Revisiting the Case for a Minimalist Approach for Network Flow Monitoring

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

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

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

ABSTRACT

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

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

Categories and Subject Descriptors

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

General Terms

Measurement, Management

Keywords

Traffic Monitoring, Sampling, Data Streaming, Anomaly Detection

IMC'10, November 1–3, 2010, Melbourne, Australia.
Copyright 2010 ACM 978-1-4503-0057-5/10/11 ...$10.00.

1. INTRODUCTION

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

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

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

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

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

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

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

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

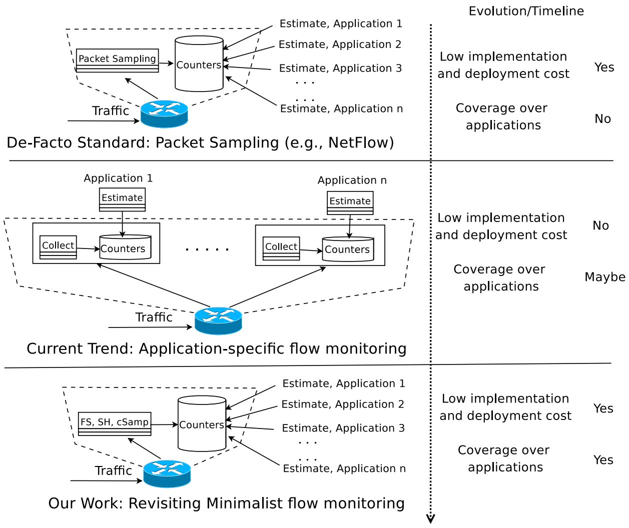
Evolution/Timeline
Packet Sampling
Counters
Estimate, Application 1
Estimate, Application 2
Estimate, Application 3
Estimate, Application n
Traffic
De-Facto Standard: Packet Sampling (e.g., NetFlow)
Low implementation and deployment cost
Yes
Coverage over applications
No
Application 1
Application n
Estimate
Collect
Counters
Current Trend: Application-specific flow monitoring
Low implementation and deployment cost
No
Coverage over applications
Maybe
FS, SH, cSamp
Our Work: Revisiting Minimalist flow monitoring
Low implementation and deployment cost
Yes
Coverage over applications
Yes

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

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

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

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

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

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

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

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

2. BACKGROUND AND RELATED WORK

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

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

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

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

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

3. DESIGN CONSIDERATIONS

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

3.1 Requirements

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

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

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

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

3.2 Design Principles

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

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

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

3.3 Challenges

Given the above considerations, two questions remain:

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

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

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

4. ARCHITECTURE

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

4.1 Router Primitives

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

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

Sample-and-Hold (SH): Sample-and-hold (SH) [17] keeps near-exact counts of "heavy hitters" — flows with high packet counts. SH works as follows. For each packet, the router checks if it is tracking this packet's *flowkey*, defined over one or more fields of the IP 5-tuple. If yes, the router updates that counter. If not, the flowkey for this packet is selected with probability p, and the router keeps an exact count for this selected flowkey subsequently. Since this requires per-packet counter updates, the counters are kept in SRAM [17].

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

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

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

4.2 Resource Management

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

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

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

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

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

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

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

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

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

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

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

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

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

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