High-Speed Per-Flow Traffic Measurement with Probabilistic Multiplicity Counting

Peter Lieven Björn Scheuermann
Heinrich Heine University, Düsseldorf, Germany
{lieven, scheuermann}@cs.uni-duesseldorf.de

***Abstract*—On today's high-speed backbone network links, measuring per-flow traffic information has become very challenging. Maintaining exact per-flow packet counters on OC-192 or OC-768 links is not practically feasible due to computational and cost constrains. Packet sampling as implemented in today's routers results in large approximation errors. Here, we present Probabilistic Multiplicity Counting (PMC), a novel data structure that is capable of accounting traffic per flow probabilistically. The PMC algorithm is very simple and highly parallelizable, and therefore allows for efficient implementations in software and hardware. At the same time, it provides very accurate traffic statistics. We evaluate PMC with both artificial and real-world traffic data, demonstrating that it outperforms other approaches.**

I. Introduction

Measuring the per-flow traffic on high-speed network links is a challenging problem. In the case of a 40 Gbps link, one can end up with as few as 12 ns for processing each packet. In such an environment, the available processing power and memory bandwidth do simply not suffice to maintain, for instance, exact statistics of the traffic per source, per application, per protocol etc.: the overall latency for looking up the memory position of the correct counter, reading its current value, incrementing it, and writing the result back to memory is too large (at least without very costly hardware) [1], [2], [3]. In fact, the problem is getting worse and worse, as network bandwidth increases even faster than processing power.[1] Solutions implemented in today's network equipment like (Sampled) Netflow [4], [5] or sFlow [6] examine only a small fraction of the packets, at the cost of significant estimation errors [7], [8], [9], [10].

In this paper, we tackle this problem with a probabilistic counting-based approach to network traffic measurement. We introduce a stream processing algorithm—Probabilistic Multiplicity Counting (PMC)—which uses probabilistic counting techniques to determine the approximate multiplicity of each element in large streams. It is particularly well suited for traffic measurements on high-speed communication links, and we will focus on this application here. However, PMC is likewise applicable for many other purposes.

PMC's main benefit is that it is able to record information on a passing-by packet (or, more generally, on a data item in a stream) by setting only one single bit in a bit field. A specific hashing mechanism determines the bit position to be enabled, based on the packet headers. This operation can be performed in constant time, without any loops or conditional branches in the code, and with write-only memory access. PMC is therefore particularly well-suited for pipelined and parallelized operation. We can then extract flow size estimates from the resulting bit field.

In the course of the paper we first review related work in the following section. In Section III, we detail the problem setting and introduce some important foundations. We then introduce the PMC algorithm, discuss design tradeoffs and interrelations in Section IV. Experimental results from applications of PMC to artificial data sets and real network traces are presented in Section V. Finally we conclude and summarize the paper in Section VI.

II. Related Work

Due to its high practical relevance, the topic of flow measurement has attracted a lot of attention in recent years. In the following, we provide an overview of the works that are most closely related to our own contribution.

Both deterministic and probabilistic approaches have been proposed. Particularly remarkable on the deterministic side are hybrid architectures which use both fast, but expensive SRAM and slower, but also cheaper DRAM storage in parallel. This has first been proposed by Shah et. al. [11], and has subsequently been further improved by Ramabhadran and Varghese [12] and by Zhao et al. [13]. Although all algorithms reduce the amount of expensive SRAM storage needed, they still include deep (e. g., 64 bit) off-chip DRAM counter operations and costly SRAM-to-DRAM updates. An important step if per-flow counters are used is to locate the correct counter(s) for the currently examined packet in memory. This problem is either neglected, or it is proposed to use (expensive) content-addressable memory (CAM) or hash tables [14], both of which limit cost effectiveness and/or scalability.

Due to these fundamental constraints, the larger fraction of existing work falls, like PMC, into the area of approximate or probabilistic counting. These algorithms can be divided into three groups: (i) sampling algorithms that reduce the complexity by sampling only a subset of all packets, (ii) algorithms that focus only on flows which represent a significant part of the overall traffic (so called *heavy hitters* or *elephants*), and (iii) algorithms that process all packets. We detail them in the following subsections.

[1] This is known as "Gilder's law", stating that available bandwidth in communication networks grows at least three times faster than processing power (which, in turn, increases exponentially according to Moore's law).

A. Packet sampling

Packet sampling is what we already find in today's routers, in Cisco's Netflow [4], IPFIX [3], or in sFlow [6], which is currently implemented by most other vendors like Juniper, Brocade, HP or Force10. In most cases, packets are sampled at a fixed frequency (typically 1 out of 512 packets or less). Cisco's Sampled Netflow [5] introduced the variant of picking packets at random; pseudorandom packet selection has been proposed for multi-point observations [15]. However, sampling as it is implemented today provides only limited accuracy, especially for small flows [7], [8], [9], [10]. To address this problem, more sophisticated techniques like Sketch-Guided Sampling [16] or ANLS [17] maintain per-flow counters to keep track of the flow size or the number of sampled packets and throttle the sample rate with increasing flow size. The larger a flow grows, the less packets are sampled. The reduced error comes at the cost of additional storage for per-flow state and higher computational complexity. With PMC, we propose an algorithm that has a very simple structure and is fast enough to examine all packets. We will demonstrate that it achieves higher accuracy than packet sampling.

B. Heavy hitter algorithms

Estan and Varghese's "Sample-and-Hold" approach [10] was the first that exploited the heavy-tailed nature of Internet traffic [18]. The idea is that most of the bandwidth is occupied by only a small fraction of the flows[2]. So, these algorithms maintain only a small set of (deterministic) counters, which count only the traffic of the heavy hitters. Heavy hitter algorithms are able to provide high accuracy for, e. g., all flows that occupy at least 0.1% of the total bandwidth. Enhancements with further guarantees or lower storage requirements have also been proposed, see, e. g., [19], [20], [21]. In contrast to heavy hitter algorithms, our design is able to provide flow size information for *all* flows, not only for the extrema.

