

A Case for a RISC Architecture for Network Flow Monitoring

Vyas Sekar*, Michael K. Reiter†, Hui Zhang*
* Carnegie Mellon University, † UNC Chapel-Hill

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 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 [18], accounting [13, 17], anomaly detection [27, 28], identifying and understanding end-user applications [10, 21], understanding traffic structure at various granularities [43], detecting worms, scans, and botnet activities [44, 41, 36], and forensic analysis [42]. These require high-fidelity estimates of traffic metrics relevant to each application.

High traffic rates 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. The de-facto standard for flow-level monitoring is NetFlow [9] and similar implementations from other router vendors (e.g., [3]), which are intended as application-agnostic primitives. These employ packet sampling where 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 [33, 20, 14, 25, 6, 36, 17]). One consequence is that the research community has been focused almost

exclusively on developing application-specific sampling strategies. This is exemplified by the proliferation of a wide range of data streaming algorithms for computing different specific traffic metrics, e.g., flow size distribution [25], entropy [29], superspreader detection [41], degree histogram estimation [44], change detection [24], and several other applications in the literature.

While this body of work has made valuable algorithmic contributions, this shift toward application-specific approaches is undesirable in practice for two reasons. First, the set of network management 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. Second, a large number of monitoring primitives operating independently greatly increases the implementation complexity and resource requirements of routers — each application-specific algorithm requires separate processing and memory resources for maintaining relevant data structures.

In this paper, we reflect on these trends in flow monitoring and answer a fundamental question:

Is such complexity necessary for flow monitoring or can a simpler “RISC” approach [35], that has a small number of primitives, provide performance comparable to application-specific approaches?

A RISC approach would employ a small number of simple, generic collection primitives — analogous to a reduced instruction set [35] — on each router and manage these in an intelligent network-wide fashion, to ensure that the collected data will support computation of metrics of interest to various applications. There are two qualitative benefits of this approach: it enables “late binding” of which application-specific traffic metrics to consider and it enables vendors to develop efficient hardware implementations of a few primitives.

The key insight is that a RISC approach decouples 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. The RISC approach, on the other hand,

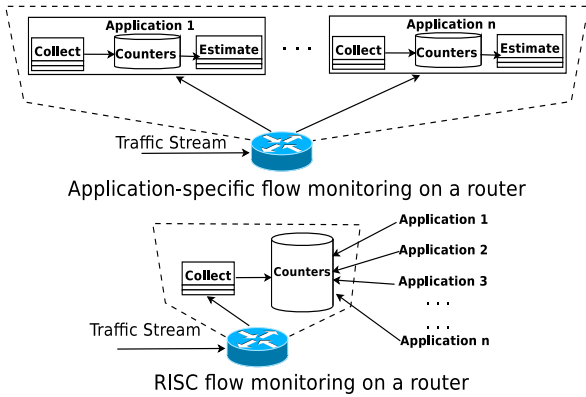


Figure 1: Schematic comparison between the application-specific architecture and the RISC architecture. The application specific architecture runs multiple algorithms each with its associated collection and computation components. The RISC architecture runs a small number of generic collection algorithms.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

collects generic data that can be used for multiple applications. The applications can use offline computation devices that need not strictly work at line rates.

One rationale to suggest that a RISC approach can perform favorably compared to application-specific alternatives is that the primary bottleneck for high-speed monitoring is maintaining counters 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, the RISC approach has an advantage.

We describe one such RISC architecture that is practical and provides sufficient fidelity for a broad class of applications. For single-router sampling algorithms, we leverage sample-and-hold [17] and flow sampling [20]. For network-wide management we use Coordinated Sampling (CSAMP) [39]. 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 [17], detecting superspreaders [41], computing the entropy of different traffic subsets [29], estimating the flow size distribu-

tion [25], computing the outdegree histogram for detecting stealthy spreaders [44], and change detection using sketches [24]. 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, aggregates the sampled packets into flow reports, and exports the flow reports. Packet sampling at a low sampling rate p (e.g., $p \leq 0.01$) has low overhead requiring only DRAM counters [9], and can support applications such as traffic engineering and accounting (e.g., [15, 14, 18]). Researchers have proposed strategies to adapt the sampling rate to changing traffic conditions, or to tune the processing, memory, and reporting bandwidth overheads [16, 23]. There have also been efforts to get better traffic estimates from sampled measurements [13, 15].

However, packet sampling is known to have several inherent limitations. For example, there are known biases towards sampling larger flows (e.g., [20, 25, 33]). Further, several studies have questioned the fidelity of packet sampling for many network management applications (e.g., see [33, 20, 14, 25, 6, 36, 17]).

