Resource/Accuracy Tradeoffs in Software-Defined Measurement

Masoud Moshref
moshrefj@usc.edu

Minlan Yu
minlanyu@usc.edu

Ramesh Govindan
ramesh@usc.edu

University of Southern California

ABSTRACT

Previous work on network measurements have explored several primitives of increasing complexity for measurement tasks at individual nodes, ranging from counters to hashing to arbitrary code fragments. In an SDN network, these primitives may require significant bandwidth, memory and processing resources, and the resources dedicated to these can affect the accuracy of the eventual measurement. In this paper, we first qualitatively discuss the tradeoff space of resource usage versus accuracy for these different primitives as a function of the spatial and temporal measurement granularity, then quantify these tradeoffs in the context of *hierarchical heavy hitter detection*.

Categories and Subject Descriptors

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

General Terms

Design, Measurement

Keywords

Data Center, Hierarchical Heavy Hitter, Software Defined Measurement, Software Defined Networking

1. INTRODUCTION

Traffic measurement plays an important role in data center and enterprise networks. Many management tasks such as traffic accounting, traffic engineering, load balancing, and performance diagnosis [3, 5, 20, 9] all rely on accurate and timely measurement of a large variety of traffic at different time-scales. For example, to perform traffic engineering [3], we need to correctly detect large flow aggregates in minutes and pick better routes for these flows. To reduce the latency of partition/aggregate workloads in data centers, we need to quickly identify short traffic bursts in hundreds of milliseconds (e.g., incast [20]).

Software-defined networks can enable programmable measurement in networks [21]. In this *software-defined measurement* approach, multiple measurement tasks can execute concurrently. The SDN controller can orchestrate measurement collection at multiple spatial and temporal scales, based on a global view of the network.

Previous work on network measurements [9, 14, 21] have explored several measurement primitives at switches/routers for measurement tasks. Some tasks can be accurately measured using simple counters, while other tasks may need hash-based data structures (e.g., sketches). Finally, some tasks may require arbitrary programmability at switches, namely, small code fragments performing measurements.

These primitives require different amounts of resources. Counters occupy expensive TCAM memory, hash-based data structures need SRAM, and code fragments require CPU for processing. All of these three approaches differ in the amount of network bandwidth they need to communicate intermediate measurement results with the SDN controller. This resource usage is a function of the time and spatial granularity at which measurement is required; intuitively, more resources are required to measure frequently varying traffic or longer prefixes. When multiple measurement tasks are run concurrently, the resources allocated to each task determines its accuracy.

In this setting, an SDN controller must manage these resources carefully in order to ensure accuracy of the measurement results. In this paper, we first discuss these resource/accuracy tradeoffs for different primitives, and present a qualitative understanding of these primitives, their resource usage, and their accuracy, as a function of the spatial and temporal granularity of measurement. Then, we take a concrete measurement task (the detection of hierarchical heavy hitters) and understand how to design software-based measurement algorithms for this task, using the different primitives that we have described. This, together with preliminary experiments, gives us a better understanding of which primitives are more accurate at which spatial and temporal granularities. It also illustrates the division of labor between the controller and switches, demonstrating the potential of software-defined measurement. For future work, we will explore the design of a comprehensive system that supports multiple concurrent measurement tasks and careful controller-driven management of measurement resources.

2. RESOURCE/ACCURACY TRADEOFFS: MOTIVATION AND CHALLENGES

In previous work [21], we have identified the potential of software-defined measurement (SDM) in SDNs. The programmability of SDNs permits multiple concurrent measurement tasks to be concurrently executed in an SDN. Moreover, an SDN controller can dynamically adapt which measurements are performed at which switch, possibly based on the traffic matrix. In the previous work, we explored this capability by using *sketch-based* measurement for *localized detection* tasks — measurement tasks that detect whether

HotSDN'13, August 16, 2013, Hong Kong, China.

a particular phenomenon (e.g., a super-spreader or a heavy hitter) occurs at each of the switches in the network. In this paper, we highlight the fact that measurement tasks can be realized using different *measurement primitives* at individual switches, each of which sits in a different point in the design space for SDM. Moreover, our vision is *global detection* tasks, ones in which the measurement results from a single switch will not, in general, suffice to detect the existence of the phenomenon [1]. An example of such a task is *hierarchical heavy hitter* detection, which identifies the longest IP prefixes with aggregate traffic above a certain threshold, with the additional property that none of the sub-prefixes have traffic that exceeds the threshold. We describe this measurement task in more detail in Section 3.1 but first discuss various measurement primitives below.

