Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 961–968
DOI: 10.15439/2014F175
ACSIS, Vol. 2

MonSamp: A Distributed SDN Application for QoS Monitoring

Daniel Raumer, Lukas Schwaighofer, and Georg Carle
Technische Universität München, Department of Computer Science, Network Architectures and Services
{raumer|schwaigh|carle}@in.tum.de

***Abstract*—Software Defined Networks are intended to be less complex, more flexible, and free of vendor-lock-ins. Therefore the Software Defined Networking (SDN) instantiation OpenFlow has been designed according to these properties. The efforts are expected to result in lower expenditure and operational costs. To reach these objectives, mechanisms of classical networks that provide established functionalities have to be revalued and either transformed or redesigned from scratch to take advantage from SDNs. In this paper we describe our vision on flow sampling suitable for traffic monitoring in those networks. Without the loss of generality our approach was specifically created to monitor the quality of service for flows. We describe monitoring as one of possibly many applications that communicate with the SDN controller via the SDN Northbound API. We implemented a prototype SDN application called MonSamp and performed tests to demonstrate the feasibility of our concept.**

***Keywords*—SDN, QoS Monitoring, SDN Application, Flow Sampling, Northbound API**

I. Introduction

TRIGGERED by OpenFlow [1] (OF) the rise of SDN breaks one of the basic network design principles that was valid since the first days of the Internet [2]: the avoidance of central instances as these can be a single point of failure that compromise the robustness of the network. The main idea of SDN is the separation of control and data plane. The most prominent SDN instantiation, OpenFlow is managed by the Open Networking Foundation (ONF [3]). It provides access to the forwarding plane of network devices. Flexible controller software that performs the control plane tasks runs on commodity hardware while switches serve the data plane according to the rules received via the OpenFlow protocol. Uniform data plane hardware avoids vendor lock-ins and is expected to reduce capital expenditure as less functionality is required from it. The wide area network of Google, B4, demonstrated the feasibility [4] of SDN deployments in large and productive wide area networks. While the OpenFlow protocol itself is qute stable, real world SDN applications and their requirements for the SDN Northbound API (NB), allowing the combination of multiple, modular SDN applications into a single consistent controller, are still research in progress [5].

At the time of writing, SDN is still an active research topic. Only very few known SDN installations used for productive, non-research networks exist. One example is the B4 network [4]. With this paper we want to add techniques for sampling traffic suitable for quality of service analysis to the available SDN tools, as the OpenFlow protocol itself only provides little feedback (similar to SNMP) about the network traffic, which is unsuitable for an in-depth quality analysis. Thus, we introduce an SDN application for traffic sampling designed with the requirements for a later quality of service (QoS) analysis in mind. Our goal is to extract a subset of the packets handled by the switch, without limiting the selected traffic to a few ports or suffering from seemingly random drops because of an over-utilized monitoring link. We achieve this using the capabilities that are already present in SDN migrated networks, i.e. commodity servers and OpenFlow switches. We do not introduce a new technique for data analysis or traffic sampling. Our focus is on moving sampling away from dedicated hardware into cheap standard network hardware using SDN techniques.

The paper is structured as follows: In section II we give an overview to the state of the art in network monitoring. Section III describes the challenges of QoS monitoring and discusses ways of integrating these into SDNs. Based on these we infer properties for the sampling and present our architecture. In section IV the prototype, the setting in which we tested our prototype, and first measurement results that demonstrate the capabilities of our model are presented. Afterwards, in section V we discuss the state of the art and related publications. We conclude with by highlighting our contributions, giving a wrap up of open research questions and providing an outlook to our intended future work in section VI.

II. Excursus to Network Monitoring

Gaining information about the network performance is a worthwhile objective that is not unique to SDNs but has been addressed in classical networks since the first days. It is important to differentiate between the productive and the monitoring functionality. The latter could be omitted without having any direct effect for the information delivery. In this chapter we provide a short overview to the state of the art in QoS-monitoring and discuss what distinguishes QoS monitoring from other monitoring objectives.

A. State of the Art in QoS-Monitoring

Network monitoring techniques can be classified according to many criteria: They can be active or passive and require end host control or just access to the network. They may run on a single node or require aggregation of information gathered on different locations, require dedicated hardware and software or be part of existing network equipment. The techniques can be vendor specific, standardized or open solutions.

The value of the information provided by the different techniques for QoS monitoring is highly diverse. Active mea-

978-83-60810-58-3/$25.00

surement tools like SmokePing are used to monitor the latency and connectivity of a specific path but are impractical for monitoring each possible connection in a network. Passively collecting information about the network traffic is a service provided by most forwarding network devices. Information about flows can be pushed to a monitor using protocols like IPFIX (formerly NetFlow) or actively pulled from a monitor via device management protocols like SNMP, NETCONF, or OpenFlow. Theoretically almost any information can be exported via these protocols. However, each network device needs the capability to collect the required network information and to be able to export it to a collector or monitor where information of different networking devices can be combined or analysed. Although OpenFlow is intended to provide vendor independent access, the heterogeneity of the networking hardware available at the market leads to a situation where only selected devices are able to provide the desired information with an acceptable performance. Thus, specific requirements may limit the choice of vendors which again creates a dependency similar to vendor lock-ins. So, in practice, the usefulness of QoS information exported by the network devices is very limited. Additionally, pulling information too frequently causes high load to the network devices which may have a negative impact on the performance of the productive network (c.f. [6]).

