Revisiting the Case for a Minimalist Approach for Network Flow Monitoring

Vyas Sekar
Carnegie Mellon University
Pittsburgh, PA
vyass@cs.cmu.edu

Michael K Reiter
UNC Chapel Hill
Chapel Hill, NC
reiter@cs.unc.edu

Hui Zhang
Carnegie Mellon University
Pittsburgh, PA
hzhang@cs.cmu.edu

ABSTRACT

Network management applications require accurate estimates of a wide range of flow-level traffic metrics. Given the inadequacy of current packet-sampling-based solutions, several application-specific monitoring algorithms have emerged. While these provide better accuracy for the specific applications they target, they increase router complexity and require vendors to commit to hardware primitives without knowing how useful they will be to meet the needs of future applications.

In this paper, we show using trace-driven evaluations that such complexity and early commitment may not be necessary. We revisit the case for a "minimalist" approach in which a small number of simple yet generic router primitives collect flow-level data from which different traffic metrics can be estimated. We demonstrate the feasibility and promise of such a minimalist approach using flow sampling and sample-and-hold as sampling primitives and configuring these in a network-wide coordinated fashion using cSamp. We show that this proposal yields better accuracy across a collection of application-level metrics than dividing the same memory resources across metric-specific algorithms. Moreover, because a minimalist approach enables *late binding* to what application-level metrics are important, it better insulates router implementations and deployments from changing monitoring needs.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*network monitoring, network management*

General Terms

Measurement, Management

Keywords

Traffic Monitoring, Sampling, Data Streaming, Anomaly Detection

IMC'10, November 1–3, 2010, Melbourne, Australia.

1. INTRODUCTION

Flow monitoring supports vital network management tasks such as traffic engineering [18], anomaly detection [28, 29], accounting [14, 17], identifying and analyzing end-user applications [12, 22], understanding traffic structure [43], detecting worms, scans, and botnet activities [44, 41, 35], and forensic analysis [42]. These require accurate estimates of different traffic metrics relevant to each application.

High traffic rates exceed the monitoring capabilities of routers,[1] and since traffic is scaling at least as fast as routers' capabilities, some form of sampling or data reduction is necessary in commodity solutions. (There are high-end solutions for full packet capture [3]. These are expensive and require specialized instrumentation.) The de-facto standard is NetFlow [11, 2] which uses packet sampling. Each packet is sampled with some probability and the selected packets are aggregated into flows.[2] NetFlow-style monitoring is sufficient for applications such as traffic volume estimation that require only a coarse view of traffic, but several studies have shown the inadequacy of packet sampling for many of the fine-grained monitoring applications mentioned earlier (e.g., see [34, 21, 15, 26, 7, 35, 17]).

A consequence of these results is that several research efforts have focused on developing application-specific monitoring techniques. This is exemplified by the proliferation of data streaming algorithms for computing specific traffic metrics, e.g., computing the flow size distribution [26], entropy estimation [30], super-spreader detection [41], degree histogram estimation [44], change detection [25], and so on.

While this body of work has made valuable algorithmic contributions, this shift to application-specific approaches is undesirable for two practical reasons. First, having many application-specific proposals increases the implementation complexity and possibly the resource requirements of routers. Second, the set of applications is a moving target, as both normal and anomalous traffic patterns change over time. This requires router vendors and network managers to commit to a fixed set of application-level metrics without knowing if these will meet future requirements.

In this work, we reflect on these developments and ask a fundamental question:

Is such complexity and early commitment necessary?
Are there simpler alternatives that can provide the requisite fidelity and generality?

[1] Our arguments apply to non-router-based monitoring solutions as well.

[2] A flow is a sequence of packets with the same IP 5-tuple ⟨ srcip, dstip, srcport, dstport, protocol ⟩.

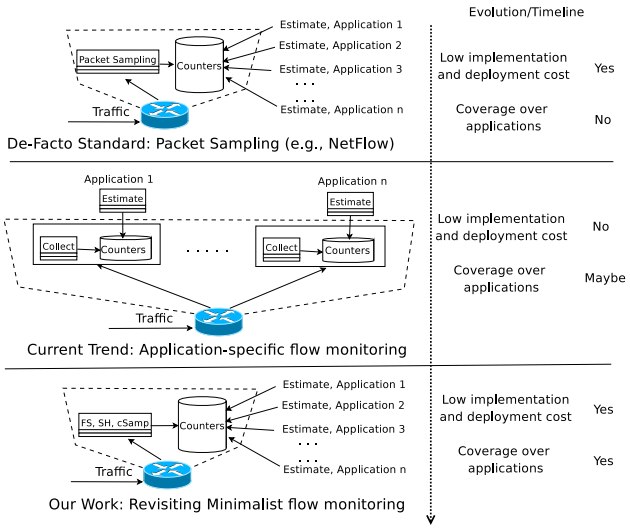


Figure 1: A minimalist approach runs a few collection algorithms. Applications can use the collected data later (possibly offline). NetFlow/packet sampling is a minimalist approach, but it is not well-suited for many applications. An application-specific architecture implements many focused algorithms. These work well for the specific applications, but increase complexity and are not robust to changing demands. We demonstrate a minimalist alternative that performs favorably compared to application-specific approaches over a wide spectrum of applications.

Approach and Intuition: We revisit the case for a *minimalist* approach that retains the simplicity of NetFlow, where routers only need to support a *few* monitoring primitives, but still provide coverage over a wide spectrum of applications.

To understand how we can achieve this, we can think of each monitoring application as being composed of two logical phases: (1) a *collection* phase that needs to operate at line rates and (2) an *estimation* phase to compute different traffic metrics that need not strictly work at line rates. Application-specific alternatives tightly couple these two components, only retaining counters and statistics relevant to a specific application context (Figure 1). In contrast, we envision a minimalist approach that *decouples* the collection and estimation phases as much as possible.

A key question is whether such an approach can provide estimation accuracy comparable to application-specific alternatives. One rationale to suggest that it can, is that the primary bottleneck for monitoring is keeping counters in fast memory (SRAM). Instead of splitting the available memory across different applications, we can aggregate it, and run a few simple primitives with high-enough sampling rates to obtain accurate estimates of traffic metrics for a wide spectrum of applications. In other words, when we look at each application in isolation, application-specific strategies are appealing. However, when we consider a portfolio of applications in aggregate, a minimalist approach might be a better alternative.

Contributions and Implications: Our goal is not to design an optimal minimalist approach. Rather, our objective is to establish a *feasible* instance.

We present a practical minimalist approach in Section 4 that combines sample-and-hold [17], flow sampling [21], and cSamp [39]. Our choice of these specific primitives is guided by the understanding that monitoring applications fall into two broad classes that analyze (1) *volume structure* (e.g., traffic engineering) or (2) *communication structure* (e.g., security applications). Flow sampling is ideally suited for the latter class [21, 34, 32] and sample-and-hold for the former [17]. cSamp provides a framework to efficiently leverage the available monitoring resources at routers to meet network-wide monitoring goals.

We use trace-driven analysis to evaluate this design against several application-specific approaches (Section 6): detecting heavy hitters [17], superspreaders [41], and large traffic changes [25]; computing entropy [30] and the outdegree histogram [44]; and estimating the flow size distribution [26]. When our approach has the same total memory resources as that used by the different application-specific algorithms in aggregate, it provides comparable or better estimation accuracy across the entire spectrum of applications. Moreover, by delaying the binding to specific applications, it enables computation of not-yet-conceived measures that will be interesting in the future.

Our work shows the promise of a minimalist approach even with a simple combination of existing techniques. We believe that this has significant implications for router vendors, network operators, and measurement researchers. First, it can reduce router complexity without compromising a vendor's ability to satisfy its customers' demands. Second, it helps insulate network deployments from the changing needs of monitoring applications. Finally, we hope that these results motivate further research in developing better minimalist primitives and estimation algorithms, and in understanding their fidelity for different applications.

2. BACKGROUND AND RELATED WORK

Packet sampling: Router vendors today use uniform packet sampling [11]: a router selects a subset of packets, and aggregates the sampled packets into flow reports. However, packet sampling has inherent limitations. There are known biases toward sampling larger flows (e.g., [21, 26, 34]) and several studies have questioned its accuracy for many management applications (e.g., see [34, 21, 15, 26, 7, 35, 17]).

Application-specific approaches: The limitations of packet sampling have motivated many application-specific data streaming algorithms. The high-level approach is to use a small number of SRAM counters that track traffic statistics pertinent to each application and then estimate the relevant application-level metrics from these counters. These include algorithms for estimating the flow size distribution [26, 36], identifying heavy hitters [17], entropy estimation [30], superspreader detection [41], degree histogram estimation [44], and change detection [25]. However, these approaches are tightly coupled to the specific applications and report summary statistics only relevant to these applications. Thus, it is difficult to estimate other measures of interest from these reports. Therefore, these lack the generality to serve as minimalist primitives.

Some data structures (e.g., sketches [13]) provide more generality. However, these have two limitations. First, they are designed primarily for coarse *volume queries* and less suited for fine-grained tasks like entropy estimation and superspreader detection. Second, sketches operate with a specific "flowkey" over one or more fields of the IP 5-tuple (srcip, dstip, srcport, dstport, protocol). Each flowkey of interest requires a separate instance. However, it is often necessary to analyze combinations of two or more fields for diagnostic purposes (e.g., for investigating anomalies). Having a separate instance for each combination incurs high overhead. Furthermore, this needs prior knowledge of which flowkeys will be useful, which may not be known until after the operator begins to investigate specific events.