C. Approximations for all flows

In recent years, increasing focus has been put on probabilistic algorithms that are fast enough to examine all packets, and at the same time provide estimates of the sizes of all flows. A number of approaches from this category use standard binary counters in a memory element, and focus on how to arrange and update these counters. A well-known example are Spectral Bloom Filters [22]. They are based on the well-known Bloom filter data structure [23], [24], but use a counter at each position in the filter instead of a single bit. Count-Min Sketches [25] hold counters in a matrix-like organization. A big caveat for both Spectral Bloom Filters and Count-Min Sketches is that the maximum multiplicity has to be known a priori quite accurately, to provide large enough counters without wasting too much memory. A recent invention called Counter Braids [14] adds additional layers which are accessed as soon as counters overflow to overcome this problem. All these algorithms need read and write memory access to multiple counters for recording information on a packet, which is comparatively expensive. PMC does not need to know the maximum frequency beforehand, and its counting operation is much simpler.

Probably the closest relative to PMC is the Multi-Resolution Space-Code Bloom Filter (MRSCBF) introduced by Kumar et al. [2]. MRSCBF, too, is based on a write-only bit field for collecting information on the packet stream. While the general direction of MRSCBF is very promising, Donghua et al. [26] found that it is difficult to apply in practice. With parameter choices as suggested in [2], MRSCBF sets an average of 5 bits per captured packet, the worst case is 49 bits. The exact number is determined by a quite complex process. Donghua et al. did an implementation on the Intel IXP 2400 network processor and observed that this complexity (including a lot of conditional branches), along with the large number of memory accesses, makes MRSCBF inefficient. The structure of PMC, in contrast, is extremely simple: it performs the very same sequence of operations for each packet, with no conditional operations, and sets exactly one bit. Despite this simplicity, we will show that PMC achieves higher precision than MRSCBF under similar working conditions.

There is also a wide range of probabilistic algorithms that deal with the estimation of flow size distributions, examples are [27], [28], [29]. Here, we are interested in the size of each individual flow.

III. Preliminaries

Before we dive into the details of our algorithm, it is helpful to take a closer look at the problem setting. Furthermore, in order to understand PMC, a basic understanding of the FM sketch data structure introduced in [30] is vital. We will cover both topics in this section, before we subsequently come to our own contributions.

A. Problem statement

We consider a router at which packets arrive, typically at a very high rate. We assume that each of these packets is attributed to exactly one *flow*. Our aim will be to (approximately) keep track of the number of passing-by packets per flow. In our context, the notion of a flow is very wide and can be adjusted (almost) arbitrarily, which results in high flexibility. A sensible definition could be that two packets belong to the same flow if their source and destination IPs are identical; other possibilities include summarizing all packets that stem from the same source in one flow, or to take the used protocol into account. We assume that a function F is given, which maps packets to flow IDs. We furthermore assume that determining the flow of a packet (i. e., evaluating F) is not a performance bottleneck. This is not a significant restriction: typically, determining the flow ID of a packet is barely more effort than extracting and concatenating one or two header fields (like source and destination IP). This is a trivial task in both software- and hardware-based designs.

After some time interval during which traffic measurements have been performed, we want to be able to ask questions

[2]This is known as a quasi-Zipfian or Pareto distribution.

like: "How many packets from flow x have passed through this router?" For this purpose, we may simplify the problem by considering a sequence of flow IDs instead of a sequence of packets. The algorithm will be given such a stream of flow IDs as its input.

From a more mathematical perspective, the stream of flow IDs during some time interval can also be interpreted as a multiset. Then, the number of packets for a given flow is equal to the *multiplicity* of the flow ID in the multiset. Alternatively, PMC can thus also be interpreted as a means to determine the multiplicities of individual elements in a multiset.

B. FM sketches

PMC builds upon a probabilistic counting algorithm which originates from the database context. It has been devised by Flajolet and Martin in 1985 [30] and is known under the name "FM sketches". FM sketches solve a problem that is quite different from the one tackled by PMC: they provide an estimate of the *cardinality* of a multiset, i. e., the number of distinct elements, whereas PMC's purpose is to estimate the multiplicities of the elements in the multiset, i. e., how often each single element occurred. Thus, FM sketches are not a solution to the problem posed here. Nevertheless, PMC makes use of the algorithmic ideas behind FM sketches, and a basic understanding of FM sketches—and especially of the way how estimates are obtained from the FM sketch data structure—is essential for understanding PMC.

An FM sketch uses a bit field $S = s_1, \ldots, s_w$ of length $w \geq 1$ as a basis. The bit field is initialized to zero at all positions. To add an element x to the sketch, it is hashed by a hash function h with geometrically distributed positive integer output, where $P(h(x) = i) = 2^{-i}$. The entry $s_{h(x)}$ is then set to one. With probability 2^{-w} we have $h(x) > w$; in this case, no operation is performed. For practical implementations, especially in the context of PMC, it is sometimes easier to avoid this special case by restricting the range of possible values of h to $[1, w]$ and setting $P(h(x) = w) = 2^{-w+1}$. Upon careful implementation, this avoids conditional branches. For reasonably large w, the differences are negligible. Note that adding an element to an FM sketch may only result in a change of the sketch if an element is encountered that has not been observed before—otherwise the respective bit position will already be one. It therefore effectively ignores duplicate additions.

