An Adaptive Flow Counting Method for Anomaly Detection in SDN

Ying Zhang
Ericsson Research
200 Holger Way
San Jose, California
ying.zhang@ericsson.com

ABSTRACT

The accuracy and granularity of network flow measurement play a critical role in many network management tasks, especially for anomaly detection. Despite its important, traffic monitoring often introduces overhead to the network, thus, operators have to employ sampling and aggregation to avoid overloading the infrastructure. However, such sampled and aggregated information may affect the accuracy of traffic anomaly detection. In this work, we propose a novel method that performs adaptive zooming in the aggregation of flows to be measured. In order to better balance the monitoring overhead and the anomaly detection accuracy, we propose a prediction based algorithm that dynamically change the granularity of measurement along both the spatial and the temporal dimensions. To control the load on each individual switch, we carefully delegate monitoring rules in the network wide. Using real-world data and three simple anomaly detectors, we show that the adaptive based counting can detect anomalies more accurately with less overhead.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network monitoring, Network management

Keywords

Network measurement; Software-defined networking

1. INTRODUCTION

Network flow counting is essential to many network management applications, ranging from network planning, routing optimization, customer accounting and anomaly detection. Today, statistics of traffic flows are reported by the routers to the centralized management system. Thus, the impact of network measurement on the network must be minimized. For example, an aggressive monitoring may result in artificial bottlenecks in the network. With too passive a scheme, it may miss important events. Thus, the key challenge is to strike a careful balance between effectiveness (supporting a wide range of applications with accurate statistics) and efficiency (intruding low overhead and cost). Among all the network management applications, in this paper, we seek answers to the question: how to count flows to provide sufficient information for the network anomaly detection?

Existing attempts to achieve a better overhead/accuracy balance is through traffic sampling [1, 2], *i.e.,* a router selectively records packets or flows randomly with a pre-configured sampling rate. The thinned traffic is then fed as input to anomaly detection. While being widely deployed as they are simple to implement with low CPU power and memory requirements, studies have shown it to be inaccurate, as it is likely to miss small flows entirely [3]. Previous studies have pointed out that packet sampling is inadequate to detect anomalies such as portscan [4]. While there are enhancements on sampling strategies [5], the solutions are generally difficult to be implemented in commodity switches.

In this paper, we tackle the problem of balancing the overhead and accuracy from a different approach. Our key idea is that the network flow measurement should provide a more *flexible* and more *interactive* interface to the anomaly detectors so that the set of flows collectted can be dynamically adjusted according to the findings of anomaly detector immediately. We propose that there are two-fold benefits of such interfaces. On the one hand, the anomaly detector can instruct the flow collection module to provide finer-granularity data once there is a suspicion of attacks, so that the anomaly can be identified sooner. On the other hand, it can inform to collect coarser-grained flow data both spatially and temporally when there is no sign of attacks such that the traffic monitoring load is reduced. Therefore, this adaptive interface can simultaneously improve the accuracy and reduce the overhead .

Although sounded promising, this method is difficult to be implemented in today's network as changing the flow collector requires manual or semi-manual configurations on the routers. The granularity of flow collection is also limited by the configuration options that vendors provide. However, we find that the problems of inflexible control plane and the vendor lock-in can be solved by the recently proposed Software-defined Networking (SDN) paradigm. SDN introduces programmability, centralized intelligence and abstractions from the underlying network infrastructure [6]. In SDN, the data plane actions are controlled a programmable, logically centralized control plane, which can easily interact with other network management systems.

We believe that SDN has two key features that enable the design of a flexible flow counting API for anomaly detection. On the one hand, SDN breaks the tight bindings between forwarding and counting. One can install separate wildcard rules on OpenFlow (OF) switches purely for monitoring purposes, which offers tremendous flexibility in defining the set of packets to count. On the other hand, thanks to the simple interface between control and forwarding plane, one can easily adjust the elements to count, by

CoNEXT'13, December 9Ð12, 2013, Santa Barbara, California, USA.

http://dx.doi.org/10.1145/2535372.2535411.

simply updating the counting rules. This property makes *realtime* adaptive counting possible.

