Shared-State Sampling*

Frederic Raspall, Sebastia Sallent and Josep Yufera
Dept. of Telematics, Technical University of Catalonia (UPC)
fredi@entel.upc.es, sallent@entel.upc.es, yufera@entel.upc.es

ABSTRACT

We present an algorithm, *Shared-State Sampling* (S^3), for the problem of detecting large flows in high-speed networks. While devised with different principles in mind, S^3 turns out to be a generalization of two existing algorithms tackling the same problem: *Sample-and-Hold* and *Multistage Filters*. S^3 is found to outperform its predecessors, with the advantage of smoothly adapting to the memory technology available, to the extent of allowing a partial implementation in DRAM. S^3 exhibits mild tradeoffs between the different metrics of interest, which greatly benefits the scalability of the approach. The problem of detecting frequent items in streams appears in other areas. We also compare our algorithm with proposals appearing in the context of databases and regarded superior to the aforementioned. Our analysis and experimental results show that, among those evaluated, S^3 is the most attractive and scalable solution to the problem in the context of high-speed network measurements.

Categories and Subject Descriptors: C.2.3 [Computer-Communication Networks]: Network Operations—traffic measurement, identifying large flows

General Terms: Algorithms, Measurement

Keywords: Per-flow measurements, scalability

1. INTRODUCTION

Network measurements are essential for a number of network management tasks like *traffic engineering* [5], the detection of *hot-spots* or *DoS* attacks [11] or *accounting*. Within traffic measurements, the *per-flow* approach provides a reasonable tradeoff between the amount of information acquired and its usefulness [16]. Unfortunately, the major drawback of the per-flow approach is its lack of scalability: link speeds improve 100% per year, while the speed of memory devices improves at a much slower pace ($7 - 9\%$) [2]. The only memory technology that can keep up with the increasing number of flows is DRAM. However, DRAM devices are already too slow for existing high-speed line rates. SRAM is the only memory technology allowing for per-packet updates at current line speeds. Unfortunately, SRAM devices are expensive, power-hungry and have limited capacity. This, together with the observation in several studies that a large share of the traffic is due to a small fraction of the total number of flows [4], have motivated the search for solutions concentrating on the measurement of *large flows*. Focusing on such flows not only suffices for many applications but may also allow for new ones: e.g. *threshold accounting* or *scalable queue management* [2].

Pioneering work by Estan and Varghese [2] showed that it was possible to *identify* or *detect* such *heavy-hitters* using slightly more memory than that required in case such flows were *known beforehand*. While subsequent research has tackled the problem of measuring flows simultaneously considering different *flow definitions* [3], the problem for predefined flow definitions is still open from a research point of view and of great practical interest. This paper contributes with a scalable algorithm, *Shared-State Sampling*(S^3), for the detection of large flows in high-speed networks. The paper is structured as follows. Section 2 summarizes previous work focusing on those our work shall be compared to. In section 3, we present our algorithm and we analyze it in section 4. Section 5 discusses how S^3 could be implemented in high-speed routers. Besides being of interest *per se*, the discussion shows the versatility of S^3 and helps understanding its scalability, an issue that we finally address in section 6. Section 7 presents some results obtained with software implementations of the algorithms and real traffic traces. While the focus is on the performance of S^3, we compare to other proposals in the literature. An exhaustive evaluation of all algorithms is not possible for space reasons. Nevertheless, the experiments presented suffice for a conceptual comparison identifying the weaknesses and strengths of the algorithms. Section 8 concludes the paper.

2. RELATED WORK

The problem of detecting *frequent* or *hot* items in streams has attracted considerable attention in the past, specially in the field of database management and data mining. In this context, Toivonen [17] proposes algorithms that use uniform random sampling for the discovery of *association rules*, which require a single pass as compared to prior work. Manku and Motwani [7] propose two algorithms, *Sticky Sampling* (S^2) and *Lossy Counting* (LC), for computing frequency counts exceeding user-specified thresholds. Cormode

*This work has been funded by grant CICYT TSI2005-06092 from the Spanish Ministery of Science and Education.

IMC'06, October 25–27, 2006, Rio de Janeiro, Brazil.

and Muthukrishnan [1] propose algorithms able to identify *hot* items in databases in a single pass while allowing for deletions. To the best of our knowledge, *Sample-and-Hold* (*S&H*) and *Multistage Filters*(*MF*) [2], while showing some similarities with previous work in the context of databases [6], were the first algorithms proposed in the context of network measurements. [10] [8] [9] address related problems, but their focus is not on detecting large flows. In what follows, we summarize the ideas behind the proposals that we compare with, while reviewing the problem at hand.

S&H and *MF* aim at identifying, within the total traffic present in some *measurement interval*[1], *large flows*, defined as those with size above some *threshold* (such as T bytes) or above some fraction of the total (e.g. $z = 1\%$). *Flows* are understood as sets of packets with common properties. The specific properties considered, the so-called *flow definition* or *pattern*, is application-specific[2]. Given a *flow definition*, each flow is uniquely identified by the specific values the properties within the definition take, what is called the *flow identifier* (FID) or *key*. The idea in *S&H* and *MF* is the following: *there can be at most* $1/z$ *large flows in a measurement interval. Thus, if these can be identified, small but fast memories can be used to individually track such flows by keeping a dedicated entry in the so-called flow memory: not only small but fast memories suffice but also, accuracy is improved since bytes arriving after detection are exactly counted. Further, the amount of measurement data to report is drastically reduced.* This is a paradigm shift compared to previous techniques (e.g. *Sampled NetFlow*), that use large (but slow) DRAM memories and alleviate the gap between link and memory bandwidth through sampling. In both algorithms, *flows are challenged to instantiate entries in the flow memory according to their size. S&H* does this by means of sampling and is memoryless and random: *flows must be sampled once to get detected.* On the contrary, *MF* is deterministic and stateful: *flows must increase some counters beyond some threshold before entering the flow memory.*

S^2 and *LC* in [7] resemble *S&H* and *MF* in that, once *item types* (i.e. flows) are *identified*, dedicated entries are supported, which get updated with subsequent occurrences of such types. *LC* splits the stream into fixed-length *buckets* where items either update or instantiate entries. In addition, entries for infrequent items are *pruned* at bucket boundaries. S^2 prunes entries at the end of variable-length buckets and only sampled items instantiate entries, where the sampling rate varies over the length of the stream. *LC* and S^2 *relax the challenge put on flows to enter the flow memory so as to improve accuracy and compensate the increased number of enries occupied by means of pruning.* Hence, the challenge put on flows is not only *entering* but also *remaining* at the synopsis structure. While both algorithms have been suggested to outperform *S&H* and *MF*, no evaluation with real traffic traces has, to our knowledge, ever been conducted.

An ideal algorithm should report, at the end of a *measurement interval*, the FID and size of *only those flows above the threshold.* Less ideal algorithms may fail in three ways: **i)** missing to report some large flows (*false negatives*) **ii)** reporting flows below the threshold (*false positives*) and **iii)** providing inaccurate estimates for large flows. Consequently, when assessing the performance of this type of algorithms the focus will be on: first, how high the *detection probability* for large flows is; second, how *accurately* can their sizes be estimated and, third, the overall amount of memory required. Due to the presence of *false positives*, more than $1/z$ entries may have to be supported in the *flow memory*: *false positives* may prevent large flows from being measured (in spite of them being *detected*), if these exhaust the entries available. The amount of memory required by algorithms, in excess to the minimum, is related to their ability to avoid *false positives.* In *S&H* and *MF*, this depends on *how much can algorithms* ***prevent*** *small flows from entering the flow memory.* Although S^2 and *LC* eliminate false positives *when queried*, their memory requirements are determined by the number of entries that may have active at any point, which depends on *how good these algorithms are at* ***expelling*** *small flows from the flow memory.* Finally, we shall also be concerned with algorithm's per-packet processing times, which ultimately depend on the number of memory accesses required: *too many accesses may preclude the implementation of the algorithms or limit their scalability even if the fastest memories are used.*

3. SHARED-STATE SAMPLING

The algorithm that we shall now present, *Shared-State Sampling* (S^3), is based on the following principle: *if the number of flows in a link is large and most of the traffic is due to a small fraction of these flows then, most of the flows must be small. Therefore, making it harder for these many small flows to get captured may be a reasonable strategy to avoid false positives.*

The major drawback of *S&H* is the strong tradeoff it exhibits between accuracy and space requirements: *to achieve reasonable accuracy, S&H must oversample, which causes the detection of small flows thereby increasing the memory requirements.* This is so since oversampling by a factor O reduces the *threshold* by a factor $1/O$. In this regard, we should carefully distinguish between the threshold of the problem definition, T, that classifies flows into *small* or *large* and the threshold of the algorithm, which is the size of a flow beyond which the *detection probability* is higher than some prescribed value. In *S&H*, flows of size $1/p$ bytes (with p the byte sampling probability) are detected with probability ≈ 0.63. S^3 aims at improving the *detection curve* of *S&H* by making it closer to that of an ideal algorithm, which should be $P_{det}^{ideal}(v) = u(v - T)$, with $u(v)$ the discrete step function, T the threshold and v the flow size. The key issue is ***how to achieve a decrease in the detection probability for small flows, without affecting the detection probability for large flows.***

S^3 works as follows. For each arriving packet, it performs a lookup in the *flow memory* to check whether that flow has *already* been captured. In that case, the corresponding dedicated byte counter is updated. That is, as in *S&H* and *MF*, packets of flows already captured *bypass* the "*challenge*" or *filter*. Packets of flows not (yet) detected are sampled with some probability. If a flow is sampled d times then, an entry is created in the flow memory. In order to count the number of times that each flow has been sampled, a set of counters are supported. When a packet gets sampled, these counters are examined: if these hold a value of $d - 1$, this makes d

[1] *Measurement intervals* can span different timescales: for instance, for traffic engineering these may be in the order of minutes to hours whereas for queue management these could be in the order of seconds.

[2] E.g., for traffic matrices estimation, flows may be defined as *packets with same source and destination prefixes*, whereas for identifying TCP DoS attacks, the focus may be on *TCP packets with some flags set and distinguished by addresses.*

samples altogether and an entry for this flow is created in the flow memory; otherwise, the counters get incremented to remember the number of samples taken so far. That is, our algorithm counts the number of times flows get sampled and entries in the *flow memory* should only be created for flows sampled d times. We refer to d as the *sample threshold*. The way in which samples are *obtained* and *counted* is explained in what follows, together with the supporting analysis.

4. ANALYSIS OF OUR ALGORITHM

4.1 The "single-flow" case

Let us start analyzing what happens in case there were only one flow. Suppose that the algorithm sampled every byte with probability p and that samples from a flow incremented "some" counter until d samples were taken. Sampling each byte with probability p, the first $d-1$ samples will increment the counter and the d^{th} sample (if there) will get to the flow memory. Clearly, the flow will not be detected if fewer than d samples from it are taken. Hence, the detection probability for a flow of size v bytes, $P_{det}^{\infty}(v)$, is the probability that we take at least d samples had we sampled all of its bytes (the ∞ superindex shall later become clear). Since each byte is independently sampled, the number of samples that we would take, N_v, is a *binomial* r.v. and we can write

$$P_{det}^{\infty}(v) = P\{N_v \geq d\} = \sum_{j=d}^{v} \binom{v}{j} p^j (1-p)^{v-j} \qquad (1)$$

Alternatively: the number of bytes that pass before we get the first sample is a *geometric*(p) r.v. Thus, the number of bytes that go before the d^{th} sample is taken, S_d, is the sum of d i.i.d geometric r.v., which has a *negative binomial* distribution. Therefore, the probability that a flow of size $v \geq d$ gets detected is the probability that the number of bytes required to take d samples be no larger than v. Think of bytes as attempts to get a sample: a flow is detected if fewer attempts than the available, v, suffice to get d samples. The probability that it takes exactly j attempts until the d^{th} sample is obtained is $P\{S_d = j\} = \binom{j-1}{d-1} p^d (1-p)^{j-d}$, where $j \geq d$. Thus, if $v < d$, the detection probability for a flow is 0 and

$$P_{det}^{\infty}(v) = P\{S_d \leq v\} = \sum_{j=d}^{v} \binom{j-1}{d-1} p^d (1-p)^{j-d} \qquad (2)$$

if $v \geq d$. Let us see what the role of the *sample threshold* is. The first observation is that, for $d = 1$, only one sample is needed to *capture* a flow. Further, no counters are required (no previous sample needs to be "*remebered*") and $P_{det}^{\infty}(v)$ in (1) and (2) become $1-(1-p)^v$, which is the *detection probability* with $S\&H$ since, then, both algorithms are identical.

