Monitoring Flows with Per-Application Granularity using Programmable Data Planes

Rhaban Hark, Mohamed Ghanmi, Ralf Kundel, Patrick Lieser, and Ralf Steinmetz
Multimedia Communications Lab, Technische Universität Darmstadt, {first.last}@kom.tu-darmstadt.de

***Abstract*—The accurate and timely knowledge of a network's internal state is essential for various network management operations like routing, resource allocation, or even intrusion detection. This especially holds true for highly flexible, programmable networks that quickly react to dynamic conditions. However, current approaches of state monitoring in such networks rely on per-rule counter information. Due to limited rule space, their granularity is strongly limited. This generally yields an aggregated and therefore altered representation of the network state. Utilizing the programmability of today's data planes, we tackle this problem and present a novel approach to increase the measurement granularity up to per-application statistics. For demonstration purposes, we show how our approach greatly improves the estimation of the Flow Size Distribution.**

I. Introduction and Motivation

Currently existing monitoring techniques in SDN systems such as OpenFlow [1] or P4 [2] can provide the controller with statistics for every forwarding/monitoring rule in their scope. These statistics can be actively fetched by the controller whenever management applications desire an update. Rules that hold traffic statistics can be installed either proactively or reactively on newly arriving flows. In the first case, the granularity of the rules is limited by the fact that any potentially traversing flow must be known in advance and handled adequately. In the latter case, the granularity can be set more flexibly. In both cases, however, the number of available forwarding/monitoring rules is severely limited by the size of the expensive TCAM memory. As a result, a single rule often directs multiple independent streams with overlapping properties such as the source and destination.

Given the likelihood of multiple streams of different applications getting merged into a single rule, management applications can only see an aggregated traffic view. Nevertheless, a wide variety of network management applications, e.g., Heavy Hitter Detection or DDoS attack detection, require a detailed view of the traffic with maximal statistics granularity. Sticking to the DDoS attack detection example, a popular DDoS attack option is to flood the victim with a tremendous number of TCP SYN packets. This would result in numerous streams that can easily be detected with a per-application statistics granularity. However, in nowadays networking switches where a huge number of DDoS and non-DDoS flows are aggregated into a single rule, their distinction is not easily possible as the monitoring information becomes aggregated as well.

This work has been performed in the framework of the CELTIC-NEXT project AI-NET-PROTECT (16KIS1295) and it is partly funded by the German Research Foundation (DFG) as part of subproject B1 and B4 within the Collaborative Research Center (CRC) 1053 MAKI.

To this end, we target a practical solution to monitor flows with application-level granularity in the data plane. While allowing management applications to aggregate traffic into a limited set of rules, we provide the control plane with information on the existence of contained subflows, as well as their statistics. We achieve this by taking advantage of recent programmable data plane technologies to leverage omnipresent meta information of TCP packets. By avoiding deep inspection of packets, the approach can cope with any TCP traffic, particularly including encrypted traffic.

We provide an expressive evaluation of the approach showing the proof-of-concept and the improvement in measurement accuracy for monitoring applications when, e.g., measuring the Flow Size Distribution, which is relevant for numerous applications as for instance DDoS attack detection.

The paper is structured as follows: The next section, Section II, describes the developed design of our proposed measurement approach. This includes the overarching architecture as well as a system model. Thereafter, Section III presents the evaluation of our prototypical implementation, including results. We note known limitations in Section IV and show related approaches in Section V. Finally, Section VI gives a conclusion of the paper and an outlook.

II. Fine-Grained Statistics Collection Design

In the following, we present the system architecture, explain how the approach can be integrated in SDN networks, and detail the data plane measurement approach.