Selective sampling: Some approaches assign different sampling rates for different classes of packets [27, 35]. Others only log flows with pre-specified patterns (e.g., [45, 1, 4, 33, 8]). While these approaches provide some flexibility, they need to know the specific classes and sampling rates to meet the applications' requirements. In contrast, we envision a minimalist approach that is largely agnostic to the specific types of analyses that may be performed.

Network-wide measurements: Many studies have stressed the importance of network-wide measurements to meet operational requirements as applications and attacks become more distributed [18, 28, 29]. For example, understanding peer-to-peer traffic [12], detecting botnets [35] and hit-list worms [32], understanding DDoS attacks [38], and network forensics [42] inherently require a network-wide view aggregated from multiple vantage points. In this respect, recent proposals show the benefits of moving beyond router-centric solutions to network-wide monitoring solutions [9, 39].

3. DESIGN CONSIDERATIONS

Given this background, we synthesize key requirements for a flow monitoring architecture and derive guiding principles for a minimalist approach, echoing the charter of the IETF PSAMP working group [5].

3.1 Requirements

Minimize router complexity: Given the hardware and development costs involved in modern router design, we want to keep router implementations as simple as possible.

Generalize to many applications: The monitoring infrastructure should cover a wide spectrum of applications and ideally be robust to future application needs.

Enable diagnostics: The monitoring architecture should support diagnostic "drill-down" tasks; e.g., by providing the capability to give different views into traffic structure.

Provide network-wide views: The monitoring architecture should provide network-wide capabilities as these are increasingly crucial for several aspects of network management and traffic analysis.

3.2 Design Principles

A few, simple, and generic primitives: A natural way to reduce router complexity is to have a few primitives that are easy to implement but powerful enough to support many management tasks.

Decouple collection and computation: Now, how can we provide generality and support diagnostics with a few monitoring primitives? We believe that this best achieved by *decoupling* the collection and computation phases involved in the monitoring tasks. Note that this is already implicit in network operations today: routers export NetFlow reports to a (logically) central collector and operators analyze this data. We retain this operational model; routers run some collection algorithms and export the collected flow reports. Once we have the flow-level reports, we can compute any traffic metric of interest and provide different views required for further diagnosis.

Network-wide resource management: To provide network-wide capabilities, we need a framework that assigns monitoring responsibilities across routers to satisfy network-wide monitoring goals. At the same time, this framework should be *resource-aware* and respect the constraints (e.g., SRAM capacity) of individual routers.

3.3 Challenges

Given the above considerations, two questions remain:

1. **Concrete Design:** What primitives should be implemented on routers to support a range of applications? How should monitoring responsibilities be assigned to meet network-wide measurement goals?

2. **Performance:** Does the intuitive appeal of a minimalist approach translate into quantitative benefits for a wide spectrum of applications?

In addressing these challenges, our goal is not to look for an "optimal" minimalist approach. (In fact, it is not clear if we can formally reason about optimality without committing to a fixed set of applications.) Rather, we want to look for a *feasible* instance that covers a broad spectrum of applications. We present one such proposal in the next section.

4. ARCHITECTURE

The first challenge above requires that we choose a small set of generic collection primitives that runs on each router and design a framework to manage them intelligently across a network of routers. Our specific proposal combines three ideas: flow sampling [21] and sample-and-hold [17] as single router sampling algorithms, and cSamp [39] for network-wide management. Keys et al. designed a system for providing traffic summaries and detecting "resource hogs" [23] using a combination of flow sampling and sample-and-hold, similar to our approach. We extend their work in two significant ways. First, we show how to combine these primitives with the network-wide capabilities of cSamp [39] in contrast to the single-vantage-point view in their work. Second, we look beyond simple traffic summaries and demonstrate that this combination can support a much wider range of applications.

4.1 Router Primitives

Choice of primitives: Flow monitoring applications can be divided into two broad classes: (1) those that require an understanding of *volume structure*; e.g., heavy-hitter detection and traffic engineering that require an understanding of the number of packets/bytes per-port or per-src and (2) those that depend on the *communication structure*; e.g., security applications and anomaly detection application that require an understanding of "who-talks-to-whom". Our choice of primitives is guided by these two broad classes. Flow sampling is well suited for security and anomaly detection applications that analyze communication structure [21, 34, 32]. Similarly, sample-and-hold is well suited for traffic engineering and accounting applications that analyze volume structure [17]. Thus, these two primitives effectively complement each other in their capabilities. We do note that there are other proposals for capturing the communication structure (e.g., [35, 27]) and volume structure (e.g., [13, 15, 24]), and flow sampling and sample-and-hold may not necessarily be the optimal primitives. Our goal is to pick a feasible point in this design space and quantitatively compare it to application-specific approaches.

For the following discussion, a flow is defined by the 5-tuple: $\langle$srcip, dstip, srcport, dstport, protocol$\rangle$. We use flow sampling and sample-and-hold at this 5-tuple granularity. The collected flows can be sliced-and-diced after the fact by projecting from this general definition to others (e.g., per destination port, per source address).

Sample-and-Hold (SH): Sample-and-hold (SH) [17] keeps near-exact counts of "heavy hitters" — flows with high packet counts. SH works as follows. For each packet, the router checks if it is

tracking this packet's *flowkey*, defined over one or more fields of the IP 5-tuple. If yes, the router updates that counter. If not, the flowkey for this packet is selected with probability p, and the router keeps an exact count for this selected flowkey subsequently. Since this requires per-packet counter updates, the counters are kept in SRAM [17].

To configure SH, we specify the flowkey(s) (e.g., srcport, srcip, or 5-tuple), the anticipated total number of packets for a specific time interval ($numpkts$), and the number of flows that can be logged (L) depending on the SRAM constraint. To ensure that the number of flow entries created does not exceed the SRAM capacity, we use $numpkts$ in configuring SH to set the packet sampling probability $p = \frac{L}{numpkts}$.[3] In our minimalist design, we use one instance of SH and configure it to operate at the 5-tuple granularity.

Hash-based flow sampling (FS): Flow sampling (FS) picks flows rather than packets at random [21]. One way to implement FS is as follows. Each router has a *sampling manifest* — a table of one or more hash ranges indexed using a key derived from each packet header. On receiving a packet, the router computes the hash of the packet's 5-tuple (i.e., the flowkey). Next, it selects the appropriate hash range from the manifest and selects the flow if the hash falls within this range. If the flow is selected, then the router uses its hash as an index into a table of flows and updates the byte and packet counters for the flow. The hash function maps the input 5-tuple uniformly into the interval $[0, 1]$. Thus, the size of each hash range determines the flow sampling rate for each category of flows in the manifest.

Similar to SH, FS requires per-packet table lookups; the flow table must therefore be implemented in SRAM. It is possible to add a packet sampling stage to make DRAM implementations possible [24]. For simplicity, we assume that the counters are stored in SRAM.

4.2 Resource Management

Having chosen FS and SH as our minimalist primitives, we address the following question. Given a fixed amount of SRAM available for monitoring on each router, how should we divide it between these primitives?

Combining FS-SH on a single router: Consider a single router with a fixed amount of SRAM that can hold L flow counters. A simple way to split L is to give a fraction f to FS and the remaining $1 - f$ to SH. We show in Section 6 that $f \approx 0.8$ is a good choice.

Network-wide case: The above split works for the single router case. Next, we see how we can manage the monitoring resources across a network of routers. Network-wide management tasks are typically specified in terms of Origin-Destination pairs, specified by an ingress and egress router (or PoP). OD-pairs are convenient abstractions that naturally fit many of the objectives (e.g., traffic engineering) and constraints (e.g., routing paths, traffic matrix) in network management. A natural extension of the single router hybrid primitive to the network-wide case is to consider the resource split per OD-pair [9, 39].

Here, we observe a key difference between FS and SH. It is possible to coordinate FS instances by assigning non-overlapping responsibilities across routers on a path [39]. However, because SH logs heavy hitters, the same set of heavy hitters will be reported across routers on a path. Thus, replicating SH across routers on a path duplicates measurements and wastes router resources.

To address this issue, we make a distinction between ingress and non-ingress routers. Ingresses implement both FS and SH, sharing the aggregate memory as in the single router case. At each such ingress router, the SH resources are split between the OD-pairs originating at the ingress, in proportion to the anticipated number of packets per OD-pair.[4] Non-ingress routers only implement FS. In order to distribute FS responsibilities across the network, we use cSamp [39], which we describe next.

Overview of cSamp: We choose cSamp because, for a given set of router resource constraints, it provides a framework to optimize fine-grained network-wide monitoring goals; it leverages the available monitoring capacity efficiently by avoiding redundant measurements; and it naturally load balances responsibilities to avoid hotspots.

The inputs to cSamp are the flow-level traffic matrix (approximate number of flows per OD-pair), router-level path(s) for each OD-pair, the resource constraints of routers, and an ISP's objective function specified in terms of the fractional flow coverages per OD-pair (i.e., the fraction of flows on this OD-pair that are logged). These input parameters are typically available to network operators [18]. The output is a set of *sampling manifests* specifying the monitoring responsibility of each router in the network. Each sampling manifest contains entries of the form $\langle OD, [start, end]\rangle$, where $[start, end] \subseteq [0, 1]$ denotes a hash range and OD is an identifier for an OD-pair. In the context of the FS algorithm, this means that the OD-pair identifier is used as the "key" to get a hash range from the sampling manifest.[5]

The main idea is to bootstrap routers with the same hash function but assign *non-overlapping* hash ranges per OD-pair. Thus, the flows sampled by different routers do not overlap. This coordination also makes it possible to optimally achieve network-wide flow coverage goals. Next, we describe the optimization model used in cSamp to assign FS responsibilities across a network.

