size and level, our *Interactive GUI* has another dialog window in order to provide it to the administrator. In the same top-right buttons group, the administrator can trigger the *info button* to show a dialog window that has the topology chart elements representation with their respective descriptions. Moreover, in the same window are shown control channel load and resource usage metrics level representation. These levels are calculated for each switch and they are classified by the proportion of control traffic generated or resource usage of each switch related to network overall. Figure 5.3 presents the information dialog window.

Figure 5.3 – Dialog window to physical topology elements descriptions.



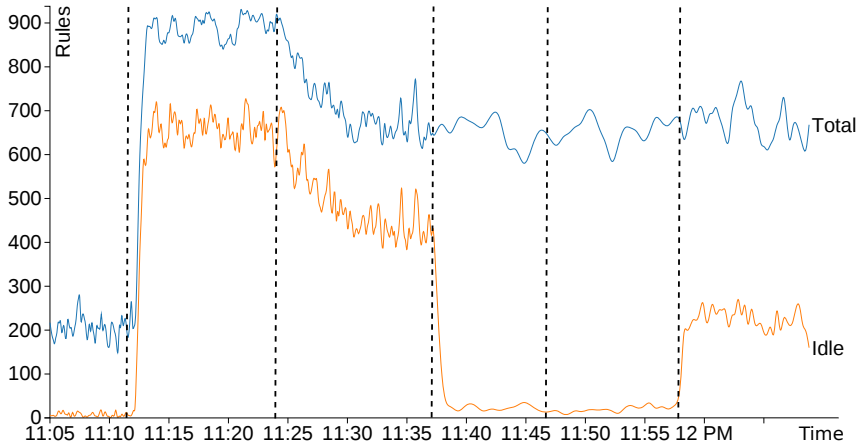
Source: by author (2015).

The percentage range to each metric level (resource usage or control traffic levels) was defined as follows: 0% to 5% as *Very Low*, more than 5% to 10% as *Low*, more than 10% to 15% as *Medium*, more than 15% to 20% as *High*, and more than 20% as *Very High* level. For example, a switch that generates 17% of control channel load related to the network overall is classified as a switch that generates a *Medium* control traffic on the network (between 15% and 20% of traffic on the control channel). On the other hand, this switch has 54 rules installed on its flow table. This 54 rules represent 48.2% of resource consumption related to overall network (54/112 rules). Thus, the switch size will be correspondent of a switch with more than 40% and less than 60%, *i.e.* a switch with a *Medium* resource usage compared to the entire network.

Below the physical topology visualization (Figure 5.1), there are interactive charts showing online resource usage in terms of installed rules (active and idle), aggregated control traffic and packet processing rates related to the whole network. Figures 5.4 to 5.6 detail interactive charts available from the *Interactive GUI* of our prototype related to resource usage and control channel load respectively. These charts present the total and idle rules, control traffic rates in kbps from the controller to switches, and control packets processed per second also in both

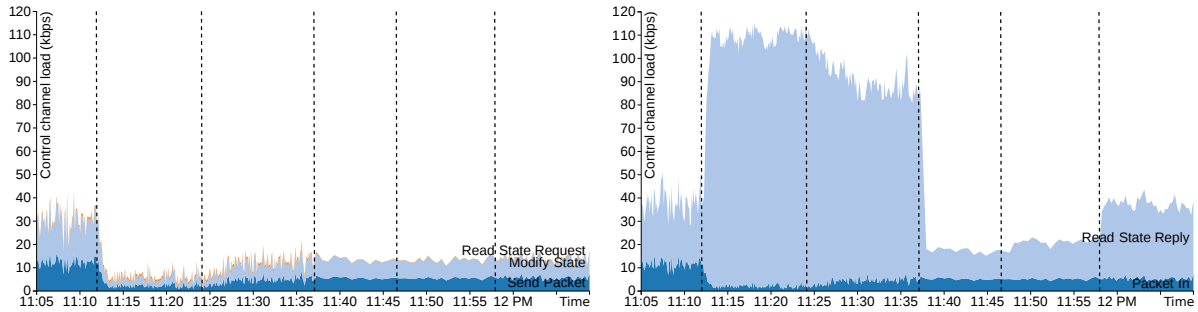
directions. Moreover, they show information for the whole network during the time span of the experiment (1 hour). Vertical dashed lines mark the moments when the administrator changes a configuration (see Table 5.1).

Figure 5.4 – Total and idle rules behavior during the experiment duration



Source: by author (2015).

Figure 5.5 – Control channel traffic rates over the experiment duration

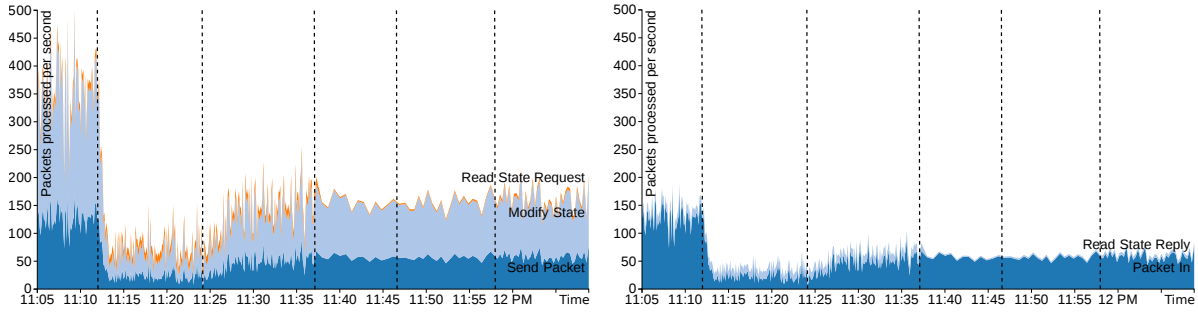


(a) Download control traffic rate over the experiment duration

(b) Upload control traffic rate over the experiment duration

Source: by author (2015).

Figure 5.6 – Control channel packet rates over the experiment duration



(a) Download packet rate over the experiment duration

(b) Upload packet rate over the experiment duration

Source: by author (2015).

At the beginning of the experiment, the amount of installed rules is approximately 200, while nearly zero are idle, as shown in Figure 5.4. As mentioned before, a rule is considered idle when its counters do not change between two monitoring polls. This small number of idle rules is a consequence of the low rule *idle timeout* value set to 5 seconds (default Floodlight configuration). On the other hand, the controller is processing a large number of *Packet-In* messages, as displayed in Figure 5.6 (b). To reduce the load on the controller, at 11:12, the administrator changes the rule *idle timeout* to 60 seconds. After this change, it is possible to visualize a dramatic decrease in control packets processed both by the controller and network devices. However, this configuration also affects immediately resource consumption, especially in terms of idle rules (Figure 5.4) and control traffic rate towards the controller (Figure 5.5 (b)). With this rule *idle timeout* configuration the interactive graph visualization of network rules (Figure 5.4) and upload traffic rate (Figure 5.5 (b)) presents that nearly 77.7% of all forwarding rules remain idle and upload control traffic increases almost threefold.

Close to 11:25, the administrator decreases the *idle timeout* to 30 seconds as an attempt to bring down traffic in the control channel and the amount of rules installed. From Figures 5.4 and 5.5 (b) it is possible to visualize such configuration does indeed decrease these values over time. Nevertheless, instead of observing a hard drop in traffic rate and installed rules, this time we notice a gradual decrease. This behavior can be explained because the new *idle timeout* settings will only be applied to newly installed rules. Thus, rules installed before the change will respect the previous configuration. Also, packet processing rates tend to increase again with this configuration, but nowhere near the values from the beginning of the experiment.

After the first two changes, the amount of traffic in the control channel towards the controller remains very high (Figure 5.5 (b)). Most of this traffic is due to *Read-State* reply messages, which are loaded with counters for as many rules as are installed in all switches. To decrease the amount of *Read-State* messages, the administrator increases the monitoring polling interval to 40 seconds at roughly 11:37. Immediately after this change, we can visualize that the control traffic rate generated by these messages is significantly reduced. However, looking at Figure 5.4, we are also able to notice an unexpected decrease in the amount of idle rules installed. This behavior is actually a distortion or loss of precision in monitoring given the way of identifying for idle rules. To be considered idle a rule needs to be monitored twice without changing its counters, which will rarely happen with too sparse monitoring polls. Finally, the administrator decreases the polling interval to 30 seconds (at 11:47), which is still insufficient to capture idle rules, and to 15 seconds (at 11:57), when the monitoring process returns to identify idle rules.

5.2 Management Through Physical Topology Visualization

In this section is presented another evaluation of our approach through our prototype. At this time, the administrator made his decisions by configuring SDN-parameters based on physical topology visualization issues. We simulated some administrator interactions based on this

interactive visualization provided by our prototype implementation. Table 5.2 presents all configurations changes simulated during the evaluation period.

Table 5.2 – Simulated administrator interactions

Parameter	**Value**			
Reconfiguration time (hour:minutes)	10:19	10:24	10:36	10:44
Rule idle timeout (seconds)	5	60	30	30
Monitoring interval (seconds)	25	25	25	60

Source: by author (2015).

At the beginning of the experiment, we applied 5 seconds to rule idle timeout configuration – *Forwarding* behavior default configuration – and 25 seconds to the monitoring polling interval. Figure 5.7 presents a screen shot of our prototype GUI with the current configuration.

Figure 5.7 – Web interface the prototype developed at the first configuration period.

Source: by author (2015).

Based on the physical topology visualization indicators (*e.g.*, switch sizes and colors), the administrator identifies that one of the network switches is bigger than the others in addition to its orange color. By positioning the cursor over this switch, the administrator can observe that this switch has a *high* level of control channel load and resource usage. It is important to emphasize that the switch size represents the resource usage level in terms of number or rules installed and the switch color represents the control channel load level both compared to the network overall. This switch with *Datapath ID* (DPID) 00:00:00:00:00:00:00:01 (shown in Figure 5.7) connects the Web and Video servers with all other network devices and due to the lower idle timeout value configured at the controller, this switch is generating many packets processed per second and control traffic on both directions.

If the controller cannot handle with rate of 20 packets to be processed per second on controller's direction, i.e., the controller cannot process all *Packet-In* and *Read-State* reply messages that arrives to him due to processing limitations, this controller can become a network bottleneck. Because of this fact, the administrator needs to act over configuration parameters to mitigate this bottleneck. One possible configuration is to increase the idle timeout value to a higher value, for example to 60 seconds, in order to decrease the rate of packets processed per second generated on both directions. This configuration chosen the inactive rules installed for a longer time in flow table. Figure 5.8 shows a screen shot of our prototype GUI some minutes after the administrator applies the newer configuration value to idle timeout configuration.

Figure 5.8 – Web interface the prototype developed at the second configuration period.

Source: by author (2015).

At this moment, the rate decreases from approximately 20 to 10 packets processed per second that are sent from switches to controller and from about 80 to 60 packets processed per second sent from controller to switches in overall. This occurs because fewer rules are installed due to more rules that are reactivated by the user traffic profile of most users. On the other hand, the drawback of this configuration is the growth of the number of rules installed on switches.

Supposing that the maximum limit of rules that can be installed on the switch's flow table is known – without exceeding TCAM available – and because the current number of rules installed is close to the limit it indicates that the administrator needs to reconfigure some network parameter to prevent running out of resources. Assuming that the limit is around about 400 rules and the central switch has almost 350, the administrator decides to act over idle timeout value again to decrease the amount of rules installed. Thus, the administrator decides to change the idle timeout configuration to an intermediate value (30 seconds) and wait a while until old flows be removed by inactive to receive the newer configuration. Figure 5.9 presents a screen shot of our prototype GUI after few minutes of experiment.

Figure 5.9 – Web interface the prototype developed at the third configuration period.

Source: by author (2015).

After this change, the administrator verifies that the amount of rules installed on that switch is under control (about almost 300 rules installed in the central switch). On the other hand, the amount of traffic generated by monitoring process on controller direction is unnecessarily high (*Read-State* reply messages) with peaks of almost 100 kbps. Therefore, the administrator decides to increase the polling interval value in order to decrease the rate of control traffic generated on controller's direction. Figure 5.10 presents a screen shot of our prototype GUI after few minutes of the last configuration performed.

Figure 5.10 – Web interface the prototype developed at the fourth configuration period.

Source: by author (2015).

When the prototype applies the newer value to the polling interval, the control traffic rate of messages that goes to the controller decreases considerably, from approximately 100 to 60 kbps, adjusting the rate of control traffic is managed according to the administrator needs. Therefore, our prototype visualizations have enabled the administrator to manage control channel load and resource usage regarding control traffic generated, packets processed per second, and number of rules installed on network devices. Our evaluation showed that the physical topology visualization indicators help the administrator to prevent running out of resources and may avoid bottlenecks at the controller by a simple change in a configuration parameter at the controller (*e.g.*, idle timeout configuration).

6 CONCLUSION

Although network monitoring, visualization, and configuration are considered common management activities, in the context of SDN, they can be considerably different from traditional networks and deserve proper attention. For example, the SDN controller behavior can be customized by network administrators and these customizations might affect resource consumption and forwarding performance. As a consequence, such impact is difficult to assess because traditional network management solutions were not designed to cope in the context of SDN. Moreover, current solutions deal with SDN monitoring, visualization, or configuration activities for automating tasks (*e.g.*, reduce control traffic overhead or protect the network) and they do not aim on helping the administrator to better understand and interact with SDN.

Currently, the state-of-the-art in SDN has addressed monitoring using the OpenFlow protocol – the most relevant SDN implementation – for different purposes (*e.g.*, proposing anomaly detection systems or mechanisms to establish a balance between control channel overhead and accuracy of collected information). These investigations tend to employ monitoring information to automatically adapt the network to specific conditions (*e.g.*, switches lowest resource usage as possible). Moreover, to the best of our knowledge, there is no solution that has integrated monitoring information with interactive visualization and configuration tools for SDN. Due to this gap, we addressed an interactive approach to SDN management through monitoring, visualization, and configuration activities with the administrator interactions. We argue that with such a solution the administrator could better understand and interact with the network, significantly improving everyday tasks of SDN management.

In this context, an OpenFlow-based architecture introduces a simple way to develop and maintain communication in SDN because of its centralized logic of controlling forwarding devices. However, this simplicity may allegedly impose a high cost on the network controller and create bottlenecks at the control channel. Recent solutions attempted to alleviate these bottlenecks by distributing the control logic of OpenFlow. Nevertheless, there is no detailed study about in which situations such bottlenecks appear and whether they can or cannot be mitigated or even avoided by simple configuration, *i.e.*, without the need to develop a specific distributed controller. Based on this argumentation, we identified that is required a detailed study about the overhead imposed by communication protocol used to understand and then prevent bottlenecks at the control channel. As a consequence, we performed an analysis of control channel traffic generated in OpenFlow networks in order to understand these bottlenecks.

6.1 Main Contributions and Results Obtained

In this work, we initially presented an analysis of the control channel traffic generated in OpenFlow networks in order to verify the impact of specific SDN-related parameters and their influence in terms of resource consumption and traffic forwarding performance resulted from

the communication between controller and network devices. To understand the variations of control traffic, we decided to vary the *idle timeout* configuration that indicates when a forwarding rule can be removed due lack of activity. The results showed how this single factor configured at the controller affects both control channel load and resource usage metrics. We verified that the higher the value of *idle timeout* configuration, the larger is the number of rules installed on switches – which most of them remains inactive during the experimentation time – in addition to less control packets related to forwarding rules installation are generated on the network. On the other hand, the opposite occurs when the *idle timeout* is configured to lower values, *i.e.*, a higher number of packets processed per second are generated and fewer rules remain installed on network devices.

Next, based on the results that we obtained through our control channel analysis, we verified the necessity to correctly adapt SDN-related parameters according to the network demand and resource available (*e.g.*, *idle timeout* configuration of forwarding rules according to user traffic profile and switch's TCAM available on forwarding devices). Due to this necessity, we have proposed an interactive approach to SDN management that integrates monitoring, visualization, and configuration activities by including the administrator in a management loop. Our approach allowed the administrator to interact with SDN and better understand and manage the control traffic generated in OpenFlow-based network.

As a proof-of-concept, we developed a prototype as a Web application that is able to:

- Perform monitoring with configurable *polling interval* in order to collect control channel statistics of an SDN controller in addition to sync and store its data into a database (FR-01 and FR-02 presented in Section 4.4).
- Identify active/idle rules (FR-03), calculate control traffic and packets processed per second rates (FR-04), aggregate data per host, switch, or even the entire network (FR-03), present interactive visualizations of physical topology (FR-05, FR-09), and control traffic metrics (FR-06, FR-07).
- Support the administrator interaction by configuring/reconfiguring SDN-related parameters as the *idle timeout* of forwarding rules installation and the monitoring *polling interval* of the monitoring process (FR-08).

In order to integrate our prototype development with an SDN controller, we needed to modify the standard implementation of the Floodlight controller aiming to enhance its statistics collector by adding the control channel messages counters (FR-10).

In our approach evaluation, we simulated several administrator interactions by changing parameter values in order to adjust rates of control traffic and packets processed per second – generated in both directions – in addition to control the amount of rules installed on network devices. We simulated variations for the *idle timeout* configuration for forwarding rules installation and the monitoring *polling interval* for monitoring requests in order to understand the monitoring process impact as well. By analyzing interactive visualizations provided by our pro-

totype, the administrator was able to adjust the proportion of control channel traffic and packets generated per second on the network in addition to resource usage according to his needs. In other words, the administrator was able to identify potential issues and change configurations of SDN parameters using our interactive Web interface to achieve his specific goals.