Clearly, the detection probability for a flow decreases when we increase d for the same p, since more samples from the flow are required. This is shown in fig. 1(bottom-right) which plots $P_{det}^{\infty}(v)$ for different values of d. The key observation is that, as d increases, $P_{det}^{\infty}(v)$ has an *inflection point* as compared to the curve of $S\&H$, *which does not have it regardless of* p. Let us see what happens if p is chosen depending on d. If each byte is sampled with probability p and d samples from a flow are required, the flow will be detected, on the average, after $d\frac{1}{p}$ bytes, which is d times the average of a *geometric*(p) r.v. Let us see what happens

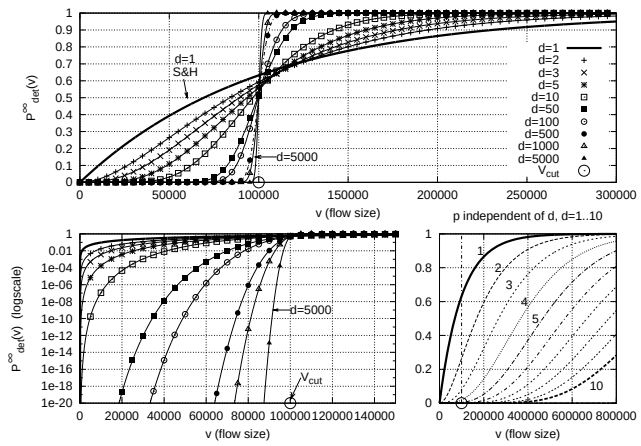


Figure 1: $P_{det}^{\infty}(v)$ **for several values of** d **when** $p = d\tilde{p}$**;** $1/\tilde{p} = 10^5$**. Bottom-right:** $P_{det}^{\infty}(v)$ **for** $p = \tilde{p}$ **and** $d = 1..10$**.**

if we set the sampling probability p so that, regardless of d, it takes, on the average, a constant amount of bytes, $1/\tilde{p}$, until a flow gets captured. It is straightforward to see that p has to be chosen as

$$p = \tilde{p}\ d \qquad (3)$$

Choosing p as in (3), the supposed *inflection point* will be located at $v = \frac{d}{p} = \frac{1}{\tilde{p}}$, *which does not depend on* d *either*. Fig. 1(top) shows this effect plotting $P_{det}^{\infty}(v)$ when p is chosen according to (3), for some values of d and $\tilde{p} = 10^{-5}$. As can be observed, d varies the steepness of $P_{det}^{\infty}(v)$. The following terms will be used throughout the remainder of the paper. We call the location of the *inflection point*, the ***cutoff size***, $V_{cut} = \frac{1}{p}$. The detection probability for a flow of V_{cut} bytes is approximately 0.63 for "small" values of d. In these terms, (3) can be written as $p = \frac{d}{V_{cut}}$. We call ***granularity***, z, the targeted capability of detecting flows no smaller than a fraction z of the total traffic in a measurement interval, C. That is, $z = T/C$. By ***selectivity*** we mean the power of discriminating (detecting) flows according to their size. Specifically, we mean the ability of algorithms to avoid detecting small flows while not compromising the detection of large flows.

Now, the reason for $P_{det}^{\infty}(v)$ exhibiting such behavior is the following. For a flow of size v, $E\{N_v\} = vp$ since N_v is *binomial(v,p)*. It is well known that the most likely values of a *binomial* r.v. are concentrated around its mean. If $v < V_{cut}$ and $p = d/V_{cut}$ then, the average number of samples taken $(\frac{v}{V_{cut}})d$ will always be smaller than d. Thus, the probability of the most likely values of N_v shall not be included in $P\{N_v \geq d\}$, thereby yielding a low detection probability. Conversely, if $v > V_{cut}$, $vp = (\frac{v}{V_{cut}})d$ shall always exceed d thereby raising $P_{det}^{\infty}(v, d)$, which will be high.

As of this preliminary analysis, we will set V_{cut} equal to the threshold T and use d to govern the *selectivity* of S^3. To see the gain in *selectivity* in S^3 compared to $S\&H$, fig. 1(bottom-left) plots $P_{det}^{\infty}(v)$ as before, but in logarithmic scale. As can be seen, the detection probability for flows above $V_{cut} = 10^5$ is very high in S^3, while being several orders of magnitude smaller than that with $S\&H$ for flows below V_{cut}; already for small values of d. Further, as d increases, $P_{det}^{\infty}(v)$ seems to approximate the detection curve of an ideal algorithm. Finally, note that d corresponds to the

average number of samples that would be taken from a flow of size V_{cut}.

4.2 The inflection point, the ideal algorithm

We shall now justify why V_{cut} is an *inflection point*. Specifically, the following discussion shows that, if $p = d/V_{cut}$, *selectivity* in S^3 improves as we increase d, in that *the detection probability for a flow increases if $v > V_{cut}$, decreases if $v < V_{cut}$ and remains approximately constant if $v = V_{cut}$.*

Let $N_{v,d}$ be the number of bytes sampled from a flow with v bytes when each is sampled with probability $p = \frac{d}{V_{cut}}$. The detection probability for a flow is $P\{N_{v,d} \geq d\}$. Note that, increasing d, the distribution of $N_{v,d}$ changes. As $binomial(v,p)$, $N_{v,d}$ is the sum of v i.i.d r.v., with mean $E\{N_{v,d}\} = vp$ and variance $\sigma^2_{N_{v,d}} = vp(1-p)$. The detection probability can be written as $1 - P\{N_{v,d} < d\}$, which is equivalent to

$$1 - P\left\{ \frac{N_{v,d} - E\{N_{v,d}\}}{\sigma_{N_{v,d}}} < \frac{d - E\{N_{v,d}\}}{\sigma_{N_{v,d}}} \right\} \tag{4}$$

By virtue of the *central limit theorem*, $Z_{v,d} = (N_{v,d} - E\{N_{v,d}\})/\sigma_{N_{v,d}}$ is, for large v, approximately a *standard normal* r.v., Z and (4) approximates $1-\Phi((d-E\{N_{v,d}\})/\sigma_{N_{v,d}})$ with $\Phi(x) = P\{Z < x\}$. Letting $p = \frac{d}{V_{cut}}$ in the expectation and variance of $N_{v,d}$, $P^{\infty}_{det}(v)$ can be approximated by

$$1 - \Phi\left(\sqrt{\frac{d}{V_{cut} - d}} \left(\frac{V_{cut} - v}{\sqrt{v}} \right) \right) \tag{5}$$

The square root in (5) monotonically increases in d. If $v < V_{cut}$, the term in parentheses is positive. Hence, increasing d, the argument of $\Phi(x)$ increases. Since $\Phi(x)$ monotonically increases with x, $1-\Phi(x) \approx P_{det}(v,d)$ decreases. Conversely, if $v > V_{cut}$, the argument of $\Phi(x)$ is negative and increasing in absolute value with d, meaning that $1-\Phi(x)$ will increase in d. When $v = V_{cut}$, the argument in $\Phi(x)$ is 0, and the detection probability slightly varies around 0.5.

LEMMA 1. *In the single-flow case, S^3 approximates an ideal algorithm as d approaches $V_{cut} = \frac{1}{p}$. In other words,*

$$\lim_{d \to V_{cut}} P^{\infty}_{det}(v) = u(v - V_{cut})$$

PROOF. As $d \to V_{cut}$, we *tend to require* V_{cut} samples from flows before detecting them. However, if $p = d/V_{cut}$, we also tend to sample every byte. Thus, when $p = 1$, every byte is sampled and therefore accounted: if $v < V_{cut}$, the number of samples *never* reaches the sample threshold (no detection); if $v \geq V_{cut}$ the threshold is *always* reached. □

4.3 The effects of limited memory

So far we have studied the detection probability in S^3 for the case with only one flow, $P^{\infty}_{det}(v)$. In reality, *we have studied the detection probability for any number of flows had the algorithm sufficient memory to **exactly** count the number of samples taken from every flow.* This is so since, in that case, the detection probability for a flow depends only on its size and on p given that the sampling process is *memoryless* and bytes get *independently sampled*. We call this, the *unlimited memory case* (hence, the reason for the superindex ∞). Unfortunately, the case will be that the number of flows is larger than the number of counters that can be afforded. Thus, some counters will have to *count* the number of samples taken from *several flows*. This may lead to flows below the threshold being detected: since counters are raised by other flows, the algorithm may think that a flow has been sampled more times than it really were. We call this effect, *interference*. For large flows the impact shall be small since $P^{\infty}_{det}(v)$ may already be close to unity. The problem may be for small flows, since the assumption is that there are many such flows. In what follows we study what happens when counters have to be *shared* among flows (hence, the name of our algorithm).

4.4 Keeping track of samples

The way samples are counted may be critical for S^3 to perform as in the *unlimited memory* case. Thus, we investigated several approaches. It turns out that supporting *several stages of counters*, where counters get assigned to flows by means of computing independent hash functions on packet FIDs (as in MF), is a good strategy. For space reasons, we do not provide the full discussion and the reader is referred to [15]. Thus, the remainder of the paper focuses on the performance of our algorithm when counters are arranged in m *parallel* stages: each sampled byte will increment by 1 a counter indexed by the result of computing a hash function on the packet FID, at each of the stages *until a sample finds **all** its stage counters at $d-1$*, in which case an entry shall be created for that flow. This completes the definition of the algorithm and the analysis motivating it. The next section studies how the detection probability is *when samples are counted using m stages of b counters.*

4.5 $P_{det}(v)$ arranging counters in stages

A flow must be sampled at least once to get detected. If a flow would have been sampled more than d times (which can only happen if $v > d$), *interference* would not help since it would be detected anyway. If a flow would have been sampled $0 < k < d$ times, it would be detected provided that *all* counters where it hashed to were, at least, $d - k$, even if $v < d$. Consequently, if N_v is the number of times a flow would be sampled, we have that:

$$P_{det}(v|N_v = k) = \begin{cases} 0 & k = 0 \\ 1 & k = d, \ldots v \\ P\{all \geq d-k\} & k = 1 \ldots d-1 \end{cases} \tag{6}$$