The other widely used monitoring technique we will briefly introduce is the collection of local network information by so called sniffers. Sniffers may be hardware components or software tools; e.g. Wireshark (former Ethereal) or tcpdump. They are able to collect and log network traffic and usually require truncating, filtering, and aggregation of the collected packets to extract useful metrics. Software tool kits like VERMONT [7] contain a set of modules that allow configuring a sniffer with adequate post-processing functionality tailored to almost any problem. Other solutions like the intrusion detection system Snort [8] can also be described as sniffers with post-processing functionality focused on detection of network attacks. In state of the art approaches sniffers recognize packet drops by looking at higher level information like TCP retransmits and round trip times. Because the sniffers only have access to traffic of a certain network link or node, the information is necessarily incomplete: This approach is unable to determine where a drop occurred and is not applicable to UDP traffic used by most time-critical applications. To allow monitoring nevertheless, the analyser needs to understand each higher level protocol (e.g. SIP) that is to be monitored. The downside of this approach is that the monitoring has to be implemented for every protocol on top of UDP and may not be available for some protocols used in the network (especially proprietary protocols can pose problems). While it is theoretically possible to combine the information from multiple sniffers, the requirements to do so are quite high. The exactly same traffic has to be monitored on multiple locations (which may be a problem with sampling). Too much aggregation will diminish the usefulness but without aggregation the load on the analyser is very high as all the monitored packets from the whole network need to be correlated.

B. Sampling for QoS-Monitoring

In opposite to security monitoring where we need to find traffic patterns outlining an attack, QoS monitoring can be tackled by sampling. Whenever we detect bad performance of a single flow, we assume that other flows that are in some ways similar, e.g. they also traverse the same overloaded path, will suffer as well. Note that this assumption holds only to performance problems that are caused by the network and not necessarily to those caused by an application running on an end host. If monitoring is applied to all network nodes the performance degradation can be quantified per hop and thus per physical link or network device. Based on our assumption information about the performance gained from the sampled network traffic can be extrapolated to the whole network traffic using the sampling rate.

The IETF working group on Internet Protocol Performance Metrics (IPPM) focuses on developing and maintaining "standard metrics" that can be applied to the quality, performance, and reliability of Internet data delivery services and applications running over transport layer protocols [9]. The QoS metrics can be gathered on almost all layers of the ISO OSI model: e.g. connection establishment time, connectivity, (maximal/minimal) round trip time, delay, jitter, out of order delivery, (maximal/minimal) throughput/goodput, or packet/information loss.

For the quality perceived by the user the term quality of experience (QoE) is used. These metrics need to differentiate between different types of traffic, e.g. when using the network for telephony and the delay becomes greater than 150 ms the user experience will be bad (c.f. [10]) but same delay for a file transfer does not impact the user experience. Another property is the relation between metrics that may be gathered on different layers. Some indicators are highly dependent on the type of application and often difficult to quantify, but network problems detected on high levels are caused by lower levels. Higher level indicators are thus correlated to lower level metrics. For example layer 1 errors lead to retransmits on layer 2 which lead to more traffic causing a lower throughput. This may lead to congestion causing full buffers on layer 3 that cause higher delays or even drops of packets. The result are retransmits or missing packets on layer 4 impacting the performance of the application and thus the user experience. So it is desirable to gain information on a low protocol layer. For practical reasons the IP layer is used when a solution is intended to be applied to network traffic.

To allow detection of single lost packets, flow level sampling of network traffic is preferable to other sampling techniques as it allows the analysis of complete flows. For instance flow level sampling still allows determining performance metrics like the number of dropped packets and one way delays on the IP layer between two monitoring points. A drawback with state of the art monitoring approaches (c.f. section II-A) is that monitoring of newly deployed protocols is hard and requires adaptation of monitoring mechanisms, whereas the flow level sampling approach would instantly recognize lost packets for any IP-based traffic. Using the flow-tuple to match specific parts of the traffic allows (pre)classification by the network and dealing with it according to its needs. For example a VoIP flow constantly needs low latency and jitter, whereas a file transfer flow just needs high bandwidth on average (it does not even suffer much from frequent bandwidth variations). For scenarios where the flow-tuple is insufficient, the packets can be classified and tagged based on more complex features (c.f. [11], [12]). If different classes of traffic have been

introduced by tagging at the network edges the matching rules can also be refined to match only a subset of the tagged traffic.

Another desirable feature is selective monitoring of certain kinds of traffic, e.g. allowing to decouple statistic and network performance collection from the data plane. A network may have various monitoring systems that focus on different aspects. E.g. a system that measures the performance of telephony traffic is not required to receive any other traffic in the network, while a second monitoring system may not require the telephony traffic.

From a network operator's point of view it is desirable to recognize bad service quality as easily and universally as possible. Network operators are usually not in control of the end nodes of the connections. This condition differs from scenarios like the Google B4 [4] where traffic can be elastically delayed because of the control over the end-hosts.