The key result of [30] is that an estimate for the number n of distinct added elements can be obtained from the length of the initial, uninterrupted sequence of ones, i. e., from

$$Z(S) := \min(\{i \in \mathbb{N}_0 \mid i < w \wedge s_{i+1} = 0\} \cup \{w\}). \quad (1)$$

Furthermore, Flajolet and Martin find that $Z(S)$ grows like $\log_2 n$, i. e., there is a proportionality factor φ such that

$$n \approx 2^{Z(S)}/\varphi. \quad (2)$$

It is possible to obtain the value of φ as follows. The probability that bit i is not set after n distinct elements have been added is

$$P(s_i = 0 \mid n) = \left(1 - 2^{-i}\right)^n. \quad (3)$$

Consequently, the probability $q_k(n)$ of having *at least* k consecutive ones at the beginning of the sketch is

$$q_k(n) = \prod_{i=1}^{k}(1 - P(s_i = 0|n)) = \prod_{i=1}^{k}(1 - (1 - 2^{-i})^n). \quad (4)$$

The probability for exactly k initial ones is $q_k(n) - q_{k+1}(n)$. This allows us to obtain the expected value of the length of the initial, uninterrupted sequence of ones $Z(n)$, which is

$$E[Z(n)] = \sum_{k=1}^{w} k \cdot (q_k(n) - q_{k+1}(n)). \quad (5)$$

We can combine this with (2) to get an expression for φ depending on the multiplicity n:

$$\varphi(n) = 2^{E[Z(n)]}/n. \quad (6)$$

The dependency on n, however, can be overcome: Flajolet and Martin prove by means of Mellin transform and residue calculation that $\varphi(n)$ goes to $\varphi \approx 0.77351$ for $n \to \infty$. It converges so quickly that the value can be considered constant in practice, and can be used for obtaining estimates using (2). A numerically easy way to obtain the value of φ is to evaluate (6) for sufficiently large n (e. g., $n = 10^5$).

The variance of $Z(S)$ is quite significant, and thus the approximation is not very accurate. This can be improved by using multiple sketches in parallel to represent a single value, trading off accuracy against memory. The respective technique is called Probabilistic Counting with Stochastic Averaging (PCSA) in [30]. With PCSA, each element is first mapped to one of the sketches by using a uniformly distributed hash function, and is then added there. If m sketches are used, denoted by $S_1, \ldots, S_m$, then the estimate for the total number of distinct items added is given by

$$C(S_1, \ldots, S_m) := m \cdot 2^{\sum_{i=1}^{m} Z(S_i)/m}/\varphi. \quad (7)$$

One can identify a PCSA set with an $m \times w$ matrix, where each row is a standard FM sketch. Upon addition of an element, a uniformly distributed hash function selects one row, and a second, independent, geometrically distributed hash function picks one column. The one single bit located at these coordinates in the matrix is then set to one.

For a sufficiently large number of elements, PCSA yields a standard error of approximately $0.78/\sqrt{m}$ [30]. For very small element counts in the order of m or below, however, there are initial inaccuracies. We will come back to this issue later.

IV. Probabilistic Multiplicity Counting

In the course of this section, we will now step by step construct a new network traffic accounting mechanism, Probabilistic Multiplicity Counting (PMC). PMC is able to determine the multiplicities of individual keys in a multiset—or, equivalently, the number of occurrences of each flow ID in a stream of network packets.

A. Counting individual elements

As our first step towards PMC, we consider a significantly simpler problem: we assume that all elements are identical, i. e., that all packets belong to one single flow. Thus, we only need to count their total number. We will do so probabilistically, by "abusing" FM sketches in a certain way. By itself, this is of limited utility—but it will subsequently become a central building block of PMC.

We can turn an FM sketch into a simple probabilistic counter in the following way. Consider a stream of elements that are all pairwise distinct. In that case, the cardinality of the multiset (which is estimated by FM sketches) and the total number of elements are identical. If the hash function h used in the FM sketches is good, then the sequence of hash values for a stream with all-distinct elements is equivalent to a sequence of independent, geometrically distributed random numbers. We can exploit this property: we take an FM sketch, and *simulate* a stream of all-distinct elements by not using a hash function at all, but instead setting a bit at a random, geometrically distributed position whenever we want to increment the "counter". Clearly, the effect on the FM sketch is the same as if a "new" element in the multiset had been observed. This can analogously be done for FM sketches with PCSA, by additionally selecting one of the m sketches at random for each addition.

Note that it may (and will) happen that the same random number occurs multiple times, and the same bit is enabled in the FM sketch. This, however, is not different from the case where the hash function in standard FM sketches yields the same hash value for distinct elements; this is taken into account in the construction of FM sketches.

In order to state this more formally, we introduce $\mathrm{rand}(m)$ to denote a function that gives us a uniformly distributed random integer in the range $[1, m]$. We furthermore write $\mathrm{georand}(w)$ for a geometrically distributed random number where the probability that $\mathrm{georand}(w) = i$ is 2^{-i} for $i = 1, \ldots, w-1$ and is 2^{-w+1} for $i = w$ (as mentioned in Sec. III-B, the exception for $i = w$ simplifies the implementation). To increment the probabilistic "counter" built from a $m \times w$ PCSA matrix, we choose i using $\mathrm{rand}(m)$ and j using $\mathrm{georand}(w)$ and set the element at coordinates (i, j) in the matrix to one. An estimate can be obtained using the standard FM sketch PCSA evaluation procedure (7).

B. Virtual PCSA matrices

Probabilistic Multiplicity Counting (PMC) consists of two central algorithms: one for capturing the statistics in a special data structure during the measurement, and another one for extracting flow size estimates from this structure. We are now ready to introduce the first of these components. While it is of course desirable that both algorithms are as simple as possible, the capturing algorithm is the really performance critical one: it must be executed for each captured packet in real-time.