While there have been many efforts on designing the interfaces between control and forwarding plane [7, 8], little thought has been put into the security applications in SDN. Two notable studies related to traffic measurements are [9] and [10]. In [11]. While the former proposes a sketch based method for general measurement tasks like heavy hitters detection, the trade-offs of load and accuracy are discussed in the latter. Differently, this paper studies the *network-wide* flow measurement for the purpose of flexibly and accurately detecting anomalies.

In this paper, we propose an adaptive flow collection method for anomaly detection in SDN. Benefiting from the nature of SDN, it is programmable, allowing flexible specification of the spatial and temporal properties of counting units (or aggregates). It is also designed to introduce low storage, computational, and bandwidth overhead. We implement the method on a prototype called OpenWatch. The main contributions are summarized below.

- We propose an adaptive algorithm to control the temporal and spatial aggregation of the counting function, which is based on a linear prediction method. It efficiently reduces the amount of processed data and lessens the overhead on the network. The linear predictor uses previous samples to estimate or predict a future measurement granularity.
- Taking an advantage of the centralized view of the network, we develop an algorithm to intelligently allocate flow counting tasks among multiple switches to reduce the overhead on a single switch.
- Finally, we evaluate the overhead reduction and accuracy improvement of OpenWatch by comparing with static aggregation approaches.

The rest of this paper is organized as follows. §2 presents an overview of the OpenWatch architecture followed by more detail the principal components of this architecture. In §3 we present evaluation results of OpenWatch on a few anomaly detection examples. Finally, we summarize the related work §4 and conclude.

2. METHODOLOGY

Since most SDN architectures employ low-end commodity switches [6], we cannot assume complex functionalities on the OF switches. Thus, our measurement framework is designed only to tries to rely on the simple match-and-count rules installed and adjusted by the controller. The switches can send traffic counters to the controller periodically. However, the granularity choices at both the temporal and spatial domain are critical. If the statistics are collected in a large period, e.g. half an hour, then the anomaly detection algorithms may not be able to pick out the short-lived anomalies. On the other hand, if the collection is done every few seconds, it can generate a lot of traffic to the network, and a large amount of load to the controller. Therefore, finding a solution that can provide accurate detection without imposing too much load to the network is important.

2.1 Overview

OpenWatch gradually and selectively instructs the switches to collect statistics in the entire flow space. It starts with collecting coarse-grained statistics on a large address block, and comparing it with historical data. Once suspicious deviation is identified, it iteratively adjusts the wildcard rules and produces finer-grained information to the anomaly detection applications. This approach certainly may lead to a short delay in detecting the traffic anomalies. But on the other hand, this trade-offs can be configurable and tunable, depending on the operatorsÕ needs.

It contains three components. The first and the most important component provides the adaptive zooming in the flow space and determines the set of rules to be installed. After that, the second component determines where the rules to be installed in the network-wide in order to minimize the space taken on the switch flow tables. Finally, the third component prepares the data for anomaly detection applications which will use these data. It anecdotes the data with the corresponding spatial/temporal aggregation information.

Algorithm 1 Dynamic rule update algorithm

procedure *Rule_Adaptation(G,A,T,d)*
  for each aggregate $a \in A$ **do**
    Insert a to TRIE
    $a.time = T$, $a.count = 0$
  while reports to process **do**
    for each report counter for $a \in A$ **do**
      compute v_p^a and $R(v_p^a)$
      if $R(v_p^a) < R_L^a$ **then**
        $a.flag = EXPAND$
      else
        if $R(v_p^a) < R_U^a$ **then**
          $a.flag = CONTRACT$
      if a is a smaller group of $a' \in A$ **then**
        if $R(v_p^a) < R_L^{a'}$ **then**
          $a'.flag = EXPAND$
        if $R(v_p^a) < R_U^{a'}$ **then**
          $a.flag = CONTRACT$
    for every node a in TRIE **do**
      if $a.flag = EXPAND$ **then**
        decrease reporting interval to $\frac{T}{d}$
        expand a to smaller groups a/d
        assign a/D to switch setS^a
      if $(a.flag = CONTRACT) and (siblings(a).flag = CONTRACT)$ **then**
        remove rule for a and $siblings(a)$
        $parent(a).time = d \times T$

2.2 Dynamic rule expansion and contraction

2.2.1 Initial set of rules

Maintaining a counter per flow is very expensive, we need spatial aggregation to reduce the number of rules. For example, instead of having a rule for each individual IP address, we can have a rule for each /24 prefix. However, large aggregation may blur the anomalies that are only visible to a subset of the flows. Our idea is to *monitor the large traffic aggregates, sample a few small aggregates, and adapt the rules dynamically*.

