Distributed and Collaborative Traffic Monitoring in Software Defined Networks

Ye Yu, Chen Qian, Xin Li
Department of Computer Science, Unversity of Kentucky
Lexington, Kentucky, USA
ye.yu@uky.edu, qian@cs.uky.edu, xin.li@uky.edu

ABSTRACT

Network traffic monitoring supports fundamental network management tasks. However, monitoring tasks introduce non-trivial overhead to network devices such as switches. We propose a Distributed and Collaborative Monitoring system, named DCM, with the following properties. First, DCM allows switches to collaboratively achieve flow monitoring tasks and balance measurement load. Second, DCM is able to perform per-flow monitoring, by which different groups of flows are monitored using different actions. Third, DCM is a memory-efficient solution for switch data plane and guarantees system scalability. DCM uses novel two-stage Bloom filters to represent monitoring rules using small memory space. It utilizes the centralized SDN control to install, update, and reconstruct the two-stage Bloom filters in the switch data plane. We study how DCM performs two representative monitoring tasks, namely flow size counting and packet sampling, and evaluate its performance. Experiments using real data center and ISP traffic data on real network topologies show that DCM achieves highest measurement accuracy among existing solutions given the same memory budget of switches.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operations—*Network monitoring*

General Terms

Algorithms, Design, Performance

Keywords

Software defined networking; Distributed network monitoring; Bloom filters

1. INTRODUCTION

Network traffic monitoring supports fundamental network management tasks, such as user application identification [17], anomaly detection [30], forensic analysis [26], and traffic engineering [4]. These tasks usually require the monitoring infrastructure to provide a complete, reliable, and accurate measurement of network-wide flow traffic. Recent studies [21] [20] [23] [30] have addressed two essential requirements of traffic monitoring, *load distribution* and *per-flow monitoring*.

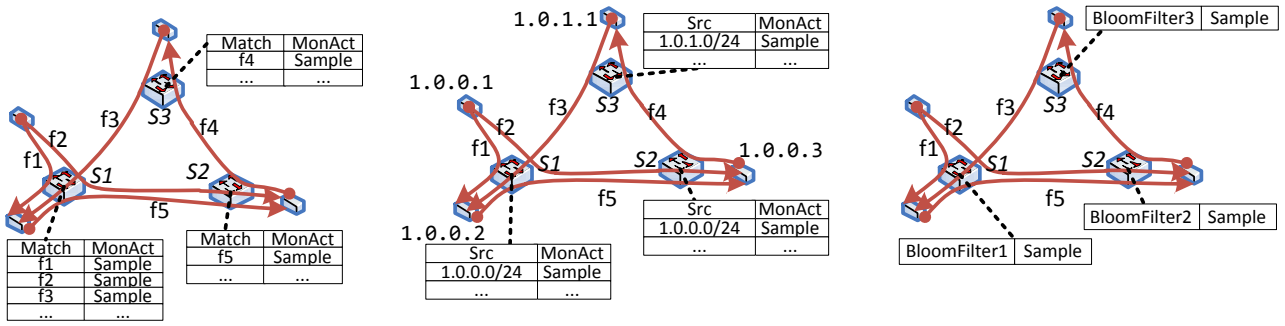
In most networks, a number of routers/switches independently monitor flows. These switches may consume tremendous resources (CPU, memory, bandwidth, etc) to perform monitoring tasks. On the other hand, some flows may not be covered by these switches [21]. To resolve this problem, *distributed and collaborative monitoring* [21] [20] [30] has been proposed to allow all the switches in the network to share monitoring load collaboratively.

Existing traffic measurement tools, e.g., Netflow [8] and sFlow [18], support generic measurement tasks based on packet sampling, where packets are selected with a given probability. Many applications, however, require *per-flow monitoring*, i.e., different monitoring actions performed on different flows. For example, a monitor may need to examine detailed traces from subsets of flows [19]. Anomaly detection prefers different sampling rates to flow groups [30]. A straightforward solution is to let switches store a monitoring rule for each flow. A monitoring rule includes matching fields and an action applied to the flow, such as sampling with a particular rate or counting packets. As demonstrated in [30], monitoring rule storage consumes non-trivial memory space (tens of thousands entries with aggregation in [30]) on a switch. As discussed in many studies [27] [29] [13] [7], fast switch memory is expensive, power-hungry, and hence very limited. Therefore rule-based per-flow monitoring has a memory scalability problem.