A. Architecture Overview

This work proposes an entirely software-based solution to enhance the measurement quality in an SDN environment. Consequently, we ensure that no additional network elements need to be integrated for the system to function correctly. Two main components build the system architecture: *(i)* On the one hand, a P4 application is deployed on a set of switches in the data plane to conduct the desired measurements. The set of switches or measurement points should be placed strategically to minimize the measurement redundancy and along with that the monitoring overhead [3]. The selected points running our proposed P4 data collecting application reflect a real-time representation of the network state and provide raw information to the control plane. *(ii)* On the other hand, an application consuming the raw measurements should be hosted

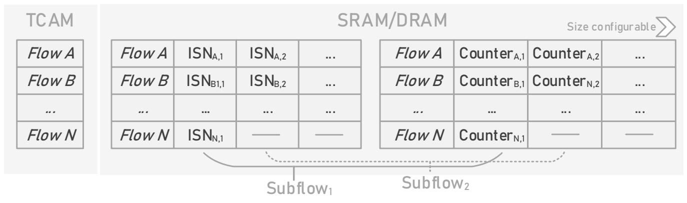


Fig. 1: Registers set up tables to *(i)* store the ISNs of SYN packets to identify subflows and *(ii)* to count packets for the corresponding subflows.

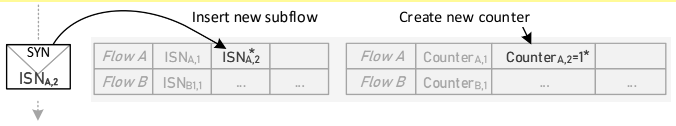
in the control plane. It has the role of collecting statistics and combining them to gain higher level insights about the state of the underlying network and yet offering these preprocessed information to other management applications.

B. Flow Model

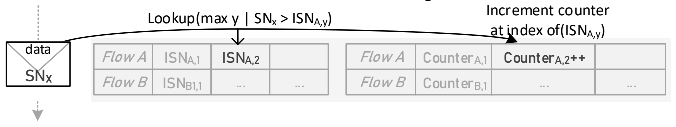
Before the function principles are explained, we give the definition of the term flow we use in the scope of this work as there is no common definition in literature. We consider a flow to be a stream of packets adhering to the same rule. Thus, one rule processes exactly one flow in an SDN environment. However, when logically independent streams from different applications (denoted *subflows*) match the same rule, they are seen as a single "aggregated" flow by the network and its monitoring application. This alters the measured network-wide state as described in Section I. We refer to this phenomenon as the aggregated flow problem. In the context of TCP/IP packets, the most present traffic in today's Internet [4], header fields can be used to identify the flow a certain TCP packet belongs to. Within the presented approach, any of the available fields can be used to *match* one flow. A typical example could be the combination of the source and destination address.

C. Subflow Measurement

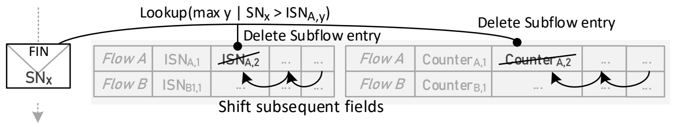
In order to solve the aggregated flow problem and, thus, provide per-application or subflow counter, we introduce a set of registers used to store the required information. Addressing the registers in the presented approach does not require, in contrast to existing approaches, additional TCAM memory such that space restrictions are more relaxed [5]. Using the registers we model two tables. Figure 1 shows both tables right of the existing TCAM table that identifies flows. Since the proposed mechanism is solely part of the performed action while processing a packet, the TCAM table entries are not modified and, thus, no additional space is requirement within the TCAM. The first table holds information on the existence of subflows within each flow. For each such flow, the first table contains an ordered list of TCP Initial Sequence Numbers (ISNs) to identify each subflows as explained later. The second table holds the counter of each identified subflow. Therefore, it follows the structure of the first table and contains the number of packets, thus, the subflow counter instead of their ISN. Entries in both tables are aligned, meaning that the index of a subflow identifier in the first table is equal to the index of



(a) On arrival of a SYN packet, it is stored to identify this new stream. Furthermore, the second table initializes a packet counter.


