DREAM: Dynamic Resource Allocation for Software-defined Measurement

Masoud Moshref† Minlan Yu† Ramesh Govindan† Amin Vahdat*
†University of Southern California *Google and UC San Diego

Today's data center and enterprise networks require expensive capital investments, yet provide surprisingly little visibility into traffic. Traffic measurement can play an important role in these networks, by permitting traffic accounting, traffic engineering, load balancing, and performance diagnosis [1, 3, 4, 5, 2], all of which rely on accurate and timely measurement of time-varying traffic at all switches in the network. Beyond that, tenant services in a multi-tenant cloud may need accurate statistics of their traffic, which requires collecting this information at all relevant switches.

Software-defined measurement [11, 6, 9] has the potential to enable concurrent, dynamically instantiated, *measurement tasks*. In this approach, an SDN controller orchestrates these measurement tasks at multiple spatial and temporal scales, based on a global view of the network. An example of a measurement task is heavy-hitter detection, which attempts to uncover flows whose traffic volume exceeds a specified threshold. In a cloud setting, multiple instances of this task can run concurrently, each measuring heavy hitters on different tenants' traffic.

Unlike prior work [11, 10, 9, 8], which has either assumed specialized hardware support on switches for measurement, or has explored software-defined measurements on hypervisors, our paper focuses on TCAM-based measurement in switches. TCAM-based measurement algorithms can be used to detect heavy hitters and significant changes [9, 12, 7], and these algorithms can leverage existing TCAM hardware on switches and so have the advantage of immediate deployability. However, to be practical, we must address a critical challenge: TCAM resources on switches are fundamentally limited for power and cost reasons. Unfortunately, measurement tasks may require multiple TCAM counters, and the number of allocated counters can determine the accuracy of these tasks. Furthermore, the resources required for accurate measurement may change with traffic, and tasks may require TCAM counters allocated on multiple switches.

Contributions. In this poster, we discuss the design of a system for TCAM-based software-defined measurement, called DREAM. Users of DREAM can dynamically instantiate multiple concurrent measurement tasks (such as heavy hitter or hierarchical heavy hitter detection, or change detection) at an SDN controller, and additionally specify a *flow filter* (e.g., defined over 5-tuples) over which this measurement task is executed. Since the traffic for each task may need to be measured at multiple switches, DREAM needs to allocate switch resources to each task.

To do this, DREAM first leverages two important observations. First, although tasks become more accurate with more TCAM resources, there is a point of diminishing returns:

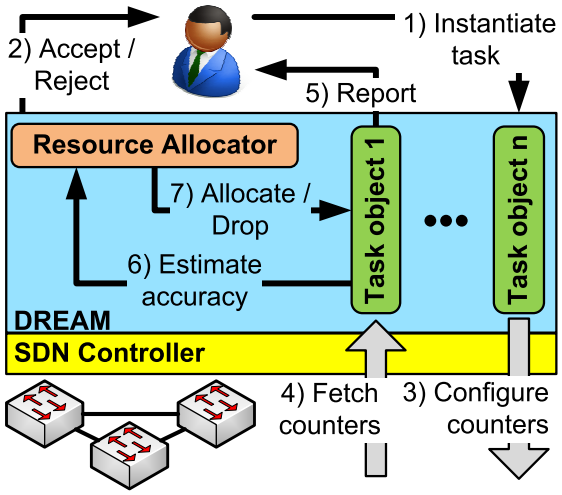
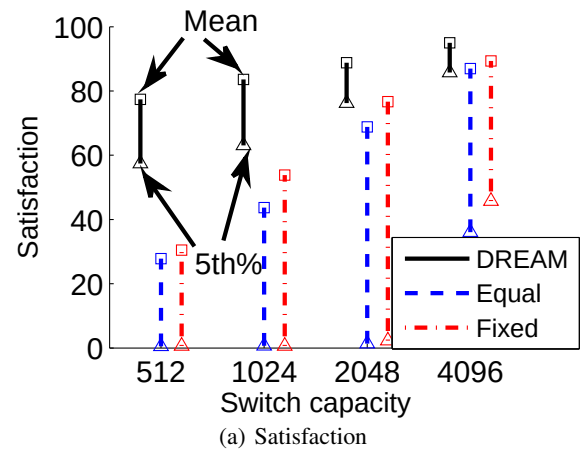


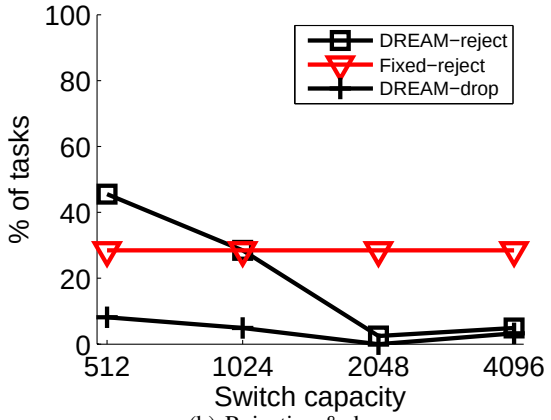
Figure 1: DREAM system overview

beyond a certain accuracy, significantly more resources are necessary to increase the accuracy of the task. Moreover, beyond this point, the *quality* of the retrieved results, say heavy hitters is marginal (as we quantify later). This suggests that it would be acceptable to maintain the accuracy of measurement tasks above a high (but below 100%) user-specified *accuracy bound*. Second, tasks need TCAM resources only on switches at which there is traffic that matches the specified flow filter, and the number of resources required depends upon the traffic volume and the distribution. This suggests that allocating just enough resources to tasks at switches and over time might provide *spatial and temporal statistical multiplexing benefits*.