III. Bringing Monitoring Functionality to SDNs

SDN, in particular OpenFlow, is expected to overcome main problems of classical networks: complexity, inflexibility, and vendor-lock-ins that cause high costs. Consequently we expect to benefit from these advantages when designing a monitoring solution for SDNs.

Software monitoring tool kits, like VERMONT [7], can easily be deployed on commodity servers, are flexible, and can thus be adapted for many different purposes including security monitoring, network logging and gathering QoS metrics (c.f. section II). However, placing a monitor to every link in order to get the complete picture of the network does not scale. Thus it is common practice to place monitors only at central nodes in the network. Therefore, in a typical scenario, the monitors are connected to one or more switches. The bandwidth of this link is orders of magnitude smaller than the backplane capacity of the switch and may act as bottleneck limiting the monitored traffic. The switch is then configured to send a copy of the forwarded packets to the port connected to the monitor. The monitoring system analyses the traffic and extracts performance metrics. These can be visualized for an operator to take appropriate actions or, in a more advanced setup, directly given to the SDN controller and its applications. For example a traffic engineering application can react to congested links by preferring alternative paths. Such an approach also provides a solution to challenges of distributed monitoring by utilizing the central controller of SDNs (c.f. section II-B).

This approach differs from another way of transferring traffic information about the network to the controller that we want to discuss briefly: The initial authors of OpenFlow [1] already intended a configuration where the controller receives all packets not matched by an OpenFlow rule on the switch. However, handling all packets that way significantly impacts the performance. To mitigate they propose an architecture for monitoring where a subset of the traffic is redirected to a programmable packet processing system (e.g. NetFPGA) as installed into the OpenFlow rules. The advantage of this approach is more flexibility as it allows modification and filtering of the network traffic but introduces extra delays, costs, and potential bottlenecks. Therefore we consider it as attractive for security motivated scenarios. On the other hand, this overhead is unnecessary if only passive monitoring and no modification or filtering is desired.

Even in scenarios where the bandwidth is sufficient, other bottlenecks and the unpredictable CPU capacity limits the monitoring capabilities even further. Braun et. al. [13] mitigated the unpredictability and demonstrated that dynamic adoption is a suitable way to avoid loss of monitoring information as long as the whole monitoring system has free capacities. Still, exceeding the link or processing capabilities leads to uncontrolled and thus random loss of packets in the monitoring components. These cannot be distinguished from real packet loss occurring in the productive network parts and are affecting the real traffic. In order to optimize monitoring performance with minimal costs packets should only enter the monitoring network and connected monitoring systems if they are likely to be processed. Our approach utilizes OpenFlow capabilities for fine grained matching of the desired packets and allows balancing the monitoring load. It provides an intelligent alternative to classical monitoring ports that are either affected by tail dropping behaviour on the monitoring link or block productive traffic if the transmit queue for the monitor port is full.

A. Northbound API

In SDN the interface between the controller and the higher level applications and network functions is the Northbound API. It is the logical consequence of the divide et impera principle in software development to hide complexity that is irrelevant for the applications behind clear interfaces. Network functionality is separated from the controller software by the Northbound API. The importance of the Northbound API increases even more with the ongoing efforts of scaling controller architectures. E.g. Kang et al. introduced the idea of providing a "One Big Switch" abstraction to the application [14]. A network operator who wants to implement a network function (e.g. security policies) does not want to think about OpenFlow rules, the underlying controller hierarchy, or the combination of these policies with others. Resulting from our monitoring application we identified the following requirements for the Northbound API:

- **Shared contexts** with different levels of (topology) abstraction should provide access to the information that is required and influenced by different applications. These levels also have to hide information, protect related configuration options for faulty access, and keep application design as easy as possible. E.g. an application for BGP routing can handle the network as one big switch [15] while traffic engineering in the network requires differentiated views of the single switches in the network [14].
- **Conflict free controller behaviour** has to be guaranteed through intelligent failover techniques [16], composition techniques for different applications [17], and prioritization of rules [18].
- **Information transparency** is required in both directions. The Northbound API not only needs to translate abstract rules into OpenFlow rules for the switches but also requires the ability to react to incoming OpenFlow packets sent from switches because none

of the installed rules matched. For `proactive` SDN applications only the translation of the rules is required while `reactive` applications need to be aware of incoming OpenFlow packets too.

Until a standardized Northbund API is established, flexible and transferable SDN applications are out of reach. The relevance of the Northbound API has been noticed by the ONF which started to analyse existing approaches for the Northbound API in 2012. The ONF classified existing Northbound APIs according to their scope and application (from OpenFlow enabled networks to general networks) and their level of abstraction. The definition of the Northbound was so unfocused in terms of the abstraction level that the spectrum ranges from the actual Southbound protocol OpenFlow, addressing a concrete switch to high level virtual network management interfaces like OpenStack (c.f. [19]). Existing SDN surveys contain comparisons of different implementations of the SDN Northbound API [20], [21]. The definitions used in the surveys assume an abstraction level somewhere in the middle of the mentioned extremes. In 2013 the ONF created a new working group with the goal to collect requirements and to implement first use cases [19].