Lookup(max y | SN_x > ISN_A,y)
Increment counter at index of(ISN_A,y)
data
SN_x
Counter_A,2++

(b) On arrival of data packets, their SN is used to identify the subflow. This information helps to identify the counter that is increased.



(c) On arrival of a FIN packet, the subflow ended. The SN identifies the subflow and allows the removable of the identifier and its counter.

Fig. 2: Usage of the subflow identification and counting tables.

its corresponding counter in the second table. Figure 1 shows the tables' correspondence, how flows are identified, and the identifier aligned with the counter.

The width of the two tables must be set prior to runtime, but is easily adjustable in our implementation. As also visible in the last row within the tables, it is of course usually the case that not all subflow fields are in use and are therefore left blank. In the figure, the last flow, Flow N has for example only one subflow (identified by ISN$_{N,1}$ and Counter$_{N,1}$, respectively).

a) Subflow Counter Identification: As already touched upon, we leverage omnipresent TCP information to obtain separate subflow measurements. The ISN is a 32-bit number assigned to the first packet in each TCP flow. The sequence number of all subsequent packets is an increment of the Initial Sequence Number. Traditionally, the ISN is chosen by a clock that is incremented every 4 microseconds. Modern TCP implementations can, however, involve a random setting of the ISN. To identify different subflows within an aggregated flow, we store the ISN whenever a new TCP subflow passes through the measuring point. As shown in Figure 2a, on the arrival of a SYN packet, the first table inserts an entry with the ISN for the corresponding flow. On top of that, the second table initializes a counter for that subflow.

b) Subflow Packet Counting: Subsequently, we use the data packet's sequence numbers (SN) to map them to the correct counters. Figure 2b shows that behavior in detail: The SN included in arriving data packets is used to look up the corresponding subflow ISN which is, intuitively spoken, the largest ISN which is smaller than the SN. Using the index of the just identified subflow, the counter in the second table increases.

c) Table Organization and Subflow Lifetime: Currently, we only consider growing ISNs as specified in the original

TCP standard [6]. However, the concept can be extended to support random ISNs by shifting values as needed (cf. section IV). To solve the counter mapping problem while lacking sorting capabilities, we organize all captured ISNs as well as the subflow counters in stacks. A newly captured ISN is always placed at the end of the stack, as ISNs grow with time, assuring an ascending order in the stack. The beginning of a subflow can be detected by the presence of the SYN flag in TCP. Therefore, whenever a SYN flag is detected, a new ISN is registered. Similarly, the FIN flag announces its end. The detection of a FIN packet triggers the deletion of the corresponding subflow ISN and packet counter, making place for new measurements as depicted in Figure 2c. To preserve the stack nature, the ISNs and subflow counters are shifted accordingly when a subflow expires. Thus, our system always provides a live overview of passing flows. To avoid missing valuable information, it is therefore necessary to actively poll flow statistics periodically in the control plane.

Using only P4 features, we did not have access to complex data types as dynamic lists, stacks, and matrices. As counter tables are primordial for our task, we opted for building them as a series of arrays. P4 requires knowledge of the counter table dimensions beforehand. The table length is given by the number of flows in the switch's flow rule set, its configurable width is the maximal number of detectable subflows that can be present in an aggregated flow at the same time. To facilitate the adaptation of the table width, we also provide a code generator that adapts the data plane P4 program to the number of supported subflows of choice.

III. Evaluation

In this section, we evaluate the proposed approach with respect to accuracy and recall with different settings. To this end, the next subsection describes our evaluation environment and the prototype implementation followed by the results and their analysis.

A. Environment

The prototypical implementation of the presented approach uses the behavioral model BMv2 as P4 switches, which talk to a Python-based controller via P4Runtime in an emulated Mininet network.

The network is kept minimalistic to avoid unwanted side-effects, so that there is a single sender and a single receiver for all flows. Flows are dynamically generated based on a configuration file with exponentially distributed inter arrival times for packets as well as for flows ($\lambda = 1$). If not stated differently, a test runs comprises 33 flows, each with one to four subflows and a total of 100 subflows.The results rely on 30 runs each which each lastet between eight to ten minutes.