The detection probability for a flow of size v is maximum if it is the last one to arrive and all other bytes, $C - v$, enter the sampler. Let S be the number of samples taken out of these $C - v$ bytes. If $S = r$, the number of counters at $d - k$ is, at most, $\lfloor \frac{r}{d-k} \rfloor$. Thus, the probability that a flow sampled k times hashes to one of these counters when $S = r$ is bounded above by $\lfloor \frac{r}{d-k} \rfloor \frac{1}{b} \leq \frac{r}{(d-k)b}$. Since the hash functions are assumed *independent* and the same sampled bytes may increment a counter at each stage, the detection probability for a flow sampled $k \geq 1$ times is bounded above by $\left(\frac{r}{(d-k)b}\right)^m$ when exactly $S = r$ samples from the rest of the traffic are taken. Thus, in general we have that:

$$P\{all \geq d-k\} \leq \sum_{r=d-k}^{C-v} \left(\frac{r}{(d-k)b}\right)^m P\{S = r\} \tag{7}$$

where S is $binomial(C - v, p)$. As such, for large C, S will be very close to $(C - v)p < Cp = d/z$ with very high probability, by virtue of the *law of large numbers*. Thus, we can approximate $P\{all \geq d-k\} \approx P\{all \geq d-k \mid S \approx d/z\}$ and (7) as $P\{all \geq d-k\} \approx min\left(\left(\frac{d}{(d-k)zb}\right)^m, 1\right)$. Note that this suggests taking $b > 1/z$ for this probability to be

below 1. Considering this approximation, (6) and unconditioning from N_v, the detection probability for a flow with v bytes is bounded above by

$$P_{det}(v) \leq P^{\infty}_{det}(v) + \frac{1}{(bz)^m} \sum_{k=1}^{\mu} P\{n=k\} \Big(\frac{d}{d-k}\Big)^m \qquad (8)$$

where $\mu = min(d-1, v)$. That is, by the detection probability in case of unlimited memory, $P^{\infty}_{det}(v)$, plus a term, $\Delta P_{det}(v, d, z, C, b, m)$ representing the increase in the detection probability caused by other traffic incrementing our flow's counters. Note that, unlike in the *unlimited memory* case, flows smaller than d may get detected if sampled. The observation is that *selectivity* may no longer depend solely on d: it shall also depend on the degree of interference between flows, which depends on the amount of memory supported. Note the following tradeoff: *on one hand, increasing the sample threshold, the term $P^{\infty}_{det}(v)$ diminishes for flows below V_{cut}, thereby improving the selectivity. On the other, the byte sampling probability also increases, which translates into more samples from other flows increasing stage counters (ΔP_{det}), thereby potentially increasing the detection probability for small flows.* This suggests the existence of an optimum value for d. In addition, for $P_{det}(v)$ to be close to $P^{\infty}_{det}(v)$, the term $1/(zb)^m$ should be small. In particular, note that if $b \to \infty$ the rightmost term in (8) vanishes. Increasing m, the same factor also vanishes if $b > 1/z$. Let us now see the role of m and b.

Let S be the number of samples taken from the rest of flows and $\epsilon_i, i = 1..m$ the amount by which the counters (one at each of m stages) where our flow hashes to get incremented by other flows. We omit, for clarity, the subindex d but recall that the distribution of the above depend on d. The detection probability for our flow is maximum if it is the last one to arrive and the rest of traffic enters the sampler, in which case S is $binomial(C-v, p)$. The probability that a flow gets detected is no larger than the probability that, at all stages, $\epsilon_i + N_v \geq d$. Considering that, for any k, $E\{\epsilon_i | S = k\} = \frac{k}{b}$, we have that $E\{\epsilon_i\} = \frac{E\{S\}}{b} = \frac{(C-v)p}{b}$. Since N_v and S are independent, so are N_v and ϵ_i, $i = 1..m$. Hence, defining $Z_i = \frac{N_v + \epsilon_i - E\{N_v + \epsilon_i\}}{\sigma_{N_v + \epsilon_i}}$, the detection probability for a flow arriving last can be written as $P\{Z_i \geq \frac{d - vd/V_{cut} - (C-v)d/(V_{cut}b)}{(\sigma^2_{N_v} + \sigma^2_{\epsilon_i})^{1/2}}, \forall i = 1..m\}$. Following the same line of reasoning as in section 4.2, it can be proven that the *inflection point* in the detection curve at the end of the measurement interval is at $V_{infl} \approx V_{cut}(1 - \frac{1}{bz})$, regardless of m. This suggests the need to support a number of counters per stage at least several times $1/z$ for the detection curve to be *selective* throughout the measurement interval. The product zb can be written as $\frac{zC}{C/b}$. If we interpret $zC = V_{cut}$ as the minimum strength of a large flow and C/b as the strength of the interference, zb can be understood as the *signal-to-interference* ratio. Fig. 2 plots (8) for different stage sizes when 2 stages are used. Note how, even for the worst-case detection curve given by (8), V_{infl} well approximates the location of the inflection point. This is shown in fig. 2, where vertical lines are plotted at V_{infl} for each value of b, whose location lies close to the value of v where the convexity of $P_{det}(v)$ changes. Note how, for $b \approx 1/z = 100$, the inflection point disappears, whereas, for $b \gg 1/z$, $V_{infl} \approx V_{cut}$.

While m does not affect the location of the *inflection point*, it shall impact on the *shape* of the detection curve.

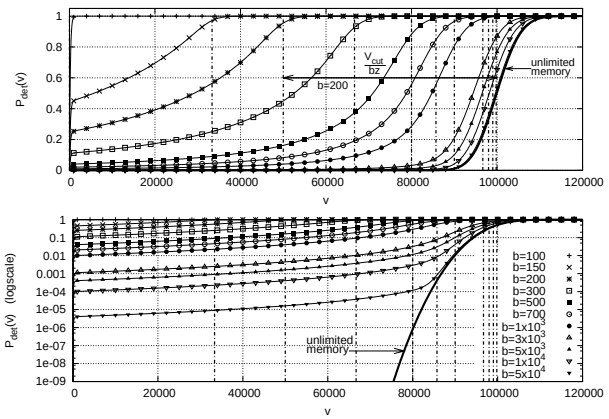


Figure 2: ***Worst-case detection probability (8) when*** $d = 500, V_{cut} = 10^5$ ***($z = 1\%$) and*** $m = 2$***, for different stage sizes.***

Specifically, the detection probability for flows much smaller than the threshold exponentially decreases with m. This suggests that, once the number of counters per stage exceeds several times $1/z$, *it is much more effective to add stages than to increase the size of the existing ones.* Fig. 3 shows the worst-case detection curve for a fixed stage size of 1000 counters, when different number of stages are used. Note that using $m = 20$ stages with 1000 counters is much more effective than using 2 stages with $b = 10^4$ counters, for the same amount of memory. The detection probability for a flow with $\approx 2\ 10^3$ bytes is $\approx 10^{-17}$ in the first case and $\approx 10^{-4}$ in the second. Finally, note how, for a large number of stages, $P_{det}(v) \approx P^{\infty}_{det}(v + \frac{V_{cut}}{zb})$. That is, flows at the end of the measurement interval *appear* as $V_{cut} - V_{infl} = V_{cut}/bz = C/b$ bytes larger.

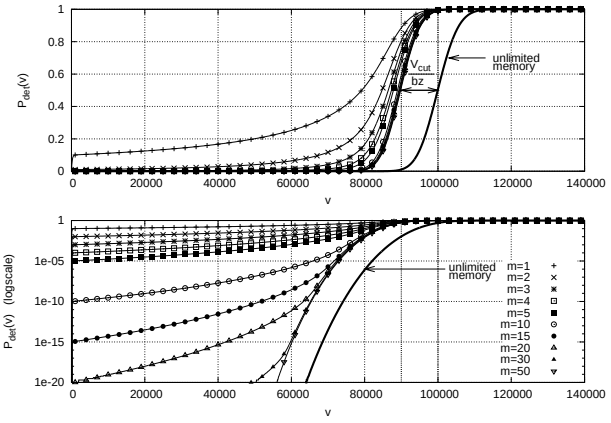


Figure 3: ***Effect of the number of stages,*** m***, in (8) for the same setup when stages are of*** $b = 1000$ ***counters.***

4.6 On the number of captured flows

In S^3, flows must be sampled at least once to get detected. Therefore, an upper bound for the detection probability for a flow of v bytes is the probability that it gets sampled at least once, i.e. $P_{det}(v) \leq 1 - (1 - \frac{d}{V_{cut}})^v$, which is the detection probability with $S\&H$ for an oversampling factor d and threshold V_{cut}. A first consequence of this observation is the following lemma, whose proof (that can be found in [15]) we omit for space reasons.

LEMMA 2. *For the same byte sampling probability, S^3 detects, on the average, fewer flows than S&H.*

The next theorem (also proven in [15]) bounds above the expected number of detected flows in S^3 when the number of counters supported at each stage is large enough. While tighter bounds for specific flow size distributions may be found, our bound is *independent of the flow size distribution.*

THEOREM 1. *With m stages of b counters each, if $b > \frac{1}{z}(d-1)^{1/m}, d > 1$ then, the expected number of flows detected by S^3 in a measurement interval is bounded above by $\frac{1}{z}\big(1+\frac{1}{(1+m)((bz)^m-1)}\big)$, regardless of the flow size distribution, packet size distribution, number of flows and amount of total traffic in a measurement interval.*

Recall that, for a granularity z, there can be at most $1/z$ large flows. Hence, if no assumption is to be made about the flow size distribution, the *flow memory* has to be at least $1/z$ entries. Thus, the term $\big(1+\frac{1}{(1+m)((bz)^m-1)}\big)$ can be understood as the factor by which we need to inflate the flow memory beyond the minimum so that, on the average, all captured flows find an entry available. The bound in theorem 1 shows that, regardless of the traffic mix (e.g. number of flows), the flow memory does not need to be overdimensioned if the amount of memory supported at the stages is sufficiently large. In this regard, if b is several times $1/z$ then, $(bz)^m - 1 \approx (bz)^m$, and the *excess memory* required decreases exponentially in m.

4.7 S^3 related to $S\&H$ and MF

While the discussion so far has focused on $d \in [1, V_{cut}]$, the complete analysis could have been formulated in terms of $p \in [1/V_{cut}, 1]$ since, in S^3, $p = \frac{d}{V_{cut}}$. When $d = 1$, S^3 behaves as $S\&H$ with $O = 1$ and $T = V_{cut}$. As d approaches V_{cut}, p tends to unity and then *every incoming byte for an undetected flow increases stage counters until one finds all stages at $d-1 = V_{cut}-1$.* Therefore, if sample counters are arranged in parallel stages, S^3 becomes MF when the sampling probability becomes 1. Hence, *$S\&H$ and MF can be understood as special instances of S^3 when d or p take their extreme values. At one edge, this gives a random memoryless algorithm, $S\&H$. At the other, a deterministic memoryful one, MF.*

In the *"middle"*, S^3 shall differ from MF in the following. In MF, counters may have to hold values as large as V_{cut} since the decision on creating entries is taken depending on whether counters reach such value. Thus, MF requires $\lceil log_2(V_{cut})\rceil$ bits per counter. In S^3, counters have to hold values of, at most, $d-1$ $\forall d \in [2, ..V_{cut}]$, thereby requiring $\approx \lceil log_2(d)\rceil = \lceil log_2(pV_{cut})\rceil$ bits per counter. Thus, for the same amount of memory, S^3 can support more counters than MF. This *memory gain* shall be approximated by $G_m = log_2(V_{cut})/log_2(d)$. Defined as such, G_m monotonically decreases for $d \in [2, ..V_{cut}]$. For $d = 1$ ($S\&H$), no counters are needed and $G_m = \infty$. For $d = V_{cut}$ (MF), $G_m = 1$. Finally, in MF, *all* packets belonging to undetected flows *increment* counters. Further, *all* packets *finding* all counters at or above V_{cut} create an entry. In S^3, *only* packets with bytes sampled increase counters and *packets that would find all counters at $d-1$ do not instantiate entries unless sampled.* The *memory gain* and the fact that only sampled bytes increment counters translate into smaller *interference.* Smaller interference (i.e. help from other flows passing the stages) and the fact that flows must be sampled at least once to get detected, translates into better *selectivity* for the same amount of memory, as we will see in section 7.

While we do not show it for space reasons, it can be more rigorously proven that S^3 becomes MF as $p \to 1$. This, together with the generality of theorem 1 allows us to prove the following lemma that gives an alternative bound to that in [2] for the number of captured flows in MF that does not depend on the number of flows.

