ARTICLE IN PRESS

Computer Networks xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Fast Filtered Sampling

Jianning Mai [a,*], Ashwin Sridharan [b], Hui Zang [b], Chen-Nee Chuah [a]

[a] ECE Department, UC Davis, One Shields Avenue, Kemper Hall, Davis, CA 95616, United States
[b] Sprint Nextel, One Adrian Court, Burlingame, CA 94010, United States

ARTICLE INFO

Article history:
Received 28 April 2009
Received in revised form 20 October 2009
Accepted 31 January 2010
Available online xxxx
Responsible Editor: J. Lopez

Keywords:
Sampling
Anomaly detection

ABSTRACT

Traffic sampled from the network backbone using uniform packet sampling is commonly utilized to detect heavy hitters, estimate flow level statistics, as well as identify anomalies like DDoS attacks and worm scans. Previous work has shown however that this technique introduces flow bias and truncation which yields inaccurate flow statistics and "drowns out" information from small flows, leading to large false positives in anomaly detection.

In this paper, we present a new sampling design: Fast Filtered Sampling (FFS), which is comprised of an independent low-complexity filter, concatenated with any sampling scheme at choice. FFS ensures the integrity of small flows for anomaly detection, while still providing acceptable identification of heavy hitters. This is achieved through a filter design which suppresses packets from flows as a function of their size, "boosting" small flows relative to medium and large flows. FFS design requires only *one* update operation per packet, has two simple control parameters and can work in conjunction with existing sampling mechanisms *without* any additional changes. Therefore, it accomplishes a lightweight online implementation of the "flow-size dependent" sampling method. Through extensive evaluation on traffic traces, we show the efficacy of FFS for applications such as portscan detection and traffic estimation.

1. Introduction

Traffic measurement is an essential task performed by large network operators for fault monitoring, traffic engineering, billing, as well as anomaly detection. Given the extremely high-speed of backbone links *e.g.,* 40 Gb/s, packet processing at a line card and the subsequent updating of flow records requires a large amount of resources from CPU and high-speed memory, both of which are at a premium. To ameliorate this situation, *sampling* is commonly performed to reduce the memory and time complexity of packet processing.

Random packet sampling is the *de facto* method deployed in high-speed backbone routers on the Internet for this purpose. The main reason for its prevalence is its simplicity and efficiency. In its simplest form, every packet is sampled with a probability p, *i.e.,* flow record updates are performed only for a fraction p of all the packets. Commercial deployments use a variant called periodic sampling that samples every Nth packet ($N = 1/p$). Even though packet sampling is known to have several drawbacks, which we shall discuss soon, its dominance in current routers clearly illustrates the influence of the complexity constraint on actual deployment.

By the very definition of random packet sampling, the likelihood of sampling a flow is proportional to its size (in packets), also termed *flow bias*. This means that at low sampling rates, the majority of flows in the sampled traffic are more likely to be medium or large flows. Although suitable for estimating "heavy hitters" or aggregating traffic billing that are dominated by large flows, flow bias makes random packet sampling perform very poorly in tracking small flows. Detection and analysis of

* Corresponding author. Tel.: +1 408 9459344.
E-mail addresses: jnmai@ucdavis.edu, chuah@ucdavis.edu, jianning.mai@gmail.com (J. Mai), Ashwin.Sridharan@sprint.com (A. Sridharan), Hui.Zang@sprint.com (H. Zang), chuah@ucdavis.edu (C.-N. Chuah).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.01.015

network anomalies such as distributed denial-of-service (DDoS) attacks and portscans typically involve small flows and are fast becoming important applications of traffic measurements.

Previous studies [1,2] have shown that anomaly detection algorithms, when applied to traffic collected through random packet sampling result in high false positives, rendering the results less meaningful. The main reason for this is the significant flow bias introduced by sampling, which leads to sampled data containing a large number of medium-sized flows compared to small flows. In other words, the relative ratio of medium-sized flows to small sized flows in sampled traffic is severely distorted from its original value. This, coupled with the fact that medium-sized flows in the original traffic also undergo "flow-thinning" due to sampling, making them hard to distinguish from small flows, introduces significant noise in the anomaly detection process, causing false positives. Techniques such as "smart sampling" [3] and "sample-and-hold" [4] that were designed to accurately estimate heavy hitters also performed poorly because of the same reason.

Our goal in this paper is to design a solution to this problem while ensuring minimal complexity or change to the existing traffic collection mechanisms. Intuitively, a simple yet elegant solution to this problem would be to lower the sampling rate of medium-sized and large flows relative to small flows. This reduces the "noise" component in the sampled traffic and can potentially allow an increase in the average sampling probability of other flows of interest, hence further increasing tracking accuracy of those flows. Note, however, that an arbitrary reduction in sampling rate for medium and large flows is not desirable because one would expect any new sampling scheme to still satisfy the traditional applications such as traffic engineering and billing, which require accurate estimation of large flows. Furthermore, implementation of such a "size-dependent sampling" scheme also requires online knowledge of flow size.

We propose Fast Filtered Sampling (FFS), a lightweight implementation for allowing "controllable" flow-size-dependent sampling that meets all of the above constraints. Fast Filtered Sampling is a two-stage mechanism comprising a streaming filter followed by a regular packet sampling module. The filter, which essentially consists of a single array of counters, pre-processes packets from a flow, hashes them to a counter, and then either admits or discards the packet based on the counter value. The filter requires very simple operations per packet: hashing of the flow ID, a counter increment or reset, and decision to drop or pass the packet based on the counter value, thus achieving minimal complexity. Packets from the truncated flow that make it through the filter are sent to the sampling module, which samples the flow further exactly as in current routers, thereby allowing for the possibility of just adding in our filter without changing the existing sampling mechanism.

Our contributions can be summarized as follows.

- First, we propose FFS – a light weight implementation of "size-dependent sampling", a concept first explored by Sketch-Guided Sampling (SGS) [5]. FFS involves installing a counter array in conjunction with existing packet sampling module to approximately estimate flow sizes, and filter packets based on the counter value. It supports both anomaly detection and traditional traffic engineering applications, while maintaining low-complexity, which is critical for implementation and deployment in practice.
- Second, we illustrate how the "sampling curve" yielded by FFS as a function of flow size can be tuned through proper filter parameterization. We also propose a simple design methodology to optimize them, which makes FFS easily configurable for a variety of monitoring applications.
- By extensive evaluations on traffic traces collected from a tier-1 backbone network, we show the effectiveness of FFS as compared to regular random packet sampling (RPS) in terms of reducing false positives for portscans while yielding comparable performance in terms of traditional goals like "heavy hitter" detection and traffic estimation.
- We also provide detailed comparison of FFS against the ideal size-dependent sampling, SGS, with respect to different monitoring applications: heavy hitter identification, and portscan detection. SGS achieves "size-dependent" sampling through online estimation of flow size to lower sampling rates for larger flows. However, the lack of specific implementation details in [5] motivates us to design FFS markedly different to achieve the same objective, but with different trade-offs between complexity and accuracy. we illustrate how FFS can be used in conjunction with adaptive sampling to achieve target measurement cost and accuracy. Note that since the filter actually reduces traffic arriving at the sampling module, the latter can run at a sampling rate $p' > p$, where p is the maximum achievable sampling rate in traditional systems. We leverage this feature and study the increment in p' as well as the resultant benefits.
- Finally, we extend FFS by incorporating two specialized multiple-hashing schemes to improve the accuracy of traffic accounting and the capability to capture unique flows, while further reducing memory consumption. First, a circular counting Bloom Filter (cCBF) is introduced to improve flow size estimation of heavy hitters using counters that are at most one byte each. Second, we propose a header-based hierarchical Bloom Filter (hHBF), which is capable of capturing almost 100% of all the unique flows on a Tier-1 ISP backbone link appearing in the 5-min measurement epoch with only 2 MB fast memory.

The remainder of the paper is structured as follows: Section 2 outlines the literature on sampling and streaming that is relevant to our work. Section 3 presents the Fast Filtered Sampling scheme, the intuition behind the design, and its properties. It also details a comparison against SGS. Empirical evaluation of FFS with respect to a variety of metrics is carried out in Section 4 over real traffic traces and its performance compared with regular packet sampling, SGS, and in some cases, flow sampling. The two extensions cCBF and hHBF are discussed in Section 5. Section 6 concludes the paper.

Please cite this article in press as: J. Mai et al., Fast Filtered Sampling, Comput. Netw. (2010), doi:10.1016/j.comnet.2010.01.015

2. Related work

Traffic measurement is key to various network management tasks such as traffic engineering (TE), accounting and billing, capacity planning, and anomaly detection. Most traffic measurement tools, such as Cisco's NetFlow [6], FlowScan [7], and sFlow [8], typically monitor a link and count the number of packets (or bytes) that satisfies some selection criteria. Flow statistics are then derived from the selected packets. Keeping such per-flow traffic profiles in today's high-speed routers can be challenging to both the processor and the memory. Therefore, deterministic or random packet sampling is commonly employed. The IETF Packet Sampling (PSAMP) working group attempts to define a set of standard capabilities for network elements to sample packets [9–11]. On the other hand, the Flow Information Export (IPFIX) working group is chartered to define the notion of a "standard IP flow" and IP flow information export based upon packet sampling [12–14].