B. Results

We first demonstrate how the approach is able to distinguish between subflows in a synthetical setup with a single flow consisting of multiple subflows. On top of this, we highlight the accuracy of the proposed method when targeting application-level flow statistics within different configurations.

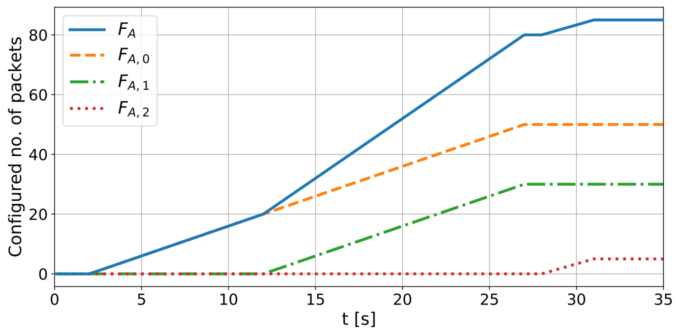


Fig. 3: Accumulated packets dispatched from the sender over time: Flow F_A consists of three subflows $F_{A,0}$, $F_{A,1}$, and $F_{A,2}$.

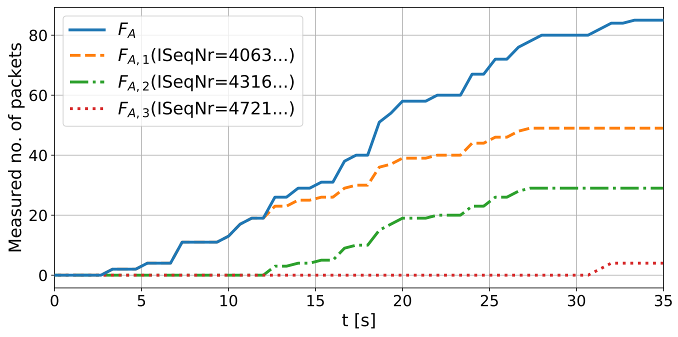
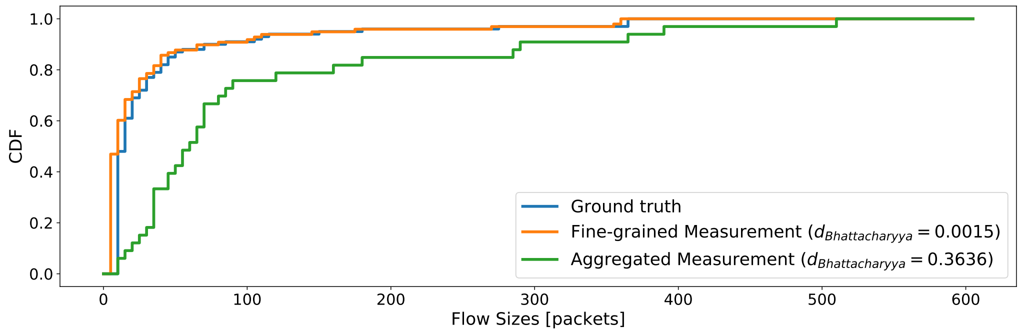


Fig. 4: Accumulated packets visible in the switch over time compared to Figure 3. The switch correctly detects three flows and distinguishes their traffic.

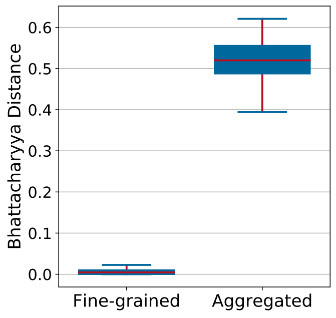
a) Demonstration of the Subflow Detection: Figure 3 shows the generated traffic at the sender: One flow, denoted F_A (solid blue), consists of three smaller flows, namely $F_{A,0}$ (dashed orange), $F_{A,1}$ (dash-dotted green), and $F_{A,2}$ (dotted red). At every point in time t, $F_A(t)$ sums up all packets contained in all three subflows: $F_A(t) = \sum_i F_{A,i}(t)$.