LEMMA 3. *The expected number of flows passing a parallel Multistage Filter with m stages of $b = \frac{1}{z}V_{cut}^{1/m}$ counters each is bounded above by $\frac{1}{z}\big(1+\frac{1}{(1+m)(V_{cut}-1)}\big)$ regardless of the flow size distribution, number of flows and amount of traffic in a measurement interval.*

PROOF. Since $S^3 \to MF$ when $d \to V_{cut}$ and theorem (1) applies for any value of $d > 1$, condition $b > \frac{1}{z}(d-1)^{1/m}$ yields $b > \frac{1}{z}(V_{cut}-1)^{1/m}$, which is always met if $(bz)^m = V_{cut}$. Thus, $\frac{1}{(bz)^m-1} = \frac{1}{V_{cut}-1}$, which yields the result when substituted in the bound in theorem 1. □

4.8 Accuracy of S^3

There are two sources of inaccuracy in S^3: the uncertainty due to *sampling* and the effect of *interference* resulting from limited memory. With unlimited memory, the number of bytes that go before a flow gets detected, S_d, has a *negative binomial(d,p)* distribution. The best estimate for the size of a large flow, v, is $\hat{v} = E\{S_d\} + R = d/p + R = V_{cut} + R$, with R the number of bytes accounted once the flow is detected. The absolute error is $\hat{v} - v = V_{cut} - S_d$ for a *mean square error* $MSE = E\{(\hat{v}-v)^2\} = Var(S_d)$. Since S_d is the sum of d i.i.d *geometric(p)* r.v., X, we have that $Var(S_d) = d\,\sigma_x^2$. Since $p = \frac{d}{V_{cut}}$, the MSE is given by $(1-\frac{d}{V_{cut}})V_{cut}^2/d$. Defining the *relative error*, ϵ_{rel}, as the square root of the MSE over the size of a flow, the relative error is (for a flow at the threshold and unlimited memory) given by $\epsilon_\infty = \sqrt{\frac{1}{d}-\frac{1}{V_{cut}}}$. Note that ϵ_∞ is, by definition, only due to the effect of sampling. In addition, note how it vanishes when $d \to V_{cut}$ and that it can be approximated by $\frac{1}{\sqrt{d}}$ for large thresholds when $d \ll V_{cut}$.

Limited memory translates into *interference* at the stages, causing the *premature* detection of flows. Consequently, estimating the number of *missed* bytes with V_{cut}, the worst case is for a flow arriving last since it shall be overestimated at the most. Let $c_1..c_m$ be the values at the stage counters that the first sample from this flow finds: $d - c_{min}$ samples from it shall be required with $c_{min} = min(c_1..c_m)$ since the flow has to raise *all* its stage counters to $(d-1)$ and get sampled once again before entering the flow memory. This implies a number of missed bytes $S_{d-c_{min}}$ with a *negative binomial$(d-c_{min}, p)$* distribution. The MSE is therefore given by $E\{(V_{cut}-S_{d-c_{min}})^2\} = V_{cut}^2 + E\{S_{d-c_{min}}^2\} - 2V_{cut}E\{S_{d-c_{min}}\}$. Now, conditioning for c_{min} and taking expectation we have that $E\{S_{d-c_{min}}\} = (1-\frac{E\{c_{min}\}}{d})V_{cut}$ and $E\{S_{d-c_{min}}^2\} = \frac{V_{cut}^2}{d^2}\big((d-E\{c_{min}\})(1-\frac{d}{V_{cut}}) + E\{(d-c_{min})^2\}\big)$. Substituting in the expression for the MSE, taking the square root and dividing by V_{cut}, the worst-case relative error is found to be

$$\epsilon_{rel} = \sqrt{\Big(1-\frac{E\{c_{min}\}}{d}\Big)\epsilon_\infty^2 + \frac{E\{c_{min}^2\}}{d^2}} \qquad (9)$$

Note that, as $d \to V_{cut}$, ϵ_∞ vanishes and the relative error is given by $\sqrt{E\{c_{min}^2\}}/V_{cut}$: there is no inaccuracy due to sampling since bytes are sampled with probability 1 and $E\{c_{min}^2\}$ is the second moment of the smallest increment caused by other flows that share our flow's counters. In addition, as we enlarge the stages, $E\{c_{min}\}$ and $E\{c_{min}^2\}$ get small and $\epsilon_{rel} \to \epsilon_\infty$.

LEMMA 4. *The worst-case relative error in S^3, ϵ_{rel}, is bounded below by ϵ_∞.*

PROOF. From (9), $\epsilon_{rel} \geq \epsilon_\infty \leftrightarrow (1-\frac{E\{c_{min}\}}{d})\epsilon_\infty^2 + \frac{E\{c_{min}^2\}}{d^2} \geq \epsilon_\infty^2 \leftrightarrow \frac{E\{c_{min}^2\}}{d} \geq \frac{E\{c_{min}\}}{d} - \frac{E\{c_{min}\}}{V_{cut}}$. Since c_{min} is integer-valued, $E\{c_{min}^2\} \geq E\{c_{min}\}$, and the above is always true. □

The observation in lemma 4 has important practical and theoretical implications: *for a certain d, the relative error may decrease with the amount of memory, but this improvement shall, at some point, become negligible since the relative error is bounded below by ϵ_∞. However, ϵ_∞ can be made arbitrarily as close to 0 as we wish*[3]. The following two lemmas shall be used in section 6 to study the scalability of S^3.

LEMMA 5. *It suffices that $\frac{E\{c_{min}^2\}}{d^2} \leq \epsilon_p^2 - \epsilon_\infty^2$ for the relative error in the estimates of large flows to be no larger than ϵ_p, where $\epsilon_p \geq \epsilon_\infty$.*

PROOF. c_{min} is, at most, $d-1$: *samples beyond the $(d-1)^{th}$ either instantiate entries or do not raise the counters.* Thus, $(1 - E\{c_{min}\}/d) \leq 1$. Considering (9), this yields $\epsilon_{rel} \leq \sqrt{\epsilon_\infty^2 + E\{c_{min}^2\}/d^2}$. Thus, for ϵ_{rel} to be below ϵ_p it suffices that $\epsilon_\infty^2 + E\{c_{min}^2\}/d^2 \leq \epsilon_p^2$. Since c_{min}, d and ϵ_∞^2 are non-negative, this is possible only if $\epsilon_p \geq \epsilon_\infty$. □

LEMMA 6. *If $P\{c_{min} \geq \epsilon d\} \leq \delta$ for some $\epsilon, \delta < 1$ then, the relative error for a large flow is bounded above by $\sqrt{\epsilon_\infty^2 + \epsilon^2(1-\delta) + \delta}$*

PROOF. c_{min}^2 is integer-valued, non-negative and restricted to $0, \ldots (d-1)^2$. Thus, it can be proven that $E\{c_{min}^2\} = \sum P\{c_{min}^2 \geq i\}$, $i = 1..(d-1)^2$. Since $P\{c_{min} \geq i\}$ is non-increasing with i, the previous summation can be bounded above by $\int_0^{(d-1)^2} P\{c_{min}^2 \geq x\}$. Since $P\{c_{min} \geq \epsilon\ d\} \leq \delta \leftrightarrow P\{c_{min}^2 \geq \epsilon^2\ d^2\} \leq \delta$, if such condition is met, the previous integral (i.e. $E\{c_{min}^2\}$) can be bounded as $E\{c_{min}^2\} \leq \int_0^{\epsilon^2 d^2} 1 + \int_{\epsilon^2 d^2}^{(d-1)^2} \delta = \epsilon^2 d^2 + ((d-1)^2 - \epsilon^2 d^2)\delta$. This implies $\frac{E\{c_{min}^2\}}{d^2} \leq \epsilon^2(1-\delta) + (1-\frac{1}{d})^2\delta \leq \epsilon^2(1-\delta) + \delta$. Proof follows by virtue of lemma 5. □

5. IMPLEMENTING S^3

Stage counters in MF have to be implemented in memories as fast as line rates, since such counters may have to be updated on each packet arrival. Given its resemblance with MF, stages in S^3 could be implemented using SRAM [2]. In this section we present some *reference design* that shows how S^3 could be implemented using slower memories. This will reveal the flexibility of S^3 adapting to the available memory technology and how S^3 could scale even if the gap between memory and link speeds increased. Two observations motivate this design: first, the fact that, in S^3, not every arriving packet translates into a memory update; second, the fact that, unlike in other per-packet processing tasks (like classification in firewalls or route lookups), *the actions out of accounting do not apply to the packets but to counters, which allows "early" releasing packets before such actions are taken*. In the following, we assume that several *processing engines* (μP for short) are available for the execution of measurement algorithms as well as two types of memory devices, fast but small ones (e.g. SRAM) and larger but slower ones (e.g. DRAM)[4]. While we shall refer to these memories as SRAM or DRAM, what we shall really be studying is the fact that S^3 smoothly relaxes the time requirements of the memories involved, no matter their technology: i.e., with SRAM we shall mean memories able to count at line rates, whereas DRAM shall refer to slower memories.

Reference design: Whenever a packet arrives (e.g. at a line card), S^3 performs a lookup in the *flow memory*, as $S\&H$ and MF. This can involve lookups as often as packets arrive. Thus, the *flow memory* needs to be implemented in high-speed memory. However, unlike in MF, only packets with sampled bytes cause read/write operations at the stages. Let us assume that, on reception of packets, an *input μP*, created some *sample handler*, S_h, and released the packet from the *measurement process*[5]. This would minimise the delay that the measurement might add since creating a S_h involves retrieving a small amount of packet data, which may already be gathered for other purposes. These S_hs could contain the information relevant to our algorithm: the FID (S_{FID}), a timestamp (S_{tstamp}), the packet length (S_{len}) and the number of bytes sampled from the packet (S_{bytes}). Finally, this *input μP* could put this handler aside in a "queue" at some memory device, until another μP read it and took over. Clearly, this queue should also be in fast memory since consecutive packets could be sampled and queueing a S_h should finish before any other S_h "*entered*". Some other processor (*server μP*) could be used to to read these S_hs from the queue, schedule the computation of the hash functions on the S_{FID} and increment the respective stage counters, held in a *larger but slower* memory device. Let us assume that the *write* operations on the counters could be *pipelined*, and that after such operations the *server μP* could know if all counters reached $d-1$ so that entries could be created in the *flow memory* when appropriate. Fig. 4 shows the idea. In what follows, we study the feasibility of such a design using fewer fast memory and the limitations that this may put on the performance of S^3.

5.1 Analysis of our reference design

A packet causes memory operations at the stages if one or more of its bytes get sampled. It can be easily proven that, the case with the worst time constraints is when traffic is composed of packets of minimum size, l_{min}, the link is permanently busy and all the traffic goes to the sampler. Let $\Delta t = \frac{l_{min}}{C_l}$ be the minimum inter-arrival time, with C_l the link capacity. A packet will cause the creation of a S_h with probability $p_s = 1 - (1-p)^{l_{min}}$. Thus, the average

[3] This is true, since, while in the discussion we have assumed integer values for d, nothing prevents us really from using real-valued sample thresholds.

[4] These assumptions are very reasonable in case of *network processors*, where we find a *core* processor performing control functions and a set of RISC processors optimized for packet-processing functions, which typically run several hardware-assisted threads. Further, NPs (for instance Intel's IXP12xx) are typically designed to interface with DRAM as well as SRAM devices.

[5] A similar concept is used in NPs, where *packet handlers*, typically held in SRAM, are created for packets. Such handlers are data structures containing control information and are typically used as pointers for tasks like queueing.

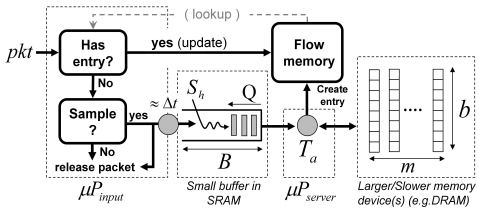


Figure 4: Reference design

worst-case inter-sample time is $\Delta t/p_s$. Let us approximate this "geometric" arrival process by assuming that the inter-sample times are exponentially distributed (the continuous counterpart of a geometric process) with the same mean, that is, by a Poisson process with rate $\lambda = p_s/\Delta t$. With this approximation, we are considering a worse case since, as Poisson, more than one sample could arrive within Δt. In order to queue a S_h in the buffer, the *access time* of the fast device should be no larger than Δt. Let T_a be the time spent by the *server* μP in reading S_hs from the buffer, computing the hash functions and updating the corresponding stage counters. Since comparisons and hash functions can be efficiently implemented in hardware, T_a will be on the order of the *access time* of the large memory device. Let us define $\alpha = \frac{T_a}{\Delta t}, \alpha > 1$ as the *disparity* between the access times of the two types of memories. If we require the fast memory scale with the link capacity, then $\alpha = \frac{T_a}{\Delta t} = \frac{1}{l_{min}} C_l / \frac{1}{T_a}$ which can be understood as the ratio link versus memory bandwidth, for the slower memory.