6.2 Final Remarks and Future Work

In future investigations, we plan to perform more experiments with other SDN-related parameters and different controller implementations. We also plan to perform experiments with other versions of the OpenFlow protocol, which have different control messages and data structures. Moreover, we intend to extend our configuration possibilities developed on the prototype, such as thresholds to an algorithm that can perform automated reconfigurations on behalf of the administrator. We also intend to identify how a forwarding rule installed by the *Forwarding* behavior affects TCAM in addition to compare the value monitored with the maximum available on switches. Thus, we will be able to adapt our prototype to warn the administrator when the current value is close to the maximum supported by switches specifications.

We also intend to improve some technical details of our prototype such as decoupling the monitoring process from the interactive GUI in order to perform this activity independently of visualizing the network. Moreover, we intend to store historical data from previous configurations changes made by the administrator that may help on making future decisions. Furthermore, we also intend to create other visualizations such as logical topology view that should present all network devices and their control channel connection to the controller. Regarding our controller implementations, we also intend to improve all code developed to support reporting control channel statistics from Floodlight v0.90 to the current controller version. Finally, after all modifications on the prototype and controller, we intend to provide both projects as OpenSource applications available on GitHub[1].

[1] https://github.com/

REFERENCES

3GPP. CDMA2000 Evaluation Methodology. *Open Networking Foundation*, Arlington, USA, v.1, n. 1, p. 1–182, dec. 2004. Available at: <http://www.3gpp2.org/Public_html/specs/>. Accessed: april. 13. 2015.

ARGYROPOULOS, C. et al. PaFloMon – A Slice Aware Passive Flow Monitoring Framework for OpenFlow Enabled Experimental Facilities. In: 1ST EUROPEAN WORKSHOP ON SOFTWARE DEFINED NETWORKING (EWSDN), 2012, Darmstadt, Germany. **Proceedings...** Darmstadt: IEEE, 2012. p. 97–102.

BARBOSA, P. T.; GRANVILLE, L. Z. Interactive SNMP Traffic Analysis Through Information Visualization. In: 12TH IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2010, Osaka, Japan. **Proceedings...** Osaka: IEEE, 2010. p. 73–79.

BONDAN, L. et al. Kitsune: A Management System for Cognitive Radio Networks Based on Spectrum Sensing. In: 14TH IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2014, Krakow, Poland. **Proceedings...** Krakow: IEEE, 2014. p. 1–9.

CHENG, X.; LIU, J.; DALE, C. Understanding the Characteristics of Internet Short Video Sharing: A YouTube-Based Measurement Study. **IEEE Transactions on Multimedia**, New York, v. 15, n. 5, p. 1184–1194, aug. 2013.

CHOWDHURY, S. R. et al. PayLess: A Low Cost Netowrk Monitoring Framework for Software Defined Networks. In: 14TH IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2014, Krakow, Poland. **Proceedings...** Krakow: IEEE, 2014. p. 1–9.

CURTIS, A. R. et al. DevoFlow: Scaling Flow Management for High-performance Networks. **SIGCOMM Computer Communication**, New York, v.41, n. 4, p. 254–265, aug. 2011.

FOUNDATION, O. N. Software-Defined Networking: The New Norm for Networks. **SDN Architecture 1.0**, California, USA, v.1, n. 1, p. 1–12, april. 2012. Available at: <https://www.opennetworking.org/sdn-resources/sdn-definition>. Accessed: april. 13. 2015.

FOUNDATION, O. N. SDN Architecture 1.0. **Project Architecture & Framework**, California, USA, v.1, n. 1, p. 1–68, june. 2014. Available at: <https://www.opennetworking.org/sdn-resources/sdn-library/technical-papers>. Accessed: april. 13. 2015.

GUIMARAES, V. T. et al. A Collaborative Solution for SNMP Traces Visualization. In: 29TH INTERNATIONAL CONFERENCE ON INFORMATION NETWORKING (ICOIN), 2014, Punket, Thailand. **Proceedings...** Punket: IEEE, 2014. p. 458–463.

JOSE, L.; YU, M.; REXFORD, J. Online Measurement of Large Traffic Aggregates on Commodity Switches. In: 11TH WORKSHOP ON HOT TOPICS IN MANAGEMENT OF INTERNET, CLOUD, AND ENTERPRISE NETWORKS AND SERVICES (USENIX), 2011, Boston, MA. **Proceedings...** New York: ACM, 2011. p. 1–6.

KATSAROS, K. V.; XYLOMENOS, G.; POLYZOS, G. C. GlobeTraff: A Traffic Workload Generator for the Performance Evaluation of Future Internet Architectures. In: 5TH

INTERNATIONAL CONFERENCE ON NEW TECHNOLOGIES, MOBILITY AND SECURITY (NTMS), 2012, Istambul, Turkey. **Proceedings...** Istambul: IEEE, 2012. p. 1–5.

KIM, H.; FEAMSTER, N. Improving Network Management with Software Defined Networking. **IEEE Communications Magazine**, New York, v.51, n. 2, p. 114–119, feb. 2013.

KLEIN, D.; JARSCHEL, M. An OpenFlow Extension for the OMNeT++ INET Framework. In: 6TH INTERNATIONAL ICST CONFERENCE ON SIMULATION TOOLS AND TECHNIQUES, 2013, Cannes, France. **Proceedings...** Brussels, Belgium: ICST, 2013. p. 322–329.

KREUTZ, D. et al. Software-Defined Networking: A Comprehensive Survey. **Proceedings of the IEEE**, New York, v.103, n. 1, p. 14–76, jan. 2015.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In: 9TH ACM SIGCOMM WORKSHOP ON HOT TOPICS IN NETWORKS, 2010, Monterey, California. **Proceedings...** New York, USA: ACM, 2010. p. 1–6.

MACHADO, C. C. et al. Towards SLA Policy Refinement for QoS Management in Software-Defined Networking. In: 28TH IEEE INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION NETWORKING AND APPLICATIONS (AINA), 2014, Victoria, CA. **Proceedings...** Victoria: IEEE, 2014. p. 397–404.

MATTOS, D. et al. OMNI: OpenFlow MaNagement Infrastructure. In: 2ND INTERNATIONAL CONFERENCE ON THE NETWORK OF THE FUTURE (NOF), 2011, Paris, France. **Proceedings...** Paris: IEEE, 2011. p. 52–56.

MCKEOWN, N. et al. OpenFlow: Enabling Innovation in Campus Networks. **SIGCOMM Computer Communincation**, New York, v.38, n. 2, p. 69–74, mar. 2008.

PFAFF, B. et al. OpenFlow Switch Specification - Version 1.0.0 (Wire Protocol 0x01). **Open Networking Foundation**, New York, USA, v.1, n. 1, p. 1–42, dec. 2009. Available at: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf>. Accessed: april. 13. 2015.

ROTHENBERG, C. et al. When Open Source Meets Network Control Planes. **Computer**, New York, v.47, n. 11, p. 46–54, nov. 2014.

SALVADOR, E.; GRANVILLE, L. Using Visualization Techniques for SNMP Traffic Analyses. In: 13TH IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS (ISCC), 2008, Marrakech, MA. **Proceedings...** Marrakech: IEEE, 2008. p. 806–811.

SYSTEMS, C. Cisco Visual Networking Index: Forecast and Methodology, 2013–2018. **Cisco Systems**, San Jose, USA, p. 1–14, june 2014. Available at: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf. Acessed: april 15. 2015.

TOOTOONCHIAN, A.; GHOBADI, M.; GANJALI, Y. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In: 11TH INTERNATIONAL CONFERENCE ON PASSIVE AND ACTIVE MEASUREMENT (PAM), 2010, Zurich, Switzerland. **Proceedings...** Berlin, Heidelberg: Springer-Verlag, 2010. p. 201–210.

WICKBOLDT, J. et al. Software-Defined Networking: Management Requirements and Challenges. **IEEE Communications Magazine - Network & Service Management Series**, New York, v.53, n. 1, p. 278–285, jan. 2015.

WICKBOLDT, J. A. et al. Resource management in IaaS cloud platforms made flexible through programmability. **Computer Networks**, United Kingdom, v.68, p. 54–70, aug. 2014.

XIE, G. et al. ReSurf: Reconstructing Web-Surfing Activity from Network Traffic. In: IFIP NETWORKING CONFERENCE, 2013, New York, USA. **Proceedings...** New York: IEEE, 2013. p. 1–9.

YEGANEH, S. H.; GANJALI, Y. Turning the Tortoise to the Hare: An Alternative Perspective on Event Handling in SDN. In: 1ST INTERNATIONAL WORKSHOP ON SOFTWARE-DEFINED ECOSYSTEMS (BIGSYSTEM 2014), 2014, Vancouver, Canada. **Proceedings...** New York: ACM, 2014. p. 29–32.

YU, C. et al. FlowSense: Monitoring Network Utilization with Zero Measurement Cost. In: 14TH INTERNATIONAL CONFERENCE ON PASSIVE AND ACTIVE MEASUREMENT (PAM), 2013, Hong Kong, China. **Proceedings...** Berlin, Heidelberg: Springer-Verlag, 2013. p. 31–41.

YU, M.; JOSE, L.; MIAO, R. Software Defined Traffic Measurement with OpenSketch. In: 10TH USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION, 2013, Lombard, IL. **Proceedings...** Lombard, IL: USENIX, 2013. p. 29–42.

YU, M. et al. Scalable flow-based networking with DIFANE. **SIGCOMM Computer Communication**, New York, v.4, n. 4, p. 351–362, aug. 2010.

ZHANG, Y. An Adaptive Flow Counting Method for Anomaly Detection in SDN. In: 9TH ACM CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES (CONEXT), 2013, Santa Barbara, USA. **Proceedings...** New York, USA: ACM, 2013. p. 25–30.

AppendixA PUBLISHED PAPER – WGRS 2014

In this paper it was published an analysis of control traffic generated in OpenFlow-based SDN. We focus on explain how the controller behavior and configuration can affect the control channel. This work was motivated because several authors argue that occurs bottlenecks of control traffic, in a context with a centralized controller, but do not address in what proportion this load may affect the control channel availability.

- **Title –**
  Uma Análise do Tráfego de Controle em Redes OpenFlow
- **Conference –**
  XIX Workshop de Gerência e Operação de Redes e Serviços (WGRS - 2014)
- **Type –**
  Main track (full-paper)
- **Qualis –**
  B5
- **URL:**
  <http://sbrc2014.ufsc.br/anais/files/wgrs/ST2-3.pdf>
- **Date –**
  May 5-9, 2014
- **Held at –**
  Florianópolis - SC, Brasil

Uma Análise Quantitativa do Tráfego de Controle em Redes OpenFlow

Pedro Heleno Isolani, Juliano Araujo Wickboldt, Cristiano Bonato Both, Juergen Rochol, Lisandro Zambenedetti Granville

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{phisolani, jwickboldt, cbboth, juergen, granville}@inf.ufrgs.br

Resumo. *Software-Defined Networking é um paradigma de redes que permite facilmente projetar, desenvolver e implantar inovações de rede, pois fornece agilidade e flexibilidade na incorporação de novos serviços e tecnologias. As redes baseadas nesse paradigma ganharam destaque a partir da especificação do protocolo OpenFlow, que define uma simples interface de programação para controlar dispositivos de encaminhamento a partir de um controlador. Apesar da rápida disseminação desse protocolo, os trabalhos relacionados sobre OpenFlow não analisam em profundidade os reais impactos das mensagens de controle e monitoramento gerado por esse protocolo. Desta forma, a principal contribuição deste artigo é uma análise quantitativa do tráfego de controle e monitoramento em redes OpenFlow. Os resultados revelam que variações do tempo limite de ociosidade das regras de encaminhamento, da frequência de monitoramento e da topologia da rede, impactam na taxa de transferência e na quantidade de mensagens geradas no canal de controle.*

1. Introdução

Software-Defined Networking (SDN) é um paradigma de redes baseado em três aspectos fundamentais: (*i*) os planos de controle e encaminhamento são claramente desacoplados, (*ii*) a lógica da rede é abstraída de uma implementação em hardware para uma camada de software programável e (*iii*) é introduzido um elemento controlador de rede que coordena as decisões de encaminhamento dos demais dispositivos [Kim e Feamster 2013, Tootoonchian e Ganjali 2010]. A utilização desses três aspectos de forma integrada, permite que inovações de redes possam ser mais facilmente projetadas, desenvolvidas e implantadas, pois possibilita a agilidade e flexibilidade na incorporação de novos serviços e tecnologias, sem que os fabricantes precisem expor o funcionamento interno de seus equipamentos [McKeown *et al.* 2008].

As redes baseadas no paradigma SDN ganharam considerável destaque a partir da especificação do protocolo OpenFlow, que define uma simples interface de programação para controlar dispositivos de encaminhamento a partir de um controlador. Desta forma, a lógica da rede é concentrada no controlador que troca mensagens para o estabelecimento de conexões, monitoramento de estatísticas, manutenção e configuração do comportamento dos dispositivos da rede [Bari *et al.* 2013]. Sendo assim, o gerenciamento de rede baseadas na especificação OpenFlow reduz, ou até mesmo elimina, problemas de gerenciamento de redes tradicionais intrinsecamente [Kim e Feamster 2013]. Por exemplo, tarefas como a descoberta de rede são resolvidas simplesmente pelo fato de que os dispositivos precisam ser registrados no controlador para pertencerem à rede efetivamente.

Devido a abordagem centralizada da lógica da rede, utilizada pelo protocolo OpenFlow, muito tem sido discutido na literatura especializada acerca do posicionamento e proporção de controladores em contraponto aos dispositivos de encaminhamento. Alternativas para distribuir a lógica da rede sobre os dispositivos de encaminhamento são desenvolvidas visando evitar um possível gargalo de mensagens de controle no controlador [Roy *et al.* 2014, Curtis *et al.* 2011, Yu *et al.* 2010]. Entretanto, não são analisados em profundidade os reais impactos das mensagens de controle e monitoramento gerado pelo protocolo OpenFlow. A especificação desse protocolo apenas define quais e como são as mensagens de controle (*e.g.* instalação de regras, coleta de estatísticas), mas não especifica como essas mensagens devem ser utilizadas para controlar e monitorar a rede sem comprometer seu desempenho. Assim, as informações referentes a frequência em que podem ser realizados o controle e monitoramento de estatísticas dos dispositivos da rede não são especificadas. Desta forma, se torna fundamental a realização de uma análise para identificar quais mensagens de controle e monitoramento mais impactam na sobrecarga do canal de controle em uma rede baseada em SDN e OpenFlow.

A principal contribuição deste artigo é uma análise quantitativa do tráfego de controle e monitoramento em redes OpenFlow. Essa análise é extremamente importante, pois fornece subsídios para mensurar o uso efetivo do canal de controle e, a partir disso, melhor compreender e gerenciar o impacto real da utilização do protocolo OpenFlow. Essa compreensão é fundamental para o projeto e desenvolvimento de novos sistemas de gerenciamento de redes baseadas no paradigma SDN. Para a obtenção dos resultados, foi realizada uma experimentação considerando aspectos de instalação de regras, além da obtenção de estatísticas através do monitoramento do controlador Floodlight [Big Switch Networks 2014]. O ambiente experimental foi emulado no Mininet [Lantz *et al.* 2010], simulando o tráfego de *streamming* de vídeo e requisições de páginas Web em duas diferentes topologias de rede, estrela e árvore. Os resultados apresentam como a frequência de monitoramento, as variações das topologias e configurações do controlador impactam na taxa de transferência e na quantidade de mensagens geradas no canal de controle.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica sobre SDN bem como sobre o protocolo OpenFlow. Na Seção 3 é descrito o cenário de experimentação e a metodologia de avaliação seguida como prova de conceito. Na Seção 4 são discutidas e analisadas as informações abstraídas da modelagem analítica e dos resultados da experimentação. Por fim, na Seção 5 são apresentadas as conclusões e as perspectivas para trabalhos futuros.

2. Fundamentação Teórica

Nesta Seção são abordados os elementos que definem os principais conceitos de SDN e OpenFlow, assim como é apresentada uma breve discussão da literatura em SDN. Na Subseção 2.1 são descritos as principais entidades presentes em SDN e OpenFlow considerando a versão 1.0 do protocolo. Posteriormente, na Subseção 2.2 são abordados os trabalhos relacionados sobre gerenciamento e, principalmente, distribuição do plano de controle em SDN.

2.1. *Software-Defined Networking e OpenFlow*

SDN introduz uma perspectiva flexível para programar e manter a operacionalidade da rede. Em SDN, existe uma clara separação entre os planos de controle e encaminha-

mento. Além disso, a lógica é abstraída dos dispositivos de encaminhamento para um ou mais elementos controladores da rede [Kim e Feamster 2013, Tootoonchian e Ganjali 2010]. A arquitetura definida para SDN é dividida em camadas de aplicação, controle e encaminhamento. A comunicação entre as camadas acontece através de *Application Programming Interfaces* (APIs) de comunicação padrão. Redes que seguem o paradigma SDN proporcionam vantagens em termos de gerência e controle da rede, principalmente, pela visão global da rede e pela flexibilidade e agilidade na incorporação de novos serviços. Outro aspecto que também contribui para a larga adoção desse paradigma é a liberdade de implementação e experimentação de novos protocolos sem se ater a detalhes de implementações proprietárias dos dispositivos. SDN foi largamente utilizado após a especificação do protocolo OpenFlow [McKeown *et al.* 2008] e, devido a flexibilidade e interface simples de programação, surgiram diversas soluções tanto na academia como na indústria.