Each OD-Pair OD_i ($i = 1, \ldots, M$) is characterized by its router-level path P_i and the estimated number T_i of IP-level flows per measurement epoch (e.g., five minutes).[6] Each router R_j is constrained by the available *memory* for maintaining flow counters. L_j captures this constraint, and denotes the number of flows router R_j can record and report per epoch. d_{ij} denotes the fraction of flows of OD_i that router R_j logs. For $i = 1, \ldots, M$, let C_i denote the fraction of flows on OD_i that is logged.

cSamp can support a variety of network-wide objectives. Here we describe its use for one particular goal: achieving the best flow coverage subject to maximizing the minimum fractional flow coverage per OD-pair. First, the largest possible minimum fractional coverage per OD-pair, $\min_i\{C_i\}$, subject to the resource constraints is found. Next, this value is used as the parameter α to the linear program shown below (in Eq (1)–(4)) and the total flow coverage $\sum_i(T_i \times C_i)$ is maximized, i.e.,

[3] Estan and Varghese use SH to track heavy hitters who contribute more than a fraction $\frac{1}{x}$ to the total traffic volume. In their proposal, p is set to $\frac{O \times x}{numpkts}$, where O is an oversampling factor [17]. Our configuration can be viewed as determining x and O from the memory budget L.

[4] We use packets and not flows since the SH configuration depends on the number of packets in the traffic. In practice, we need only approximate estimates to divide the available memory resources.

[5] This formulation assumes that routers in the middle of the network can infer the OD-pair from packet headers using MPLS labels or ingress-egress prefix maps [6]. cSamp can also be implemented without OD-pair identifiers [40]. For clarity, we describe the simpler approach using OD-pairs.

[6] For simplicity, we assume that each OD-pair has one route, though cSamp accommodates multi-path routing [39].

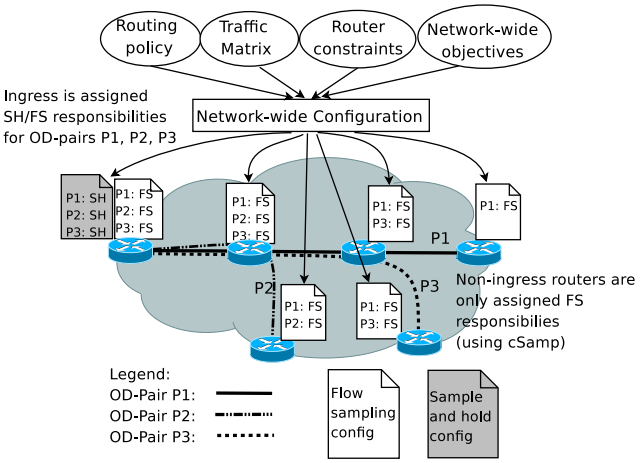


Figure 2: Overview of our network-wide approach

$$\text{Maximize} \sum_i (T_i \times C_i), \text{ subject to}$$

$$\forall j, \quad \sum_{i:R_j \in P_i} (d_{ij} \times T_i) \leq L_j \tag{1}$$

$$\forall i, \quad C_i = \sum_{j:R_j \in P_i} d_{ij} \tag{2}$$

$$\forall i, \forall j, \quad d_{ij} \geq 0 \tag{3}$$

$$\forall i, \quad \alpha \leq C_i \leq 1 \tag{4}$$

The rationale behind this objective is as follows. Maximizing the minimum coverage provides *fairness* in allocating resources across OD-pairs. Since it is hard to ascertain which OD-pairs might show interesting traffic patterns, allocating resources fairly is a reasonable choice. Given a fair allocation, we use the residual resources *efficiently* to achieve maximum aggregate coverage. The optimal solution $d^* = \{d^*_{ij}\}$ to this optimization problem is then translated into the *sampling manifests* specifying the FS responsibilities for each router.

Example configuration: Figure 2 shows how the different components are combined in the network-wide case. There are three OD-pairs P1, P2, and P3 originating at the left-most router. We envision a configuration module at the network operations center which disseminates configurations to the routers. This module takes into account the prevailing network conditions, routing policies, router constraints, and the flow monitoring objectives to generate the FS and SH configurations for each router. In the example, the ingress router is assigned SH responsibilities for P1, P2, and P3. The non-ingress routers are not assigned any SH responsibilities for these OD-pairs. (The other edge routers could be assigned SH responsibilities for OD-pairs for which they are the origin, but these are not shown.) The FS responsibilities are generated using cSamp. Each router is assigned FS responsibilities only for the paths of OD-pairs on which it lies, and these are specified as non-overlapping hash ranges per OD-pair.

5. EVALUATION METHODOLOGY

Our goal is to compare the minimalist design from the previous section against an application-specific architecture when both approaches are given the same total resource budget. In order to do so, we need to specify the different applications of interest, the corresponding application-specific algorithms, and the configurations for determining the resources provisioned for each algorithm.

First, we describe the different applications, the corresponding data streaming algorithms, and accuracy metrics in Section 5.1.

Application	Accuracy/Error Metric	Algorithm	Parameters (defaults)
FSD estimation (5-tuple)	WMRD	[26]	fsd (0.7)
Heavy hitter detection (5-tuple,sip,dip, sport,dport sip-dip)	Top-k detection rate	[17]	hh, k (0.3, 50)
Entropy estimation (5-tuple,sip, dip,sport,dport)	Relative Error	[30]	ϵ, δ (0.5, 0.5)
Superspreader detection	Detection accuracy	[41]	K, b, δ (100, 4, 0.5)
Change detection (sip,dip)	falsepos + falseneg	[25]	h, q, θ (10, 1024, 0.05)
Deg. histogram estimation	JS-divergence	[44]	–

Table 1: Summary of applications, accuracy metrics, algorithms, and default parameters. The parentheses in the first column specify the flowkey(s) for the application (e.g., FSD uses 5-tuple; heavy-hitter has six flowkeys). fsd and hh are expressed as a fraction of the number of distinct IP flows per epoch. ϵ, δ denote error tolerances. K, b means that any IP contacting $\geq K$ distinct IPs is a superspreader and any IP contacting $\leq \frac{K}{b}$ distinct destinations is a false positive. h is the number of hash functions and q is the number of counters per hash function in the sketch data structure, and θ is the change detection threshold.

Then, in Section 5.2, we describe how we normalize the resource usage of the minimalist and application-specific algorithms. We explain our assumptions and justify why these are conservative in that we underestimate the performance of an equivalently provisioned minimalist approach. Finally, in Section 5.3, we describe the configuration parameters for the different algorithms and the estimation phase for the minimalist approach in Section 5.4.

5.1 Applications and Accuracy Metrics

We pick a set of diverse monitoring applications that span the spectrum of traffic engineering, security, and anomaly detection tasks of interest to network operators. Table 1 summarizes the applications and the corresponding application-specific algorithms, accuracy metrics, and configuration parameters. The table also shows the default parameters we use in each case.

Flow size distribution (FSD) estimation: Let F denote the total number of flows in a traffic stream and F_l be the number of flows of size l pkts per flow. The FSD estimation problem is to determine $\forall l = 1 \ldots z, \phi_l = \frac{F_l}{F}$, where z is the largest flow size. Understanding the FSD is useful for many management tasks such as estimating gains from caches, configuring flow-switched networks, attack detection, and traffic matrix estimation [15, 26]. We use the data streaming and expectation-maximization algorithm proposed by Kumar et al. [26]. (We choose this because Kumar et al. show that their approach is significantly better than prior approaches based purely on packet sampling [15].)

The accuracy metric for FSD estimation is the weighted mean relative difference (*WMRD*) between the true FSD $\{F_l\}$ and the estimated FSD $\{\hat{F}_l\}$ [26]. The WMRD is defined as $\frac{\sum_l |F_l - \hat{F}_l|}{\sum_l \frac{F_l + \hat{F}_l}{2}}$.

Heavy-hitter detection: The goal here is to identify the top k items (e.g., srcip, srcport) with the most traffic volume. These are used by operators to understand application patterns and resource hogs, as well as for traffic engineering and accounting.

We use the SH algorithm [17] described earlier. We configure it to run with six instances, one each for the following flowkeys: source port, destination port, source address, destination address, 5-tuple, and source-destination address pairs. The accuracy metric is the *top-k detection rate* – the set intersection between the exact top-k and estimated top-k heavy hitters. Our minimalist approach also uses SH; the main difference is that we use only one instance of SH that runs at the 5-tuple granularity and use offline projections to determine heavy hitters for the other flowkeys.

Entropy estimation: The entropy of traffic distributions (e.g., distribution of pkts per dstport) is useful for anomaly detection [29] and traffic classification [43]. In particular, entropy-based analysis captures fine-grained properties that cannot be obtained with just volume-based analysis. The entropy of a random variable X is $H(X) = -\sum_{i=1}^{N} Pr(x_i) \log_2 (Pr(x_i))$, where $x_1, \ldots, x_N$ is the range of values for X, and $Pr(x_i)$ is the probability that X takes the value x_i. It is useful to normalize the entropy between zero and one as $H_{norm}(X) = \frac{H(X)}{\log_2(N_0)}$, where N_0 is the number of distinct x_i values observed in a given measurement epoch [29].

