Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 110 (2017) 516–523

Procedia Computer Science

www.elsevier.com/locate/procedia

International Workshop on Applications of Software-Defined Networking in Cloud Computing (SDNCC)

SOFTmon - Traffic Monitoring for SDN

Marc Hartung[a], Marc Körner[b]

[a]FernUniversität in Hagen, Universitätsstraße 11, 58084 Hagen, Germany
[b]International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, CA 94704-1198, USA

Abstract

Software Defined Networking (SDN) substrates are basic enabler for the network virtualization. They provide many opportunities but also require new solutions for well known legacy mechanisms. Thus, in this paper we present an innovative network monitoring tool which is able to operate with the usual available OpenFlow controllers. The presented tool extends the controller monitoring capabilities by providing utilization charts and statistics up to a flow level. In order to present the feature set, the tools architecture and implementation will be introduced. Further, an evaluation on a virtualized Mininet network using Open vSwitch is presented as well as an evaluation on our SDN research cluster with a typical data center fat tree topology composed out of NEC IP 8800 switches.

Peer-review under responsibility of the Conference Program Chairs.

Keywords: Software Defined Networking; OpenFlow; Monitoring; Testbed

1. Introduction

Since the invention of computer networks, their monitoring had always played a central role regarding the performance management. It is necessary to identify important status parameter in order to determine the network's health status. Network monitoring in particular basically means frequently measuring and observing several network related parameters like the utilized bandwidth or the latency between nodes. Moreover, also parameters that are based on network nodes like link failures or packet drops are very important indicators if everything in the network is working as expected. Network monitoring helps to avoid congestion and to identify architectural bottlenecks. Furthermore, physical failures due to broken cables or offline compute and network nodes can also be identified and resolved promptly. These examples demonstrate clearly how important network monitoring is. Especially today's network complexity and scale makes monitoring indispensable. In detail, the growing complexity of data centers and networks in gen-

* Corresponding author.
E-mail address: marc.hartung@studium.fernuni-hagen.de

1877-0509
Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2017.06.138

eral, makes it essential to have a contemporary monitoring solution. In addition, virtualization based on the Cloud Computing (CC) and the SDN paradigm are redefining the challenges for network monitoring on a daily basis.

Thus, we would like to introduce our open source SDN monitoring tool SOFTmon. This tool is designed to deliver a Network Operating System (NOS) independent monitoring solution with extended capabilities. Therefore, it extends the conventional NOS based monitoring capabilities by providing additional graphical transmission charts with several utilization information on switch, port, and even on a flow level. For instance, it supports the opportunity to differentiate several IP related flows and their load in context of the overall network utilization and capabilities.

The remaining paper is structured as follows. In section 2 some background information is given and some related work is presented. This is followed by the tool's architecture as described in section 3 and the prototypical implementation in section 4. Finally, an evaluation is presented in section 5, which is followed by a brief conclusion in section 7.

2. Background and related work

Monitoring in general is a very important and almost underestimated topic. Networks are usually monitored with several different mechanism. For instance, the host based latency measurements via Internet Control Message Protocol (ICMP) or network node based queries via the Simple Network Management Protocol (SNMP). However, these applications need to be configured and tested in a decentralized way. Thus, a centralized monitoring server component, like e.g. Zabbix [1] or Nagios [2], is required. This monitoring server collects, analyze, and visualize the frequently obtained information. Although these particular mechanisms can also be applied to SDN networks, technologies like the OpenFlow protocol provide and support direct access to the network nodes and a variety of statistic information. Thus, a monitoring solution utilizing SDN would be more powerful and would also provide a lot more new opportunities to gather network information, for instance, different flow statistics which can be directly obtained from the flow tables of the switches. Moreover, OpenFlow managed switches typically report any network status changes like e.g. a failed link status, instantly to the NOS. They also exchange frequent keep alive messages with the NOS in order to determine the network status as a whole. On the other hand, the NOS is frequently using a mechanism similar to the Link Layer Discovery Protocol (LLDP) to obtain the current network topology and the regarding interconnects. Furthermore, the NOS can be triggered to query the network nodes via the OpenFlow protocol in order to obtain the flow tables, flow entries, as well as their counters and statistics. This particular mechanism is utilized by SOFTmon to provide a very fine granular flow-based monitoring solution.