O OpenFlow é um protocolo *Open Source* que utiliza uma tabela para armazenamento de regras de encaminhamento e uma interface padronizada para adicionar e remover essas regras [McKeown *et al.* 2008]. Neste contexto, uma regra de encaminhamento é um conjunto de *matches* e *actions* instalados em um *switch* para implementar um fluxo ou parte dele, *i.e.* uma conexão lógica, normalmente, bidirecional entre duas máquinas terminais da rede. OpenFlow oferece programabilidade padronizada aos dispositivos de encaminhamento, permitindo desenvolver novos protocolos, *e.g.*, protocolos de roteamento, módulos de segurança, esquemas de endereçamento, alternativas ao protocolo IP, sem a necessidade de ser exposto os funcionamentos internos dos equipamentos. Um *switch* com suporte OpenFlow consiste basicamente em pelo menos três partes: uma tabela de fluxos, onde são associadas *actions* para cada *match*; um canal seguro, por exemplo *Secure Socket Layer* (SSL); e o protocolo OpenFlow para comunicação entre controlador e os *switches*.

O protocolo OpenFlow versão 1.0, abordado na análise deste trabalho e amplamente implementado pelos dispositivos com suporte a SDN, considera três tipos de mensagens de controle: (*i*) do controlador para o *switch*, (*ii*) assíncronas e (*iii*) simétricas. As mensagens do controlador para o *switch* são utilizadas para gerenciar o estado dos dispositivos de encaminhamento (*e.g.*, ler informações de estatísticas das tabelas de encaminhamento). As mensagens assíncronas são iniciadas pelos *switches* utilizadas para informar ao controlador sobre as modificações na rede e no estado desses dispositivos (*e.g.*, chegada de novos fluxos na rede para o qual o *switch* não possui um *match* correspondente). Por fim, as mensagens simétricas podem ser iniciadas tanto pelo controlador como pelos *switches* e são enviadas sem solicitação (*e.g.*, *echo request* e *reply* para certificar que um dispositivo da rede está ativo). O detalhamento das mensagens é apresentado na Tabela 1.

Apesar de todas as mensagens impactarem no tráfego gerado no canal de controle, as mensagens de coleta de estatísticas *Read-State* e de instalação de regras de encaminhamento *Packet-In/Out* e *Modify-State* são consideradas as mais relevantes, pois são mais frequentemente utilizadas pelo controlador e dispositivos de encaminhamento. Portanto, a partir da análise dessas mensagens, é possível identificar o impacto e definir o nível de granularidade de um possível monitoramento periódico da rede.

Tipos	Mensagens	Descrição
Controlador para o *switch*	*Features*	Enviadas para obter conhecimento sobre as capacidades dos *switches*
	Configuration	Específicas para parâmetros de configuração dos *switches*
	Modify-State	Gerenciam o estado dos *switches*, comumente utilizadas para adicionar, remover e modificar fluxos nas tabelas e alterar propriedades de portas
	Read-State	Utilizadas para coletar estatísticas sobre as tabelas, portas, fluxos e filas dos *switches*
	Send-Packet	Utilizadas pelo controlador para enviar pacotes por uma porta específica
	Barrier	Obtenção de conhecimento se as dependências das mensagens foram alcançadas ou para receber notificações sobre tarefas concluídas
Assíncrona	*Packet-In*	Enviadas ao controlador toda vez que o *switch* não tenha regra instalada para determinado pacote ou quando a regra for para enviar o pacote ao controlador
	Flow-Removed	Relativas a remoção de regras dos *switches*
	Port-Status	Obtenção de *status* das portas dos *switches*
Simétrica	*Hello*	Para início de conexão entre *switch* e controlador
	Echo	Estabelecimento de conexão entre *switch* e controlador e podem ser utilizadas para obter conhecimento de latência, banda ou conectividade
	Barrier	Utilizadas para obter conhecimento se as dependências das mensagens foram alcançadas ou para receber notificações sobre tarefas concluídas

Tabela 1. Mensagens de controle do protocolo OpenFlow versão 1.0

2.2. Trabalhos Relacionados

Existem pesquisas que investigam o problema de gargalos no canal de controle entre o controlador e os dispositivos de encaminhamento. A solução amplamente adotada é descentralizar o controle para a rápida e ágil reação a possíveis modificações no comportamento da rede, sem que seja necessário recorrer a um único elemento controlador [Tootoonchian e Ganjali 2010]. Entretanto, ainda existem discussões sobre o gerenciamento centralizado e distribuído do controle da rede, visto que, pode influenciar e comprometer a disponibilidade do canal de controle [Levin *et al.* 2012]. Devido a esse fato, a análise realizada neste trabalho foi inspirada pela ausência de informação referente ao impacto das mensagens de controle na configuração, manutenção e monitoramento dos dispositivos de encaminhamento.

A solução apresentada por DevoFlow visa manter os fluxos no plano de encaminhamento, ou seja, faz com que os *switches* encaminhem os pacotes com o mínimo de solicitações possíveis ao controlador [Curtis *et al.* 2011]. Quando uma regra não se encontra na tabela de regras de um *switch*, o comportamento padrão do OpenFlow é invocar o controlador, encapsulando o pacote e o enviando para descoberta da regra apropriada. DevoFlow evita esse processo através da delegação de controle aos *switches* na clonagem de regras, acionamento de *actions* locais e pela coleta de estatísticas. Para justificar o propósito do trabalho, foram apresentadas estimativas de sobrecargas da utilização do OpenFlow. Entretanto, o trabalho apresenta somente estimativas médias o que pode variar dependendo do tipo da rede.

Uma solução semelhante ao DevoFlow é a apresentado por DIFANE, que visa manter a lógica de controle próxima ao plano de encaminhamento, de forma a atribuir o controle aos *switches* próximos ao controlador, chamados de *switches* de autoridade [Yu *et al.* 2010]. Quando uma regra não se encontra na tabela de regras de um *switch* o comportamento é invocar um *switch* de autoridade mais próximo. Ambos os trabalhos, DevoFlow e DIFANE, apresentam informações relativas a sobrecargas geradas no controlador, mas não detalham sobre o impacto específico das mensagens trafegadas no canal de controle.

A solução de Heller *et al.* [Heller *et al.* 2012] investiga o posicionamento e proporção de controladores necessários para atender ao plano de encaminhamento sem comprometer o desempenho da rede. Para estimar o consumo de banda utilizada no canal de controle, foram estabelecidas as métricas de latência média e latência para o pior caso para comunicação entre controlador e dispositivos de encaminhamento. Entretanto, o cálculo realizado para essa estimativa é aproximado conforme a distância dos nodos em um grafo e não proporcional ao uso ou ao tipo das mensagens trafegadas no canal de controle. Nesse contexto, Bari *et al.* [Bari *et al.* 2013] propuseram uma ferramenta que adapta a quantidade de controladores necessários para determinada demanda da rede. Para realizar essa adaptação é calculado um valor aproximado do tempo mínimo para instalação de regras nos dispositivos de encaminhamento.

A análise realizada neste trabalho visa fornecer subsídios sobre o impacto do tráfego de controle e monitoramento para futuros projetos de ferramentas de gerenciamento no contexto de OpenFlow versão 1.0. Portanto, a definição da quantidade de controladores necessários, posicionamento em relação aos dispositivos de encaminhamento e granularidade de monitoramento aceitável podem ser estimadas e melhor abordadas pelos sistemas de gerenciamento. Assim, pretende-se estimar o tráfego de controle para que não comprometa o desempenho e a disponibilidade do canal de controle e, consequentemente, o desempenho da rede.

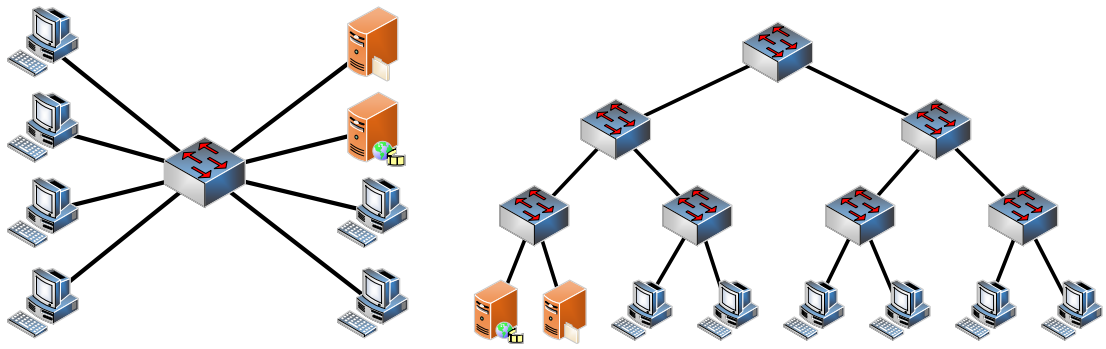
3. Cenário e Metodologia de Avaliação

Nesta Seção é apresentado em detalhes o cenário de experimentação e a metodologia utilizada para a análise quantitativa do tráfego de controle e monitoramento em redes OpenFlow versão 1.0. As características sobre o cenário e as duas topologias utilizadas na experimentação (estrela e árvore) são apresentadas na Subseção 3.1. Em seguida, na Subseção 3.2, a metodologia utilizada para obtenção dos resultados da análise é apresentada, incluindo métricas, parâmetros e fatores utilizados na experimentação.

3.1. Cenário

O cenário de experimentação inclui uma rede emulada no Mininet [Lantz *et al.* 2010] com *Open vSwitches* operando em modo *kernel* e um controlador Floodlight versão 0.90 [Big Switch Networks 2014] gerenciando a rede como um elemento externo. Tanto a rede emulada como o controlador Floodlight se encontram na mesma máquina física em que foram realizadas as experimentações. As experimentações foram realizadas dessa forma para que não haja uma latência adicional entre o controlador e os *switches* emulados no Mininet. Cada experimento foi realizado em duas topologias: (*i*) uma em estrela com um *switch*, seis máquinas, um servidor de arquivos e um servidor de *stream* e (*ii*) uma em árvore com profundidade três e largura dois, constituída por sete *switches*. Assim como a topologia estrela, a topologia em árvore é constituída por seis máquinas, um servidor de arquivos e um servidor de *stream* de vídeo. O posicionamento dos servidores estão localizados nas extremidades da rede. Na Figura 1 pode-se observar as topologias utilizadas para as experimentações.

Para a realização da coleta das informações relativas ao tráfego de controle e monitoramento, foram necessárias modificações no módulo gerenciador de estatísticas existente no controlador Floodlight. Essas modificações foram realizadas, pois esse controlador não possui uma maneira padrão de adquirir as informações relativas a estatísticas de



(a) Topologia em estrela (b) Topologia em árvore

Figura 1. Topologias de rede

uso do canal de controle. Uma vez realizadas tais modificações, é possível tanto coletar informações a respeito do tráfego de dados como a respeito do tráfego de controle gerado na rede. A partir disso, foram analisadas informações referentes a regras instaladas nos dispositivos, bem como o monitoramento de estatísticas nos *switches* da rede.

3.2. Metodologia de Avaliação

A análise quantitativa do tráfego de controle e monitoramento em redes OpenFlow versão 1.0 é realizada considerando as métricas: (*i*) taxa de transferência média das mensagens de monitoramento para regras instaladas nos *switches* (*Read-State*), (*ii*) taxa de mensagens do tipo *Packet-In* processadas por segundo pelo controlador e (*iii*) taxa de mensagens do tipo *Packet-Out/Modify-State* processadas pela rede. Essas métricas foram escolhidas com o intuito de avaliar a quantidade de mensagens enviadas em função do tempo, tanto para mensagens enviadas para o controlador como para os dispositivos da rede. Os parâmetros de experimentação são fixados para todos os experimentos realizados e estão dispostos na Tabela 2. Utilizando sempre os mesmos parâmetros nos experimentos, é possível elaborar variações de cenários e gerar resultados que sejam comparáveis.

Toda a experimentação foi realizada em uma única máquina com sistema operacional Linux Ubuntu 12.10 64-bit com processador Intel® Core™ 2 Duo CPU E8400 @ 3,00GHz x 2 e 2Gb de memória RAM. O ambiente experimental foi emulado no Mininet versão 2.0.0. O software Apache versão 2.2.20 foi utilizado como servidor de arquivos. O tamanho médio dos arquivos transferidos foi configurado em 54 KB (conforme definição do tamanho médio de uma página Web [3GPP2 2004]). O software VLC media player 2.0.8 Twoflower foi utilizado como servidor de *stream* de vídeo utilizando os *Codecs Advanced Systems Format* (ASF) e *Windows Media Video* (WMV) para uma duração do vídeo fixada em 2 minutos [Cheng *et al.* 2007]. Em ambos servidores foram utilizados o protocolo HTTP. A largura de banda de todos os enlaces da rede foi fixada em 100 Mb e as requisições dos arquivos e vídeos foram baseadas em uma função exponencial apresentada em mais detalhes na Tabela 2 (também com base em [3GPP2 2004]).

Já os fatores a serem variados nos experimentos compreendem: (*i*) o intervalo de tempo entre monitoramentos que varia em 1, 5 e 10 segundos, (*ii*) tempo de ociosidade de regras de encaminhamento (tempo que uma regra fica instalada em um *switch* antes de ser descartada por inatividade) em 1, 30 e 60 segundos e (*iii*) as respectivas topologias (a) e (b) apresentadas na Subseção 3.1. Para os experimentos relativos ao monitoramento de

Parâmetros de configuração	Valores
Tamanho médio do arquivo	54 KB
Codec de vídeo	ASF/WMV
Duração do vídeo	2 minutos
Protocolo de transmissão	HTTP
Largura de banda	100 Mb
Tempo entre requisições	Função exponencial μ = 30 segundos, λ = 0.033

Tabela 2. Parâmetros de configuração dos servidores de arquivo e vídeo

estatísticas (mensagens *Read-State*) o tempo de ociosidade da regra foi fixado 5 segundos. Com isso, se pretende mostrar como variações do intervalo entre monitoramentos podem influenciar a precisão das informações obtidas da rede e o impacto dessas mensagens irão gerar no canal de controle. Nos experimentos realizados para instalação de regras de encaminhamento (mensagens *Packet-In/Out* e *Modify-State*) o tempo entre monitoramentos é fixado em 1 segundo para obter os resultados no menor tempo possível.

As técnicas de avaliação utilizadas neste trabalho são a modelagem analítica do comportamento das mensagens de controle *Read-State*, *Packet-In/Out* e *Modify-State* e experimentação para as mesmas mensagens. Para validar as experimentações realizadas a modelagem analítica foi conduzida inicialmente como forma de identificar a frequência e impactos das mensagens no canal de controle, a fim de, inferir o comportamento da rede de acordo com as topologias e tamanho das mensagens. A experimentação foi realizada em duas topologias com o mesmo número de *hosts*, porém variando a disposição e o número de *switches* da rede. Além disso, o estudo foi conduzido em duas etapas: (*i*) para as mensagens de *Packet-In/Out* e *Modify-State* foram variados os fatores de tempo de ociosidade da regra e topologias e, (*ii*) para as mensagens de *Read-State* os fatores a serem variados foram a frequência de monitoramento e as topologias. A realização dessas variações foram utilizadas, pois apresentam diferentes níveis de granularidade de monitoramento e expiração das regras instaladas nos *switches*. Os experimentos realizados seguiram o *design* fatorial completo [Jain 2008], onde são realizadas todas as combinações possíveis de fatores dentre os estabelecidos nas etapas (*i*) e (*ii*).

4. Análise dos Resultados

Esta seção apresenta os detalhes sobre as experimentações realizadas para avaliar o comportamento do canal de controle para as mensagens de coleta de estatísticas (*Read-State*) e de instalação de regras de encaminhamento (*Packet-In/Out* e *Modify-State*). Como maneira de validação da experimentação foi realizada primeiramente uma modelagem analítica sobre o comportamento esperado dos experimentos em relação à atividade da rede e implementação do controlador. Posteriormente, os resultados obtidos por experimentação são apresentados e discutidos.

4.1. Modelagem Analítica

Em relação às mensagens do tipo *Read-State*, existem duas variações que correspondem (*i*) as mensagens encaminhadas do controlador para o *switch*, requisitando por estatísticas, e (*ii*) as mensagens do *switch* para o controlador, informando as estatísticas coletadas. As mensagens enviadas do controlador para os *switches* são chamadas de *Stats Request* e as enviadas dos *switches* de volta para o controlador são chamadas de *Stats Reply*. Com base na análise da especificação do protocolo OpenFlow é possível estabelecer o tamanho das

mensagens de *Stats Request* e *Stats Reply* baseado em características dos *switches* (*e.g.*, número de portas ou quantidade de tabelas) ou na quantidade de regras de encaminhamento instaladas nos mesmos.

Todas as mensagens do tipo *Read-State* possuem um mesmo cabeçalho de tamanho fixo de 12 bytes e uma parte variável (corpo da mensagem) cujo tamanho depende do tipo de estatística requisitada ou dos dados coletados na resposta. Mensagens do tipo *Stats Request* possuem tamanho previsível correspondente ao tipo de estatística requisitada, isto é, mensagens dos tipos *Individual-Flow-Request*, *Aggregate-Flow-Request*, *Port-Request* e *Queue-Request* incluem um corpo adicional correspondente a 44, 44, 8 e 8 bytes, respectivamente. Mensagens de *Description-Request* e *Table-Request* não requerem mais informações para serem enviadas aos *switches*, portanto não adicionam bytes ao pacote.