The key idea of PMC is to use one FM sketch matrix for each flow ID, in combination with the counting methodology from the previous subsection. However, these matrices are not stored explicitly and individually, because this would require to look up the correct matrix in some way whenever a packet is processed—an expensive operation which, as it turns out, can be avoided entirely. Instead, PMC uses what we call a *virtual PCSA matrix* for each flow. The physically stored data structure in which PMC collects information about the packet stream is a simple bit field B of size l bits, where l can be chosen arbitrarily. For any given flow ID f, the individual entries of the virtual PCSA matrix are mapped to positions in B by means of a hash function H with the following signature:

$$H : \mathcal{F} \times [1, m] \times [1, w] \to [1, l],$$

where $\mathcal{F}$ is the set of possible flow IDs and m and w are the dimensions of the virtual PCSA matrices. If the bit at row i and column j in the virtual PCSA matrix of flow f is to be accessed, we evaluate $H(f, i, j)$. This yields the corresponding position in the bit field B. It is straightforward to use any uniformly distributed hash function with sufficiently random output in the role of H: the input parameters can simply be concatenated to a single bit string. The idea of virtual PCSA matrices and mapping their entries to one long, single bit field is visualized in Figure 1.

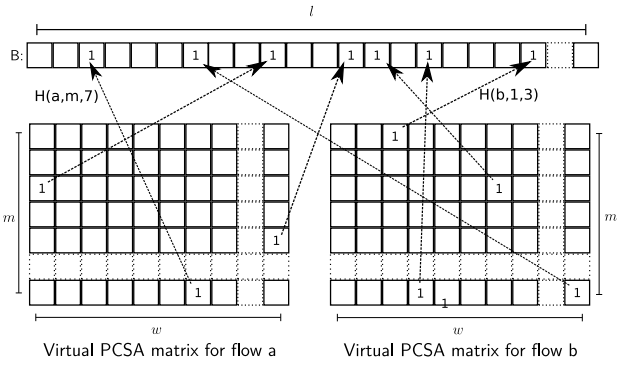


Figure 1. Mapping of individual entries in two distinct matrices a, b to B.

The dimensions of the virtual PCSA matrices—that is, m and w—can be chosen arbitrarily. We will come back to both parameters later on. Of course, it may happen that multiple positions from the same or different virtual matrices are mapped to the same bit in B. We will soon show that these *hash collisions* can effectively be dealt with.

The PMC data acquisition algorithm consequently works as follows: initially, all positions in B are set to zero. Whenever we encounter a packet from flow f, we generate a uniformly distributed random number i in the range $[1, m]$ and a geometrically distributed random number j in the range $[1, w]$. These are the row and column of the virtual matrix entry that is set to one. We then evaluate $H(f, i, j)$ and set the resulting bit position in B. Algorithm 1 summarizes this operation.

Algorithm 1 PMCCOUNT(f)

1: $i \leftarrow \mathrm{rand}(m)$
2: $j \leftarrow \mathrm{georand}(w)$
3: $B[H(f, i, j)] \leftarrow 1$

Algorithm 2 GETZSUM(f)

```
1: Z ← 0
2: for i = 1 ... m do
3:     for j = 1 ... w do
4:         if B[H(f,i,j)] = 0 then break
5:     end for
6:     Z ← Z + (j − 1)
7: end for
8: return Z
```

Algorithm 1's properties make very efficient implementations in both software and hardware possible if suitable pseudorandom number generators and hash functions are chosen. For generating uniform randomness, two examples of very fast pseudorandom number generators are complementary multiply-with-carry [31] and xorshift [32]. They combine very good randomness with a very simple, branchless algorithmic structure. It might seem that implementing $\mathrm{georand}(w)$ is particularly difficult, but starting from uniform randomness it is actually very easy: one may simply determine the position of the first one in a uniformly distributed random bit string [30]. This is very easy in hardware, and for software implementation there is valuable support in many current CPUs. For instance, on the x86 architecture the `bsf` and `bsr` opcodes can be used (starting from the Intel P6 architecture, they even require only one single CPU cycle). Regarding the hash functions, Henke et al. [33] compare various hash functions with respect to their suitability for network measurement applications. This includes well-suited candidates for our purposes. Upon careful implementation, Algorithm 1 can thus make optimal use of modern pipelined CPU designs for maximum performance.

C. Obtaining estimates from the bit field

We now turn towards the question how we can obtain an estimate for the multiplicity of a given flow ID f from B. To this end, we extract f's virtual matrix from B, by iterating over the parameter tuples (f, i, j), where $i = 1, \ldots, m$ and $j = 1, \ldots, w$ and evaluating H for each combination. In combination with calculating the sum of the lengths of the initial sequences of ones in each row (as it is used in the PCSA evaluation formula), this is what Algorithm 2 accomplishes.

Then, however, the question arises how we can map Algorithm 2's output to an estimate. As an initial approach, we could use the standard FM sketch PCSA evaluation according to (7). This, however, will not work: hash collisions can lead to bits that are incorrectly set to one ("false positive bits"), if their storage position in B coincides with another bit (from the same or another virtual matrix) that has previously been set to one. This does not occur for FM sketches, so it is not taken into account there.

Observe that bits in a virtual matrix will never be incorrectly set to zero (because the capturing algorithm can only enable, but never disable bits). Thus, basically, our virtual matrix is a standard PCSA FM sketch matrix, where some of the zero bits may be flipped to one. Given that the hash function H is sufficiently random, all bits in the virtual matrix are mapped to de-facto-random positions in B. Thus, the probability of a false positive bit is solely determined by the fraction p of bits that is set to one in B, i. e., by the *fill rate* of B.

With the standard FM sketch evaluation, this effect will lead to overestimation: the length of the initial sequence of ones in an FM sketch may increase due to false positive bits. This in turn results in a higher value of Z in (7) and consequently in a higher estimate. The higher p, the more severe this effect will be. Upon evaluation, p can easily be determined from B by counting the bits set to one and dividing this number by the bit field length l. We therefore revisit Flajolet and Martin's arguments in the light of false positive bits, and derive an evaluation formula that takes p into account. This will enable us to obtain unbiased estimates even in the presence of a significant number of collisions.