We use the data streaming algorithm proposed by Lall et al. [30]. We consider five distributions: 5-tuple, src port, dst port, src address, and dst address. The accuracy metric is *relative error* — if the actual value is H_{norm} and the estimated value is $\hat{H}_{norm}$, the relative error is $\frac{|H_{norm} - \hat{H}_{norm}|}{H_{norm}}$.

Superspreader detection: Security applications like scan, worm, and botnet detection need to detect "superspreaders" – source IPs that contact a large number of distinct destination IPs. Note that this is different from heavy-hitter detection; we want to find sources talking to many *unique* destinations rather than sources generating a large volume of traffic.

We use the one-level superspreader detection algorithm proposed by Venkataraman et al. [41]. The algorithm has three parameters K, b, and δ; the goal is to detect all hosts that contact $\geq K$ distinct destinations with probability $\geq 1 - \delta$, and guarantee that a source that contacts $\leq \frac{K}{b}$ distinct destinations is reported with probability $\leq \delta$. The accuracy metric is the *detection accuracy*: the number of true superspreaders detected. (For brevity, we do not report the false positive rate since it was zero for the minimalist and application-specific approaches in almost all cases.)

Change detection: Change detection is used to detect DDoS attacks, flash crowds, and worms [25]. At a high-level, the goal is to detect IP addresses or ports whose behavior deviates significantly from some expected behavior based on a history-based forecast model. The problem can be formally described as follows.

Suppose we bin a traffic stream into measurement epochs ($t = 1, 2, \ldots$). Let $I_t = \beta_1, \beta_2, \ldots$ be the input traffic stream for epoch t. Each packet β_i is associated with a flowkey y_i and a count c_i (e.g., #bytes or just 1 if we are counting packets). $Obs_y(t) = \sum_{i:y_i=y} c_i$ denotes the aggregate count for flowkey y in epoch t. Let $Fcast_y(t)$ denote the forecast value (e.g., using exponentially weighted moving average, EWMA) for item y in epoch t. The forecast error for y then is $Err_y(t) = Obs_y(t) - Fcast_y(t)$. $F2Err_t = \sum_y Err_y(t)^2$ is the second moment of the forecast errors. The goal is to detect all ys with $Err_y(t) \geq \theta \times \sqrt{F2Err_t}$, where θ is a user-defined threshold. We define the *change detection accuracy* as the sum of the false positive (flowkeys whose volume did not change significantly but were incorrectly reported) and false negative rates (flowkeys that changed but were not reported).

We use the sketch-based change detection algorithm proposed by Krishnamurthy et al. [25] as sketches have a natural "linearity" property that makes them well-suited for change detection. We use an EWMA model $Fcast(t) = \gamma Obs(t) + (1 - \gamma) Fcast(t - 1)$, with $\gamma = 0.9$. Note that since we are only interested in the relative performance of the minimalist vs. sketch-based approaches, the specific forecast model we use is not important. We consider two instances to identify changes in (1) the number of packets per source address and (2) the number of packets per destination address.

Degree histogram estimation: The outdegree d of a source IP is the number of distinct IPs it contacts in a measurement epoch. We construct the degree histogram as follows. For bucket i, let m_i denote the number of sources with outdegree d such that $2^i \leq d \leq 2^{i+1} - 1$. The goal is to estimate these m_i values. A specific application is to detect botnets involved in coordinated scans [44] by detecting changes in the outdegree histogram. The outdegree distribution is independently useful for understanding traffic structure. We use the sampling algorithm proposed by Gao et al. [44]. Given the exact distribution $\{m_1, m_2, \ldots\}$ and an estimated distribution $\{\hat{m}_1, \hat{m}_2, \ldots\}$, we use the *Jensen-Shannon (JS) divergence* between the two distributions as the accuracy metric.[7]

5.2 Assumptions and Justification

In order to compare the minimalist and application-specific approaches, we need to normalize their total resource footprints. We discuss our assumptions along three dimensions: hardware implementation, processing requirements, and memory use. We justify why our specific assumptions are *conservative* in that they underestimate the performance of our minimalist approach.

Hardware feasibility: We assume that both the application-specific algorithms and the minimalist primitives have feasible implementations that can operate at line rates. Some application-specific algorithms require a simple array of counters (e.g., [25, 44]), while others (e.g., [17, 30, 41]) and the minimalist primitives FS, SH [21, 17] involve key-value data structures. Previous work has demonstrated that it is possible to efficiently implement such key-value data structures in routers [20, 37, 31].

Processing requirements: There are two processing components: online collection and offline computation. By construction, the online collection overhead of the minimalist approach is lower. In the application-specific architecture, each packet requires as many counter updates as the number of application instances. (Further, each different flowkey for heavy-hitter detection, entropy estimation, and change detection requires separate updates.) With the minimalist approach, each packet requires only two updates, one for FS and one for SH.

We currently run estimation algorithms on the collected flow data without further sampling. Thus, the offline processing cost of the minimalist approach could be higher because the application-specific schemes only need to process compact summaries. We believe that offline processing costs are not a serious issue, given the costs/capabilities of commodity hardware today. That said, our

[7] Gao et al. [44] use the Kullback-Leibler (KL) divergence. However, it is not always well-defined. The JS divergence is based on KL divergence, but is always well-defined.

estimation procedures can be augmented with additional directed sampling, if necessary, to reduce the offline compute cost.

Memory consumption: For FS and SH, the flow record (the IP 5-tuple and other meta-data) need not be maintained in SRAM; these can be offloaded to DRAM. Only the counters tracking the byte or packet counts need to be in SRAM [31].

In terms of data structures, the above monitoring algorithms use either counter arrays or key-value structures. For example, FS and SH maintain key-value pairs, whereas sketch and FSD algorithms use a simple array of counters. In general, key-value structures use more memory than counter arrays. In order to normalize the resource consumption across the minimalist and application-specific approaches, we assume that each flow counter (i.e., the key-value pair) for the minimalist approach uses $4\times$ the memory required by a counter used in the application-specific approaches. For example, if we have an aggregate SRAM capacity of 20000 bytes and each packet counter requires 2 bytes of SRAM, the application-specific approach can keep 10000 per-flow counters whereas the minimalist approach can only have 2500 per-flow counters. We justify why this $4\times$ factor is *conservative*.

1. Each counter for the application-specific algorithms is typically at least 2 bytes [46]. We ran experiments with a sparse hash data structure [19] and found that it can store 10^6 flow counters in 8 MB, i.e., 8 bytes per counter. In other words, a *commodity, software only* implementation of a key-value structure uses only $\frac{8}{2} = 4\times$ the memory required by an array of counters.

2. Some of the application-specific algorithms (e.g., entropy estimation, heavy hitter detection) also require key-value style counters. We conservatively assume that these incur no overhead compared to an array of counters. That is, if each entry in a counter array is 2 bytes, we assume that it takes 8 bytes to store one key-value pair for the minimalist primitives but only 2 bytes to store one key-value pair for the application-specific algorithms.

3. With smarter hardware for storing flow counters such as counter braids [31], the memory requirement of the minimalist approach will be even lower. For example, maintaining 1 million flow counters using counter braids only requires 1.4 MB of memory, i.e., an effective overhead $\frac{1.4}{2}\times \ll 4\times$.

Summarizing the above discussion, we see that: the hardware requirements of our primitives are similar to the application-specific algorithms; the online processing overhead of the minimalist approach is strictly lower; and the minimalist primitives have at most a $4\times$ memory overhead. Thus, for the rest of the paper, we only consider the conservative $4\times$ memory overhead to generate an equivalent resource configuration for the minimalist approach. In other words, given the same amount of available SRAM the minimalist algorithms can only maintain one-fourth as many flow counters as the application-specific approach.

5.3 Configuring the different algorithms

Application-specific case: To configure the different algorithms, we follow the guidelines and recommended parameters from the literature:

1. The FSD estimation algorithm uses an array of $fsd \times F$ counters, where F is the number of distinct flows in a measurement interval. Following the guidelines of Kumar et al. [26], we set $fsd = 0.7$.

2. We configure the heavy-hitter detection algorithm with $hh \times F$ counters with $hh = 0.3$, divide these equally among the six instances, and focus on the top-50 detection rate.

3. The entropy estimation algorithm is an (ϵ, δ) approximation, i.e., the relative error is at most ϵ with probability at least $1-\delta$. The number of counters it uses increases as we require tighter guarantees (lower ϵ and δ). However, Lall et al. [30] show that in practice it works well even with loose bounds. Thus, we set $\epsilon = \delta = 0.5$.

4. For superspreader detection, we set $K = 100$ and $b = 4$. Again, since loose bounds work well in practice, we set $\delta = 0.5$.

5. The sketch data structure has three parameters: h, the number of hash functions; q, the size of the counter array per hash function; and the detection threshold θ. Following Krishnamurthy et al. [25], we set $h = 10$, $q = 1024$, and $\theta = 0.05$.