2.1. Inaccuracy due to sampling

While *packet sampling* is simple to implement, extensive research has shown that it leads to inaccurate inference of flow statistics such as the flow size distribution [3,15]. Adaptive schemes that dynamically adjust the packet sampling rate and other operational parameters to reduce the number of flow records while maintaining accuracy have been proposed in [16,17]. On the other hand, Hohn and Veitch [15] discuss the inaccuracy of estimating flow distribution from sampled traffic, when sampling is performed at the packet level, and show that sampling at the flow level leads to more accurate estimations. However, *flow sampling* incurs higher memory and CPU consumption. Duffield et al. propose mechanisms for sampling flow records under hard resource constraints [18], and for consistent sampling across multiple network elements [19]. Other variants such as *sample-and-hold* [4] and *smart sampling* [3] focus on obtaining accurate estimation of heavy hitters at the expense of lower accuracy in tracking small flows.

Recently, sampled data has been used as input for anomaly detection, *e.g.,* detecting denial-of-service (DoS) attacks or worm scans. Anomaly detection often operates on a different region of information than those required for TE and accounting purposes. For example, information such as per source behavior and small flows is important for portscan detection. Recent work [2,1] has shown that random packet sampling and flow sampling with emphasis on heavy hitters are not suitable for detecting portscans.

2.2. Combined streaming and sampling

Data streaming implemented with counter arrays in fast memory has been proposed to obtain accurate flow size information on high-speed links [20]. Unlike sampling, streaming itself does not record the identity of flows. Therefore, streaming combined with sampling has been proposed to obtain traffic features such as detection of super sources or destinations [21]. Kumar et al. [22] combine packet sampling and streaming to statistically estimate flow size distribution for any arbitrary subpopulation including anomalous flows. Venkataraman et al. [23] design new streaming algorithms for fast detection of superspreaders such as worm scanners, which connect to a large number of distinct destinations. In this algorithm, a traditional hash-based flow sampling scheme followed by another hash table is used to count the fan-out values for each sampled source. Another family of approach is to use sketches as high-speed counting devices, and select packets with dominant counts in such sketches for sampling or further processing. Examples include an automated worm fingerprinting mechanism that classifies content segments that are seen coming from multiple sources and going to multiple destinations as possible worm signatures [24].

Our FFS scheme is related to size-dependent flow sampling, of which both *offline* and *online* variants have been proposed. Offline approaches assume flow records have been collected in advance and the exact flow size information is known. The proposed techniques then select certain flow records according to pre-defined constraints [3,18] to keep or to export. The offline approaches are difficult to implement because of the memory/CPU requirement to obtain a *full* flow table in advance. The online approaches decide the sampling rate of a flow on-the-fly and examples are sample-and-hold [4] and sketch guided sampling (SGS) [5]. As mentioned, sample-and-hold focuses on flows of large sizes only. SGS, on the other hand, allows a designer to tailor the sampling rate as a function of flow size, while providing flexibility in the allocation of resources.

SGS is the closest related sampling scheme compared to FFS. Its main idea is to make the probability with which an incoming packet is sampled a decreasing sampling function of the size of the flow the packet belongs to. This way SGS is able to significantly increase the packet sampling rate of the small and medium flows at the expense of the large flows to improve the accuracy of flow size estimation especially for small and medium flows. Both SGS and FFS utilize sketches or counter arrays to keep the flow size information and computes sampling probabilities $f(i)$ for each packet as a function of its flow size i. It is shown in SGS that in order to achieve linear growth of the estimation error ϵ as flow size i^{α} increases, where $1/2 \leqslant \alpha \leqslant 1$ is a tunable constant, the sampling function $f(i)$ needs to be $1/(1+\epsilon^2 i^{(2\alpha-1)})$, with i being the actual flow size. When $\alpha = 1/2$, the sampling probability $f(i) = 1/(1+\epsilon^2)$ is a constant which corresponds to uniform sampling. If $\alpha = 1$, the sampling function becomes $f(i) = 1/(1+\epsilon^2 i)$, which proves to have a constant relative error in flow size estimation. SGS adopts this sampling function to ensure that later packets from large flows get sampled less and less to reduce resources. Another parameter β can be introduced to further lower the sampling probability to match other sampling methods. In our evaluation, we choose $f(i) = \beta/(1+\epsilon^2 i^{(2\alpha-1)})$, where $0 < \beta \leqslant 1$.

Similar to SGS, the goal of FFS is to increase the effective sampling rate of small flows, which is essential for detecting anomalies such as portscans. We however differ from SGS in that our design is much simpler and decoupled from the sampling mechanism, allowing for incremental upgrades without affecting existing sampling modules in

Please cite this article in press as: J. Mai et al., Fast Filtered Sampling, Comput. Netw. (2010), doi:10.1016/j.comnet.2010.01.015

a router. Section 3.3 provides a more in-depth comparison of FFS and SGS. Quantitative performance comparison between FFS and SGS in the context of supporting portscan detection and other applications is presented in Section 4.3.

2.3. Sampling with Bloom Filters

Estan and Varghese [25] applied Bloom Filters to network measurement problem of detecting heavy hitters in traffic. Each packet entering a router is hashed k times into a counting Bloom Filter. The counters are incremented by the number of bytes in the packet. In order to reduce the false positives significantly, a conservative update is performed. More recently, Kumar et al. [26] propose a novel technique called space-code Bloom Filter, for approximate measurement of all flows instead of just heavy hitters. Lu et al. [27] designed a novel data structure "counter braids" which is inspired by Bloom Filter and sparse random graph codes. It is shown that counter braids measures all flow sizes error-free with only 5 bits per flow. Another recent work FlexSample [28] dynamically extracts traffic from subpopulations that operators define using conditions on packet header fields. FlexSample uses a fast, flexible counting Bloom Filter to provide rough estimates of packets' membership in respective subpopulations.

The latest work on traffic measurement [29,30] provides more flexible sampling of traffic programmable set of flows tailored to application requirements and/or traffic conditions. The programmable flow set partition language [29] and the generic language for application-specific flow sampling [30] both assume a priori knowledge of flow set composition such as elephant flows, small flows, or certain subpopulations. FFS and its extensions, however, is designed as generic sampling methods that single out small, medium, and large flows, which servers as a first step towards zooming-in on specific group of flows or subpopulations.

3. Fast Filtered Sampling

The broad motivation behind FFS is to design a low-complexity mechanism that can provide some degree of control over the sampling rate perceived by a flow as a function of its size. More specifically, as alluded to previously, we wish to lower the sampling rate as the flow size increases. While similar in objective to SGS [5], our motivation and design is different. From the perspective of anomaly detection, it is beneficial to reduce the likelihood of sampling mid-sized flows (relative to small flows) since they have been shown to act as a source of noise as far as anomalies are concerned [1]. At the same time, we would like to detect large flows since they are useful for billing and traffic engineering purposes, and do not unduly affect anomaly detection and IP source addresses with large fan-out/fan-in values that are usually candidates for potential anomalies.

3.1. The design

We now discuss the motivation for our design principles and how they are in line with our dual objectives. The Fast Filtered Sampling (FFS) architecture shown in Fig. 1 consists of two modules. The *sampling* module performs random or uniform packet sampling with some probability p as is common in current routers. The intelligence of the scheme lies in the first phase, the *filter* module which comprises of an array of N counters of m bits each. The counting array filters out packets based on two *a priori* specified parameters (s, l) in the following fashion. Every incoming packet is hashed based upon its flow id to one of the N counters and the value of that counter incremented *modulo* l. If the value of the updated counter is less than s, the packet is passed onto the sampling module where it may be selected with probability p or discarded otherwise. Note that by virtue of the modulo l operation, if the counter value exceeds l, it is reset to zero.

To see how this scheme influences the actual packet sampling rate, consider a flow that initially hashes to an "empty" counter. If the flow size $i \leqslant s$, it passes through the filter undisturbed (barring hash collisions from other flows) and each packet will be sampled with probability p by the sampling module. If $s < i \leqslant l$, the flow will get truncated to s packets by the filter, which reduces their chance of getting sampled comparing to the case where no filter is presented. Finally, flows of size $i > l$ will get "thinned" by a constant factor $\lceil \frac{l}{s} \rceil$ before arriving at the sampling module, since the counter resets to zero when more than l packets hash to the counter due to the modulo l operation.

Therefore, the "sampling" curve of the filter sampling mechanism as a function of the flow size can be succinctly captured with following equations:

$$\Pr\left\{\begin{matrix}\text{A packet sampled}\\ \text{from flow of size } i\end{matrix}\right\} = \begin{cases} p & \text{if } 1 \leqslant i < s, \\ ps/i & \text{if } s \leqslant i < l, \\ ps/l & \text{otherwise.} \end{cases} \quad (1)$$

The intuition behind our filter design is based on the Zipfian nature of network traffic, which has been validated by previous works through extensive measurements. Specifically, small flows are numerous, but contribute a small percentage to the total traffic; while large flows, though few, account for a major fraction of the actual traffic. Anomalies usually manifest as small flows with large fan-outs, *e.g.,* portscans, or have large fan-ins *e.g.,* DDoS attacks. By setting s to an appropriately small value, this type of traffic should pass through the filter undisturbed since they would very likely hash to different locations in the counter array and have size smaller than s. They still get

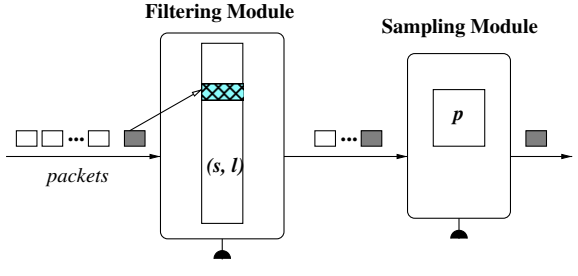


Fig. 1. The two-stage architecture using a high-speed streaming filter ahead of sampling module.

sampled with probability p by the sampling module though. Moderately sized flows that are part of regular behavior would typically get truncated to s packets, reducing their sampling likelihood. Since they are fewer in number, the presence of the filter has the potential to dramatically reduce their presence after sampling. Finally, even though the large flows experience the lowest sampling rate of $p\lceil\frac{s}{l}\rceil$, they are still measurable for traffic engineering purposes due to their proportionately large size.

