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.

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.

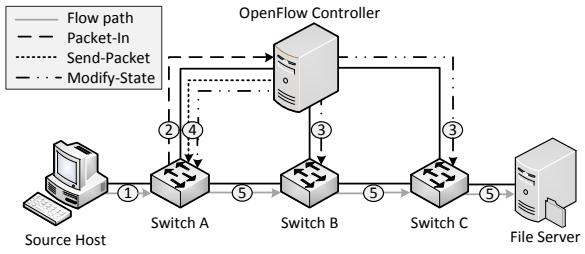


Fig. 1. OpenFlow controller *Forwarding* behavior example

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.

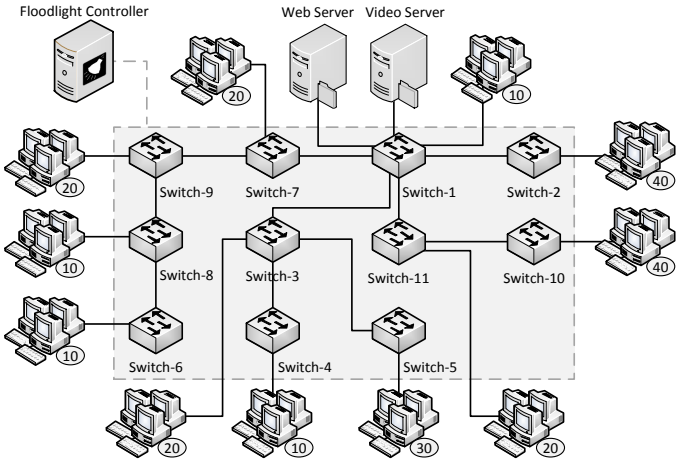


Fig. 2. Campus network topology

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.

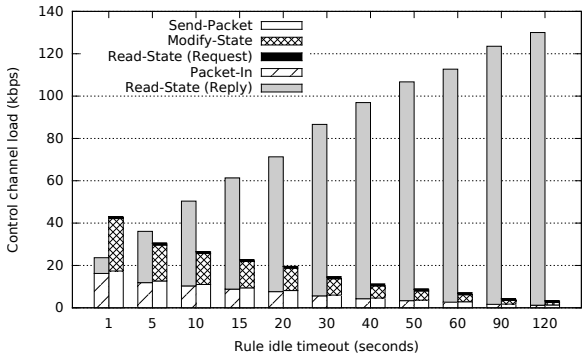


Fig. 3. Control channel load vs. *idle timeout* configuration

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.

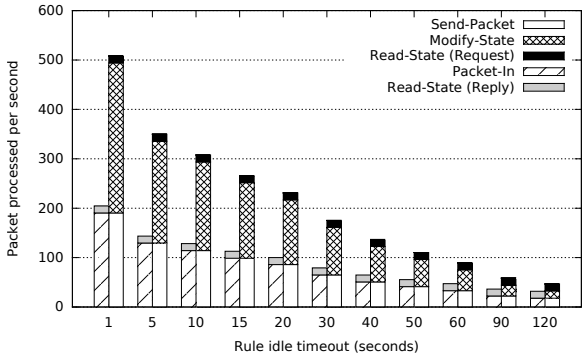


Fig. 4. Packets processed per second vs. *idle timeout* configuration

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.

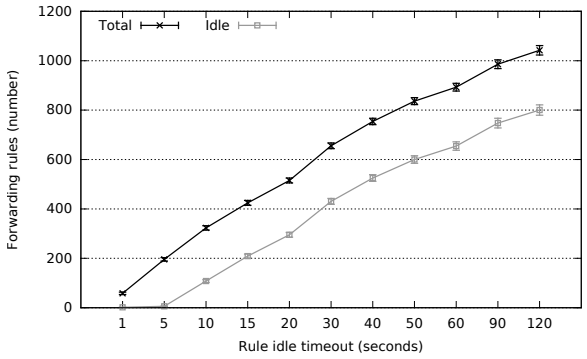


Fig. 5. Total/idle rules vs. *idle timeout* configuration

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.