6. For degree histogram estimation, we use the same configuration as Gao et al. [44].

Minimalist case: The minimalist approach has two configuration parameters: the number of flow records it can collect (L) and, for ingress routers, the FS-SH split (f). To determine L, we configure the application-specific algorithms using the above guidelines. For each trace, we measure the total number of counters used across the different application-specific algorithms and scale it *down* by a factor of 4 as discussed earlier to calculate the value of L to be used for that setup. We set $f = 0.8$, giving 80% of the resources on each router to the FS component.

5.4 Estimation phase in minimalist approach

The estimation phase for the minimalist approach is conceptually simple. Since we have the actual flow records (i.e., the 5-tuples along with the packet counts), we can run exact estimation algorithms. For example, we can compute the flow size distribution of the reported flows and use that as our estimate of the true flow size distribution. Similarly, we can compute the observed (normalized) entropy of different flowkey combinations from the reported flows and use it as the estimate of the true (normalized) entropy.

The only issue is how to use the flow reports from the FS and SH components for the different estimation tasks. We use the following heuristic. First, we take the union of the flow records reported by SH (after normalizing packet counts by the sampling rate [17]) and the flow records reported by FS.[8] We compute the FSD and entropy, and detect heavy hitters or changes per-source (or destination), on this merged set of flow records. Second, we use the set of flow records reported by FS for detecting superspreaders and to compute the outdegree histogram. Again, we note that our procedures for merging the flow reports from the FS and SH components for various estimation tasks are only heuristics and may not be optimal. Finding the optimal estimator for specific application-level metrics from a given set of collection primitives is an interesting direction of future work.

Note that the minimalist approach exports the actual flow records. Thus, it is possible to run any estimation procedure on these flow records to compute any application metric, even unforeseen ones.

[8]If the same flow is reported by both FS and SH, we use the FS record because the packet count in FS is exact.

Trace	Description	Avg # pkts (millions)	Avg # flows (thousands)
Caida 2003	OC-48, large ISP	6	400
Univ-2	UNC, 2003	2.5	91
Univ-1	USC, 2004	1.6	93
Caida 2007-2	OC-12	1.3	45
Caida 2007-1	OC-12	0.7	30

Table 2: Traces used in the single router experiments; averages are over 5-minute epochs

6. TRACE-DRIVEN EVALUATION

In this section, we compare the minimalist approach against the different application-specific algorithms using packet and flow-level traces collected from different settings. We start with a single router evaluation and then proceed to a network-wide evaluation.

6.1 Single Router Case

Using trace-driven evaluations, we answer the following questions:

- How does the accuracy of the minimalist approach compare with the application-specific approaches when configured with the aggregate memory used by the application-specific algorithms? (Section 6.1.1)
- How sensitive is individual application performance to the amount of memory available to the minimalist approach? (Section 6.1.2)
- How does the success of the minimalist approach depend on the set of application-specific algorithms that are implemented on the router (we call this an *application portfolio*)? That is, when does it make sense to adopt a minimalist approach instead of implementing each application-specific alternative? (Section 6.1.3)
- How should we split resources between FS and SH? (Section 6.1.4)

Table 2 summarizes the five different one-hour packet header traces (binned into 5-minute epochs) used for the single-router evaluation.

6.1.1 Accuracy: minimalist vs. application-specific

We use the default parameters from Table 1 and run the minimalist approach configured with the total normalized memory used by the six algorithms. Then we compute the relative accuracy difference for each application. Let $Acc_{specific}$ denote the accuracy of the application-specific algorithm and let $Acc_{minimalist}$ denote the accuracy of the minimalist approach for that application. The *relative accuracy difference* is $\frac{Acc_{minimalist} - Acc_{specific}}{Acc_{specific}}$. By construction, a positive value indicates that the accuracy of the minimalist approach is better; a negative value indicates otherwise.[9]

All the algorithms are inherently randomized; we present the results over five independent runs with different seeds. Figure 3 shows the relative accuracy difference using a box-and-whiskers plot for the different traces. Each box shows the 25%ile, median, and 75%ile values. The whiskers extend to the most extreme data points not considered outliers. This corresponds to a length of at most $1.5\times$ the difference between the 25%ile and 75%ile values.

The result shows that the median value of this metric is positive in most cases; i.e., the minimalist approach outperforms the application-specific alternative in most applications. Further, even the 25%ile is positive in many cases; i.e., the minimalist approach consistently outperforms the application-specific approaches. Only in heavy-hitter detection (Figure 3(b)) does the minimalist approach perform worse; even then the median accuracy gap is at most 0.08. This result gives a high-level answer to the second challenge from Section 3:

The minimalist approach provisioned with the total resources used by the six applications performs better than or comparable to the application-specific approaches.

We now proceed to answer to two natural questions: (a) what if we consider each application class in isolation and (b) what types of application portfolios does the minimalist approach perform favorably in. For brevity, we only present the results from the Caida 2003 trace.

6.1.2 Application Sensitivity

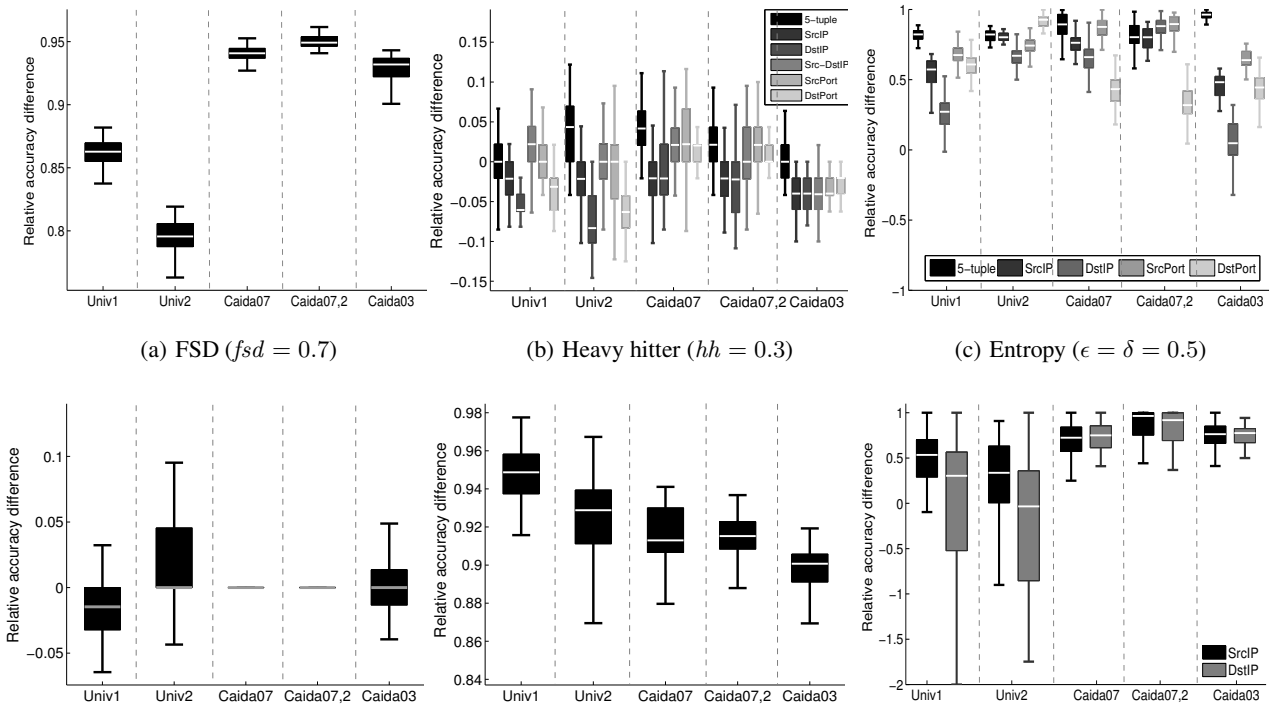
In the following experiments, we try 2-3 configurations for each application-specific algorithm. For each configuration, we consider a minimalist approach provisioned with G times as much memory as that used by the algorithm *in isolation*. For example, if the application-specific algorithm uses 10000 counters at 2 bytes/counter, the minimalist approach has $G \times 10000 \times 2$ bytes of SRAM.

As before, we focus on the relative accuracy difference between the minimalist and application-specific approach. Figure 4(a) plots the relative accuracy difference between the minimalist approach and the FSD estimation algorithm. We show three different configurations with the FSD algorithm using $fsd = 0.7$, 1, and 1.5. For some configurations (e.g., $fsd = 1.5$, $G = 1$), the minimalist approach performs worse. The large negative values of the metric result from the low WMRD values at these points. Since we normalize the difference by the WMRD of the application-specific case, the relative difference gets magnified. The absolute accuracy of the FSD algorithm improves (i.e., the WMRD goes down) as it is provisioned with more resources (not shown). For example, for the configuration $fsd = 1.5$ and $G = 1$, the WMRD for the FSD algorithm was 0.02 and the WMRD for the minimalist approach 0.05. Both values are small for many practical purposes [26].

Figure 4(b) shows similar results for heavy-hitter detection, with hh set to $0.3, 0.5$, and 0.7. For clarity, we average the relative accuracy difference across the six heavy-hitter instances. The minimalist approach is worse than the application-specific approach. However, as G increases, the accuracy gap closes significantly. One reason for the poor accuracy of the minimalist approach is that we configure the SH algorithm to operate at the 5-tuple granularity and then subsequently project results to other dimensions. In fact, the minimalist approach performs better if we only consider the 5-tuple granularity (Figure 3(b)). The relative accuracy for the other flowkeys is negative because the application-specific SH instances can directly determine heavy hitters without going through this projection step. We could also configure the SH algorithm in the minimalist approach to operate at multiple flowkeys. We trade-off a small reduction in accuracy for a significant reduction in online processing overhead complexity since we only need to run one instance of the SH algorithm instead of six instances.