Current traffic monitoring tools either cannot meet both of the two requirements or are hard to deploy in practical networks. For example, cSamp [21] uses the hash values of the 5-tuples of packets to distribute sampling load among routers. However, cSamp requires all packets to carry their ingress-egress pairs, which are not available in practical networks [20]. The only two approaches that can achieve per-flow monitoring and load distribution are rule-based and aggregation-based flow monitoring. Figure 1a shows an example of rule-based monitoring. According to the rules stored on switches, the five flows f_1 to f_5 will be sampled separately at S_1, S_2, and S_3. As discussed, rule-based monitoring is limited by the switch memory space. Figure 1b shows a solution by aggregate-based approach to sample the five flows. The sampling task of f_1, f_2, and f_5 are assigned to S_1, f_4 is assigned to S_2, and f_3 is assigned to S_3. Source aggregation uses less memory to store rules. However, aggregation still requires a large rule table [30]. In addition, potential duplicate samples may occur. For example, f_5 is sampled twice at S_1 and S_2. More importantly, if an aggregate includes multiple flows, changes of small flows are hard to observe.

HotSDN'14, August 22, 2014, Chicago, IL, USA.

http://dx.doi.org/10.1145/2620728.2620739 .



(a) **Rule-based:** requires large rule storage cost

(b) **Aggregation-based:** flow $f2$ is monitored twice by $S1$ and $S2$.

(c) **DCM:** $f1$, $f2$, $f5$ match Bloom filter $BF1$; $f4$ matches $BF2$; $f3$ matches $BF3$.

Figure 1: Three distributed and collaborative monitoring methods

In this paper, we propose *a memory-efficient system for Distributed and Collaborative per-flow Monitoring*, named DCM. DCM uses Bloom filters [5] to represent monitoring rules using a small size of memory. It utilizes the tremendous convenience brought by the software defined networking (SDN) paradigm to install a customized and dynamic monitoring tool into the switch data plane. The novel monitoring tool used by DCM is called *two-stage Bloom filters*, including an admission Bloom filter to accept all flows assigned to the switch and a group of action Bloom filters to perform different measurement actions. SDN also allows DCM to perform updates or reconstruction of the two-stage Bloom filters in the switch data plane. Figure 1c shows an example of how to DCM to sample the five flows. Switch S_1 finds that f_1, f_2, and f_5 match its Bloom filter BF_1 and then samples packets of the three flows. Similarly S_2 samples f_4 and S_3 samples f_3. Although Bloom filters may introduce false positives, the design of two-stage Bloom filters can reduce the false positive rate to a negligible value with small memory cost. In addition, the SDN controller can detect all false positives and limit their negative influence due to its central view of the switches and flows.

The rest of the paper is organized as follows. In Section 2 we introduce the background knowledge of this work. In Section 3 we present the system design of DCM. In Section 4 we introduce how DCM performs two representative monitoring tasks: flow size counting and packet sampling. We also evaluate the performance of DCM for the two tasks using real data center and ISP traffic data and network topologies in the same section. Finally we conclude this work and present future work in Section 5.

2. BACKGROUND

2.1 Bloom Filter

A Bloom filter [5] B is a simple but space-efficient probabilistic data structure that represents a set of items S and supports membership queries. An item i may match B or fail to match B, depending on whether i is in S. One key problem of Bloom filters is false positives. The false positive probability of B is $(1 - e^{\frac{kn}{m}})^k$, where n is the size of S, m is B's length in bits, and k is the number of hash functions. Bloom filter and its variations have been widely used in the network community to solve various problems, distributed caching [10], P2P data management [6], unicast and multicast routing [27] [14], and network measurement [11] [24].

2.2 Related works

Traffic monitoring and measurement supports many network management tasks. The de-facto traffic monitoring standard is packet-based sampling, such as Netflow [8] and sFlow [18]. Furthermore, authors in [22] state the importance of using per-flow monitoring. cSamp [21] coordinates network-wide routers using a hash-based method to sample packets that carry the OD pair information. To make cSamp practical, cSamp-T [20] removes the assumption of OD pair information on packet headers and Decor [23] applies local information to avoid using central controller.