Since it is reasonable to consider T_a constant, we can model our design as an M/D/1/k system, as shown in fig. 4. With such model, the *load* of the "buffer+server μP" system is given by $\rho = \lambda T_a = \frac{p_s}{\Delta t} T_a = p_s \alpha$. Now, for the system to be stable it must be that $\rho < 1$. Thus, the first constraint is that we cannot oversample at will: $p_s < \frac{1}{\alpha}$. Thus, using memories α times slower may limit p, which shall in turn limit the *selectivity* (for the same z or V_{cut}) or the *granularity* for a fixed *selectivity*. To easily see this, let us approximate $p_s \approx\ p\ l_{min} = \frac{d}{z\ C_l D}\ l_{min}$, where D is the duration of the measurement interval for a total of $C_l D$ bytes. The inequality bove becomes $\frac{d}{z} < \frac{C_l D}{l_{min}\alpha}$. In addition, as per section 4.8 this constraint for d shall limit accuracy since the worst-case relative error is above $\epsilon_\infty \approx 1/\sqrt{d}$.

Now, since $T_a > \Delta t$, some S_hs may find the server μP "busy" updating counters at the slow memory[6] or previously created S_hs in the queue. Since this buffer will in practice have limited space, it could happen that some S_h found it full and had to be discarded. Losing S_hs would reduce the detection probability of flows, since the algorithm would think that flows were sampled fewer times. Let us see how large should this buffer be for this to *rarely* happen.

5.1.1 Dimensioning the buffer in fast memory

In order to dimension the size of the buffer, we can use the relations for the M/G/1 for constant service times. In case of an infinite queue (i.e. M/D/1 instead of M/D/1/k), the average waiting time for a S_h would be $E\{W\} = \frac{\rho T_a}{2(1-\rho)}$ for, by virtue of Little's formulae, an average occupation $E\{Q\} = \lambda E\{W\} = \frac{\rho^2}{2(1-\rho)}$. The variance of the waiting time would be $\sigma_W^2 = \frac{T_a^2 \rho}{(1-\rho)^2}\left(\frac{4-\rho}{12}\right) \leq \frac{T_a^2 \rho}{3(1-\rho)^2}$ and its standard deviation $\sigma_W \approx \frac{T_a \sqrt{\rho}}{(1-\rho)\sqrt{3}}$. Hence, the standard deviation for the number of samples in the queue can be bounded by $\sigma_Q \approx \frac{\sqrt{\rho}}{(1-\rho)\sqrt{3}}$. Now, since the arrival of samples is approximately Poisson, the PASTA property holds. Thus, the probability that some S_h finds exactly k samples in the buffer is $P\{Q = k\}$ and the probability of a S_h finding more than k samples in the queue is $P\{Q \geq k\}$. By virtue of Chebyshev's inequality, $P\{Q > E\{Q\} + n\sigma_Q\} < \frac{1}{n^2},\ \forall n > 0$. Hence, if the buffer is of size $B = E\{Q\} + \frac{\sigma_Q}{\sqrt{\epsilon_o}}$ then, the probability that a S_h gets discarded will be $< \epsilon_o$. For small values of ϵ_o, the behavior of our M/D/1/k may be well estimated using the relations for the M/D/1. Thus, substituting each term by the values found before, we get to $B = \frac{\rho^2}{2(1-\rho)} + \frac{\sqrt{\rho}}{(1-\rho)\sqrt{3\epsilon_o}} = \frac{\sqrt{3\epsilon_o}\rho^2 + 2\sqrt{\rho}}{2\sqrt{3\epsilon_o}(1-\rho)}$. For $\rho < 1$, we have that $\rho^2 < \sqrt{\rho}$ and B can be bounded above as $B \leq \frac{\sqrt{\rho}(\sqrt{3\epsilon_o}+2)}{2\sqrt{3\epsilon_o}(1-\rho)} \approx \frac{\sqrt{\rho}}{(1-\rho)\sqrt{3\epsilon_o}}$, where the latter approximation comes from considering $\epsilon_o \ll 1$. Concluding, if the buffer is, in number of S_hs, as large as

$$B = \frac{\sqrt{\rho}}{(1-\rho)\sqrt{3\epsilon_o}} \tag{10}$$

then, the overflow probability will be smaller than ϵ_o.

5.1.2 Impact of update delay on accuracy

The following impairment due to this buffering should be noted. Whenever a S_h is "queued", it takes some time until stage counters for that flow get updated. If one of these S_h turns out to be the one finding all counters at $d-1$, the creation of an entry for this flow will be delayed the time it took that S_h to get to the counters. If subsequent packets of that flow arrive within this time interval, they will find no entry and will be either sampled or ignored. Thus, *while losing S_hs affects the detection probability, delaying them may affect accuracy.* Again, the worst case is when the link is always busy and all packets arriving after the flow *just detected* belong to it. If it takes, on the average, τ for a S_h to get serviced then, the maximum number of bytes that the algorithm shall *miss* is, on the average, τC_l. The average time it takes for a S_h to get serviced is $\tau = E\{W\} + T_a = \frac{(2-\rho)T_a}{2(1-\rho)}$, with W the *waiting time* in the buffer. Thus, the average number of bytes *missed* from a flow is bounded above by $E\{v_{miss}\} \leq \frac{(2-\rho)T_a}{2(1-\rho)} C_l = \left(1 + \frac{1}{1-\rho}\right)\frac{\alpha\ l_{min}}{2}$. Note that this error tends to infinity as ρ tends to unity. Thus, if either the *disparity* of speeds or the byte-sampling probability is high, this error will be considerable. However, for e.g. $\rho < 0.75$, the error will be bounded above by $\leq 2.5\ \alpha\ l_{min}$, which is $2.5\ \alpha$ minimum-sized packets. For the same load, $B = 2000$ ensures an overflow probability below 10^{-6}. Thus, for moderate loads (achieved reducing p), this error should not be a concern. In case the load were high, our design can be easily *upgraded* so as to estimate this error and bound it above as shown in [15], where we also show some numerical example showing the potential of the solution.

Final remark and section summary: At very high speeds the *flow memory* may have to be implemented with CAMs [2] (currently $1ns$). In our design, both μPs (input and server) access the buffer of S_hs. Memories simultaneously supporting read/write operations do actually exist [13]. For the same amount of memory, S^3 can support

[6] In reality, the case might be that μP support several threads and that the accesses to memory are "queued" at some memory interface. This is the case with Intel's IXP12xx where the so-called *micro-engines* issue memory operations to the memory controllers. Nevertheless, we model the situation as a *server* being busy with a *sample handler* for some time T_a.

more stage counters than MF. Further, stages in S^3 can be implemented in memories slower than link bandwidths. This allows implementing S^3 at speeds where an implementation of MF would be either difficult, costly or not possible. Further, the use of DRAM translates into considerable savings in terms of space and power consumption. Using memories α times slower limits above the sampling probability, thereby theoretically limiting *selectivity* and the worst-case *accuracy*. Both can be compensated in case of DRAM using more and larger stages. Further, even for high (pessimistic) values of α, S^3 can be very selective for values of d (or p) much below the limit due to α. Thus, small buffers may already ensure no overflow and the impact on accuracy due to the delay may be negligible. Finally, schemes combining both memory types have been proposed in related contexts[7].

6. SCALABILITY OF S^3

We shall now see how S^3 scales and compare it to the other approaches mentioned in this paper. The problem at hand gets harder when *finer* granularities are targeted, when *higher accuracy* is required or when the amount of traffic in a measurement interval increases. As a consequence, the error *relative to the smallest size of interest* (rather than the absolute error, or the error relative to the total) is the natural metric of interest. Hence, the focus in this section is on *the amount of memory (and number of accesses) required by algorithms to detect flows above a fraction z of the total traffic C while incurring a relative error no larger than some prescribed ϵ_p.*

For the sake of comparison, we shall assume that stage counters in S^3 are of fixed size. For its practical relevance, we shall distinguish between *entries* and *stage counters* not only because stages could be held in DRAM, but also since, even if implemented in SRAM, *entries* may be significantly larger than *stage counters* (a $10:1$ relationship is suggested in [2]). This suggests the convenience of minimising the size of the *flow memory*. S^3 requires three parameters to be set: the stage size (b), number of stages (m) and the sample threshold (d). This gives three degrees of freedom. For what accuracy concerns, the worst case is for a large flow arriving last as discussed in section 4.8. Let c_{min} be the smallest of the m counters, $c_1..c_m$, that this flow shall find and let us start dimensioning m and b so that

$$P\{c_{min} \geq \epsilon d\} \leq \delta \tag{11}$$

for some $\epsilon < 1, \delta < 1$. Assuming $c_1..c_m$ i.i.d (which is reasonable since hash functions tend to uniformly spread flows at each stage), (11) is equivalent to $P\{c_i \geq \epsilon\, d\}^m \leq \delta$. Now, the average number of samples taken altogether is bounded above by $Cp = \frac{d}{z}$. Since stage counter values are, at most, $d-1$, we have that $E\{c_i\} \leq min(d-1, \frac{d/z}{b})$. Since we shall require $b > 1/z$ and $d \gg 1$, we have that $E\{c_i\} \leq \frac{d/z}{b}$. Thus, from the non-negativity of $c_i, i = 1..m$ and by virtue of Markov's inequality it suffices that $\left(\frac{E\{c_i\}}{\epsilon\ d}\right)^m \leq \delta$ for (11) to hold. Considering the bound for $E\{c_i\}$, the inequality above is satisfied if $\left(\frac{1}{\epsilon\ bz}\right)^m \leq \delta$, meaning that stages with $b \geq \frac{1}{z\epsilon}\left(\frac{1}{\delta}\right)^{1/m}$ counters suffice. Taking the equality, the total number of stage counters for (11) to hold is given by

$$M = bm = \frac{1}{z\epsilon}\left(\frac{1}{\delta}\right)^{1/m} m \tag{12}$$

regardless of the value of $d > 1$. As a function of m, (12) renders a minimum at $m = log(1/\delta)$ for any $\delta > 0, \epsilon > 0$. Taking $m = log(1/\delta)$ stages means supporting stages with $b = \frac{e}{z\epsilon}$ counters, for a total of $M = \frac{e}{z\epsilon} log(1/\delta)$ counters. Now, by virtue of lemma 6, for the worst-case relative error to be no larger than some ϵ_p it suffices that

$$\epsilon_\infty^2 + \epsilon^2(1-\delta) + \delta = \epsilon_p^2 \tag{13}$$

where $\epsilon_\infty^2 = \frac{1}{d} - \frac{1}{V_{cut}} \leq \epsilon_p^2$. This last condition requires $d \geq (\epsilon_p^2 + 1/V_{cut})^{-1}$, or $p \geq (\epsilon_p^2 V_{cut} + 1)^{-1}$, which can always be satisfied. Solving for ϵ in (13), the number of required stage counters is given by

$$M = \frac{e}{z}\sqrt{\frac{1-\delta}{(\epsilon_p^2 - \epsilon_\infty^2) - \delta}}\ log(\frac{1}{\delta}) \tag{14}$$

Observations:

i) (14) decreases in $r_d = (\epsilon_p^2 - \epsilon_\infty^2)$, where $\epsilon_\infty^2 < \epsilon_p^2$. As a function of δ, (14) exhibits a minimum in $(0, r_d)$. For values of ϵ_p of practical significance (say between 10^{-8} and 10^{-2}), $\delta \approx \frac{r_d}{10}$ well approximates the location of this minimum.