Entropy estimation (Figure 4(c)) with $\epsilon = \delta$ set to 0.2 and 0.5 and superspreader detection (not shown) show similar trends. If we consider each application in isolation, the minimalist approach

[9] Some metrics denote "error" while others denote "accuracy". For error metrics (FSD, entropy, degree histogram, change detection) the relative accuracy as defined is negative when the minimalist approach performs better. For ease of presentation, we reverse the sign of the numerator in these cases.



(a) FSD ($fsd = 0.7$) (b) Heavy hitter ($hh = 0.3$) (c) Entropy ($\epsilon = \delta = 0.5$)

(d) Superspreader (K, b, δ=100, 4, 0.5) (e) Degree histogram (f) Change Detection

Figure 3: Each result shows a box-whiskers plot with the median, 25%ile, 75%ile, and extreme values. A positive value on the y-axis means that the accuracy of the minimalist approach was better; a negative value indicates otherwise. For most applications, the minimalist approach outperforms the application-specific alternatives. In the cases where the performance is worse, it is only worse by a small relative margin.

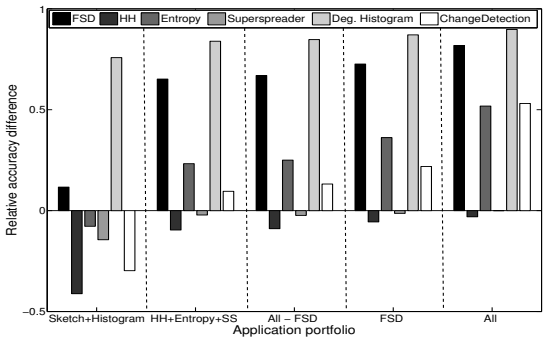


Figure 5: Effect of application portfolio on the relative accuracy difference. The portfolios are in increasing order of memory usage from left to right.

performs worse. But, the gap closes as G increases and the minimalist approach eventually outperforms the application-specific algorithm.

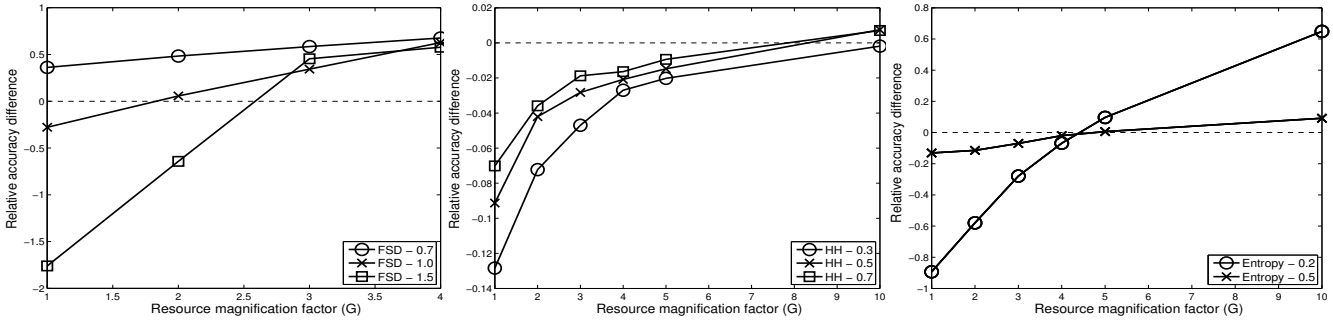
6.1.3 Sensitivity to Application Portfolio

Next, we evaluate the effect of varying the application portfolio. That is, we consider the case where the router implements only a subset of the six applications described earlier. For a fixed portfolio, we use the default configurations from Table 1 and run the minimalist approach configured with the aggregate resources contributed by only the applications within this portfolio. The relative accuracies are computed with respect to the default configurations for the different applications (even for those not in the portfolio). For example, the configuration labeled "Sketch + Histogram" uses resources only from sketch-based change detection and degree histogram estimation (the least resource-intensive applications). At the other extreme, the configuration labeled "All" uses the aggregate resources (as in Figure 3).

Figure 5 shows the portfolios in increasing order of memory usage. For clarity, we show averages across the different flowkeys for heavy-hitter detection, entropy estimation, and change detection. We observe two effects. First, for larger application portfolios (i.e., as the requirements of management applications increase), a minimalist approach is actually a better alternative as the relative accuracy difference becomes positive for almost all applications. Second, if there are some resource-intensive applications (e.g., FSD estimation), then it is better to adopt a minimalist approach because it benefits all potential applications, even those not specified in the current application portfolio.

6.1.4 Split between FS and SH

So far, we fixed the FS-SH split to be $f = 0.8$. Figure 6 shows the effect of varying f. The x-axis is f, the fraction of resources allocated to FS. For most applications, increasing f improves the accuracy of the minimalist approach, but there is a diminishing returns effect. For heavy-hitter detection, as expected, giving more resources to SH helps, but the improvement is fairly gradual. In light of this, the 80-20 split is a reasonable tradeoff across the different application classes.



(a) FSD (b) Heavy hitter (c) Entropy

Figure 4: Exploring the sensitivity of applications in isolation. The zero line represents the point at which the minimalist approach starts to outperform the application-specific approach. The resource magnification factor captures the sharing effect of aggregating resources across applications.

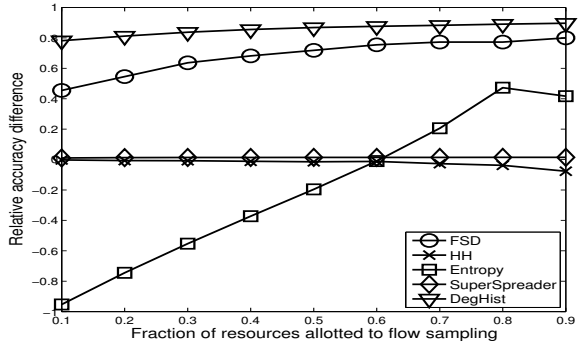


Figure 6: Varying the split between FS and SH

6.2 Network-wide evaluation

Dataset and Setup: We use a one-hour snapshot of flow data collected across eleven routers from the Internet2 backbone. There are roughly 1.4 million distinct flows and 9.5 million packets in aggregate per 5-minute interval. We map each flow entry to the corresponding network ingress and egress points [18]. Unlike the packet traces used earlier, these are flow records with sampled packet counts (with 1-in-100 sampling). We assume that the sampled flow records represent the actual traffic and use the sampled counts as the actual packet counts. Also, IP-addresses in the dataset are anonymized by zero-ing out the last 11 bits. We treat each anonymized IP as a unique IP. Thus, the entropy and outdegree measures are computed at this granularity. Since we are only interested in the *relative* performance, this dataset is still valuable for understanding network-wide effects.

In a network-wide setting, operators often want to compute the different traffic metrics such as FSD, entropy, heavy hitters, etc., over multiple *spatial views* [22, 43, 45, 29]. For example, we might want to understand traffic patterns on a per-ingress basis, or a per OD-pair basis, or over the entire network.

As such, we configure the application-specific algorithms on a per-ingress basis. That is, at each node, we run these algorithms only on packets originating from this node and ignore transit or terminating traffic. (In this topology, each node is an ingress for some traffic and there are no pure transit nodes that do not originate any traffic.) For example, the FSD algorithm at ATLA estimates the FSD for the traffic originating at ATLA and the superspreader algorithm at ATLA tracks only the source IPs that originate traffic at ATLA.

From this configuration, we obtain the total memory usage at each node by the application-specific algorithms. The coordinated minimalist approach from Section 4 operates on a per OD-pair granularity using this equivalent per-router memory (after scaling it down by the $4\times$ normalization factor). Given the flow records reported for each OD-pair, we estimate the traffic metrics over three spatial views: per-ingress, per-OD, and network-wide.