Já o tamanho do *frame* OpenFlow das mensagens *Stats Reply* varia tanto pelo tipo de estatística coletada e quanto pela quantidade de informações retornadas. Mensagens de resposta de estatísticas por porta, por exemplo, incluem um corpo adicional de 104 bytes para cada porta de cada *switch* da rede. Assumindo que um *switch* não varia a sua quantidade de portas ao longo do tempo (em *switches* virtuais isso seria possível), um *switch* de 24 portas gera sempre mensagens de *Stats Reply* para estatísticas de porta de 2058 bytes. No entanto, para estatísticas individuais por regras de encaminhamento (*Individual-Flow-Stats*), por exemplo, é significativamente mais complexo prever o tamanho das mensagens, uma vez que a quantidade de regras instaladas em cada *switch* depende do tráfego gerado na rede e da lógica de funcionamento do controlador.

Assumindo que cada fluxo de dados é uma conexão lógica, normalmente bidirecional e *unicast*, entre dois *endpoints/hosts* da rede que passa por um conjunto de dispositivos de encaminhamento (caminho do fluxo) e que pelo menos duas regras de encaminhamento precisam ser instaladas em cada *switch* para estabelecer o fluxo de dados (regras de ida e de volta), a Equação 1, 2 e 3 representam genericamente a sobrecarga de monitoramento (SM) gerada por mensagens de *Stats Reply* em relação a quantidade de fluxos ativos em uma rede.

$$SM = SM_F + SM_V \tag{1}$$

$$SM_F = N_{switches} * (H_{OF} + H_{SR}) \tag{2}$$

$$SM_V = \sum_{f \in F} \left(\sum_{s \in Path_f} (Body_{IFS} * 2) \right) \tag{3}$$

A sobrecarga de monitoramento SM é dada por uma parte fixa SM_F (Equação 2) e uma parte variável SM_V (Equação 3). A parte fixa corresponde ao cabeçalho padrão do OpenFlow (H_{OF}) mais o cabeçalho específico das mensagens *Stats Reply* (H_{SR}), totalizando 12 bytes. Esse valor é ainda multiplicado pela quantidade de *switches* na rede, uma vez que cada dispositivo responderá individualmente sobre as suas estatísticas de regras de encaminhamento (tendo ou não regras instaladas). A parte variável depende da

quantidade de fluxos ativos (conjunto F) e da quantidade de dispositivos no caminho que esses fluxos percorrem ao longo da rede (conjunto $Path_f$). Sendo assim, para cada fluxo f ativo na rede e para cada *switch* s no caminho de cada fluxo são somados os dados referentes às informações das regras de encaminhamento *Individual-Flow-Stats* ($Body_{IFS}$) para ida e volta. O tamanho de $Body_{IFS}$ é de 88 bytes mais 8 bytes por *action* totalizando 96 bytes, considerando que os fluxos *unicast* terão sempre apenas uma *action* associada.

Outro tipo de mensagem que aparece em quantidade significativa no canal de controle de redes OpenFlow são as mensagens utilizadas para instalação de regras de encaminhamento: *Packet-In*, *Packet-Out* e *Modify-State*. A forma como essas mensagens são utilizadas depende da implementação das aplicações no controlador, mas basicamente para imitar o comportamento de redes Ethernet, existem pelo menos três implementações possíveis [Klein e Jarschel 2013]. A primeira seria o que os autores Klein e Jarschel chamam de *Hub behavior*, onde para cada *Packet-In* gerado por um *switch* é gerado um *Packet-Out* que faz *flood* do pacote para todas as interfaces (o que é obviamente ineficiente). A segunda implementação é chamada *Switch behavior*, onde o controlador aprende a localização dos *hosts* conforme recebe mensagens *Packet-In*, instala uma regra de encaminhamento com *Modify-State* e envia o pacote recebido seguindo essa regra para cada *Packet-In* recebido de cada *switch* do caminho do fluxo. A terceira e mais otimizada implementação é chamada *Forwarding behavior*. Nesta última, ao receber o primeiro *Packet-In* para estabelecimento de um fluxo o controlador primeiramente instala as regras com mensagens *Modify-State* nos demais *switches* do caminho, para depois instalar a última regra no *switch* que gerou o *Packet-In* enviando o pacote com uma mensagem *Packet-Out*. Obviamente, o controlador somente conseguirá operar nesse modo depois de conhecer a localização dos *hosts*, o que deve acontecer depois que estes enviarem tráfego para a rede, até então o controlador terá que utilizar, por exemplo, uma implementação *Hub behavior*.

Mensagens do tipo *Packet-In* são enviadas pelos *switches* ao controlador sempre que estes não encontram uma regra na tabela local para encaminhar um pacote recebido. Essas mensagens são compostas de um cabeçalho padrão de 20 bytes mais o pacote inteiro recebido que é encapsulado na mensagem. Geralmente, os pacotes enviados nesse tipo de mensagem serão ARP ou TCP-SYN, por exemplo, que incrementarão em 42 ou 74 bytes, respectivamente no tamanho do *Packet-In*. A frequência com que novos fluxos aparecerão na rede pode ser estimada estatisticamente através de modelos como os apresentados na Tabela 2. Além disso, a geração de mensagens *Packet-In* é influenciada pelo tempo de ociosidade das regras de encaminhamento, o que é uma configuração feita pelo controlador. Quanto mais longo o tempo de ociosidade das regras, menos mensagens *Packet-In* devem ser enviadas ao controlador.

Mensagens dos tipos *Packet-Out* e *Modify-State* são geralmente utilizadas em resposta do controlador às mensagens *Packet-In* enviadas pelos *switches*. Considerando a implementação *Forwarding behavior* que é utilizada pelo controlador Floodlight, a Equação 4 expressa a sobrecarga de instalação de fluxos (SF) gerada em uma rede OpenFlow.

$$SF = \begin{cases} N_{switches} * (H_{OF} + H_{PO} + M_{Original}) & se\ host\ desconhecido \\ \left(\sum_{s \in Path_f} (H_{OF} + H_{MS}) \right) + (H_{OF} + H_{PO} + M_{Original}) & se\ host\ conhecido \end{cases} \quad (4)$$

SF é calculada diferentemente em duas situações. Primeiro, no caso da posição do *host* destino do fluxo na rede seja desconhecido pelo controlador, este irá enviar apenas mensagens de *Packet-Out* ($H_{OF} + H_{PO} + M_{Original}$) para todas as portas de todos os *switches* da rede ($N_{switches}$) (isso acontece assim que forem recebidas no controlador todas as mensagens de *Packet-In* correspondentes), como um *Hub behavior*. Segundo, quando a posição do *host* é conhecida, o controlador envia mensagens *Modify-State* ($H_{OF} + H_{MS}$) para cada *switch* s no caminho do fluxo $Path_f$ e envia apenas uma mensagem *Packet-Out* ($H_{OF} + H_{PO} + M_{Original}$) de volta ao *switch* que originou o primeiro *Packet-In*. Em geral, esse procedimento ocorrerá uma vez para o estabelecimento do caminho de ida do fluxo. No caminho de volta o *host* destino já será conhecido pelo controlador (já que ele originou o primeiro pacote), sendo assim, será necessário apenas mais uma execução do caso onde o *host* é conhecido.

Vale ressaltar que o custo das mensagens *Packet-In* não está incluído no cálculo de SF – assim como os custos das mensagens *Stats Request* não estavam em SM – uma vez que representam tráfego em sentidos diferentes no canal de controle. No entanto, estimar os valores de sobrecarga de *Packet-In* para as duas situações é trivial. No caso da posição do *host* destino do fluxo na rede seja desconhecido pelo controlador, serão geradas uma mensagem *Packet-In* para cada *switch* e já quando a posição do *host* é conhecida, é enviada uma mensagem *Packet-In* apenas. Também é importante salientar que não foram considerados nos cálculos apresentados a sobrecarga adicional de transporte. O canal de controle OpenFlow possui diversas configurações possíveis de transferir dados, incluindo TCP, UDP, SSL, o que ocasiona ainda mais sobrecarga tanto de tráfego de rede quanto de processamento por parte do controlador e dos demais dispositivos da rede.

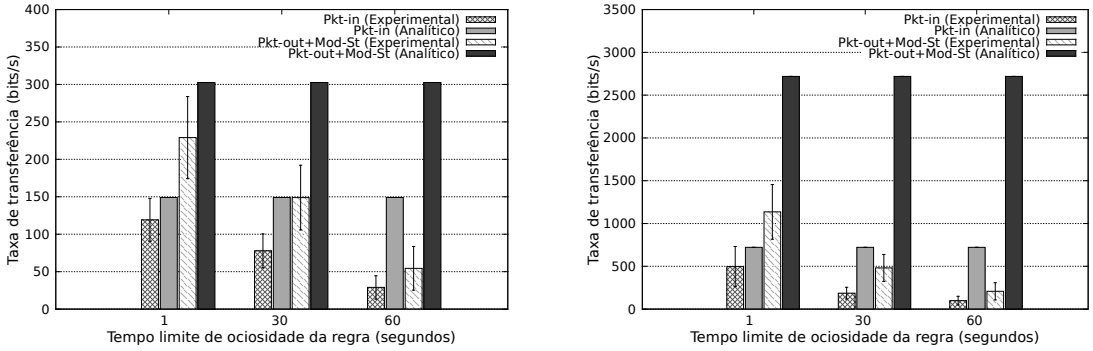
4.2. Resultados Experimentais

Os resultados da experimentação apresentam informações relevantes tanto para as mensagens de instalação de regras de encaminhamento (etapa (*i*), mensagens de *Packet-In/Out* e *Modify-State*) como para as mensagens de monitoramento de estatíticas (etapa (*ii*), mensagens de *Read-State*). Para fins comparativos, a modelagem analítica foi utilizada como *baseline* para a situação onde existe a maior sobrecarga de mensagens na rede, dado os parâmetros da experimentação em cada uma das duas topologias.

As Figuras 2(a) e 2(b) apresentam o comportamento das mensagens *Packet-In/Out* e *Modify-State* para as topologias de estrela e árvore respectivamente. O comportamento das mensagens *Packet-In/Out* e *Modify-State* apresentam grandes variações durante o período do experimento. Variação que acontece pelo fato dessas mensagens só ocorrem na rede quando os fluxos são iniciados e os *switches* não possuem regras instaladas ou quando as regras para um determinado fluxo expiram baseadas em um tempo limite de ociosidade da regra instalada no *switch*. Devido a essas variações, com 90% de confiança é possível afirmar que para ambas as topologias a taxa de transferência das mensagens

do tipo *Packet-In* enviadas para serem processadas no controlador diminuiu significativamente conforme aumenta o tempo limite de ociosidade da regra instalada expirar. Isso indica que em redes onde as regras são acionadas mais frequentemente, por exemplo com intervalo de menos de 30 ou 60 segundos, não é necessária a desinstalação imediata das regras de encaminhamento quando o fluxo fica inativo por 1 segundo.

As mensagens de *Packet-Out* e *Modify-State* obedecem a mesma variação das mensagens *Packet-In*, porém, são enviadas em direção dos dispositivos de encaminhamento ao invés do controlador. De acordo com a experimentação, 'é possível afirmar com 90% de confiança para ambas as topologias a taxa de mensagens do tipo *Packet-Out* e *Modify-State* ocorrem com mais frequência de acordo com o tempo limite de ociosidade da regra instalada no *switch*. Isso indica que em ambientes onde as regras são acionadas com frequência não é necessária a desinstalação da regra imediatamente, podendo causar sobrecarga de mensagens de controle desnecessárias no canal de controle.



(a) Topologia em estrela

(b) Topologia em árvore

Figura 2. Taxa de transferência das mensagens *Packet-It* e *Packet-Out/Modify-State* de acordo com o tempo limite de ociosidade das regras para as topologias de estrela e árvore

Uma vez compreendido o comportamento dos usuários na rede e conhecendo a topologia pode se estimar o caso onde existe a maior sobrecarga de mensagens *Packet-In* para o controlador. A partir da modelagem analítica pode ser mensurado o impacto das mensagens no canal de controle para o caso onde tempo limite de ociosidade das regras seja quase que imediato (1 segundo). Nas Figuras 2(a) e 2(b) é possível perceber que as taxas de transferência para os resultados analíticos se mantém estáveis para todos os tempos de ociosidade. Isso ocorre pois a modelagem não levou em consideração esses tempos, estimando um caso onde as regras sempre expiram antes da próxima comunicação entre dois *hosts*. A partir desses resultados, é possível afirmar que é necessário o conhecimento a respeito do comportamento dos fluxos gerados na rede para não gerar pacotes desnecessários ao controlador nem manter regras inativas instaladas desnecessariamente nos dispositivos de encaminhamento.

A Figura 3 apresenta o comportamento da taxa de pacotes por segundo das mensagens do tipo *Packet-In/Out* para as topologias de estrela e árvore. Foram calculadas também as taxas de 0.17, 0.113, 0.005 e 0.33, 0.22 e 0.008 pacotes por segundo de mensagens *Packet-In* processadas no controlador nas topologias de estrela e árvore respectivamente. Para as mensagens de *Packet-Out* e *Modify-State* foram calculados 0.5, 0.3, 0.17 e 1.5, 0.728, 0.33 pacotes por segundo enviados em direção dos *switches* também para as topologias de estrela e árvore. Valores que em primeiro momento não demonstram relevância,

porém, devido aos 6 *switches* mais que a topologia em árvore apresenta, a taxa de pacotes por segundo processados pelo controlador aumenta em 51,5%, 51,4% e 62,5% para as mensagens do tipo *Packet-In* nas três variações do experimento. Para as mensagens *Packet-Out* e *Modify-State* as variações ficam em torno de 33.3%, 41.2% e 51,1% de aumento, devido a topologia com maior número de *switches*. Assim, pode-se afirmar que o número de *switches* impacta diretamente na sobrecarga de mensagens *Packet-In* enviadas ao controlador, ainda que, a modelagem analítica não tenha como prever problemas como pacotes fora de ordem e retransmissões, os quais podem ocasionar em mais chegadas de mensagens *Packet-In* no controlador.

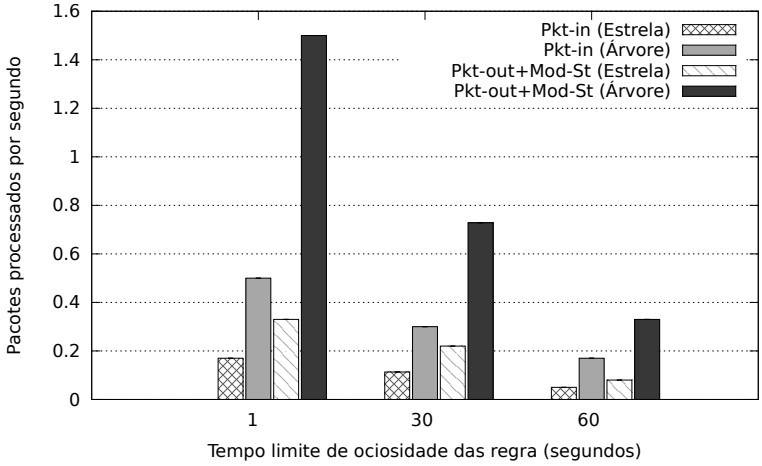
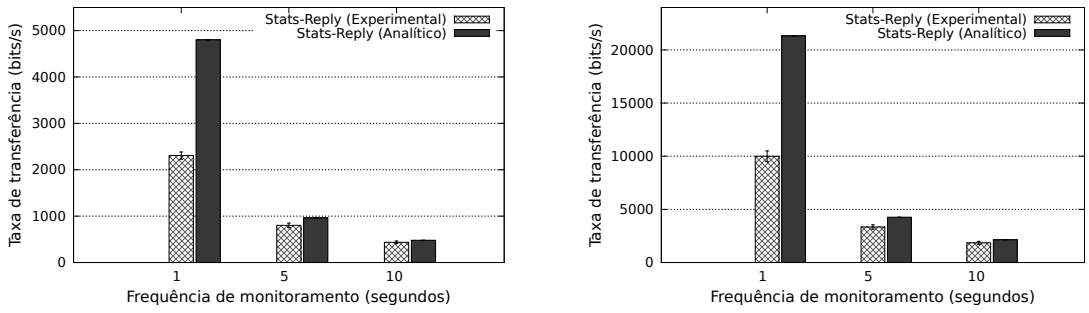


Figura 3. Taxa de pacotes processados por segundo de mensagens *Packet-In*, *Packet-Out/Modify-State* de acordo com o tempo limite de ociosidade das regras para as topologias de estrela e árvore

A Figura 4(a) e 4(b) apresentam os resultados para a taxa de transferência de mensagens *Stats-Reply* para as topologias de estrela e árvore, respectivamente. Se tratando das mensagens para monitoramento de estatísticas (*Read-State*), os resultados apresentaram que para 95% de confiança existe pouca variabilidade na taxa de transferência média dos pacotes de estatísticas coletados. As mensagens de *request* não foram exibidas em gráficos, pois são fixas e podem ser facilmente calculadas de forma similar ao cálculo de SM_F já explicadas pela Equação 1. As mensagens de *reply* dependem do número de regras instaladas nos *switches* da rede e essa variabilidade impacta no tamanho das mensagens recebidas pelo controlador.