3.2. Setting filter parameters

Generally, the sampling probability p is decided by a router's CPU, memory, and peak traffic load. However, in practice, determining the optimal sampling rate is difficult because it would require *a priori* information about the traffic mix and dynamics. Consequently, we assume that p is a *fixed* input to FFS. Note that our filter actually reduces the traffic offered to the sampling module. Hence, one could potentially increase the sampling rate to boost performance. We also envision an adaptive sampling scheme [16] that varies the sampling rate as a function of memory usage, which undoubtedly allows an increase in sampling rate. We do indeed identify the associated benefits through empirical evaluation in Section 4.

Assuming the filter has a sufficiently large number of counters available at its disposal [31] so as to minimize hash collisions, the performance of the filter is controlled by the filter module parameters (s, l) and the sampling probability p. We present here a simple methodology that guides the selection of the filter parameters. Let flows with size in the range $[\mathcal{S}, \mathcal{L}]$ be defined as medium-sized flows, where $\mathcal{S}$ and $\mathcal{L}$ are inputs to the design process provided by the network operator/designer. Our goals of the filter design are as follows:

1) Minimize the likelihood that we sample a flow of size $i \in [\mathcal{S}, \mathcal{L}]$ to improve anomaly detection,
2) Ensure that we sample large flows (*i.e.*, $i > \mathcal{L}$) for useful traffic accounting with confidence $1-\epsilon$, where ϵ is a small value satisfying $0 \leqslant \epsilon < 1$.

The probability of detecting a flow of size $i \geqslant s$ is given by

$$D_i = 1 - (1-p)^{s\lceil\frac{i}{l}\rceil}. \tag{2}$$

We treat detection of a medium-sized flow as *distortion*, therefore we wish to minimize D_i for $i \in [\mathcal{S}, \mathcal{L}]$. Equivalently, we can maximize

$$\ln(1-D_i) = s\left\lceil\frac{i}{l}\right\rceil \ln(1-p) = s\left\lceil\frac{i}{l}\right\rceil q, \tag{3}$$

where $q = \ln(1-p)$. If $f(i)$ is the likelihood that a flow is of size i, a suitable cost function to maximize is: $\sum_{i=\mathcal{S}}^{L} f(i) s\lceil\frac{i}{l}\rceil q$. In practice, however, $f(i)$ cannot be predetermined. There are a few choices, for example, we could minimize the cumulative relative distortion $\sum_{i=\mathcal{S}}^{L} s\lceil\frac{i}{l}\rceil q$ instead, which essentially simulates an uniform distribution. An alternative is to assume a Zipfian distribution to optimize the parameters.

Our second objective states that we wish to ensure a large flow sampled with certain confidence:

$$D_{\mathcal{L}+1} = 1 - (1-p)^{s\lceil\frac{\mathcal{L}+1}{l}\rceil} \geqslant 1-\epsilon. \tag{4}$$

Similarly we could simplify it as:

$$s\left\lceil\frac{\mathcal{L}+1}{l}\right\rceil \geqslant \frac{\ln\epsilon}{q}. \tag{5}$$

Putting it all together, we wish to solve the following optimization problem for the uniform distribution:

$$\mathbf{A_1}: \max_{s,l\in\{\mathcal{Z}^+\}} \sum_{i=\mathcal{S}}^{\mathcal{L}} s\left\lceil\frac{i}{l}\right\rceil q \tag{6}$$

$$\text{subject to} \quad s\left\lceil\frac{\mathcal{L}+1}{l}\right\rceil \geqslant \frac{\ln\epsilon}{q}, \tag{7}$$

$$1 \leqslant s \leqslant \mathcal{S} \leqslant l. \tag{8}$$

The constraint $s \leqslant \mathcal{S} \leqslant l$ ensures that flows in the range $[\mathcal{S}, \mathcal{L}]$ experience sampling governed by filter constraints. Observe in the above cost function Eq. (6), that for a *feasible* value of l, we should choose as small an s as possible in order to maximize the cost since q is negative. This allows us to efficiently search the solution space. Specifically we vary l from $\mathcal{S}$ to $\mathcal{L}$, and for each l we set $s = \left\lceil\frac{\ln\epsilon}{q} / \lceil\frac{\mathcal{L}+1}{l}\rceil\right\rceil$ and compute the cost function. The (s, l) combination that yields the largest cost is then chosen. Table 1 shows optimal values of (s^*, l^*) for some example values of $\mathcal{S}$ and $\mathcal{L}$. Note we choose l to be power of 2 to align with counter boundaries.

3.3. Effect of filtering

To demonstrate how FFS filter with parameters (s, l) affects the traffic sampled, we list packet and flow counts from a real traffic trace by FFS compared against RPS and SGS in Table 2. To ensure fair comparison, we fix the packet sampling rate to 1/100 for both RPS and the second stage of FFS. The parameters for SGS used in the evaluation were $\alpha = 1$, $\epsilon = 0.5$, thus the sampling function is set to be $1/(1+0.5^2 i)$, where i is the flow size. We then re-sample the output trace from SGS with a rate of 1/80, so that effectively the first packet in each flow gets picked at a probability of 1/100, which is exactly the same for both FFS and RPS. Note that the *effective* packet sampling rate of FFS is hence much lower than 1/100 as a result of the first-stage filter, leading to much lower resource consumption. For example, FFS with parameters $(s, l) = (4, 8)$ samples only 0.54% packets, an equivalent to 1/200 RPS. When $(s, l) = (3, 64)$, FFS captures 0.21% of the total pack-

Table 1
Optimal filter parameters $(s^*, l^*)_u$ for uniform flow distribution, and $(s^*, l^*)_z$ for Zipfian flow distribution ($\epsilon = 0.01$).

$\mathcal{S}$	$\mathcal{L}$	p	$(s^*, l^*)_u$	$(s^*, l^*)_z$
1	1000	0.01	(1,2)	(1,2)
5	1000	0.01	(4,8)	(4,8)
10	1000	0.01	(8,16)	(8,16)
1	10,000	0.01	(1,16)	(1,16)
5	10,000	0.01	(3,64)	(1,16)
10	10,000	0.01	(3,64)	(1,16)

Please cite this article in press as: J. Mai et al., Fast Filtered Sampling, Comput. Netw. (2010), doi:10.1016/j.comnet.2010.01.015

Table 2
Number (percentage) of packets and flows captured by RPS, FFS, and SGS from Trace *BB-West*.

Sampling	No. of packets (%)	No. of flows (%)
Original	61,998,113	8,083,202
RPS	619,983 (1.00%)	351,928 (4.35%)
FFS (4,8)	332,633 (0.54%)	239,250 (2.95%)
SGS	174,606 (0.28%)	171,295 (2.12%)
FFS (3,64)	132,624 (0.21%)	129,084 (1.60%)

ets, which is approximately 1/400 in RPS. The design of the FFS indeed guarantees that the flow table in routers do not see more traffic than with existing sampling techniques regardless of the values of (s, l).

We found that the flow counts for different flow sizes observed by FFS is different from RPS. Since our filter is designed to suppress the relative sampling rate of medium-sized flows while maintaining reasonable estimates of large-sized flows, it is not surprising that flow sampling ratio for mid-sized flows is much lower for FFS compared to RPS. We shall see the benefits of this suppression in Section 4.3 when we use the sampled traffic to detect portscans.

4. Performance evaluation

Our evaluation of FFS was carried out on packet traces collected on two OC-48 links in a Tier-1 ISP's backbone network. These traces were collected by a passive monitoring system that captures the first 48 bytes of the IP header of every IP packet traversing the monitored link. Details of the traces are presented in Table 3. Our evaluation was carried out by applying different sampling techniques to the traces and then analyzing metrics related to different applications/traffic features on these sampled traces.

The performance of FFS on capturing traffic features as well as detecting scanners are compared against two other sampling schemes: (a) random packet sampling (RPS), which emulates the behavior of the Cisco NetFlow sampling process [6], and (b) sketch guided sampling (SGS), the closest related work to FFS. For portscan detection, we also consider random flow sampling (RFS) as another benchmark for comparison, since RFS has been shown to provide more accurate estimation of flow distribution [15] although it is prohibitively expensive to implement.

Our study focuses on quantifying how accurate FFS is in preserving traffic features despite the much lower overall sampling rate. Design features compared with SGS are studied in Section 4.1. Section 4.2 investigates the application of FFS to identifying large flows. Section 4.3 compares portscan detection performance among all sampling schemes. In Section 4.4, we explore how the accuracy of FFS can be improved by adapting the second-stage packet sampling rate to meet the same resource budgets (*e.g.*, number of packets processed) as RPS.

Table 3
Trace data statistics.