ii) For a given ϵ_p^2, the only way to increase r_d is making ϵ_∞^2 small, that is, increasing d or p, for a given V_{cut}. Hence, (14) is smallest when $\epsilon_\infty = 0$, i.e. when $p = 1$. In this case, the minimum is for $\delta \approx \epsilon_p^2/10$, which gives the bound $M \approx (e/z\epsilon_p) log(10/\epsilon_p^2)$ for MF, that does not depend on V_{cut} nor on C. Recall that C increases with either the speed of links or with measurement interval lengths.

iii) For small ϵ_p^2, d may have to be chosen close to V_{cut} for ϵ_∞^2 to be below ϵ_p^2, that is, the sampling probability may have to be high. However, as link speeds increase (right when timing constraints get harder and there are potentially more flows), ϵ_∞ can be made, *for the same granularity z*, significantly smaller than ϵ_p for $d \ll V_{cut}$ or $p \ll 1$. Thus, S^3 *may meet the error bound, with roughly the same number of stage counters that MF, at small sampling probabilities, allowing for an implementation of the stages in DRAM.*

iv) The number of counters to update per sampled packet is m. Choosing m as $log(10/\epsilon_p^2)$ slowly increases with ϵ_p^{-1}. For $\epsilon_p = 1\%$, this is 12 stages and 20 for an error one hundred times smaller. Even in this worst-case analysis this should not preclude a DRAM implementation of the stages, since only sampled packets shall increase counters.

v) Several d guarantee $\epsilon_\infty^2 < \epsilon_p^2$. Increasing d increases the speed requirements but also $r_d \leq \epsilon_p^2$, thereby diminishing (14)). In addition, $\delta \approx r_d/10$ optimizes in size. However, $b = e/(z\epsilon)$ and $m = log(1/\delta)$ subject to (13) allow multiple setups that guarantee the error bound. Thus, in the circumstances that taking $m = log(10/\epsilon_p^2)$ put excessive time requirements for a specific memory technology and value of d, larger δ and smaller ϵ can be chosen. This is supporting fewer but larger stages. Larger stages should not be a concern if DRAM can be used. Note the flexibility of the approach adapting to the specific speed/capacity features of the memories.

vi) With $b \approx e/(z\epsilon_p)$, $m \approx 2log(\sqrt{10}/\epsilon_p)$ and $d > 1/\epsilon_p^2$, condition $b > \frac{1}{z}(d-1)^{1/m}$ in theorem 1 translates approximately into $d < (e/\epsilon_p)^{log(10/\epsilon_p^2)}$, which poses no practical constraint. For instance, for $\epsilon_p = 1\%$, the constraint is $d < 3.45\ 10^{39}$. Plugging the values for b and m, the bound

[7] For instance [14] propose a *counter management algorithm*(CMA) for a general-purpose counting architecture previously proposed in which DRAM is used to store statistics counted at line rates using counters in SRAM. We directly count samples in DRAM instead of using small SRAM to update a large number of counters in DRAM and thus we do not use a CMA.

for theorem 1 yields a value negligibly above $1/z$, *meaning that a flow memory with* $1/z$ *entries suffices.*

vii) As per (14) and **(vi)**, neither the number of stage counters nor the size of the flow memory required depend on the total amount of traffic. As per theorem 1 this amount of memory does not depend on the number of flows, flow size distribution and packet size distribution. That is, S^3 can, theoretically *detect flows with some granularity* z*, with prescribed accuracy, using a memory that does not depend on the total amount of traffic nor on its specific properties.*

In $S\&H$, the relative error for a flow at the threshold is given by $\sqrt{1-p}/O$ (for the best estimator). Considering $V_{cut} \gg O$, it suffices that $O \geq 1/\epsilon_p$ for this error to be no larger than ϵ_p, which gives an upper bound for the average number of captured flows of $1/(z\epsilon_p)$.

In LC and S^2, the error in frequency estimates is bounded above by ϵN, with N the number of items and where $\epsilon < z$ (typically $\epsilon \approx z/10, z/20$). Since the smallest size of interest is zN, for the relative error to be $\leq \epsilon_p$, it must be that $\epsilon \leq \epsilon_p z$. Thus, the bounds in [7] translate into those in table 1, where δ_{fail} corresponds to the *failure probability* in S^2 and N has been substituted, in the bound for LC, by C/l_{min} where l_{min} is the smallest packet size. The reason for this relates to how LC could be implemented and shall become clear in the following section. Note that, in table 1, we distinguish between entries and stage counters (if any) and whether each *can* be "held in DRAM" or not. Note that it is debatable that LC and S^2 outperform MF and $S\&H$. In addition, note that the number of entries required with MF and S^3 is much smaller than with the other approaches. This is clearly an advantage if the flow memory is to be implemented using CAMs. Further, unlike entry sizes, stage sizes do not depend on the *flow definition*. This independence is also an advantage since the suggested 10 : 1 ratio may easily increase in our context with the advent of IPv6, where FIDs may be significantly larger. Finally, S^3 *has the advantage over* MF *that these counters can be smaller and potentially in slower memories.*[8]

Algorithm	Entries (SRAM)	Stage counters
$S\&H$	$\frac{1}{z\epsilon_p}$	$\emptyset$
S^2	$\frac{2}{z\epsilon_p} log(\frac{1}{\delta_{fail} z})$	$\emptyset$
LC	$\frac{1}{z\epsilon_p} log(z\epsilon_p \frac{C}{l_{min}})$	$\emptyset$
MF	$\frac{1}{z}$	$\frac{2e}{z\epsilon_p} log(\frac{\sqrt{10}}{\epsilon_p})$ (SRAM)
S^3	$\frac{1}{z}$	$\approx \frac{2e}{z\epsilon_p} log(\frac{\sqrt{10}}{\epsilon_p})$ (DRAM)

Table 1: Memory requirements

7. EXPERIMENTAL EVALUATION OF S^3

We present now a summary of experimental results obtained with software implementations of the algorithms and real traces from [12]. The first experiments aim at understanding the effect of the different parameters $(d(p), b, m)$ on the performance of S^3. For proximity, we also compare S^3 with $S\&H$ and MF. Next, we present our findings for LC and S^2. Finally, we show results for longer traces that support the discussion on the comparison of the algorithms.

[8] This requires the usage of some pre-buffering, which we do not show in table 1, since, for reasonable values of α, it is minimal compared to the other memories and the benefit of using slower memories outweighs the requirement to keep such a small buffer.

7.1 The effect of d, b, m in S^3

In the experiments that follow, we observed, for S^3, MF and $S\&H$, the *number of false positives* (N_{fp}), the *number of false negatives* (N_{fn}) and the *relative error* for large flows (ϵ_{rel}). Unless otherwise stated, each result corresponds to the average of 40 experiments, where both the sampling process and the hash functions changed[9]. For a meaningful comparison, $S\&H$ and S^3 used the same sampling probability; that is, we set $O = d$. To compare S^3 with MF, both used *the same amount of memory*: we set m stages and b counters per stage for MF and this translated into $\lceil log_2(V_{cut}) \rceil b / \lceil log_2(d) \rceil$ counters for S^3 due to the *memory gain*(G_m), for the same number of stages. We first present some results for one of the traces (CESCA) and then show that our observations hold for other traces.

7.1.1 The effect of the sample threshold (d)

d is the key parameter governing the behavior of S^3. Thus, we studied its effect on the three metrics of interest. Recall that d affects the sampling probability and the memory gain. Fig. 5(left) shows the average number of false posi-

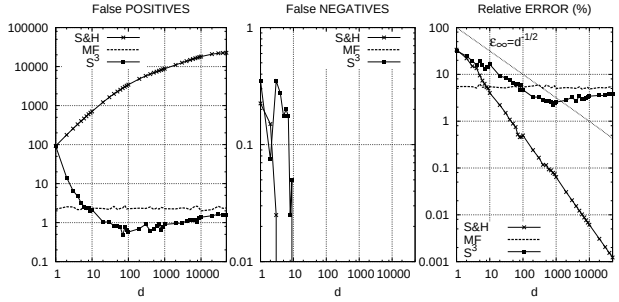


Figure 5: The three metrics of interest for CESCA when $z = 0.01$**,** $m = 2$**,** $b = 400$**.** d **ranged in** $[1.. < V_{cut} \approx 580KB]$**.**

tives with $S\&H$, MF and S^3 for CESCA, a 1 *sec.* fragment with $58MB$, when targeting $z = 1\%(z = 0.01)$. Flows were defined as packets with same (*source,destination*) address pairs. With this definition, there were 22.737 flows, two of which were large for this value of z. MF used 2 stages of 400 counters. As can be observed, the number of false positives in $S\&H$ quickly increases as we increase $O = d$. On the contrary, this makes S^3 more selective until some *saturation point* beyond which the number of false positives, N_{fp}, increases. Clearly, this is the effect of limited memory: *if the sampling probability is high, most of the counters reach their maximum value at some "saturation" point beyond which few or even only one sample are required for flows to get detected.* For $d = 1$, S^3 and $S\&H$ coincide. As $d \to V_{cut}$, all metrics for S^3 approximate those for MF. This is in accordance with $S\&H$ and MF being the two extreme cases of S^3. Further, the N_{fp} with S^3 is always smaller than with $S\&H$, as lemma 2 predicts. As regards the 2 large flows, fig. 5(center) shows the average number of *false negatives*. For MF, this is 0, as expected. Increasing d, the probability of missing a large flow quickly vanishes for $S\&H$, so does for S^3 but at a slower pace: raising d, the byte sampling probability increases but also makes S^3

[9] For the hash functions we used randomly generated mappings between the flow FIDs and the counters, for each of the stages. Further, as in [2], we considered that packets arrived back to back, as if the link were fully occupied.

require more samples prior detection. For $d \geq 9$, S^3 always detected the two flows. Finally, fig. 5(right) shows the *relative error*. Accuracy improves much faster with d in $S\&H$ than in S^3, however at the cost of a N_{fp} several orders of magnitude higher. For $d < 500$ (i.e. before *"saturation"*), the slope of ϵ_{rel} is, in S^3, half of that in $S\&H$. This is consistent with the relative error decreasing as $1/\sqrt{d}$ in the former and as $1/O$ in the latter.