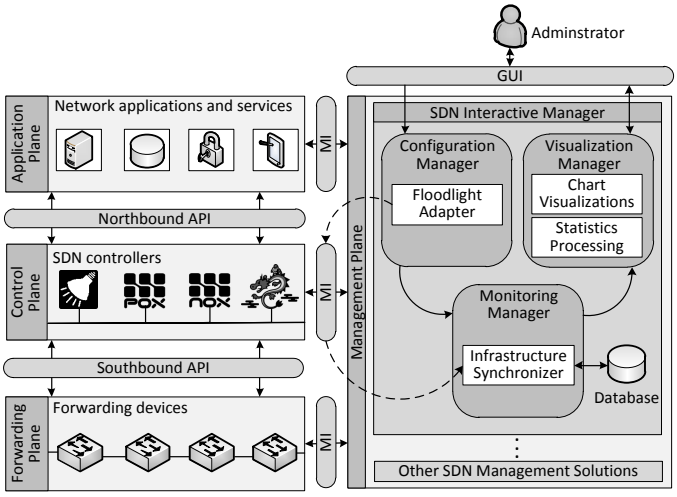


Fig. 6. Conceptual Architecture

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

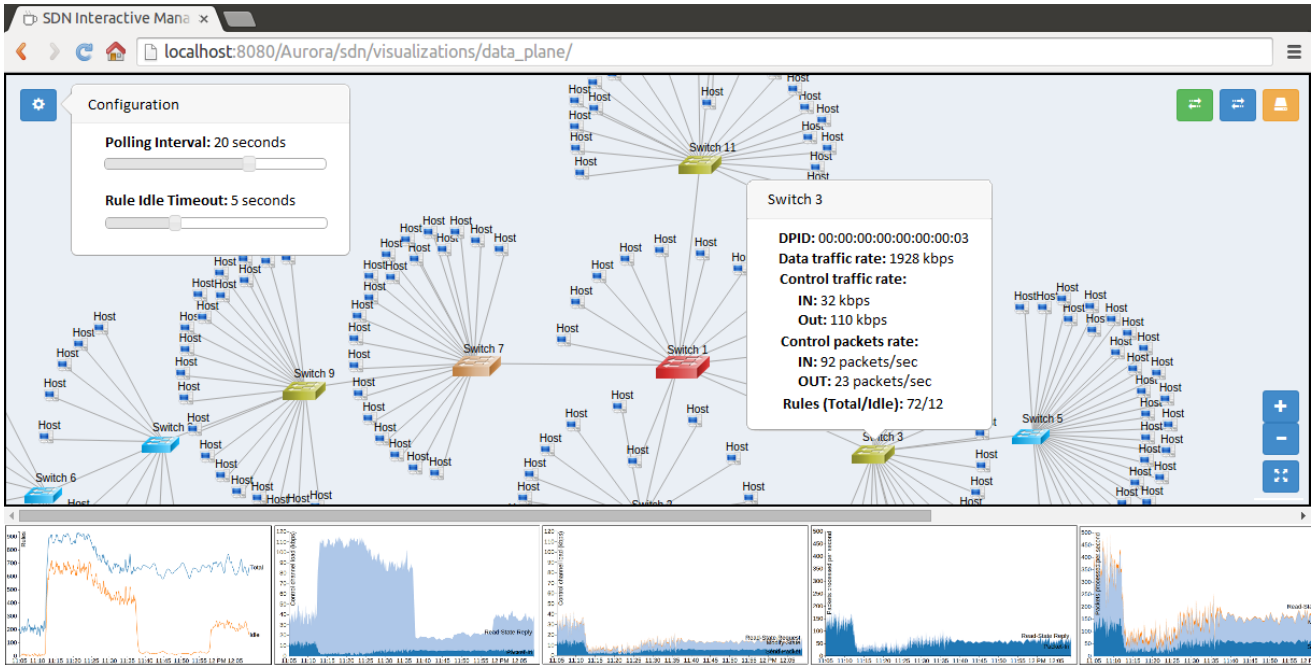


Fig. 7. Web interface of the prototype developed

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.

