A Case for a RISC Architecture for Network Flow Monitoring

Vyas Sekar[1], **Michael K. Reiter**[2], **Hui Zhang** [1]

CMU-CS-09-125

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]Carnegie Mellon University, [2]UNC-Chapel Hill

This work was supported in part by NSF awards CNS-0326472, CNS-0433540, and ANI-0331653.

Keywords: Network Monitoring, Sampling, Coordination, RISC

Abstract

Several network management applications require high fidelity estimates of flow-level metrics. Given the inadequacy of current packet sampling based solutions, many proposals for application-specific monitoring algorithms have emerged. While these provide better accuracy, they increase router complexity and require router vendors to commit to hardware primitives without knowing how useful they will be to future monitoring applications. We argue that such complexity is unnecessary and build a case for a "RISC" approach for flow monitoring, in which generic collection primitives on routers provide data from which other traffic metrics can be computed using separate, offline devices. We demonstrate one such RISC approach by combining two well-known primitives: flow sampling and sample and hold. We show that allocating a router's memory resources to these generic primitives can provide similar or better accuracy on metrics of interest than dividing the resources among several metric-specific algorithms. Moreover, this approach better insulates router implementations from changing monitoring needs.

1 Introduction

Flow monitoring supports several critical network management tasks such as traffic engineering [20], accounting [14, 18], anomaly detection [30, 31], identifying and understanding end-user applications [11, 24], understanding traffic structure at various granularities [47], detecting worms, scans, and botnet activities [48, 45, 39], and forensic analysis [46]. These require high-fidelity estimates of traffic metrics relevant to each application.

High traffic rates can exceed the monitoring functionality of modern routers, and since traffic is scaling at least as fast as router monitoring capability, some form of sampling or data reduction is inevitable. There are two alternatives in this space: *application-agnostic* strategies (e.g., uniform packet sampling or uniform flow sampling) or *application-specific* strategies such as data streaming algorithms for estimating specific traffic metrics of interest.

The de-facto standard for flow-level monitoring is NetFlow [10] and similar implementations from other router vendors (e.g., [3]), which by nature are intended to be application-agnostic. These employ packet sampling – each packet is selected with a sampling probability and the selected packets are aggregated into flow records. However, several studies have demonstrated the inadequacy of packet sampling for many of the applications mentioned above (e.g., see [36, 23, 15, 28, 7, 39, 18]). One consequence of these studies is that the research community has eschewed application-agnostic approaches in favor of application-specific ones, because of the perception that they provide better accuracy.

In this paper, we revisit this perception and argue, instead, for a "RISC" approach [38] for network traffic monitoring. There are two qualitative motivations for such an approach. First, the set of network management and security applications is a moving target, and new applications arise as the nature of both normal and anomalous traffic patterns changes over time. Application-specific alternatives require router vendors and network managers to commit in advance to their metrics of interest, whereas application-agnostic alternatives enable "late binding" of what metrics to consider. Second, a small number of monitoring primitives reduces the complexity of routers and monitoring devices, and enables vendors to develop highly efficient hardware implementations of these primitives.

The goal of this paper is to understand how we can redesign application-agnostic network monitoring solutions to be practical and to provide sufficient fidelity for a broad class of applications. The key insight behind a RISC approach is to decouple the *collection* and *computation* involved in traffic monitoring (Figure 1). Application-specific alternatives (e.g., data streaming algorithms) work well for the specific applications for which they are designed, precisely because they tightly couple the collection to the metrics to be computed. In contrast, we argue that a RISC approach would employ simple collection primitives on each monitoring device and manage them in an intelligent network-wide fashion, to ensure that the collected data will support computation of metrics of interest to various applications. This network-wide view becomes increasingly important as many network management and security applications inherently require a network-wide understanding of traffic patterns. The computation engines associated with the RISC approach can use separate offline devices that are not strictly required to work at line rates.

While a RISC approach intuitively has merit, it is questionable whether this approach

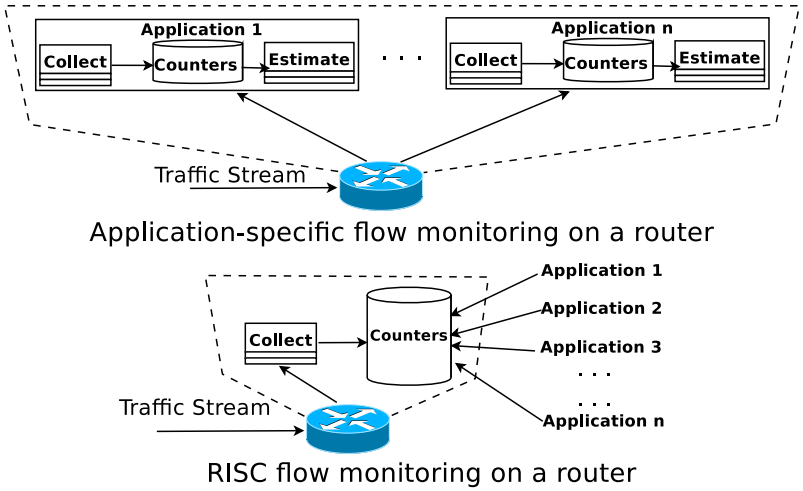


Figure 1: Schematic comparison between the application-specific architecture and the RISC architecture. The application specific architecture runs multiple algorithms each with its associated collection and computation components. The RISC architecture runs a small number of generic collection algorithms. The applications can use the collected data later (possibly offline).

can perform comparably or better than the application-specific alternatives. One rationale to suggest it can is that the primary bottleneck for high-speed monitoring is maintaining counters with sufficient fidelity in fast memory (SRAM). Each application-specific alternative not only requires independent hardware implementations, but also requires dedicated data structures and counters in SRAM. By aggregating this available pool of memory resources for use by a small number of RISC primitives, we hope to operate the RISC primitives at sufficiently high fidelity (i.e., high sampling rates) so as to enable accurate estimation of traffic metrics relevant to a wide spectrum of applications. In other words, when we look at each application in isolation, application-specific strategies are likely to work better. When we consider the portfolio of applications in aggregate, however, application-agnostic strategies have an advantage.

We present a RISC architecture that realizes this promise. The architecture has two components: single-router sampling algorithms and network-wide resource management. For single-router sampling algorithms we leverage sample and hold [18] and flow sampling [23]. For network-wide management we use Coordinated Sampling (cSamp) [42]. Our contribution is to synthesize these components in a RISC architecture and to quantitatively demonstrate the benefits of this approach.

We use trace-based analysis to evaluate the generality of this approach with respect to six application scenarios and their respective application-specific algorithms: detecting heavy hitters [18], detecting superspreaders [45], computing the entropy of different traffic subsets [32], estimating the flow size distribution [28], computing the outdegree histogram for

detecting stealthy spreaders [48], and change detection using sketches [27]. When the RISC approach has the same memory resources as required by these applications in aggregate, it provides comparable or better estimation performance relative to the application-specific approaches. Moreover, because this approach is application-agnostic, it enables computation of not-yet-conceived measures that will be interesting in the future.

2 Background and Related Work

Packet sampling and extensions: In uniform packet sampling, a router selects a subset of packets to log, merges the sampled packets into flow reports, and exports the flow reports. Packet sampling at a low sampling rate p (e.g., $p \leq 0.01$) imposes low overhead, can be implemented using just DRAM counters [10], and can support applications such as traffic engineering and accounting (e.g., [16, 15, 20]). Researchers have proposed strategies to adapt the sampling rate to changing traffic conditions, to tune the processing, memory, and reporting bandwidth overheads [17, 26]. There have also been efforts to get better traffic estimates from sampled measurements [14, 16].

However, packet sampling is known to have several inherent limitations. For example, there are known biases towards sampling larger flows (e.g., [23, 28, 36]). Further, several studies have questioned the fidelity of packet sampling for many network management applications (e.g., see [36, 23, 15, 28, 7, 39, 18]).

The inadequacy of current packet sampling alternatives has motivated the development of a large number of application-specific data streaming algorithms and proposals for flexible sampling algorithms (some of them application-specific and a few that are application-agnostic). Further, there is a growing body of work that demonstrates the need for network-wide vs. single-vantage-point solutions. We briefly describe these three classes of related work next.

