ZOOM: Lightweight SDN-based Elephant Detection

Steffen Gebert, Stefan Geissler, Thomas Zinner, Anh Nguyen-Ngoc, Stanislav Lange, Phuoc Tran-Gia
University of Wuerzburg, Germany
Email: {steffen.gebert|stefan.geissler|zinner|anh.nguyen|stanislav.lange|trangia}@informatik.uni-wuerzburg.de

***Abstract*—Software Defined Networking (SDN) offers a holistic view of the network through a centralized control plane. Consequently, routing decisions can be made based on global knowledge about the network topology as well as its current state. As long living flows are suitable candidates for rerouting, their detection is crucial for efficient flow based traffic management. This work proposes the ZOOM algorithm for elephant detection in SDN networks. To this end, ZOOM follows a very lightweight approach that only uses packet counters implemented by OpenFlow switches and thus does not require any additional hardware. By exploiting this feature of OpenFlow switches, ZOOM allows lightweight and cost-effective elephant detection.**

I. Introduction

The switching fabric and hardware of modern networks have to handle the continuosly growing demand for high bandwidth applications that leads to a rapid growth of total network traffic. To counter this trend, datacenters implement highly sophisticated network topologies and employ traffic engineering as it is a crucial aspect when it comes to efficiently using available resources. Furthermore, the adoption of SDN by telecommunication providers enables the deployment of traffic management functionality in backbone networks. Through intelligent allocation of flows to specific network paths within the network, resource utilization can be increased significantly. As reported in [1], a 1,500 server cluster is subject to up to 100,000 flow arrivals per second. This makes allocating a path to each of the flows an impossible task. Hence, modern network monitoring tools often perform elephant detection – the detection of flows that carry a large portion of the total network traffic. By selecting this subset of flows and allocating them to specific network paths, the available bandwidth can be used more efficiently. Most previously proposed mechanisms for identifying elephant flows are either not scalable or introduce a huge measurement overhead that puts additional strain on the network components, while others require modifications on host or application level [2]. In constrast, the approach presented in this paper leverages means of flow monitoring that come with the introduction of Software Defined Networking (SDN) and OpenFlow.

The proposed detection mechanism is based on network-side monitoring and thus does not require any support by the end hosts or involved applications. Instead, it leverages already existing counters of OpenFlow switches that are automatically updated whenever a packet matching an existing flow rule passes the switch. Through iterative refinement of flow rules, the proposed algorithm allows narrowing down the aggregation level from coarse grained rules to very fine grained rules and counters. Hence, the proposed *ZOOM* algorithm offers a very lightweight monitoring solution focused on elephant detection that requires no additional hardware. While it is unlikely to achieve 100% accuracy when following such a lightweight approach to traffic monitoring from the network side, it can serve as a preselection stage for subsequent in-depth analyses. The algorithm is evaluated with respect to its accuracy using a publicly available traffic trace. In order to analyze the algorithm's capability of identifying high-bandwidth flows (elephant flows), the produced results are evaluated using global knowledge about the used traffic trace.

The remainder of this work is structured as follows. An overview of related work and SDN monitoring solutions is provided in Section II. The ZOOM algorithm is introduced and evaluated in Sections III and IV. Limitations and possible extensions of the proposed mechanism are discussed in Section V, before Section VI summarizes this work.

II. Related Work

This section covers relevant research work on the topic of elephant detection and SDN-based monitoring.

A. Elephants and Mice

The relation between large and small (in terms of bytes), as well as short- and long-lived flows is discussed in [3]. The elephants, small in number, carry the biggest part of the entire traffic while the mice, large in number, only contribute a small part of the total traffic volume. This has been observed in different data center measurement studies [4, 1]. Different approaches for elephant detection are summarized in the following.

Application-side labeling: One approach enabling application-specific flow monitoring is to label network flows within the application [5, 6]. Thus, every flow present in the network is labeled, e.g., with the type of application or the total amount of data to transfer. This approach, however, requires the modification of all involved applications as well as a dedicated trust relationship between the hosts and the network.

End-Host-Based detection: Instead of in the application, monitoring can also be performed in the operating system on the end-hosts. *Mahout* [2] detects elephant flows by monitoring socket buffers of end-hosts and is realized via a shim layer that marks packets which belong to an elephant flow before emitting them into the network. However, again the network has to trust information coming from the end-hosts.