SDN-based traffic monitoring has been studied recently. OpenSketch [28] is a software defined traffic measurement architecture that applies sketches for various monitoring tasks, but it only discusses measurement actions on a single switch. A following paper [16] discusses the tradeoffs between the resource and accuracy of heavy hitter detection.

The SDN data plane scalability problem, i.e., limited rule storage space, has been addressed by recent work. DIFANE [29] and and Palette [13] propose to partition or distribute rules over the switches to reduce per-switch rule storage. They are mainly designed for packet forwarding rules rather than monitoring rules. Payless [7] and OpenWatch [30] use flow aggregates to complete different tasks and reduce the number of rules per-switch.

3. SYSTEM DESIGN

In this section, we detail the design of our Distributed Collaborative Monitoring (DCM) system.

3.1 Model and Assumptions

The objective of DCM is to distribute the monitoring duty of the targeted flows to the entire network, so that rule storage and packet processing overhead on switches can be reduced. DCM guarantees the following two properties: 1) every packet of a targeted flow should be monitored by at least one switch on its path; 2) if a packet is monitored by more than one switches, duplicate monitoring can be detected.

System Model:

- Flows are identified by the 5-tuple, i.e., $\langle$ *SrcIp*, *DstIp*, *SrcPort*, *DstPort*, *Protocol* $\rangle$.
- There is a centralized SDN controller that knows the information (including paths and 5-tuple) of all flows in the network. The controller maintains a monitoring table of the targeted flows and the corresponding monitor actions. Different flows may have different actions. The controller can communicate with a switch to install, update, and delete software-based monitoring tools in the switch data plane.

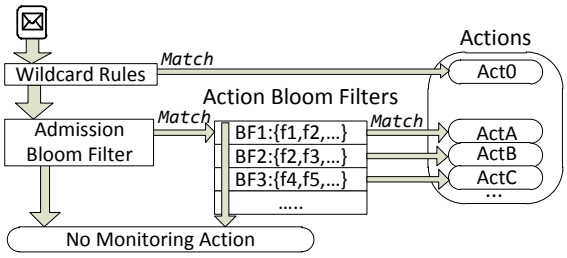


Figure 2: DCM switch data plane

- A switch installs monitoring tools and processes the packets it encounters. It records measurement results in its local memory and reports the results to the controller periodically. When a switch receives a packet of a new flow, it forwards the flow information to the controller. The controller decides whether, how, and where the flow should be monitored.

We assume that the memory space for monitoring tasks in a switch is limited while the controller has enough space to store detailed flow information and monitor actions.

3.2 DCM Data Plane on Switches

When a switch receives a packet to forward, the DCM data plane has three steps to process the packet, as shown in Figure 2. The flow-to-filter matching are based on the hash of a 5-tuple.

Step 1. The wild card matching step is to check whether the packet matches one of the wild card monitor rules. A wild card rule applies an action to an aggregate of flows. For example, if DCM wants to sample all flows whose sources have the same prefix, a wild card rule can specify such a monitoring task in a memory-efficient way. A packet that matches a wild card rule can then be processed by the specified action and then can skip the remaining.

Our main contributions are in the second and third steps using *two-stage Bloom filtering*.

Step 2. The first part of two-stage Bloom filtering is called the *admission Bloom filter* (admBF). The admBF represents the set of flows that should be monitored but the actions are not specified by any wildcard rule. The admBF does not specify any monitoring action. If a packet matches the admBF, it will then be further processed to get its action. If a packet does not match the admBF, the DCM data plane knows that it does not belong to any flow under monitoring and then skips the remaining step. Therefore the function of the admBF is to filter the flows that are not of interest.

Step 3. Flows that match the admBF will be further checked by the *action Bloom filters* (actBFs) to decide the corresponding monitoring actions. In the example of Figure 2, packets of flows f_1 and f_2 match BF_1 and are processed using Action A. Note that a flow may match multiple actBFs. For example, a flow may match one actBF for packet sampling and another actBF for counting at a same switch. It may also match actBFs at multiple switches due to false positives.