(a) Topologia em estrela

(b) Topologia em árvore

Figura 4. Taxa de transferência para mensagens de *Stats-Reply* de acordo com a frequência de monitoramento para as topologias de estrela e árvore

Além do número de regras instaladas nos *switches* da rede, que impacta no tamanho das mensagens, o número de *switches* a serem consultados também influencia no desempenho do controlador. Para a topologia em árvore por exemplo, o envio e recebimento de mensagens de monitoramento é 6 vezes maior do que na topologia estrela, pois o controlador precisa enviar uma mensagem de *request* e recebe uma *reply* de cada *switch* da rede. Escalando para topologias maiores como de *data centers* por exemplo, o impacto e frequência do monitoramento pode ser maior, comprometendo a comunicação entre controlador e dispositivos de encaminhamento. A partir desse estudo pode se afirmar que a frequência do monitoramento pode afetar significativamente dependendo do número de dispositivos de encaminhamento na rede. Além disso, deve-se avaliar a possibilidade de realizar o monitoramento direcionado ao um conjunto menor de dispositivos de mais interesse, evitando uma possível coleta desnecessária.

5. Conclusões e Trabalhos Futuros

Neste trabalho foi discutido acerca da centralização e distribuição da lógica de controle no contexto de SDN e OpenFlow. Partindo dessa discussão, foram apresentadas soluções que resolvem problemas relacionados aos gargalos gerados pela centralização do controle da rede proposta no contexto do protocolo OpenFlow. Porém, não são quantificados o custo e nem a frequência em que tais gargalos podem ocorrer. Portanto, devido a ausência de um estudo quantitativo acerca das mensagens de controle utilizadas nesse contexto, foi proposta uma análise para as mensagens que aparecem mais frequentemente no canal de controle em redes OpenFlow. A análise foi constituída de (*i*) uma modelagem analítica, para identificação do comportamento das mensagens (*Packet-In/Out*, *Modify-State* e *Read-State*) trocadas entre controlador e *switches* na rede e; (*ii*) a experimentação em duas topologias distintas, variando frequência de monitoramento e tempo limite de ociosidade das regras instaladas nos dispositivos de encaminhamento.

Os resultados da análise realizada mostram que as mensagens de instalação de regras de encaminhamento *Packet-In/Out* e *Modify-State* possuem um comportamento dependente da implementação das aplicações realizadas no controlador, por exemplo, do tempo limite de ociosidade das regras. Já as mensagens de coleta de estatísticas (*Read-State*) são influenciadas principalmente devido a quantidade de dispositivos monitorados e pela frequência em que são monitorados. A modelagem analítica apresenta as equações para estimar a sobrecarga gerada pela transmissão das mensagens de controle assumindo que a rede não possui atrasos nem retransmissões. Fato que explica os resultados apresentados, onde todos os valores calculados analiticamente são superiores aos experimentais. Com base na análise apresentada, espera-se contribuir para o projeto e desenvolvimento de novos sistemas de gerenciamento de redes baseadas no paradigma SDN e OpenFlow.

Como trabalhos futuros pretende-se expandir os experimentos acerca das topologias abordadas, variando tanto o número de *switches* da rede como o número de *hosts*. Fatores como a frequência de monitoramento podem ser explorados em maior escala, a fim de estabelecer uma relação entre a sobrecarga gerada na rede e a precisão dos dados obtidos sobre os fluxos monitorados. Também pode ser expandida a análise para outras versões do protocolo OpenFlow utilizando níveis de prioridades, além de instalação de regras mais genéricas ou mais específicas. Por fim, além desses aspectos pretende-se realizar novos experimentos em uma rede com *switches* reais a fim de que se possa analisar o limite de processamento das regras instaladas e monitoradas através do canal de controle.

Referências

3GPP2 (2004). CDMA2000 Evaluation Methodology. Disponível em: <http://www.3gpp2.org/Public_html/specs/C.R1002-0_v1.0_041221.pdf>. Acesso em: Março 2014.

Bari, M. F., Roy, A. R., Chowdhury, S. R., Zhang, Q., Zhani, M. F., Ahmed, R., e Boutaba, R. (2013). Dynamic controller provisioning in software defined networks. In *Proceedings of the 9th IEEE/ACM/IFIP International Conference on Network and Service Management 2013 (CNSM 2013)*.

Big Switch Networks (2014). Floodlight an Open SDN Controller. Disponível em: <https://www.projectfloodlight.org/floodlight/>. Acesso em: Março 2014.

Cheng, X., Dale, C., e Liu, J. (2007). Understanding the characteristics of internet short video sharing: Youtube as a case study. *arXiv preprint arXiv:0707.3670*.

Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., e Banerjee, S. (2011). Devoflow: scaling flow management for high-performance networks. *SIGCOMM-Computer Communication Review*, 41(4):254.

Heller, B., Sherwood, R., e McKeown, N. (2012). The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 7–12. ACM.

Jain, R. (2008). *The art of computer systems performance analysis*. John Wiley & Sons.

Kim, H. e Feamster, N. (2013). Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119.

Klein, D. e Jarschel, M. (2013). An openflow extension for the omnet++ inet framework. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, SimuTools '13, pp. 322–329, Brussels, Belgium.

Lantz, B., Heller, B., e McKeown, N. (2010). A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pp. 19:1–19:6, New York, NY, USA. ACM.

Levin, D., Wundsam, A., Heller, B., Handigol, N., e Feldmann, A. (2012). Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 1–6. ACM.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., e Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

Roy, A. R., Bari, M. F., Zhani, M. F., Ahmed, R., e Boutaba, R. (2014). Design and management of dot: A distributed openflow testbed. In *Preceedings of the 14th IEEE/IFIP Network Operations and Management Symposium (NOMS 2014)(To appear)*.

Tootoonchian, A. e Ganjali, Y. (2010). Hyperflow: a distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pp. 3–3. USENIX Association.

Yu, M., Rexford, J., Freedman, M. J., e Wang, J. (2010). Scalable flow-based networking with difane. *ACM SIGCOMM Computer Communication Review*, 40(4):351–362.

AppendixB PUBLISHED PAPER – IM 2015

In this paper it was published an extension of the control channel analysis and a management approach through monitoring, visualization, and configuration of OpenFlow-based SDN. We motivated our proposed approach by extending our control channel analysis considering control channel load and resource consumption. Next, we propose an interactive approach to SDN management through monitoring, visualization and configuration. Our main goal was to include the administrator in a management loop, where SDN-specific metrics are monitored, processed, and displayed in interactive visualizations. By integrating these management activities with the administrator interaction, we allowed the administrator to better understand and control the network.

- **Title –**
  Interactive Monitoring, Visualization, and Configuration of OpenFlow-Based SDN
- **Symposium –**
  14th IFIP/IEEE International Symposium on Integrated Network Management (IM - 2015)
- **Type –**
  Main track (full-paper)
- **Qualis –**
  B1
- **URL –**
  <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=XXXXX>
- **Date –**
  May 11-15, 2015
- **Held at –**
  Ottawa, Canada
- **Digital Object Identifier (DOI) –**
  <http://dx.doi.org/XX.XXXX/IM.2015.X>

Interactive Monitoring, Visualization, and Configuration of OpenFlow-Based SDN

Pedro Heleno Isolani*, Juliano Araujo Wickboldt*, Cristiano Bonato Both†,
Juergen Rochol*, and Lisandro Zambenedetti Granville*
*Federal University of Rio Grande do Sul, Porto Alegre, Brasil
†University of Santa Cruz do Sul, Santa Cruz do Sul, Brasil
Email: {phisolani, jwickboldt, juergen, granville}@inf.ufrgs.br, cboth@unisc.br

***Abstract*—Software-Defined Networking (SDN) is an emerging paradigm that arguably facilitates network innovation and simplifies network management. However, in the context of SDN, management activities, such as monitoring, visualization, and configuration can be considerably different from traditional networks. An SDN controller, for example, can be customized by network administrators according to their needs. Such customizations might pose an impact on resource consumption and traffic forwarding performance, which is difficult to assess without an SDN-devoted management system. In this paper, we initially present an analysis of control traffic in SDN aiming to better understand the impact of the communication between the controller and forwarding devices. Afterwards, we propose an interactive approach to SDN management through monitoring, visualization, and configuration that includes the administrator in the management loop. To show the feasibility of our approach a prototype has been developed. The results obtained with this prototype show that our approach can help the administrator to better understand the impact of configuring SDN-related parameters on the overall network performance.**

I. INTRODUCTION

Software-Defined Networking (SDN) is an emerging paradigm that enables network innovation based on four fundamental principles: (*i*) network control and forwarding planes are clearly decoupled, (*ii*) forwarding decisions are flow-based instead of destination-based, (*iii*) the network forwarding logic is abstracted from hardware to a programmable software layer, and (*iv*) an element, called controller, is introduced to coordinate network-wide forwarding decisions [1]. SDN is grabbing the attention of both academia and industry, since it allows the easy creation of new abstractions in networking, simplifying management, and facilitating network innovation [2]. In this sense, SDN reduces or even eliminates some traditional network management problems, such as enabling network configuration in a high-level language or providing support for enhanced network diagnosis and troubleshooting [3].

Monitoring, visualization, and configuration are fundamental management activities to understand and control the network behavior [4] [5] [6] [7] [8]. In the context of SDN, these activities can be considerably different than in traditional networks, thus deserving proper attention [9]. For example, the behavior of an SDN controller can be customized by network administrators according to their needs. However, these controller customizations might impact in terms of resource consumption and traffic forwarding performance. As a consequence, such impact is difficult to assess because traditional network management solutions were not designed to cope with the context of SDN. Therefore, an SDN-tailored management system must be able to help the administrator to understand and control how the SDN controller behavior affects the network.

The state-of-the-art in SDN has addressed monitoring for different purposes [10] [11] [12] [13] [14]. Most of these investigations are focused on proposing anomaly detection systems or mechanisms to establish a balance between control channel overhead and accuracy of collected information. These investigations tend to employ monitoring information to automatically adapt the network to specific conditions. However, to the best of our knowledge, no solution is available to integrate monitoring information with interactive visualization and configuration tools for SDN. We argue that with such a solution the administrator could better understand and interact with the network, significantly improving everyday tasks of SDN management.

In this paper, we initially perform an analysis of the control traffic in SDN to understand the overall impact in terms of resource consumption and network performance resulted from the communication between the controller and network devices. Based on this analysis, we propose an interactive approach to SDN management through monitoring, visualization, and configuration. Our goal is to include the administrator in the management loop, where SDN-specific metrics are monitored, processed, and displayed in interactive visualizations. Thus, the administrator is able to make decisions and configure/reconfigure SDN-related parameters according his/her needs. Our main contribution is to integrate these three activities allowing the administrator to easily understand and control the network.

To emphasize the feasibility of our approach, we developed a prototype that uses the OpenFlow protocol [15], currently the most relevant SDN implementation. Our prototype is able to: (*i*) perform monitoring with configurable polling intervals specifically focused in metrics related to resource usage and control channel load, (*ii*) present aggregated statistics in interactive visualizations that emphasize these metrics, and (*iii*) support the configuration of network parameters that affect the analyzed metrics. To evaluate our approach, we use the Floodlight controller [16] and a campus network scenario emulated over Mininet [17]. Our results show that, by interacting with visualizations, the administrator is able to adjust the network, improve the controller configuration, and thus reduce the resource consumption and control channel usage.

978-3-901882-76-0 @2015 IFIP

The remainder of this paper is organized as follows. In Section II, we provide a brief description of the main background concepts associated with our approach and related work. In Section III, we present an analysis of control traffic in OpenFlow-based SDN. In Section IV, we present our approach in detail and our developed prototype. In Section V, we present the evaluation of our proposed approach. Finally, in Section VI, we conclude the paper with final remarks and perspective for future work.

II. Background and Related Work

This section provides a brief overview of the main background concepts required by our proposed approach to interactive SDN monitoring, visualization, and configuration.

A. Related Work

Currently, many investigations consider using monitoring information obtained with the OpenFlow protocol for different purposes. Zhang [10] proposed the use of monitoring messages to develop algorithms for anomaly detection systems based on methods for statistics information collection. Jose *et al*. [13] used the same monitoring messages to propose mechanisms for online measurement of large traffic aggregates, which can be used for both anomaly detection and traffic engineering. Yu *et al*. [11] proposed FlowSense, which is aimed to keep the lowest control channel overhead and the highest information accuracy as possible in OpenFlow monitoring. FlowSense, however, uses only *Asynchronous* messages of the OpenFlow protocol to collect statistics information (further detail on the message types are presented in Section II-B). Although FlowSense introduces zero cost in the monitoring process, the statistics information may be inaccurate in the case where OpenFlow messages received by the controller are too sparse.

Chowdhury *et al*. addressed SDN monitoring from a different perspective. The authors proposed a framework, called Payless [12], able to deal with monitoring considering the polling frequency and data granularity. This framework is able to adjust the polling frequency to balance the control channel overhead, imposed by monitoring messages, and accuracy of monitored information. Payless relies on OpenTM [14] to select only important switches to be monitored, also aiming to reduce the overhead imposed in the control channel. The authors employed a modularized architecture with an algorithm to set the polling frequency for monitoring a set of selected switches. Therefore, it is possible to extract just the relevant information while keeping the control channel overhead low.

The aforementioned proposals are focused on the use of monitoring information to automate tasks, such as reducing control traffic overhead and protecting the network. No previous investigation aims to employ monitoring information to help the administrator understanding the network behavior nor interact with it. To the best of our knowledge, there are no solutions that leverage OpenFlow monitoring messages to create network visualizations and address the interactive configuration of SDN-related parameters. Before presenting how we approached such a problem, we review in the next subsections key aspects of OpenFlow, focusing on its messages and on the OpenFlow controller behavior. Reviewing this aspects is important because they are central to our solution.

B. OpenFlow Brief Background

To establish flow paths over a network, an OpenFlow controller adds and removes forwarding rules in network switches. An OpenFlow forwarding rule is composed of a *match* of packet header fields, *e.g.*, source and destination IP addresses, and a set of *actions* to be performed, *e.g.*, forward or drop a packet. The OpenFlow specification also defines that an OpenFlow switch is composed of (*i*) a flow table to store forwarding rules, (*ii*) a secure communication channel with the controller, and (*iii*) the OpenFlow protocol itself.

OpenFlow version 1.0, which is considered in this paper, defines three message types: (*i*) *Controller-to-switch*, (*ii*) *Asynchronous*, and (*iii*) *Symmetric* [18]. Controller-to-switch messages are initiated by the controller and are used to manage or inspect the state of OpenFlow switches. Asynchronous messages are initiated by the switch and are used to send notification of network events and changes to the controller. Symmetric messages can be initiated by either the controller or switches and are mainly used for network bootstrap, latency measurement, and to keep alive the control channel. Each of these message types is composed of multiple sub-types that are used for specific network coordination actions.

Some of the main message sub-types defined by the OpenFlow specification are: *Modify-State* and *Flow-Removed* to install and remove rules on forwarding devices; *Send-Packet* to send a packet to a specific switch port; and *Packet-In*, to notify the controller when a switch receives a packet that does not match with a forwarding rule entry in the switch flow table. Another message sub-type that is used by monitoring solutions to retrieve statistics from switches is *Read-State*. As a result, the OpenFlow specification enables both the configuration of forwarding devices and monitoring traffic statistics. Nevertheless, the OpenFlow specification does not state how messages should be used to actually manage an OpenFlow-based network. It is on the account of the administrator to understand the OpenFlow specification and the controller's behavior, and then decide how OpenFlow can be employed to accomplish everyday management tasks.

C. Controller Behavior

OpenFlow messages are used to coordinate forwarding devices in different ways, depending on the controller behavior implementation. Three well known controller behaviors that affect the installation of forwarding rules and, consequently, the network operation are: *Hub*, *Switch*, and *Forwarding* [19]. Throughout this paper, we adopted the *Forwarding* behavior because it is the most sophisticated out of these three mentioned behaviors, presenting a lower control overhead.

An example of how the *Forwarding* behavior coordinates rule installation between *Source Host* and the destination *File Server* is depicted in Figure 1. First, when *Source Host* sends a data packet to *Switch A* (1), this switch checks whether there is a forwarding rule entry matching this packet's header fields in the flow table. If the header fields match with an entry, the corresponding actions should be applied to this data packet, *e.g.,* forward or drop. However, if no match exists, by default, *Switch A* generates a *Packet-In* message to the controller (2). Upon receiving the *Packet-In*, the controller calculates the flow path and sends a set of *Modify-State* messages to install

forwarding rules in all switches in the path between source and destination, except to *Switch A* (3). Afterwards, the controller sends *Send-Packet* and *Modify-State* messages to *Switch A* that originated the *Packet-In* (4). Finally, once all forwarding rules are installed on switches, the data packet is sent through the flow path (5) to the destination.

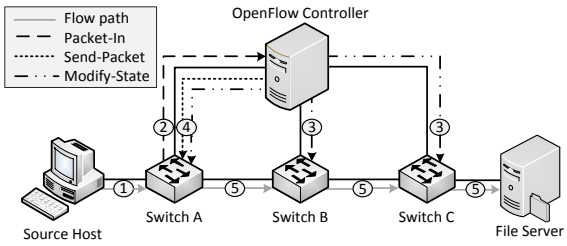


Fig. 1. OpenFlow controller *Forwarding* behavior example

A few extra points need to be clarified regarding the *Forwarding* behavior. First, the example in Figure 1 assumes that the location of the destination *File Server* is known by the controller in advance. In practice, before every host originates traffic, the controller will generally need to discover these locations by flooding the network. Also, the controller does not guarantee that rule installation messages, *i.e.*, *Modify-State*, arrive in switches properly ordered, since these messages can be sent in parallel. Thus, if *Modify-State* messages do not arrive in the switch before the data packet, extra *Packet-In* messages will be generated and sent to the controller. In addition, the controller behavior does not orchestrate how *Read-State* messages are used by either the controller or forwarding devices. As such, a monitoring solution is still required alongside the OpenFlow controller to fill this gap and help in managing OpenFlow-based networks.