Network-side flow statistics: Another approach, and probably the most common one, is to monitor within the network to keep statistics for every flow present at a given time. Statistics are exported from the network entities to the monitoring stations. Systems using this approach beside *sFlow* are [7] and [8]. Providing such per-flow statistics for every flow in the network requires a severe amount of resources on the network elements, which might not be possible for all environments, especially data center or WAN networks. While packet sampling allows to apply this mechanism to even larger network environments, elephant detection becomes yet harder with higher sampling factors [9].

B. SDN Monitoring

Programmability and packet matching in hardware switches offers multiple ways of using OpenFlow for monitoring. The most important approaches are described in the following.

NEC FlowSense [10] uses OpenFlow messages (*flow-removed* and *packet-in*) to measure the duration of flows. Furthermore, the amount of traffic measured via flow counters and inbound ports is logged in order to provide detailed information on link utilization. *OpenTM* [11] leverages OpenFlow features for passive monitoring and is based on periodically querying flow statistics from selected switches. Furthermore, the authors investigate the trade-off between the load of the switches and the accuracy in terms of flow rate estimation. In contrast to the proposed ZOOM algorithm, FlowSense and OpenTM do not actively define flow rules besides the ones set up by the controller for forwarding traffic. While *OpenSketch* [12] offers a large feature set for flexible flow monitoring in SDN hardware, it relies on specialized, programmable hardware. In contrast, the proposed ZOOM approach exploits existing features of OpenFlow hardware.

Currently available commercial SDN-based monitoring systems require large investments, e.g., *Big Tap Monitoring Fabric* [13] by Big Switch Networks, which is based on an out-of-band monitoring infrastructure comprising additional SDN switches. All production traffic is mirrored into fabric that then splits up the flows to different monitoring servers. VSS Monitoring's *Network Packet Brokers* (NPB) follow a similar approach. A central controller manages a certain number of NPBs, which analyze and forward the monitoring traffic through a dedicated network, the *vMesh* [14].

III. ZOOM: Network Monitoring using SDN

This section describes the ZOOM algorithm, a lightweight approach leveraging packet counters in OpenFlow switches for elephant detection. After the algorithm is introduced, its accuracy is evaluated using a prototypical implementation.

A. Elephant Flows

Elephant flows are often defined by a certain, fix data volume they transfer, e.g. 100 MB. This work follows a slightly different approach as we use the definition provided in [15]. Lan et al. define elephants as flows that carry more data than the mean of all flows plus three times the standard deviation. We will base our definition on this metric while introducing a new variable called s_{ele} which enables us to define different elephant thresholds by using different values for s_{ele} in the following equation.

$$S = mean(\text{flowsize}) + s_{ele} \times std(\text{flowsize}) \qquad (1)$$

Thereby, $mean(\text{flowsize})$ refers to the average size of all flows present in the system and $std(\text{flowsize})$ describes the standard deviation of the encountered flow sizes. This definition leads to a threshold for elephant flows that is based on the traffic in the network and can thus be used independently of the monitored network.

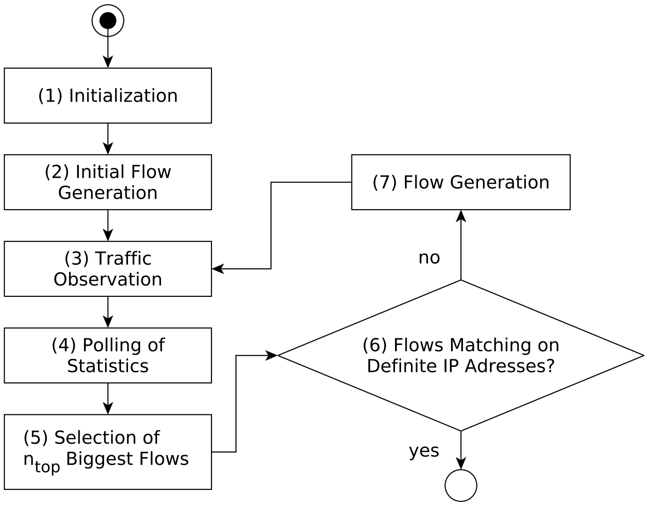
(1) Initialization
(2) Initial Flow Generation
(3) Traffic Observation
(4) Polling of Statistics
(5) Selection of n_{top} Biggest Flows
(6) Flows Matching on Definite IP Adresses?
no
(7) Flow Generation
yes

Fig. 1. Flow chart of the ZOOM algorithm.