B. Architecture

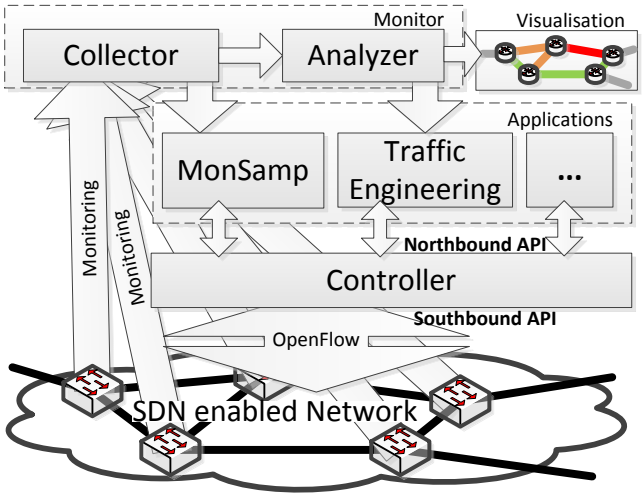


Fig. 1: SDN with integrated monitoring application

Fig. 1 displays an SDN with integrated monitoring functionality. The monitor receives parts of the network traffic for analysis. A sampling application called MonSamp decides which flows to duplicate and send to the monitor. The monitor consists of different modules (here collector and analyser) that are connected adequately for monitoring purposes. The reactive flow sampling application (MonSamp) receives information from the collectors regarding their current workload. Using this information the flow sampling can reactively decide to reduce the number of flows (to avoid random drops). If the links are not working to capacity and the collectors are not fully utilized, the amount of flows can be also be dynamically increased. The selection range for traffic replication is transformed into OpenFlow rules that are installed on the switches by the controller and the Northbound API. In order to allow different applications on top of a controller to interact within the same network slice the Northbound API has to provide abstraction and the ability to combine the behaviour of different applications to a consistent controller behaviour. For example, the monitoring rules must not overwrite or supersede rules required for forwarding of the productive traffic. On the other hand we define a specific interface for communication between the monitor and the flow sampling application. MonSamp uses thresholds to adjust the amount of monitoring load that is forwarded by the SDN enabled devices to allow monitoring without drops. Thereby it mitigates the tail drop behaviour of classical monitoring ports on network devices that occurs whenever a buffer is full. These thresholds are dependent on the concrete monitoring setup and have to be set according to it. MonSamp allows the configuration of monitoring destinations, so that the workload can be split amongst multiple monitoring instances, either though a dedicated physical monitoring network or through a virtual monitoring network. The sampling application also ensures that packets belonging to the same flow are replicated and monitored on all monitoring devices (capacity permitting), thus allowing meaningful correlation. The design supports horizontal and vertical scaling of monitoring systems: It can balance the load to both different monitors with redundant capabilities and different monitors with different objectives (e.g. to a security and a QoS monitor).

In general we cannot assume that all applications running on the Northbound API are reactive. In fact, reactive rules are expected to make the controller a bottleneck of the network [22]. So whenever rules are installed before they match flows or when rules use wildcards the flow sampling application needs information about which flows to replicate for monitoring. The use of event-based OpenFlow messages can again degrade the performance of the controller. To bootstrap the knowledge we imagine to have an ordinary monitoring link. The traffic on this link can be analysed and can serve as input for the flow sampling application. For gaining knowledge about existing flows the random drop behaviour is unproblematic. Another solution is the use of wildcard rules that may be problematic for the switch performance (c.f. section IV-E).

One of the design goals of MonSamp is to avoid any negative influence to the productive traffic. That includes the resources used on the OpenFlow enabled devices. Therefore MonSamp limits the number of installed OpenFlow rules for the monitoring. However, controllers currently do not have any awareness of the performance implications of the installed OpenFlow rules, because the OpenFlow protocol lacks this kind of information [23]. This limitation will be discussed more thoroughly in section IV-E on infrastructural requirements.

IV. Case Study

To provide a proof of concept and to demonstrate the feasibility of our proposed architecture we implemented MonSamp as a prototypic flow sampling application for SDNs and performed experiments for evaluation.

A. Prototype

As discussed in section III-A the definition of the Northbound API is an ongoing process in the community. For our

prototype implementation the Northbound API Pyretic seemed the most fitting. The domain-specific language Pyretic is a successor of Frenetic [24], was published in 2013 [17], [25], and rapidly gained attention. We decided to use it for the following reasons: Pyretic allows for automatic combination of different SDN applications. It provides the required level of abstraction and internally uses the POX controller to communicate with the OpenFlow switches. Pyretic supports both `reactive` and `proactive` applications. It can also be run in an `interpreted` mode, where every packet is sent through the controller. Both Pyretic and its applications are written in Python making things fairly easy to debug and extend in case of unexpected roadblocks. The downside is the relatively low performance, which is not critical for a prototype. We do not claim our choice of the Northbound API to be the silver bullet or a general best practice decision.