Measurement primitives:

(1) Counting: The flow-based switches (e.g., OpenFlow) allow operators to flexibly specify the flows to monitor based on different packet fields (e.g., source and/or destination IP addresses), and *count* the number of bytes or packets for these flows. These switches are already implemented by various vendors (e.g., HP, NEC). Since these flow-based counters are maintained in a power-hungry TCAM, we can only use a limited number of TCAM entries for measurement.

Flow-based switches only update flow-based counters, and rely on the controller to perform analysis on these counters. Based on the analysis of the counters, the controller then periodically adjusts the measurement rules at the switches. Due to the delay and bandwidth constraints between the controller and the switches, the controller can only achieve accurate analysis for measurements at large time-scales (e.g., seconds). In addition, since the controller adjusts the rules based on the traffic history, this approach only works for traffic that changes at time-scales larger than the periodicity at which the controller adapts measurements.

(2) Hashing: Recent work [9, 21] proposed to implement hash-based data structures at switches to count traffic. These hash-based switches are not implemented in today's switches but are feasible to implement with existing commodity switch components. Most of the counters in these switches are stored in SRAM, which is cheaper and larger than TCAM.

These hash-based switches extract summaries of the traffic (e.g., count unique elements [11] and calculate flow size distribution [15]), and then transfer the results to the controller for further analysis. Intuitively, hash-based switches can cope with more traffic variability. The controller can also periodically reconfigure the data structures at switches according to the traffic characteristics. Since these solutions still rely on sending all the counters to the controller for analysis, they can only support measurement analysis at relatively large time granularities.

(3) Programming: We can further leverage the local CPU at switches to run *simple* measurement programs to collect and analyze more data at switches (e.g., [19]). These switches are harder to implement in practice, but when implemented, can significantly improve the switch measurement capabilities.

If we can perform local analysis at the switches locally, we can easily support small time-scale measurement and highly variable traffic at line speed. However, for global detection tasks, we still need the controller to collect the measurement data from these switches and perform the global analysis.

[1]We focus on a single switch case in this paper as the base for multiple switch case discussed in Section 6.

The tradeoff between resource usage and accuracy. These different primitives have different expressivity and each primitive is suitable for measurements at different spatial and temporal granularities. Put differently, a given measurement task can be achieved with different accuracy by using different primitives; e.g., for highly variable traffic, counting may be less accurate than hashing. Now, these primitives use different types (CPU, memory, network) and amounts of resources for measurement tasks. In the future, if SDNs were to support all three primitives for measurement, an interesting design challenge for SDM would be to manage resources across multiple concurrent tasks. Implicitly, the resource allocated to a particular measurement task (e.g., TCAM space for counters, bandwidth limits for transmitting measurements to the controller) constrains the accuracy of the results.

For example, to monitor the traffic from source IP prefix 10.1.1.5 in a data center, the operator may want to save a rule on a top-of-rack switch. However, if there are insufficient TCAM entries, the controller can sacrifice accuracy by saving the rule at the aggregator switch or merge the rule with another one and monitor 10.1.1.4/31, for example. The other option is to switch to a hash-based approach and use SRAM resources of the switch. Each of the aforementioned solutions represents a different resource/accuracy tradeoff.

Thus, understanding these resource/accuracy tradeoffs for SDM and designing methods to manage measurement resources is an important research challenge for future SDNs. In this paper, we take the first step towards this challenge, by exploring these questions for hierarchical heavy hitter detection.