There are two principal reasons to design such two-stage Bloom filtering. First, using the admBF, most packets that are not monitored will be filtered and are not checked by the actBFs. Thus it saves the switch processing resource. Second, although some flows that should not be monitored do pass the admBF, the number of flows checked by the actBFs reduces significantly. Recall that the false positive probability of a Bloom filter is $(1 - e^{\frac{kn}{m}})^k$ where n is the size of the item set. Two-stage Bloom filtering reduces n for two potential performance gains: 1) given an actBF with size m, smaller n will result in fewer false positives; 2) given a certain false positive rate threshold, the length requirement of actBF decreases as n goes smaller.

All wild card rules, admBF, and actBFs are determined by the controller and installed on switches. Note that the DCM component does not perform any packet forwarding task.

3.3 Controller Operations

The DCM component on the controller is responsible for the allocation of monitoring load to switches, Bloom filter construction and updates, and false positive detection.

3.3.1 Monitoring load allocation

Given a set of flows to be monitored, the DCM controller distributes the monitoring load to all switches in the network. Such load distribution provides two main advantages. First, compared to today's approach where a switch independently monitors its flows, the collaborative monitoring reduces per-switch computing and recording overhead. Second, collaborative monitoring may achieve more accurate measurement results, because many measurement tools such as Bloom filters and sketches [28] have higher accuracy with lower per-switch load. The accuracy of many network monitoring tools have strong relations with per-switch monitoring load. For example, a Count-Min sketch [9] will demonstrate low accuracy when it processes too many different flows.

The main considerations for designing monitoring load distribution can be presented as follows. When there are a small number of flows to be monitored by an action A, we prefer to restrict the monitoring load of A on a few switches rather than all available ones in the network. This is because any switch performing A should store an individual actBF. When many flows need to be monitored by A, DCM introduces more switches to balance the load.

For a monitor action A, we define a threshold as ρ_A. If the number of flows that are processed by A on a switch exceeds ρ_A, we say the switch is overloaded of A. Actions may have different threshold because they consume different levels of resources. For example, packet sampling requires more storage space than counting.

For a new flow f to be monitored by action A, if there is at least one switch on f's path whose current monitoring load of A is less than ρ_A, it will be assigned to one of these switches. Otherwise, the controller assigns f to a switch on f's path that has no acfBF of A. In some extreme cases, all switches on f's path are overloaded, the controller will pick the one with the minimal load.

We provide pseudo-code of the algorithm **BalanceAllocate** for a group of flows.

BALANCEALLOCATE()

```
1  while there is a new flow f_i to be monitored.
2      do Let A be the monitoring action of f_i.
3          if All switch on f_i's path are unavailable or overloaded
   for action A
4              then choose switch s with most available resource.
5                   Add f_i to admBF_s;
6                   Add f_i to actBF_{s,A};
7      else Select a random switch s on f_i's path, whose load is
   lower than the threshold ρ_A.
8           Add f_i to admBF_s;
9           Add f_i to actBF_{s,A};
```

Note that all allocation results are only stored on the controller. The controller does not communicate with switches at this stage.

3.3.2 Bloom filter construction and updates

Based on flows assigned to a switch, the controller computes the admBF and actBFs for different actions of the switch. The false positive rates are pre-determined by the trade-offs between memory cost and accuracy. We recommend that an admBF should be constructed with a very low false positive rate for two reasons: 1) its false positives may be propagated to actBFs; 2) spending more memory on an admBF is cost-efficient as there is only one admBF on a switch. After constructing the admBFs and actBFs for all switches and actions, the controller encapsulates the Bloom filters in control messages and sends them to the switches.

The controller also needs to update Bloom filters according to flow dynamics. New flows may join the network and existing flows may end. Additionally, if the number of flows supported by a Bloom filter increases and the false positive rate is higher than the accuracy requirement, the Bloom filter needs to be reconstructed. It is known that a Bloom filter is easy to perform item addition operations on but hard to perform deletion operations on. Based on this property, the controller applies a policy called "*real-time addition and periodical reconstruction*" (RAPR). When the controller receives a flow to monitor, it will immediately notify the responsible switch to revise its Bloom filters to monitor the new flow. When the controller finds a flow finishes, it does not perform any operation. Instead, for every period of time T, the controller reconstructs all Bloom filters on a switch to remove finished flows and to adjust the filter sizes to meet the accuracy requirement. RAPR guarantees that all flows to monitor will be immediately monitored and reduces the computing and communication cost caused by frequent Bloom filter reconstructions. To maintain low false positive rates, the controller also periodically checks each filter using a timeout $T' < T$. If the false positive rate of a filter is higher than its requirement, the controller is also triggered to reconstruct a new filter.

