Performance Evaluation of Selective Flow Monitoring in the ONOS Controller

Anh Nguyen-Ngoc*, Stanislav Lange*, Thomas Zinner*,
Michael Seufert*, Phuoc Tran-Gia*, Nieke Aerts§, David Hock§
*Institute of Computer Science, University of Würzburg, Germany.
{anh.nguyen, stanislav.lange, zinner, seufert, trangia}@informatik.uni-wuerzburg.de
§Infosim GmbH & Co. KG, Würzburg, Germany.
{aerts, hock}@infosim.net

***Abstract*—One of the benefits when network operators adopt the Software Defined Networking (SDN) paradigm is the ability to monitor the traffic in the network without an additional network management system. Usually, SDN controllers utilize OpenFlow statistics messages in order to regularly gather information about all flows in the network. However, using the same polling interval for all flows does not take into account the heterogeneity of real world traffic and thus results in an imbalance between monitoring accuracy and control plane overhead. In particular, frequent querying results in a high resource consumption at the controller. This work proposes a Selective Flow Monitoring (SFM) mechanism that allows administrators to classify flows according to their individual requirements in terms of monitoring frequency, e.g., less frequent polling of elephant flows and frequent polling of QoS sensitive VoIP connections.**

We compare the performance of the SFM mechanism with the default monitoring scheme in a testbed featuring the Open Network Operating System (ONOS) controller. In this context, the CPU utilization of the controller is used as performance indicator. After identifying relevant influence factors like the number of flows and switches in the network, we investigate the viability of the approaches in different scenarios. Finally, we provide guidelines regarding their choice.

I. Introduction

Nowadays, the Software Defined Networking (SDN) architecture is getting more and more attention due to its benefits in terms of flexible configuration, programmability, and cost-efficiency [1]. In addition to separating the control plane from the data plane and centralizing the former in the controller, SDN-based networks also provide open interfaces that allow simple monitoring and management. One key element that enables this advantage is the OpenFlow protocol [2]. Furthermore, OpenFlow provides flow-level monitoring functionality, i.e., information exchange schemes and counters for ports, tables, and queues. Per-flow statistics are regularly requested according to a predefined polling interval. Doing that ensures reliable and timely monitoring information that is relevant for management and configuration of SDN-based networks. However, it is unfeasible to perform regular polling at a high frequency for all devices and flows in large scale networks due to overhead.

In order to minimize the overhead at the controller during monitoring, several monitoring mechanisms are proposed in literature. Adaptive monitoring [3] is based on flow characteristics like the lifetime, i.e., long-lived flows are queried less frequently than short-lived ones. The selective approach, which is the focus of this paper, is based on the classification of flows defined by the network operator. In this context, a particular flow can be monitored with a predefined polling interval that matches its needs. In addition, this work compares the selective method with regular polling based on the CPU utilization at the controller via measurements in a testbed. Furthermore, key influence factors that have an impact on the controller's CPU load are identified. These include network conditions like the composition of flows in the network and the number of switches as well as the hardware specifications of the controller.

This work is structured as follows. In Section II, related work is discussed. Section III provides an overview of the testbed setup and experiment parameters, as well as the proposed selective flow monitoring mechanism. While Section IV presents the results of experiments, Section V concludes the paper and gives a perspective on future work.

II. Background and Related Work

Firstly, this section provides a brief overview of the information exchange scheme that is defined in the OpenFlow standard. Secondly, related work regarding flow monitoring in SDN-based networks is presented alongside the flow monitoring mechanisms that are available in the ONOS controller.

The statistics information of all flows in an SDN-based network is sent from OpenFlow switches to the controller by using a *request-reply* scheme. The controller regularly queries the devices in the network about the status of several elements such as ports, tables, queues, groups, or meters. Since OpenFlow version 1.3 [4], information regarding flow and table statistics is aggregated into *multipart messages* in order to reduce the amount of packets that is exchanged. OpenFlow provides per-flow monitoring with a diversity of gathered data, including byte and packet counters, details regarding matched fields, and the lifetime of each flow.

PayLess [5] introduces a network monitoring framework for SDN-based networks. In an effort to balance the accuracy of statistics and the overhead at the controller and switches, the authors propose an approach that dynamically adapts the frequency of the flow statistics collection for all flows.