The flow sampling application is responsible for deciding whether a newly arriving flow is monitored. The flow sampling application maintains the currently free capacity of the monitor (and the link to it) in its knowledge base. Based on this information the flow sampling application limits the number of flows to be monitored. It does so by computing a direction-independent hash of the flow 5-tuple and selecting the flows within an adjustable hash-range. We achieve direction independence by lexicographically sorting of the destination and source (IP, port)-tuple before hashing. For each monitored flow MonSamp installs a rule into the OpenFlow switches that triggers the action to send a copy of the matched packets to the monitor.

The described prototype focuses on the evaluation of the MonSamp application. Therefore we implemented the monitor as a stub that runs on Linux. It gives feedback about the current level of received traffic to the flow sampling application. Currently we transmit this feedback on the same link that connects the monitor to the OpenFlow switch. There is no interference between the incoming monitoring traffic and the outgoing feedback as Ethernet provides exclusive channels in each direction. A permanent OpenFlow rule on the switch creates connectivity from the monitor to the flow sampling application.

B. Implementation of the Test Scenario

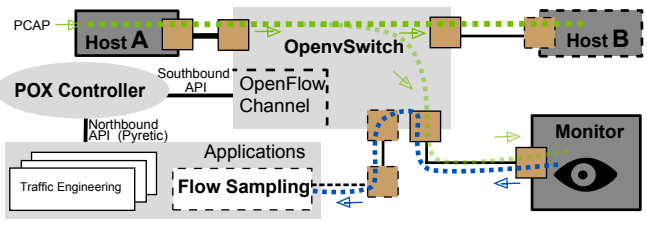


Fig. 2: Realization of the test setup with Mininet

For our case study we used the network emulator Mininet [26] as it allows execution of reproducible network experiments with real world traffic and functional realism [27]. The setup is illustrated in Fig. 2. We created a Mininet topology containing three nodes connected by an Open vSwitch. Mininet separates the hosts by using separate network name spaces, a feature provided by the Linux kernel. As the controller in the default Mininet setup is not part of separate namespace, we added an additional port to the switch and connected it to the root namespace. The flow sampling application, which runs on top of the Pyretic Northbound API, uses this port to communicate with the Monitor. To simulate an application for steering the productive traffic we implemented an application that simply forwards all traffic from Host A to Host B regardless of layer 2 addressing. This allows us to inject traffic containing multiple conversations between different hosts into our test setup. For simplicity we assumed that the productive application acts reactive. Thus the controller notifies the flow sampling application on the arrival of new flows.

C. Measurement Results

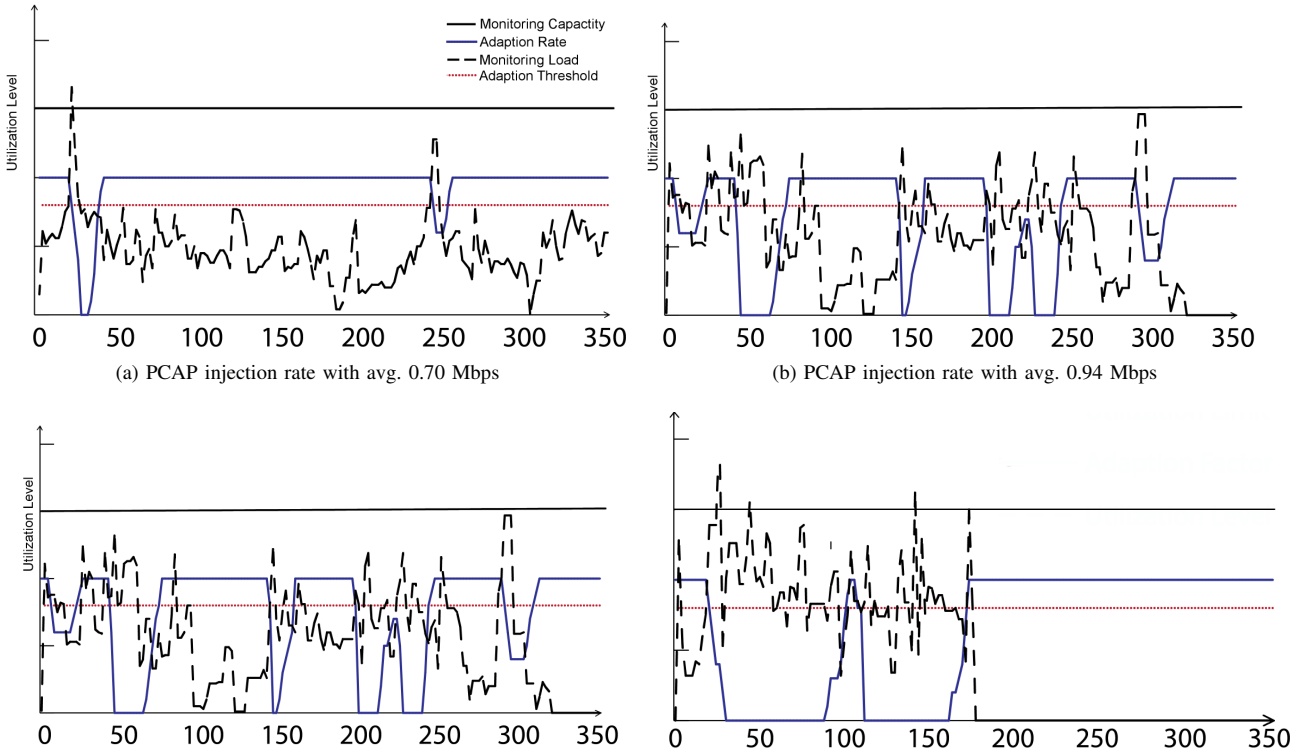
For a first concept evaluation we performed test runs with a real network trace taken from parts of our campus network. Although pseudonymized, we consider the traffic as realistic as the trace contains all connections passing our gateway node which is used by a group of more than 40 students and researchers. The PCAP trace contains 18,000 packets, distributed into 140 TCP and 53 UDP flows. Therefore we created a testing scenario where from the view of the switch and the controller not only two (host A and B) communicate with each other, but more than hundred pairwise different hosts. However, this scenario is representative for real world networks as host A and B represent next hop neighbours of the switch. The monitoring link has a bandwidth of 1.0 Mbps, which we chose as an exemplary link limitation that the Mininet tetbed still can serve without problems.