We start with the probability $q_k(n)$ that at least the first k bits in a sketch row are set after n additions as given in (4). We observe that q_k is now also a function of p, and obtain a modified version of (4) as follows:

$$q_k(n,p) = \prod_{i=1}^{k} \left[1 - \left(1 - 2^{-i}\right)^n \cdot (1-p)\right]. \tag{8}$$

This formula can be understood as follows: a bit is zero if and only if 1) it has not been enabled by an addition to the respective virtual matrix and 2) it is not a false positive bit. Observe that for the case $p = 0$ (8) is identical to (4).

Analogously, we obtain $E[Z(n,p)]$ as follows:

$$E[Z(n,p)] \quad = \quad \sum_{k=1}^{w} k \cdot (q_k(n,p) - q_{k+1}(n,p)). \tag{9}$$

For $\varphi(n,p)$, now also depending on p, we get

$$\varphi(n,p) = 2^{E[Z(n,p)]}/n. \tag{10}$$

As in the case of FM sketches, $\varphi(n,p)$ converges very quickly to a constant value for large n; the limit, however, now depends on p. Just as above in Sec. III-B, we can obtain the value $\varphi(p)$ from the above formula numerically, by evaluating for large n. Two example values for $\varphi(p)$ are $\varphi(0) = 0.78$ and $\varphi(0.5) = 1.85$. If we then replace φ in (7) by $\varphi(p)$, we obtain an estimation methodology that takes false positives into account.

It should be noted that it is also possible to adapt Flajolet and Martin's analytical arguments regarding φ to include false positive bits. This yields an analytical expression for $\varphi(p)$. However, the calculations are rather laborious, so that they exceed scope (and page limit) of this paper significantly. Moreover, evaluating the resulting expression is computationally much more costly than determining $\varphi(p)$ using (10), without giving higher accuracy.

D. Dealing with small multiplicities

We are now at a point where we have mechanisms for adding elements to a PMC bit field, and for extracting estimates. However, it cannot estimate flow sizes below $m/\varphi(p)$ (typically a few ten, at most a few hundred packets), because

Algorithm 3 GETEMPTYROWS(f)

```
1: k ← 0
2: for i = 1 ... m do
3:     if B[H(f, i, 1)] = 0 then k ← k + 1
4: end for
5: return k
```

Algorithm 4 PMCESTIMATE(f)

```
1: k ← GETEMPTYROWS(f)
2: if k/(1 − p) > 0.3 · m then return −2m · log k/(m·(1−p))
3: Z ← GETZSUM(f)
4: return m · 2^{Z/m}/φ(p)
```

the exponent in (7) cannot take on values below zero. This effect has been observed before for FM sketches [30], [34], and it likewise exists in PMC. Fortunately, we can do significantly better by a slight modification. This modification affects only the evaluation algorithm; the capture part remains untouched.

We propose to alleviate the small value problem by a modified variant of HitCounting [35]. HitCounting solves the same problem as FM sketches—determining the cardinality of a multiset—with different means. It uses a bitmap of size m and a uniformly distributed hash function that maps elements to bit positions. The bit field is initialized to zero, positions are set to one when they are "hit". From the number of positions that are still zero, the number of insertion operations with distinct elements is estimated: after how many additions can we expect that k out of m positions are still zero? This is closely related to the coupon collector's problem.

In standard HitCounting, after n additions, each of the m bit positions had n "chances" of being hit, each with probability $1/m$. So, the probability π that any individual position is still zero is

$$\pi = (1 - 1/m)^n \approx e^{-n/m}. \tag{11}$$

Consequently, we may expect a total of $k = m \cdot \pi$ bit positions that are still zero. By combining this and solving for n, the cardinality estimator is obtained:

$$n = -m \cdot \log(k/m). \tag{12}$$

We apply this idea to our problem in the following way: we introduce a second evaluation method that is especially designated to small flows. We then discuss how one can dynamically choose the better suited method upon extracting an individual estimate. The alternative method does not work on the full virtual matrix, instead, we consider only the first matrix column. Due to the geometric distribution of the column hash function the probability that an addition happens in the first column is $1/2$. Assume that n packets of flow f have been sampled. If B's fill rate (and thus the false positive bit probability) is p, then the probability π' that the first bit in any given row in f's virtual matrix is zero is

$$\pi' = (1 - 1/2m)^n \cdot (1 - p) \approx e^{-n/(2m)} \cdot (1 - p). \tag{13}$$

In analogy to above, we can expect $k = m \cdot \pi'$ such rows. We can determine k as shown in Algorithm 3, and obtain an estimator for n:

$$n = -2m \cdot \log k/(m \cdot (1 - p)). \tag{14}$$

Asked for the multiplicity estimate of a flow ID f, we must make a choice which evaluation methodology to use: the one from the previous subsection, or modified HitCounting. The virtual matrix evaluation can take much more information into account and is clearly preferable if larger multiplicities are estimated, whereas modified HitCounting avoids the problems with small estimates. Whang et al. [35] note that HitCounting yields most accurate results if the fraction of non-set bits is above 30 %. We make use of this observation and design a hybrid evaluation algorithm. We must again take false positive bits into account, so we adjust Whang et al.'s criterion. To this end, we estimate the number k' of zero first column bits *if there were no false positives*. From (13) we see that k' can be obtained from k as $k' = k/(1 - p)$. If $k' > 0.3 \cdot m$, we use modified HitCounting. This will be the case for very small flows. Otherwise, the method from Sec. IV-C is applied.

We emphasize again that the decision between HitCounting and the sketch-based methodology is made upon evaluation of an individual flow multiplicity. Both algorithms are based on the same bit field, and the procedure for collecting flow information is identical for all packets in all flows. The final PMC estimation procedure is summarized in Algorithm 4.

E. Parameter choices

There are a number of parameters in PMC, the choice of which deserves discussion. These include the dimensions m and w of the virtual matrices and the size l of the bit field B. Fortunately, the role and impact of all these parameters is quite straightforward, so that they can easily be used for well-directed tradeoffs.

The least critical parameter is w. In fact, once w is large enough, it does not have any significant impact at all: very high values of $\mathrm{georand}(w)$ are extremely unlikely. So, even for very large flows, we can do very well with, for instance, $w = 32$. Because the virtual matrices are not explicitly stored, w does not affect storage requirements either.