OpenNetMon [6] implements a POX Controller module that monitors not only the throughput but also per-flow QoS metrics like path delays on the network and application layer. Furthermore, a timer is used to increase the polling interval for new flows and flows with high fluctuations w.r.t. their statistics while stable flows are queried less frequently.

Similarly, OPEN-TAM [3] aims at reducing the overhead by introducing "Adaptive Flow Monitoring" in the ONOS controller. In this context, flows are classified according to their lifetime and the polling intervals are adapted accordingly, resulting in a high polling frequency for short flows and a low polling frequency for long flows. However, this approach can lead to an increase in control plane traffic since the controller queries flows from each category individually rather than requesting statistics of all flow entries in a switch. The intensity of this effect is proportional to the number of flows in the network, which in turn is proportional to the number of *FlowStatsReply* messages that are processed by the controller.

Unfortunately, both works did not measure the resource consumption at the controller. Instead, this work provides insights into the trade-offs regarding this aspect of adaptive flow monitoring by analyzing the CPU utilization at the controller.

The study conducted in [7] features a monitoring scheme that minimizes the communication overhead by aggregating the request and reply messages. The proposed approach is based on the assumption that querying only a small number of switches is sufficient to obtain statistics of the majority of flows in the network.

While OpenTM [8] chooses the switches based on the routing information to periodically poll flow statistics, ProgME [9] collects flow information based on the proposed *flowset*, an arbitrary set of flows that depends on the requirements of an application or a particular traffic condition.

III. Methodology

In this section, the operation of Selective Flow Monitoring (SFM) as well as the differences from Standard Flow Monitoring (STD) are described. In addition, the experimental scenarios are provided alongside the notation for adjustable parameter values as well as details regarding the hardware and software used in the measurement setup.

A. Measurement Setup

In order to investigate the performance of the SFM approach, a testbed as shown in Figure 1 is set up. The Ibis release[1] of the ONOS controller is installed on a PC[2] which uses the Ubuntu 16.04 operating system. Another PC with the same specification runs Mininet[3] and is used to create a network of OpenFlow switches.

At the beginning of the measurement, after the controller is started, a predefined number of flows is installed in the switches via the REST API by using cURL[4]. The term *flow* is resprented a *flow entry* in OpenFlow Table in the switch. Each flow has a timeout of 5,000 seconds to ensure that it lasts the whole 5 minutes of the experiment. Depending on the scenario, up to 408,000 flows are installed. Since this work focuses on the performance of the controller when dealing with control plane information, there is no data plane traffic on the network. Each run is repeated 5 times in order to obtain 95% confidence intervals. Pidstat[5] is used for monitoring the ONOS process and details regarding the average CPU consumption are exported regularly during its runtime.

Fig. 1: Logical Testbed Setup.

In the first scenario, one OpenVswitch[6] is created in Mininet, and the flows in the network are partitioned into three groups, i.e., all flows in a group have the same destination IP address (IP_DST) but different source IP addresses. All IP addresses are declared with the network mask 255.255.225.255, which makes all flows unique and avoids the aggregation of rules in the flow table of the OpenFlow switch. When SFM is enabled, the flows are queried for their statistics according to three polling intervals: SHORT (5 s), MIDDLE (10 s), and LONG (15 s). At every polling time, the controller asks the switch for information of the flows that have the same combination of output port and destination IP, i.e., (OUTPUT; IP_DST). The switch replies with a *StatsRequest* by sending *StatsReply* messages only for the flows that match this condition and aggregates them in a *MultipartReplyMessage*. When using the standard method, the ONOS controller inquires statistics of all flows every 5 seconds and does not require any match condition. The portions of the number of flows from each group are varied in order to investigate the impact of these factors on the controller's CPU utilization.

In the second set of experiments, we evaluate the performance impact of scenarios where more than one statistics message is required to fetch information about each class of flows. In the following, we refer to this number of messages as the number of sub-groups since each class of flows is partitioned based on a set of flow rules. In the i-th group, the tuple for matching its statistics becomes (OUTPUT; IP_DST$_i$). Additionally, testbeds with multiple switches based on a tree topology are considered.

B. Standard Flow Monitoring in ONOS Controller

As mentioned above, the ONOS controller regularly sends *FlowStatsRequest* messages every 5 seconds to get the infor-

[1] http://onosproject.org/, version 1.8.0
[2] Intel(R) Core(TM) i7-4770 CPU @3.40GHz / 16GB RAM
[3] http://mininet.org/, version 2.2.1
[4] https://curl.haxx.se/
[5] http://sebastien.godard.pagesperso-orange.fr/man_pidstat.html
[6] http://openvswitch.org/

mation of all flows in the network. After receiving a *FlowStatsReply* message, the controller processes it and gets the details of all flows.