Application-specific data streaming algorithms: To address the limitations of packet sampling for estimating fine-grained traffic statistics, several application-specific data streaming algorithms and counting data structures have been proposed; see [37] for a survey. The high-level approach in these is similar: use a small number of SRAM counters pertaining to the specific metric of interest and subsequently apply an estimation algorithm to recompute the traffic statistics from these counters. The seminal work of Alon et al. [5] provided such a framework for estimating frequency moments. Kumar et al. use a combination of counting algorithms and Bayesian estimation for accurate estimation of the flow size distribution [28]. Streaming algorithms have also been proposed for identifying heavy hitters [18] and for computing traffic distribution statistics such as entropy [32]. Sketch-based techniques have been used for detecting changes and anomalies [27]. However, as a result of application-specific optimizations, these algorithms lack the generality to be implemented as first-order primitives on routers.

There are a few efforts to design summary data structures that can allow a variety of queries to be answered efficiently. Notably, count-min sketches [12] can answer queries regarding frequent items, quantiles, etc. However, sketches have two key limitations. First,

they are designed primarily for *volume queries* and thus less suited for more fine-grained applications such as entropy estimation, superspreader detection, degree histogram reconstruction, or for understanding the flow size distribution. Second, a sketch data structure operates over a specific "flowkey" defined over one or more fields of the IP 5-tuple. Every distinct combination of interest would require an independent instance of a sketch data structure on a router. For example, understanding combinations of two or more fields is often necessary when operators run diagnostics or investigate anomalies. A separate sketch structure per flowkey not only incurs memory and processing overhead but also requires advance knowledge of which flowkeys will be useful; which may not be known until after the operator begins to investigate specific events.

Flexible sampling extensions: A natural extension to uniform packet sampling is to classify packets into different categories and assign a different sampling rate to each category. One class of such approaches is size-dependent sampling [29, 39], where the sampling rate depends on the flow size. Other approaches allow a network operator to define specific flow categories and only log flows relevant to these categories (e.g., [49, 1, 4, 35, 8]). These approaches can be considered as additional primitives for a RISC approach. However, they need to be configured to application requirements in advance (the specific categories and their sampling rates). In contrast, our RISC approach operates at the granularity of a generic IP flow 5-tuple, agnostic to the specific types of analyses that may be performed on the collected flows. The collected flows can then subsequently projected appropriately to answer specific queries of interest. As Section 6 shows, our approach works well for a wide class of applications.

The work closest in spirit to our approach is due to Keys et al. [25]. They design a system for providing summaries of global traffic counters and "resource hogs" along several dimensions. To do so, they use a combination of flow sampling [23] and sample and hold [18], similar to our approach in Section 4. Our work extends theirs in two significant ways, however. First, we take a network-wide perspective and show how to combine these primitives with the resource management capabilities of cSamp, in contrast to the single-vantage-point view in their work. Second, we look beyond simple traffic summaries and heavy hitters, and demonstrate that a hybrid approach can in fact support a much wider range of applications.

Network-wide sampling: There is a growing body of work that stresses the importance of network-wide measurements, e.g., to meet operational requirements [20, 30] or for anomaly detection [30, 41, 31]. In this theme, Cantieni et al. [9] provide a formulation to optimally set the packet sampling rates on a set of routers to achieve traffic engineering objectives. Our architecture builds from cSamp [42], a proposal for coordinating routers to ensure that the available monitoring resources are used in an efficient, non-redundant manner.

3 Design Considerations

Drawing on the rich body of work discussed above, we now synthesize some key requirements for a RISC architecture for flow monitoring and also derive some design principles to guide our approach.

3.1 Requirements

Generality across applications: There are already a wide spectrum of network management and network security applications as highlighted in the introduction. Further, the set of applications continues to grow and evolve as both normal and anomalous traffic patterns change over time. Since it is hard to ascertain the application mix a priori, flow monitoring primitives on routers should be *application-agnostic*.

Low complexity: The ideal monitoring primitive is full packet (or flow) capture. However, due to technological and resource constraints on routers, this is simply not a viable alternative for high-speed networks and some form of *sampling* (i.e., data loss) is inevitable. The question then is what sampling primitives should be implemented. Each such primitive may require a different set of operations and attendant data structures on routers. Implementing a separate algorithm for each monitoring application increases router complexity as the set of applications grows over time.

Support a network-wide view: For network operators of medium-to-large ISPs, the utility of measurement data is based on gaining a network-wide view of events. For example, the importance of combining network-wide measurements to meet operational requirements has been stressed in recent work [20, 30]. This network-wide view is crucial as user applications and attacks become massively distributed [41, 30, 31, 11]. For example, understanding the structure of peer-to-peer traffic [11], detecting botnets [39] and hit-list worms [34], understanding DDoS attacks [41], and network forensics [46] inherently require a network-wide view aggregated from multiple vantage points.

Most monitoring primitives today are designed for singular vantage points, i.e., a single router observing a traffic stream. This has two main drawbacks. First, it does not provide operators the ability to translate their network-wide goals into configurations for the single-vantage-point algorithms. Second, having single-vantage-point algorithms operating independently may be inefficient in using router resources. For example, routers along a routing path might duplicate their monitoring efforts [43, 42].

Enable diagnostics: Network operators not only want to understand the properties of the traffic traversing their network, but also need to go one step further to diagnose the root cause of certain events (e.g., anomalies or attacks). Thus, monitoring primitives should be able to support diagnostic tasks (e.g., decomposing the traffic into different subsets or considering different combinations of traffic patterns). For example, NetFlow style flow reports not only provide the ability to compute statistical properties of the traffic but also enable more fine-grained diagnostics.

3.2 Design Principles

Decouple collection and computation: The key insight is to decouple the *collection* and *computation* involved in traffic monitoring. This is already implicitly consistent with the operational model of many ISPs that collect flow records using NetFlow, export these flow reports to a central collection center, and run analytical/diagnostic tasks on the data. Application-specific algorithms tightly couple the collection and computation and only report

summary statistics relevant to each application. As a consequence, they are suitable only for the applications to which they cater and do not enable diagnostic drill-down capabilities.
Few, simple, and generic primitives: Routers should implement a small number of primitive operations and export configurable interfaces for these primitives to external network management tools. Recent trends in network management increasingly suggest moving the complexity out of routers and placing it in more centralized operations [6, 21]. In this vein, we argue that the monitoring functionality on routers should be amenable to simple implementations, and at the same time provide a sufficiently general abstraction to support a wide variety of management tasks. We hypothesize that such simple collection primitives, if provided with resources (e.g., processing, memory) commensurate with those consumed by the various application-specific alternatives in *aggregate*, will perform similar to the application-specific algorithms, while retaining independence from today's applications and thus improving the likelihood of supporting tomorrow's.
Network-wide management: A network-wide approach will need to take into account (a) the resource constraints (e.g., CPU, memory, processing capacity) on each router in the network, and (b) the network-wide objectives outlined by the particular management applications. Based on (a) and (b) the network-wide approach can partition monitoring responsibilities efficiently across different routers to meet the objectives. Here, "efficient" means that each router operates within its resource constraints and the resources are utilized effectively toward the network-wide goals.

3.3 Challenges

Given these requirements and high-level principles, two fundamental questions remain:

1. **Concrete Design:** What is the monitoring functionality that should be present on each network element to support a range of applications? How should monitoring responsibilities be divided across multiple measurement devices to satisfy the ISP's network-wide measurement objectives?

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

We address these questions in the rest of the paper.

4 Architecture

The "RISC" architecture we develop in this paper combines three ideas: flow sampling [23], sample and hold [18], and CSAMP [42]. Our contribution is not a new sampling mechanism. Rather, we demonstrate that these existing primitives can be combined in a suitable manner to form the basis of a generic, application-agnostic traffic monitoring architecture that meets the challenges outlined in the previous section.

4.1 Primitives

Choice of primitives: The management applications that use flow-level information can be broadly divided into two classes: those that require an understanding of volume structure (e.g., heavy-hitter detection, traffic engineering) and those that depend on the communication structure of the traffic (e.g., network security applications, anomaly detection). Our choice of primitives is guided by these two rough classes. Flow sampling is a good primitive for security and anomaly detection applications that depend on understanding communication structure, e.g., "who talks to whom" [23, 36, 34]. Similarly, sample and hold is a good primitive for traffic engineering and accounting applications that depend on volume estimates [18].

For the following discussion, a flow refers to the IP 5-tuple: the source address, destination address, source port, destination port, and protocol. We use flow sampling and sample and hold at this 5-tuple granularity. The rationale is to record flows at the most general definition possible. The collected flows can be sliced-and-diced after the fact by projecting from this general definition to more specific definitions (e.g., per destination port, per source address).