The inadequacy of packet sampling has motivated a large number of application-specific data streaming algorithms and proposals for flexible sampling algorithms (both application-specific and application-agnostic). Further, there is a growing body of work that demonstrates the need for network-wide sampling solutions. We briefly describe these three classes of related work next.

Application-specific data streaming algorithms: Several application-specific data streaming algorithms and counting data structures have been proposed; see [34] for a survey. The high-level approach is to use a small number of (SRAM) counters pertaining to the specific metric of interest and subsequently use an estimation algorithm to recover traffic statistics from these counters. The seminal work of Alon et al. [5] provides 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 [25]. Streaming algorithms have also been proposed for identifying heavy hitters [17] and for computing traffic distribution statistics such as entropy [29]. Sketch-based techniques have been used for detecting changes and anomalies [24]. However, these lack the generality to serve as primitive operations on routers.

There are some efforts in designing data structures that allow a variety of metrics to be computed effi-

ciently. Notably, count-min sketches [11] can estimate 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, super-spreader 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 (srcip, dstip, srport, dstport, protocol). Each such flowkey of interest requires a separate instance of the sketch on a router. 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. For example, in size-dependent sampling [26, 36], the sampling rate depends on the flow size. Other approaches allow network operators to define specific flow categories and only log flows relevant to these categories (e.g., [45, 1, 4, 32, 7]). These approaches can serve as additional primitives for a RISC approach. However, they need to be configured with the categories and sampling rates to suit the application requirements in advance. 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 be 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. [22]. They design a system for providing summaries of global traffic counters and “resource hogs”. To do so, they use a combination of flow sampling [20] and sample-and-hold [17], similar to our approach in Section 4. Our work extends theirs in two significant ways, however. First, we take a network-wide view 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 this hybrid approach can in fact support a much wider range of applications.

Network-wide sampling: Recent work has stressed the importance of network-wide measurements, e.g., to meet operational requirements [18, 27] or for anomaly detection [27, 38, 28]. In this theme, Cantieni et al. [8] provide a formulation to optimally set the packet sampling rates on a set of routers to achieve traffic engi-

neering objectives. Our architecture uses CSAMP [39], to ensure that the available monitoring resources are used in an efficient, non-redundant manner.

3. DESIGN CONSIDERATIONS

Given this background, we synthesize key requirements for a RISC architecture for flow monitoring and also derive some guiding design principles.

3.1 Requirements

Generality across applications: There is 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* or provide sufficient coverage over a large class of applications.

Low complexity: Due to technological and resource constraints on routers, *sampling* 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. This network-wide view is crucial for several operational requirements [18, 27] and as user applications and attacks become massively distributed [38, 27, 28, 10]. For example, understanding the structure of peer-to-peer traffic [10], detecting botnets [36] and hit-list worms [31], understanding DDoS attacks [38], and network forensics [42] inherently require a network-wide view aggregated from multiple vantage points.

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 such as 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 implicit in the operational model of ISPs using NetFlow, export-

ing flow reports to a central collection center, and running analysis/diagnostics on the flow 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: The monitoring primitives 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 commensurate to that consumed by the application-specific alternatives in *aggregate*, will perform similar to the application-specific algorithms. In addition to retaining independence from today’s applications, this also improves the likelihood of supporting future applications.

Network-wide management: A network-wide approach will need to take into account (a) the resource constraints (e.g., memory) 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 should assign monitoring responsibilities to routers that both respect each router’s resource constraints and effectively utilize these resources toward network-wide goals.

3.3 Challenges and Roadmap

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 divided across routers to meet an 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 describe our RISC architecture in the next section to answer the first question. Then, we compare this specific RISC design against six applications and their associated application-specific algorithms.

4. ARCHITECTURE

Our RISC architecture combines three ideas: flow sampling [20], sample-and-hold [17], and CSAMP [39].

4.1 Single router sampling

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, traffic engineering) and (2) those that depend on the *communication structure* of the traf-

fic (e.g., network security applications, anomaly detection). Our choice of primitives is guided by these two broad classes. Flow sampling is well suited for security and anomaly detection applications that depend on understanding communication structure, i.e., “who talks to whom” [20, 33, 31]. Similarly, sample-and-hold is well suited for traffic engineering and accounting applications that depend on volume estimates [17].

For the following discussion, a flow refers to the IP 5-tuple: $\langle srcaddr, dstaddr, sport, dport, protocol \rangle$. We use flow sampling and sample-and-hold at this 5-tuple granularity. The rationale is to collect 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 sample-and-hold [17] for tracking heavy hitters, i.e., flows with large packet counts. While packet sampling can detect heavy hitters, the estimation errors are quite high. The motivation for SH is to keep near-exact counts of the heavy hitters. We briefly describe how SH works below.