Trace	Date	Average rate	Duration
BB-West	03-08-2003	55 Mbps	1 h
BB-East1	04-07-2003	208 Mbps	18 h
BB-East2	04-07-2003	269 Mbps	18 h

4.1. FFS design features

The core guiding principle in the FFS design is to ensure low *per packet* processing complexity, which is critical on high-speed links. For example, on an OC-762 (40 Gbps) link, forwarding and monitoring functions for each packet must be completed in roughly 25 ns assuming 1000 bit packets. This requires that the operations per packet are both simple and low in number. Indeed, such a severely resource constrained environment is the primary reason why present day routers deploy periodic sampling rather than *random* sampling since the latter involves expensive computation. FFS adheres to such requirements by requiring only three additional updates to packet processing in order to achieve flow-size-dependent sampling compared to random packet sampling: computation of the hash function, incrementing the counter array, and decision to drop or pass the packet. All three operations can be implemented in binary arithmetic which significantly speeds up processing. FFS also has low memory requirements since the *width* of each counter need only be $\log_2 l$. Since typically $l \leqslant 128$, a one byte counter is sufficient.

It is worthwhile to compare the FFS design against its closest counterpart SGS [5] at this stage. SGS aims to reduce the sampling rate with increasing flow size. It utilizes a "smooth curve" function of the *estimated* flow size i, to determine the packet sampling probability p_i for each packet belonging to the flow, *e.g.*, $p_i = \frac{\beta}{1+\epsilon^2 f_i^{2\alpha-1}}$, where α, β, and ϵ are constants. In fact, one could view the sampling rate curve generated by FFS in Eq. (1) as a piecewise approximation of the SGS curve. To illustrate this, we have plotted both sampling curves in Fig. 2. Note that we used a value of $\beta = 0.0125$ so that the sampling rates for the first packet are equal from both SGS and FFS. $\alpha = 1$ and $\epsilon = 0.5$ were chosen based on parameters used in [5].

The filter module used in our experiment for FFS and SGS has an array of $N = 8M$ counters, which handles the traces with minimum collisions. In our evaluations the maximum value of $l = 128$ and hence FFS requires only a

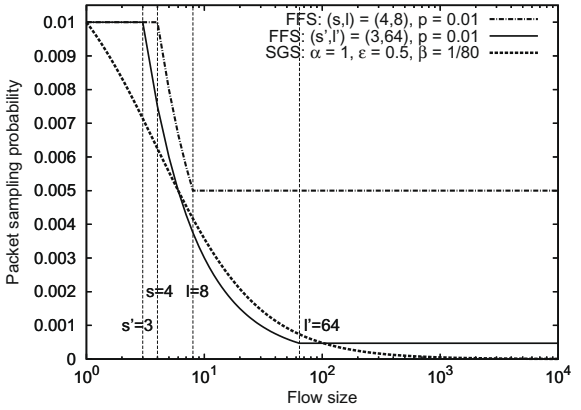


Fig. 2. The packet sampling probability vs. flow size for FFS and SGS.

7-bit counter. The design principle of SGS presented in [5] naturally leads us to pick a 32-bit wide counter to accommodate large flow sizes and prevent counter overflow. While one could potentially envision small counter sizes for SGS, in which case, the sampling probability would be 'frozen' when the counter fills-up, these aspects were not discussed in [5] and hence we remain true to the original design proposal.

The benefit of sampling with a flexible "smooth curve" in SGS comes at the price of increased complexity. Not only does it demand an overall change in the existing sampling mechanism, SGS also requires more processing capability. For each packet to be sampled, SGS requires either on-the-fly arithmetic computation of the sampling probability or a look-up table that *a priori* stores all sampling probabilities. The former incurs processing penalty and the latter memory resources. Moreover, changing the sampling probability function when reconfiguring the router will introduce heavy overhead on packet processing. FFS, on the other hand, embodies the *decoupling* of filtering and sampling. It brings an important benefit by allowing insertion of the filter without disturbing the existing sampling mechanism (be it random or systematic packet sampling), or requiring any additional operations from the existing sampling module.

Next we will compare the actual performance between FFS and other sampling schemes including SGS, and shows that in spite of its simplicity FFS yields similar or better results.

4.2. Catching heavy hitters

In this section, we investigate if a simple scheme like FFS, which focuses on tracking small flows, can provide reasonably good estimates of heavy hitters. We design the streaming filter to detect flows of size larger than $\mathcal{L}$ with probability close to 1. This guarantees at least one packet will be sampled from those flows. To infer the original flow size from the sampled size with certain accuracy, however, requires more than just one sampled packet. We know that for a large flow of size n, the average size after uniform sampling is $k = pn$, which follows a binomial distribution. Hence the estimator $\hat{n} = k/p$ is an unbiased estimator of n with variance [32]: $\mathrm{Var}\left(\frac{k}{p}\right) = \frac{1}{p^2} \cdot np(1-p) = \frac{n(1-p)}{p}$. Note that $\hat{n} = k/p$ is the estimator for RPS. In case of FFS, the filter scaling needs to be considered, *i.e.*, $\hat{n} = \lceil kl/(ps) \rceil$.

From the original packet trace, we can easily identify a list of heavy hitters (*i.e.*, flows with more than $\mathcal{L}$ packets) as the ground truth. We then compute the estimated flow sizes using the sampled data from RPS, FFS, and SGS and classify flows with estimated flow size greater than $\mathcal{L}$ as heavy hitters. Table 4 shows the number of successfully detected heavy hitters (N_s), false positives (N_{f+}), and false negatives (N_{f-}) for RPS, FFS, and SGS as compared to the ground-truth. In this test, we set $\mathcal{L} = 1000$ packets and estimate the number of the large flows with size greater than $\mathcal{L}$.

With a sampling rate of $p = 1/100$, RPS detects over 75% of the heavy hitters in Trace BB-East 1 and 60% of the heavy hitters in Trace BB-East 2. Note that FFS manages to catch half of the heavy hitters detected by RPS because it samples much less packets as indicated in Table 2. SGS, on the other hand, detects slightly more heavy hitters observed by FFS, but with almost tripled false positives. This is because re-sampling must be done to ensure that packet rates of all sampling schemes are comparable and this causes performance deterioration for SGS. However, it is important that one should not hold this against SGS since reducing measurement cost is not the primary design goal of SGS. It is our purpose to illustrate that it is not practical to deploy SGS in conjunction with a second-stage random packet sampling with the hope of reducing measurement resources, because it may introduces huge inaccuracy. Without re-sampling, SGS provides much better accuracy in flow size estimation. However, in order to achieve that, SGS would end up sampling about 25% of the packets, a rate impractical due to heavy processing overload.

Figs. 3 and 4 compare the estimated flow size distribution against the original flow sizes for the *BB-East1* and *BB-East2* Traces, respectively. Compared to RPS, the inverted flow size by FFS and rSGS is not as accurate as RPS. This explains why FFS $(3, 64)$ with p = 1/100 fails to detect some of the heavy hitters observed by RPS. Note that with this parameter setting, the *effective* packet sampling rate of FFS is 1/169, which is approximately half of RPS. FFS with parameter (3,64) is also comparable with SGS in terms of packets/flows captured as indicated from both Table 2 and the sampling curves in Fig. 2. Given that FFS can leverage the much lower resource consumption to raise the second-stage sampling probability (as shown in Section 4.4), flow size estimation can be expected to be more accurate than RPS with equivalent sampling rate.

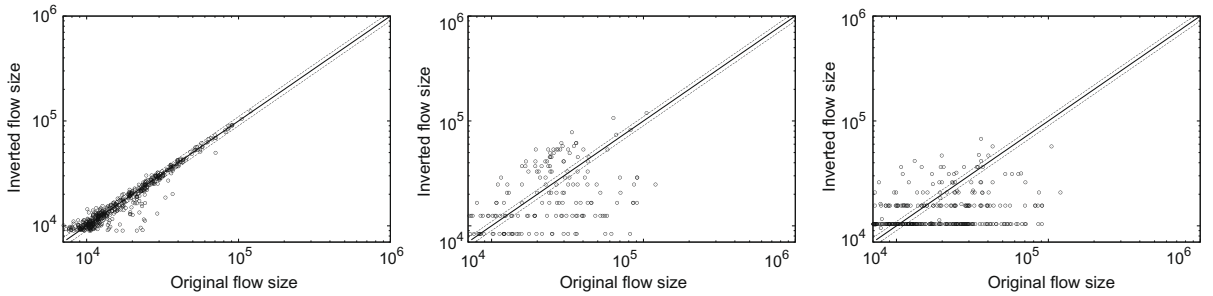
4.3. Portscan detection

One of the common classes of anomalies that are visible to ISPs is portscan, which is usually associated with worm or virus propagation. TAPS [33] was previously proposed as an effective portscan detection mechanism for backbone

Table 4
Heavy hitters detection results with RPS $p = 1/100$, FFS $(s, l) = (3, 64)$, $p = 1/100$ and re-sampled SGS (rSGS) $\alpha = 1$, $\epsilon = 0.5$, $p = 1/80$ from Trace *BB-East1* (total: 606), and Trace *BB-East2* (total: 1717) for 1 h.