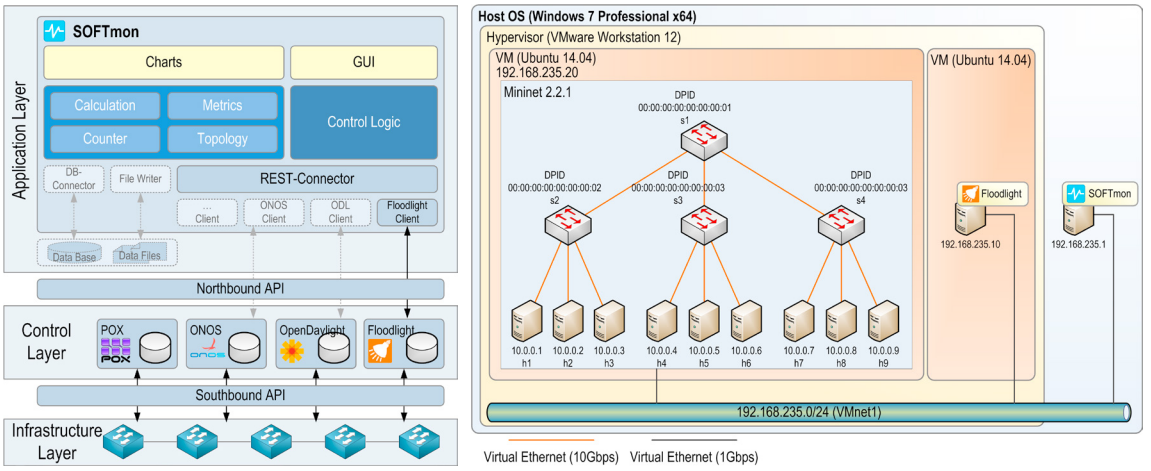
There are several open source OpenFlow based SDN NOS available. OpenDaylight [3] and Floodlight [4] for instance, are very common at the moment. As previously mentioned, they generally support basic monitoring capabilities like the visualization of network topology or the flow statistics in a tabular representation. Nevertheless, the presentation of this information can be evolved, since it is not really human readable neither it provides an appropriate understanding of the current network utilization. Thus, several papers try to address this issue with proposals and approaches for SDN based network monitoring [5,6,7]. However, almost all of them mainly deal with different measurement approaches and procedures in order to increase the measurement accuracy concerning time. On the other hand, some papers [8,9] present controller module extensions, which are bound to a particular NOS and interact directly with the packet forwarding process. Others again, describe just some early work prototypes [10], which are not available for testing or downloading.

In contrast, the presented SOFTmon tool presented in this paper introduces a method of flow monitoring using the northbound NOS interface. The tool is completely decoupled from other network or software components and acts as an additional utility to observe the network utilization. Furthermore, the prototype implementation including the Floodlight connector is available on GitHub [11].

3. Architecture

The key idea of SOFTmon is to implement a NOS independent traffic monitoring tool which adds additional monitoring capabilities and provides them within a proper visualization. Thus, SOFTmon implements a traffic measurement method that exclusively relies on the switch, port and flow statistics, which are defined by the OpenFlow standard and can be queried and obtained by every common NOS. SOFTmon is a business application on the network

application layer of the SDN paradigm as described in [12] and depicted in Figure 1(a). It interacts with the northbound NOS interface via the platform specific application programming interface (API).