3.3.3 False positive detection

Though DCM can control false positive rates, it does not completely eliminate false positives. Thus a flow may be monitored at multiple times on different switches, resulting in duplicate measurements. However, the controller is able to *detect all false positives* and *limit the negative influence of them*. The controller can maintain copies of Bloom filters installed on switches and the record of flow information. By testing a flow f using all Bloom filters on the switches along the flow path of f, the controller may identify all possible duplicate measurements. For example, if f is assigned to be sampled at a switch s_1 but also accidentally matches the Bloom filters at another switch s_2 on f's path, the controller knows the false positives and drops all samples of f reported by s_2.

3.4 Discussion of implementation

The DCM data plane on switches includes three functional components: hash functions, wildcard rule lookup, and Bloom filters. We find that all three components have already been implemented by existing work [12, 28, 27]. In particular, Yu *et al.* [28] uses NetFPGA to implement wildcard lookup and up to 8 hash functions, which is enough to implement the DCM data plane because all actBFs can use a same set of hash functions. The hash function implementation in [28] is efficient and has no effect on data plane throughput. Bloom filters can be implemented either in TCAM [12] or in SRAM [27] with slower speed.

4. CASE STUDY AND EVALUATION

In this section we show how DCM supports single-action and multi-action monitoring by studying two representative measurement tasks: flow size counting using Count-Min (CM) sketch and packet sampling.

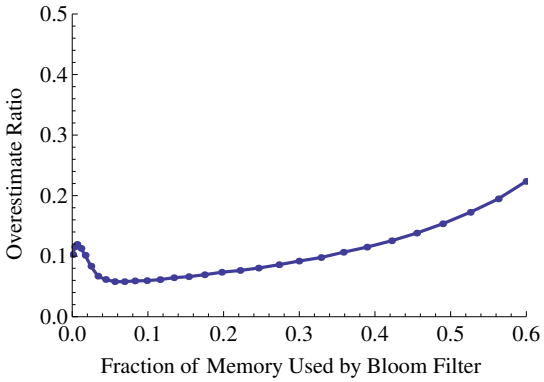


Figure 3: Overestimate ratio v.s. fraction of memory for Bloom filter

We also compare DCM with two existing monitoring methods: Aggregation-based monitoring [30] and Monitor-All, which is a naive solution where each switch independently monitors all flows. For Monitor-All, we reuse the code of OpenSketch implementation [28].We conduct experiments to evaluate the counting accuracy and false positive samples by varying the memory size used for monitoring on each switch. Rule-based monitoring is not feasible in all experiments , using the limited size of memory.

We conduct the experiments using two real traffic traces: the EDU1 traffic data from a university data center network [3] and the CAIDA Anonymized Internet Traces 2013 dataset [1].

Three network topologies are used: 1) EDU1, a dual-core, star-shaped topology of the campus data center network in [3]; 2) Fat-Tree, a typical multi-rooted tree topology [2]; and 3) RocketFuel 3967, the router-level ISP network topology of AS 3967 [25]. We apply the EDU1 data on topologies EDU1 and Fat-Tree, and the CAIDA data on RocketFuel.

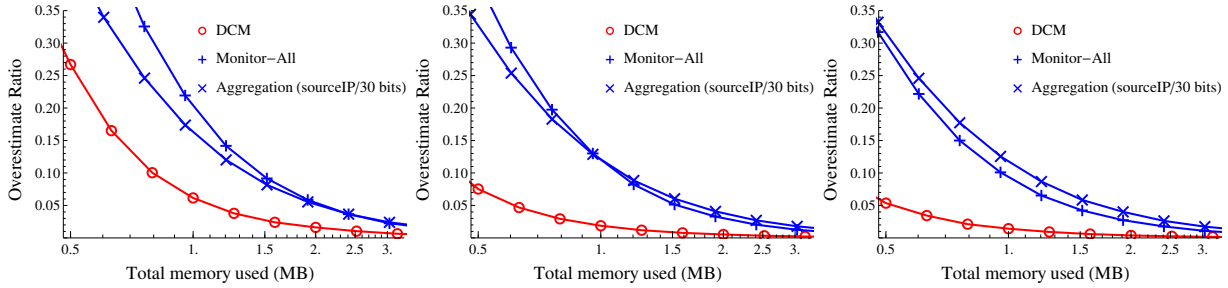
4.1 Flow Size Counting with Count-Min Sketch