A larger number m of rows in the virtual matrices means that the PMC data capture algorithm will distribute its "hits" over a larger number of different bits for the same flow ID. Thus, we may expect that higher values of m result in higher accuracy, but at the same time lead to the bit field B running full more quickly. For our evaluations, we used values between $m = 32$ and $m = 256$, because they constitute good tradeoffs between accuracy and resource requirements.

By the bit field size l we can adjust the total number of bits that is available to all virtual matrices. If we use a larger bit field, the fill rate (and thus the false positive bit probability) will be lower after the same number of additions. This will result in higher accuracy. However, more memory for B results in higher cost, so practical implementations will have to find a good tradeoff. Typically, a few megabit will constitute a reasonable design point. When the bit field is in danger of

becoming too full, it is always possible to mirror B to some background storage, and continue with an empty bit field, as it has also been proposed for MRSCBF [2]. For continuous operation we envision using a frame buffer-like architecture with two identical bit fields: while PMC is streaming bits to one bit field, the other field can be exported to an external device for evaluation and zeroed afterwards.

Note that adjustments of l, m, or w do all not affect the effort for collecting data—when we tune PMC for lower memory resource usage or higher accuracy, Algorithm 1 stays just as simple as it is. This is a very interesting property, which distinguishes PMC from existing approaches.

V. Evaluation

In this section we evaluate the performance of PMC in comparison to MRSCBF [2]. Unless otherwise stated, we used $m = w = 32$ for PMC. The parameters of MRSCBF were set as proposed in [2] ($g = 32$ or $g = 64$ hash groups, $r = 9$ resolutions [with 3, 4, 6, 6, 6, 6, 6, 6, 6 bits per bucket] and $\alpha = 0.25$). Two evaluation methods for MRSCBF are discussed in [2], MLE and MVE. Here we use MVE; according to [2], the accuracy is comparable to MLE, but MVE is much easier to evaluate.

A. Accuracy

In the course of this section, we will proceed from rather abstract, artificial settings to increasingly realistic setups. While abstract setups allow for a detailed understanding of the results without non-deterministic real-world effects, more realistic traffic patterns provide insight into the performance that may be expected in real applications.

As a first step, we assume an infinitely large bit field, i. e., $p = 0$. We can do so by not implementing the algorithms with a real bit field, but instead storing the hash function parameter tuples of all set bits explicitly, so that no false positive bits occur. This is possible for both PMC and MRSCBF. It gives us, in some sense, a "best case" picture of the algorithms' accuracy if the fill rate of the bit field is very small.

Figure 2 shows the standard error[3] for both PMC and MRSCBF (with $g = 32$) for an increasing multiplicity in such a setting. These figures were obtained by processing flows with a fixed number of packets and subsequently determining the estimation error. We used 4096 samples per x value, the y-axis error bars show the range of the relative errors with 5 % extremal values cut off above and below. As one might have expected, the standard error for PMC without false positive bits equals the theoretical value of $0.78/\sqrt{m} = 0.138$ found by Flajolet and Martin [30] for FM sketch PCSA sets with identical parameters. Regardless of the multiplicity, the standard error of MRSCBF is higher.

Based on the same implementations of PMC and MRSCBF, we can also simulate false positive bits, by setting each zero bit to one with a given probability p in the evaluation. This

[3]The standard error is a measure for the relative deviation of the estimates from the correct value. It is defined as the standard deviation of the normalized error samples.

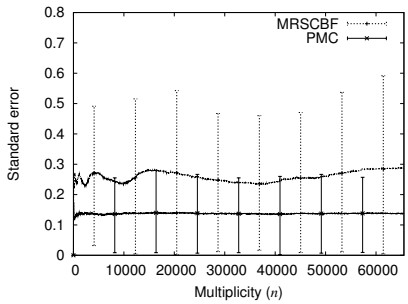


Figure 2. Accuracy of PMC and MRSCBF without false positive bits.

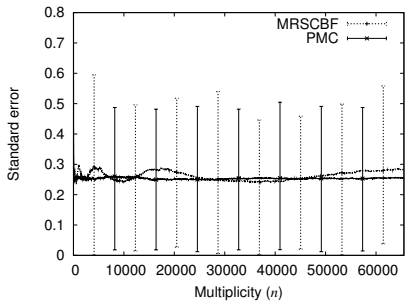


Figure 3. Accuracy of PMC and MRSCBF with $p = 0.5$.

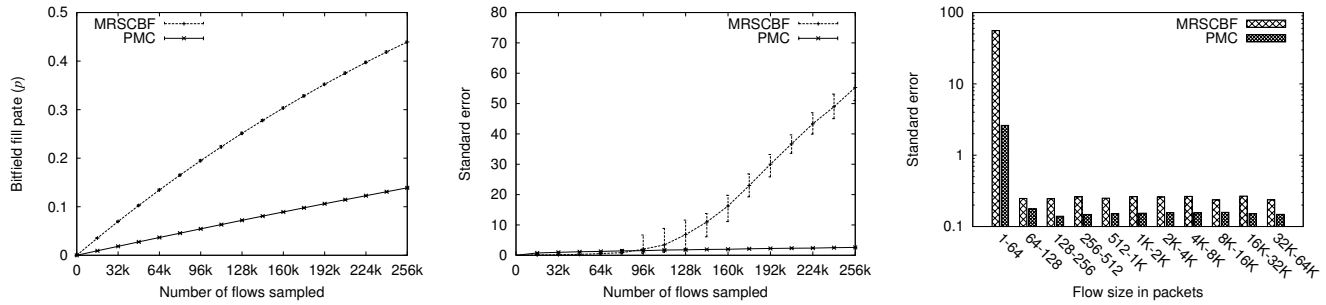
allows us to examine PMC's and MRSCBF's performance in a deterministic setting with arbitrary false positive bit rates. In Figure 3 we have done so with $p = 0.5$. It shows that PMC and MRSCBF perform almost equally well in this setup. For MRSCBF, the individual "resolutions" are clearly visible as variations of the error over the multiplicity, whereas PMC's error is largely independent from the multiplicity.