The number of messages which ONOS needs to handle at every polling point in case of standard flow monitoring (STD) is denoted as N_p^{STD} and can be calculated as follows:

$$N_p^{STD} = N_{(sent)}^{STD} + N_{(recd)}^{STD}$$

In this equation, $N_{(sent)}^{STD}$ and $N_{(recd)}^{STD}$ represent the amount of messages that are generated and sent as well as received and processed by the controller, respectively. The network consists of N_{SW} switches and the total number of flows is N_F. In case of STD, the controller only generates one single $FlowStatsRequest$ message corresponding to a switch. Thus, the equation above becomes:

$$N_p^{STD} = N_{SW} + N_F$$

Considering an experiment duration of T_{exp} with regular polling interval t_{poll}:

$$N^{STD} = \frac{T_{exp}}{t_{poll}} \cdot N_p^{STD} = \frac{T_{exp}}{t_{poll}} \cdot (N_{SW} + N_F) \qquad (1)$$

C. Selective Flow Monitoring

The SFM approach allows administrators to increase the monitoring accuracy for particular flows that are of special interest, e.g., those with strict application-specific QoS demands or SLAs. Furthermore, SFM introduces several configurable types of polling intervals, which can be set by the operator actively, depending on how often the flows are to be monitored.

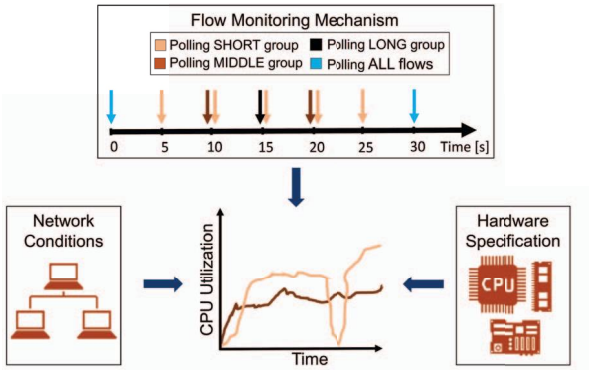


Fig. 2: Influence factors on CPU utilization of the SDN controller when monitoring flow statistics.

Figure 2 illustrates several components that affect the CPU utilization of an SDN controller which are investigated in this work. First, the hardware configuration of the controller such as its RAM capacity and the particular CPU model affects message processing time as well as the amount of flow rules that can be managed. Second, network conditions like the number of connected switches or the composition of traffic flows dictate the amount and the rate of messages that are exchanged between the switches and the controller. Finally, the used flow monitoring mechanism, i.e., SFM or STD, has an effect on the message rate, too.

Figure 2 also gives an example of how SFM works. The controller classifies flows into three different groups based on a specified condition and each group's statistics are queried with the respective polling interval. At the beginning, all flows are polled. After that, every 5 seconds the controller asks the state of the flows that are marked as SHORT which is indicated by the light brown arrows. Then, every 10 seconds, the dark brown arrow shows that flows in the MIDDLE group are queried, and every 15 seconds, *FlowStatReply* messages of flows in the LONG group are sent to the controller. Finally, an ENTIRE polling happens each 30 seconds to ensure that all flows are updated. It is worth noting that when an ENTIRE polling is performed, no other polling takes place. In contrast to this mechanism, the standard approach in the ONOS controller frequently inquires information of all flows every 5 seconds. The differences between SFM and STD are the variation of the number of polling intervals as well as the classification of flows into groups. Hence, only the flows of a particular group are queried rather than all flows.