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 y , and y has not been selected earlier, the router keeps an exact count for y subsequently. Since this might require per-packet counter updates, the counters are maintained in SRAM [17].

To configure SH, we specify the flowkey (e.g., sport, srcaddr, or 5-tuple), the total number of packets a router will see in a specific time interval ($numpkts$), and the total memory resources available (L). The probability p is set to $\frac{L}{numpkts}$.¹ Instead of using a separate instance for each possible flowkey, we use one instance defined on the full IP 5-tuple.

Hash-based flow sampling (FS): The key idea behind flow sampling (FS) is to pick flows rather than packets independently at random. One possible implementation of FS 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

¹If the goal is to track heavy hitters who contribute more than a fraction $\frac{1}{x}$ to the total volume, then the probability 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 .

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 router for each category of flows in the sampling manifest. The above approach implements flow sampling [20], since only those flows whose hash lies within the hash range are monitored.

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 [23]. 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. We show in Section 6.1.4 that $f \approx 0.8$ is a good choice.

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 [8, 39].

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, because SH logs heavy hitters, the same set of heavy hitters will be reported across routers on the same path. Thus, replicating SH functions across routers on a path will result in duplicate measurements and waste memory resources.

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 the routers (both ingress and non-ingress), we use CSAMP [39] 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 op-

timal 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. Each sampling manifest 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 ($i = 1, \dots, M$) 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).³ Each router R_j ($j = 1, \dots, N$) 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 $i = 1, \dots, M$, let C_i denote the fraction of flows on OD_i that is logged.

CSAMP can support a variety of network-wide objectives, though here we describe its use for one in particular, namely 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 (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

²CSAMP can be reformulated to not require a router to determine the OD-pair for each packet [40], though here we describe the simpler approach using OD-pairs.

³For simplicity, we assume that each OD-pair has one route, though CSAMP accommodates multi-path routing [39].

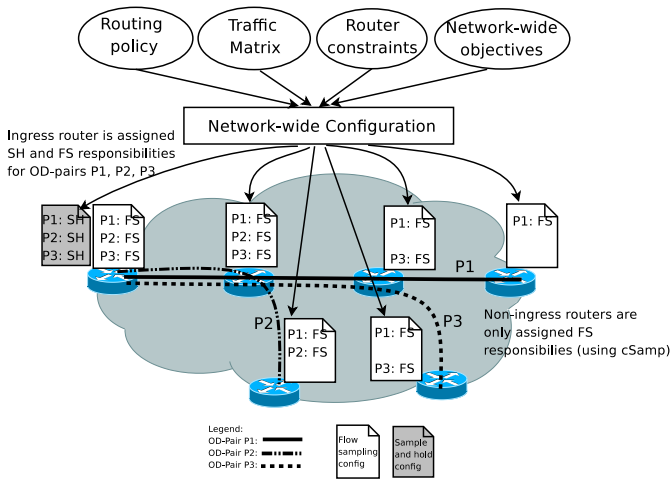


Figure 2: Example of a network-wide configuration for the RISC approach

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.

$$\text{Maximize } \sum_i (T_i \times C_i), \quad \text{subject to} \quad (1)$$

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

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

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

$$\forall i, \quad \alpha \leq C_i \leq 1 \quad (4)$$

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 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

Overview: The goal of our evaluation is to compare

the performance of the RISC approach against application-specific approaches when their total resource consumptions are suitably normalized. In order to do so, we need to specify the different applications of interest, the application-specific algorithms, and the configurations for determining the amount of resources provisioned for each algorithm and the RISC approach.

This section describes the applications and configurations we use in our evaluations. For each application, we first describe the corresponding data streaming algorithms, i.e., 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.

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 $\forall l = 1 \dots z, \phi_l = \frac{F_l}{F}$, 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 [14, 25]. We use the data streaming and expectation-maximization algorithm proposed by Kumar et al. [25].

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 [25]. 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 [12, 2] as well as for traffic engineering and accounting [17].