Sample and Hold (SH): Estan and Varghese proposed the sample and hold algorithm [18] for tracking heavy hitters, i.e., items with large packet counts. While packet sampling can detect heavy hitters, the estimation errors are quite high. The motivation for the algorithm is to keep near-exact counts of the heavy hitters. As each packet arrives, the router checks if it is already maintaining a counter corresponding to the *flowkey* for the packet, defined over one or more fields of the IP 5-tuple. If yes, then the router simply updates that counter. In addition, each packet is sampled independently with probability p. If the flowkey corresponding to the sampled packet is a, and a has not been selected earlier, the router keeps an exact count for a subsequently. Since this might require per-packet counter updates, the counters are maintained in SRAM [18].

One way to configure SH is to specify the flowkey (e.g., source ports, source addresses), the total number of packets (*numpkts*), and the total memory resources available (L). The sampling probability p is set to $\frac{L}{numpkts}$.[1] Instead of running a separate SH instance for all possible flowkeys, we use a single instance defined on the full IP 5-tuple.

Hash-based flow sampling (FS): The key idea behind flow sampling is to pick flows rather than packets independently at random. One possible implementation of flow sampling is as follows. Each router has a *sampling manifest* – a table of one or more hash ranges indexed using a key derived from the packet headers. On receiving a packet, the router computes the hash of the packet's 5-tuple (i.e., the flow identifier). It then selects the appropriate hash range from the manifest and logs the flow if the hash falls within this range. In this case, the hash is used as an index into a table of flows and it updates the byte and packet counters and other statistics for the flow.

We can treat the hash as a function that maps the input 5-tuple uniformly into the interval $[0, 1]$. Thus, the size of each hash range determines the flow sampling rate of the

[1] If the goal is to track heavy hitters who contribute more than a fraction $\frac{1}{x}$ to the total volume, then the sampling probability p is set to $\frac{O \times x}{numpkts}$, where O is an oversampling factor [18]. Our configuration can be viewed as determining x and O from the memory budget L.

router for each category of flows in the sampling manifest. The above approach implements flow sampling [23], since only those flows whose hash lies within the hash range are monitored.

Similar to SH, flow sampling might require flow table lookups for each packet; the flow table must therefore be implemented in SRAM. It is possible to add a packet sampling stage prior to flow sampling to make DRAM implementations possible [26]. For simplicity, we consider only configurations in which the counters are stored in SRAM.

4.2 Resource management

Combining the primitives on a single router: Let us first consider the single router case with a fixed memory (SRAM) budget L split between the SH and FS primitives. A simple way to split L is to give a fraction f to FS and the remaining $1 - f$ to SH. Since SH requires fewer counters to track heavy hitters, typical values would be $f \geq 0.7$.

Network-wide case: Let us now consider the network-wide case. Typical network management tasks are specified in terms of Origin-Destination pairs, specified by an ingress and egress router (or PoP). OD-pairs are convenient abstractions since they naturally fit many of the objectives (e.g., traffic engineering) and constraints (e.g., routing paths) for network-wide resource management. A natural extension to the single router combined primitive for the network-wide case is to consider the resource split per OD-pair [9, 42].

Here, we observe a key difference between FS and SH. It is easy to split and coordinate the FS functionality by assigning non-overlapping sampling responsibilities across routers on the path for the OD-pair. However, replicating SH functions across routers on a path will result in duplicated measurements and thus waste memory resources on routers. Since SH logs heavy hitters, the same set of heavy hitters will be reported.

To address this issue, we make a distinction between ingress and non-ingress routers. Ingress routers implement both FS and SH, splitting the aggregate memory as in the single router case. At each ingress router, the SH resources are split between the OD-pairs originating at the ingress, proportional to the traffic volume in packets per OD-pair. Non-ingress routers only implement FS. Given the resources available for FS on each router (both ingress and non-ingress), we use the cSamp framework [42] for assigning FS responsibilities in a network-wide coordinated fashion. We choose cSamp because for a given set of router resource constraints it (1) provides the optimal flow coverage (number of distinct flows logged), (2) provides a framework to specify fine-grained network-wide flow coverage goals, (3) efficiently leverages available monitoring capacity and minimizes redundant measurements, and (4) naturally load balances responsibilities to avoid hotspots.

Overview of cSamp: 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 objective function specified in terms of the fractional flow coverages per OD-pair. The output is a set of *sampling manifests* specifying the monitoring responsibility of each router in the network. The sampling manifest in cSamp is a set of tuples of the form $\langle OD, [start, end] \rangle$, where $[start, end] \subseteq [0, 1]$ denotes a hash range. In the context of the FS algorithm described earlier, this means that the OD-pair identifier is used as the "key" to

get a hash range from the sampling manifest.[2]

The key idea is to bootstrap routers with the same hash function but assign *non-overlapping* hash ranges per OD-pair, so that flows sampled by different routers do not overlap. This coordination makes it possible to achieve network-wide flow coverage goals specified as a function of per-OD-pair flow coverage values.

cSAMP **formulation:** Each OD-Pair OD_i $(i = 1, \ldots, M)$ is characterized by its router-level path P_i and T_i, the estimated number of IP-level flows per measurement epoch (e.g., five minutes).[3] Each router R_j $(j = 1, \ldots, N)$ is constrained by the available *memory* for maintaining per-flow counters in SRAM; L_j captures this constraint, and denotes the number of flows R_j can record and report in a given measurement interval. 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.

The specific goal is a two-step objective. 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 (4)) and the total flow coverage $\sum_i(T_i \times C_i)$ is maximized. The rationale behind the two-step objective is as follows. Maximizing the minimum coverage provides *fairness* in apportioning 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 such a fair allocation, the second step ensures that the residual resources are used in an *efficient* manner to achieve maximum aggregate coverage.

$$\begin{aligned}
&\text{Maximize} \textstyle\sum_i(T_i \times C_i), \quad \text{subject to} \\
&\forall j, \qquad \textstyle\sum_{i:R_j \in P_i}(d_{ij} \times T_i) \leq L_j && (1)\\
&\forall i, \qquad C_i = \textstyle\sum_{j:R_j \in P_i} d_{ij} && (2)\\
&\forall i,\ \forall j, \quad d_{ij} \geq 0 && (3)\\
&\forall i, \qquad \alpha \leq C_i \leq 1 && (4)
\end{aligned}$$

The solution $d^* = \{d^*_{ij}\}$ to this two-step procedure yields the optimal sampling strategy. This solution is then translated into the *sampling manifests* specifying the FS responsibilities per 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 routers in the network. This module takes into account the prevailing network conditions, policies, 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

[2] cSAMP has been reformulated to not require a router to determine the OD-pair for each packet [44], though here we describe the simpler approach using OD-pairs.

[3] For simplicity, we assume that each OD-pair has one route, though cSAMP accommodates multi-path routing [42].

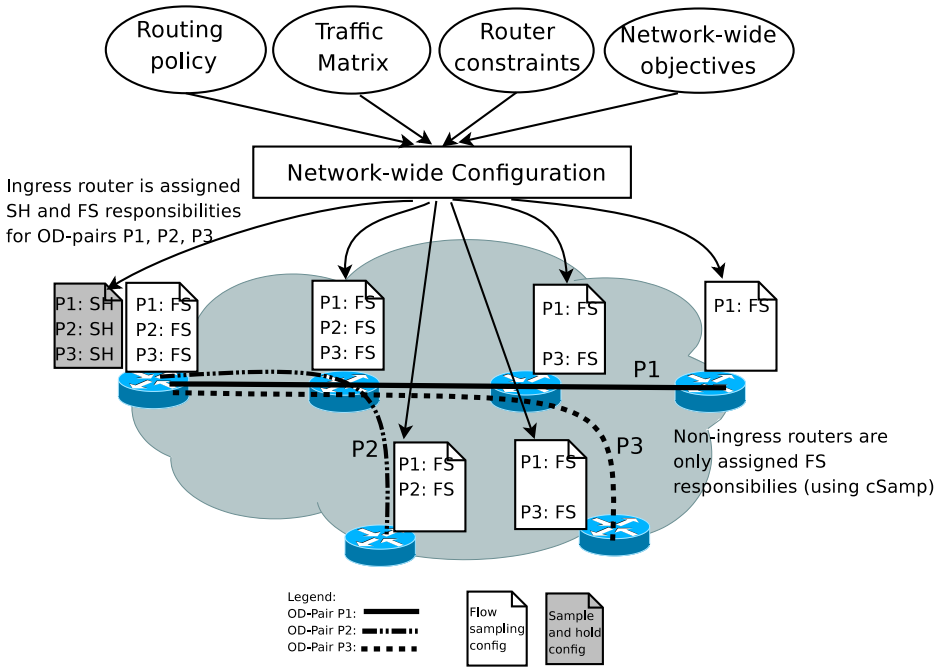


Figure 2: Example of a network-wide configuration for the RISC approach

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 as discussed earlier – each router is only assigned FS responsibilities for the paths of OD-pairs it lies on and these are specified as non-overlapping hash ranges per OD-pair.