B. The ZOOM Algorithm

The proposed algorithm is based on polling of flow statistics from OpenFlow switches. Such statistics are automatically maintained by the switches for each currently installed flow rule. Therefore, the main idea is to define flow entries that do not modify the forwarding behavior of switches, but still enable flow monitoring via packet counters. An iterative refinement of the IP address ranges matched by the flow rules then allows to narrow down the elephant flows. The match fields of these flow entries are set so that – in the simplest case – a binary search over the whole IP space is performed for source and destination IP addresses. Thus, the IP range of possible source and destination addresses is divided into sections, each covered by one of the created flow rules, which then trigger statistics collection within the switch. By splitting up IP ranges into more than two parts, the algorithm can proceed faster, however, at the cost of creating more flow entries.

In contrast to NetFlow/sFlow-style monitoring, counters are not created for every single flow. Instead, the amount of data that needs to be processed by the algorithm is independent of the number of flows present in the system. In addition, if prefixes of the IP source or destination addresses are known, the run time can be shortened even further. In particular, the runtime grows linearly with the number of wildcard bits in the IP range that is to be searched. The following equation describes the runtime of the ZOOM algorithm.

$$t_{\text{ZOOM}}(n_{bit}, n_{flows}, n_{top}, t_{wait}) = \frac{n_{bit}}{\log_2(\frac{n_{flows}}{n_{top}})} \times t_{wait} \qquad (2)$$

Thereby, $n_{bit} \in [0, 32]$ is the number of remaining wildcard bits in the IP range.

The ZOOM algorithm is listed in Algorithm 1, visualized in Figure 1, and described in the following.

1) Initialization: The algorithm's behavior can be adjusted with three input parameters. The first parameter n_{flows} defines the number of flow entries that are created per cycle. This defines the number of sections into which the remaining IP range is divided. While $n_{flows} = 2$ corresponds to a binary search, higher values allow faster advancing at the cost of more flow rules. How many of the n_{flows} sections covering the most traffic are treated as candidates to contain elephants and are thus further analyzed is defined by the n_{top} parameter. This also determines the total number of elephant flows contained in the output produced by the algorithm. Finally, t_{wait} defines the waiting time in seconds between the creation of flow rules and polling of corresponding statistics. Hence, it represents the interval during which passing traffic is monitored.

2) Initial Flow Generation: As the algorithm searches for source and destination addresses of end-to-end flows, it is required to search the whole IP address range for both sources and destinations concurrently. Hence, the number of flow rules that are generated is n_{flows}^2. In this first step, n_{flows} flow entries covering all addresses from 0.0.0.0 to 255.255.255.255 are generated for source and destination respectively. If not the whole IP range needs to be covered, the initial flow rules change depending on the already known bits of the IP address range. For $n_{flows} = 2$, the resulting flow entries are given in Table I. It can be seen that all fields except source and destination IP are wildcards. Instead of matching pairs of definite IP addresses, the set of possible pairs of source and destination addresses is partitioned into a number of n_{flows}^2 flow entries.

3) Traffic Observation: During t_{wait}, while the algorithm pauses, the packet counters of the switch are automatically updated for packets matching one of the specified flow rules.

4) Polling of Statistics: After t_{wait}, flow statistics are requested from the switch.

5) Selection of n_{top} Biggest Flows: Packet counters of previously retrieved flow statistics are evaluated and the n_{top} biggest flows regarding average bandwidth are selected for further processing.

6) Termination Condition: The subsequent action is to check whether the termination condition is satisfied. This is the case if the section covered by each of the selected n_{top} biggest flows contains only connections between a definite source/destination IP address pair, i.e., if all 32 bits of the source and destination IP match fields are specified. If this termination condition is met, the identified n_{top} biggest flows are returned as result of the algorithm. As long as the IP address match still contains wildcard bits, execution continues.

7) Flow Generation ("Zoom In"): Again, a number of n_{flows} flow entries is defined. Using these, the source and destination IP ranges covered by the previously found n_{top} biggest flows are split up into $\frac{n_{flows}}{n_{top}}$ entries each. After removing all previously defined flow rules the newly generated entries are pushed to the switch and the algorithm is repeated starting from Step 3.