Figure 3 shows the monitoring utilization during the replay of previously recorded real world traffic through our test setup with four different average speed levels: 0.70 Mbps, 0.94 Mbps, 1.17 Mbps, and 1.40 Mbps. Our application is configured to keep monitoring utilization at 50 % of the theoretical capacity (red doted line). For these initial measurements we assume the limiting factor for monitoring to be a constant link bandwidth. Another relevant factor can be the dynamically changing CPU utilization [13]. The blue line represents the adaptation factor that is the input parameter for the decision function to determine the ratio of new flows to be monitored. The dashed line is the real bandwidth processed by the monitoring system. The Monitor reports this back to the MonSamp application.

The tests show that monitoring of flows without uncontrolled drops in the monitoring system is possible. The few areas where the monitoring load exceeds the capacity are mainly caused by the relative big impact of elephant flows, which are flows that contain a very large share of transferred bits of the overall traffic. We do not expect this to be a problem when applied to high-bandwidth scenarios where new flows are created and old flows are finished more frequently. Thus the impact of a single flow in relation to the total amount of traffic is lower.

D. Open Issues with the Northbound API

Unfortunately, Pyretic – the Northbound API we chose for our implementation – is currently not suitable for realising our



(a) PCAP injection rate with avg. 0.70 Mbps

(b) PCAP injection rate with avg. 0.94 Mbps

(c) PCAP injection rate with: avg. 1.17 Mbps

(d) PCAP injection rate with: avg. 1.4 Mbps

Fig. 3: Sampling adaptation for different bandwidth utilizations

monitoring vision in a meaningful way. The main limitation is the lack of a policy that can make a decision based on an incoming packet. The dynamic policy, as supported by Pyretic, only allows adjusting the policy for future packets, not for the one that is being processed. This means that a default policy decision for the MonSamp application is required – either the first packet of each flow is always monitored or it is never monitored. Because (unless running in interpreted mode) the default policy decision for each flow gets pushed to the OpenFlow enabled hardware, subsequent packets are handled directly on the OpenFlow hardware and not sent to MonSamp any more, rendering the policy-adjustment useless.

Furthermore, Pyretic and other reviewed Northbound APIs for SDNs lack control for the rules that are installed on the OpenFlow switch. Rules are automatically installed based on every decision and applied to the whole flow. It is not possible to define additional rules (e.g. the reverse flow) to be installed or changing the header-fields used for identifying a flow.

E. Infrastructural Requirements

The application of our vision requires both, hardware and software features that provide fields for research. Our monitoring concept requires a scalable controller architecture that provides a reasonably powerful Northbound API. We expect that the desired controller infrastructure will exist in the near future, since first steps were already taken [22], [19]. Thereby the development process of controller software and architectures benefit from the low market barriers, numerous players, and almost no proportional costs for additional features of software developments. These characteristics do not apply for hardware OpenFlow-switches. These switches may support OpenFlow 1.3 – at least after updates of firmware – but still suffer from processing in software on the general purpose CPU if certain OF rules are defined. This results in an extreme heterogeneity of switch behaviour [23] that may lead to poor forwarding performance and an unpredictable capacity for stored OpenFlow rules on the switch. As an example the HP OpenFlow Guide [28], which applies to switches in our testbed, only states forwarding to a single port, dropping, and modification of the IP TOS and VLAN-PCP field as actions that can be executed in hardware; all other actions are executed in software and therefore limit the processing capacity of the whole switch to a rate of around 10,000 packets per second. In virtual switches like Open vSwitch the replication of traffic only introduces a comparatively small and constant overhead per processed packet. The exact overhead is defined by the network stack that is used to send out the traffic to the monitoring system (e.g. to a VM, or to a physical connection). Better optimized hardware for OpenFlow and feedback channels from the OpenFlow switches via the Northbound API to the applications can solve these problems.

V. Related Work

Related work in network monitoring is primary motivated by security concerns and dates back some years. E.g. in 2005 Schaelicke et al. [29] discussed requirements for parallel network monitoring and proposed an architecture for adapting

load balancing of security monitoring traffic. Limmer and Dressler created an adaptive load balancer for NIDS systems. It uses sampling to cope with high bandwidths [30]. It dynamically maps flows to NIDS processes. To alleviate issues resulting from packet drops they use flow sampling, which guarantees loss-free analysis for selected flows.