(a) Conceptual SOFTmon software architecture

(b) Mininet based development environment and topology

Fig. 1. Architecture

The conceptual architecture of SOFTmon itself is likewise based on the pattern of a layered software architecture. The lowest layer is the data access layer and includes database, file I/O, and representational state transfer (REST) support. This layer provides the basic functionality required for the communication with the NOS. The majority of open source NOS implementations provide an interfaces which follows the REST paradigm as northbound API [13] facilitating a programming language independent interface for network applications. Thus, this interface was favored as a connector between SOFTmon and the NOS. Since the NOS northbound API has not yet been standardized, using REST seems to be the best way of adding new NOS connectors for other network controllers easily. Therefore SOFTmon's architecture contains an abstraction layer called REST connector, which defines the methods and the data model that have to be provided by a specific communication module for the regarding NOS implementation. The next higher layer includes the data model. The data model computes the performance metrics by using the statistics provided by the NOS. The NOS again obtained these information via OpenFlow from the network nodes. However, the data model is composed out of three main elements. First of all this is the topology. It is composed out of all network devices, which typically are switches, and their interconnects. Further, it contains the counters which are keeping the statistical data. The last element is the metrics. They are needed to visualize the network performance. The topology and counter object model is predominantly based on the OpenFlow v1.3 specification [14]. Thus, the functionality of transforming the data model obtained by the NOS non-standardized REST API into SOFTmons's data model has to be provided by the particular REST client of the data access layer. The topmost layer contains the graphical user interface for user interaction and data visualization. It is a selection of tabs for measurement options, buttons for starting and stopping the visualization, and the chart component for a graphical presentation of the performance metrics in soft real time.

During the SOFTmon development process, a virtual test environment based on the Mininet [15] network emulator and the Floodlight [4] SDN controller has been used. This development environment is composed of two virtual machines (VM) with an Ubuntu 14.04 Linux guest operating system nested in a VMware Workstation hypervisor hosted on a Windows 7 Professional system. The first VM contains the Mininet emulator. Mininet is configured with a tree topology with depth two and fanout three, as depicted in figure 1(b). During the entire development process this topology was recurrently used in order to obtain reproducible return values from the NOS. The NOS is encapsulated in the second VM whereas the development itself was carried out directly on the Windows host operating system using the Java and the Eclipse integrated development environment (IDE).

Floodlight was chosen for the first implementation in SOFTmon, because it is widely-used in the area of research and development. Moreover, it is comparably less complex then e.g. OpenDaylight and also well documented. It

further includes a collection of network applications. Almost all of these applications are built as Java modules and directly compiled together with Floodlight. Other built in network applications again are using the REST API. Thus, it comes with a lot of helpful implementation examples. In particular, for the SOFTmon development and testing the modules *Forwarding* and *Learning Switch* came into operation.

4. Implementation

The architecture introduced in section 3 supports an incremental and modular development of the overall software system. However, the monitoring capabilities of SOFTmon are always limited by the information provided through the RESTful interface of the underlying NOS. In order to integrate a particular NOS, it is necessary to implement the regarding REST client. The SOFTmon prototype works with Floodlight. Albeit a detailed documentation of the unified resource identifiers (URIs) for the specific Floodlight REST calls[16] exist, no data model description for the used Javascript Object Notation (JSON) return structure is given. Thus, the data model needed to be identified first by reverse engineering and probes. In order to support further NOS in the future, the REST client implementation may turn out to be the most complex and time consuming part. This depends highly on the completeness of the NOS REST API documentation.

Table 1. Metrics and underlying counters

Type	Metric	Unit	Counter	Unit
Switch Stats.	Flow Count	n	Active Entries (Tables)	n
	Packet Rate	n/s	Received Packets (Flows)	n
	Byte Rate	Byte/s	Received Bytes (Flows)	Byte
Port Stats.	RX Packet Rate	n/s	Received Packets	n
	TX Packet Rate	n/s	Transmitted Packets	n
	RX Byte Rate	Byte/s	Received Bytes	Byte
	TX Byte Rate	Byte/s	Transmitted Bytes	Byte
	RX Port Usage	%	Received Bytes	Byte
	TX Port Usage	%	Transmitted Bytes	Byte
Flow Stats.	Packet Rate	n/s	Received Packets	n
	Byte Rate	Byte/s	Received Bytes	Byte

The calculation of the performance metrics is based upon the port and flow statistics defined in the OpenFlow v1.3 standard. Table 4 shows the calculated metrics in relation to the underlying statistics (counters). The switch counters represent aggregated values that are not part of the OpenFlow specification, whereas the aggregation is carried out by the NOS. Since a performance metric $m(t)$ is a time-related value, it can be calculated through the time derivation of the corresponding time-dependent counter $c(t)$:

$$m(t) = \frac{d}{dt}c(t) \tag{1}$$

Counter values are available only in time-discrete form. Thus, the calculation of a metric can be approximated by using the corresponding time interval Δt:

$$m(t) = \frac{c(t) - c(t - \Delta t)}{\Delta t} \tag{2}$$

The OpenFlow specification defines duration counters for port statistics, as well as flow statistics (since OpenFlow version 1.3), which could be used as a time base for the time-dependent counter values. This would help to achieve measurements with a theoretical accuracy up to one nanosecond. However, the counter for the nanosecond portion of the duration is marked as optional in the OpenFlow specification. Thus, the maximum feasible and guaranteed time resolution is one second. In addition, the port duration counters do not exist in earlier OpenFlow versions.