Now picking up these rates, in Figure 4, we observe the flows the switch measures. Although one only suspects the connection between the total traffic (solid blue, F_A) of both figures, the connection of the subflows is clearly visible. The first subflow $F_{A,0}$ starts and rises similar with the same total number of packets of 50 at the very same timestamp of approximately 27 seconds. This holds also true for $F_{A,1}$ and $F_{A,2}$. The switch identifies the subflows based on the ISNs as indicated and assigns subsequent packets based on their SN as described in Section II. All ISNs are successfully identified and the packets are correctly mapped to their corresponding subflows.

b) Fine-Grained Measurement Accuracy: Next, we showcase the accuracy with respect to total flow sizes in terms of measured packets per flow. We compare the measured flow sizes using our mechanism and aggregated naive measurements with the flow sizes of actually dispatched flows. Figure 5a shows an exemplary CDF of flow sizes for one



(a) Flow size distribution of an exemplary evaluation run.



(b) Statistical results.

Fig. 5: Accuracy comparison using the measured flow size distribution of the proposed fine-grained measurement approach with aggregated measurements. The actually dispatched application-level flows are used as ground truth.

simulation run.

The solid blue curve represents the actually dispatched subflows from the sender (“Ground truth”). A typical pattern of a large number of small *mice* flows is observable with a long tail, thus, a small fraction of larger *elephant* flows. The dotted green curve shows the naive measurement approach (“Aggregated Measurement”), where the installed rules define the granularity. It is clearly visible that a smaller flow sizes occur less often and the curve is generally shifted to the right, so towards longer flow sizes. The obvious reason for this is the aggregation of multiple subflows into a flow (rule). Nevertheless, taking the dashed orange line that shows the fine-grained measurements based on our measurement approach into consideration (“Fine-grained Measurement”), we see that flow size CDF is almost identical to the ground truth. Thus, the approach correctly detects all subflows, is able to match packets to subflows and therefore also the flows' total sizes. Taking a closer look on the curves reveals that the measured flow sizes differ by a tiny gap. By the fact that it is necessary to periodically ask for currently existing subflows, the last few packets traversing the switch after the last statistic request just before the subflow ended (indicated with a TCP FIN flag) can be missed. Potential workarounds and improvements are left for future work and the trade-off between slightly higher fidelity and required overhead must be considered.

Figure 5b shows the comparison between the ground truth and the measurements, both, fine-grained and aggregated. As metric to measure the distance between the ground truth CDF and the measurement CDF we use the Bhattacharyya distance [7] which is a qualified measure for the similarity of distributions. The smaller the Bhattacharyya distance, the closer the distributions. The statistical results shown in the figure confirm the observations of the exemplary run in Figure 5a. Fine-grained measurements provide flow sizes close to the actually sent sizes whereas the aggregated measurements differ significantly due to non-distinctive statistics.

c) Influence of the Measurement Rate: In this paragraph, we further discuss the accuracy of the approach while changing the statistic request rate, which has an important influence.

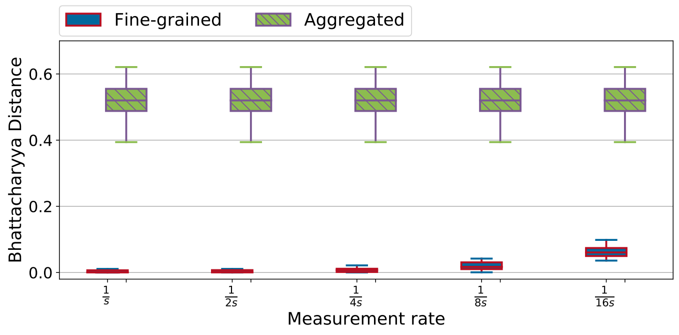


Fig. 6: Accuracy in terms of Bhattacharyya distance between ground truth and measurements for the FSD applying the proposed fine-grained measurement approach and aggregated measurements with different measurement rates.