Figure 2 illustrates the refinement process using $n_{flows} = 4$ and $n_{top} = 2$. Depicting the first 4 cycles of the algorithm, it can be seen that the $n_{top} = 2$ sections covering the biggest traffic (shown in black/gray) get iteratively refined by getting split into $\frac{n_{flows}}{n_{top}} = 2$ segments each.

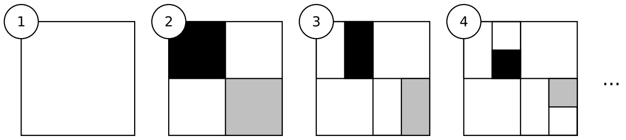

Fig. 2. Example of refinement ("Zoom In" step) for $n_{flows} = 4, n_{top} = 2$.

C. Implementation

In order to evaluate the accuracy of the proposed algorithm, a proof-of-concept has been implemented[1] as a module for the OpenDaylight controller (ODL).

The current implementation is limited to statistics collection. In order to still allow correct forwarding of production traffic, the action of the flow entries defined by ZOOM should, e.g., set the *goto-table* action to a table containing the actual forwarding rules. This, however, is not relevant for evaluating the ZOOM algorithm's accuracy. Contrary to the proposed algorithm, our prototype implementation is limited with respect to legal values regarding the parameters introduced earlier. This leads to the following constraints:

$$n_{flows} \in \{1, 2, 4, 16\} \quad (3)$$
$$n_{top} \in \{1, 2, 4, 8\} \quad (4)$$
$$\frac{n_{flows}}{n_{top}} \in \{1, 2, 4\} \quad (5)$$

Furthermore, ODL does not allow creating flow entries that match the 0.0.0.0 IP address. Therefore, the proof-of-concept implementation of the ZOOM algorithm has to create more than n_{flows} entries in the initial step (Step 1 in Figure 1). Instead of 0.0.0.0/1 and 128.0.0.0/1, a reasonably low number of matches[2] are defined. Afterwards, the algorithm continues as originally described in Section III.

IV. Evaluation

The results that are presented and discussed in the following are obtained by running the aforementioned OpenDaylight implementation. The accuracy is evaluated by replaying a publicly available traffic trace from the *Waikato Internet Traffic Storage* (WITS). Characteristics of the trace that is used to evaluate the ZOOM algorithm are listed in Table II. In addition to the general trace information, the table shows statistics of elephant flows resulting from different elephant thresholds. The values of $s_{ele} \in \{1, 10, 30\}$ are selected as they show that the total number of elephants and s_{ele} behave roughly inversely proportional. The maximum value for $s_{ele} = 30$ is chosen since, due to the low elephant density, the accuracy decreases significantly for this threshold. Furthermore, the average number of active elephants at each point in time as well as the average duration of elephant flows is provided. Finally, the table shows the traffic contribution and the percentage of elephants compared to the total number of flows.

[1] Source code: https://github.com/lsinfo3/zoom-odl

[2] 1.0.0.0/8, 2.0.0.0/7, 4.0.0.0/6, 8.0.0.0/5, 16.0.0.0/4, .., 240.0.0.0/4

TABLE I. EXAMPLE OF GENERATED FLOW ENTRIES MATCHING THE WHOLE IP RANGE (n_{flows} =2).

Input Port	Ethernet Type	Source MAC	Dest MAC	Source IP	Dest IP	ToS	Source Port	Dest Port	Protocol
*	0x800	*	*	0.0.0.0/1	0.0.0.0/1	*	*	*	*
*	0x800	*	*	0.0.0.0/1	128.0.0.0/1	*	*	*	*
*	0x800	*	*	128.0.0.0/1	0.0.0.0/1	*	*	*	*
*	0x800	*	*	128.0.0.0/1	128.0.0.0/1	*	*	*	*