More specifically, we propose a header space divider D to be applied to the entire flow space. For example, the aggregates can be based on destination address or source address. It can also be based on different port numbers, reflecting different applications. The operators can define and change the divider flexibly. To overcome the problem of larger aggregates covering changes in smaller groups within this aggregate, we also perform random sampling of smaller groups with the rate r. The smaller r is, the fewer loads it introduces. In summary, it first performs spatial aggregation using the pre-defined divider D. For each large group a, its sub-group a' is sampled randomly with probability r. In the experiment, we show

that even a very small r is helpful. Each large group together with the selected sub-groups are inserted into the set to be monitored, A. The rules are pulled once every T time window.

2.2.2 Adaptive mechanism

Identifying significant traffic changes is a key building block for most anomaly detection applications. We propose that based on the normal traffic pattern, OpenWatch can install a few more rules to monitor different groups of traffic, and/or change the duration each report is aggregated. The adaptive control contains two aspects.

- *Spatial adjustment* controls the aggregation over the space in the packet header (D). It could be finer-granularity rules as a zoom-in of existing rules, or courser-grained rules as a zoom-out. For the former case, OpenWatch can expand the rules to ultimately find the specific aggregates responsible for the traffic change. For the latter case, it happens when there is no sign of anomalies so smaller groups can be combined to a larger group in order to save resources.

- *Temporal adjustment* controls the aggregation across time (T). Initially, each switch reports the counters every t seconds. When there is a sign of anomaly, OpenWatch wants to react fast to detect it. Thus, t can be reduced, generating records with a shorter duration. Similarly, if there is no sign of anomaly, the reporting duration can be set back to normal.

Algorithm 1 shows the steps for adaptive control. Upon receiving a new set of counters, OpenWatch compares the latest records with the history, and triggers the action of expansion or contraction if needed. For efficiency, we maintain a trie structure for each aggregate under monitored. The key of each node is the spatial aggregation property (e.g. destination prefix or port range). Each node has two attributes, the counter and the reporting period. If it is to be expanded, a set of new nodes will be inserted under this node, and a new set of switches will be assigned to count them. On contraction, the children nodes will be removed and the counters will be updated for their parent node.

It is challenging of making the decision for expanding and contracting. We propose a linear prediction method below.

2.2.3 Linear Prediction

The Linear Prediction method uses previous samples to estimate or predict a future measurement. It attempts to predict the value of the next sample. If the value falls into the range of the prediction, it indicates the possibility of contraction. Otherwise, the deviation between the predicted and observed values indicates a change in the network behavior and require an expansion and more frequent counting to determine the new pattern.

Specifically, we maintain a list of n records for each aggregate. The predicted value of aggregate a in time t_n is in Eq 1.

$$v_p^a = v_n^a + \frac{(t_n^a - t_{n-1}^a)}{n-1} \sum_{i=1}^{n-1} \frac{v_{i+1}^a - v_i^a}{t_{i+1}^a - t_i^a} \quad (1)$$

The predicted output v_p^a is then compared with the actual value of v_{n+1}^a using the ratio metric:

$$R(v_p^a) = |\frac{v_p^a - v_n^a}{v_{n+1}^a - v_n^a}| \quad (2)$$

Similarly, we define the acceptable range for $R(v_p^a)$: $R_L^a = R(mean_a - 3 \times std_a)$ and $R_U^a = R(mean_a + 3 \times std_a)$, where $mean_a$ and std_a are the average and standard deviation of aggregate a computed using EWMA.

Algorithm 2 Rule placement algorithm

procedure *Rule_Placement(G,A,N,r)*
  for each aggregate $a \in A$ **do**
    Gather all switches in G that can cover a, denoted as S^a
    Increase flow covering count C_s of each switch s.
  sort aggregates according to $|S^a|$ ascending to SA
  for every aggregate $a \in SA$ **do**
    assign a to switch s with smallest C_s value
    N_s++;
    if $N_s > N$ **then**
      remove s from candidate set

If $R(v_p^a)$ is below R_L, the measured parameter is changing faster than the prediction. Such behavior indicates more activity than predicted, so the counting granularity should be decreased to yield more accurate data values on which to base future predictions.