5 Evaluation Methodology

This section describes the applications and configurations we use in our evaluations. For each application, we first describe the corresponding data streaming algorithms (both the online collection and offline inference components) and accuracy metrics in Section 5.1. Then in Section 5.2, we describe how we compare the RISC approach against the application-specific algorithms. Table 1 summarizes the applications and the corresponding application-specific algorithms, accuracy metrics, and configuration parameters. The table also shows the default configuration parameters we use for each case.

5.1 Applications and accuracy metrics

For each application, we give a brief formal description of the problem, some representative network management applications that require their use, and the respective accuracy criteria.

Application	Accuracy/Error Metric	Algorithm	Parameters (defaults)
FSD estimation	WMRD	[28]	fsd (0.7)
Heavy hitter detection	Top-k detection rate	[18]	hh, k $(0.3, 50)$
Entropy estimation	Relative Error	[32]	ϵ, δ $(0.5, 0.5)$
Superspreader detection	Detection accuracy	[45]	K, b, δ $(100, 4, 0.5)$
Change detection	falsepos + falseneg	[27]	h, k, θ $(10, 1024, 0.05)$
Deg. histogram estimation	JS-divergence	[48]	–

Table 1: Summarizing the applications, accuracy metrics, algorithms, and parameters

Flow size distribution (FSD) estimation: Consider the set of all flows in a traffic stream. Let F denote the total number of flows and F_l denote the number of flows with size l (in number of packets per flow). The FSD estimation problem is to determine $\forall l = 1 \ldots z, \phi_l = \frac{F_l}{F}$, where z is the size of the largest flow. Understanding the FSD is useful for a number of measurement and management applications such as estimating gains from proxy caches, configuring flow-switched networks, accounting, attack detection, and traffic matrix estimation [15, 28]. We use the data streaming and expectation-maximization algorithm proposed by Kumar et al. [28].

A natural accuracy metric for the FSD estimation problem is the weighted mean relative difference (*WMRD*) between the actual distribution F_l and the estimated distribution $\hat{F}_l$. 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 of heavy-hitter detection is to identify the top k items with the most traffic along specific traffic dimensions (e.g., source addresses, source ports etc.). Such measures are routinely used by network operators to understand end-to-end application patterns and resource hogs [13, 2] as well as for traffic engineering and accounting [18].

We use the SH algorithm [18]. We configure it to run with six instances (i.e., defining different flowkeys for the algorithm): source port, destination port, source address, destination address, 5-tuple, and source-destination address pairs. The accuracy metric is the *top-k detection rate*: get the top k exact heavy hitters and the top k estimated heavy hitters and compute the set intersection between these two sets. (The RISC approach also uses SH. As discussed earlier, the difference is that we use only one instance at the 5-tuple granularity and use offline projections from this to other dimensions.)

Entropy estimation: The entropy of a random variable X is defined as $H(X) = -\sum_{i=1}^{N} Pr(x_i)\log(Pr(x_i)$ where $x_1, \ldots, x_N$ is the range of values for X, and $Pr(x_i)$ represents the probability that X takes the value x_i. For many traffic applications, it is useful to normalize the entropy between zero and one as $H_{norm} = \frac{H}{\log(N_0)}$, where N_0 is the number of distinct x_i values present in a given measurement epoch [31].

The entropy of traffic distributions has been shown to aid a wide variety of network monitoring applications such as anomaly detection [19, 31] and traffic classification [47]. The motivation for entropy-based analysis is that it can capture fine-grained properties that cannot be understood with simple volume-based analysis.

We use the data streaming algorithm proposed by Lall et al. [32]. Similar to the heavy hitter detection application, we consider five traffic dimensions of interest: 5-tuple, source port, destination port, source address, and destination address. The accuracy metric is the *relative error* in estimating the normalized entropy. 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: For many security applications such as scan, worm, and botnet detection, it is useful to identify "superspreaders" – source IPs that contact a large number of distinct destination IPs. This is different from the heavy-hitter detection problem since it involves finding sources that communicate with distinct destinations as opposed to finding sources generating large traffic volumes.

We use the one-level superspreader detection algorithm proposed by Venkataraman et al. [45]. The algorithm is characterized by three parameters K, b, and δ. The goal of the algorithm is to detect all hosts that contact at least K distinct destinations with probability at least $1 - \delta$, while guaranteeing that a source that contacts at most $\frac{K}{b}$ distinct destinations is reported with probability at most δ. The accuracy metric is the *detection accuracy* which is the number of true superspreaders that are reported. For brevity, we do not consider the false positive rate since it was close to zero.

Change detection: Change detection is an important component of anomaly detection for detecting DDoS attacks, flash crowds, and worms [27]. The problem can be formally described as follows. Suppose we discretize the traffic stream into five-minute measurement epochs ($t = 1, 2, \ldots$). Let each $I_t = \alpha_1, \alpha_2, \ldots$ be the input traffic stream for epoch t. Each packet α_i is associated with a flowkey a_i and a count c_i (e.g., number of bytes in the i^{th} packet or simply 1 if we are interested in counting packets). Let $Obs_a(t) = \sum_{i:a_i=a} c_i$ denote the aggregate observed count for flowkey a in epoch t. Let $Fcast_a(t)$ denote the forecast value (e.g., computed using exponentially weighted moving average) for item a in epoch t. The forecast error for a then is $Err_a(t) = Obs_a(t) - Fcast_a(t)$. Let $F2Err_t = \sum_a Err_a(t)^2$ denote the second moment of the forecast errors. The goal of the change detection is to detect all a with $Err_a(t) \geq \theta \times \sqrt{F2Err_t}$, where θ is a user-defined threshold. We define the *change detection accuracy* as the sum of the false positive (flowkeys that didn't change much but were reported) and false negative rates (flowkeys that changed but weren't reported).

We use the sketch-based change detection algorithm proposed by Krishnamurthy et al. [27]. Sketches are particularly appealing since they have a natural "linearity" property – computing functions which are linear combinations just involves linear operations on the sketch

data structure. Thus, they are naturally amenable to forecasting and error-detection. We consider two instances focused on identifying changes in the number of packets in source and destination address distributions.

Degree histogram estimation: Finally, we consider the problem of computing the *out-degree histogram* of a traffic stream. The outdegree of a source IP address is the number of distinct destination IP addresses it contacts within a fixed measurement epoch. Consider the following histogram. For bucket i, let m_i denote the number of sources whose outdegree d is at least 2^i and at most $2^{i+1}-1$. The goal is to estimate these m_i values. While understanding the property of the outdegree distribution might be independently useful for understanding traffic structure, a specific application is to detect botnets involved in coordinated scans [48] by detecting changes in the outdegree histogram. We use the "sampling algorithm" proposed by Gao et al. [48].

Given the exact distribution $\{m_1, m_2, \ldots\}$ and an estimated distribution $\{\hat{m}_1, \hat{m}_2, \ldots\}$, the accuracy metric we consider is the *Jensen-Shannon (JS) divergence*.[4]

5.2 Assumptions and Approach

Assumptions: We make three assumptions in our evaluation – we revisit these in Section 7.

- Both the application-specific algorithms and the RISC primitives have feasible implementations that can operate at line-rates. Some algorithms require key-value style data structures while others require simple counter arrays. We assume that both incur similar processing costs.
- Each key-value pair for the RISC primitives use $4\times$ as much memory as a corresponding "counter" for the application specific algorithms. Some streaming algorithms also require key-value structures – we conservatively assume that these do not incur any memory overhead. For example, if each array entry is 2 bytes, we assume that it takes 8 bytes to store one key-value pair for the RISC primitives but that it only takes 2 bytes to store one key-value pair for the application-specific algorithms.
- The RISC approach can use exact, possibly offline, computation resources.