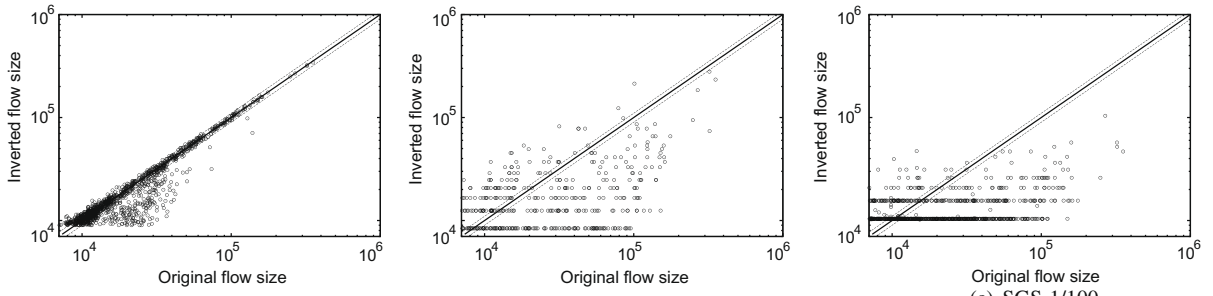
Sampling	*BB-East1*			*BB-East2*		
	RPS 1/100	FFS (3, 64)	rSGS	RPS 1/100	FFS (3, 64)	rSGS
Success (N_s)	464 (76.6%)	202 (33.3%)	253 (41.7%)	1093 (63.7%)	517 (30.1%)	654 (38.1%)
False positive (N_{f+})	65 (10.7%)	125 (20.6%)	383 (83.8%)	119 (6.93%)	358 (20.9%)	837 (48.7%)
False negative (N_{f-})	142 (23.4%)	404 (66.7%)	353 (58.3%)	624 (36.3%)	1200 (69.9%)	1063 (61.9%)

Please cite this article in press as: J. Mai et al., Fast Filtered Sampling, Comput. Netw. (2010), doi:10.1016/j.comnet.2010.01.015



(a) RPS 1/100 (b) FFS (3,64) 1/100 (c) SGS1/100

Fig. 3. Inverted flow size vs. the original flow size using RPS, FFS with $(s,l) = (3,64)$ and SGS at 1/100 from the *BB-East1* Trace.



(a) RPS 1/100 (b) FFS (3,64) 1/100 (c) SGS 1/100

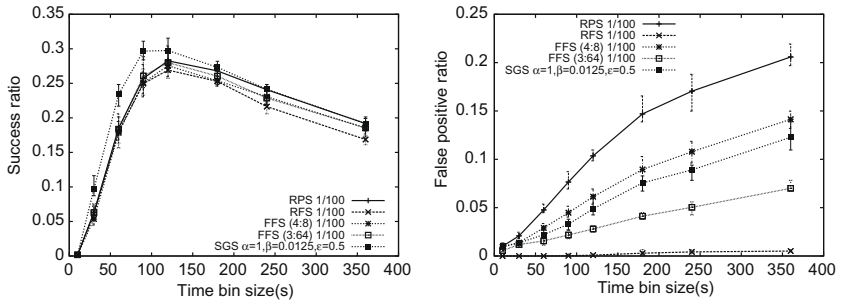
Fig. 4. Inverted flow size vs. the original flow size using RPS, FFS with $(s,l) = (3,64)$ and SGS at 1/100 from *BB-East2* Trace.

environment. TAPS utilizes the access pattern to separate scanners from benign hosts. This is based on the observation that a scanner often initiates connections to a larger spread of destination IP addresses, or port numbers. TAPS combines a rate limiting scheme for event generation with the sequential hypotheses test to achieve fast detection as well as low false positive rate.

TAPS forms two hypothesis, H_0 that a source is a "benign" host and H_1 that a source is a "scanner", characterized by the likelihood of the success or failure of a connection. The rationale for this definition is based on the observation that a scanner often initiates connections to a larger spread of destination IP addresses (horizontal scan), or port numbers (vertical scan). In other words, the ratio γ between distinct destination IP addresses and port numbers (or its reciprocal, whichever is bigger) for a scanner is far larger than a non-scanner. The algorithm works as follows. In each time bin (say i seconds), for each source, the ratio γ is computed and compared to a pre-defined threshold k. The event variable in the hypothesis testing associated with that time bin is then set to 0 or 1 depending on whether γ exceeds or lies below the threshold. The likelihood ratio of the hypothesis testing is then updated based on the value of the set event variable. Decision regarding which hypothesis applies to the source is made based on whether the likelihood ratio crosses a scanner threshold or a benign host threshold. It is clear from the above discussion that the threshold k affects the accuracy of detection, while time bin size controls the promptness of decision making.

In this section, we evaluate the performance of TAPS under FFS compared to RPS and SGS. Similar to previous study in the literature [1], we consider the following metrics: the success ratio R_s that indicates the effectiveness of the detection algorithm, and the false positive ratio R_{f+} that measures the relative error of mistakenly tagging a benign flow as port scanner. The metrics were normalized by the ground-truth so as to be comparable across different sampling schemes. Note when comparing performance of different detectors, ROC curves are often cited. However, our focus in the paper is to compare the relative performance of TAPS using data collected by different sampling methods. Hence we set $k = 3$ and vary the time bin value to plot the above defined success ratio and false positive ratio instead.

It turns out that comparing to RPS, FFS maintains a similar success ratio, while producing a lower false positive ratio. Fig. 5 shows the success and false positive ratios for FFS with $(s,l) = (4,8)$ and $(s,l) = (3,64)$ versus random packet sampling (RPS) at rate 1/100 and random flow sampling (RFS) with probability 0.01, and SGS using the same parameters as in Fig. 2. All four approaches have similar performance in terms of success ratio with SGS being slightly better than the rest. Both SGS and FFS perform better than RPS in terms of false positives. Particularly, FFS with parameter $(s,l) = (3,64)$ introduces much lower false positive ratio compared to rSGS. Previous work [2,1] has shown that false positives are mainly caused by *flow-thinning*, wherein a multi-packet flow is shortened to a single packet flow by packet sampling. FFS suppresses the frac-



(a) BB-West Success (b) BB-West False Positive

Fig. 5. Portscan detection results under FFS compared to RPS, RFS, and SGS.

tion of medium flows being sampled by design, hence resulting in lower false positive ratios.

4.4. Adaptive sampling rate

Often in practice the sampling probability p is already known based on memory, CPU, and peak load requirements. Consequently, we assume that p is a *fixed* input to our process. Note that the filter actually reduces the traffic offered to the sampling module. Hence, one could potentially *increase* the sampling rate compared to a system without the filter, and in the more general case also treat p as a variable that needs to be optimized. While this would definitely boost performance, it is problematic to *a priori* estimate p since it would require information about the traffic mix and dynamics to quantify the reduction by the filter, which is hard to obtain.

Alternatively, one could envision an adaptive sampling scheme (*e.g.,* [16]) that varies the sampling rate as a function of memory usage. This would also undoubtedly allow an increase in sampling rate because of lower traffic arriving to the sampling module and we do indeed identify the associated benefits through empirical evaluation in Sections 4.2 and 4.3. However, even in this case, there is a notion of *average* sampling probability that must be honored.

Obviously from Eq. (1), the effective packet sampling rate for FFS is at most p, under the condition that almost all flows are with size $i \leqslant s$. Typical scenarios where this happens include flooding attacks and worm breakouts. In these extreme cases, the filter in FFS becomes an all-pass filter, with no filtering effects. We have to lower the sampling probability p in the second stage of FFS to alleviate the load on router's CPU and memory and network bandwidth. In most cases, filtering often reduces the number of packets passed to the sampling stage. Therefore, under normal traffic load conditions, we could increase the sampling rate to whatever resource constraints allow us.

Adaptive sampling has previously been proposed in the literature. Choi et al. [17] suggested that sampling rate should be adapted based on packet size distribution and total packet count within given time blocks to achieve target load measurement accuracy. Estan et al. [16] recommended adaptive sampling to overcome major shortcomings of NetFlow, which include router memory and network bandwidth overflow during flooding attacks, and static-rate sampling is not suitable for all applications. We propose to adapt the sampling rate in the second stage of FFS so that the composite filtering and sampling fraction meets the rate of RPS. As we show next, the advantages of the rate increase include improvement in flow size estimation of heavy hitters and better detection of portscans.

Table 5 lists the number and percentage of packets and flows captured through FFS if we were to adapt the sampling rate to match the effective packet sampling rate of RPS. For instance, FFS with $(s, l) = (1, 16)$ allows us to increase the random sampling rate to $p = 1/10$ so that the equivalent packet sampling rate is around $1/60$. The benefit of raising p is shown in Fig. 6, where the success detection ratio using TAPS is improved for FFS, while the false positive ratio remains at the same level as RPS with similar sampling rate of 1/50.

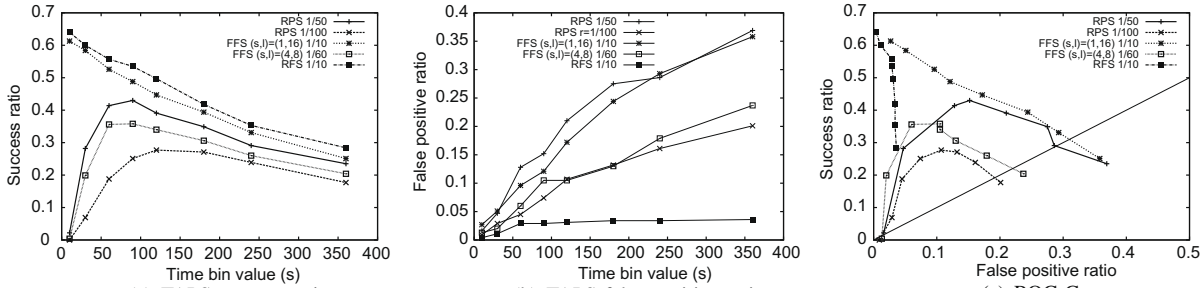
5. Filtered sampling with memory constraints

Fast Filter Sampling (FFS) comprises two stages – an independent counting array to estimate flow size and filter packets based on the estimated size, and a sampler at choice to further sample packets. FFS amends the integrity of small flows for anomaly detection, while still providing acceptable identification of heavy hitters. Yet there are

Table 5
Number and percentage of packets and flows captured by FFS and RPS with adaptive sampling rate for the *BB-West* Trace (total 61,998,113 packets and 8,083,202 flows).