B. Stress test

Further benefits of PMC become clear if we do a "stress test": we examine the performance for an increasing amount (up to 256k flows) of artificially generated traffic with Pareto distributed flow sizes (scale=1, shape=1.2). We give PMC and MRSCBF a bit field of size 1 MB each.[4] Figure 4(a) shows how the bit fields fill up as this traffic is processed. The experiment was repeated 64 times with different random seeds. Again the y-axis error bars indicate the extremal values with 5 % cut-off (some error intervals are so small that they are barely visible in the figures). We stopped the experiment after 256k flows have been processed. At this point, MRSCBF had reached a fill rate of about 44 %, whereas PMC had set only ca. 14 % of the bits to one. As we saw above, lower fill rates increase the accuracy.

The standard error for these experiments can be seen in Figure 4(b). Starting from about 100k flows, the mean standard error of MRSCBF (then at about 30 % fill rate) increases

[4]As a side note, if we were to give up the write-only paradigm and used one explicit 64 bit counter per flow, then, for 256k flows, these counters would take up 2 MB of memory alone—not counting the additional space for a lookup data structure to locate the right counter for an incoming flow ID.



(a) Fill rate for increasing number of flows. (b) Standard error for increasing number of flows. (c) Standard error of different flow sizes.

Figure 4. Results from PMC and MRSCBF in the Pareto flow stress test.

massively. The reason is that occasionally very large errors for very small flows occur.[5]

In Figure 4(c), the situation after 256k flows have been processed is analyzed in more detail. The figure shows the standard error for different flow size ranges. Note the log scale on both axes. For both algorithms, the errors are largest for very small flows up to 64 packets. However, PMC outperforms MRSCBF over the whole spectrum of flow sizes. The figure ends with the range 32K–64K, since, due to the Pareto distribution, there are only very few flows above that size.

C. Real-world traffic

To test PMC and MRSCBF with real-world traffic, we monitored the traffic on our university network's Internet uplinks over 24 hours (on May 28th, 2009). These links have a total bandwidth of 310 MBit/s, which still allows for exact traffic statistics on standard hardware for comparison purposes. We observed a total of about $1.5 \cdot 10^9$ packets. As some subnets are charged by traffic usage, we decided to evaluate the total amount of traffic per internal IP address, i. e., all packets with the same internal IP belong to one flow. We felt that traffic below 1 MB/day is negligible and focus on hosts with higher traffic volume.

For both PMC and MRSCBF we used a single bit field of 4 MB size (without bit field swapping, one bit field for the whole day), and parameters to maintain a reasonable fill rate at the highest possible accuracy (exact parameter values are stated in the figure titles). Since we want to collect statistics on the total amount of data transferred (not the total number of packets), we need a way to adapt PMC and MRSCBF accordingly. For each packet, we draw a random number in the range $[1, \mathrm{MTU}]$, and process the packet (i. e., set bits in the bit field) only if its size exceeds the random number. With this simple trick, we (probabilistically) count "MTU equivalents" instead of packets, without a significant increase in complexity per processed packet. The flow size estimate is obtained by multiplying the estimate with the MTU.

[5]Tracking this issue down revealed that—speaking in terms of the MRSCBF paper—sometimes very high resolutions have a smaller relative incremental inaccuracy due to false positives, which leads to gross overestimation of small flows. We do not see an easy way to fix this problem.

Figures 5(a) and 5(b) show the results for PMC and MRSCBF, respectively. Each point in the figures stands for one flow. The x value is the true flow size, the y value is the estimate. Ideally, all points should lie on the diagonal line. For comparison purposes, we did the same experiment with sFlow (that is, with packet sampling). We used the highest sample rate (and thus the highest complexity and the highest accuracy) that the standard allows: 1 in 512 packets. Figure 5(c) shows the outcome.

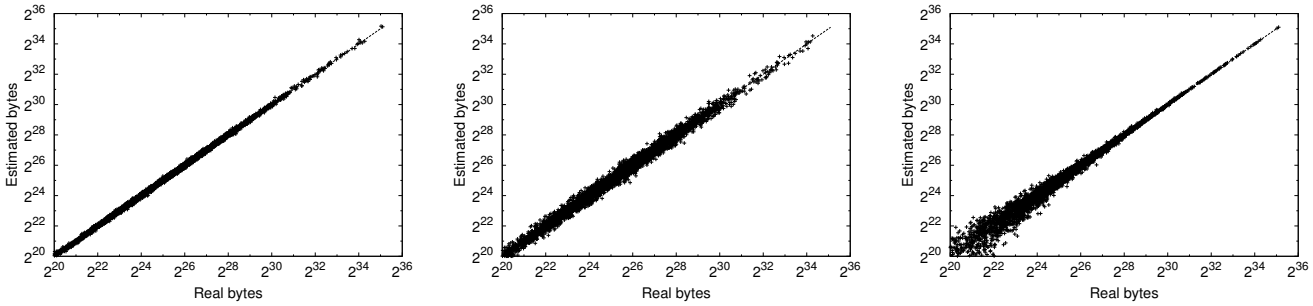
From the results it can be clearly seen that packet sampling has severe accuracy problems especially for flows up to 1 GB traffic volume. It results in a standard error of 0.239. We also observe that PMC (with a standard error of 0.056 at 22 % fill rate) again outperforms MRSCBF (standard error 0.178 at 23 % fill rate). It provides much higher accuracy at lower algorithmic complexity.