Unfortunately, one second is not sufficient to achieve a fluent visualization in soft real time. The solution for this issue is to create an additional time base by generating and adding a system time stamp to the counter values based on the arrival time of the corresponding JSON object received from the NOS. This is also necessary for adding the functionality of presenting historical values. The resulting error of the time stamp approach will be analyzed in detail in section V.

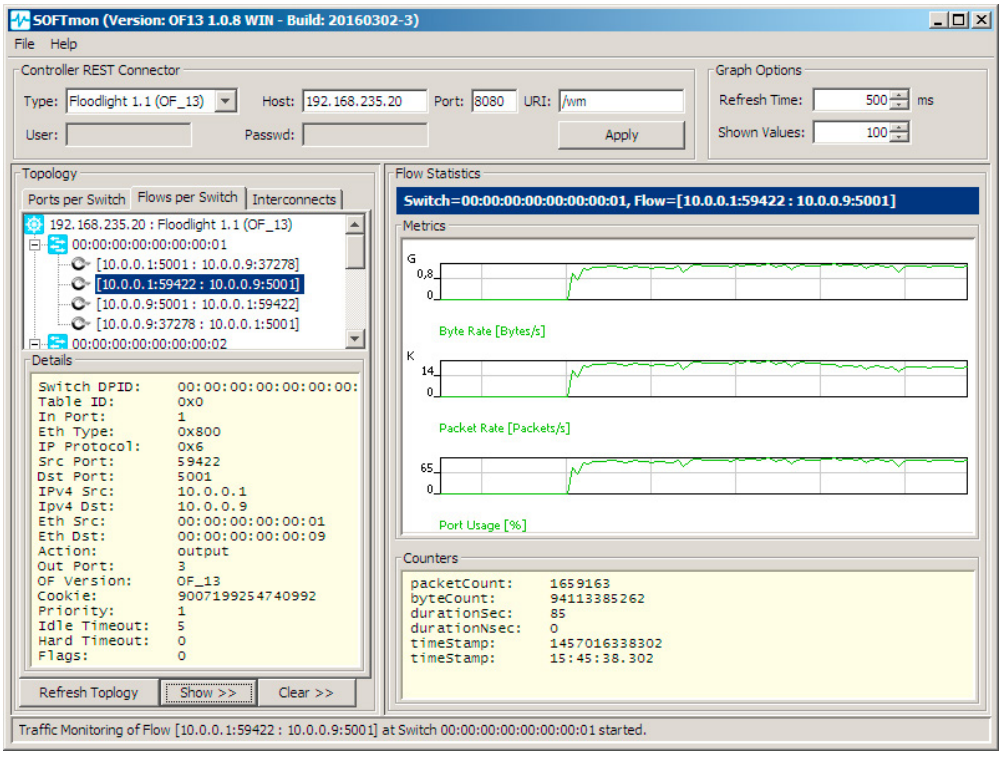


Fig. 2. SOFTmon GUI

Figure 2 presents an overview of SOFTmon's graphical user interface on a Windows 7 OS. The parameters and credentials for the REST connection and the regarding NOS can be configured in the upper left area. To the right is an other area where the refreshing time and the amount of values for the visualization can be adjusted. The tree view on the left allows to chose different points of measurement dependent on the selected tab. The tree view also reflects the network topology, while the tabs allow to switch the presentation according to the ports per switch, the flows per switch or the switch interconnects. Further details of the selected sample are displayed on the lower left side, whereas the sample is presented as a chart on the right side. The OpenFlow based statistics values are monitored in soft real time.