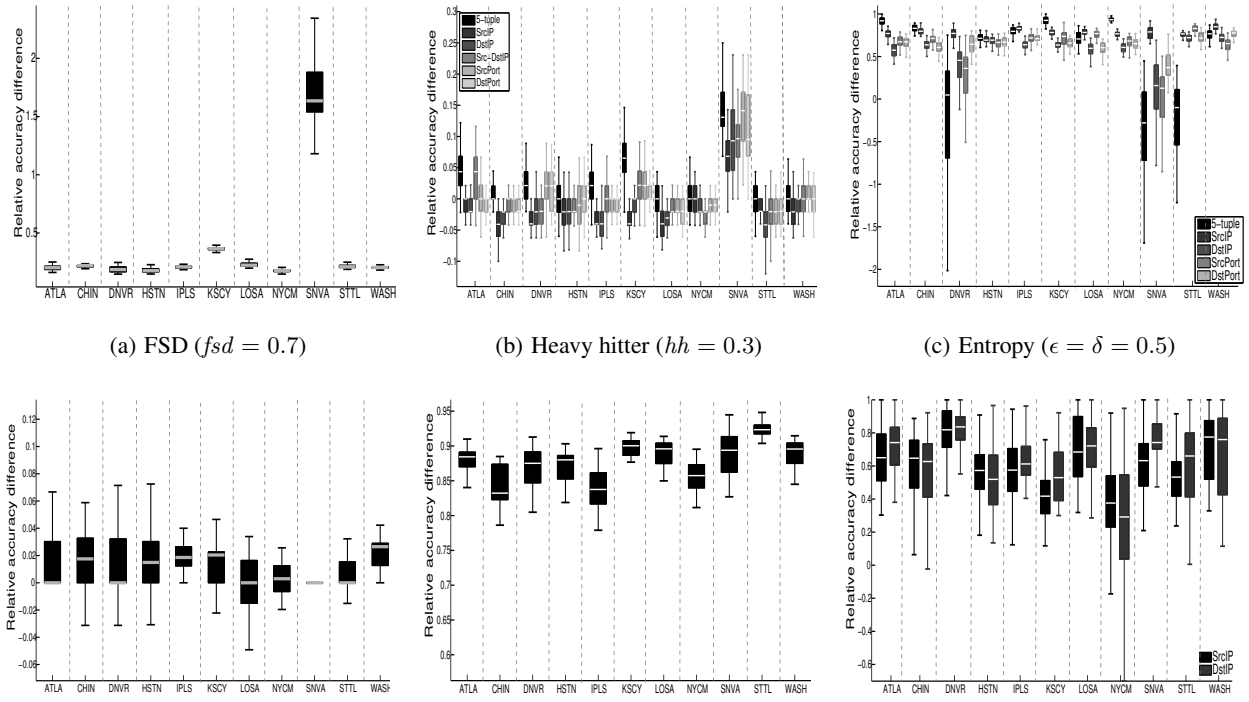
Per-ingress results: Figure 7 shows, for each ingress, the relative accuracy difference between the coordinated minimalist approach and the application-specific algorithms configured per ingress. Recall that a positive value indicates that the accuracy of the minimalist approach was better; a negative value indicates otherwise. As with the single router evaluation, we see that the minimalist approach outperforms the application-specific algorithms, except in heavy-hitter detection. (SNVA looks different from the others in the magnitude of the relative accuracy metric, but not in the qualitative sense that the minimalist approach is still better. While we have not been able to conclusively explain this observation, we noticed that the traffic volumes for SNVA were an order of magnitude lower than the rest. We suspect that this as a potential cause for the anomalous behavior.)

One potential concern is the high variability in the relative accuracy in some cases (e.g., DNVR and SNVA in Figure 7(c)). In each of these cases, we analyzed the raw accuracy values and found that the variability in fact comes from the application-specific case. That is, the accuracy of the minimalist approach has low variance, but the application-specific case can have a higher variance.[10]

Network-wide result: Next, we consider the application metrics on a *network-wide* basis. As a point of comparison, we consider an *uncoordinated minimalist* approach. Here, each node has the same resources as the coordinated case, but independently runs FS and SH on the traffic it sees.

Given the per-ingress results for the application-specific algorithms obtained earlier, we compute network-wide estimates by merging the reports from each ingress. While computing the network-wide FSD and the outdegree histogram, we need to normalize the per-ingress outputs by the number of flows and source IPs seen at each ingress. For example, to obtain the network-wide FSD, we take each per-ingress FSD and normalize it by the number of flows originating at that ingress. (One concern with merging per-ingress results to get the network-wide distributions is the risk of "double

[10]The high variance in the application-specific case is not an inherent flaw — the variance decreases with more memory. But as Figure 5 shows, adding a few memory-intensive applications makes the case for the minimalist approach stronger.



(a) FSD ($fsd = 0.7$) (b) Heavy hitter ($hh = 0.3$) (c) Entropy ($\epsilon = \delta = 0.5$)

(d) Superspreader (K, b, δ=$100, 4, 0.5$) (e) Degree histogram (f) Change Detection

Figure 7: Result showing the relative accuracy difference between the coordinated minimalist approach and the application-specific algorithms per ingress router (denoted by the abbreviated city names of the PoPs in the Internet2 backbone). A positive value indicates that the accuracy of the minimalist approach was better; a negative value indicates otherwise.

Application (error metric)	App Specific	Uncoord minimalist	Coord minimalist
FSD (WMRD)	0.16	0.19	0.02
Heavy hitter (miss rate)	0.02	0.3	0.04
Entropy (relative error)	n/a	0.03	0.02
Superspreader (miss rate)	0.02	0.04	0.009
Deg. histogram (JS)	0.15	0.03	0.02

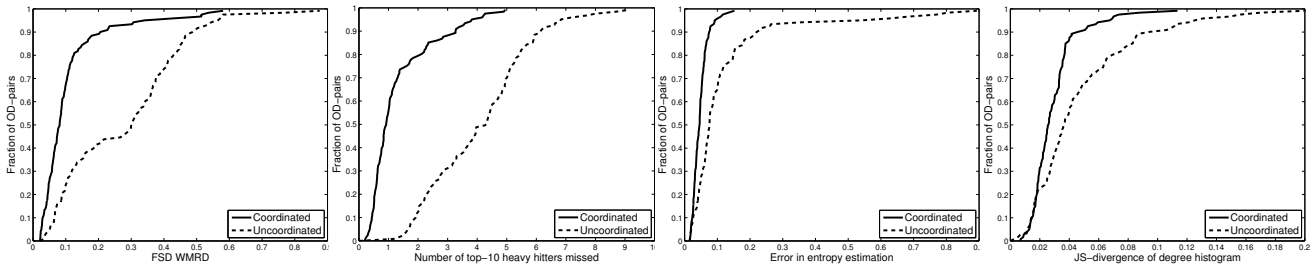
Table 3: Absolute error for network-wide metrics. Lower values imply better performance.

counting" some flows or sources. Because the per-ingress setup implicitly partitions the network-wide traffic into non-overlapping subsets, we can simply merge reports from the different ingresses for each application without worrying about double counting.) However, we cannot estimate the network-wide entropy from the per-ingress entropy values as this does not give us sufficient information. For the coordinated approach, we simply take the union of the flow records obtained for each OD-pair and run the estimation procedures on this merged set of the flow records. The estimation step for the uncoordinated case proceeds similarly, but needs additional processing to remove duplicate flow reports.

Table 3 compares the application-specific, uncoordinated, and coordinated approaches for the network-wide case in terms of the *absolute error* values. (The entropy row is empty for the application-specific column because of the aforementioned reason.) There are two main observations. First, the coordinated approach has the lowest error overall. The benefits of coordination are particularly significant for the heavy-hitter and FSD estimation applications. Second, while the uncoordinated approach provides some generality (e.g., it can also provide per OD-pair estimates whereas the per-ingress application-specific algorithms cannot), it performs worse in this evaluation. One reason is that the per-ingress application-specific algorithms are implicitly coordinated and avoid ambiguity/biases when we merge the results for the network-wide case. The uncoordinated minimalist approach does not have this property and multiple sources of ambiguity/bias arise when we merge flow reports from multiple routers: (i) different routers may have different sampling rates as they see different traffic volumes, (ii) flows traversing longer paths get higher sampling probabilities, and (iii) large flows are reported multiple times by SH. An additional practical benefit of the coordinated approach is that the merging and estimation algorithms are simpler and more accurate.

Per OD-pair results: Finally, we consider the different application metrics on a *per OD-pair* basis. Note that the application-specific alternatives as configured cannot provide per OD-pair results. They work at a coarse per-ingress level and we cannot compute the application metrics on a more fine-grained per-OD basis. Again, this is not an inherent limitation of application-specific approaches; we can also configure them on a per-OD basis. However, this significantly increases the complexity since we need an instance per application per OD-pair. Thus, we only consider the minimalist approaches for this result.

Figure 8 shows four application metrics for the per OD-pair case. Since superspreader detection and change detection are meaningful only when viewed across all OD-pairs, we do not consider these. Also, we focus on the top-10 heavy hitters per OD-pair. The CDFs show that the coordinated approach performs well across most OD-pairs. The 80th percentile of the WMRD, heavy-hitter miss rate,



(a) FSD (b) Heavy hitter (c) Entropy (d) Degree histogram

Figure 8: Comparing the coordinated and uncoordinated approaches on a per-OD basis.

average relative error in entropy estimation, and JS-divergence for the degree histogram are $0.1, 2, 0.05$, and 0.03 respectively. The corresponding results for the uncoordinated case are $0.4, 5, 0.15$, and 0.06. Further, the OD-pairs where the coordinated approach has poor accuracy have low traffic volume (not shown), which indicates that it performs very well for the dominant traffic patterns. The results for network-wide and per OD-pair views demonstrate the benefits of a systematic coordinated approach for network-wide monitoring.

6.3 Summary of main results

- The accuracy of the minimalist approach configured with the aggregate resources used by the six different applications is better than or comparable to the application-specific approaches.
- With large application portfolios or if there are one or more resource-intensive applications in the portfolio, there is a clear win for a minimalist approach vs. application-specific approaches.
- A 80-20 split between FS and SH is a reasonable tradeoff across the spectrum of applications.
- In a network-wide setting, a coordinated minimalist approach provides more flexibility and better accuracy while projecting results to different spatial views compared to uncoordinated and application-specific approaches.