3. CASE STUDY: HIERARCHICAL HEAVY HITTERS

In this section, we use identifying hierarchical heavy hitters (HHH) to show how to support different measurement requirements using different switch primitives under different resource constraints. Table 1 summarizes the design space for resource/accuracy tradeoffs. When the traffic is stable, we can leverage the controller to install monitoring rules and the flow-based switches to collect counters. When the traffic is variable, hash-based switches which collect counters for all the prefixes at line speed have higher accuracy. In order to detect HHHs at smaller time-scales, we must leverage programmable switches while using the controller for configuring these switches.

3.1 The hierarchical heavy hitter problem

Operators need to detect important traffic in the network for accounting, traffic engineering [3], and anomaly detection. One way to define important traffic is the *heavy hitter (HH)* along certain dimensions. For example, we can define heavy hitters as the source IP prefixes that contribute more than a fraction T of link capacity over the past epoch of p seconds. Figure 1 shows an example of bandwidth usage percentage for each IP prefix during an epoch. With a threshold of 10%, there are a total of six heavy hitters as notated in double circles. However, reporting *all* heavy hitter prefixes can be redundant; for example, reporting 01* as a heavy hitter is not as useful as reporting its descendent 010 which contributes the most to the traffic volume.

Therefore, *Hierarchical Heavy Hitters (HHH)* are introduced in [7] to denote the longest IP prefixes that contribute a large amount of traffic (based on threshold T) after *excluding* any HHH descendants in the prefix tree. For example, in Figure 1, prefix 00* is a HHH as IPs 000 and 001 *collectively* have large traffic, but prefix 0** is not a HHH because excluding descendent HHHs (00* and 010) its traffic is less than the threshold. Note that the prefix tree can be defined on one or multiple packet fields such as source and

Sec	Traffic	Time-scale	Switch	Controller
3.2	Stable	Large	Flow-based: counters for some prefixes	Max-Cover algorithm to pick the prefixes to monitor
3.3	Variable	Large	Hash-based: one Count-Min sketch for each prefix length	Configure sketches to reduce resource usage
3.4	Variable	Small	Programmable: identify HHs with sketches or space-saving algorithm	Extract HHHs from HHs
3.5	Stable	Multiple	Flow-based: counters for selected prefixes and time-scales	Pick the prefixes and time-scales to monitor
3.5	Variable	Multiple	Programmable: counters/sketches with exponential buckets for all time scales	Configure prefixes to monitor/sketches to reduce resource usage

Table 1: Understanding the design space for identifying HHHs with different traffic stability and in different time-scales

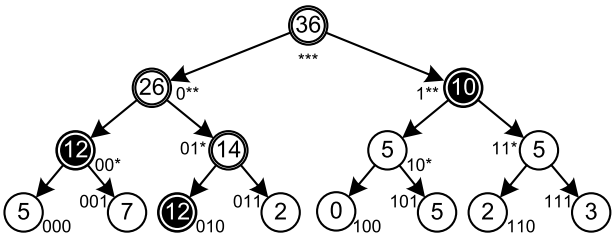


Figure 1: A prefix tree (trie) of source IPs where the number on each node shows the fraction of link capacity used by the associated IP prefix. With threshold 10, the nodes in double circles are heavy hitters and the nodes with shaded background are hierarchical heavy hitters.

destination IPs. In this paper, we focus on one-dimensional prefix tree on the source IP field.

The difficulty of identifying HHHs highly depends on the traffic stability. For example, operators often set up rate limiters to backup traffic, making it stable in volume. In other settings too (high performance computing [4]), inter-process communications vary slowly if ever. For such slowly-varying traffic, the HHH at one epoch is likely to stay as a HHH in the next epoch. In contrast, individual virtual machines can send ad-hoc large flows. The partition/aggregation traffic is also bursty (e.g., incast [20]). HHHs caused by these traffic are often harder to catch because they appear suddenly and often require in-time detection. As a result, it is harder for the remote controller to collect timely measurement information and analysis than switches.

Different network management tasks need to detect HHHs in different time-scales (i.e., epoch sizes). To perform traffic engineering, we need to detect and route large flows in minutes [3] or seconds [5]. To set up optical links on demand [10], we need to understand the traffic matrix in seconds. For troubleshooting performance problems, operators care about the HHHs both in hours/days for identifying permanent problems, and in seconds or milliseconds for identifying network congestion and packet losses. Due to the latency and bandwidth overhead of sending measurement data to the controller, we need to leverage the switches to help for small time-scale HHHs while relying on the controller for larger time-scale HHHs.