Sampling schemes	No. of packets	% of packets	No. of flows	% of flows
RPS (No filter), p = 1/100	619,983	1.0%	351,928	4.35%
FFS $s = 1,\ l = 16,\ p = 1/10$	1,025,582	1.65%	872,489	10.8%
FFS $s = 4,\ l = 8,\ p = 1/60$	608,146	0.98%	392,059	4.85%
FFS $s = 4,\ l = 128,\ p = 1/30$	561,464	0.91%	530,024	6.56%

Please cite this article in press as: J. Mai et al., Fast Filtered Sampling, Comput. Netw. (2010), doi:10.1016/j.comnet.2010.01.015



(a) TAPS success ratio (b) TAPS false positive ratio (c) ROC Curve

Fig. 6. Performance of TAPS under FFS with increased packet sampling rate compared to RPS and RFS.

quite a few avenues to improve in FFS design and performance. While we try to ameliorate hash collisions in updating counters in the filter by using sophisticated hashing functions such as MD5, the collision ratio drops only insignificantly, but per packet processing time increases well above the budget of online processing time. In addition, the amount of fast SRAM memory needed for the filter and flow table is much higher than practically possible. Strict memory constraints are required to design the data structure and sampling schemes.

In this section, we present two enhancements that can improve the performance of FFS while minimizing memory consumptions. First, we propose a circular counting Bloom Filter (cCBF) for monitoring the flow sizes of heavy hitters to improve the accuracy of traffic accounting. Second, we present an IP packet header-based hierarchical Bloom Filter (hHBF) design that is capable of capturing all or most of the unique flows in a measurement epoch to improve the performance of per-flow measurement. Both cCBF and hHBF exploit a multi-hashing scheme called Bloom Filter.

5.1. Circular counting Bloom Filter

The design principles behind circular counting Bloom Filter (cCBF) is to improve heavy hitter estimations with minimum memory consumption. Before we present the details of cCBF, we first provide a brief overview of Bloom Filter.

A Bloom Filter [34] represents a set $S = \{x_1, x_2, \ldots, x_n\}$ of n elements with an array of m bits, initially all set to 0. A group of k independent hash functions $h_1, \ldots, h_k$ map each item in the universe to a random number uniformly over the range $\{1, \ldots, m\}$. To add an element $x_i \in S$, the bits $h_j(x_i)$ are set to 1 for $1 \leqslant j \leqslant k$. To query if an item x is in S, we check whether all $h_j(x)$ are set to 1. If not, x is clearly not a member of S. Otherwise, either x is in S, or it is a false positive (with a small probability).

Unfortunately, Bloom Filter cannot perform a deletion by reversing the process. To solve the problem, Fan et al. [35] introduced the idea of a counting Bloom Filter (CBF). CBF extends each entry in the classic Bloom Filter from a single bit to a small counter. Now insertion increments the corresponding counters and the deletion obviously involves decrementing the corresponding counters.

It is shown that the false positive rate of Bloom Filters can be estimated by [36]:

$$f_+ = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx (1 - e^{-kn/m})^k. \quad (9)$$

Bloom Filter risks false positives due to hash collisions. Table 6 lists the false positive ratios for a few common combinations of m/n and k in Eq. (9). Obviously, the probability of false positives increases as n, the size of the set S increases, and decreases as m, the number of counters increases. For given m and n, the optimal number of hash functions $k = (m/n) \cdot \ln 2$, which gives the minimum false positive rate at $0.5^k \approx 0.6185^{m/n}$.

The cCBF architecture is shown in Fig. 7. The counting Bloom Filter hashes flow IDs of an incoming packets to update the corresponding counters, and packets get sampled depending on current counter values and filtering rules. For example in Fig. 7, the flow ID f of the packet is hashed to three buckets $h_1(f)$, $h_2(f)$, and $h_3(f)$ in the counter array. The sampling rule compares the minimum counter value c against the pre-defined threshold S: if $c \leqslant S$, the packet passes; otherwise, it is dropped.

To avoid counter overflow, we usually need sufficiently large counters. However, since the counters in cCBF increments in a *modulo L* fashion similarly to the updating process of FFS, we can reduce the counter width significantly. Generally, cCBF samples packets based on two *a priori*

Table 6
False positive rate of bloom filters with different parameters ($k \leqslant m/n$).

m/n	4	6	8	10	12
$k = 6$	–	6.38%	2.16%	0.844%	0.371%
$k = 3$	14.7%	6.09%	3.06%	1.74%	1.08%
$k = 1$	22.1%	15.4%	11.8%	9.52%	8.00%

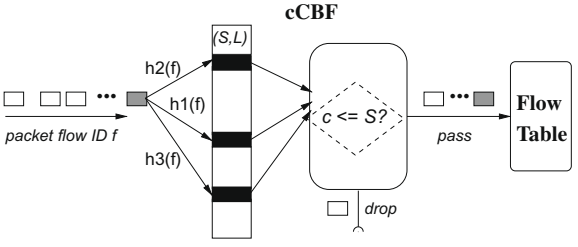


Fig. 7. The architecture of cCBF using a lightweight circular Bloom Filter for sampling.

specified parameters (S, L) in the following fashion. Every incoming packet is hashed multiple times based upon its flow id, and corresponding counters are updated. If the minimum value of the updated counter is no larger than S, the packet is passed onto the sampling module, otherwise it is discarded. Note that by virtue of the modulo L operation, if the counter value exceeds L, it is reset to zero.

5.1.1. Catching heavy hitters

Estan and Varghese [25] applied the Bloom Filter to network measurement problem of detecting heavy hitters in traffic. Each packet entering a router is hashed k times into a counting Bloom Filter. The counters are incremented by the number of bytes in the packet. If all counters hashed by the packet exceed a certain threshold, the corresponding flow is placed in a table of heavy hitters. Large flows can thereby be detected with a small amount of space and a small number of operations per packet. However, the small number of counters in this situation causes false positives where either small flows happen to map into the same locations of large flows, or several small flows sum up to pass the filter. In order to reduce the false positives significantly, no counter is incremented by more than the size of the minimum among all k counters plus the current packet size. This conservative update reflects the most possible flow size, and reduces the possibility of small flows raising the counter values over the threshold.

Although the number of counters used in this scheme can be bounded due to the small number of heavy flows in a relatively short measuring interval, the width of the counters has to be wide enough to accommodate the large flow size in bytes. Furthermore, every subsequent packet from those identified heavy flows has to be processed so that counters be incremented and flows table updated.

We can further reduce the resources needed to catch heavy hitters by the aid of circular counter design. Using 1 byte wide counter, we configure the most significant bit as a flag bit, and the rest 7 bits as counter bits. The flag bit is set once the counter value reaches L. The cCBF counter updating process is illustrated in Fig. 8. In the example, we let $S = 0$ and $L = 100$, which means a packet is sampled if and only if after all the counters corresponding to the flow ID f saturate. The process works as follows.

When the packet arrives, the three corresponding counters contain values of 37, 99, and 2, respectively. The top and bottom flags are set (*i.e.,* 1), the middle flag is unset (*i.e.,* 0). Obviously the middle counter contains the minimum value among all counters. In the first step to update counters, we increment all the three counters by 1. Since the middle counter value reaches L, which is 100 in this case, we reset the counter to 0 and set the flag bit to 1. These operations marks the point where all the counters have saturated because the last unset flag is just set to 1. Therefore, we reset all three flag bits to 0 in step 2, and increment the flow size by $L = 100$ in setup 3. When a flow is terminated, which is not shown in the figure, we check the all the corresponding counters for the smallest residual counter value, and add it on top of the flow size in the flow table.

Following this counter updating process, we are able to catch large flows with size L and above. The circular counter and its module operation ensure a static sampling rate of $1/L$ for each heavy hitters, and only $1/L$ operations on Flow Table look-up and update compared to Estan's design [25]. Note the overall effective packet sampling rate for the whole traffic is much less than $1/L$ due to the fact that large flows only contribute to a small percentage in the flow size distribution. We could also adapt cCBF to count flow size in bytes with a wider counter and larger L value.

Fig. 9 shows the top 50 heavy hitter flows caught by cCBF. In the evaluation, we apply cCBF on the BB-West trace for an interval of 5 min. This measurement epoch sees more than 6 million packets in about 65 K flows. The parameters for cCBF are set at $(S, L) = (0, 100)$ to catch flows of 100 packets and above. We achieve very accurate flow size estimations with little errors for the top heavy hitters using cCBF of a mere 128 KB in size, which is less than 2% of the counter size used in FFS evaluation. Resetting counters at the beginning of each measurement interval is not necessary for cCBF, however, it helps maintaining strong performance. Since the cCBF design is an enhancement to FFS for efficient flow size estimation, it is also feasible to add a second stage sampling module to further reduce processing load.

5.2. Header-based hierarchical Bloom Filter

As shown above, cCBF catches heavy hitters with small memory fingerprint due to the Zipf's law nature of the flow size distribution. On the other hand, it is desirable to reduce the likelihood of sampling mid-sized to large flows

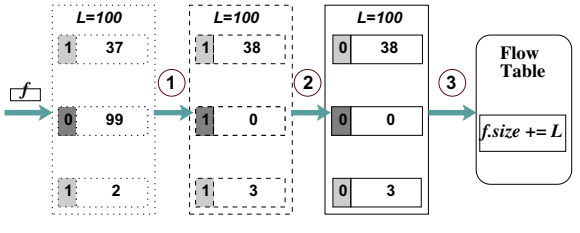


Fig. 8. An exemplary cCBF updating process: flow size is incremented by L after all counters saturate.