7. DISCUSSION

Bandwidth overhead: In the application-specific architecture, each router only reports summary estimates of the various traffic metrics (e.g., FSD, entropy). Thus the bandwidth overhead for aggregating these reports is negligible. A practical concern with our proposal is the bandwidth overhead for transferring flow records to a logically centralized collector. We give a back-of-the-envelope calculation to estimate the worst-case overhead. The Internet2 dataset has roughly 1.7GB of 1-in-100 packet-sampled flow data per PoP per day. This conservatively translates into 170 GB per PoP per day or 0.6GB per PoP per five minutes for full flow capture. (This is conservative because we are normalizing the number of flows by the packet sampling rate.) Suppose, we collect this data every five minutes with a near real-time requirement that the data be sent before the start of the next five-minute interval. The bandwidth per PoP required for full flow capture would be $\frac{0.6\times 8 \text{ Gbits}}{300 \text{ seconds}} = 0.016$ Gbps. Given OC-192 backbone line rates of 10 Gbps today, it is not unreasonable to expect ISPs to use $\ll 1\%$ of the network bandwidth per PoP for measurement traffic to aid network management.

Adaptation: Another natural question is how does our minimalist approach deal with network dynamics. Estan et al. [16] and Keys et al. [23] have in-depth discussions on how to adapt the sampling rates for packet sampling, FS, and SH to changing traffic conditions. Our previous work discusses how cSamp can adapt to network dynamics [39]. We can leverage these existing techniques to make the minimalist approach robust to network dynamics.

Beyond monitoring 5-tuples: Some settings require more fine-grained monitoring capabilities that look beyond flow-level statistics. These include analyzing end-to-end performance metrics (e.g., loss, throughput, latency), deep packet inspection for signature matching, and on-demand analysis (e.g., analyze hosts that show specific patterns). Our minimalist primitives as described in this paper do not provide these capabilities. However, we believe that the broad principles underlying a minimalist approach will still apply and assume more importance with more complex monitoring requirements. One possible solution is to include other flexible primitives that provide such capabilities [10, 45] in the minimalist framework.

8. CONCLUSIONS

This paper is a reflection on recent trends in network monitoring. There is a growing demand for estimating a wide variety of traffic metrics to support different network management applications. The inadequacy of current packet-sampling-based solutions has given rise to a proliferation of many application-specific algorithms, each catering to a narrow application.

In contrast to these application-specific alternatives, we revisit the case for a minimalist architecture for flow monitoring. Such an architecture dramatically reduces router complexity and enables router vendors to focus their energies on building efficient implementations of a small number of primitives. Further it allows late binding to what traffic metrics are important, thus insulating router implementations from the changing needs of flow monitoring applications.

We demonstrated a proof-of-concept minimalist approach that combines flow sampling, sample-and-hold, and cSamp. We showed that this approach performs favorably across a wide spectrum of applications compared to application-specific approaches. Our proposal is by no means "optimal" or the final word in this problem space — the goal of this paper was to demonstrate the *feasibility* of a minimalist approach. In this respect, there are three avenues for future work: (i) developing better minimalist primitives, (ii) designing estimation algorithms that optimally leverage the data collected across different primitives, and (iii) providing formal models to reason about application requirements and performance. We hope that our work motivates further research in these directions.

Acknowledgments

We thank Ramana Kompella, Jeff Pang, and Amar Phanishayee for their feedback on this work. We would also like to thank Abhishek Kumar and Jim Xu for sharing their implementation of the FSD estimation algorithm. This work was supported in part by the National Science Foundation under grant numbers ANI-0331653, CNS-0756998, and CNS-0433540, the U.S. Army Research Office under grant number DAAD190210389.

9. REFERENCES

[1] Flexible Netflow. http://www.cisco.com/en/US/products/ps6965/products_ios_protocol_option_home.html,.

[2] Juniper cflowd. http://www.juniper.net/techpubs/software/junos/junos91/swconfig-policy/cflowd.html.

[3] Narus Intercept Solution. http://www.narus.com/index.php/solutions/intercept.

[4] NetFlow Input Filters. http://www.cisco.com/en/US/docs/ios/12_3t/12_3t4/feature/guide/gtnfinpf.html.

[5] Packet Sampling, IETF Working Group Charter. http://www.ietf.org/html.charters/psamp/charter/.

[6] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and Delay Accountability for the Internet. In *Proc. ICNP*, 2007.

[7] D. Brauckhoff, B. Tellenbach, A. Wagner, A. Lakhina, and M. May. Impact of Traffic Sampling on Anomaly Detection Metrics. In *Proc. IMC*, 2006.

[8] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proc. ACM SIGMOD*, 2003.

[9] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. Reformulating the Monitor Placement problem: Optimal Network-Wide Sampling. In *Proc. CoNeXT*, 2006.

[10] L. D. Carli, Y. Pan, A. Kumar, C. Estan, and K. Sankaralingam. PLUG: Flexible Lookup Modules for Rapid Deployment of New Protocols in High-speed Routers. In *Proc. SIGCOMM*, 2010.

[11] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954.

[12] M. P. Collins and M. K. Reiter. Finding Peer-to-Peer File-sharing using Coarse Network Behaviors. In *Proc. ESORICS*, 2006.

[13] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55, 2005.

[14] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *Proc. IMW*, 2001.

[15] N. Duffield, C. Lund, and M. Thorup. Estimating Flow Distributions from Sampled Flow Statistics. In *Proc. of ACM SIGCOMM*, 2003.

[16] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a Better NetFlow. In *Proc. ACM SIGCOMM*, 2004.

[17] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. In *Proc. ACM SIGCOMM*, 2002.

[18] A. Feldmann, A. G. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. In *Proc. ACM SIGCOMM*, 2000.

[19] Google sparse hash project. http://code.google.com/p/google-sparsehash/.

[20] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood. Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing. In *Proc. ACM SIGCOMM*, 2005.

[21] N. Hohn and D. Veitch. Inverting Sampled Traffic. In *Proc. IMC*, 2003.

[22] T. Karagiannis, D. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *Proc. ACM SIGCOMM*, 2005.

[23] K. Keys, D. Moore, and C. Estan. A Robust System for Accurate Real-time Summaries of Internet Traffic. In *Proc. SIGMETRICS*, 2005.

[24] R. Kompella and C. Estan. The Power of Slicing in Internet Flow Measurement. In *Proc. IMC*, 2005.

[25] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proc. ACM IMC*, 2003.

[26] A. Kumar, M. Sung, J. Xu, and J. Wang. Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Distribution. In *Proc. ACM SIGMETRICS*, 2004.

[27] A. Kumar and J. Xu. Sketch Guided Sampling – Using On-Line Estimates of Flow Size for Adaptive Data Collection. In *Proc. IEEE Infocom*, 2006.

[28] A. Lakhina, M. Crovella, and C. Diot. Diagnosing Network-Wide Traffic Anomalies. In *Proc. ACM SIGCOMM*, 2004.

[29] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc. ACM SIGCOMM*, 2005.

[30] A. Lall, V. Sekar, J. Xu, M. Ogihara, and H. Zhang. Data Streaming Algorithms for Estimating Entropy of Network Traffic. In *Proc. ACM SIGMETRICS*, 2006.

[31] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter Braids: A Novel Counter Architecture for Per-Flow Measurement. In *Proc. SIGMETRICS*, 2008.

[32] M. P. Collins and M. K. Reiter. Hit-list Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. In *Proc. RAID*, 2007.

[33] H. Madhyastha and B. Krishnamurthy. A Generic Language for Application-Specific Flow Sampling. *ACM CCR*, 38(2):7–15, Apr. 2008.

[34] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang. Is Sampled Data Sufficient for Anomaly Detection? In *Proc. IMC*, 2006.

[35] A. Ramachandran, S. Seetharaman, and N. Feamster. Fast Monitoring of Traffic Subpopulations. In *Proc. IMC*, 2008.

[36] B. Ribeiro, D. Towsley, T. Ye, and J. Bolot. Fisher information of sampled packets: an application to flow size estimation. In *Proc. IMC*, 2006.

[37] S. Kumar and P. Crowley. Segmented Hash: An Efficient Hash Table Implementation for High Performance Networking Subsystems. In *Proc. ACM ANCS*, 2005.

[38] V. Sekar, N. Duffield, K. van der Merwe, O. Spatscheck, and H. Zhang. LADS: Large-scale Automated DDoS Detection System. In *Proc. USENIX ATC*, 2006.

[39] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. Kompella, and D. G. Andersen. cSamp: A System for Network-Wide Flow Monitoring. In *Proc. NSDI*, 2008.

[40] V. Sekar, A. Gupta, M. K. Reiter and H. Zhang. Coordinated Sampling sans Origin-Destination Identifiers: Algorithms and Analysis. In *Proc. COMSNSETS*, 2010.

[41] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum. New Streaming Algorithms for Fast Detection of Superspreaders . In *Proc. NDSS*, 2005.

[42] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang. Worm Origin Identification Using Random Moonwalks. In *Proc. IEEE Symposium on Security and Privacy*, 2005.

[43] K. Xu, Z.-L. Zhang, and S. Bhattacharya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *Proc. ACM SIGCOMM*, 2005.

[44] Y. Gao, Y. Zhao, R. Schweller, S. Venkataraman, Y. Chen, D. Song and M.-Y. Kao. Detecting Stealthy Attacks Using Online Histograms. In *Proc. IWQoS*, 2007.

[45] L. Yuan, C.-N. Chuah, and P. Mohapatra. ProgME: Towards Programmable Network MEasurement. In *Proc. SIGCOMM*, 2007.

[46] Q. Zhao, J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. In *Proc. ACM SIGMETRICS*, 2006.