In order to measure the network utilization caused by a particular flow, some effort has to be spent in filtering the flow tables of a certain switch and locating the statistic entries of interest. The Floodlight REST interface only allows to query the complete list of all flows in all flow tables of a certain switch at once. This list is arranged by the table ID and the processing sequence of each table regarding the switch's matching process. The current SOFTmon prototype only supports flow monitoring for the network layer. This means in order to become a selectable item in the SOFtmon GUI , a flow needs to have valid entries in the fields IPv4 source and destination address as well as Ethernet source and destination address. Further, the instruction field must contain a valid action. However, flows are installed and deleted dynamically by the Floodlight's Learning Switch module. Thus, flow list obtained by the NOS and it's flow statistics can differ in length and sequence from one measurement cycle to the subsequent one. Therefore, the flow that is selected for monitoring has to be identified in the list through an internal matching process in which the following fields are compared: flow table ID, IPv4 source and destination address, Ethernet source and destination address, Ethernet type, IP protocol, transport protocol source and destination port and physical input port. Not mandatory

values (e.g. transport protocol ports) are substituted by a wildcard for the search process. In order to not disrupt a flow's utilization visualization metrics when it has been deleted, the statistic values of a missing flow are marked as invalid. This causes the calculation module to return zero as a value. Therefore the graph of the measured metric drops also to zero, but is continued to be drawn until the selected flow is probably active again. The GUI elements that can be selected for monitoring are: switches, switch ports, and flows. They are presented in a tree structure corresponding to the network's topology. Since flows can be installed and deleted within short time frames, this tree structure for selecting the measurement has to be updated manually by the user. The visual presentation of a metric is implemented with the JChart2D library [17]. It is intended especially for engineering tasks and therefore optimized for the dynamic and precise visualization of data with a minimal configuration overhead. The user can configure the duration of a measurement cycle d_M, as well as the amount of values displayed in a graph N_M via the GUI.

5. Evaluation

In order to determine the error that emerges from the proposed and implemented system time stamp approach to label the probes, as described in section IV, the deviation of a switch port metric $m(\Delta t_S)$ is measured. This metric is calculated for a time interval Δt_S based on the time stamps for the metric $m(\Delta t_C)$, which again is calculated for the time interval Δt_C of the time counters. As shown in table 2, the experimental evaluated and calculated relative deviation of the time interval increases slightly with a decreasing duration of the measurement cycle d_M. In contrast, the mean deviation of the calculated metric is constantly lower than 0,005 percent.

Table 2. Empirical identified error with time stamp approach

d_M	1000 ms	500 ms	250 ms	50 ms
d_O	27.24 ms	21.60 ms	22.39 ms	27.49 ms
d_R	6.28 ms	6.32 ms	7.11 ms	5.13 ms
	Mean relative Deviation from Δt_C based Values			
Δt_S	-0.002 %	0.001 %	0.004 %	0.110 %
$m(\Delta t_S)$	0.002 %	0.002 %	0.000 %	-0.001 %

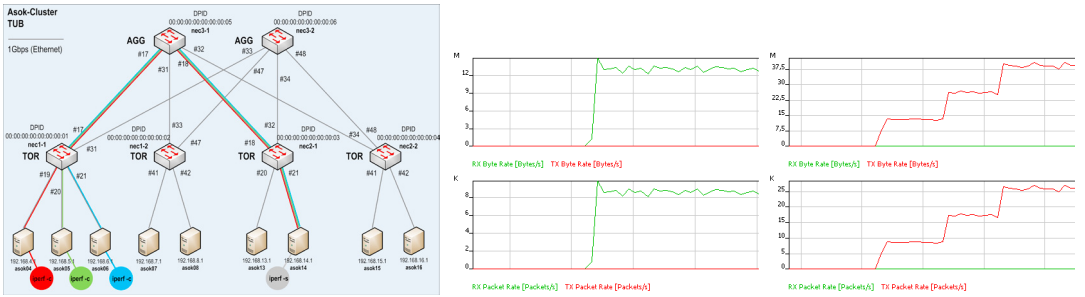
The mean REST call execution time d_R of the test system results in comparatively constant values between approximately five and seven milliseconds with regard to the measurement cycle d_M. However, there is an offset d_O from the instant of time t_S based on time stamps to the instant of time t_C based on time counters. This offset has an averages time of around 25 milliseconds. That means, the metrics that are obtained from the NOS are visualized and displayed around 25 milliseconds later than they actually occur. This is negligible for the applicability as anetwork monitoring tool. In a nutshell, the obtained results demonstrate that even commodity hardware is able to deliver a sufficient sample rate and resolution for the usage of SOFTmon.