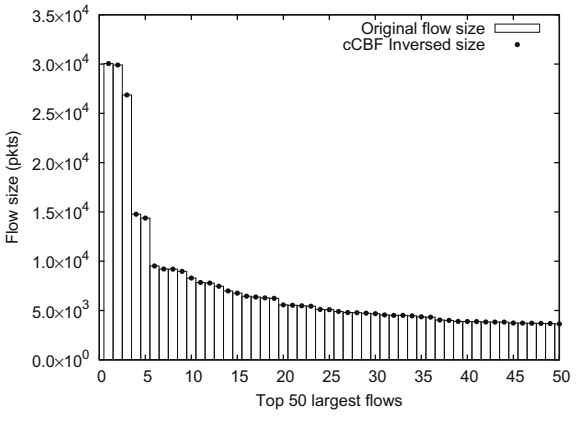


Fig. 9. Catching heavy hitters by cCBF: flow size estimation with little errors for the top 50 largest flow.

since they have been shown to act as a source of noise as far as anomalies are concerned [1]. Hence, we would like to detect as many unique small flows as possible for anomaly detection. One plausible usage of such a data set, for example, is to mining for potential malicious host or network addresses with large fan-out/fan-in values. However, it is impractical and unnecessary to keep per-flow counters for all the flows, since most of them are just small ones with no more than a dozen of packets. Our design goal in this section is to sample exact one packet from each flow, thus generate a flow set of all the unique flows in a measurement epoch for user query and anomaly detection purposes.

When sampling packets or flows with a Bloom Filter or a counting Bloom Filter, the false positives of the Bloom Filter affects the performance of the sampling schemes negatively. For example, in order to sample unique flows, the false positives translate into missing packets or flows, because the membership testing indicates the packets or flows being sampled already. With limited fast memory in routers and large number of packets and flows to sample, that is, capped m/n value, the only way to reduce the false positive ratio of the Bloom Filter is by increasing k in Eq. (9) (assuming $k \leqslant (m/n) \cdot \ln 2$). However, larger k exaggerates the processing load and may not lower the false positive rate by much. Table 6 shows if $m/n \geqslant 8$, even when we double k from 3 to 6, the false positive ratio is only reduced by less than a percent.

Hierarchical Bloom Filter (HBF) [37] was proposed for a payload attribution system that attributes reasonably long excerpts of payloads to their source and/or destination hosts. An HBF creates compact digests of payloads and provides probabilistic answers to membership queries on the excerpts of payloads, and performs superior to a basic block Bloom Filter. We design a similar data structure called header-based hierarchical Bloom Filter (hHBF) for the purpose of sampling unique flows, based on non-mutable header fields: *source IP* (**srcIP**), *source port* (**srcPt**), *destination IP* (**dstIP**), *destination port* (**dstPt**), and *protocol* (**prot**). Fig. 10 illustrates a simple example of such a hierarchy. In this example, fields "**srcIP**", "**srcPt**", "**dstIP**", and "**dstPt**" are blocked into the bottom level of the hierarchy. Then "**srcIP srcPt**" and "**dstIP dstPt**" are inserted at middle level, and "**srcIP srcPt dstIP dstPt prot**" at the top level.

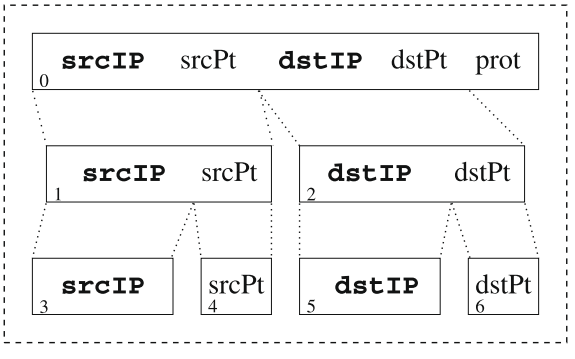


Fig. 10. The hierarchy of non-mutable header fields srcIP, srcPt, dstIP, dstPt, prot in hHBF.

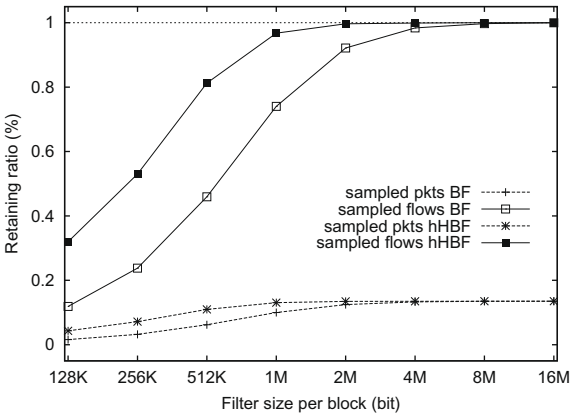


Fig. 11. Percentage of unique flows and packets captured by hHBF compared to a basic Bloom Filter.

Thus, if a field has occurred in the packet header, going one level up in the hierarchy allows us to verify whether the fields occurred together.

5.2.1. Capturing unique flows

Aggregating hashing results from multiple queries within a particular level and from multiple levels in the hierarchy, we can improve the confidence of the result and reduce the impact of collisions in hHBF. We compare the performance of hHBF against a basic Bloom Filter on the five tuples with no hierarchy for the same measurement interval tested in Section 5.1.1. Fig. 11 demonstrates the performance enhancement. For this 5 min epoch, the percentage of unique flows sampled by hHBF of 1MB is above 95% of all the original flows, which is 20% higher than a basic Bloom Filter. Since FFS uses a single hash function, it is not surprising that the performance of FFS (even without sampling, *i.e.*,, $p = 1$) in terms of capturing unique flows is below that of the basic Bloom Filter. As a result, hHBF achieves much better performance when counter size is limited to around 512 K $\sim$ 1 M.

It is obvious that the hierarchical nature of the hHBF resolves collisions automatically. In addition, hHBF mappings can also be saved for convenient query and limited pattern matching. Suppose we would like to verify if we have actually seen a subset of flows with ID "srcIP srcPt * dstPt". As in a basic Bloom Filter, the flow ID needs to be constructed by trying all possible "dstIP"s and "prot"s. However, in hHBF we can simply break the flow ID into two parts "srcIP srcPt" in block 1 and "dstPt" in block 6 in Fig. 10, and perform two individual queries in the middle and the bottom levels. Another advantage of hHBF is the unique flow set generated, which can be used for online or offline data mining and anomaly detection.

6. Conclusions

In this paper, we presented a low-complexity sampling technique, Fast Filtered Sampling (FFS), to achieve "flow-size-dependent sampling". This type of sampling was shown previously to improve flow distribution statistics

through "appropriate" adjustment of sampling rates that are inversely proportional to the flow size. In this work, we demonstrated that it also allows for accurate *anomaly detection* while still providing reasonable estimation of heavy hitters and traffic usage. Specifically, previous work had shown that information about small flows, which is critical for detection of anomalies, can be corrupted due to preferential sampling of medium-sized flows (*flow bias*). Consequently, relatively higher suppression of medium-sized flows over small flows is desirable from this perspective, which is one of the motivations leading to the design of FFS.

The main contribution of this work is the design of an extremely low complexity filter that can act in conjunction with existing random packet sampling mechanisms to achieve better estimation accuracy of both small and large flows with low resource consumption. Our filter requires only one update per packet and its decoupled design allows it to work independently of sampling modules in existing routers. In fact, it typically lowers the load seen by the sampling module, while retaining more relevant information, thus allowing for the possibility of increasing the sampling rate to achieve higher accuracy. We also provide a simple design methodology for the choice of s and l, which are the two control parameters of the filter. Through extensive evaluations using traffic traces collected from a tier-1 provider, we showed that FFS is very effective in eliminating false positives for portscan detection while performing comparably to random packet sampling for traditional applications like identifying heavy hitters with less error.

The broad motivation behind FFS is to design a low-complexity mechanism that can provide some degree of control over the sampling rate perceived by a flow as a function of its size. On one hand, we would like to detect large flows since they are useful for billing and traffic engineering purposes. From the perspective of anomaly detection, it is beneficial to have the complete flow IDs to detect those IP addresses with large fan-out/fan-in values that are usually candidates for potential anomalies. Therefore we extend FFS to two more specialized filtered sampling designs with various Bloom Filters suitable for different purposes with better accuracy and performance. A circular counting Bloom Filter (cCBF) monitoring the flow sizes of heavy hitters is introduced. cCBF consists of a counting Bloom Filter and a sampler. The counters increments in a *modulo* fashion, which reduces the size significantly. With counters a byte wide at most, cCBF accurately captures all the heavy hitters that are important for accounting and traffic engineering purposes. Next, we propose a header-based hierarchical Bloom Filter (hHBF) to sample one packet from each unique flows to form a flow set for user queries and anomaly detection. hHBF hashes five tuples in a hierarchy to reduce the impact of collisions. Evaluation shows hHBF can capture almost all the unique flows in a moderate measurement interval with a relatively small memory fingerprint.

We are currently exploring even more extensions as well as variations of our filter designs. For example, while the focus of this work was on detection of small and very large flows, one could envision applications that focus on only "medium-sized flows". In such a scenario, our filter could easily be converted into a "bandpass" filter to allow finer resolution of such flows. Identification of such variants as well as appropriate "inversion" techniques for our scheme will be part of our future work.