III. Control Channel Analysis

In this section, we provide an analysis of the control channel traffic load of OpenFlow 1.0 messages, emphasizing those messages that occur most frequently in our scenario of study. In sub-section III-A, we present the motivation and the methodology of analysis. In sub-section III-B, we detail the scenario and the workload used in our case-study. Finally, in sub-section III-C, we explain and discuss the analysis.

A. Motivation and Methodology of Analysis

OpenFlow-based SDN introduces a simple way to develop and maintain communication networks because of its centralized logic of controlling forwarding devices. However, this simplicity may allegedly impose high cost on the network controller and create bottlenecks at the control channel [20]. Recent solutions, such as Devoflow [21] and DIFANE [22], attempted to alleviate these bottlenecks by distributing the control logic of OpenFlow. Nevertheless, to the best of our knowledge, no other study has detailed in which situations such bottlenecks appear and whether they can or cannot be mitigated or even avoided by simple configuration, *i.e.*, without the need to develop a specific distributed controller. Therefore, in our analysis, we initially quantify the load of OpenFlow 1.0 control messages that appear most frequently in our specific scenario of study. Afterwards, we also point out configuration parameters that can influence this load and may be set to reduce the chance of bottlenecks in the control channel.

Our analysis is divided into two perspectives of resource consumption: (*i*) control channel load related to installation of rules on forwarding devices (*Packet-In*, *Modify-State*, and *Send-Packet*) and request/reply messages for monitoring flow statistics (*Read-State*); and (*ii*) resource usage in terms of forwarding rules, active and idle, installed on network devices. These four sub-types of messages have been selected because, in our scenario of study, they represent the absolute majority of control traffic (accounting for 97.78% of the number of messages exchanged between the controller and forwarding devices, and 99.70% of the overall control traffic). Furthermore, regarding resource usage, the amount of rules installed in switches' expensive and limited Ternary Content-Addressable Memories (TCAM) needs to be carefully managed to avoid network devices running out of resources [21].

The metrics we analyze for control traffic load are: (*i*) bit rate and number of messages per second for monitoring, (*ii*) bit rate and number of messages for rule installation processed on the controller, and (*iii*) bit rate and number of messages for rule installation processed by network devices. The analyzed metrics for resource usage on forwarding devices are: (*i*) the total number of installed forwarding rules and (*ii*) the amount of those installed forwarding rules which are idle, *i.e.*, counters unchanged between two monitoring intervals.

B. Case-study: A Campus Network

To conduct our control channel analysis and also to evaluate our approach to SDN management (later discussed in Section V), we have chosen to use campus network scenario inspired in our own university premises. This type of network infrastructure is well suitable to benefit from features of SDN with OpenFlow, which has actually been originally conceived for this type of scenario [15]. Our emulated campus network consists in 11 OpenFlow switches connecting 230 hosts from laboratories and administration offices forming the topology shown in Figure 2. Each switch of the network is connected through a dedicated channel to a remote controller. More specifically, a centralized Floodlight v0.90 controller is set to coordinate network-wide forwarding decisions.

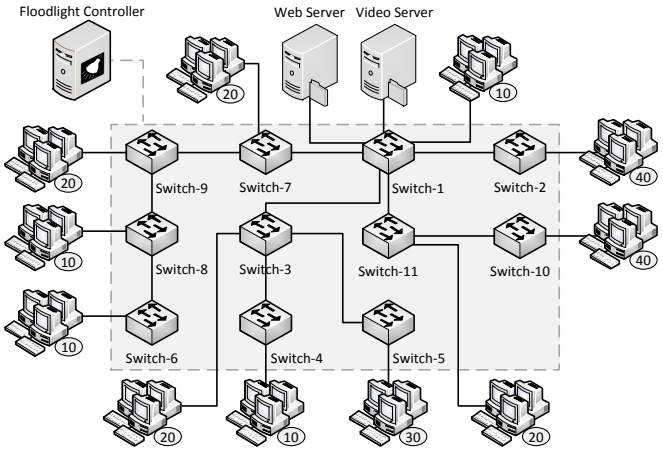


Fig. 2. Campus network topology

The general prevalent traffic profile of users is characterized by everyday Internet surfing and access to the university's virtual learning environment. We model user behavior to generate emulated Internet traffic based on several previous studies [23] [24] [25] [26] [27]. Table I summarizes the main parameters used to emulate user traffic profile during experimentation. In this experiment we generate only video and Web traffic, in a proportion of 75% to 25% respectively. Given the size of requests, *i.e.*, video streams generate more traffic than Web requests, we selected one user to place video requests for every 6 Web users. We include in the campus topology one Web Server and one Video Server to respond to users' requests, so that we are able to control precisely the size of responses (in a real campus network these requests would normally go through a gateway or proxy). We also assume that all 230 hosts are active during the whole experiment time and place a request in average every 30 seconds.

TABLE I. CONFIGURATION PARAMETERS OF USER TRAFFIC PROFILE

Parameter	Value
Web request size	Lognormal Distribution ($\mu = 11.75$, $\sigma = 1.37$) Mean: 324 KBytes, Std. Dev.: 762 KBytes
User reading time	Exponential Distribution ($\lambda = 0.033$) Mean: 30 seconds
Video watch time	180 seconds
Video bit rate	300 kbps
Traffic Mix	Video: 75%, Web: 25%
User Mix	1 video user for every 6 Web users
Monitoring	Polling frequency: 5 seconds
Controller behavior	Floodlight's default *Forwarding Behavior* implementation
Experiment duration	30 min

The experimentation workload was emulated on Mininet in one Dell PowerEdge R815 with 4 AMD Opteron Processor 6276 Eight-Core processors and 64GB of RAM server.

C. Experiments & Discussion

To understand the variations of control traffic, we choose to vary only one factor present in any OpenFlow-based network, which is the *idle timeout* of forwarding rules. This factor indicates when an entry of the flow table of an OpenFlow switch, *i.e.*, a forwarding rule, can be removed due to a lack of activity. The default *idle timeout* (in seconds) is configurable in the Floodlight controller and is applied for every new forwarding rule installed. Figures 3, 4, and 5 show how the *idle timeout* configuration affects both the control channel load and resource consumption. Moreover, to analyze *Read-State* messages, we fixed the polling frequency in 5 seconds to understand how the variation in size (not frequency) of these messages impacts control traffic, specially related to the size of *Read-State* (Reply) messages which contain per flow statistics. All experiment samples were sized so to achieve a 95% confidence and an error no higher than 2%. We suppressed error bars from Figures 4 and 5 for the sake of visualization.

Figure 3 shows how the control traffic load (in Kbps) varies as the *idle timeout* increases. The height of each bar shows the total control traffic in both directions, *i.e.*, from the controller to the network (*Send-Packet*, *Modify-State*, and *Read-State (Request)*) and form the network to the controller (*Packet-In* and *Read-State (Reply)*). It is clear to notice that the average traffic of *Read-State requests* remains constant, while *Read-State reply* traffic increases significantly. This happens because *Read-State replies* contain forwarding rule statistics, thus as the *idle timeout* increases so does the chance of a given forwarding rule being installed at a switch in each monitoring poll. However, *Packet-In*, *Send-Packet*, and *Modify-State* traffic decreases in average as the *idle timeout* increases. This happens because all these messages will appear mostly if users remain inactive for a period longer than the configured *idle timeout*, which is fairly unlikely to happen for long *idle timeout* values.

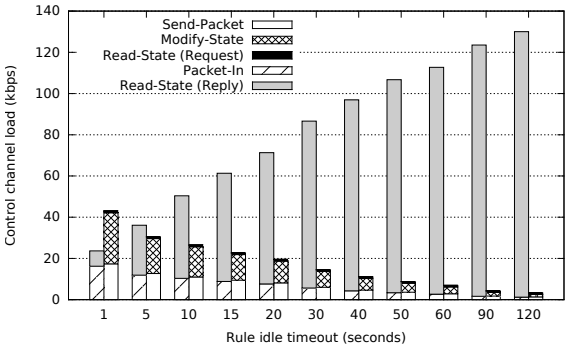


Fig. 3. Control channel load vs. *idle timeout* configuration

Figure 4 shows the number of packets processed per second by both the controller and network devices. Similar to Figure 3, this chart also shows in the total bar height an accumulated value, *i.e.*, the number of messages flowing in each direction. Mainly, the results in Figure 4 show that when the rule *idle timeout* configuration is set to low values, the average of *Packet-In* generated to the controller direction and *Send-Packet*/*Modify-State* messages sent to network devices both increase. Most importantly, the administrator needs to watch over this configuration to avoid bottlenecks at the controller.

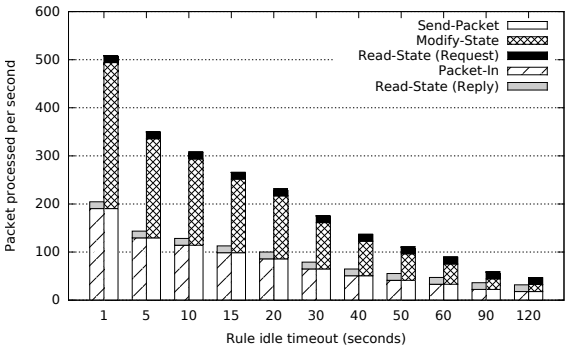


Fig. 4. Packets processed per second vs. *idle timeout* configuration

Figure 5 shows the total number of forwarding rules and which of those are idle for a given *idle timeout* configuration. A forwarding rule is idle when its counters do not change between two monitoring polls. The results show that increasing the *idle timeout* value causes a larger number of idle rules installed in switches. This occurs because the user traffic profile of most users generates sparse and short-lived requests (*e.g.*, Web requests). Considering our scenario, when the *idle timeout* is set to 120 seconds, for example, the average rules installed in the network is about 1042, being 77% of these rules inactive. Given that the switches' TCAM are an expensive resource to be wasted and the number of rules that can be stored in these memories is limited, it is important to control this configuration closely to avoid running out of resources.

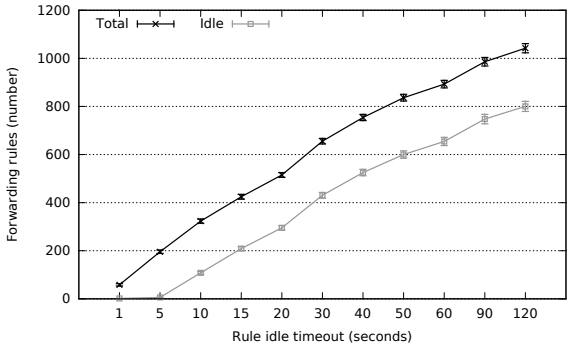


Fig. 5. Total/idle rules vs. *idle timeout* configuration

In summary, this analysis shows how a single factor configured at the controller can significantly affect the control channel traffic and resource consumption of an OpenFlow-based network. It is important to mention that we discovered a hard limit to collect statistics with *Read-State* messages on the controller implementation (in our experiment this limit was 682 rules). Thus, when a switch has more installed rules than the controller supports, collected statistic can be inaccurate.

IV. An Approach to Interactive SDN Management

In this section, we present our interactive approach to SDN management through monitoring, visualization, and configuration. First, in Section IV-A we detail components and how our approach lines up with the general SDN conceptual architecture. Second, in Section IV-B we present our prototype implementation and the extensions developed on the Floodlight controller to support advanced control channel management.

A. Conceptual Architecture

Figure 6 shows the SDN architecture along with the components added to enable management via monitoring, visualization, and configuration. To introduce concepts didactically, we organize the explanation in two steps. First, we detail the SDN architecture. Second, we explain all components of our approach and how the *Administrator* can interact with them.

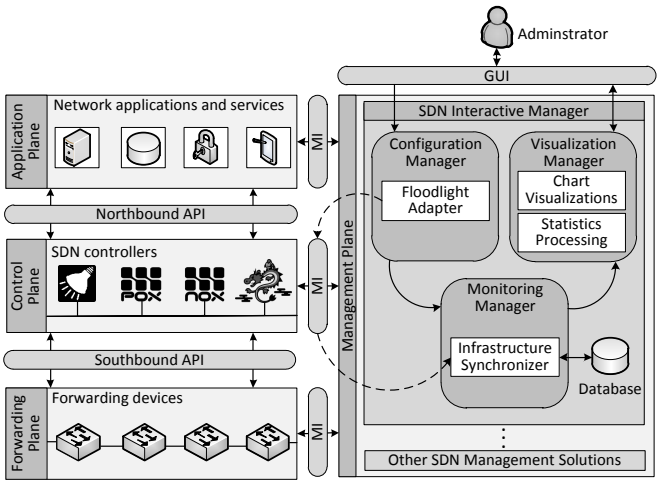


Fig. 6. Conceptual Architecture

Overall, SDN introduces an architecture with four planes: *management*, *application*, *control*, and *forwarding* [28]. All planes communicate with each other through interfaces. For example, the *management* plane uses a set of Management Interfaces (MI) to exchange information and to control elements in other planes. In addition, an interface called *northbound* API establishes bidirectional communication between *application* and *control* planes, while the *southbound* API does the same for *control* and *forwarding* planes. Ideally, all these interfaces should be standardized to allow easy replacement of devices and technologies. In practice, the OpenFlow protocol is the current *de facto* standard *southbound* API. All other interfaces are undergoing discussion and development.

Conceptually, each of the four planes has a set of specific functions that need to be fulfilled, as following explained:

- **Management Plane:** is responsible for managing elements in other SDN planes (*e.g.*, monitoring device status, allocating resources, and enforcing access control policies).
- **Application Plane:** contains one or more applications that can serve several different purposes (*e.g.*, firewall, circuit establisher, and load balancer). Each of these applications are granted with access to a set of resources by one or more SDN controllers.
- **Control Plane:** contains one or more controllers that coordinate network devices. At least an SDN controller needs to execute the requests coming from the application plane. Commonly, these controllers also include internal logic to handle network events and make traffic forwarding decisions (*e.g.*, the aforementioned hub, switch, and forwarding behaviors).
- **Forwarding Plane:** comprises a set of simple forwarding elements with transmission capacity and traffic processing resources. The notion of keeping forwarding elements simple and making decisions at higher software layers is central in SDN.

In this paper, we propose a solution called *SDN Interactive Manager* that includes three main components: *Monitoring Manager*, *Visualization Manager*, and *Configuration Manager*. The *SDN Interactive Manager* sits in the *management* plane of SDN alongside other existing or yet to be developed solutions. Since our approach to SDN management comprises interactive network management, we also depict in the architecture the *Administrator* who interacts primarily with the *Visualization Manager* and the *Configuration Manager* components through a *Graphical User Interface (GUI)*. Therefore, our solution creates a loop of activities by integrating these components with the *Administrator* interactions. Each of these components performs independent tasks described as follows.

Monitoring Manager – This component is responsible for retrieving updated information about the network and storing it in a local *Database*. This is performed mainly through a module called *Infrastructure Synchronizer*, which collects information, such as traffic statistics, network topology, and device data, by accessing an MI to one or more controllers situated in the *control* plane. Currently, there is no standard MI, though we envision that such interface could present at

least the same functionality as the *northbound* API. Also, customizations in this API could be added to support information relevant to network management (*e.g.*, control traffic counters and resource usage data). Then, the *Infrastructure Synchronizer* module stores both control and data traffic statistics, maintaining a history of network changes and SDN-related configurations performed by the *Administrator*.

Visualization Manager – This component comprises the *Statistics Processing* and *Chart Visualizations* modules. With information stored in the *Database*, the *Statistics Processing* module is able to aggregate data per host, switch, controller, or even the entire network to be used by the *Chart Visualizations* module. Furthermore, the *Statistics Processing* module is also able to identify which rules are active and which are idle on forwarding devices. To build interactive visualizations that can be analyzed by the *Administrator*, the *Chart Visualizations* module uses a rich library of interface components (*e.g.*, graphs, charts, and diagrams) that enables data updates in real time. Then, based on these visualizations, the *Administrator* can be aware of possible issues or bottlenecks and plan for adjustments and improvements in the network configuration.

Configuration Manager – This component allows the *Administrator* to check and configure SDN-related parameters on network controllers through the MI. The *Floodlight Adapter* module permits setting up the polling interval for monitoring network devices through a friendly GUI. Moreover, through the same GUI, the *Administrator* can trigger the *Floodlight Adapter* module to set the *idle timeout* to be configured globally, per device, or per flow. All SDN-related parameter configurations on the controller are performed using an extension of the *northbound* API implementation that is embedded in MI between *management* and *control* planes.

B. Prototype Implementation

We designed our prototype as a modular application with independent and integrated functionalities for SDN monitoring, visualization, and configuration. We followed the Model-View-Template (MVT) design pattern and we chose Python 2.7 with Django 1.6 [29] as development framework. The *Database* to store network devices information as well as traffic statistics has been implemented as Django Models. Each of monitoring, visualization, and configuration functionalities were developed as Views and the GUI as Templates. Moreover, we also integrated our prototype with Floodlight, which is a Java-based SDN controller, supported by a community of developers and engineers of Big Switch Networks [16]. As a consequence of our unusual requirements, we had to implement a module for the Floodlight controller to support advanced management features, such as reporting control traffic statistics and dynamically configuring the *idle timeout* of forwarding rules.