Algorithm 1: The ZOOM Algorithm.

```
Input: $n_{flows} \in \{2, 4, 16\}$, $n_{top} \in \{1, 2, 4, 8\}$
Assert $\frac{n_{flows}}{n_{top}} \in \{2, 4, 16\}$;
$S_{part} = \{n_{flows}$ partitions of source IP range$\}$;
$D_{part} = \{n_{flows}$ partitions of destination IP range$\}$;
Generate flow entries $E_{init} = S_{part} \times D_{part}$;
Push $E_{init}$ into switches;
Sleep for $t_{wait}$ seconds;
Set foundFlowRuleToRefine = true;
while foundFlowRuleToRefine do
    Collect flow statistics;
    Set foundFlowRuleToRefine = false;
    for $i = 1$ to $n_{top}$ do
        Select $i$-th biggest flow;
        $S$ = Extract source IP range;
        $D$ = Extract destination IP range;
        if $|S| > 1$ or $|D| > 1$ then
            Set foundFlowRuleToRefine = true;
            $S_{part} = \{\frac{n_{flows}}{n_{top}}$ partitions of $S\}$;
            $D_{part} = \{\frac{n_{flows}}{n_{top}}$ partitions of $D\}$;
            Generate $E_i = S_{part} \times D_{part}$ flow entries;
        end
    end
    Remove previously installed flow rules;
    Push $E_1, \ldots, E_{n_{top}}$ to switches;
    Sleep for $t_{wait}$ seconds;
end
Write results;
return;
```

The evaluation is performed by using *tcpreplay* to replay the trace into an Open vSwitch connected to ODL running the ZOOM module. In order to calculate the accuracy of the algorithms, the experiment results are compared to the data available due to global knowledge about the traffic trace. This is obtained by processing[3] the packet trace to obtain information on which flows (and elephants) are active at which time of the trace. These are then matched against the data obtained by the algorithm to calculate the accuracy of the results.

In order to conduct the measurements, the algorithm is started at different time offsets (5, 10, 20, 30, 40, 50 seconds) for every combination of parameters while replaying the traffic trace. In the following, the influence of different parameter settings on the accuracy of the ZOOM algorithm is evaluated.

The evaluation is performed by analyzing the accuracy of the results produced by the ZOOM algorithm. Accuracy thereby describes the percentage of retrieved results that are relevant and represents the precision in a standard precision and recall scenario. Hence, the accuracy represents the fraction of the detected flows that are considered true positives.

TABLE II. ISPDSL-II TRAFFIC TRACE.

Trace Metadata			
Trace URL	http://wand.net.nz/wits/ ... ispdsl/2/20100106-030946-0.dsl.php		
Trace duration	1214 seconds		
Traffic type	DSL Subscribers		
Total number of flows	1,124,575		
Concurrently active flows (min/avg/max)	3,641 / 18,916 / 20,919		
Elephant Statistics	$s_{ele} = 1$	$s_{ele} = 10$	$s_{ele} = 30$
Total number of elephants	2,760	362	91
Average active elephants	1,062	186	56
Average duration (seconds)	443.9	595.0	715.4
Traffic contribution	77.6%	47.1%	28%
Count contribution	0.2%	0.03%	0.008%

TABLE III. INFLUENCE OF WAITING TIME t_{wait} ON THE ACCURACY.

t_{wait}	Mean Accuracy	Confidence Inteval
1	0.3969	0.0224
2	0.4710	0.0201
5	0.5236	0.0167

A. Time of Traffic Observation (t_{wait})

The study investigates the impact of the length of the monitoring interval t_{wait}, during which the flow entries are active in the switch and passing traffic is monitored. Therefore, we examine the influence of t_{wait} on the mean accuracy over all available parameter combinations. Table III shows the mean accuracy as well as the 95% confidence interval of the ZOOM algorithm for $t_{wait} \in \{1, 2, 5\}$. It can be seen, that the accuracy increases with growing t_{wait} for the evaluated trace. This can be explained by the influence t_{wait} has on the behavior of the algorithm. While low t_{wait} values decrease the overall runtime of the algorithm and thus allow for faster detection of elephants, the short observation time makes the algorithm vulnerable to short, bursty transmissions that are not elephants by definition. Higher values of t_{wait} on the other hand, increase the traffic monitoring interval and the algorithm becomes more robust against short-lived flows. As the flows that are to be detected are the ones that transmit large amounts of data, these flows also have a higher duration.

Hence, the parameter choice of t_{wait} controls the trade-off between short runtime and fluctuation resistance.

B. Elephant Threshold (s_{ele})

This section examines the influence of the elephant threshold s_{ele} on the accuracy of the ZOOM algorithm. As defined in Section III-A, the threshold s_{ele} describes the total amount of data a flow has to transmit during its lifetime in order to be considered as elephant. Figure 3 shows the data obtained by calculating the accuracy for $s_{ele} \in \{1, 10, 30\}$.