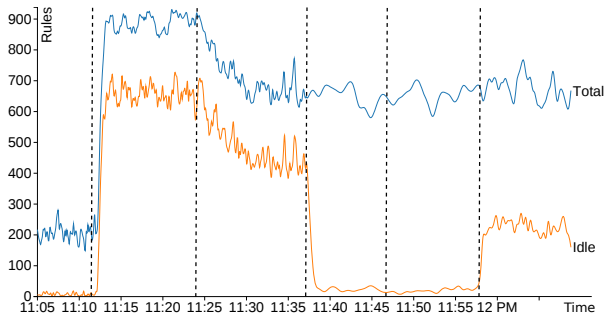
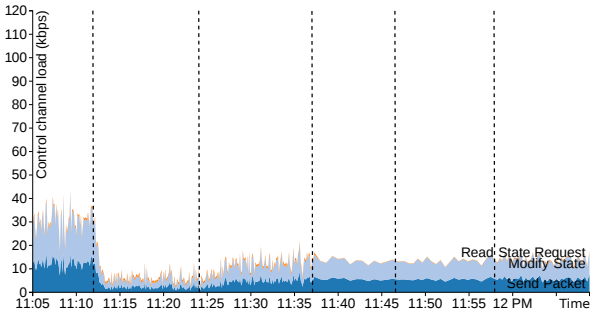
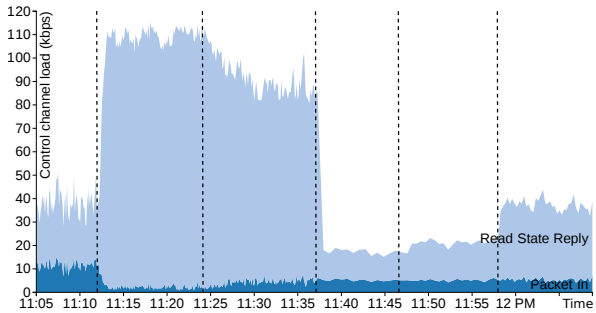


Fig. 8. Total and idle rules behavior during the experiment duration

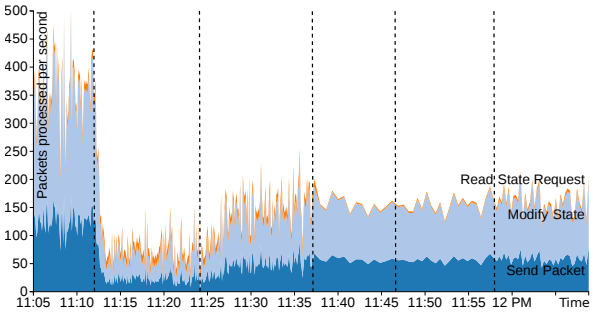


(a) Download control traffic rate over the experiment duration

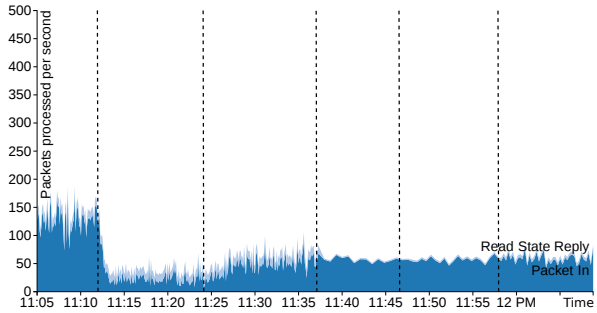


(b) Upload control traffic rate over the experiment duration

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.