As already pointed out in the previous paragraph, the controller cannot capture packets of a subflow arriving between a statistic request and the subflow's FIN packet. This is not the case for the aggregated measurements where subflows are included entirely as part of the superordinated flow. Figure 6 depicts the accuracy for both cases when increasing the rate of statistic requests, thus, the time in which packets of small flows can potentially be missed. We observe that the aggregated measurements are independent of the measurement rate as the flow sizes are always fully covered – of course with the disadvantage of being aggregated in a single statistic. Therefore, their accuracy in terms of Bhattacharyya distance is comparably poor. In contrast, the fine-grained measurement method we propose shows very high fidelity with moderate measurement rates of once per second (distance is close to 0). Higher rates would deliver even better accuracy and are therefore not depicted. Only when decreasing the measurement rate, e.g., to a statistic request just every 8 or 16 seconds, we observe that the accuracy suffers from missing packets of ending subflows between the measurements. Note that the numbers strongly depend on the fluctuation of flow in the respective context. However, even with very low rates the

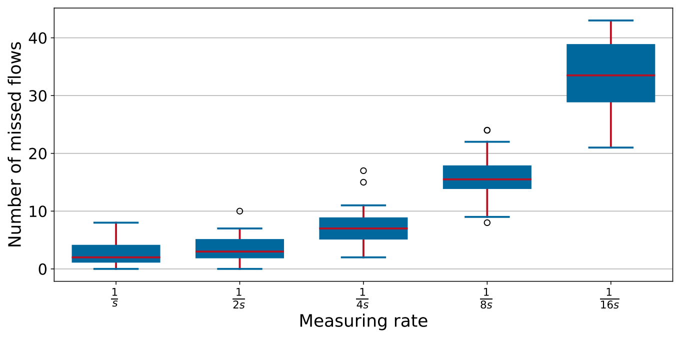


Fig. 7: The number of undetected/missed subflows with different measurement interval lengths (measurement rate).

accuracy is considerably better such that we find the accuracy loss negligible when targeting application-level statistics.

Figure 7 supports this observation further: The figure shows the number of missed subflows (y-axis) for different measurement rates. For the rate of $f = \frac{1}{s}$, so once per second, about two subflows are missed in median (see Section III-A for flow generation details). As each box represents the number of missed subflows for a total of 100 subflows, the total number equals the percentage. Halving the measurement rates does not increase the number of undetected subflows significantly. Further decreasing the measurement rate also increases the number of missed flows. For small measurement rates of $f = \frac{1}{16s}$, 34 subflows are not being detected in median, meaning that the lifetime of 34 subflows were smaller than the measurement of 16 seconds. Note again that the actual number strongly depends on the fluctuation of flows.

C. Evaluation Take-Aways

Our evaluations have shown that the proposed method is able to detect TCP subflows as part of one flow rule and collect their statistics appropriately. We underpin expected theoretic measurement results with experiments using a prototypical implementation. Furthermore, using the exemplary Flow Size Distribution use case, which immediately reflects the subflow statistics, the experiments show a huge accuracy gain. On top of this, we highlight that (and to what extend) very short subflows might be missed in the approach depending on the measurement rate.

IV. Limitations

The developed approach faces some limitations that must be considered prior to its use:

Ascending ISNs: The mapping algorithm presented in Section II requires sorted ISNs to be efficiently solvable in the data plane. With the growing ISN assumption, this condition is fulfilled. However, when random ISNs are used, the controller should intervene to sort a specific flow's stored ISNs in the P4 measuring point. The remainder of the concept remains unchanged. Also, overlapping SN ranges must be considered as they are likely to occur in a real world setting. However, this issue can be solved easily by storing the latest sequence number of each flow instead of the ISN.