Assume that the amounts of flows that are requested with the short, middle, and long polling intervals are N_s, N_m, and N_l, respectively. Hence, in Equation 1, the total number of flows corresponds to $N_F = N_s + N_m + N_l$. When SFM is enabled, four types of requests can happen when polling:

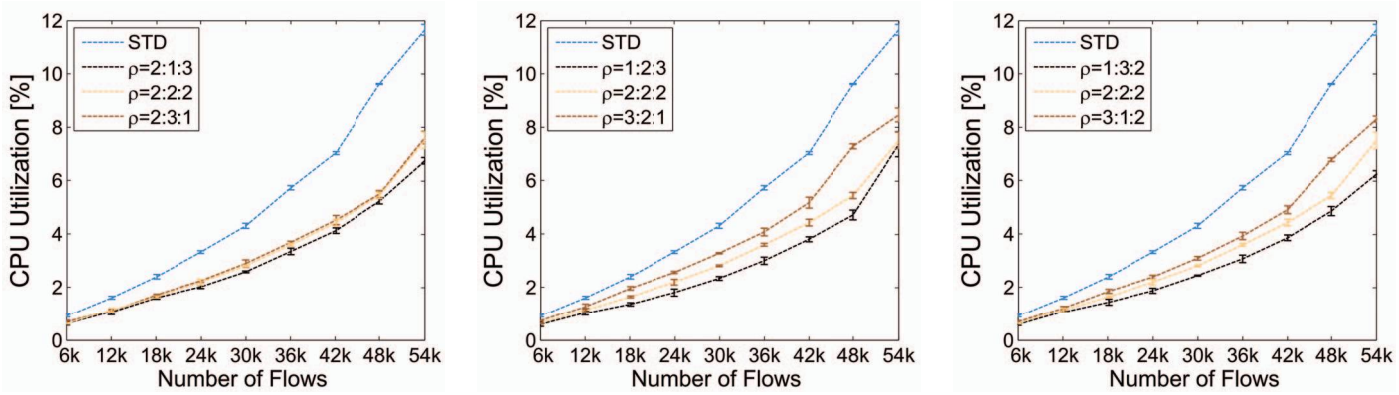
1) Only SHORT flows are queried:
$$N_1^{SFM} = N_{1(sent)}^{SFM} + N_{1(recd)}^{SFM} = N_{SW} \cdot n_s^g + N_s$$
2) Flows that are either SHORT or MIDDLE are queried:
$$N_2^{SFM} = N_{SW} \cdot (n_s^g + n_m^g) + (N_s + N_m)$$
3) Flows that are either SHORT or LONG are queried:
$$N_3^{SFM} = N_{SW} \cdot (n_s^g + n_l^g) + (N_s + N_l)$$
4) All flows are queried (ENTIRE-polling):
$$N_4^{SFM} = N_{SW} + N_F$$

In this context, n_*^g denotes the number of sub-groups within each group and N_* refers to the total number of flows within each group.

During an experiment with duration T_{exp}, t_i is the polling interval that corresponds to short, middle, long, and entire interval. Hence, the total number of messages during an experiment can be calculated as follows:

$$N^{SFM} = T_{exp} \cdot \sum_{i=1}^{4} \frac{1}{t_i} \cdot N_i^{SFM} \qquad (2)$$

Suppose that the polling intervals for the MIDDLE and LONG groups, t_m and t_l, are multiples of the polling interval for the SHORT group, t_s, i.e., $t_m = \alpha \cdot t_s$, $t_l = \beta \cdot t_s$. In an analogous fashion, the numbers of flows in each group are



(a) Fraction of SHORT flows is constant. (b) Fraction of MIDDLE flows is constant. (c) Fraction of LONG flows is constant.

Fig. 3: Influence of relative group size on CPU utilization.

$N_m = m \cdot N_s$ and $N_l = n \cdot N_s$. Hence, from Eq. 2, the number of messages in case of SFM, N^{SFM}, is calculated as follows:

$$\frac{T_{exp}}{t_s} \cdot \left[N_{SW} \cdot \left[2n_s^g + \frac{2(n_s^g + n_m^g)}{\alpha} + \frac{1 + 2 * (n_s^g + n_l^g)}{\beta} \right] + \left(2N_{SW} + \frac{2 + 2m}{\alpha} + \frac{3 + m + 3n}{\beta} \right) \cdot N_s \right] \quad (3)$$

Equations 1 and 3 present several influence factors which impact the number of flow statistics messages, such as the number of sub-groups (n_s^g, n_m^g, n_l^g), the relative size of each group ($\rho = N_s : N_m : N_l$), the ratios between the corresponding polling intervals ($\gamma = t_s : t_m : t_l$), as well as the number of switches in the network N_{SW}. Details regarding the investigation of those parameters are presented in Section IV.