3.2 Stable traffic and large time-scale

When traffic is stable and the time-scale is large, we propose to leverage flow-based switches to count traffic for different prefixes and report these counters to the controller. The controller can then analyze these counters to identify HHHs. It also *adjusts* the counters based on the traffic history to improve HHH detection in the next epoch. This adjustment works for relatively stable or slowly varying traffic where history can guide finding future HHHs. Besides, this solution works for large time-scale because its accuracy depends directly on the number of available counters and the control loop latency: the number of counters is limited by the amount of expensive power-hungry TCAMs at the switches and control network bandwidth. Sending the counters to the controller *every epoch* uses control network bandwidth, and at larger time-scales, more control traffic can be sent per epoch (for fixed control traffic bandwidth). For example with 256Kbps link, we can send 16K two-byte counters in 1 second epochs while only 1.6K are available in 0.1 second. The control loop latency is the delay from sending the counters to the controller at the switches to the end of installing the new counting rules and includes the propagation delay, controller algorithm delay and switch rule installation delay. The counter adjustments are only valid if this latency is much smaller than the time-scale.

Max-Cover HHH algorithm on flow-based switches: To pick which prefixes to monitor, given a limit on the number of counters (i.e., a resource limit), we propose *Max-Cover HHH algorithm* that runs in each epoch in the controller. The algorithm works based on the assumption that with slowly-varying traffic, the HHHs in the previous epoch are likely to stay as HHHs in the next epoch and new HHHs appear where the traffic is large.

As a result, given the counters for prefixes in the previous epoch, we first pick the monitored prefix with maximum traffic and monitor its two children instead. Since, we do not have records of the counters for children in the previous epoch, we just assign half of the parent's traffic to each child.[2] For example in Figure 1, the controller monitored four prefixes in the previous epoch: 0**, 10*, 110, 111. Prefix 0** uses maximum bandwidth (26%), so we monitor its children 00* and 01* and assign 13% to each child.

Second, if the total number of monitoring prefixes goes beyond the resource limit, we pick a pair of sibling prefixes in the set with the minimum total traffic and monitor their parent instead to decrease monitoring prefixes. For example in Figure 1, suppose we can monitor at most *four* prefixes in each epoch. We have five monitoring prefixes after splitting the node 0**, so we pick sibling nodes 110 and 111 and monitor their parent 11*.

The algorithm continues splitting and merging until there is no pair of sibling prefixes whose total traffic is less than the maximum traffic. For Figure 1 example, the algorithm continues by splitting node 01* to 010 and 011 with 6.5% estimated traffic and merging 10* and 11* to 1**. At this step, there is no pair of sibling prefixes in the monitored set, whose total traffic is smaller than the maximum traffic of the monitored nodes.[3]

[2]In future work, we will explore better ways to estimate the traffic based on the traffic history.

[3] Note that this is based on the estimated traffic volume because the real ones are unknown to the controller.

With the monitoring prefix set generated by the Max-Cover algorithm, we can easily calculate the traffic of all the ancestors of prefixes in the monitoring prefix set. We call these ancestors together with prefixes in the monitoring prefix set the *coverage set*. The Max-Cover algorithm tries to maximize the traffic in this set and based on its assumptions maximizes the probability of finding HHHs.

3.3 Variable traffic and large time-scale

For large threshold T, the HHHs are mostly among prefixes with shorter length covering many servers, whose traffic are often more stable due to statistical multiplexing. However, when we decrease the threshold T, there are more HHHs with longer prefixes, whose traffic may change more frequently (e.g., it is more likely that a single VM sends bursty flows). When the traffic changes significantly over time, accuracy may be compromised because the controller can no longer pick the right prefix set to monitor based on the traffic history. A solution is to monitor more prefixes which is expensive because it uses more TCAMs.