Flow size counting using CM sketches [9] has been implemented by OpenSketch [28] for single-switch traffic measurement. Here we discuss how to use DCM and CM sketches for distributed and collaborative monitoring across the network.

A CM sketch is an efficient and probabilistic data structure to support cardinality queries of multiple sets. A CM sketch consists of k arrays $A_1, A_2, \cdots A_k$, which each includes multiple counters. On processing each of the packets of flow f, the switch computes k hash values and increments the counter at $A_i[h_i(f)]$. To answer the query for the number of packets of f, the value $MIN\{A_i[h_i(f)]\}$ is returned as an estimation of f's size. CM sketches introduce overestimation because of hashing collisions. The counting accuracy degrades with increase of overall packet numbers and improves with increase of memory size allocated to store the sketch.

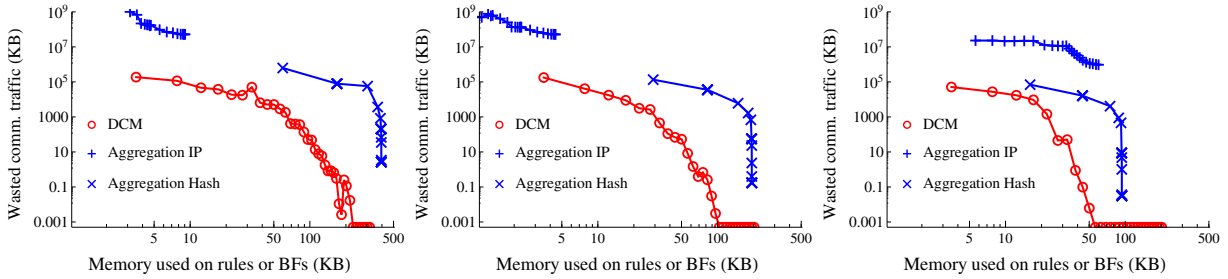
Flow size counting is a single-action monitoring task. Hence we only need one Bloom filter if no other task is performed at the same time. In the DCM data plane of a switch, a fixed size of memory may be allocated for the Bloom filter and CM sketch.Note that the memory sizes of both the Bloom filter and CM sketch impact the accuracy of flow size counting.

We conduct the experiments using the EDU1 data and topology. Fig 3 shows how the average overestimate ratio of the network changes against the fraction of memory used by the Bloom filter, with the total memory limited to 1 MB per switch. We find that the



(a) EDU1 (b) Fat Tree (c) Rocketfuel

Figure 4: Flow size count: overestimate ratio v.s. total memory consumption



(a) EDU1 (b) Fat Tree (c) Rocketfuel

Figure 5: Single-rate sampling: wasted communication traffic v.s. memory used on rules or Bloom filters

Bloom filter only requires a small fraction of memory (less than 5%) to achieve the lowest overestimate ratio (about 0.05). When it takes more memory, the accuracy becomes worse because the CM sketch uses less memory.

We compare DCM with Monitor-All and source IP aggregation using 30-bit mask length in Fig 4. We find for all three networks, when provided with same amount of memory, DCM achieves much smaller overestimate ratio than both Aggregation and Monitor-All. Given 2 MB memory, DCM has very little overestimate (0.02 for EDU1 and less than 0.01 for Fat Tree and Rocketfuel). Note that Monitor-All can allocate all memory to the CM sketch, but it has a main problem: each switch is responsible for all flows passing through it. When more packets are mapped to a CM sketch, its accuracy degrades.