Regarding the *Monitoring Manager* component, our implementation focused on periodically updating network information to the *Database*. For this purpose, we used the RESTful API provided by the Floodlight controller. From this API one is able to access information about the physical topology, including links, switches, and hosts. Moreover, the *Monitoring Manager* also gathers data traffic counters of every rule installed on each switch of the topology. However, by default, the Floodlight controller does not address control traffic counters. Therefore, we developed a module for Floodlight controller, which we called *Control Statistics Aggregator*, to gather these counters and extended the RESTful API to report them. The *Control Statistics Aggregator* module watches control channel connections to inspect and count OpenFlow messages generated either by the controller or switches. This module gathers all these control messages, keeps separated counters (number of packets and packet lengths) by device, message type, and sub-type.

The *Visualization Manager* component is implemented as a Web based application relying mainly on the D3.js library [30]. The major purpose of this component is to provide an interactive way for the administrator to understand the current network status and visualize the impact of his/her configurations in real time. Thus, as the network is periodically monitored by the *Monitoring Manager* component, visualizations are updated in the same pace. The integration between these two components allows our prototype to display network visualizations, such as: topology view with intuitive hints on where resource bottlenecks might be occurring, charts of idle and active rules installed on forwarding devices, amount of OpenFlow messages flowing in control channels, and the traffic rate these messages generate.

The implementation of the *Configuration Manager* aims to provide an easy way for the *Administrator* to change the network configurations through the same GUI where visualizations are displayed. For that purpose, we developed a module as an extension of the Floodlight default RESTful API so that configuration parameters can be sent from our prototype to the controller. Our prototype currently supports only the configuration of the forwarding rule *idle timeout* parameter, but the implementation can be easily extended to support others. Another parameter that can be set is the polling interval of the *Infrastructure Synchronizer* module. This parameter affects the frequency of reading information from counters of network devices. The impact of these parameters on the control channel load and resource consumption, as well as how visualizations reflect their changes are presented in the next section.

V. Evaluation

In this section, we describe the evaluation of our approach using the developed prototype. Our goal is to measure control channel load and resource usage considering the administrator interactions over the experiment timespan. To understand the impact of changing SDN-related parameters, we simulated some administrator interactions to control the campus topology, workload, and controller behavior presented in Section III. In addition to varying the rule *idle timeout* configuration, we also change the monitoring polling interval to understand the impact of *Read-State* messages as well. Table II presents all configurations changes simulated during the evaluation period.

TABLE II. Simulated administrator interactions

Parameter	**Value**					
Reconfiguration time (hour:minutes)	11:05	11:12	11:25	11:37	11:47	11:57
Rule idle timeout (seconds)	5	60	30	30	30	30
Monitoring interval (seconds)	5	5	5	40	30	15

Figure 7 depicts the user-friendly Web interface developed in order to provide the administrator with interactive visualizations and to allow easy configuration of SDN-related

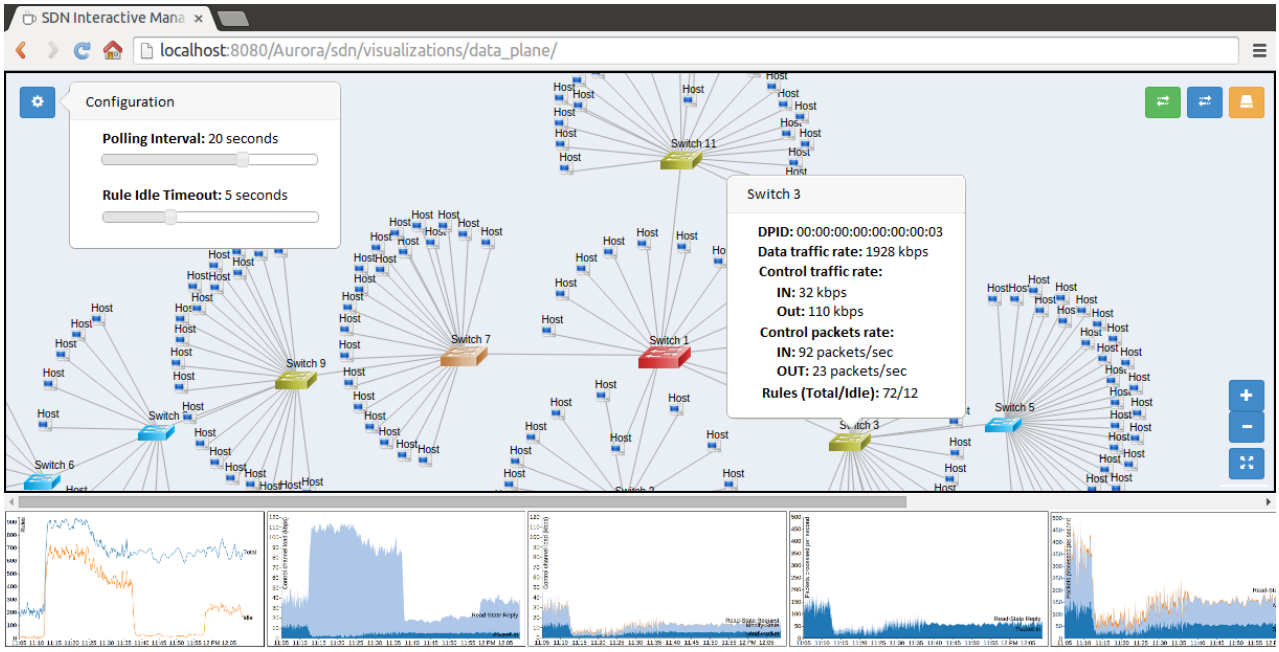


Fig. 7. Web interface of the prototype developed

parameters. The visualization of the physical topology allows to alternate between three different perspectives (from the top-right corner): data traffic, control traffic, and resource usage. Depending on the active perspective, different sizes and colors of switches and hosts, as well as link widths and colors, are used to represent different levels of resource usage, control traffic, and data traffic. On the top-left corner of the Web interface two configuration parameters can be adjusted: *polling interval* for monitoring and *idle timeout* of forwarding rules. Below the physical topology visualization, we placed interactive charts showing online resource usage in terms of installed rules (active and idle), traffic rates, and packet processing rates. In the example of Figure 7, these charts display results for the aggregated control traffic of the whole network. However, by selecting a device the administrator is also able to filter this information per switch or host.

Figures 8 to 10 present the interactive charts available from the Web interface of our prototype in more detail. These charts present the total and idle rules (Figure 8), control traffic rates in kbps from the controller to switches (Figure 9(a)) and *vice versa* (Figure 9(b)), and control packets processed per second also in both ways (Figures 10(a) and 10(b)). These charts show information for the whole network during the timespan of the experiment (1 hour). Vertical dashed lines mark the moments when the administrator changes a configuration (see Table II).

At the beginning of the experiment, the amount of installed rules is approximately 200, while nearly zero are idle, as shown in Figure 8. As mentioned before, a rule is considered idle when its counters do not change between two monitoring polls. This small number of idle rules is a consequence of the low rule *idle timeout* value set to 5 seconds (default Floodlight configuration). On the other hand, the controller is processing a large number of *Packet-In* messages, as displayed in Figure 10(b). To reduce the load on the controller, at 11:12, the administrator changes the rule *idle timeout* to 60 seconds. After this change, it is possible to visualize a dramatic decrease in control packets processed both by the controller and network devices. However, this configuration also affects immediately resource consumption, specially in terms of idle rules (Figure 8) and control traffic rate towards the controller (Figure 9(b)). With this rule *idle timeout* configuration nearly 77.7% of all forwarding rules remain idle and upload control traffic increases almost threefold.

Close to 11:25, the administrator decreases the rule *idle timeout* to 30 seconds as an attempt to bring down traffic in the control channel and the amount of forwarding rules installed. From Figures 8 and 9(b) it is possible to visualize such configuration does indeed decrease these values over time. Nevertheless, instead of observing a hard drop in traffic rate and installed rules, this time we notice a gradual decrease. This behavior can be explained because the new *idle timeout* settings will only be applied to newly installed rules. Thus, rules installed before the change will respect the previous configuration. Also, packet processing rates tend to increase again with this configuration, but nowhere near the values from the beginning of the experiment.

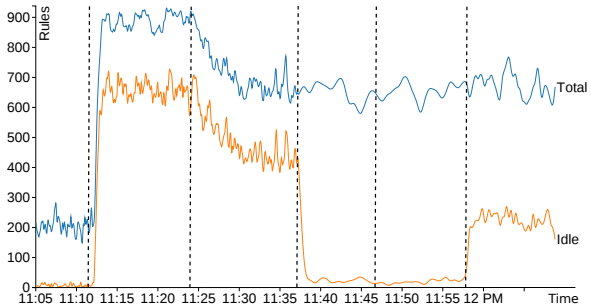
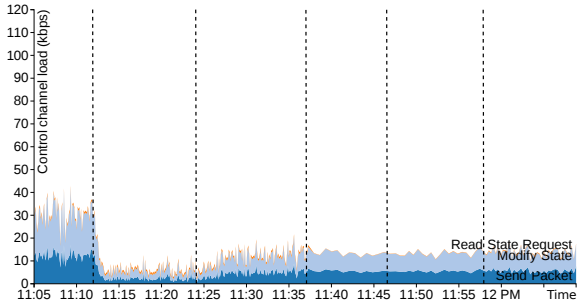
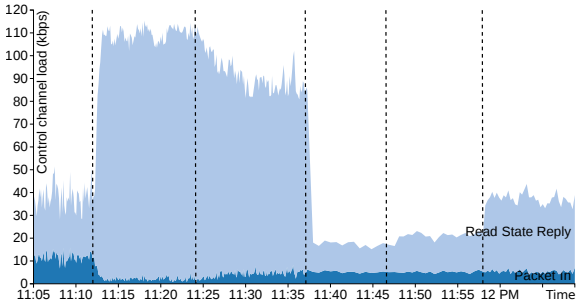


Fig. 8. Total and idle rules behavior during the experiment duration

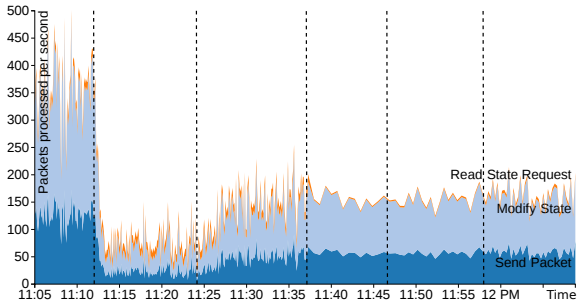


(a) Download control traffic rate over the experiment duration

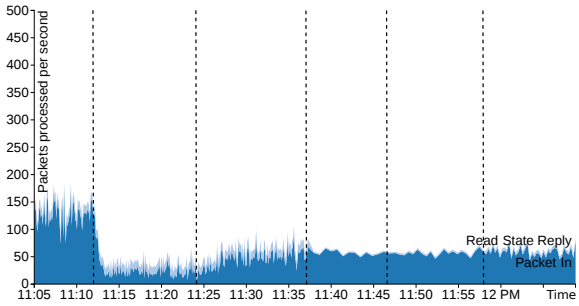


(b) Upload control traffic rate over the experiment duration

Fig. 9. Control channel traffic rates over the experiment duration



(a) Download packet rate over the experiment duration



(b) Upload packet rate over the experiment duration

Fig. 10. Control channel packet rates over the experiment duration

After the first two changes, the amount of traffic in the control channel towards the controller remains very high (Figure 9(b)). Most of this traffic is due to *Read-State* reply messages, which are loaded with counters for as many rules as are installed in all switches. To decrease the amount of *Read-State* messages, the administrator increases the monitoring polling interval to 40 seconds at roughly 11:37. Immediately after this change, we can visualize that the control traffic rate generated by these messages is significantly reduced. However, looking at Figure 8, we are also able to notice an unexpected decrease in the amount of idle rules installed. This behavior is actually a distortion or loss of precision in monitoring given the way of identifying for idle rules. To be considered idle a rule needs to be monitored twice without changing its counters, which will rarely happen with too sparse monitoring polls. Finally, the administrator decreases the polling interval to 30 seconds (at 11:47), which is still insufficient to capture idle rules, and to 15 seconds (at 11:57), when the monitoring process returns to identify idle rules.

VI. Conclusion and Future Work

Although network monitoring, visualization, and configuration are common management activities, in the context of SDN, these activities can be considerably different from traditional networks and deserve proper attention. The SDN controller behavior, for example, can be customized by network administrators, which might affect resource consumption and traffic forwarding performance. In this paper, we initially presented an analysis of the control channel traffic in OpenFlow networks to verify the impact of specific SDN-related parameters and their influence in the overall network resource consumption. Then, we have proposed an interactive approach that integrates SDN monitoring, visualization, and configuration activities, allowing the administrator to interact with and better understand the network. Moreover, we also developed a prototype as a proof-of-concept and evaluated our approach to SDN management simulating the administrator interactions.

By analyzing the control channel in OpenFlow-based SDN, we showed how resource usage and control channel load are affected by the configuration of SDN-related parameters (*i.e.*, *idle timeout* of forwarding rules). Our developed prototype included a monitoring component that retrieves statistics of control channel traffic, a feature which most SDN management approaches do not consider. Moreover, our approach allowed the administrator to easily visualize the number of packets processed by both controller and forwarding devices, the control channel load also in both directions, and the proportion of idle rules installed on forwarding devices. Based on this information, administrators are able identify potential issues and change configurations of SDN parameters using our interactive Web interface to achieve their specific goals.

In future investigations, we plan to perform more experiments with other SDN-related parameters and different controller implementations. We also plan to perform experiments with other versions of the OpenFlow protocol, which have different control messages and data structures. Finally, we intend to provide more configuration possibilities on the prototype, such as thresholds to an algorithm that can perform automated reconfigurations on behalf of the administrator.

Acknowledgment

This work is supported by FAPERGS, a research project named SDN Man – Gerenciamento de Redes Definidas por Software # 001/2013 PqG.

REFERENCES

[1] C. Rothenberg, R. Chua, J. Bailey, M. Winter, C. Correa, S. de Lucena, M. Salvador, and T. Nadeau, "When Open Source Meets Network Control Planes," *Computer*, vol. 47, no. 11, pp. 46–54, November 2014.

[2] Open Networking Foundation, "Software-Defined Networking (SDN) Definition," 2014, available at: <https://www.opennetworking.org/sdn-resources/sdn-definition>. Accessed: January 2015.

[3] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, February 2013.

[4] E. Salvador and L. Granville, "Using Visualization Techniques for SNMP Traffic Analyses," in *Proceedings of the 13th IEEE Symposium on Computers and Communications (ISCC)*, July 2008, pp. 806–811.

[5] P. Barbosa and L. Granville, "Interactive SNMP Traffic Analysis Through Information Visualization," in *Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2010, pp. 73–79.

[6] V. T. Guimaraes, G. L. Santos, G. C. Rodrigues, L. Z. Granville, and L. M. R. Tarouco, "A Collaborative Solution for SNMP Traces Visualization," in *Proceedings of the 29th International Conference on Information Networking (ICOIN)*, February 2014, pp. 458–463.

[7] L. Bondan, M. Marotta, M. Kist, L. Roveda Faganello, C. Bonato Both, J. Rochol, and L. Zambenedetti Granville, "Kitsune: A Management System for Cognitive Radio Networks Based on Spectrum Sensing," in *Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–9.

[8] C. C. Machado, L. Z. Granville, A. Schaeffer-Filho, and J. A. Wickboldt, "Towards SLA Policy Refinement for QoS Management in Software-Defined Networking," in *Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, May 2014, pp. 397–404.

[9] J. Wickboldt, W. De Jesus, P. Isolani, C. Bonato Both, J. Rochol, and L. Zambenedetti Granville, "Software-Defined Networking: Management Requirements and Challenges," *IEEE Communications Magazine - Network & Service Management Series*, vol. 53, no. 1, pp. 278–285, January 2015.

[10] Y. Zhang, "An Adaptive Flow Counting Method for Anomaly Detection in SDN," in *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, December 2013, pp. 25–30.

[11] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost," in *Proceedings of the 14th International Conference on Passive and Active Measurement (PAM)*, March 2013, pp. 31–41.

[12] S. Chowdhury, M. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for Software Defined Networks," in *Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–9.

[13] L. Jose, M. Yu, and J. Rexford, "Online Measurement of Large Traffic Aggregates on Commodity Switches," in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, October 2011, pp. 13–13.

[14] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks," in *Proceedings of the 11th International Conference on Passive and Active Measurement (PAM)*, April 2010, pp. 201–210.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, March 2008.

[16] Big Switch Networks, "Floodlight an Open SDN Controller," 2014, available at: <https://www.projectfloodlight.org/floodlight/>. Accessed: January 2014.

[17] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*, October 2010, pp. 19:1–19:6.

[18] Open Networking Foundation, "OpenFlow Switch Specification - Version 1.0.0 (Wire Protocol 0x01)," 2009, available at: <https://www.opennetworking.org/sdn-resources/technical-library>. Accessed: January 2015.

[19] D. Klein and M. Jarschel, "An OpenFlow Extension for the OMNeT++ INET Framework," in *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques (SIMUTools)*, March 2013, pp. 322–329.

[20] S. Hassas Yeganeh and Y. Ganjali, "Turning the Tortoise to the Hare: An Alternative Perspective on Event Handling in SDN," in *Proceedings of the 2014 ACM International Workshop on Software-defined Ecosystems (BigSystem)*, June 2014, pp. 29–32.