IV. Experimental Evaluation

This section provides the results of the experiments that are described in Section III. Firstly, a comparison between the STD and SFM approaches in terms of the controller's CPU utilization is provided alongside the impact of the size ratios of flow groups. Secondly, the impact of changing the number of sub-groups and the number of switches in the network is presented. Finally, we discuss how the amount of memory that is reserved for the Java Virtual Machine (JVM) affects the efficiency of the controller.

A. CPU Utilization in Case of a Single Switch

Figure 3 displays the impact of different portions of the number of flows in each group. For a given total number of flows in the network on the x-axis, the y-axis shows the corresponding mean CPU usage during the controller's run time. In this experiment, the maximum memory that is assigned to the JVM equals 512 MB which is the default setting of the ONOS controller. As a result, there is a limitation of 54,000 flows that the controller can handle without throwing an "overhead limit exceed" exception. The whiskers show 95% confidence intervals which are obtained after 5 experiment repetitions. The blue curve represents the results in case of Standard Flow Monitoring (STD), while the other colors indicate the measurement data when enabling SFM with different ratios of each flow type. The three subfigures highlight the CPU utilization that results from setting the ratio of one flow class to a constant while varying the ratios of the two other classes.

For all scenarios, with the same number of flows, STD consumes more CPU resources than SFM. The gap between the blue curve and the others becomes wider when increasing the number of flows and achieves a maximum value of nearly 4%. In case of SFM, the ratio of the SHORT group has the largest impact on the controller's CPU usage, which is displayed in Figure 3b and Figure 3c. When keeping the portions of MIDDLE and LONG groups constant, the highest CPU utilization is reached if the SHORT group has the highest ratio (brown curves). A reasonable explanation is that the flows in the SHORT group are queried most frequently among all groups. Therefore, a higher number of flows in this group results in more messages and tasks per second that the CPU needs to carry out and leads to a high CPU load. Nevertheless, in this worst case of SFM, the resource consumption at the controller is still less than in the case of STD.

The portion of SHORT flows is defined as a main influence factor on the CPU usage of the controller according to the above discussion. In order to perform a deeper investigation, measurements with different numbers of sub-groups in the SHORT group are carried out and the results are presented in Figure 4. There are six dashed lines in different shades of brown that correspond to different numbers of sub-groups within the SHORT group. These numbers directly affect the number of destination IP addresses in the flow table, and range from one to 50 groups. The number of flows in each group (SHORT, MIDDLE, LONG) is unchanged and follows the ratio $\rho = 1:2:3$. For example, at the first point on the x-axis, the total number of flows equals 6,000, of which 3,000 flows are classified as LONG, 2,000 flows as MIDDLE, and 1,000 flows as SHORT. In these 1,000 flows, the number of destination IP addresses changes from 1 to 50, which makes the controller generate different numbers of $FlowStatsRequest$ messages every 5 seconds, coressponding to the number of IP addresses

(also known as the number of sub-groups). However, when the controller request the statistic of all those SHORT flows, the quantity of $FlowStatsReply$ messages is constant and equals 1,000 - the number of SHORT flows are installed.

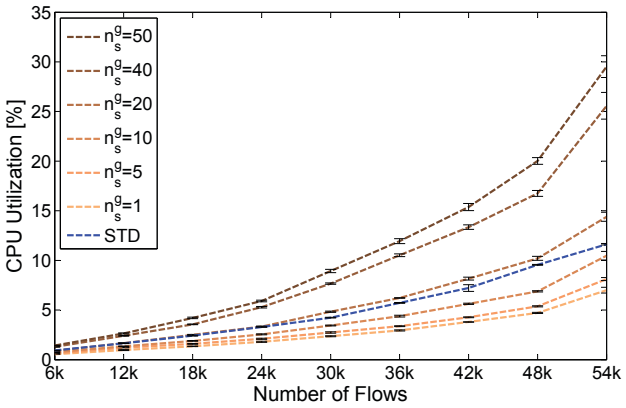


Fig. 4: Impact of the number of sub-groups in the SHORT group.

The blue line represents the data that is obtained when STD is enabled. A trade-off between using the two mechanisms is illustrated visually. When the number of sub-groups is smaller than 20, SFM shows a better result, i.e., less CPU usage. At 20 sub-groups, there is no significant difference between STD and SFM unless 54,000 flows are installed, as illustrated by overlapping confidence intervals between those two cases. However, if more sub-groups exist, more resources are required to adapt to the large number of messages that arrive at the controller. This observation implies that in some cases, SFM results in a higher CPU usage than the standard method. However, due to its ability to query the information of particular flows, SFM provides significant resource savings when the number of sub-groups is low.