Configuring the application-specific algorithms: The FSD estimation algorithm uses a counter array of size $fsd \times F$, where F is the number of distinct flows in the measurement epoch. Following the guidelines of Kumar et al. [28], we set $fsd = 0.7$. 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. 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 (i.e., lower ϵ and δ). However, Lall et al. found that it works well in practice even with

[4]Gao et al. [48] 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.

loose bounds [32]. Thus, we set $\epsilon = \delta = 0.5$. For superspreader detection, we set $K = 100$ and $b = 4$. Again, since loose bounds work well in practice, we set $\delta = 0.5$. The sketch data structure has three parameters: h, the number of hash functions; k, the size of the counter array per hash function; and the detection threshold θ. Based on the observations of Krishnamurthy et al. [27], we set $h = 10$, $k = 1024$, and $\theta = 0.05$. For degree histogram estimation, we use the same configuration as Gao et al. [48].

Configuring the RISC approach: The RISC approach has two configuration parameters – the number of flow records it can collect (L) and the FS-SH split (f). L is determined by the configurations of the individual application-specific primitives described above. We measure the aggregate memory usage $L_{app\text{-}spec}$ of the different algorithms and scale it *down* by a factor of 4. (This models a key-value data structure being more memory intensive than a counter array as discussed earlier.) We set $f = 0.8$ giving 80% of the resources to FS.

Computing estimates in the RISC approach: Given the flow records reported by FS and SH (after the normalization by sampling rate [18]), we take the union of the flow records giving preference to reports from FS. (The count reported for a given flow record with FS is exact; the count reported by SH is approximate.) We use this merged set and run exact algorithms to compute the FSD, entropy, to detect heavy hitters or changes per-source (or destination). Additionally, we logically retain the set of FS records alone. We use this set for detecting superspreaders and computing the degree histogram, since it is more appropriate to normalize flow-level estimates using just the FS rate.

Since the RISC approach exports the actual flow records, it is possible to run a simple exact algorithm on these flow records to compute any application metric, even unforeseen ones.

Measure of success: Let $Acc_{specific}$ denote the accuracy of the application-specific algorithm and let Acc_{risc} denote the accuracy of the RISC approach for that application after the merge operation and using an exact algorithm to compute the relevant metric on the merged data. We define the *relative accuracy difference* as $\frac{Acc_{risc} - Acc_{specific}}{Acc_{specific}}$. By construction, a positive value indicates that the accuracy of the RISC approach is better; a negative value indicates otherwise.[5]

6 Results

6.1 Single router evaluation

Datasets and roadmap: Table 2 summarizes the five different one-hour packet header traces (binned into 5-minute epochs) used in this section. Using trace-driven evaluations, we answer the following questions:

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

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

- How does the accuracy of the RISC approach compare with the application-specific approaches when configured to use the aggregate resources on a single router? (Section 6.1.1)
- How sensitive is each application to the operating regime of the RISC approach? (Section 6.1.2)
- How does the success of the RISC approach depend on the set of application-specific algorithms that we assume are implemented on the router (we call this an *application portfolio*) – at what point does it make sense to adopt a RISC approach instead of implementing the application-specific alternatives (Section 6.1.3)?
- How should we split the resources between FS and SH? (Section 6.1.4)

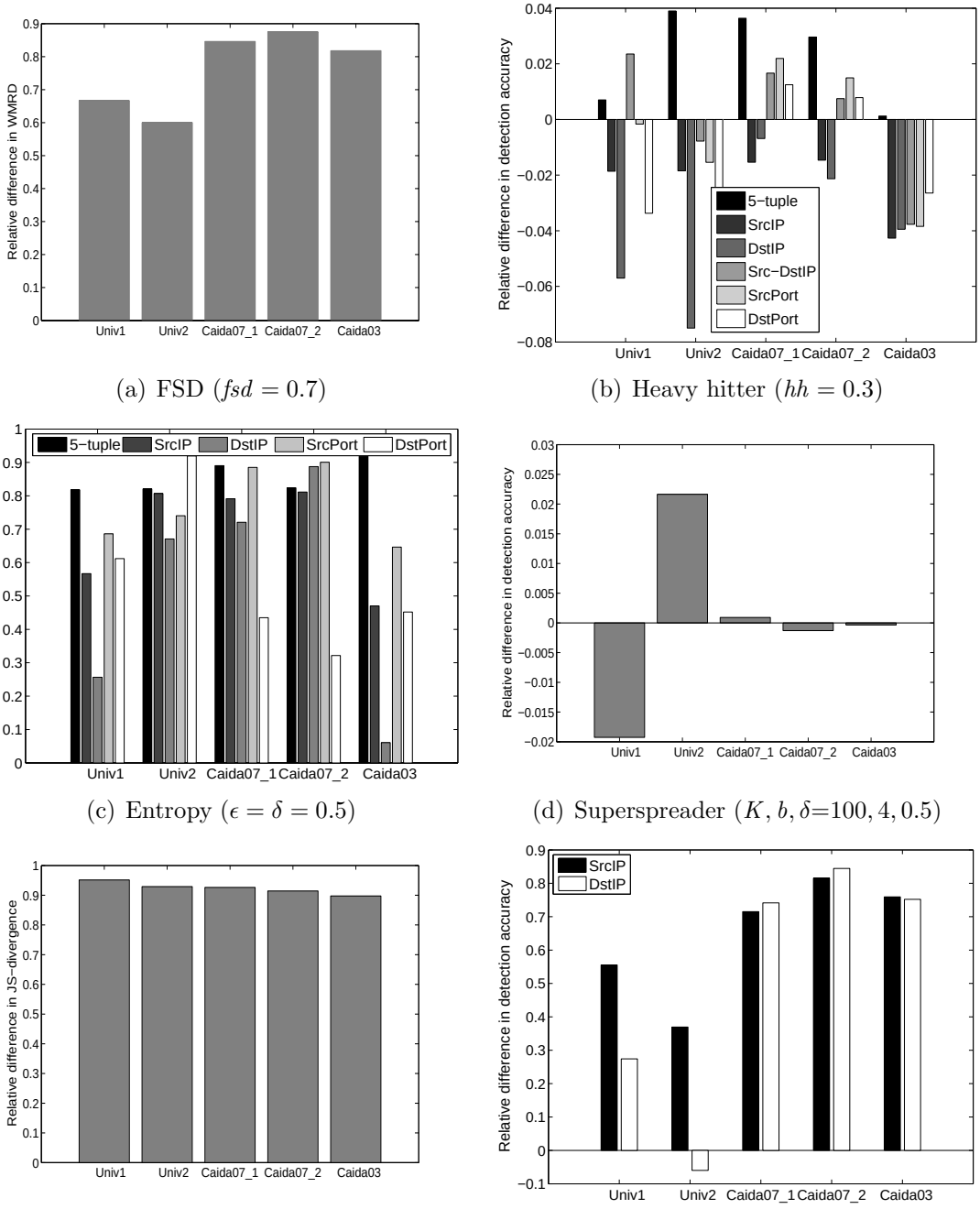
6.1.1 RISC vs. application-specific

Here, we use the default parameters from Table 1. We run the RISC approach configured with the total memory used by the six algorithms and compute the relative accuracy difference for each application.

Figure 3 shows the relative accuracy difference on the different traces. Recall that this metric is positive when the RISC approach has better accuracy and negative otherwise. These results are very promising – the RISC approach outperforms the application-specific alternative in most applications. Only heavy hitter detection (Figure 3(b)) does the RISC approach perform worse; even then the accuracy gap is at most 0.08.

Figure 3 answers a key challenge from Section 3:
The accuracy of the RISC approach is better than or comparable to the application-specific approaches.

These results show that a RISC approach provisioned with the total resources used by the six algorithms performs comparably or better than the specific algorithms. 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 RISC approach perform favorably in. For brevity, we only present the results from the Caida 2003 trace.



(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: RISC vs. application-specific approach. In each graph, a positive value of the relative accuracy indicates that the accuracy of the RISC approach was better; a negative value indicates otherwise. For most applications, the RISC approach outperforms the application-specific alternatives. In the cases where the performance is worse, it is only worse by a small relative margin.

6.1.2 Application Sensitivity

In the following experiments, we try two to three configurations for each application-specific algorithm. For each configuration, we consider an equivalent RISC approach provisioned with $G\times$ more memory resources used by the algorithm *in isolation* (i.e., we do not aggregate resources across applications). The resource magnification factor G captures the *sharing effect* across applications and configuring the RISC approach with the aggregate resources used by an application portfolio. (In the next section we explore the effect of changing the application portfolio.) As before, we focus on the relative accuracy difference between the RISC and application-specific approach.