7.1.2 The effect of stage sizes (b)

Assigning counters to flows by means of hash functions applied on packet FIDs partitions the traffic into b subsets (one per counter) where each set has, on the average, N_{flows}/b flows that interfere, with N_{flows} the total number of flows. Hence, enlarging stages should diminish the *interference*. Fig. 6(left) shows the N_{fp} in CESCA with S^3/MF, for different stage sizes, when, to better see the effect, algorithms targeted a granularity of 0.1%. In that case, the number of large flows was 133 ($\approx 0.58\%$ of N_{flows}). For the sake of comparison, the results for S^3 when $b = 100, 1000, 10000$ are shown darker and we show the results for MF (flat lines) for these values of b. On the right of each plot, we show the results for MF *when averaging throughout all experiments*. As can be observed, S^3 outperforms MF for small stages, relative to $1/z$. For b large enough MF performs very well, but even in those cases, S^3 is more selective for some choices of d due to the G_m and to the effect of sampling: interference is reduced and only sampled packets can instantiate entries. However, the larger the stages, the larger that d must be set for S^3 to perform better than MF: *the more stage counters are supported, the higher the sampling probability can be set before saturating the stages.* A similar behavior can be observed in fig. 6(right) for the

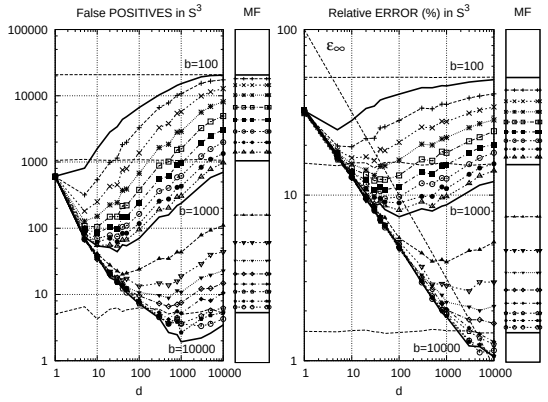


Figure 6: N_{fp} **(left) in** S^3/MF **and** ϵ_{rel}**(right) with** $m = 2$ **for** $b = 100, 200, 300 \ldots 1000, 2000, \ldots 10000$**.**

relative error, ϵ_{rel} : larger stages means finer partitions (i.e. more sets, each with fewer flows) and less *pollution* at the counters where large flows hash to. This translates into higher accuracy since $S_{d-c_{min}} \rightarrow S_d$. The dashed slanted line corresponds to the worst-case relative error with unlimited memory ($\epsilon_\infty \approx 1/\sqrt{d}$). Note how, for sufficiently large stages, ϵ_{rel} differs from ϵ_∞ only in a factor: this is due to the fact that, in ϵ_∞, the MSE is divided by V_{cut}, whereas large flows in the experiment were above V_{cut}. Further, for large stages (large b), the relative error does not significantly improve with b and only decreases if we increase d. Both observations are in accordance with our analysis in section 4.8: *enlarging stages reduces the interference and the relative error but, at some point, accuracy does not improve anymore since the latter is dominated by the effect of sampling.*

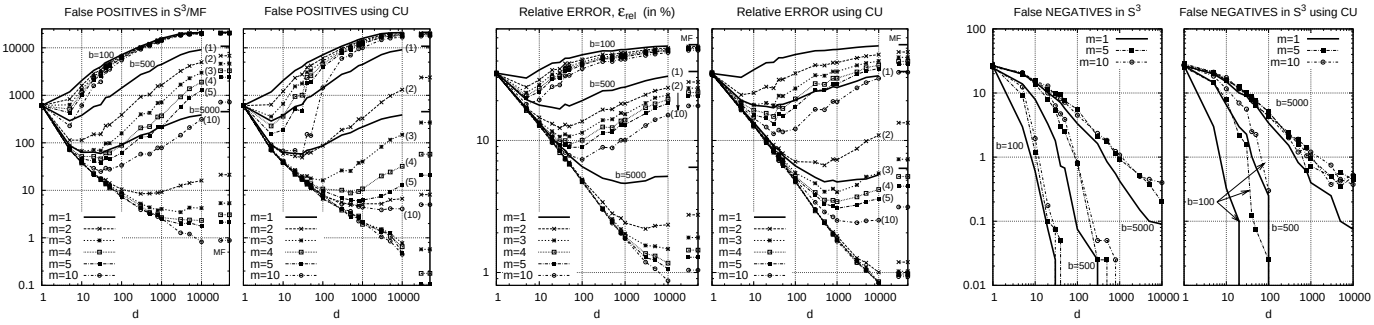
7.1.3 Boosting selectivity: the effect of m and CU

When several stages and distinct hash functions are supported, more partitions of the traffic are "*produced*". This, together with the requirement that *all* counters exceed $d-1$ for flows to get detected is the basis for the exponential improvement in selectivity in S^3/MF in the number of stages, resulting from an exponential reduction in interference. Fig. 7(a) shows the effect of adding stages to MF/S^3 in the N_{fp}, when $z = 0.1\%$ for different stage sizes ($b = 100, 500$ and 5000). The flat lines on the right correspond to the N_{fp} for MF. For each stage size, the curve when $m = 1$ is shown darker. For small stages (e.g. $b = 100$), most of the counters get "saturated" since the number of samples hashing to each is large compared to d. Thus, the N_{fp} increases for small ds and adding stages does not improve the selectivity of MF/S^3. However, note the improvement when adding the second stage for larger stages. Again, S^3 outperforms MF in *selectivity* for a wide range of values of d. Further, with $m = 10$ stages with $b = 500$ counters each, the selectivity is better than with $b = 5000$ and $m = 1$, for the same amount of memory. This is consistent with our analysis in section 4.5 in that, *once stages are sufficiently large, adding stages is more effective than enlarging the existing ones.*

Fig. 7(b) plots ϵ_{rel} in MF/S^3 for the same setup. Again, supporting many stages is not effective if these are small, relative to $1/z$. However, once b is close to or exceeds $1/z = 1000$, adding stages makes the curve of ϵ_{rel} decay as $1/\sqrt{d}$. Fig. 7(c) shows the N_{fn} (among the 133 large flows) for different values of d, number of stages and stage sizes. *False negatives* in S^3 should not surprise and are the consequence of choosing V_{cut} as the threshold[10] T: $P_{det}(V_{cut}) < 1$ for $d < V_{cut}$. As we can see, the smaller the stages the fewer the N_{fn}. The same happens with the number of stages. As many as m counters have to reach $d-1$ for a sample from a flow to create an entry. The more counters have to reach $d-1$, the more likely it is that the number of samples required *from that flow* be d. In other words, with large m and b, the "help" given to small flows to get captured is reduced, *so is the help for large flows.* However, for sufficiently large ds, the probability of not detecting flows slightly above V_{cut} is very small: it is bounded above by $1 - P^{\infty}_{det}(v)$ (since this is the case when flows get no help from the other flows) and this is very small as of our analysis, even for small ds, relative to V_{cut}. Further, as $d \rightarrow V_{cut}$, $N_{fn} \rightarrow 0$ since $S^3 \rightarrow MF$.

S^3 also benefits from *conservative update* (CU). The plots on the right of each subfigure in fig. 7 show the three metrics for the same setup when CU is used in both MF/S^3. Note how none of the metrics decrease with CU when $m = 1$, as expected. However, note the great reduction in the N_{fp} and ϵ_{rel} in the two algorithms with more than one stage. The net effect of CU is as *if memory were larger, since it diminishes the rate of growth at the counters (without compromising the*

[10] Note the following interesting dilemma: had we chosen $V_{cut} < T$, flows of size $\approx T$ would have been detected with higher probability; however, flows of size $\approx V_{cut} < T$ would have also been detected. Thus, choosing $V_{cut} < T$ to ensure the detection of flows of size T may be at the cost of some *uncertainty* in knowing whether detected flows are *large* or *small*.



(a) False positives (b) Relative error (c) False negatives

Figure 7: The three metrics of interest for CESCA when $z = 0.1\%$, for different stage sizes (b), number of stages (m) and sample thresholds (d). The plots on the right of each subfigure correspond to the results using *conservative update.*

detection of large flows) and, consequently, the undesired effects of interference. Regarding the N_{fn}, these slightly increased using CU. Note how these overlap for $b = 500, 5000$ when $m > 1$: once interference is removed, the detection of large flows depends only on d. In the experiments that follow, CU was always used.

7.1.4 General behavior of S^3

Experiments with other traces revealed that S^3 obeys some *general behavior*. To see this, we show the results for several traces of the same length ($70MB$) summarized in table 2. To compare the results for several z, these are shown as a function of $p \in [\frac{1}{zC}, .. \approx 1)$. To compare the results for different traces, we plot the N_{fn} *relative to the number of large flows for each* z, N^z_{lar}, and the N_{fp} *relative to the number of small flows*, $N_{flows} - N^z_{lar}$.

Fig. 8 shows the N_{fp} as a function of p for six of the traces when two granularities are targeted ($z = 1\%$ and 0.1%) for two stage sizes ($b = 400, 2000$ and $m = 2$). For $p = \frac{1}{V_{cut}}$ (marked with the symbols ◯ in the plot for Mydoom32 that indicate the begining of the curves for each z), S^3 and $S\&H$ coincide. Thus, the ordinates of ◯ do not depend on the amount of memory supported. When $p \to 1$ then, $S^3 \to MF$ and the ordinate of □ depends only on the amount of memory supported, the properties of the traffic and the threshold. The same observations hold for the relative error and number of false negatives, shown in fig. 9 for the two target z when $b = 400$. Note the resemblance of the results. The reason why the N_{fn} differ between distinct traces has to do with the size or share of large flows. For instance, Abilene exhibited a 1% false negative ratio even when $p = 0.1$. This is because, for $z = 1\%$, there is a large flow which exceeds z in a 0.3% only, as can be seen in table 2, that shows the *minimum, average and maximum share of large flows* (in %) at each target z, S^z_{lar}, for each trace.

The performance of S^3 has two components: the effect of *sampling* and the *interference between flows*. The former is set by d, which is limited above by $V_{cut} = zC$. The latter decreases in the size and number of stages. Our results and analysis suggest the existence of an optimum value for p (or d) as regards selectivity: some p_{opt} beyond which stages get *saturated* and selectivity no longer improves with d. As b and m increase, p_{opt} increases. With several stages larger than $\approx \frac{1}{z}$, the effect of interference vanishes and all metrics

TRACE	N_{flows}	$N^{1\%}_{lar}$	$N^{0.1\%}_{lar}$	$S^{1\%}_{lar}$(min/avg/max)	$S^{0.1\%}_{lar}$(min/avg)
BUF	2,662	15	110	1.021/4.30/19.7	0.1005/0.80
Abilene	5,766	13	147	1.003/3.28/22.9	0.1009/0.54
Sandiego	6,621	8	123	1.084/6.55/20.2	0.1001/0.48
Cdred	11,194	16	158	1.050/2.21/6.83	0.1001/0.48
Mydoom32	11,851	12	165	1.039/2.46/6.95	0.1001/0.41
Slammer03	23,798	18	50	1.016/3.87/9.51	0.1172/1.57
NCAR	65,725	17	74	1.057/3.67/19.5	0.1020/1.13

Table 2: Trace specifics.

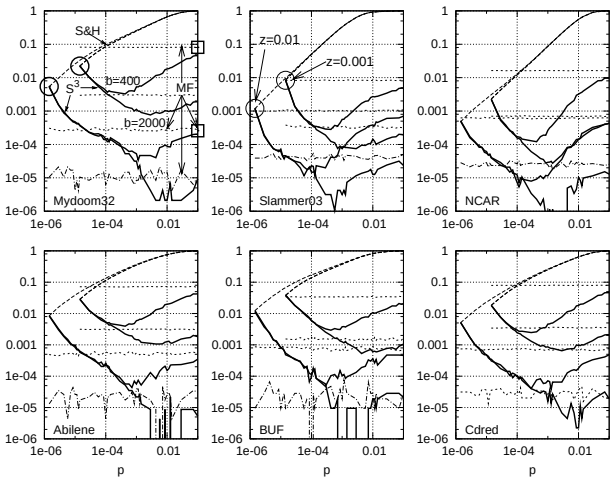


Figure 8: N_{fp} relative to the number of small flows, for several traces, granularities and stage sizes.

decrease in d, as can be seen for CESCA in fig. 7. Hence, for the same granularity z, the performance of S^3 improves as the amount of traffic increases since larger values of d can be attained for the same sampling probability. To see this, we shall show the results for AIX, a trace roughly ten times larger ($785MB$ and 168.539 flows).