B. CPU Utilization in Case of Multiple Switches

Equation 3 shows that the number of switches in the network N_{SW} is also a factor that impacts CPU utilization. In order to highlight this effect, Figure 5 displays cumulative distributions of the controller's CPU usage in both cases, SFM and STD with different values of N_{SW}. In this context, the total number of OpenVswitches varies between 1, 20, and 40 switches, represented by the line colors, respectively. The resulting distributions in case of SFM are shown as solid lines, and the dashed lines present measured data when using STD. For this evaluation, we aggregate measurements from scenarios that feature between 6,000 and 54,000 flows in increments of 6,000 flows. While the CPU load is displayed on the x-axis, the y-axis indicates the fraction of measurements in which the CPU utilization is below the corresponding threshold. Again, the memory limit for the JVM equals 512 MB, and the value $\rho = 1:2:3$ is considered for the portions of groups. Since the

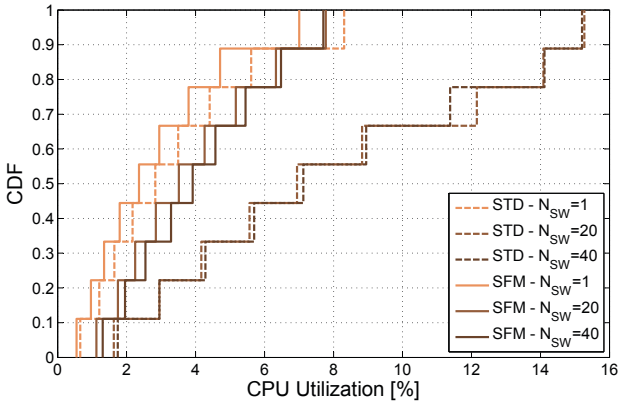


Fig. 5: Distribution of the CPU utilization for different values of N_{SW} when using STD and SFM.

number of $FlowStatsRequest$ messages is exactly equal to the amount of OpenFlow devices in the network, more switches in the network lead to more communication from the controller at each polling event. Consequently, an extra amount of work needs to be handled by the CPU and its load rises substantially in comparison to the single switch case. Especially in case of STD with a single switch, the CPU load never exceeds 9%. In contrast, when using multiple switches, the CPU utilization exceeds 10% in more than 30% of cases, as shown in Figure 5.

Additionally, the maximum observed CPU utilization when using SFM is below 8%, which equals only half the value in the context of the standard method. In both cases, increasing N_{SW} leads to higher CPU utilization. However, the growth gradually converges when the network contains more than 20 switches, as exhibited by the overlapping lines that correspond to $N_{SW} = 20$ and $N_{SW} = 40$. The largest gap between those configurations and the one featuring a single switch is significantly smaller in the case of SFM than when using STD, with the largest difference being 2% compared to nearly 7%.

C. Impact of Java Virtual Machine Memory

Due to the fact that the ONOS controller runs as a Java program, it is necessary to take into account the memory space for the Java Virtual Machine (JVM) when investigating the performance of the controller. The amount of memory that is dedicated to the ONOS controller can be controlled by means of an environment variable[7].

Figure 6 displays the maximum number of flows that the controller can handle before throwing exceptions regarding memory issues. While the x-axis denotes the number of flows which ranges between 24,000 and 408,000 in steps of 24,000, the y-axis represents the mean CPU utilization. Differently colored bars correspond to two configurations regarding the maximum assigned memory for the JVM, i.e., 2 GB and 4 GB,

[7]To allocate *minValue* of heap memory at the start and up to a maximum of *maxValue*, the following command can be used: `export JAVA_OPTS="-server -Xms[minValue] -Xmx[maxValue]"`

respectively. The whiskers provide confidence intervals that are obtained from 5 runs. The same combination of the flow class ratio ρ and polling time ratio γ as in previous experiments is used for the SFM scenario.