The figure shows that for a high elephant threshold the t_{wait} parameter has significant influence on the accuracy. For

[3] Source code: https://github.com/lsinfo3/zoom-evaluation

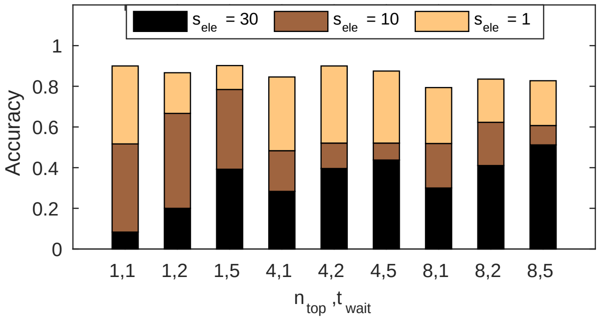

Fig. 3. Influence of s_{ele} and t_{wait} on the accuracy.

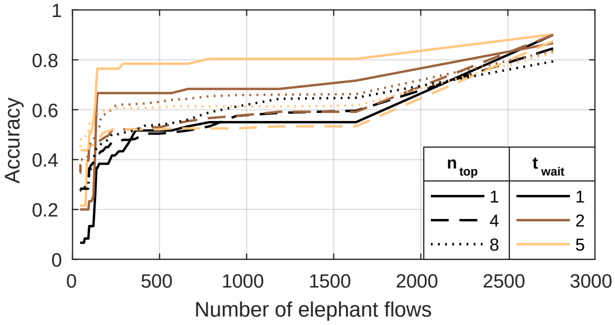

Fig. 4. Accuracy for $n_{flows} = 16$, different n_{top}, t_{wait} and elephant thresholds.

lower s_{ele} on the other hand, the influence is less decisive. This is due to the composition of elephants for different thresholds. As a high s_{ele} leads to a small set of large, long-living elephants that feature a high mean duration, the high t_{wait} value leads to increased accuracy as short-lived flows are filtered by the long observation period. On the other hand, decreasing s_{ele} leads to a larger amount of flows that are considered elephants. This reduces the mean duration of elephants which results in t_{wait} not having a significant influence. Due to the now larger set of elephants, the overall accuracy is increased.

C. Number of Requested Elephant Flows (n_{top}) and Available Flow Rules (n_{flows})

In the following, the relation between the ratio of n_{flows} and n_{top} and the accuracy of the ZOOM algorithm is evaluated. Figure 4 shows the accuracy for different parameter combinations and elephant thresholds. Thereby, the parameter n_{top} is indicated by the line style while the color describes the value of t_{wait}. The x-axis describes the total number of elephants resulting from different elephant thresholds as described in Section III-A ($s_{ele} \in [1, 30]$). The data indicates that most parameter combinations achieve similar accuracies for similar elephant thresholds. Only the parameter combinations of $n_{top} = 1$ and $t_{wait} \in \{2, 5\}$ show significantly different behaviour. This is most likely due to the higher granularity resulting from $n_{flows} = 16$ and $n_{top} = 1$. This fine grained segmentation reduces the risk of falsely selecting a flow aggregate of many small flows instead of a single elephant flow. As our proof-of-concept implementation does not support backtracking, a once deselected segment is never investigated again in further steps of the algorithm. We call this the *Aggregation Problem*. This may lead to false positives as the algorithm eventually selects an aggregate of many small flows with a large cumulative size over a single large flow.

Furthermore, the results show that if the elephant threshold is set too high, the accuracy of the algorithm decreases significantly. This can also be explained by the aggregation problem. As the number of elephants decreases with a growing threshold, the average size of non-elephant flows and thereby the extent of the aggregation problem increases even further.

V. Discussion / Future Work

The presented ZOOM approach allows several options for future extensions. Some of them are discussed in the following.

Evaluation using hardware switches: The evaluations presented in this work have been conducted using an Open vSwitch software switch. Evaluations using different hardware OpenFlow switches[4] failed for different reasons, including a lack of control plane performance, as well as flow statistics being returned only partially or not at all. Such issues have already been reported in [16]. Assuming compatible hardware switches, identical accuracy can be expected.