We use the SH algorithm [17] described earlier. We configure it to run with six instances, one each for the six 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*: get the top k exact heavy hitters

Application	Accuracy/Error Metric	Algorithm	Parameters (defaults)
FSD estimation (5-tuple)	WMRD	[25]	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	[29]	ϵ, δ (0.5, 0.5)
Superspreader detection	Detection accuracy	[41]	K, b, δ (100, 4, 0.5)
Change detection (sip, dip)	falsepos + falseneg	[24]	h, k, θ (10, 1024, 0.05)
Deg. histogram estimation	JS-divergence	[44]	–

Table 1: Summary of applications, accuracy metrics, algorithms, and default parameters. In the first column, the attributes in parentheses specify the *flowkeys* over which the application is run (e.g., FSD estimation uses the IP 5-tuple; heavy-hitter detection uses 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 implies that any host contacting $\geq K$ distinct destinations is a superspreader; and flagging any host contacting $\leq \frac{K}{b}$ distinct destinations is a false positive. h is the number of hash functions and k is the number of counters per hash function in the sketch data structure and θ is the change detection threshold.

and the top k estimated heavy hitters and compute the set intersection between these two sets. The RISC approach also uses SH; the difference is that it uses only one instance at the 5-tuple granularity and uses offline projections to other dimensions.

Entropy estimation: The entropy of traffic distributions (e.g., packets per destination port) has been shown to aid monitoring applications such as anomaly detection [28] and traffic classification [43]. The motivation for entropy-based analysis is that it can capture fine-grained properties that cannot be understood with simple volume-based analysis. For these applications, 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 present in a given measurement epoch [28] and $H(X) = -\sum_{i=1}^N Pr(x_i) \log_2(Pr(x_i))$ is the entropy of the random variable X , where x_1, \dots, x_N is the range of values for X , and $Pr(x_i)$ represents the probability that X takes the value x_i .

We use the data streaming algorithm proposed by Lall et al. [29]. We consider five traffic distributions 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. [41]. 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 report the false positive rate since it was zero for the RISC and application-specific approaches in almost all cases.

Change detection: Change detection is an important component of anomaly detection for detecting DDoS attacks, flash crowds, and worms [24]. The problem of change detection can be formally described as follows. Suppose we discretize the traffic stream into five-minute measurement epochs ($t = 1, 2, \dots$). Let each $I_t = \beta_1, \beta_2, \dots$ 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., number of bytes in the i^{th} packet or simply 1 if we are interested in counting packets). Let $Obs_y(t) = \sum_{i:y_i=y} c_i$ denote the aggregate observed count for flowkey y in epoch t . Let $Fcast_y(t)$ denote the forecast value (e.g., computed using exponentially weighted moving average) for item y in epoch t . The forecast error for y then is $Err_y(t) = Obs_y(t) - Fcast_y(t)$. Let $F2Err_t = \sum_y Err_y(t)^2$ denote the second moment of the forecast errors. The goal of the change detection is to detect all y 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 that 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. [24] because sketches have a natural “linearity” property that makes them well-suited for change detection. We use an EWMA model $Fcast(t) = \alpha Obs(t) + (1 - \alpha) Fcast(t - 1)$, with $\alpha = 0.9$. Note that since we are only interested in the relative performance of the RISC vs. sketch-based approaches, the specific forecast model we use is not

important. We consider two instances for identifying changes in (1) the number of packets per source address and (2) the number of packets per destination address.

Degree histogram estimation: Finally, we consider the problem of computing the *outdegree histogram* of a traffic stream. The outdegree of a source IP is the number of distinct destination IPs it contacts in a 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. A specific application is to detect botnets involved in coordinated scans [44] by detecting changes in the outdegree histogram; the outdegree distribution might also be independently useful for understanding traffic structure. We use the “sampling algorithm” proposed by Gao et al. [44].

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

5.2 Assumptions and Approach

In order to compare the RISC and application-specific approaches, we need to normalize the total resource usage in each case. To do so, we make three *conservative* assumptions shown below. We realize and acknowledge the difficulty in exactly normalizing the resource usage and implementation costs of the different algorithms. In Section 7, we discuss why these assumptions are reasonable and conservative in that they underestimate the performance of a RISC approach compared to application-specific algorithms.

- 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.
- All approaches can use offline computation resources. In the RISC approach, this means that we run estimation algorithms over the collected flow records without further sampling.

⁴Gao 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.

5.2.1 Configuring the application-specific algorithms

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. [25], 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. [29] 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; k , the size of the counter array per hash function; and the detection threshold θ . Following Krishnamurthy et al. [24], we set $h = 10$, $k = 1024$, and $\theta = 0.05$.
6. For degree histogram estimation, we use the same configuration as Gao et al. [44].

5.2.2 Configuring the RISC approach

The RISC approach has two configuration parameters per router: the number of flow records it can collect (L) and, for ingress routers, 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 $L_{app-spec}$ 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 on each ingress router.