4.2 Flow Sampling

Single-rate sampling. The objective of single-rate sampling is to obtain a fraction of packets from particular flows. Single-rate sampling is an example of single-action monitoring, where a switch requires only one Bloom filter to identify the flows to monitor.

We compare DCM with two types of aggregation-based methods, for single-rate sampling. Aggregation IP groups IP addresses by both source and destination prefixes in a certain length. Aggregation Hash aggregates flows by their prefixes of the hash values of 5-tuples. Both DCM and Aggregation have false positives which can be detected. However the communication cost of the report messages from switches to the controller is wasted for the false positive samples.

Figure 5 shows the wasted communication cost versus the memory used for rule or Bloom filter storage. Given the same memory size, DCM causes much less wasted communication cost than Aggregation methods by about two orders of magnitude. Using 100 KB for Bloom filters, DCM only wastes 100 KB traffic in EDU1

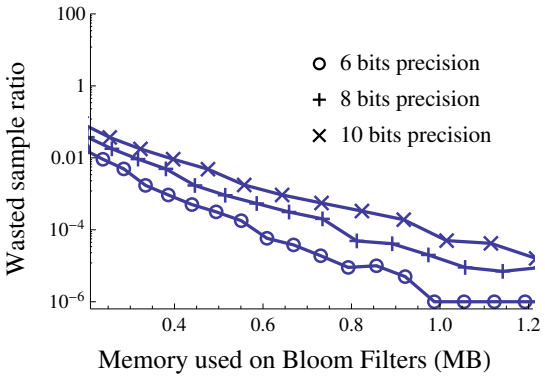


Figure 6: Multi-rate sampling: wasted sample ratio v.s. memory for Bloom filters

and almost none in Fat-Tree and Rocketfuel. Aggregation IP can only use a limited range of memory because the mask length cannot be longer than 32. Using 32-bit source and destination masks, false positives still occur due to different port numbers.

In aggregation solutions, a better granularity may reduce the wasted communication cost, but requires larger memory size. Moreover , the granularity is limited by the length of IP address and TCAM keyword length [15]. In our experiments, the granularity is limited to 64 bits.

Multi-rate sampling. As a multi-action monitoring task, multi-rate sampling requires DCM to use multiple actBFs. Consider a hash function H that maps the packet-related data of packet p, e.g., 5-tuple plus sequence number (for TCP) or checksum (Non-TCP), to a value $H(p)$ uniformly in (0, 1). There are a set of monitor actions $A_1, A_2, \cdots A_k$, where A_i specifies that p should be sampled

if $H(p)$ falls between $\frac{1}{2^i}$ and $\frac{1}{2^{i-1}}$. Hence a flow of packets will be sampled by A_i with a rate of 2^{-i}. For a given ratio p, we construct a number sequence b_1, b_2, $\cdots$, where b_t is the position of the t-th 1 in p's binary expression. Thus, $p = \sum 2^{-b_t}$. For example, if a flow should be sampled with rate $\frac{11}{16} = (0.1011)_2$, its 5-tuple can match three actBFs whose actions are A_1, A_3, and A_4. There is no duplicate sampling by different monitor actions , because the hash of a particular packet will fall into the interval of at most one action A_i. Note that a coefficient can always be applied on a sample action to get a lower rate.

In our evaluation, each flow is given a random sample rate. We vary the precision of the rate binary expression by 6, 8, and 10 bits. Due to false positives, a packet could be sampled on multiple switches. Duplicate samples can always be detected by the controller as discussed in Section 3.3.3. These duplicates are considered wasted samples. Figure 6 shows the wasted sample ratio versus the memory used by Bloom filters. When more than 1 MB is used, multi-rate sampling of all levels of precision has negligible wasted samples.

5. CONCLUSION AND FUTURE WORK

We propose a Distributed Collaborative Monitoring (DCM) system for SDN-enabled flow monitoring and measurement. We have designed novel two-stage Bloom filters as the DCM data plane to represent monitoring rules in an efficient and reliable way. Experiments using real traffic data and network topologies show that DCM provides accurate and memory-efficient flow measurement for two representative tasks, i.e., flow size counting and packet sampling. Compared to current solutions, DCM provides network-wide flow coverage and achieves high measurement accuracy using the same memory size.

In the future, we will explore the following problems.