An alternative is to track the traffic for those prefixes with variable traffic by leveraging hash-based switches. Hash-based switches use SRAM memory that can be larger than TCAM memory (millions vs thousands of counters) and is cheaper. For example, we can implement *Count-Min sketches* on hash-based switches, which leverages multiple hash functions to approximate the traffic of different IP prefixes in a compact way [8]. To detect HHHs, the switch maintains one Count-Min sketch for each level of the prefix tree and exports the sketches to the controller every epoch. The controller then traverses down the prefix tree and identifies the HHHs [7]. We compare the accuracy of this proposal with Max-Cover algorithm for equal cost switches in Section 4.

However, maintaining sketches for *all* prefix lengths takes too many resources (both switch memory and the bandwidth to the controller), especially when the threshold T is small which requires more accurate traffic approximation. To reduce the resource usage, the controller must use counters for prefixes with slowly-varying traffic while using sketches for prefixes with more variable traffic. We can also leverage the controller to configure the resource usage across the sketches. For example, if the controller is sure that only one prefix needs more accurate traffic counters, it can use fewer resources for general sketches and instantiate a specific sketch for that prefix to save memory and bandwidth. We have left a detailed exploration of these to future work.

3.4 Variable traffic and small time-scale

For detecting HHHs in small time-scales, the bandwidth overhead and delay of sending the counters or sketches to the controller every epoch become too large. Therefore, we need to run a simple piece of code using programmable switches to report heavy hitters to the controller, and the controller can analyze these heavy hitters to extract HHHs.

There are two approaches to report heavy hitters with different code complexity and efficiency: (1) we can leverage Count-Min Sketch to collect counters for prefixes at each prefix length as in Section 3.3, and then pick the largest ones as heavy hitters, and reports them to the controller in each epoch. (2) We can run the Space-Saving algorithm [17] on the programmable switches. We maintain counters for a list of IPs at the switch. When the switch receives a packet whose source IP is not listed, it adds a counter for this IP. If the list becomes full given the monitoring budget (e.g., switch memory size), the switch picks the counter with minimum value and replaces it with the counter for the new IP. In each epoch, the switch reports heavy hitters based on the list of counters. The second approach runs more complex code at switches than the first approach, but is faster and more accurate with less memory usage [6].

3.5 Multiple time-scales

Identifying HHHs in multiple time-scales is an example of supporting multiple measurement tasks and is even more challenging. We can track the HHHs in each time-scale separately, but as the resources are limited, it is important to optimize the resource allocation by leveraging the shared information among tasks. We propose three approaches in the increasing order of complexity at switches:

(1) The controller decides both which prefix to measure and at which time-scale for stable traffic. The controller can decide which prefix to measure by leveraging the Max-Cover algorithm we proposed in Section 3.2. The controller can also decide which time-scales to measure based on the stability of the traffic. For example, if the traffic is slowly-varying at small time-scales, the controller can install rules at the switches to monitor only large time-scales. This approach only requires flow-based switches and has lower bandwidth overhead, but it depends on the traffic stability.

(2) The controller decides the prefix set to measure and the switch monitors each prefix in all time-scales. With programmable switches, we can measure the traffic volumes for a prefix across multiple time-scales using *exponential bucketing* [19]. For each prefix, the switch maintains counters for exponentially increasing time-scales (e.g., for 1, 2, 4 seconds, etc.). The controller can combine these buckets together to estimate the traffic in any time-scale. This approach is more accurate than the first approach, but requires programmable switches and more bandwidth to collect traffic at all time-scales.

(3) The switch maintains sketches at all time-scales; the controller configures the parameters for sketches for variable traffic. We can replace the counter in the sketches with the exponential buckets to monitor traffic at all time-scales [18]. This approach can handle variable traffic at multiple time-scales because switches keep track of all the traffic at all time-scales, but it needs the most bandwidth and is more complex than the two previous approaches.

For future work, we will explore the tradeoffs of the three approaches and identify the right primitives at switches with the best accuracy and efficiency.