When monitoring the QoS it is an advantage that a network problem resulting in degraded performance will affect a whole class of flows that share some properties (i.e. routed through an overloaded device and use the same QoS class). Thus, unlike for security purposes, it is sufficient to monitor a representative subset of the network flows which makes QoS-Monitoring eligible for sampling. As we do not provide a new sampling technique we just highlight the long history of sampling dating back to mathematical work on statistics over many decades. For a focused introduction we refer to work of Claffy et al. [31] (general overview), Carela-Español et al. [32] (study of sampling influence for traffic analysis), Braun et al. [13], who recently implemented an adaptive sampling within a monitoring system to mitigate tail dropping behaviour within the overloaded system, and referenced in there.

Using OpenFlow switches for load balancing services (e.g. web servers) in data centres was one of the first use cases of OpenFlow switches. In 2009 Handigol et al. used a reactive NOX controller to minimize response time for load balanced web servers without IP address rewriting [33]. Uppal and Brandon also described a reactive NOX-based load balancer that does address rewriting [34]. Wang et al. presented a NOX based, proactive load balancer that uses wildcard OpenFlow rules with the motivation that reactive approaches causes undesired load to the controller [35]. The controller assumed that the load of each server is proportional to the number of flows directed to it, but did not consider feedback from the servers. Due to their flexibility and the resulting capabilities, OpenFlow switches have also been recognized as a powerful tool for solving scalability issues in network monitoring. For instance big switch network already sells Big Tap, a network monitoring solution [36], [37]. Big Tap uses a separate OpenFlow enabled (monitoring) network equipped with monitoring systems to deliver, filter, and analyse traffic in a scalable manner. Recently Shirali-Shahreza et al. proposed a concept for OpenFlow based traffic sampling called FleXam [38], [39]. They demonstrated how OpenFlow functionality can help detecting attacks in the network [38].

VI. Conclusion

In this paper we described our vision on traffic monitoring in SDNs. Our architecture is scalable and cost efficient because utilizing the already existing OpenFlow functionality does not add any additional costs. All special monitoring functionality is provided by flexible software. We designed an SDN application for extraction and sampling of network traffic directly from the data plane that can be added to other applications via the SDN Northbound API. The application addresses unsolved and solved problems in (QoS) network monitoring with high flexibility and low costs. Thereby we also contributed to the open topic of the canon of SDN applications and requirements for them (c.f. [5]). We implemented a prototype and performed different measurements on it to demonstrate the feasibility of our approach.

Our future work comprises the transfer of our application into a real test environment which also provides more than one switch. Thus we can extend the value of our architecture evaluation beyond functionality (Mininet [27] only provides functional realism). We plan to perform a more sophisticated evaluation, transfer the application to other controller platforms like Ryu, and combining it with monitoring systems like the one proposed by Braun et al. [13]. We also plan further refining to support multiple different limits where each may dynamically interfere as bottleneck and consider sampling coordination together with vertical and horizontal balancing of monitoring loads. These steps should point out required features in future SDNs and demonstrate (e.g. technical) limitations of available SDNs.

Acknowledgments

This research has been supported by the EU as part of KIC EIT ICT Labs on Software-Defined Networking (SDN) and the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung; BMBF) under support code 01BP12300A; EUREKAProject SASER. We also would like to acknowledge the valuable contributions from our colleagues Peter Schaab, Florian Wohlfart, and Lothar Braun to the implementation and maturing of our vision.

References

[1] 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, Mar. 2008. [Online]. Available: http://dx.doi.org/10.1145/1355734.1355746

[2] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff, "A Brief History of the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 5, pp. 22–31, October 2009. [Online]. Available: http://dx.doi.org/10.1145/1629607.1629613

[3] "ONF - Open Networking Foundation," https://www.opennetworking.org/, April 2014.

[4] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013. [Online]. Available: http://dx.doi.org/10.1145/2534169.2486019

[5] B. Munch, "Hype cycle for networking and communications," Gartner, Report, 2013.

[6] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "Oflops: An Open Framework for OpenFlow Switch Evaluation," in *Passive and Active Measurement*. Springer, 2012, pp. 85–95. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28537-0_9

[7] "VERMONT - VERsatile MONitoring Toolkit," https://github.com/constcast/vermont/, April 2014.

[8] "Snort," http://www.snort.org/, April 2014.

[9] "IPPM - IP Performance Metrics," http://tools.ietf.org/wg/ippm/, April 2014.