In addition to the Mininet based development environment presented in section III, SOFTmon was also intensively evaluated on a local SDN research cluster. This cluster is named Asok and has a typical SDN enabled data center fat tree network topology. The SDN network is composed out of dedicated OpenFlow switches from NEC. Table 3 lists all components and their hardware and software specifications as used for the cluster based evaluation.

Table 3. Asok Cluster Hardware Configuration

System	Hardware	OS/Firmware
Cluster Node	2 Intel Xeon Quad-Core 2,66 GHz, 32GB RAM	Ubuntu Server 14.04.3 LTS 64 bit
NOS Node	Pentium Dual-Core E5500 2,80GHz, 4GB RAM	Ubuntu Desktop 14.04.3 LTS 64 bit
Monitoring PC	Intel Core i7 2,8 GHz, 8GB RAM	Windows 7 Professional 64 bit
Switches	NEC IP8800/S3640-48T	OS-F3L Ver. 11.1.C.Af

In order to evaluate the monitoring performance with SOFTmon, network traffic was generated using the iperf tool [18]. Figure 3(a) depicts the evaluation deployment as well as the iperf server and client configuration. The NOS



(a) Asok Research Cluster - Evaluation configuration

(b) Port traffic on cluster switches

Fig. 3. Evaluation on a SDN cluster

and the SOFTmon application are running on dedicated nodes, which are not directly part of the cluster. They are not connected to the SDN data network, but to the separated management network via 1Gbps Ethernet. This network is used for the NOS to switch communication and vice versa.

Figure 3(b) shows port traffic probes that were collected with SOFTmon on the cluster. It shows the throughput as byte and packet rate. This particular example was generated with the iperf setup that previously has been introduced and described. The iperf clients were configured to use a to 100Mbit/s limited transmission rate in order to avoid traffic congestion. The graph depicted in figure 3(b), shows the measured and visualized throughput on port 19 of switch nec1-1. This is the incoming (RX) traffic from client asok04, which reaches an averages of 12,5 MByte/s. This correlates with the configured 100 Mbit/s transmission rate. Moreover, the graph on the right shows the outgoing (TX) throughput of port 18 of switch nec3-1 which is the sum of the iperf traffic of all three clients (asok04 to asok06) that was started successively. The traffic that was limited to 100 Mbit/s per client reaches an average overall amount of 37,5 MByte/s, which again correlates to the configured 300 Mbit/s transmission rate.

For further evaluation of SOFTmon under real traffic conditions, the development environment, as introduced by fig. 1(b), was used for video streaming experiments. The charts that are presented in 4 were collected while retrieving a video live stream with a web browser that was started on a virtual host in the Mininet environment. The curves are showing data bursts which are typical for video streaming.

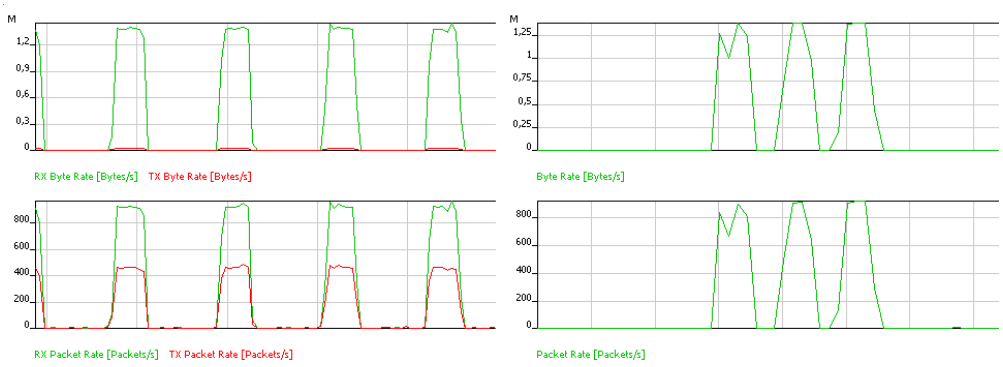


Fig. 4. Port and flow metrics with Youtube traffic evaluated on Mininet

All charts that are presented in this paper are screenshots of the current version of the SOFTmon tool. They reflect samples taken during the evaluation and validation process.