4. PRELIMINARY EVALUATION

We compare the accuracy of flow-based and hash-based HHH detection algorithms for single switch with equal switch resource usage cost. We use Max-Cover algorithm (Section 3.2) for flow-based switches and hierarchical Count-Min sketches (Section 3.3) for hash-based switches. The simulation uses the CAIDA packet trace [1] in different time-scales (500ms and 5s) and runs for 3 minutes with 10 warm-up epochs. We change the threshold value from 1% to 10% as the fraction of the maximum amount of traffic in all measurement epochs. We size the SRAM memory for the hash-based switch based on the switch control network bandwidth (1 Mbps, 256 Kbps and 64 Kbps[4]) used for sending two bytes counters. Then we use an equal-cost TCAM memory for Max-Cover algorithm assuming that each TCAM entry is 80 times more expensive than SRAM [16]. For example if we have 256 Kbps, 8K counters can be reported per 0.5 second epoch. So Count-Min will

[4] At larger capacities, the tradeoff happens for smaller time-scales. However, we believe the chosen numbers are fair considering the number of switches, measurement tasks and other control traffic in the data centers.

have $d = 4$ hash functions each with $w = 80$ counters for each prefix length, but Max-Cover can only have 100 TCAM entries. To maintain the accuracy in Count-Min hashes for small bandwidth and thresholds, we set minimum w to $2/threshold$ [8] and just trim the prefix tree to satisfy the bandwidth constraint. We compare the accuracy of algorithms using *precision* and *recall* calculated using a ground truth that monitors all IPs.

Figure 2(a) shows that for a large threshold, Max-Cover algorithm has good recall and lower traffic overhead (recall it uses 80 times fewer counters) compared to the Count-Min sketch. However, with smaller threshold, its accuracy drops and it is better to use Count-Min sketch. For example, for the threshold of 1% and 64Kbps control link bandwidth, Count-Min finds twice true HHHs comparing to the Max-Cover algorithm (70% vs 30%). Note that precision also follows the same trend (70% vs 35%) in Figure 2(b). We believe that this is because smaller thresholds create HHHs in deeper levels of the prefix tree (smaller spatial granularity) which are more variable and require Max-Cover algorithm to use more TCAM entries to find them. Our experiments with injected HHHs validate the effect of variability and level of HHHs on Max-Cover algorithm.

Figure 2(c) shows that for smaller epochs (0.5s), where we have 10 times fewer counters, the accuracy of both algorithms decreases. However, our conclusions that 1) The accuracy of sketch-based switches is better for restricted switch cost and small thresholds 2) We can save control network bandwidth with Max-Cover algorithm for large threshold and more expensive switches without losing accuracy are still valid. To summarize, software-defined measurement can optimize the resource usage by using the right primitives based on the measurement task parameters and traffic properties.

5. RELATED WORK

Centralized HHH detection algorithms: There are many proposals for HHH detection [7, 12, 17, 23] in *one* server that monitors all the traffic. We propose a solution for the distributed case where there are limited capabilities and resources for monitoring and the general controller is connected to monitors with a limited link. In future work, we will adapt the *two-dimensional* HHHs detection algorithms in [12, 17, 23] for our setting.

Software-defined measurement: The idea of controlling measurement tasks for a single switch using a controller has also been used in [14, 21, 22]. ProgME [22] and [14] use flow-based switches while OpenSketch [21] introduces a hash-based switch architecture. Our work explores the right primitives to support different measurement tasks at various time-scales for different traffic properties and resources at switches.

6. CONCLUSIONS AND FUTURE WORK

We have explored resource/accuracy tradeoffs for global detection tasks and multiple granularities, when different primitives (counting, hashing, programming) are used at individual switches. Our preliminary results indicate that at finer time-scales and with more variability, hashing and programming offer better resource/accuracy tradeoffs.

In the future, we will explore the following directions.

Exploring other primitives: We will explore sampling [2] and SNMP counters, and devise new primitives for programmable switches. For the case of detecting hierarchical heavy hitters, for example, combining sketches and sampling seems promising [13].

Allocating resources among multiple measurement tasks: A motivation for a software-defined measurement system is to serve multiple measurement tasks while choosing the right resource/accuracy tradeoff for them. The resource allocation among these measurement tasks plays an important part in the tradeoff algorithm. For example, under high traffic variability, the controller may decide to assign more resources to the hierarchical heavy hitter detection task (to maintain its accuracy), taking away resources from an invalid traffic detection task.