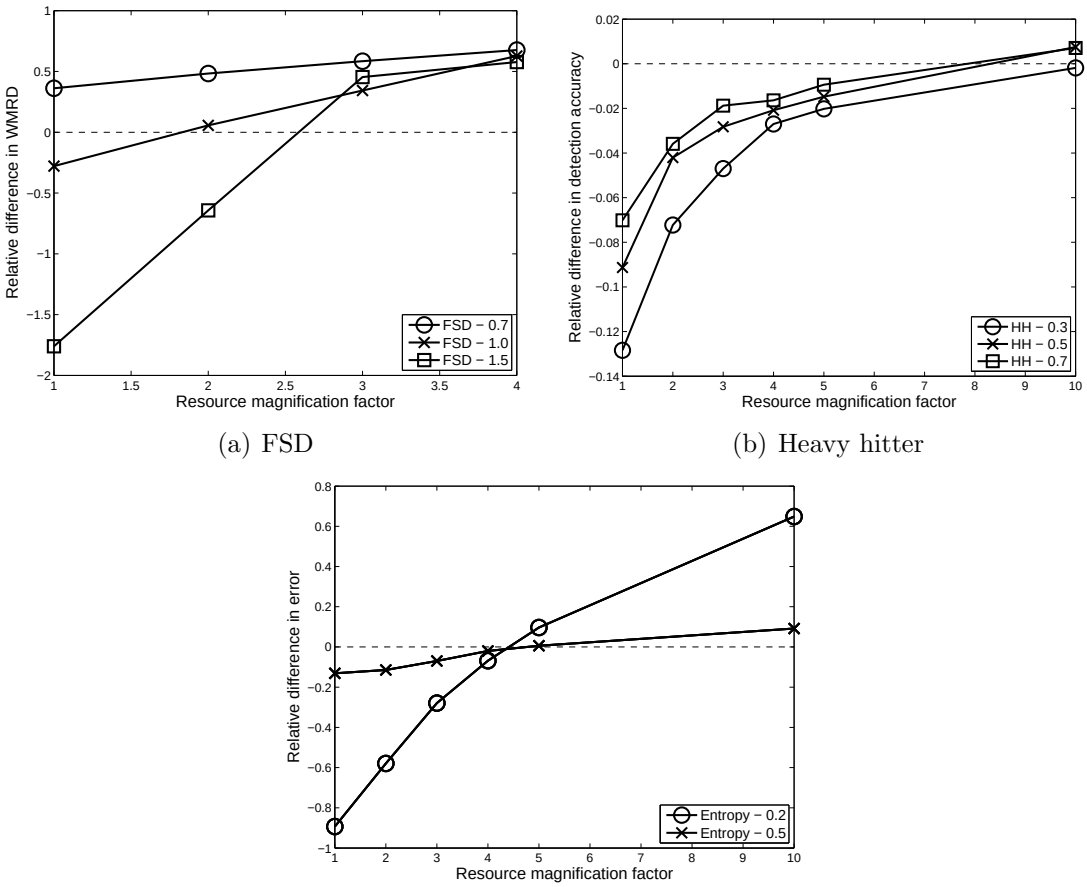
Figure 4(a) plots the relative accuracy difference between the RISC approach and the FSD estimation algorithm. We show three different configurations with the FSD algorithm using $fsd = 0.7$, 1, and 1.5. For some configurations (e.g., $fsd > 0.7$, $G \leq 2$), the RISC approach performs worse. The large negative values of the relative accuracy of RISC in these is an artifact of the low WMRD values at these points. Since we normalize the difference by the WMRD of application-specific case, the gap 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 EM algorithm was 0.02 and the WMRD for the RISC approach 0.05. Both values are small for many practical purposes [28].

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 RISC approach is indeed worse than the application-specific approach. But as G increases, the accuracy gap closes significantly. One reasons for the poor accuracy is that we configure the SH algorithm in the RISC approach to operate at the 5-tuple granularity and then subsequently project results to individual subpopulations. In fact, if we only consider the 5-tuple granularity, the RISC approach performs better. Note that in Figure 3(b), the 5-tuple is positive but the rest of are negative for the Caida 2003 dataset. However, there is some loss of accuracy during the projection phase. We could also configure the SH algorithm in the RISC approach to operate at all flowkey granularities. We tradeoff a small reduction in accuracy for a significant reduction in implementation complexity since we only need to run one instance of the algorithm on a router as opposed to 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 RISC approach performs worse. But, the gap closes as G increases and the RISC approach eventually outperforms the application-specific algorithm.

6.1.3 Sensitivity to Application Portfolio

For each portfolio, we use the default configurations from Table 1 and run the RISC approach configured with the aggregate resources contributed by this portfolio. The relative accuracy is computed with respect to these default configurations. For heavy-hitter detection and entropy estimation, we average the accuracy across the different instances.



(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 RISC approach starts to outperform the application-specific approach. The resource magnification factor captures the sharing effect of aggregating resources across applications.

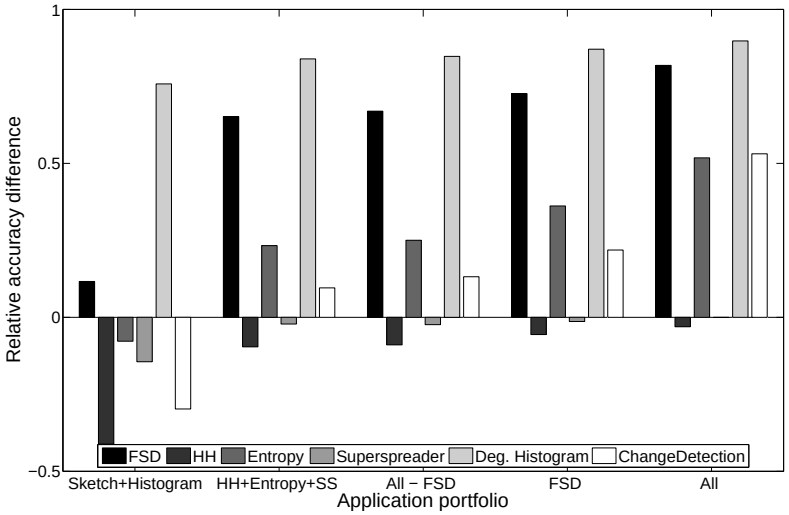


Figure 5: Effect of application portfolio on the accuracy of the RISC approach

Figure 5 shows the portfolios in increasing order of memory usage. The configuration labeled "Sketch + Histogram" uses resources only from sketch-based change detection and degree histogram estimation (the most lightweight applications.) At the other extreme, the configuration labeled "All" uses the aggregate resources (as in Figure 3).

We observe two effects. First, for larger application portfolios (i.e., as the requirements of network management applications grow), there is a clear win for the RISC approach (the relative accuracy difference becomes more positive), as the sharing effect improves the accuracy across the entire portfolio. Second, if there are some resource-intensive applications in the portfolio (e.g., FSD estimation), then it makes more sense to adopt a RISC approach since it provides improvements across the entire spectrum of applications.

6.1.4 Configuring the 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 RISC approach, but there is a diminishing returns effect. For heavy-hitter detection, expectedly, 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.