5.2.3 Computing estimates in the RISC approach

We take the union of the flow records reported by SH (after normalizing the packet counts by the sampling rate [17]) and the flow records reported by FS. If the same flow record is reported by both FS and SH, we use the FS record because the packet count in FS is exact whereas the count reported by SH is approximate. We compute the FSD, entropy, and detect heavy hitters or changes per-source (or destination) on this merged set of flow records. Additionally, we logically retain the set of FS records alone. We use this set for detecting superspreaders and computing the degree histogram.

Note that, since the RISC approach exports the actual flow records, it is possible to run offline estimation

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

procedures on these flow records to compute any application metric, even unforeseen ones.

5.2.4 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 computing 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. TRACE-DRIVEN EVALUATIONS

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:

- 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 parameters 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*)? That is, 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 Accuracy: RISC vs. application-specific

For this comparison, we use the default parameters from Table 1, run the RISC approach configured with

⁵Some 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.

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 performs better and negative otherwise.

The result shows that the RISC approach outperforms the application-specific alternative in most applications. Only in heavy-hitter detection (Figure 3(b)) does the RISC approach perform worse; even then the accuracy gap is at most 0.08. This answers the second challenge from Section 3:

The RISC approach provisioned with the total resources used by the six application classes 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 RISC 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 two to three configurations for each application-specific algorithm. For each configuration, we consider a RISC approach provisioned with $G \times$ as much memory as that used by the algorithm *in isolation*. The magnification factor G captures the *sharing effect*—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 $fsd = 0.7, 1, \text{ and } 1.5$. 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 the 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 [25].

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

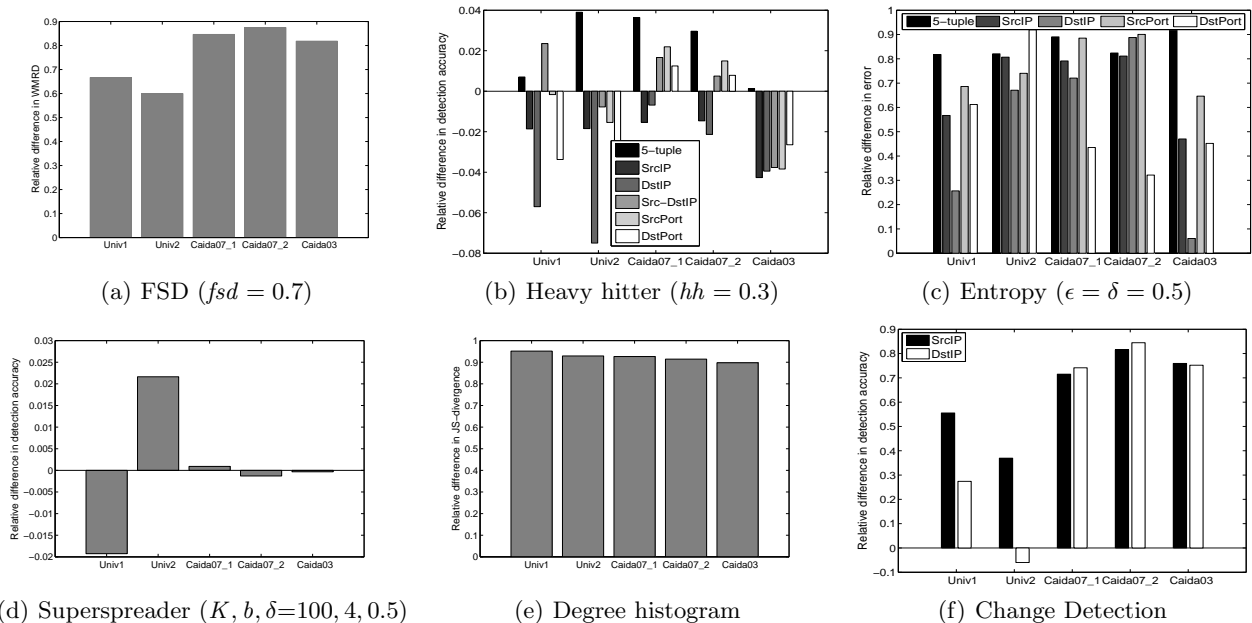


Figure 3: RISC vs. application-specific approach. In each graph, a positive value of the relative accuracy indicates that the accuracy of the RISC approach was better; a negative value indicates otherwise. For most applications, the RISC approach outperforms the application-specific alternatives. In the cases where the performance is worse, it is only worse by a small relative margin.

reason 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, the RISC approach performs better if we only consider the 5-tuple granularity, but it performs worse for the other flowkeys due to some loss of accuracy in the projection phase (see Figure 3(b)). 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.