Use of hard timeouts: Instead of triggering flow statistics collection after t_{wait}, it is assumed that the accuracy can be further improved by setting hard timeouts of the flow entries to t_{wait} and thus let the switch report back statistics counter exactly after t_{wait}. This would avoid different life times of the flow entries between the rules pushed first and last and thus a falsification of traffic counters. As mentioned, the processing speed of flow-mod messages in current OpenFlow hardware is far from ideal [16].

Transport Layer ports and protocol: Currently, only IP addresses are returned by the ZOOM algorithm. As searching for port numbers and protocols does not offer the bit-wise matching of IP addresses, a detection of flows could, e.g., be established by scanning well-known ports and protocols first.

Multiple flow tables: More advanced use cases of matching header fields could make use of multiple flow tables in OpenFlow switches.

Specialized ZOOM version: The generic ZOOM algorithm can be adjusted to detect elephant users [17] (users generating significant amounts of traffic) instead of elephant flows.

VI. Conclusion

This work introduces a lightweight algorithm for elephant detection which leverages built-in features of OpenFlow enabled switches. In particular, the algorithm combines the dynamic creation of flow entries and the counters maintained by switches to perform cost-effective elephant detection without introducing network overhead. Thus, the proposed mechanism does not require modification of software or any additional hardware. Furthermore, the amount of data that needs to be analyzed is independent of the amount of traffic present in the network.

The ZOOM algorithm is capable of detecting elephant flows and their source and destination IP addresses. To this end,

[4] NEC PF5240, Pronto 3290, HP 2920-24G

the algorithm defines coarse-grained flow entries in OpenFlow switches, monitors them, and iteratively refines them until they match specific IP addresses. Evaluations of our proof-of-concept implementation show that an accuracy of more than 80% can be achieved.

Our analysis shows that the ZOOM approach for elephant detection is a promising concept. In order to improve the algorithm itself as well as to overcome the limitations of our current implementation, further investigations are required. In-depth analyses of the behavior of elephant flows and further mechanisms provided by SDN and OpenFlow enabled switches may improve the accuracy and performance of the ZOOM algorithm in the future.

Acknowledgments

This work has been performed in the framework of the SARDINE project and is partly funded by the BMBF (Project ID 16KIS0261) and DFG (CRC 1053, MAKI). The authors alone are responsible for the content of the paper.

References

[1] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[2] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-Overhead Datacenter Traffic Management Using End-Host-based Elephant Detection," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1629–1637.

[3] N. Brownlee and K. Claffy, "Understanding Internet traffic streams: dragonflies and tortoises," *Communications Magazine, IEEE*, vol. 40, no. 10, pp. 110–117, Oct 2002.

[4] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280.

[5] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-Based Application-Aware Networking on the Example of YouTube Video Streaming," in *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, Oct 2013, pp. 87–92.

[6] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Ko, J. Rexford, and M. J. Freedman, "Serval: An End-Host Stack for Service-Centric Networking," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX, 2012, pp. 85–98.

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *NSDI*, vol. 10, 2010, pp. 19–19.

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 339–350, 2011.

[9] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto, "Identifying Elephant Flows Through Periodically Sampled Packets," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 115–120.

[10] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost," in *Passive and Active Measurement*. Springer, 2013, pp. 31–41.

[11] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks," in *Proceedings of the 11th International Conference on Passive and Active Measurement*, ser. PAM'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 201–210.

[12] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 29–42. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/yu

[13] B. S. Networks, "Big Tap Monitoring Fabric Ver 4.5 Datasheet." [Online]. Available: http://bigswitch.com/sites/default/files/big_tap_monitoring_fabric_v4.5.pdf

[14] "The VSS Unied Visibility Plane," 2015. [Online]. Available: http://www.vssmonitoring.com/unified-visibility-plane/pdf/Unified-Visibility-Plane-Whitepaper.pdf

[15] K.-c. Lan and J. Heidemann, "A measurement study of correlations of internet flow characteristics," *Computer Networks*, vol. 50, no. 1, pp. 46–62, 2006.

[16] M. Kuzniar, P. Peresini, and D. Kostic, "What You Need to Know About SDN Flow Tables," in *Passive and Active Measurements Conference (PAM)*, no. EPFL-CONF-204742, 2015.

[17] P. Liu, F. Liu, and Z. Lei, "Model of network traffic based on network applications and network users," in *Computer Science and Computational Technology, 2008. ISCSCT'08. International Symposium on*, vol. 2. IEEE, 2008, pp. 171–174.