Conversely, if $R(v_p^a)$ is above R_U, the measured value is changing more slowly than the prediction so the counting granularity can be increased. Similarly, we also compare the predicted value of the sampled sub-group with its covering group. The condition for contraction is that if all the children of a node are marked as *CONTRACT*, then they are collapsed into one single node.

2.3 Rule placement

Once the set of flow aggregates to be monitored is determined, we need to determine **who** (which switch) **counts what** (which flow aggregates). While [12] proposes a rule optimization method for each switch, we focus on a network-wide optimization. In today's approach, each router *independently* counts packets or flows, which results in large duplications of flows collected. We argue that the fundamental limitation is the distributed router centric view in this type of measurement architecture. With the logically centralized view of SDN, we have the opportunity to reduce such redundancy.

In SDN flow counting, a simple way is to let each ingress switch record the counters for all the flows coming into the network. However, this may overload the ingress switches. Therefore, we design a simple yet efficient algorithm to *offload the monitoring loads from the ingress switches to other switches along the paths*. Since the controller has the complete network topology and routing path for each flow, it then computes the optimal balanced task assignment for all the switches in the network.

The problem can be described as follows. Given a network G with V set of OF switches and a set of flows to be counted F, we need to find the best mapping from F to V under the following constraints. The number of assigned rules in $v \in V$, should not exceeds the number of table entries available for the monitoring tasks on v. The set of constraints of each switch, including the number of table entries and the bandwidth capacity limits, are denoted as N.

Algorithm 2 shows the details of the algorithm. We map each element to be placed in A to the set of switches that can monitor it. The set for aggregate a is denoted as S^a, called *the covering set for* a. For each switch s, we keep a counter for the number of rules it can monitor, C_s. We iteratively assign aggregates to switches, following two rules. Rule 1: we always select the aggregate with smallest covering set, *i.e.,* $|S^a|$ is the smallest. It means that we consider the aggregate with most constraints first. Rule 2: for each a, we select the switch with smallest C_s to cover it. It means that we save the switches which can cover more rules for later assignment. On the other hand, for each assignment decision, we will examine the switch capacity is not exceeded. Certainly more complex algorithm can be developed to obtain optimized results, in this

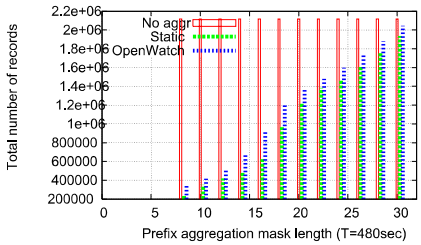

Figure 1: Network load with different spatial aggregations.

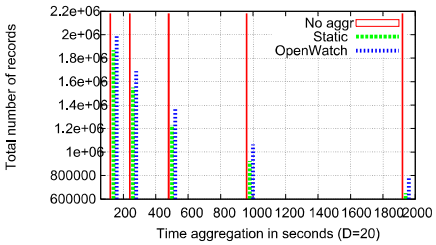

Figure 2: Network load with different temporal aggregations.

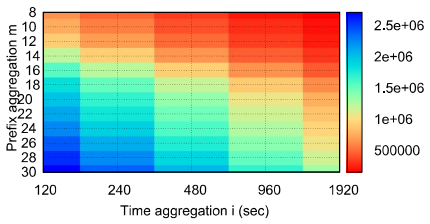

Figure 3: Network load with different temporal and spatial aggregations.

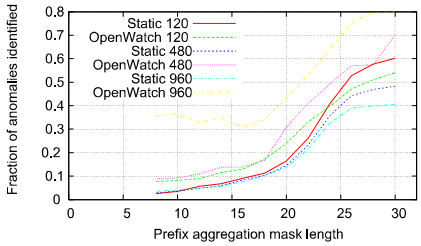

Figure 4: Found anomalies with Volume based detector.