Figure 5 shows the portfolios in increasing order of memory usage. The configuration labeled “Sketch + Histogram” uses resources only from sketch-based change

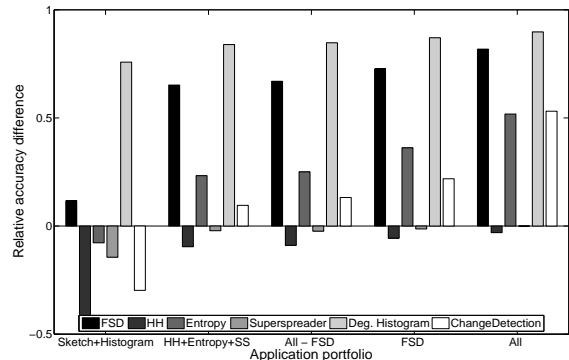


Figure 5: Effect of application portfolio on the accuracy of the RISC approach

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 $f = 0.8$. Fig-

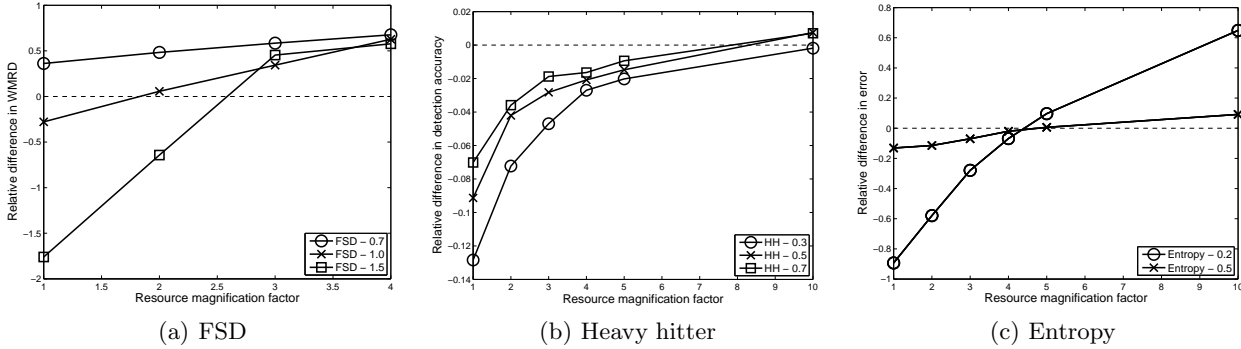


Figure 4: Exploring the sensitivity of applications in isolation. The zero line represents the point at which the RISC approach starts to outperform the application-specific approach. The resource magnification factor captures the sharing effect of aggregating resources across applications.

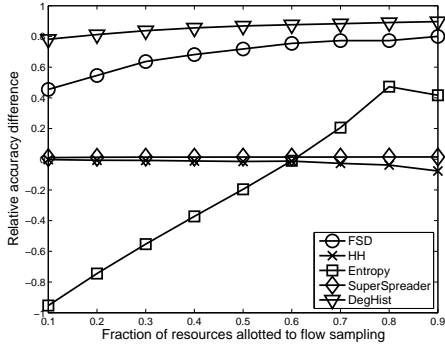


Figure 6: Varying the split between FS and SH

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, 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.

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]. 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 work on a per-ingress basis and we cannot compute the application metrics on per-OD projections. This is not an inherent limitation of application-specific approaches; we can configure them 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 an *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., add up the resources used by the per-ingress application-specific algorithms and divide by 4). The key difference is that each router independently runs the RISC algorithms on 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 application-specific column because we cannot recover the network-