Flow trail/Mice flow detection: In its current design, the approach immediately de-allocates memory used for expired subflows. This means that a subflow is not anymore included in any statistic report after its expiration and, therefore, a flows last packets, i.e. every packet arriving between the last statistic request and the FIN packet, are not factored in in any counter and not detected by the applications. Very short flows, so-called mice flows, could even occur between two subsequent statistic requests, meaning they are not detected by the application at all. A number of workarounds are thinkable and left for future work.

TCP flag dependency: The data plane identifies subflows and their packets based on TCP header information. As a logical consequence, any other protocol is not supported and must, if possible, be included manually or monitored on the default per-flow level. However, TCP predominates the Internet's traffic.

TCP FIN flag dependency: In our design, we require the detection of FIN flags to expire flows and free up memory. In the case of a SYN attack, where the network is flooded with flows containing only one SYN packet, the counter matrix will quickly be overwhelmed by non persistent counters holding a value of 1. On the one hand, this hinders the measurement of traffic of interest. On the other, the counter matrix filled with ones, is a good indication of SYN attacks. Using this information, the detection of such attacks can take place on the data plane.

Memory consumption: As already mentioned in Section II, the approach allocates a static number of registers to store counter (two 64-bit registers for each possible subflow one might measure). However, the system allows to configure the allowed maximal number of measurable subflows per flow and space restrictions are rather relaxed as register are not stored in TCAM but SRAM or DRAM. With 10.000 rules and 5 subflows per rule, space of 800KByte is required, which is acceptable for nowadays SRAM/DRAM switch capacity [5].

V. Related Work

So far, existing works in this context cover the aggregation granularity problem of monitoring through counter-based statistics predominantly by splitting and merging the spatial flow room space, in other words by *zooming into rules*. Popular approaches for this include the adaptive measurement framework DREAM, follwed up by SCREAM, presented by Moshref et al. [8], [9] or a related approach by Jose et al. [10]. Both allow to select prefixes of choice and increase the granularity within these flows by splitting flow rules. Furthermore, Zhang [11] presents an OpenWatch, a flow counting method that is adaptive in the spatial dimension but also the temporal dimension by adjusting the statistic sampling rate. Those approaches all require decoupling of monitoring from routing, hence, to modify the flow tables. With an eye on the very limited TCAM memory due to its excessive costs and power consumption that is used to maintain flow rules, there is a twofold problem: Freely splitting rules occupies valuable resources and therefore reduces the amount of available space for routing or other applications. Also, introducing rules solely used for monitoring appears to be inefficient such that we target an alternative solution in this paper. Orthogonal approaches, e.g. from Afek et al. [12], flexibly sample packets within the streams. Theoretically, this allows to infer approximate flow statistics on application-level granularity. However, such approaches have a high chance of missing mice flows as given in the introducing example of DDoS attacks via TCP SYN floods and anyway lack of certainty of measurement accuracy.

While various approaches to enhance monitoring in Software-defined Networks under the use of programmability of the data plane exist [13]–[16], none of them particularly targets the aggregation problem tackled in this work. The closest approaches we identified are ELASTIC SKETCH by Yang et al. [17] and UMON by Wang et al [18]. The former uses additional memory to store sketches of network statistics which can be dynamically adapted with respect to their size. The authors implement the approach, among other platforms, on a P4 switch. The latter, UMON uses additional sub flow tables within an OpenFlow-based environment to monitor with higher granularity than rules allow. The approach is implemented for Open vSwitches, limiting its applicability, and it consumes expensive TCAM memory.