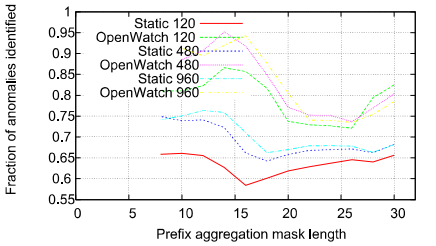

Figure 5: Found anomalies with Huge jump detector.

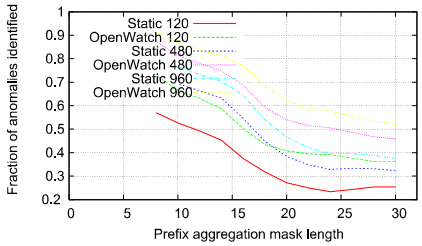

Figure 6: Found anomalies with Wavelet detector.

paper we take the first attempt and achieve already good performance. Our future work is to find optimized solutions, evaluate computational complexity trade-offs and integrate with per-switch optimization.

3. EVALUATION RESULTS

In order to gauge the performance of the adaptive coating method, we compare it with static aggregation, which has a fixed reporting interval and fixed grouping of flows. We evaluate it using an unsampled packet trace collected in 2011 in a European country. It is collected in a mobile network on a Gn interface between a Gateway GPRS Support Node (GGSN) and a Serving GPRS Support Node (SGSN) in a cellular network. It contains 168 hours of data with 3.9 million sessions from 50k IP addresses.

We build a simulator to evaluate different monitoring strategies. We evaluate OpenWatch from two aspect: the overhead and the accuracy for anomaly detection. For spatial aggregation, we use prefix with different mask length for both source and destination addresses. In the future, we plan to evaluate other aggregation methods.

3.1 Overhead

Figure 1 shows the total number of reports as the prefix mask length varies from 8 to 30. For example, 8 means that IP address 1.2.3.4 is counted as 1.0.0.0/8. We compare OpenWatch with the *No aggr* method (counting for each IP address) and *Static* (aggregating using the fixed mask length during the entire counting period). We can clearly observe that both OpenWatch and *Static* reduces the load drastically. Comparing with *Static*, OpenWatch only introduces a slight extra load (less than 10% in all cases), caused by the dynamic expansion. Similarly, Figure 2 shows the growth of total records as the initial reporting interval increases. OpenWatch generates at most 30% more records than *Static*, but still remains 30% to 90% of the *No aggr* method.

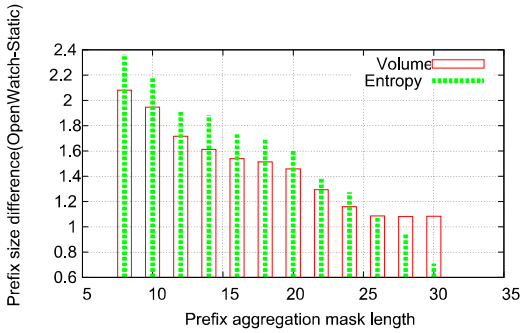

Figure 7: Prefix size difference.

Finally, we evaluate how OpenWatch performs under different configurations of prefix length and time interval in Figure 3. The color represents the number of total records. Clearly smaller mask length and larger interval lead to fewest records. But comparing with the same figure for *Static*, the changes of colors are less drastically, meaning that the impact of initial configuration is less pronounced.

3.2 Accuracy

We examine the accuracy using two simple anomaly detection heuristics and wavelet detector [13], serving as examples. In the future, we plan to evaluate on a more complete set of anomaly detectors.

- Volume based detector: detects the high volume prefixes and periods based on the traffic volume. We compute the average and the standard deviation of all records for each prefix, and output an alarm if any record is above $mean + 3 \times std$.

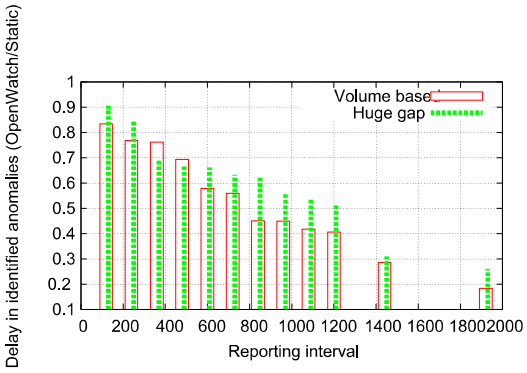

Figure 8: Delay in detection.