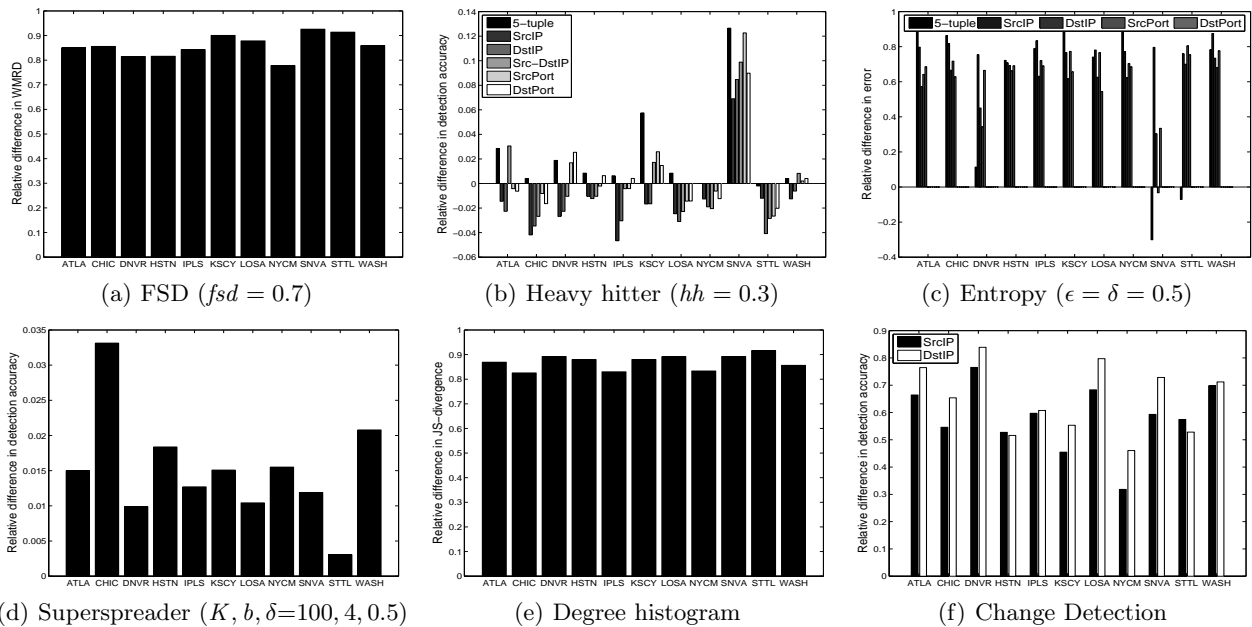


Figure 7: Comparing the relative accuracy difference between the coordinated RISC approach and the application-specific algorithms per ingress router. A positive value indicates that the accuracy of the RISC approach was better; a negative value indicates otherwise.

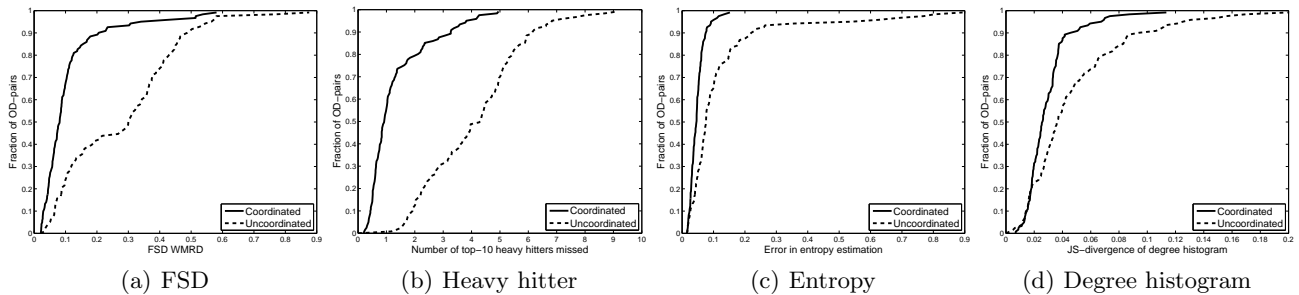


Figure 8: Comparing the coordinated and uncoordinated approaches on a per-OD basis.

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

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 for the application-specific algorithms implicitly coordinates routers avoiding redundancy when we merge the results for the network-wide case. The uncoordinated RISC approach does not have this advantage

and multiple sources of ambiguity and bias arise when we try to merge flow reports across multiple routers. For example, routers may have different flow sampling rates as they see different traffic volumes, flows that traverse longer paths get higher sampling rates resulting in some bias, and large flows are reported multiple times by SH. An additional practical benefit of the coordinated approach is that the merging and estimation algorithms are much simpler and more accurate.

Finally, Figure 8 shows four accuracy 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 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, 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.

6.3 Summary of main results

- The accuracy of the RISC 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 RISC 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 RISC approach gives greater accuracy in projecting results to different spatial aggregations compared to uncoordinated and application-specific approaches.

7. DISCUSSION

Hardware feasibility: Some application-specific algorithms require an array of counters (e.g., [24, 44]), while others (e.g., [17, 29, 41]) and the RISC primitives FS, SH [20, 17] 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 [19, 37]. Further, Lu et al. [30] show that it is possible to implicitly maintain such key-value pairs with low overhead using an online “counter braid” architecture and an offline decoding algorithm.

Memory overhead: 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 [30, 39].

We assume a $4\times$ overhead for maintaining flow counters as key-value pairs in SRAM for the RISC approach compared to the application-specific approaches. We justify why this $4\times$ factor is *conservative*.