VI. CONCLUSION AND OUTLOOK

In this work, we tackled the problem that flow rules usually aggregate multiple application flows and, as a consequence, limit today's counter-based statistic granularity. To this end, we propose a mechanism for P4 switches to identify subflows handled by a single flow rule and collect their individual statistics. We do so by parsing the packets TCP header fields: SYN packets declare a new subflow, the sequence number reveals the subflow a packets belongs to, and lastly the FIN packets signals the end of the packet stream. We organize a set of registers to hold multiple subflow identifiers per rule, i.e., the initial sequence number and counter. The approach implements an effective fragmentation of the allocated memory and allows to configure the number of allowed subflows per flow rule to respect potential memory restrictions. Our evaluation reveals that the mechanism works as desired and is able to increase the accuracy of flow-size-consuming statistics tremendously.

Based on the identified limitations in Section IV, we intend to further enhance the design: Because of the instant de-allocation of memory on flow expiration, packets at the end of a flow might not be included in the statistics, which can be avoided using different methods. An obvious yet effective solution is to introduce subflow counter timeouts as they exist for regular flow rules. Due to the lack of P4 support, a potential alternative is to hold subflow counter until the next statistic report is dispatched. Another option to improve the usability of the mechanism is to provide more profound information on the subflows such as for instance used protocols, byte counter in addition to packets counter etc. However, memory restrictions might apply. Since memory usage might be a restricting factor in some situations, the use of sketches has been proven very handy [8], [9], [17]. For example, count-min sketches can be easily implemented in P4 and relieve the memory constraints. Eventually, an implementation on a real device is left for future work to identify potential threads wrt. to its implementability.

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," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[3] R. Hark, M. Ghanmi, S. Kar, N. Richerzhagen, A. Rizk, and R. Steinmetz, "Representative measurement point selection to monitor software-defined networks," in *IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2018, pp. 511–518.

[4] R. Kundel, J. Wallerich, W. Maas, L. Nobach, B. Koldehofe, and R. Steinmetz, "Queueing at the telco service edge: Requirements, challenges and opportunities," in *Workshop on Buffer Sizing*, 2019, pp. 1–6.

[5] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proceedings of the ACM SIGCOMM Workshop on Information-centric networking*, 2011, pp. 44–49.

[6] IETF RFC 793, "Transmission control protocol procotol specification," 1981. [Online]. Available: https://tools.ietf.org/html/rfc793

[7] A. Bhattacharyya, "On a measure of divergence between two multinomial populations," *Sankhyā: The Indian Journal of Statistics*, pp. 401–406, 1946.

[8] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: dynamic resource allocation for software-defined measurement," in *ACM conference on SIGCOMM*, 2014, pp. 419–430.

[9] ——, "Scream: Sketch resource allocation for software-defined measurement," in *ACM Conference on Emerging Networking Experiments and Technologies*, 2015, pp. 1–13.

[10] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches." in *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2011.

[11] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *ACM conference on Emerging networking experiments and technologies*, 2013, pp. 25–30.

[12] Y. Afek, A. Bremler-Barr, S. Landau Feibish, and L. Schiff, "Sampling and large flow detection in sdn," in *ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 345–346.

[13] J. Geng, J. Yan, Y. Ren, and Y. Zhang, "Design and implementation of network monitoring and scheduling architecture based on p4," in *International Conference on Computer Science and Application Engineering*, 2018, pp. 1–6.

[14] L. Castanheira, R. Parizotto, and A. E. Schaeffer-Filho, "Flowstalker: Comprehensive traffic flow monitoring on the data plane using p4," in *ICC IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[15] B. Guan and S.-H. Shen, "Flowspy: An efficient network monitoring framework using p4 in software-defined networks," in *IEEE Vehicular Technology Conference (VTC2019-Fall)*. IEEE, 2019, pp. 1–5.

[16] R. Hark, D. Bhat, M. Zink, R. Steinmetz, and A. Rizk, "Preprocessing monitoring information on the sdn data-plane using p4," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2019, pp. 1–6.

[17] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 561–575.

[18] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Umon: Flexible and fine grained traffic monitoring in open vswitch," in *ACM Conference on Emerging Networking Experiments and Technologies*, 2015, pp. 1–7.