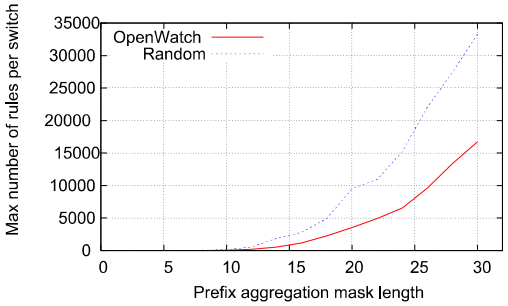

Figure 9: Maximum number of rules per switch.

- Huge jump detector: detects the large jump in traffic volume between consecutive records. We compute the average and standard deviation of increasing ratio between any pairs of consecutive records for all prefixes. An alarm is raised if any gap is above $mean + 3 \times std$.
- Wavelet detector: we apply *DMeyer (dmey in Matlab)* wavelet for transformation and the deviation score method proposed in [13] for detection.

For each method, we compute the set of anomalies using inputs from *No aggr* as a base line. We then compute the set of anomalies using inputs of *Static*, and OpenWatch separately, and compare the fraction of anomalies in *No aggr* that are also reported in the latter two methods.

Figure 4 shows the fraction of detected anomalies using *Static* and OpenWatch, with different reporting intervals. For example, the curve of "Static, 120" means the fraction of anomalies identified using *Static* method with a reporting interval of 120 seconds. Figure 5 shows the results for Huge jump detector. In both figures we can see OpenWatch consistently identifies more anomalies than *Static*. It performs even better under the Huge jump detector. It is also less sensitive to the choice of mask length. For Volume based detector, when the interval is small, the difference is negligible or even worse for OpenWatch. This is because with a small interval, there is little room for adaptation.

Similarly, Figure 6 plots the hits under wavelet detector. The result is less sensitive to the choice of prefix length nor time interval. Still, consistently, OpenWatch performs better, but the different is less pronounced compared to the previous two detectors.

For the set of commonly identified anomalies, we further evaluate how fast the alarm is raised and how precise the aggregate is. Both metrics affect the usefulness of anomaly detection results. For example, reporting an alarm for the entire 1.0.0.0/8 is less useful than reporting the more specific prefix 1.2.3.0/24, if the true anomaly is caused by 1.2.3.0/24. Thus, we compute the difference of prefix size between *Static* and OpenWatch for anomalies raised at the same time, using the latter minus the former. Figure 7 shows that, the mask lengths of OpenWatch are 0.9-2.3 larger than *Static*, meaning that they are more specific.

We evaluate it as the time differences of the commonly identified alarms between OpenWatch and *Static*. Figure 8 shows that OpenWatch takes around 20% to 80% of the time that *Static* needs to detect the anomaly. The advantage is larger with large time intervals.

3.3 Evaluation of rule placement

Finally, we present our preliminary evaluation of the rule placement algorithm in terms of resource saving. We evaluate the rule placement algorithm using Internet2's network topology [14] and the traffic from the packet traces mentioned above. We first divide the source addresses to 11 groups, assigning each group to one PoP as the ingress point of this group of sources. Similarly, we divide all the destination addresses to 11 groups and assign each group to a PoP as their egress points. Then we compute the shortest path between any pairs of ingress and egress PoPs. Figure 9 shows the maximum number of rules per switch, comparing with the random scheme, *i.e.,* each rule is randomly assigned to one switch. Overall, the maximum number of table entries in OpenWatch is around 33% of the random scheme, showing the even spreading of rules across the network.

4. RELATED WORK

Our work contributes to and draws inspiration from a large set of existing work in traffic monitoring, as well as anomaly detection. In particular, ProgME [15] provides a platform to measure any arbitrary set of flows. We were also inspired by the reconfiguration approach in [16]. But they are difficult to be implemented on commodity hardware. Our work is also related to the proposals of spreading the load of collecting netflow records [17, 18]. We solve similar problem in the SDN context.

On the other hand, various anomaly detection methods have been proposed [19, 13, 20], most of which use Netflow. Our work is also related to the previous studies on how sampling impacts network monitoring applications. [4] analyzed how packet sampling impacts the port scan detection methods under different sampling schemes [4]. [21] further points out that the impact varies across metrics.