1. Notice that some application-specific algorithms also require key-value counters—we conservatively assume that these incur no memory overhead compared to an array of counters.
2. Suppose each counter for the application-specific algorithms is 2 bytes [46]. We ran experiments with a sparse hash data structure and found that it can store 10^6 flow counters in 8 MB, i.e., 8 bytes per counter. In other words, a *simple, software only* implementation has just $\frac{8}{2} = 4\times$ overhead.
3. With smarter hardware for storing flow counters such as counter braids [30], the overhead will be even lower—maintaining 1 million flow counters requires 1.4 MB of memory, i.e., $\frac{1.4}{2} < 4\times$.

Bandwidth overhead for data collection: A natural concern is the bandwidth overhead for transferring flow records from routers to a network operations center. We give a back-of-the-envelope calculation to estimate the worst-case overhead. The Internet2 dataset has roughly 1.7GB of flow data per PoP per day. Given a 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 $\frac{0.6 \times 8 \text{ Gbits}}{300 \text{ seconds}} = 0.016 \text{ Gbps}$. Given OC-192 backbone line rates 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. With the RISC approach, each packet requires only two updates, one for FS and one for SH.

With respect to the offline computation, we currently assume that it is possible to run exact estimation algorithms on the collected flow data (i.e., without further sampling) to provide near real-time estimates. However, this 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 use application-specific streaming algorithms if necessary.

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 [22]. Similarly, Sekar et al. discuss how CSAMP can adapt to network dynamics or deal with estimation errors in inputs [39]. 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 revisit the case for a RISC architecture for flow monitoring. A RISC architecture dramatically reduces the

complexity of routers; enables router vendors and researchers to focus their energies on building efficient implementations of a small number of primitives; 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 RISC architecture combining flow sampling, sample-and-hold, and CSAMP performs favorably across a wide spectrum of applications compared to application-specific approaches. However, we do not claim that this is the “optimal” RISC architecture. We believe that there is great scope for designing better RISC primitives, implementing better techniques for merging data collected from multiple RISC primitives, developing better models for reasoning about application performance using such primitives, and providing efficient router implementations of these primitives. We hope that our work spurs the community to focus on these directions.

9. 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] D. Brauckhoff, B. Tellenbach, A. Wagner, A. Lakhina, and M. May. Impact of Traffic Sampling on Anomaly Detection Metrics. In *Proc. IMC*, 2006.
- [7] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proc. ACM SIGMOD*, 2003.
- [8] 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.
- [9] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954.
- [10] M. P. Collins and M. K. Reiter. Finding Peer-to-Peer File-sharing using Coarse Network Behaviors. In *Proc. ESORICS*, 2006.
- [11] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55, 2005.
- [12] 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.
- [13] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *Proc. IMW*, 2001.
- [14] N. Duffield, C. Lund, and M. Thorup. Estimating Flow Distributions from Sampled Flow Statistics. In *Proc. of ACM SIGCOMM*, 2003.
- [15] 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.
- [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] 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.
- [20] N. Hohn and D. Veitch. Inverting Sampled Traffic. In *Proc. IMC*, 2003.
- [21] T. Karagiannis, D. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *Proc. ACM SIGCOMM*, 2005.
- [22] K. Keys, D. Moore, and C. Estan. A Robust System for Accurate Real-time Summaries of Internet Traffic. In *Proc. SIGMETRICS*, 2005.
- [23] R. Kompella and C. Estan. The Power of Slicing in Internet Flow Measurement. In *Proc. IMC*, 2005.
- [24] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proc. ACM IMC*, 2003.
- [25] 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.
- [26] A. Kumar and J. Xu. Sketch Guided Sampling – Using On-Line Estimates of Flow Size for Adaptive Data Collection. In *Proc. IEEE Infocom*, 2006.
- [27] A. Lakhina, M. Crovella, and C. Diot. Diagnosing Network-Wide Traffic Anomalies. In *Proc. ACM SIGCOMM*, 2004.
- [28] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc. ACM SIGCOMM*, 2005.
- [29] 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.
- [30] 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.
- [31] M. P. Collins and M. K. Reiter. Hit-list Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. In *Proc. RAID*, 2007.
- [32] H. Madhyastha and B. Krishnamurthy. A Generic Language for Application-Specific Flow Sampling. *ACM CCR*, 38(2):7–15, Apr. 2008.
- [33] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang. Is Sampled Data Sufficient for Anomaly Detection? In *Proc. IMC*, 2006.
- [34] S. Muthukrishnan. Data streams: algorithms and applications. <http://athos.rutgers.edu/~muthu/stream-1-1.ps>.
- [35] 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.
- [36] A. Ramachandran, S. Seetharaman, and N. Feamster. Fast Monitoring of Traffic Subpopulations. In *Proc. IMC*, 2008.
- [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, Analysis, and Evaluation. Technical Report CMU-CS-09-104, Carnegie Mellon University, 2009.
- [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.