Extending primitives to multiple switches: In data centers, for example, no single switch sees all the traffic, so some measurement tasks such as detecting hierarchical heavy hitters require composing measurements from multiple switches. For example, using counters, increasing the accuracy in one prefix may need more TCAM entries in multiple switches if its sources are distributed in different racks. For sketch-based switches, we need to implement *composable* sketches to aggregate statistics from multiple switches [18].

7. REFERENCES

[1] CAIDA Anonymized Internet Traces 2012. `http://www.caida.org/data/passive/passive_2012_dataset.xml`.

[2] NetFlow. `http://www.cisco.com/go/netflow/`.

[3] AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., HUANG, N., AND VAHDAT, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI* (2010).

[4] BARKER, K. J., BENNER, A., HOARE, R., HOISIE, A., JONES, A. K., KERBYSON, D. K., LI, D., MELHEM, R., RAJAMONY, R., SCHENFELD, E., SHAO, S., STUNKEL, C., AND WALKER, P. On the Feasibility of Optical Circuit Switching for High Performance Computing Systems. In *Supercomputing* (2005).

[5] BENSON, T., ANAND, A., AKELLA, A., AND ZHANG, M. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *ACM CoNEXT* (2011).

[6] CORMODE, G., AND HADJIELEFTHERIOU, M. Finding Frequent Items in Data Streams. In *VLDB* (2008).

[7] CORMODE, G., KORN, F., MUTHUKRISHNAN, S., AND SRIVASTAVA, D. Finding Hierarchical Heavy Hitters in Data Streams. In *VLDB* (2003).

[8] CORMODE, G., AND MUTHUKRISHNAN, S. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms 55*, 1 (2005).

[9] CURTIS, A., MOGUL, J., TOURRILHES, J., YALAGANDULA, P., SHARMA, P., AND BANERJEE, S. DevoFlow: Scaling Flow Management for High-Performance Networks. In *SIGCOMM* (2011).

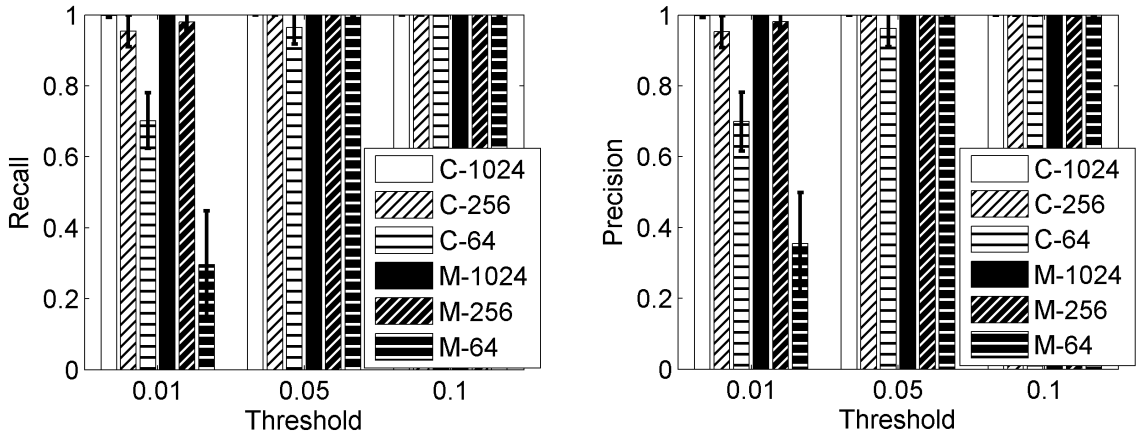
[10] FARRINGTON, N., PORTER, G., RADHAKRISHNAN, S., BAZZAZ, H. H., SUBRAMANYA, V., FAINMAN, Y., PAPEN, G., AND VAHDAT, A. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *SIGCOMM* (2010).

[11] FLAJOLET, P., AND MARTIN, G. N. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences 31*, 2 (1985).