Our work builds on top of the existing proposals of SDN [22, 6, 7, 23, 24]. A large body of work on improving the SDN programmability is orthogonal to our work [25, 12, 26]. The work on SDN closest to ours are [11, 9, 10, 27]. [11] focuses on detecting heavy hitters on a single OF switch, while [9] proposes using sketch to construct a programmable measurement framework. Closely related, [10] also discusses the tradeoffs between accuracy and resources but under the context of heavy hitter detection. Our work tackles on a different problem, *i.e.,* providing inputs to anomaly detectors, and we consider optimization in the network wide.

5. CONCLUSION

To conclude, in this work, we present OpenWatch for measuring flows under the application of anomaly detections. Built on top of SDN, OpenWatch can support dynamic adaption of flow counting for the purpose of anomaly detection. It employs a prediction based dynamic adjustment scheme to provide dynamic zooming

into the flow space in both temporal and spatial dimensions. Using a greedy based rule assignment algorithm, it can reduce the average number of rules assigned to each switch. For future work, we plan to develop more accurate adjustment algorithms and change the anomaly detection algorithms that take the dynamics into account.

6. REFERENCES

[1] T. ZSEBY, "Netflow."

[2] M. WANG, B. LI, and Z. LI, "sflow: Towards resource-efficient and agile service federation in service overlay networks," in *Proc. ACM SIGCOMM*, 2004.

[3] B.-Y. CHOI, J. PARK, and Z.-L. ZHANG, "Adaptive random sampling for total load estimation," in *In IEEE International Conference on Communications*, 2003.

[4] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?," in *Proc. ACM SIGCOMM IMC*, pp. 165–176, 2006.

[5] C. ESTAN, K. KEYS, D. MOORE, and G. VARGH-ESE, "Building a better netflow," in *Proc. ACM SIGCOMM*, 2004.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, March 2008.

[7] "OpenFlow 1.1." `http://www.openflow.org/wk/index.php/OpenFlow_v1.1`.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 254–265, August 2011.

[9] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. Symposium on Networked Systems Design and Implementation*, 2013.

[10] M. Moshref, M. Yu, and R. Govindan, "Resource/Accuracy Tradeoffs in Software-Defined Measurement," in *Proceedings of HotSDN*, August 2013.

[11] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proceedings of HotCloud*, pp. 13–13, 2011.

[12] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," in *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pp. 351–362, 2010.

[13] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, pp. 71–82, 2002.

[14] `http://www.internet2.edu/network/`.

[15] L. Yuan, C.-N. Chuah, and P. Mohapatra, "Progme: towards programmable network measurement," in *Proc. ACM SIGCOMM*, pp. 97–108, 2007.

[16] F. Khan, N. Hosein, C.-N. Chuah, and S. Ghiasi, "Streaming solutions for fine-grained network traffic measurements and analysis.," in *ANCS*, 2011.

[17] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "Csamp: a system for network-wide flow monitoring," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, 2008.

[18] C.-W. Chang, G. Huang, B. Lin, and C.-N. Chuah, "Leisure: A framework for load-balanced network-wide traffic measurement.," in *ANCS*, 2011.

[19] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *Proc. ACM SIGCOMM*, pp. 217–228, 2005.

[20] J. D. Brutlag, "Aberrant behavior detection in time series for network monitoring," in *Proceedings of the 14th USENIX conference on System administration*, LISA '00, pp. 139–146, 2000.

[21] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics," in *Proc. ACM SIGCOMM IMC*, pp. 159–164, 2006.

[22] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *SIGCOMM Comput. Commun. Rev.*, vol. 35, October 2005.

[23] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, 2010.

[24] R. Wang, D. Butnariu, and J. Rexford, "Openflow-based server load balancing gone wild," in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, Hot-ICE'11, 2011.

[25] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: a network programming language," in *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, ICFP '11, 2011.

[26] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: change you can believe in!," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, 2011.

[27] F. Huici, A. di Pietro, B. Trammell, J. M. Gomez Hidalgo, D. Martinez Ruiz, and N. d'Heureuse, "Blockmon: a high-performance composable network traffic measurement system," in *Proc. ACM SIGCOMM*, 2012.