6. Disclosure

This work was supported by a fellowship within the FITweltweit programme of the German Academic Exchange Service (DAAD).

7. Conclusion

The introduced monitoring tool presents a new and innovative approach for network monitoring in OpenFlow networks. It extends the topology based monitoring capabilities that are provided by common NOS and can be used to to determine any kind of network behavior. The implementation is open source and available on GitHub[11]. Moreover, its reliable software architecture is waiting for contributions from the community, in order to extend the existing implementation of the Floodlight REST client or add support of further NOS. Its implementation is based on Java, so it can be used on any operating system.

The presented application was successfully evaluated with Mininet and OpenFlow version 1.3. Moreover, it was evaluated on a SDN research cluster with the typical data center network topology and OpenFlow version 1.0. Furthermore, a live video streaming was used to evaluate the tool under real network traffic conditions. Thus, SOFTmon has already proven that it's capital S does not mean simple in terms of limited capabilities, it means simple in terms of being easy to operate. The high usability of the tool was actually one of the design requirements. In order to deploy SDN in productive environments, a simple manageable tool like SOFTmon could be very helpful to provide e.g. Network Operation Control (NOC) operators with a simple but powerful administrative application. Furthermore, another benefit is that SOFTmon does not require direct access to the network. Thus, a local admin for example can use the tool to debug a network issue, while the main control over the network still remains by the NOC.

References

1. Zabbix :: The enterprise-class monitoring solution for everyone. 2016. URL: `http://www.zabbix.com`.
2. Nagios – the industry standard in it infrastructure monitoring. 2016. URL: `https://www.nagios.org`.
3. Opendaylight platform. 2016. URL: `https://www.opendaylight.org`.
4. Project floodlight. 2016. URL: `http://www.projectfloodlight.org/floodlight/`.
5. Baik, S., Lim, Y., Kim, J., Lee, Y.. Adaptive flow monitoring in sdn architecture. In: *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*. 2015, p. 468–470. doi:10.1109/APNOMS.2015.7275368.
6. Isolani, P.H., Wickboldt, J.A., Both, C.B., Rochol, J., Granville, L.Z.. Interactive monitoring, visualization, and configuration of openflow-based sdn. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, p. 207–215. doi:10.1109/INM.2015.7140294.
7. Pajin, D., Vuleti, P.V.. Of2nf: Flow monitoring in openflow environment using netflow/ipfix. In: *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. 2015, p. 1–5. doi:10.1109/NETSOFT.2015.7116138.
8. van Adrichem, N.L.M., Doerr, C., Kuipers, F.A.. Opennetmon: Network monitoring in openflow software-defined networks. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014, p. 1–8. doi:10.1109/NOMS.2014.6838228.
9. Grover, N., Agarwal, N., Kataoka, K.. liteflow: Lightweight and distributed flow monitoring platform for sdn. In: *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. 2015, p. 1–9. doi:10.1109/NETSOFT.2015.7116160.
10. Raumer, D., Schwaighofer, L., Carle, G.. Monsamp: A distributed sdn application for qos monitoring. In: *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. 2014, p. 961–968. doi:10.15439/2014F175.
11. Softmon. 2016. URL: `https://github.com/mha-net/SOFTmon`.
12. Open networking foundation. 2016. URL: `https://www.opennetworking.org/about/onf-overview`.
13. Kreutz, D., Ramos, F.M., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., Uhlig, S.. Software-defined networking: A comprehensive survey 2015;**103**(1):14–76.
14. Open Networking Foundation, . OpenFlow switch specification 1.3.0. 2012. URL: `https://www.opennetworking.org`.
15. Bob Lantz, . Mininet VM images. 2016. URL: `https://github.com/mininet/mininet/wiki/Mininet-VM-Images`.
16. Floodlight, P.. Floodlight controller - rest api. 2016. URL: `https://floodlight.atlassian.net`.
17. Achim Westermann, . Trace2dltd (JChart2d API documentation, version 3.2.2). 2016. URL: `http://jchart2d.sourceforge.net`.
18. iperf2. 2016. URL: `https://sourceforge.net/projects/iperf2/`.