[12] HERSHBERGER, J., SHRIVASTAVA, N., SURI, S., AND TOTH, C. D. Adaptive Spatial Partitioning for Multidimensional Data Streams. *Algorithmica 46*, 1 (2006).

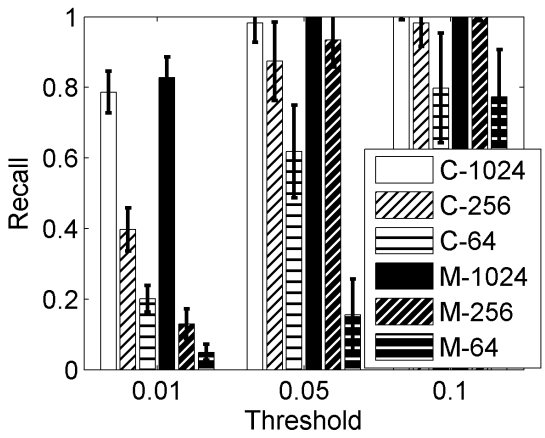
[13] HUANG, G., LALL, A., CHUAH, C.-N., AND XU, J. Uncovering Global Icebergs in Distributed Streams: Results and Implications. *Journal of Network and Systems Management 19*, 1 (2011).

[14] JOSE, L., YU, M., AND REXFORD, J. Online Measurement of Large Traffic Aggregates on Commodity Switches. In *Hot-ICE* (2011).

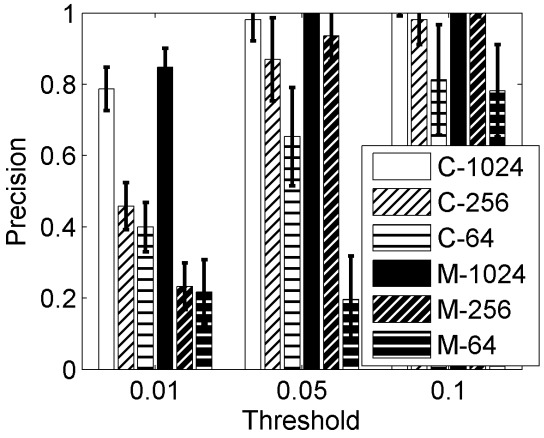


(a) Recall for 5s epoch

(b) Precision for 5s epoch



(c) Recall for 0.5s epoch



(d) Precision for 0.5s epoch

Figure 2: Recall comparison for Count-Min(C) and Max-Cover(M) for limited control link bandwidth(Kbps) and different thresholds

[15] KUMAR, A., SUNG, M., XU, J. J., AND WANG, J. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution. In *SIGMETRICS* (2004).

[16] LIAO, J. SDN System Performance, 2012. `http://pica8.org/blogs/?p=201`.

[17] MITZENMACHER, M., STEINKE, T., AND THALER, J. Hierarchical Heavy Hitters with the Space Saving Algorithm. *arXiv:1102.5540* (2011).

[18] PAPAPETROU, O., GAROFALAKIS, M., AND DELIGIANNAKIS, A. Sketch-based Querying of Distributed Sliding-Window Data Streams. In *VLDB* (2012).

[19] UYEDA, F., FOSCHINI, L., BAKER, F., SURI, S., AND VARGHESE, G. Efficiently Measuring Bandwidth at All Time Scales. In *NSDI* (2011).

[20] VASUDEVAN, V., PHANISHAYEE, A., SHAH, H., KREVAT, E., ANDERSEN, D. G., GANGER, G. R., GIBSON, G. A., AND MUELLER, B. Safe and Effective Fine-Grained TCP Retransmissions for Datacenter Communication. In *SIGCOMM* (2009).

[21] YU, M., LAVANYA, J., AND MIAO, R. Software Defined Traffic Measurement with OpenSketch. In *NSDI* (2013).

[22] YUAN, L., CHUAH, C.-N., AND MOHAPATRA, P. ProgME: Towards Programmable Network MEasurement. *Transactions on Networking 19*, 1 (2011).

[23] ZHANG, Y., SINGH, S., SEN, S., DUFFIELD, N., AND LUND, C. Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Applications. In *IMC* (2004).