6.2 Network-wide evaluation

Dataset and Setup: We use a one hour snapshot of flow data from eleven routers from the Internet2 backbone. There are roughly 1.4 million distinct flows and 9.5 million packets

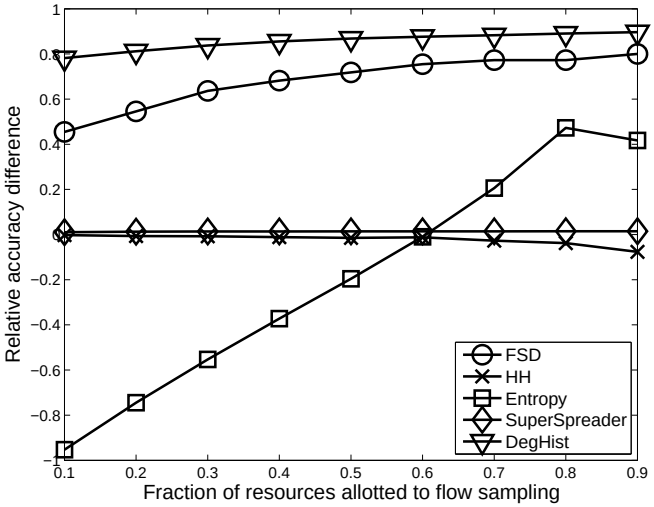


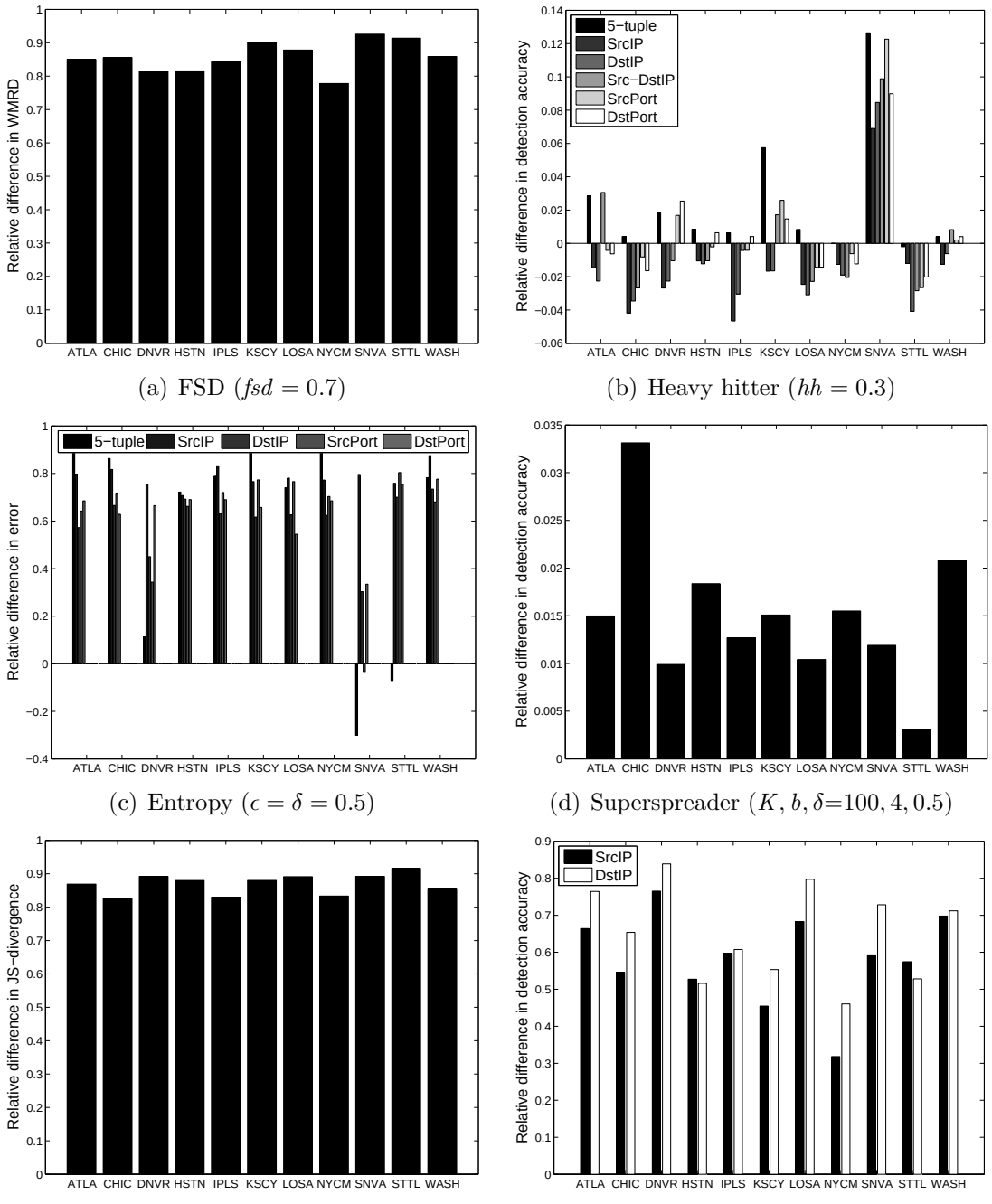
Figure 6: Varying the split between FS and SH

in aggregate per 5-minute interval. We map each flow entry to the corresponding network ingress and egress points [20]. The dataset has two limitations. First, unlike the packet traces used earlier, these are flow records with sampled packet counts (with $p = 0.01$). We assume that the sampled flow records represent the actual traffic in the network, i.e., the sampled counts are used as the actual packet counts. Second, the IP-addresses in the dataset are anonymized by zero-ing out the last 11 bits; this may affect some applications (e.g., entropy, outdegree). We ignore this effect and treat each anonymized IP address as a unique IP address. Thus, the entropy and outdegree measures are computed at this granularity. Since we are only interested in the *relative* accuracy difference, this dataset is still valuable for understanding network-wide effects. (This is the only network-wide dataset we are aware of.)

We configure each application specific algorithm on a per-ingress basis, i.e., operating on packets originating from the router. From this, we obtain the total memory usage on each router. The coordinated RISC approach from Section 4 operates on a per OD-pair granularity using the equivalent per-router memory obtained above and scaling it down by a factor of 4.

Per-ingress results: Figure 7 shows for each ingress router, the relative accuracy difference between the coordinated RISC approach and the application-specific algorithms configured per ingress. As before, a positive value indicates that the accuracy of the RISC approach was better; a negative value indicates otherwise. As with the single router evaluation, we see that the RISC approach outperforms the application-specific algorithms, except in heavy-hitter detection.

Benefits of coordination: We consider two other usage scenarios: computing the application metrics on a *network-wide* basis and on a *per OD-pair* basis. Note that the application-specific alternatives as configured for Figure 7 cannot provide per OD-pair results. They



(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: Comparing the relative accuracy difference between the coordinated RISC approach and the application-specific algorithms per ingress router. A positive value indicates that the accuracy of the RISC approach was better; a negative value indicates otherwise.

Application/Metric	App-Specific	Uncoord	Coord
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

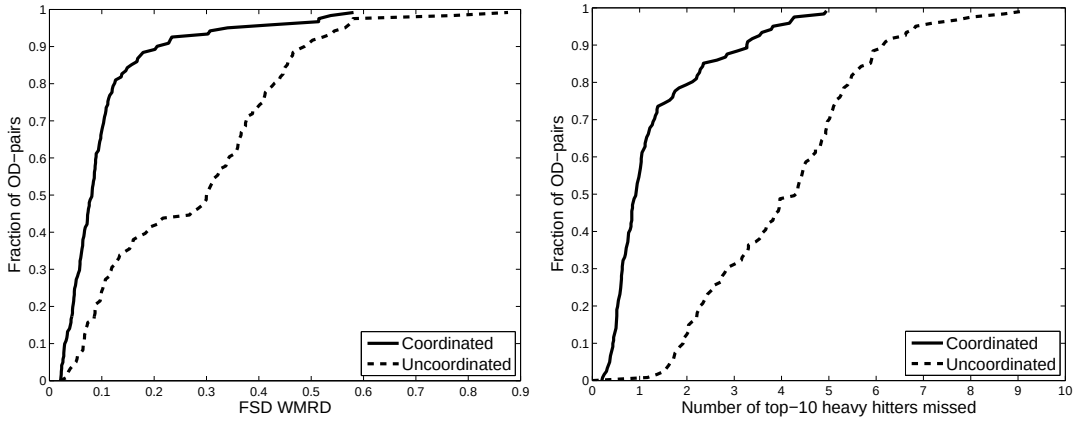
Table 3: Comparing the error rates of different approaches for network-wide metrics

work on a per-ingress basis and we cannot recover the application metrics on per-OD projections. This is not an inherent limitation of application-specific approaches; one can configure them to operate on a per-OD basis. However, this significantly increases the complexity since we need an instance per application per OD-pair.

As a point of comparison, we consider a hypothetical *uncoordinated RISC* approach: a per-router RISC approach without network-wide resource management. Each router is provisioned with the same resources as the coordinated case (i.e., the aggregate resources used by the per-ingress application-specific algorithms). The key difference is that each router independently runs these algorithms on all the traffic it sees.