[10] M. Hassan, A. Nayandoro, and M. Atiquzzaman, "Internet telephony: services, technical challenges, and products," *Communications Magazine, IEEE*, vol. 38, no. 4, pp. 96–103, 2000. [Online]. Available: http://dx.doi.org/10.1.1.43.1441

[11] A. Callado, C. Kamienski, G. Szabó, B. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification," *Communications Surveys & Tutorials, IEEE*, vol. 11, no. 3, pp. 37–52, 2009. [Online]. Available: http://dx.doi.org/10.1109/surv.2009.090304

[12] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, 2008. [Online]. Available: http://dx.doi.org/10.1109/SURV.2008.080406

[13] L. Braun, C. Diekmann, N. Kammenhuber, and G. Carle, "Adaptive Load-Aware Sampling for Network Monitoring on Multicore Commodity Hardware," in *IFIP Networking 2013*, New York, NY, May 2013. [Online]. Available: http://dx.doi.org/10.1.1.395.9415

[14] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the one big switch abstraction in software-defined networks," *Proc. ACM CoNEXT*, 2013. [Online]. Available: http://dx.doi.org/10.1145/2535372.2535373

[15] A. Vidal12, F. Verdi, E. L. Fernandes, C. E. Rothenberg, and M. R. Salvador, "Building upon routeflow: a sdn development experience," in *XXXI Simpsio Brasileiro de Redes de Computadores, SBRC'2013*, 2013.

[16] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "Fattire: declarative fault tolerance for software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 109–114. [Online]. Available: http://dx.doi.org/10.1145/2491185.2491187

[17] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 1–14. [Online]. Available: http://dx.doi.org/10.1145/1384609.1384625

[18] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Hierarchical policies for software defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 37–42. [Online]. Available: http://dx.doi.org/10.1145/2342441.2342450

[19] Open Networking Foundation, "Northbound interface working group charter," 2013.

[20] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *COMMUNICATIONS SURVEYS AND TUTORIALS*, 2014. [Online]. Available: http://dx.doi.org/10.1145/505733.505735

[21] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," 1414.

[22] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265. [Online]. Available: http://dx.doi.org/10.1145/2043164.2018466

[23] P. Perešíni, M. Kuzniar, and D. Kostic, "Openflow needs you! a call for a discussion about a cleaner openflow api," in *The Second European Workshop on Software Defined Networking, EWSDN 2013*, 2013. [Online]. Available: http://dx.doi.org/10.1109/EWSDN.2013.14

[24] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *SIGPLAN Not.*, vol. 46, no. 9, pp. 279–291, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1145/2034574.2034812

[25] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN Programming with Pyretic," *USENIX ;login*, vol. 38, no. 5, pp. 128–134, Oct. 2013. [Online]. Available: http://dx.doi.org/10.1.1.403.3030

[26] "Mininet - An Instant Virtual Network on your Laptop," http://mininet.org/, April 2014.

[27] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation." in *CoNEXT*, C. Barakat, R. Teixeira, K. K. Ramakrishnan, and P. Thiran, Eds. ACM, 2012, pp. 253–264. [Online]. Available: http://dx.doi.org/10.1145/2413176.2413206

[28] H. O. Switches, "Openflow configuration guide," 2012.

[29] L. Schaelicke, K. B. Wheeler, and C. Freeland, "Spanids: a scalable network intrusion detection loadbalancer." in *Conf. Computing Frontiers*. ACM, 2005, pp. 315–322. [Online]. Available: http://dx.doi.org/10.1145/1062261.1062314

[30] T. Limmer and F. Dressler, "Adaptive Load Balancing for Parallel IDS on Multi-Core Systems using Prioritized Flows," in *20th IEEE International Conference on Computer Communication Networks (ICCCN 2011)*. Maui, HI: IEEE, August 2011, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/ICCCN.2011.6006063

[31] K. C. Claffy, G. C. Polyzos, and H.-W. Braun, "Application of sampling methodologies to network traffic characterization," in *ACM SIGCOMM Computer Communication Review*, vol. 23, no. 4. ACM, 1993, pp. 194–203. [Online]. Available: http://dx.doi.org/10.1145/167954.166256

[32] V. Carela-Español, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Solé-Pareta, "Analysis of the impact of sampling on netflow traffic classification," *Computer Networks*, vol. 55, no. 5, pp. 1083–1099, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.11.002

[33] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and J. Ramesh, "Plug-n-serve: Load-balancing web traffic using openflow," in *ACM SIGCOMM Demo*, 2009.

[34] H. Uppal and D. Brandon, "Openflow based load balancing," Tech. Rep., 2010.

[35] R. Wang, D. Butnariu, and J. Rexford, "Openflow-based server load balancing gone wild," in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'11. Berkeley, CA, USA: USENIX Association, 2011. [Online]. Available: http://dx.doi.org/10.1.1.227.7761

[36] "Big Tap," http://www.bigswitch.com/products/big-tap-network-monitoring, April 2014.

[37] big switch networks, "Sollution guide: Open sdn for network visibility - simplifying large scale network monitoring systems with big tap."

[38] S. Shirali-Shahreza and Y. Ganjali, "Efficient implementation of security applications in openflow controller with flexam," *High-Performance Interconnects HotI 2013, Symposium on*, pp. 49–54, 2013. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/HOTI.2013.17

[39] ——, "Empowering software defined network controller with packet-level information," in *Proceedings of the 1st IEEE Workshop on Traffic Identification and Classification for Advanced Network Services and Scenarios, TRICANS'13*. IEEE, 2013, pp. 1355–1359. [Online]. Available: http://dx.doi.org/10.1109/ICCW.2013.6649444