DREAM uses both of these observations to permit more concurrent tasks than is possible with a static allocation of TCAM resources. To do this, DREAM needs to estimate the TCAM resources required to achieve the desired accuracy bound. Unfortunately, the relationship between resource and accuracy for measurement tasks cannot be characterized *a priori* because it depends upon the size of the wildcard flow and traffic characteristics. If this relationship could have been characterized, an optimization-based approach would have worked. Instead, DREAM contains a novel resource adaptation strategy for determining the right set of resources assigned to each task at each switch. This requires measurement algorithm-specific estimation of task accuracy, for which we have designed *accuracy estimators* for several common measurement algorithms. Using these, DREAM increases the resource allocated to a task at a switch when its overall (global) estimated accuracy is below the accuracy bound and its accuracy at the switch is also below the accuracy bound. In this manner, DREAM decouples resource allocation, which



(a) Satisfaction



(b) Rejection & drop

Figure 2: Prototype result for combined workload

is performed locally, from accuracy estimation, which is performed globally. DREAM continuously adapts the resources allocated to tasks, since a task's accuracy and resource requirements can change with traffic. Finally, if DREAM is unable to get enough resources to a task to satisfy its accuracy bound, it *drops* the task.

Figure 1 shows the DREAM workflow, illustrating both the interface to DREAM and the salient aspects of its internal operation. A user instantiates a task and specifies its parameters (step 1). Then, DREAM decides to accept or reject the task based on the available resources (step 2). For each accepted task, DREAM initially configures a default number of counters at one or more switches (step 3). DREAM also creates a *task object* for each accepted task: this object encapsulates the resource allocation algorithms run by DREAM for each task. Periodically, DREAM retrieves counters from switches and passes these to task objects (step 4). Task objects compute measurement results and report the results to users (step 5). In addition, each task object contains an *accuracy estimator* that measures current task accuracy (step 6). This estimate serves as input to the *resource allocator* component of DREAM, which determines the number of TCAM counters to allocate to each task and conveys this number to the corresponding task object (step 7). The task object determines how to use the allocated counters to measure traffic, and may re-configure one or more switches (step 3). If a task is dropped for lack of resources, DREAM removes its task object and de-allocates the task's TCAM counters.

DREAM is at a novel point in the design space: it permits multiple concurrent measurements without compromising their accuracy, and effectively maximizes resource usage. We demonstrate through extensive experiments on a DREAM prototype (in which multiple concurrent tasks three different types are executed) that it performs significantly better than other alternatives, especially at the tail of important performance metrics, and that these performance advantages carry over to larger scales evaluated through simulation. Figure 2 presents the performance of DREAM when we vary the capacity of switches to change loads. DREAM's satisfaction metric (the fraction of task's lifetime that its accuracy is above the bound) is 2× better at the tail for moderate loads than an approach which allocates equal share of resources to tasks: in DREAM, almost 95% of the tasks have a satisfaction higher than 80%, but for equal allocation, 5% have a satisfaction less than 40%. At high loads, DREAM's average satisfaction is nearly 3× that of equal allocation. Figure 2(a) presents satisfaction metric where the mean value is the upper end of each vertical bar, and the 5^{th} percentile is the lower end. Some of these relative performance advantages also apply to an approach which allocates a fixed amount of resource to each task, but rejects tasks that cannot be satisfied. However, this fixed allocation rejects an unacceptably high fraction of tasks: even at low load, it rejects 30% of tasks, while DREAM rejects 5% (Figure 2(b)). Finally, these performance differences persist across a broad range of parameter settings.

1. REFERENCES

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.

[2] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pFabric: Minimal Near-optimal Datacenter Transport. In *SIGCOMM*, 2013.

[3] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *ACM CoNEXT*, 2011.

[4] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *WREN*, 2009.

[5] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling Flow Management for High-Performance Networks. In *SIGCOMM*, 2011.

[6] L. Jose, M. Yu, and J. Rexford. Online Measurement of Large Traffic Aggregates on Commodity Switches. In *Hot-ICE*, 2011.

[7] F. Khan, N. Hosein, C.-N. Chuah, and S. Ghiasi. Streaming Solutions for Fine-Grained Network Traffic Measurements and Analysis. In *ANCS*, 2011.

[8] S. Meng, A. K. Iyengar, I. M. Rouvellou, and L. Liu. Volley: Violation Likelihood Based State Monitoring for Dataceners. *ICDCS*, 2013.

[9] M. Moshref, M. Yu, and R. Govindan. Resource/Accuracy Tradeoffs in Software-Defined Measurement. In *HotSDN*, 2013.

[10] V. Sekar, M. K. Reiter, and H. Zhang. Revisiting the Case for a Minimalist Approach for Network Flow Monitoring. In *IMC*, 2010.

[11] M. Yu, L. Jose, and R. Miao. Software Defined Traffic Measurement with OpenSketch. In *NSDI*, 2013.

[12] Y. Zhang. An Adaptive Flow Counting Method for Anomaly Detection in SDN. In *CoNEXT*, 2013.