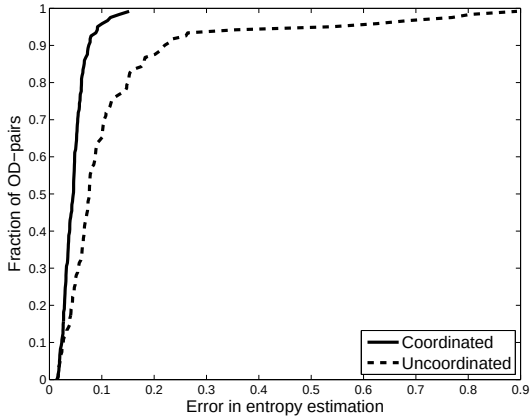
Table 3 compares the application-specific, uncoordinated, and coordinated approaches. The entry corresponding to the entropy row is empty for the app-specific column because we cannot recover the network-wide entropies from the per-ingress entropy values. There are two main observations here. First, the coordinated RISC approach has the lowest error overall. The gain in accuracy for the heavy hitter and FSD estimation applications with coordination is especially significant. Second, while the uncoordinated RISC approach is general (e.g., it can also provide per OD-pair estimates whereas the per-ingress application-specific algorithms cannot), it performs worse in the network-wide evaluation. One reason for this is that the per-ingress configuration is actually favorable to the application-specific algorithms. By construction, this *implicitly* coordinates the actions across routers by avoiding any redundancy. The uncoordinated RISC approach does not have this advantage and part of the poor accuracy can be attributed to ambiguity arising during the merging stage. An additional practical benefit of the coordinated approach is that the merging and mapping algorithms are much simpler. There is no need to identify duplicate flow records or spend extra effort in appropriate renormalization factors for the network-wide evaluation.

Finally, Figure 8 shows four accuracy metrics for the per OD-pair case. We do not consider the superspreader and change detection applications on a per OD-pair basis since they are more meaningful only with coarser aggregations. Also, we focus on the top-10 heavy hitters per OD-pair. The CDFs show that the coordinated RISC approach performs well across most OD-pairs. The 80th percentile of the WMRD, heavy-hitter miss rate, 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 inspection reveals that most of the OD-pairs where the coordinated approach has poor accuracy tend to have very low aggregate traffic per epoch

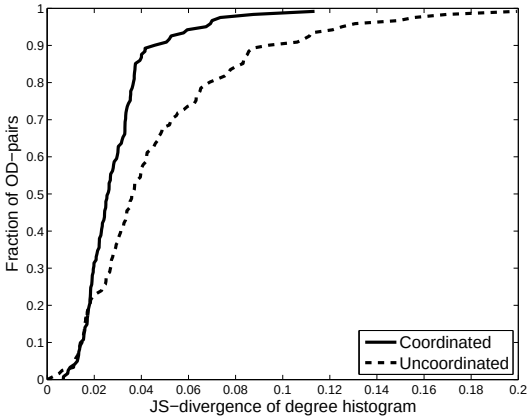


(a) FSD

(b) Heavy hitter



(c) Entropy



(d) Degree histogram

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

(not shown), which indicates that it performs well for the dominant traffic patterns. These results further confirm the benefits of network-wide coordination and resource management.

7 Discussion

Hardware feasibility: While we assumed that the implementation complexity of the different approaches is the same, not all counter updates are equal. Some application-specific algorithms require an array of counters (e.g., [27, 48]), while others (e.g., [18, 32, 45]) and the RISC primitives FS, SH [23, 18] involve key-value data structures. That said, recent proposals have demonstrated that it is possible to efficiently implement such key-value data structures in routers [22, 40]. Further, Lu et al. [33] show that it is possible to implicitly maintain such key-value pairs without much overhead using an online "counter braid" architecture and an offline decoding algorithm.

Memory overhead: We assume a 4× overhead for maintaining the key-value data structure for the flow counters involved in the RISC approach. Note that the entire flow record (the IP 5-tuple, and counters) need not actually be maintained in SRAM; only the counters for byte and packet counts need to be in SRAM. Thus, we can offload most of the flow fields to DRAM and retain only those relevant to the online computation. Suppose we assume that each counter for the application-specific algorithms is 2 bytes wide [50]. Experiments with a sparse hash map data structure in software showed that we can keep 1 million 2-byte flow counters with a total memory requirement of 8 MB. Further, with techniques such as counter braids, the memory overhead will be even lower; maintaining 1 million per-flow counters requires only 1.4 MB of memory [33]. Thus, the 4× overhead factor is quite conservative.

Bandwidth overhead for data collection: A natural concern is the bandwidth overhead for transferring flow records from routers to the network operations center. We give a simple back-of-envelope calculation to estimate a worst-case overhead. In the Internet2 dataset, we observe on average 1.7GB of flow data per PoP per day. Accounting for the sampling rate of 0.01, this conservatively translates into 170 GB per PoP per day or 0.6GB per five minutes. (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 is shipped 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 linerates of 10 Gbps today, it is not unreasonable to expect ISPs to use 0.16% of the network bandwidth per-PoP for measurement traffic to aid network management.

Processing overhead: There are two processing components in the RISC approach: online collection and offline computation. By construction, the online collection overhead of a RISC approach is lower. In the application-specific architecture, each packet requires as many counter updates as the number of application instances. The RISC approach each packet requires only two updates, one each for FS and SH.

With respect to the offline computation, we currently assume that it is possible to run exact algorithms on the collected flow data to provide near real-time estimates. However,

the exact computation is not strictly necessary. Once we collect the flow measurements, the RISC approach can accommodate multiple computation modes – either an exact mode if its feasible or the same application-specific streaming algorithms if not.

Adaptivity: Another natural question is how does the RISC approach deal with network dynamics and adversarial traffic conditions. Keys et al. discuss how to adapt the single-router primitives (FS, SH) to changing traffic conditions [25]. Similarly, Sekar et al. discuss how cSamp can adapt to network dynamics or deal with estimation errors in inputs [42]. The RISC approach can leverage these techniques as well.

8 Conclusions

This paper is a reflection on recent trends in network monitoring. There is a growing demand for a wide variety of high-fidelity traffic estimates 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 articulate the case for a RISC architecture for flow monitoring. A RISC architecture dramatically reduces the implementation complexity of monitoring elements; enables router vendors and researchers to focus their energies on building efficient implementations of a small number of primitive operations; and allows late binding to what traffic metrics are important, thus insulating router implementations from the changing needs of flow monitoring applications.

As a starting point, we showed that a simple combination of existing primitives – flow sampling, sample and hold, and cSamp– already provides significant benefits across a wide spectrum of applications. However, by no means is this solution perfect or comprehensive. We hope that our work spurs the research community to embark on a quest for better application-agnostic primitives and efficient implementations of such primitives.

References

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

[2] Ipmon - packet trace analysis. `http://ipmon.sprintlabs.com/packstat/packetoverview.php`.

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

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

[5] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. STOC*, 1996.

[6] H. Ballani and P. Francis. CONMan: A Step Towards Network Manageability. In *Proc. ACM SIGCOMM*, 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] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954.

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

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

[13] D. Moore, K. Keys, R. Koga, E. Lagache, and k. claffy. CoralReef software suite as a tool for system and network administrators. In *Proc. LISA*, 2001.

[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] N. Duffield, C. Lund, and M. Thorup. Learn more, sample less: Control of volume and variance in network measurement. *IEEE Transactions in Information Theory*, 51(5):1756–1775, 2005.

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

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

[19] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical Approaches to DDoS Attack Detection and Response. In *Proc. DARPA Information Survivability Conference and Exposition*, 2003.

[20] 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.

[21] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Meyers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM CCR*, 35(5), Oct. 2005.

[22] 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.

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

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

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

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

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

[28] 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.

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

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

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

[32] 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.

[33] 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.

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

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

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

[37] S. Muthukrishnan. Data streams: algorithms and applications. `http://athos.rutgers.edu/~muthu/stream-1-1.ps`.

[38] D. A. Patterson and D. R. Ditzel. The case for the reduced instruction set computer. *ACM SIGARCH Computer Architecture News*, 8(6):25–33, Oct. 1980.

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

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

[41] 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.

[42] 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.

[43] M. R. Sharma and J. W. Byers. Scalable Coordination Techniques for Distributed Network Monitoring. In *Proc. PAM*, 2005.

[44] V. Sekar, A. Gupta, M. K. Reiter and H. Zhang. Coordinated Sampling sans Origin-Destination Identifiers: Algorithms, Analysis, and Evaluation. Technical Report CMU-CS-09-104, Department of Computer Science, Carnegie Mellon University, 2009.

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

[46] 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.

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

[48] 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.

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

[50] 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.