[21] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, August 2011, pp. 254–265.

[22] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable Flow-based Networking with DIFANE," in *Proceedings of the ACM SIGCOMM 2010 Conference*, August 2010, pp. 351–362.

[23] Cisco Systems, "Cisco Visual Networking Index: Forecast and Methodology, 2013–2018," 2014, available at: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html. Acessed: January 2015.

[24] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Y. Jin, "ReSurf: Reconstructing Web-Surfing Activity from Network Traffic," in *IFIP Networking Conference*, May 2013, pp. 1–9.

[25] X. Cheng, J. Liu, and C. Dale, "Understanding the Characteristics of Internet Short Video Sharing: A YouTube-Based Measurement Study," *IEEE Transactions on Multimedia*, vol. 15, no. 5, pp. 1184–1194, August 2013.

[26] K. Katsaros, G. Xylomenos, and G. Polyzos, "GlobeTraff: a traffic workload generator for the performance evaluation of future Internet architectures," in *Proceedings of the 5th International Conference on New Technologies, Mobility and Security (NTMS)*, May 2012, pp. 1–5.

[27] 3GPP2, "CDMA2000 Evaluation Methodology," 2004, available at: <http://www.3gpp2.org/Public_html/specs/C.R1002-0_v1.0_041221.pdf>. Accessed: January 2015.

[28] Open Networking Foundation, "SDN Architecture 1.0," 2014, available at: <https://www.opennetworking.org/sdn-resources/technical-library>. Accessed: January 2015.

[29] Django, "Django: the Web framework for perfectionists with deadlines," 2015, available at: <https://www.djangoproject.com/>. Accessed: January 2015.

[30] D3.js, "D3: Data-Driven-Documents," 2015, available at: <https://www.d3js.org/>. Accessed: January 2015.

AppendixC PUBLISHED DEMO – IM 2015

In this 2-page paper it was published an demonstration of our developed prototype named SDN Interactive Manager. Our prototype is a management approach through monitoring, visualization, and configuration of OpenFlow-based SDN that we have already an accepted paper on 14th IFIP/IEEE International Symposium on Integrated Network Management (IM - 2015). We developed this prototype to considering using the monitoring information to present interactive visualizations an allow to the administrator configure/reconfigure the network through an GUI. We used the same metrics of control channel load and resource consumption to show to the overhead, imposed by control messages, to the administrator.

- **Title –**
  SDN Interactive Manager: An OpenFlow-Based SDN Manager
- **Symposium –**
  14th IFIP/IEEE International Symposium on Integrated Network Management (IM - 2015)
- **Type –**
  Demo track (2-page paper)
- **Qualis –**
  B1
- **URL –**
  <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=XXXXX>
- **Date –**
  May 11-15, 2015
- **Held at –**
  Ottawa, Canada
- **Digital Object Identifier (DOI) –**
  <http://dx.doi.org/XX.XXXX/IM.2015.X>

SDN Interactive Manager: An OpenFlow-Based SDN Manager

Pedro Heleno Isolani*, Juliano Araujo Wickboldt*, Cristiano Bonato Both†,
Juergen Rochol*, and Lisandro Zambenedetti Granville*
*Federal University of Rio Grande do Sul, Porto Alegre, Brasil
†University of Santa Cruz do Sul, Santa Cruz do Sul, Brasil
Email: {phisolani, jwickboldt, juergen, granville}@inf.ufrgs.br, cboth@unisc.br

I. Related Work

Currently, many investigations addressed SDN management considering using monitoring information for different purposes. Zhang [1] used monitoring information to develop algorithms for anomaly detection and traffic engineering. Jose *et al*. [2] used the same monitoring information to propose mechanisms for online measurement of large traffic aggregates. Yu *et al*. [3] proposed FlowSense, which is aimed to keep the lowest control channel overhead and the highest information accuracy as possible in OpenFlow monitoring. Chowdhury *et al*. addressed SDN monitoring from a different perspective. The authors proposed Payless [4] framework that is able to deal with monitoring considering the polling frequency and data granularity. Payless relies on OpenTM [5] to select only important switches to be monitored, also aiming to reduce the overhead imposed in the control channel. Thus, this framework allows to adjust the polling frequency parameter to balance the control channel overhead, imposed by monitoring messages, and accuracy of monitored information.

The aforementioned proposals are focused on the use of monitoring information to automate tasks, such as reducing control traffic overhead and protecting the network. No previous investigation aims to employ monitoring information to help the administrator understanding the network behavior nor interact with it. To the best of our knowledge, there are no solutions that leverage OpenFlow monitoring messages to create network visualizations and address the interactive configuration of SDN-related parameters. Before presenting how we approached such a problem, we present our motivation and all objectives to archieve with our propototype, focusing on didactly detach important central points of our solution.

II. Motivation and Objectives

In summary, SDN management activities are considerably different from traditional networks, thus deserving proper attention. The SDN controller customizations might impact in terms of resource consumption and traffic forwarding performance. As a consequence, such impact is difficult to asses because traditional network management solutions were not designed to cope in the context of SDN. Moreover, several solutions deal with SDN monitoring, visualization, or configuration for automated tasks (*e.g.*, reduce control traffic or network protection) and they not aim to help the administrator to understand the network behavior and interact with it. Therefore, our objectives in building our SDN management prototype are the following:

- Perform SDN monitoring with configurable polling intervals specifically focusing on inspect resource usage and control channel load metrics.
- Present monitoring information obtained in interactive visualizations that emphasize these metrics.
- Support the administrator interaction by configuring and reconfiguring SDN-related parameters.

III. SDN Interactive Manager

In this section, we describe our prototype to manage SDN through monitoring, visualization, and configuration. Our prototype is an Web application named *SDN Interactive Manager* that allows the administrator to understand and control the network behavior through a *Graphical User Interface* (GUI).

Figure 1 shows the SDN architecture along with our prototype. Overall, SDN introduces an architecture with four planes: *management*, *application*, *control*, and *forwarding*. All SDN planes communicate with each other through interfaces. For example, the *management* plane uses a set of *Management Interfaces* (MI) to exchange information and to control elements in other planes. In addition, an *northbound* API interface establishes bidirectional communication between *application* and *control* planes, while the *southbound* API does the same for *control* and *forwarding* planes.

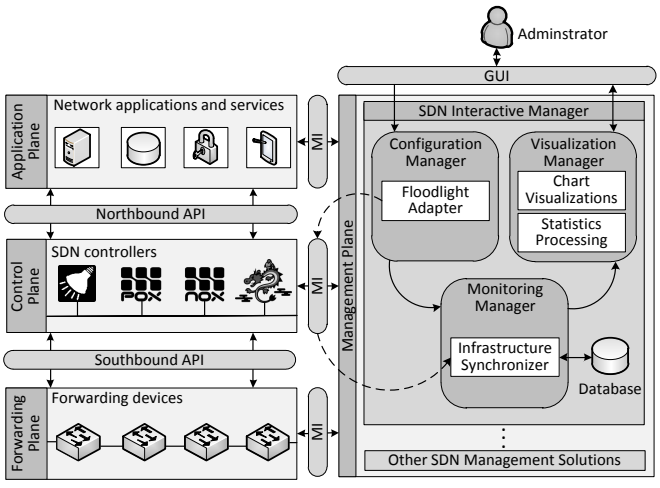


Fig. 1. Conceptual Architecture

978-3-901882-76-0 @2015 IFIP

Our prototype includes three main components: *Monitoring Manager*, *Visualization Manager*, and *Configuration Manager*. The *SDN Interactive Manager* sits in the *management* plane of SDN alongside other existing or yet to be developed solutions. Since our approach to SDN management comprises interactive network management, we also depict in the architecture the *Administrator* who interacts primarily with the *Visualization Manager* and the *Configuration Manager* components through an GUI (shown in Figure 2). Therefore, our solution creates a loop of activities by integrating these components with the *Administrator* interactions. Each of these components performs independent tasks described as follows.

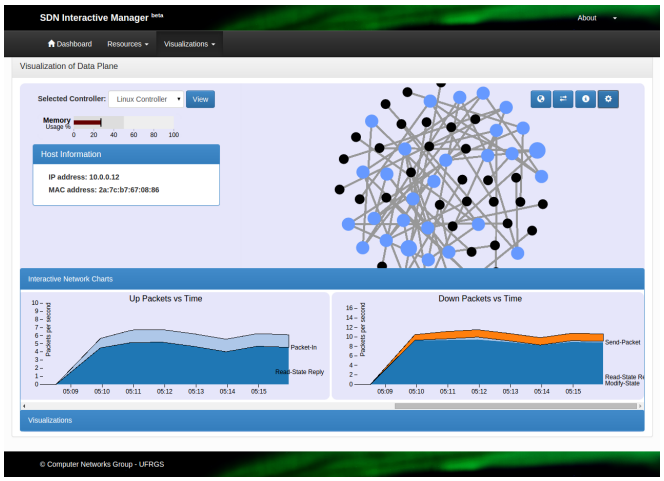


Fig. 2. SDN Interactive Manager GUI

Monitoring Manager – Responsible for retrieving updated information about the network and storing it in a local *Database*. The *Infrastructure Synchronizer* module collects information, such as traffic statistics, network topology, and device data, by accessing an MI to one or more controllers situated in the *control* plane. In this case, we used the RESTful API provided by the Floodlight controller to access information about the physical topology, including links, switches, and hosts. Moreover, the *Monitoring Manager* also gathers data traffic counters of every rule installed on each switch in the network. However, by default, the Floodlight controller does not address control traffic counters. Therefore, we developed a module for Floodlight controller, which we called *Control Statistics Aggregator*, to gather these counters and extended the RESTful API to report them. The *Control Statistics Aggregator* module monitors control channel connections to inspect and count OpenFlow messages generated either by the controller or switches.

Visualization Manager – Comprises the *Statistics Processing* and *Chart Visualizations* modules. With information stored in the *Database*, the *Statistics Processing* module is able to aggregate data per host, switch, controller, or even the entire network to be used by the *Chart Visualizations* module. Furthermore, the *Statistics Processing* module is also able to identify which rules are active and which are idle on forwarding devices. To build interactive visualizations that can be analyzed by the *Administrator*, the *Chart Visualizations* module uses a rich library of interface components (*e.g.*, charts and diagrams) that enables data updates in real time. This component is implemented as an interactive Web based application relying mainly on the D3.js library. Thus, as the network is periodically monitored by the *Monitoring Manager* component, visualizations are updated in the same pace. The integration between these two components allows our prototype to display network visualizations, such as: topology view with intuitive hints on where resource bottlenecks might be occurring, charts of idle and active rules installed on forwarding devices, amount of OpenFlow messages flowing in control channels, and the traffic rate these messages generate.

Configuration Manager – Allows the *Administrator* to check and configure SDN-related parameters on network controllers through the MI. The *Floodlight Adapter* module permits setting up the polling interval for monitoring network devices through a friendly GUI. Moreover, the *Administrator* can trigger the *Floodlight Adapter* module to set the *idle timeout* to be configured on rule installation process. All SDN-related parameter configurations on the controller are performed using an extension of the *northbound* API implementation that is embedded in MI between *management* and *control* planes. We developed *Floodlight Adapter* module to support the configuration of the forwarding rule *idle timeout* parameter, but the implementation can be easily extended to support others. Another parameter that can be set is the polling interval of the *Infrastructure Synchronizer* module. This parameter affects the frequency of reading information from counters of network devices and also impacts on the control channel load generated in both directions.

Our prototype was designed following the Model-View-Template (MVT) design pattern and we chose Python 2.7 with Django 1.6 as development framework. The *Database* used to store network devices information as well as traffic statistics is PostgreSQL 9.3.5 and has been implemented as Django Models. Each of monitoring, visualization, and configuration functionalities were developed as Views and the GUI as Templates. As a consequence of our unusual requirements, we had to implement a module for the Floodlight controller to support our advanced management features, such as reporting control traffic statistics and dynamically configuring the *idle timeout* of forwarding rules.

References

[1] Y. Zhang, "An Adaptive Flow Counting Method for Anomaly Detection in SDN," in *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, December 2013, pp. 25–30.

[2] L. Jose, M. Yu, and J. Rexford, "Online Measurement of Large Traffic Aggregates on Commodity Switches," in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, October 2011, pp. 13–13.

[3] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost," in *Proceedings of the 14th International Conference on Passive and Active Measurement (PAM)*, March 2013, pp. 31–41.

[4] S. Chowdhury, M. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for Software Defined Networks," in *Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–9.

[5] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks," in *Proceedings of the 11th International Conference on Passive and Active Measurement (PAM)*, April 2010, pp. 201–210.

AppendixD DEMO POSTER – IM 2015

In this poster is highlighted main aspects of our developed prototype to manage SDN through monitoring, visualization and configuration activities. We focus on explain the motivation, objectives, the Floodlight controller modifications, our prototype – SDN Interactive Manager – implementation, and then show our GUI interface. Our prototype is a management approach through monitoring, visualization, and configuration of OpenFlow-based SDN that we have already an accepted paper on 14th IFIP/IEEE International Symposium on Integrated Network Management (IM - 2015). We developed this prototype to considering using the monitoring information to present interactive visualizations an allow to the administrator configure/reconfigure the network through an GUI. We used the same metrics of control channel load and resource consumption to show to the overhead, imposed by control messages, to the administrator.

- **Title –**
  SDN Interactive Manager: An OpenFlow-Based SDN Manager
- **Symposium –**
  14th IFIP/IEEE International Symposium on Integrated Network Management (IM - 2015)
- **Type –**
  Demo track (2-page paper)
- **Qualis –**
  B1
- **URL –**
  <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=XXXXX>
- **Date –**
  May 11-15, 2015
- **Held at –**
  Ottawa, Canada
- **Digital Object Identifier (DOI) –**
  <http://dx.doi.org/XX.XXXX/IM.2015.X>

SDN Interactive Manager: An OpenFlow-Based SDN Manager

Federal University of Rio Grande do Sul
Institute of Informatics

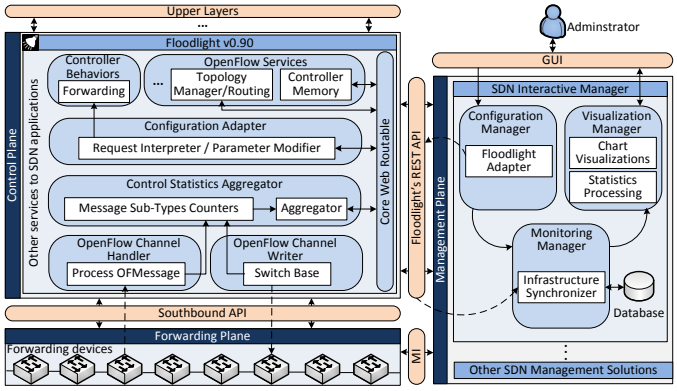
Introduction

Currently, many investigations addressed SDN management using monitoring information for different purposes. These proposals are focused on the use of monitoring information to automate tasks, such as reducing control traffic overhead and protecting the network. No previous investigation aims to employ monitoring information to help the administrator understanding the network behavior nor interact with it.

Objectives

I. Perform SDN monitoring with configurable polling intervals specifically focusing on inspecting resource usage and control channel load metrics.
II. Present monitoring information obtained in interactive visualizations that emphasize these metrics.
III. Support the administrator interaction by configuring and reconfiguring SDN-related parameters.

Controller Modifications



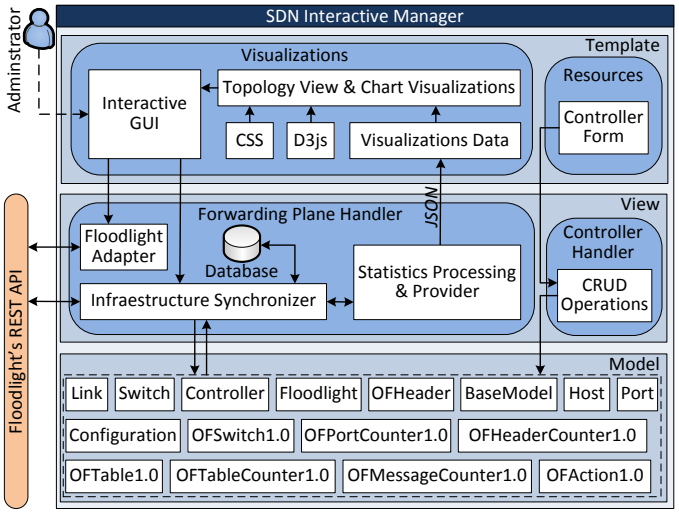
Motivation

In summary, SDN management activities are considerably different from traditional networks, thus deserving proper attention. The SDN controller customizations might impact in terms of resource consumption and traffic forwarding performance.

As a consequence, such impact is difficult to assess because traditional network management solutions were not designed to cope with SDN.

Moreover, several solutions deal with SDN monitoring, visualization, or configuration for automated tasks (e.g., reduce control traffic or network protection) and they not aim to help the administrator to understand the network behavior and interact with it.

Prototype Implementation



Prototype GUI

Pedro Heleno Isolani[1], Juliano Araujo Wickboldt[1], Cristiano Bonato Both[2], Juergen Rochol[1], and Lisandro Zambenedetti Granville[1]
[1]Federal University of Rio Grande do Sul, Porto Alegre, Brasil
[2]University of Santa Cruz do Sul, Santa Cruz do Sul, Brasil
Email: {phisolani, jwickboldt, juergen, granville}@inf.ufrgs.br, cboth@unisc.br