D. Complexity of the counting operations

In Table I, we compare the counting operations of PMC and MRSCBF in terms of the required number of steps for processing a packet. For the parametrizations used in the previous subsections, we show how many random numbers need to be generated in order to process a packet, how many hash operations need to be performed, how many bits are written, and how many conditional branch instructions are involved. For PMC, these numbers are constant; for MRSCBF, the exact steps vary from packet to packet, so we show the possible range of values for each operation type. The much simpler structure of PMC is immediately evident from these numbers.

Table I
COMPARISON OF PMC AND MRSCBF COUNTING OPERATIONS.

	PMC	MRSCBF
Random number generations	2	9–18
Hash operations	1	3–49
Bit write operations	1	3–49
Conditional branches	0	8



(a) PMC ($l = 4$ MB, $m = 256$, $w = 32$). (b) MRSCBF ($l = 4$ MB, $g = 64$, $r = 9$, $\alpha = 0.25$). (c) sFlow ($r = 1/512$).

Figure 5. Scatterplots of real-world traffic experiments.

VI. Conclusion

In this paper, we have introduced Probabilistic Multiplicity Counting (PMC), a novel probabilistic technique to determine the multiplicity of elements in a multiset. We showed how it can be applied to high-performance per-flow traffic measurement. We introduced the PMC algorithms for sampling data into a bit field, and for extracting multiplicity estimates from that bit field. Finally, we assessed PMC's performance in comparison to MRSCBF—a previously proposed technique with similar aims—and to standard sFlow. It became clear that PMC achieves superior accuracy with a very simple algorithm.

Acknowledgements

The authors are grateful to Detlef Lannert, Wolfgang Müller, Stephan Olbrich, and Klaus Szymanski from the NOC of our university for their support with the real-world experiments. We furthermore thank our student helpers Adam Görtz and Benito van der Zander for their great implementation work.

References

[1] N. Hohn and D. Veitch, "Inverting sampled traffic," in *IMC '03*, 2003.
[2] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *IEEE Journal on Selected Areas of Communications*, vol. 24, no. 12, pp. 2327–2339, Dec. 2006.
[3] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall, "Sampling and filtering techniques for IP packet selection," RFC 5475, Mar. 2009.
[4] Cisco Systems, "NetFlow," http://www.cisco.com/web/go/netflow.
[5] ——, "Sampled NetFlow," http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/12s_sanf.html.
[6] InMon Corp., "sFlow Version 5," http://sflow.org/sflow_version_5.txt.
[7] N. Duffield, C. Lund, and M. Thorup, "Charging from sampled network usage," in *IMW '01*, 2001, pp. 245–256.
[8] InMon Corp., "sFlow Accuracy & Billing," http://www.inmon.com/pdf/sFlowBilling.pdf.
[9] B.-Y. Choi and S. Bhattacharyya, "Observations on Cisco sampled NetFlow," *SIGMETRICS Performance Evaluation Review*, vol. 33, no. 3, pp. 18–23, 2005.
[10] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems*, vol. 21, no. 3, pp. 270–313, 2003.
[11] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Analysis of a statistics counter architecture," *HOTI '01*, 2001.
[12] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," in *SIGMETRICS '03*, 2003, pp. 261–271.
[13] Q. Zhao, J. Xu, and Z. Liu, "Design of a novel statistics counter architecture with optimal space and time efficiency," *SIGMETRICS Performance Evaluation Review*, vol. 34, no. 1, pp. 323–334, 2006.
[14] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: a novel counter architecture for per-flow measurement," in *SIGMETRICS '08*, 2008, pp. 121–132.
[15] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 280–292, 2001.
[16] A. Kumar and J. Xu, "Sketch guided sampling – using on-line estimates of flow size for adaptive data collection," in *INFOCOM '06*, 2006.
[17] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, and Y. Chen, "Accurate and efficient traffic monitoring using adaptive non-linear sampling method," in *INFOCOM*, 2008, pp. 26–30.
[18] L. A. Adamic and B. A. Huberman, "Zipf's law and the Internet," *Glottometrics*, vol. 3, pp. 143–150, 2002.
[19] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *VLDB '02*, 2002, pp. 346–357.
[20] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Transactions on Database Systems*, vol. 28, no. 1, pp. 51–55, 2003.
[21] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: an efficient algorithm for finding heavy hitters," *SIGCOMM Computer Communications Review*, vol. 38, no. 1, pp. 5–5, 2008.
[22] S. Cohen and Y. Matias, "Spectral bloom filters," in *SIGMOD '03*, 2003.
[23] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[24] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Mathematics*, 2005.
[25] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, pp. 29–38, 2004.
[26] R. Donghua, L. Chuang, C. Zhen, N. Jia, and P. D. Ungsunan, "Handling high speed traffic measurement using network processors," in *ICCT '06*, 2006, pp. 1–5.
[27] B. Ribeiro, T. Ye, and D. Towsley, "A resource-minimalist flow size histogram estimator," in *IMC '08*, 2008, pp. 285–290.
[28] A. Kumar, M. Sung, J. J. Xu, and E. W. Zegura, "A data streaming algorithm for estimating subpopulation flow size distribution," in *SIGMETRICS '05*, 2005, pp. 61–72.
[29] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 933–946, 2005.
[30] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, no. 2, pp. 182–209, Oct. 1985.
[31] G. Marsaglia, "Random number generators," *International Journal of Applied Mathematics & Statistics*, vol. 2, no. 1, pp. 2–13, May 2003.
[32] ——, "Xorshift RNGs," *Journal of Statistical Software*, vol. 8, no. 14, pp. 1–6, 2003.
[33] C. Henke, C. Schmoll, and T. Zseby, "Empirical evaluation of hash functions for multipoint measurements," *SIGCOMM Computer Communications Review*, vol. 38, no. 3, pp. 39–50, 2008.
[34] B. Scheuermann and M. Mauve, "Near-Optimal Compression of Probabilistic Counting Sketches for Networking Applications," in *Dial M-POMC '07*, Aug. 2007.
[35] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems*, vol. 15, pp. 208–229, 1990.