DCM configuration under traffic dynamics. In practice, monitoring load may change dynamically, which motivates us to design sophisticated DCM data plane construction and improve algorithms. We will quantitatively analyze and evaluate the impact of different DCM data plane configurations by varying a number of parameters, including size and number of Bloom filters, fractions of memory allocated for admBf and actBFs, and reconstruction period.

Load assignment optimization. We also plan to design and analyze different load assignment algorithms to achieve optimal load balance, memory efficiency, and accuracy.

Prototype implementation. We plan to implement a DCM prototype and try to apply it for real traffic monitoring tasks in our campus network, where OpenFlow switches have already been deployed for other network management purposes.

6. ACKNOWLEDGEMENT

We thank the anonymous reviewers for their constructive comments and suggestions.

7. REFERENCES

[1] The caida ucsd anonymized internet traces 2013 - 2014. mar. `http://www.caida.org/data/passive/passive_2013_dataset.xml`.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. of ACM SIGCOMM*, 2008.

[3] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of ACM IMC*, 2010.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: fine grained traffic engineering for data centers. In *Proc. of ACM CoNEXT*, 2011.

[5] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[6] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. In *Proc. of ACM SIGCOMM*, 2002.

[7] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost netowrk monitoring framework for software defined networks. In *Proc. of IEEE/IFIP NOMS*, 2014.

[8] B. Claise. Cisco systems netflow services export version 9, 2004.

[9] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[11] W. Feng, K. G. Shin, D. D. Kandlur, and D. Saha. The blue active queue management algorithms. *IEEE/ACM Transactions on Networking*, 10(4):513–528, 2002.

[12] A. Goel and P. Gupta. Small subset queries and bloom filters using ternary associative memories, with applications. In *Proc. of ACM SIGMETRICS*, 2010.

[13] Y. Kanizo, D. Hay, and I. Keslassy. Palette: Distributing tables in software-defined networks. In *Proc. of IEEE INFOCOM*, 2013.

[14] D. Li, H. Cui, Y. Hu, Y. Xia, and X. Wang. Scalable data center multicast using multi-class bloom filter. In *Proc. of IEEE ICNP*, 2011.

[15] T. Mishra and S. Sahni. Duo-dual tcam architecture for routing tables with incremental update. In *Proc. of IEEE ISCC*, 2010.

[16] M. Moshref, M. Yu, and R. Govindan. Resource/accuracy tradeoffs in software-defined measurement. In *Proc. of ACM HotSDN*, 2013.

[17] T. Pan, X. Guo, C. Zhang, J. Jiang, H. Wu, and B. Liu. Tracking millions of flows in high speed networks for application identification. In *Proc. of IEEE INFOCOM*, 2012.

[18] P. Phaal and M. Lavine. sflow version 5, 2004.

[19] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani. Fast monitoring of traffic subpopulations. In *Proc. of ACM IMC*, 2008.

[20] V. Sekar, A. Gupta, M. K. Reiter, and H. Zhang. Coordinated sampling sans origin-destination identifiers: algorithms and analysis. In *Proc. of IEEE COMSNETS*, 2010.

[21] 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 *Proc. of USENIX NSDI*, 2008.

[22] V. Sekar, M. K. Reiter, and H. Zhang. Revisiting the case for a minimalist approach for network flow monitoring. In *Proc. of ACM IMC*, 2010.

[23] S. Shen and A. Akella. Decor: a distributed coordinated resource monitoring system. In *Proc. of IEEE IWQoS*, 2012.

[24] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent, and W. T. Strayer. Single-packet ip traceback. *IEEE/ACM Transactions on Networking*, 10(6):721–734, 2002.

[25] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. 2002.

[26] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang. Worm origin identification using random moonwalks. In *Proc. of IEEE S&P*, 2005.

[27] M. Yu, A. Fabrikant, and J. Rexford. Buffalo: Bloom filter forwarding architecture for large organizations. In *Proceedings of ACM CoNEXT*, 2009.

[28] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Proc. of USENIX NSDI*, 2013.

[29] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with difane. In *Proc. of ACM SIGCOMM*, 2010.

[30] Y. Zhang. An adaptive flow counting method for anomaly detection in sdn. In *Proc. of ACM CoNEXT*, 2013.