Remarks: i) while we do not show the location of p_{opt}, the performance of S^3 is very good for a *wide* range of values of p. Thus, S^3 may perform well, even if badly *tuned*. ii) With minimal memory, S^3 can be very selective if p is chosen considerably below 1. By doing so, the worst-case relative error may not be below $1/\sqrt{pV_{cut}}$. Thus, for small thresh-

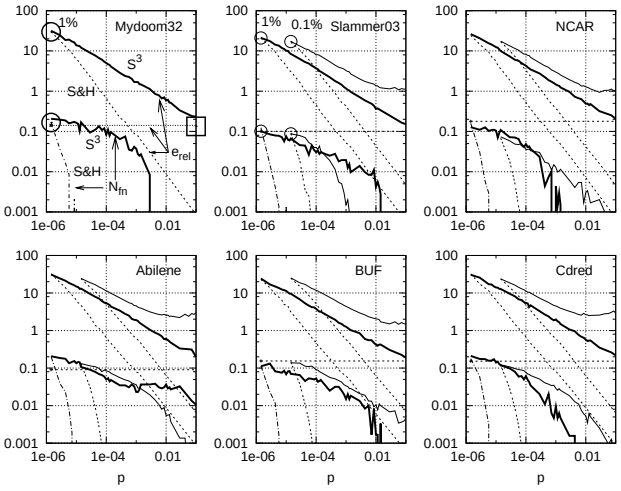


Figure 9: Relative error and false negatives.

olds, while very selective, S^3 may not be very accurate. This can be compensated *preserving entries* accross measurement intervals. iii) Most importantly, as the amount of traffic C increases, the limitation in accuracy disapears for the same z. iv) Using memories α times slower limits above the sampling probability to $\approx 1/(l_{min}\alpha)$, which may be smaller than p_{opt} and prevent from reaching the optimum. However, this may allow supporting more counters if DRAM were used and perform better. To see this, fig. 10 shows the N_{fp} in AIX ($785MB$, 168.539 flows) for different z. For $z = 0.1\%$, the dashed line corresponds to the results using 2 stages with 1000 counters. p_{opt} is, in this case, $\approx 10^{-4}$. If we used memories $\alpha = 19$ times slower and some pre-buffering, p should be no larger than 10^{-3} (assuming $l_{min} = 40$, for a worst-case load $\rho = 75\%$). The darker curves correspond to the results using 4 stages with 2000 counters (i.e. four times more memory), as if we had used larger but slower memories. Note how, for the same sampling probability $\approx 10^{-3}$, using DRAM would translate into a number of false positives almost two orders of magnitude below. Had we set $p \approx p_{opt}$ for SRAM, the number of false positives would be smaller but the relative error would be around 5%, compared to $\approx 1.5\%$ that we could achieve using DRAM. Finally, note that plots start at $p = 1/zC$, hence, for the same granularity, as C increases, the curves get shifted to the left and smaller relative errors are attained at the same sampling probability. This can be seen comparing the results for AIX with those for the $70MB$ traces.

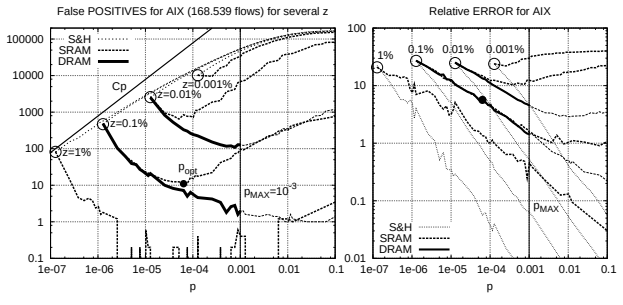


Figure 10: Memory speed/space tradeoffs: False positives and relative error for AIX, for serveral z.

7.2 Comparison with S^2 and LC

The number of packets alone gives little information on flow sizes: assuming typical IP packet sizes, flow sizes can vary in a factor of 37 *depending on whether packets are of minimum or maximum length.* Thus, we implemented S^2 and LC *naturally* identifying bytes as items. However, this led to S^2 being unable to prune flows. A byte-oriented implementation of LC is also not suited: *LC splits the arriving stream into buckets of $\lceil 1/\epsilon \rceil$ items. In terms of bytes, this can be very few packets (or even less than one).* A more intelligent implementation of the algorithms in this context is to let them *virtually split* packets into *minimum-sized* ones (*minps*) and let frequency estimates, f, relate to the number of *minps*: a correction factor of l_{min} on f provides the size of flows. To cope with the fact that packet sizes are not multiple of l_{min}, we let the variables involved take real numbers. We verified this produced the same results as if packet sizes were multiples of l_{min}, without penalizing accuracy nor pruning effectiveness. This is how we implemented LC/S^2 and the reason why the size of the stream N is written as C/l_{min} in the bound for LC in section 6.

For the traces in table 2, LC and S^2 provided relative errors below those with MF and S^3, being LC the most accurate with relative errors between 0.01% and 1% and S^2 with errors between 0.1% and 1%. Figure 11 shows the evolution in the number of entries used by LC (left) and S^2 (right), for the $70MB$ traces when $z = 0.1\%$. The flat line is at $1/z$, the maximum number of large flows that can be present. Note how LC outperforms S^2 in pruning effectiveness; however, at the cost of traversing the flow memory much more often. Note that S^2 is unable to prune small flows and requires a large number of entries. The reason for this is the following: *true, the case potentially leading to the instantiation of the largest number of entries is when the stream is composed of unique items as suggested in* [7]; *however, it is also easier to prune such entries.* For instance, fig. 11(bottom-right) shows that S^2 is able to prune many entries for NCAR (with 65.725 flows); however, it cannot prune those for SLAMMER03 (with 23.798 flows) in spite of it having much fewer flows. In this connection, the narrow figures besides each plot in fig. 11 show the number of entries used in both algorithms *relative to the number of flows in each trace.* As can be observed, the number of entries used in both algorithms amounts to a large fraction of the number of flows. This is specially true for S^2, which requires as many entries as 80% of the number of flows in CESCA. Thus, while S^2 and LC are very accurate, these require a number of entries that is large compared to the number of flows, and that heavily depends on the traffic. Note, comparing figs. 8 and 11, the potential of S^3: for $z = 0.1\%$, with only 2 stages with 2000 counters, less than 0.1% of the small flows instantiated entries in all traces and less than 0.01% in case of NCAR and SLAMMER.

Finally, we verified that, if configured as shown in sec. 6, S^3/MF kept the number of entries below $1/z$ while meeting the demanded accuracy, for the targeted granularities. In fact, much fewer memory than the theoretical sufficed: *for small z, with half of the stages and stages* 1000 *times smaller, S^3 met the demanded accuracy.* For space reasons, we do not show these results, which are in [15]. Instead, to summarize some of our findings, we show the performance of S^3/MF as we *stress the algorithms demanding finer granularities*, for one of the $785MB$ traces. Fig. 12 shows the

number of entries used and the relative error in all algorithms, for different targeted z, where MF/S^3 used *constant amount of memory*. Specifically, we set $m = 3$ and $b = 1000$ for MF and, in S^3, we limited $p = 10^{-3}$ (as if slower memories were used) and compensated this letting it use four times more memory. Note that MF and S^3 outperform the other approaches in terms of space requirements. Further, the combined effect of sampling together with the fact that larger memories could be used, allows S^3 to effectively detect only large flows. This is shown in fig.12, where we also plot the number of entries used *relative to the number of large flows in each case*. As can be seen, the number of entries used by S^3 is insignificantly above the number of large flows. Further, the average number of false negatives in S^3 was always below 1.3 for $z > 0.04\%$ being 5 for $z = 0.01\%$, where the number of large flows was 376.

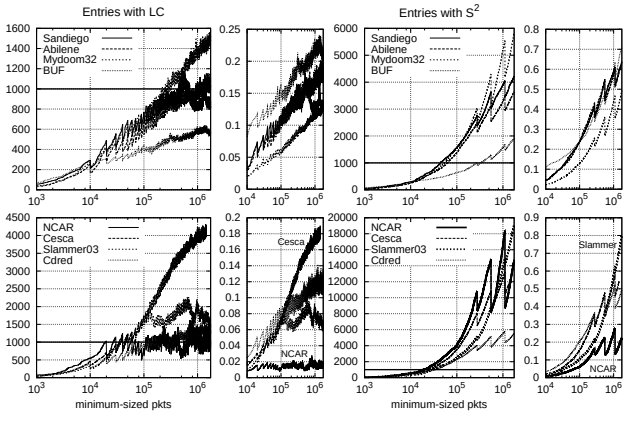


Figure 11: Entries used in LC(left) and S^2(right).

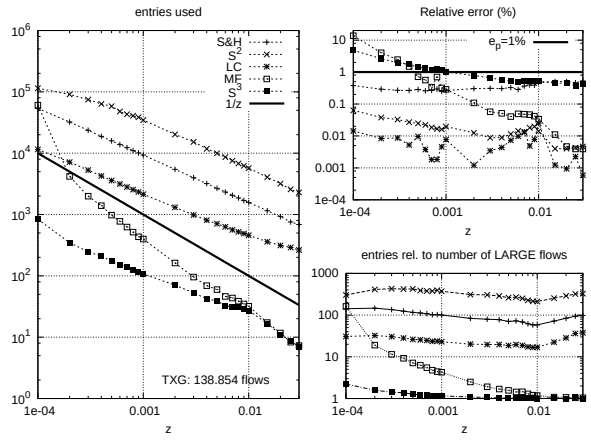


Figure 12: Results for TXG (785MB, 138,854 flows)

8. SUMMARY AND CONCLUSION

We have presented an algorithm, S^3, for the measurement of large flows in high-speed networks that turns out to be a generalization of two existing algorithms, $S\&H$ and MF. By *adding memory* to sampling, S^3 outperforms $S\&H$ in terms of *selectivity*. *Because of sampling*: ***i)*** *S^3 does not ensure the detection of flows at the threshold, T; however, the probability that it detects flows slightly above T is very high;* ***ii)*** *it makes a more efficient memory usage than MF does, thereby supporting more counters for the same amount of memory;* ***iii)*** *interference at the "shared" counters is reduced;* ***iv)*** *despite the interference, flows must "still" be sampled to get detected.* This makes S^3 outperform MF in terms of selectivity. We have also compared S^3 to S^2 and to LC, both regarded to outperform $S\&H$ and MF. Effectively, LC and S^2 outperform $S\&H$, MF and S^3 in terms of accuracy. However, S^2 is insufficient at pruning small flows and tends to capture the majority of them, which translates into high memory requirements. LC is very accurate and keeps the number of entries bounded. However: first, the number of entries used in LC shows considerable dependency on the traffic, which requires overdimensioning its flow memory; second, pruning is at the cost of traversing the flow memory very often, which yields high and *unpredictable* per-packet processing times. *MF/S^3 require a minimal flow memory*, as compared to $S\&H$, S^2 and LC. This is a considerable advantage since entries are larger than counters and may have to be implemented in CAMs at high speeds. Finally, S^3 is, compared to MF, extremely flexible in adapting to the memory technology available and exhibits mild tradeoffs between *selectivity*, *accuracy* and *the speed of memories*.

9. REFERENCES

[1] G. Cormode and S. Muthukrishnan. What's hot and what's not:tracking most frequent items dynamically. In *PODS'03*.

[2] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proc. of ACM SIGCOMM Internet Measurement Workshop*, 2001.

[3] C. Estan and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proc. of ACM SIGCOMM*, 2003.

[4] W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *IEEE Global Internet Symposium'99*.

[5] A. Feldmann, A. G. Greenberg, C. L. and Nick Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: methodology and experience. In *Proc. of ACM SIGCOMM*, 2002.

[6] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMETRICS'98*.

[7] G.Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. of VLDB'02*.

[8] F. Hao, M. S. Kodialam, and T. V. Lakshman. ACCEL-RATE: a faster mechanism for memory efficient per-flow traffic estimation. In *SIGMETRICS'04*.

[9] A. Kumar, M. Sung, J. J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *SIGMETRICS/Performance'04*.

[10] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li. Space-code Bloom filter for efficient per-flow traffic measurement. In *Proc. IEEE INFOCOM*, 2004.

[11] R. Mahajan, S. Bellovin, S. Floyd, J. Vern, and P. Scott. Controlling high bandwidth aggregates in the network. Technical Report, 2001.

[12] NLANR. http://pma.nlanr.net/traces/.

[13] QDR-Consortium. http://www.qdrsram.com.

[14] S. Ramabhadran and G. Varghese. Efficient implementation of a statistics counter architecture. In *SIGMETRICS'03*.

[15] F. Raspall, S.Sallent, and J. Yufera. Shared-State Sampling. http://broadband.upc.es/pub/raspall/s3.pdf.

[16] R. Sommer and A. Feldmann. NetFlow: Information loss or win? In *ACM SIGCOMM Internet Meas. Workshop'02*.

[17] H. Toivonen. Sampling Large Databases for Association Rules. In *Proc. of VLDB'96*.