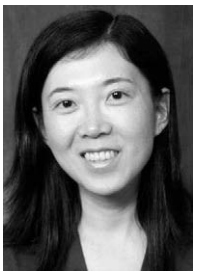
References

[1] J. Mai, A. Sridharan, C.-N. Chuah, H. Zang, T. Ye, Impact of packet sampling on portscan detection, IEEE Journal on Selected Areas in Communication (December) (2006).
[2] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, H. Zang, Is sampled data sufficient for anomaly detection? in: ACM Internet Measurement Conference (IMC) 2006, Rio de Janeiro, Brazil, October 2006.
[3] N. Duffield, C. Lund, M. Thorup, Properties and prediction of flow statistics from sampled packet streams, in: Proceedings of the ACM SIGCOMM IMW'02, Marseille, France, November 2002.
[4] C. Estan, G. Varghese, New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice, ACM Transactions on Computer Systems 21 (3) (2003) 270–313.
[5] A. Kumar, J. Xu, Sketch guided sampling – using on-line estimates of flow size for adaptive data collection, in: Proceedings of IEEE Infocom 2006, Barcelona, Spain, April 2006.
[6] Cisco IOS SoftwareNetFlow, %3chttp://www.cisco.com/warp/public/732/Tech/nmp/netflow/%3e.
[7] D. Plonka, FlowScan: a network traffic flow reporting and visualization tool, in: Proceedings of the USENIX LISA, 2000.
[8] P. Phaal, S. Panchen, N. McKee, InMon Corporation's sFlow: a method for monitoring traffic in switched and routed networks, RFC 3176, 2001. [Online] Available from: %3chttp://www.ietf.org/rfc/rfc3176.txt%3e.
[9] Psamp, %3chttp://www.ietf.org/html.charters/psamp-charter.html%3e.
[10] D. Chiou, B. Claise, N. Duffield, A. Greenberg, M. Grossglauser, P. Marimuthu, J. Rexford, A framework for packet selection and reporting, draft-ietf-psamp-framework-10.txt.
[11] T. Zseby, M. Molina, N. Duffield, S. Niccolini, F. Raspall, Sampling and filtering techniques for ip packet selection, draft-ietf-psamp-sample-tech-06.txt.
[12] http://www.ietf.org/html.charters/ipfix-charter.html.
[13] G. Sadasivan, N. Brownlee, B. Claise, J. Quittek, Architecture for ip flow information export, draft-ietf-ipfix-architecture-06.
[14] J. Quittek, S. Bryant, J. Meyer, Information model for ip flow information export, draft-ietf-ipfix-info-06.
[15] N. Hohn, D. Veitch, Inverting sampled traffic, in: Proceedings of the ACM SIGCOMM IMC'03, Miami Beach, Florida, USA, October 2003.
[16] C. Estan, K. Keys, D. Moore, G. Varghese, Building a better NetFlow, in: Proceedings of SIGCOMM'04, Portland, Oregon, USA, August 2004.
[17] B.-Y. Choi, J. Park, Z.-L. Zhang, Adaptive random sampling for traffic load measurement, in: Proceedings of the IEEE international conference on communications (ICC'03), Anchorage, Alaska, USA, May 2003.
[18] N. Duffield, C. Lund, M. Thorup, Flow sampling under hard resource constraints, in: Proceedings of the SIGMETRICS/PERFORMANCE'04, New York, NY, USA, 2004, pp. 85–96.
[19] N. Duffield, M. Grossglauser, Trajectory sampling for direct traffic observation, IEEE/ACM Transactions on Networking 9 (3) (2001) 280–292. June.
[20] A. Kumar, M. Sung, J. Xu, J. Wang, Data streaming algorithms for efficient and accurate estimation of flow size distribution, in: Proceedings of SIGMETRICS/Performance'04, New York, NY, USA, June 2004.
[21] Q. Zhao, A. Kumar, J. Xu, Joint data streaming and sampling techniques for detection of super sources and destinations, in: Proceedings of USENIX/ACM Internet Measurement Conference (IMC 2005), Berkeley, CA, USA, October 2005, pp. 77–90.
[22] A. Kumar, M. Sung, J. Xu, E.W. Zegura, A data streaming algorithms for estimating subpopulation flow size distribution, in: Proceedings of ACM SIGMETRICS'05, June 2005.
[23] S. Venkataraman, D. Song, P. Gibbons, A. Blum, New streaming algorithms for superspreader detection, in: Proceedings of Network and Distributed Systems Security Symposium (NDSS'05), San Diego, CA, USA, February 2005.
[24] S. Singh, C. Estan, G. Varghese, S. Savage, Automated worm fingerprinting, in: Proceedings of the ACM/USENIX Symposium on

Operating System Design and Implementation, OSDI'04, San Francisco, CA, USA, December 2004, pp. 45–60.
[25] C. Estan, G. Varghese, New directions in traffic measurement and accounting, in: Proceedings of ACM SIGCOMM'02, Pittsburgh, PA, USA, August 2002.
[26] A. Kumar, J.J. Xu, L. Li, J. Wang, Space-code bloom filter for efficient traffic flow measurement, in: Proceedings of ACM IMC'03, Miami Beach, FL, USA, October 2003.
[27] Y. Lu, S. Dharmapurikar, A.K. Kabbani, A. Montanari, B. Prabhakar, Counter Braids: an efficient minimum-space statistics counter architecture, in: Proceedings of ACM SIGMETRICS'08, Annapolis, MD, USA, June 2008.
[28] A. Ramachandran, S. Seetharaman, N. Feamster, V. Vazirani, Fast monitoring of traffic subpopulations, in: Proceedings of ACM IMC'08, Vouliagmeni, Greece, October 2008.
[29] L. Yuan, C.-N. Chuah, P. Mohapatra, ProgME: towards programmable network measurement, in: Proceedings of ACM SIGCOMM, Kyoto, Japan, August 2007.
[30] H. Madhyastha, B. Krishnamurthy, A generic language for application-specific flow sampling, ACM SIGCOMM Computer Communication Review 38 (2) (2008).
[31] Q. Zhao, J. Xu, Z. Liu, Design of a novel statistics counter architecture with optimal space and time efficiency, in: Proceedings of the ACM SIGMETRICS IFIP Performance 2006, June 2006.
[32] N. Duffield, C. Lund, Predicting resource usage and estimation accuracy in an IP flow measurement collection infrastructure, in: Proceedings of the IMC '03, Miami Beach, Florida, USA, October 2003.
[33] A. Sridharan, T. Ye, S. Bhattacharyya, Connectionless port scan detection on the backbone, in: Malware Workshop held in Conjunction with IPCC, Phoenix, Arizona, USA, April 2006.
[34] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM 13 (7) (1970) 422–426.
[35] L. Fan, P. Cao, J. Almeida, A.Z. Broder, Summary cache: a scalable wide-area web cache sharing protocol, IEEE/ACM Transactions on Networking 8 (3) (2000) 281–293.
[36] A. Broder, M. Mitzenmacher, Network applications of bloom filters: a survey, Internet Mathematics 1 (4) (2005) 485–509.
[37] K. Shanmugasundaram, H. Bronnimann, N. Memon, Payload attribution via hierarchical bloom filters, in: Proceedings of ACM CCS'04, Washington, DC, USA, October 2004.

Jianning Mai received his B.E. from Tsinghua University, Beijing, China in 1995, and M.E. from National University of Singapore, Singapore in 1998, and his Ph.D. from Department of Electrical and Computer Engineering, University of California, Davis under Professor Chen-Nee Chuah in 2008. Before joining UCD in 2003, he worked as networking protocol software developer at IBM Singapore, and several start-ups in Silicon Valley. He is now working as a pentent engineer.

Ashwin Sridharan (S'99, M'04) has been a research scientist with the Sprint Applied Research Group since 2004. He obtained his Masters from the Indian Institute of Science, Bangalore in 1999 and his Ph.D. from University of Pennsylvania in 2004 where he worked on traffic engineering and routing algorithms. His current focus is on performance enhancement of wireless networks, fast and accurate sampling and more generating data mining of network information. He has participated as a chair, committee member and reviewer in various networking conferences.



Hui Zang is a research scientist at Sprint Research, Burlingame, CA, USA. She received her B.S. degree in computer science from Tsinghua University, Beijing, China in 1997 and the M.S. and Ph.D. degrees in computer science from the University of California, Davis in 1998 and 2001, respectively. She joined Sprint in 2000. Dr. Zang is the author of the book "WDM Mesh Networks - Management and Survivability" (Kluwer Academic, 2002). She has published over 50 conference papers and journal articles and currently has three US patents issued and over a dozen pending in the field of networking and communications. Her research spans across different network layers and both wired and wireless networks. She has led or participated in research projects in the context of network measurements, resource allocation, cross-layer adaptation, wireless and IP network security, and mobile user behavior analysis. Dr. Zang has been a senior member of IEEE since 2004.

Chen-Nee Chuah is currently an Associate Professor in the Electrical and Computer Engineering Department at the University of California, Davis. She received her B.S. in Electrical Engineering from Rutgers University, and her M.S. and Ph.D. in Electrical Engineering and Computer Sciences from the University of California, Berkeley. Her research interests lie in the area of computer networks and wireless/mobile computing, with emphasis on Internet measurements, network anomaly detection, network management, multimedia, online social networks, and vehicular ad hoc networks. She received the NSF CAREER Award in 2003, and the Outstanding Junior Faculty Award from the UC Davis College of Engineering in 2004. In 2008, she was selected as a Chancellor¥¯s Fellow of UC Davis. She has served on the executive/technical program committee of several ACM and IEEE conferences and is currently an Associate Editor for IEEE/ACM Transactions on Networking.

Please cite this article in press as: J. Mai et al., Fast Filtered Sampling, Comput. Netw. (2010), doi:10.1016/j.comnet.2010.01.015