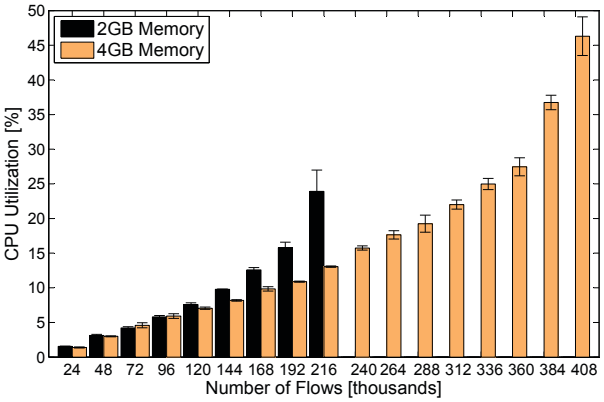


Fig. 6: Influence of JVM Memory.

The first observation is that doubling the memory for the JVM does not mean that the controller is able to process two times more flows. In case of using 2 GB, the maximum number of flows in the network is 216,000. The value when increasing the memory limit to 4 GB equals 408,000 flows and is therefore 1.89 times higher. Additionally, when the controller has more capability for storing processed information, the CPU consumption decreases. While the reduction is not significant in the context of less than 120,000 flows, it is equal to up to 45% when 216,000 flows are involved. Since the controller reserves the set amount of memory regardless of actual usage and number of flows, it is important to determine a value that balances the trade-off between potentially unused memory resources and performance benefits. Otherwise, the memory might become a performance bottleneck of the controller. This is not only true for the ONOS controller but also for other Java-based SDN controllers like OpenDaylight, Floodlight, and Beacon. Given the network state in terms of the number of active flows, an administrator can derive viable memory limits.

V. Conclusion

In this work, we compare two mechanisms for monitoring flow statistics in the ONOS controller in terms of the CPU usage at the controller. These mechanisms include the default approach that is based on regular and flow-independent polling as well as a selective approach in which flows are classified based on their requirements w.r.t. the polling frequency. Furthermore, we investigate the influence of several parameters on the performance of both approaches. This is achieved by first analyzing the total number of control plane messages that are exchanged between the switches and the controller in each case. We then use the resulting equations in order to identify the main influence factors on the performance of the monitoring approaches. Afterwards, experiments in a testbed featuring the ONOS controller and a Mininet-based network are performed in order to investigate the quantitative influence of parameters like the number of flows and switches, as well as the size ratios of flow classes.

The measurement results show that in most cases, SFM performs significantly better in terms of CPU usage than the standard method. However, polling more groups with the same interval can result in higher CPU utilization due to the fact that statistics are queried on a per-group basis rather than on a per-switch basis in the context of SFM. Furthermore, investigations regarding the amount of memory that is allocated to the Java Virtual Machine indicate that not only the hardware in terms of the CPU is relevant but that the RAM also needs to be taken into account. Given the results presented in this work, an operator can identify appropriate parameter combinations based on the composition of the network and flow characteristics.

There are several directions for future work. On the one hand, experiments can be extended by generating traffic in the data plane in order to assess the accuracy of monitoring mechanisms. On the other hand, the effects of the control plane overhead on the network devices can be evaluated by tracking the resource usage on the machine that hosts the Mininet environment. Finally, an appropriate controller model could provide additional insights into the trade-offs between different monitoring approaches.

VI. Acknowledgments

This work has been performed in the framework of the IuK project SDN-PERF and is partly funded by the Bavarian Ministry of Economic Affairs and Media, Energy and Technology (Project ID IUK-1507-0003). The authors alone are responsible for the content of the paper.

References

[1] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, Attributes, and Use Cases: A Compass for SDN," *Communications Magazine, IEEE*, 2014.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM CCR*, 2008.

[3] "OPEN-TAM: Traffic Analysis and Monitoring," ONOS. [Online]. Available: https://wiki.onosproject.org/display/ONOS/OPEN-TAM%3A+Traffic+Analysis+and+Monitoring

[4] "OpenFlow Switch Specification version 1.3," The Open Networking Foundation. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[5] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A Low Cost Netowrk Monitoring Framework for Software Defined Networks," in *14th IEEE/IFIP Network Operations and Management Symposium (NOMS 2014)*, 2014.

[6] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network Monitoring in OpenFLow Software-Defined Networks," in *Network Operations and Management Symposium (NOMS)*, 2014.

[7] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "Flowcover: Low-cost flow monitoring scheme in software defined networks," in *2014 IEEE Global Communications Conference*, 2014.

[8] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: Traffic matrix estimator for openflow networks," in *PAM*, 2010.

[9] L. Yuan, C. N. Chuah, and P. Mohapatra, "Progme: Towards programmable network measurement," *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, pp. 115–128, Feb 2011.