# Distributed Probabilistic Network Traffic Measurements

**Alexander Marold*[1], Peter Lieven[2], and Björn Scheuermann[2]**

1 **University of Duisburg-Essen**
**Institute for Experimental Mathematics**
alexander.marold@iem.uni-due.de
2 **Heinrich Heine University Düsseldorf**
**Mobile and Decentralized Networks Group**
{lieven,scheuermann}@cs.uni-duesseldorf.de

**Abstract**

Measuring the per-flow traffic in large networks is very challenging due to the high performance requirements on the one hand, and due to the necessity to merge locally recorded data from multiple routers in order to obtain network-wide statistics on the other hand. The latter is non-trivial because traffic that traversed more than one measurement point must only be counted once, which requires duplicate-insensitive distributed counting mechanisms. Sampling-based traffic accounting as implemented in today's routers results in large approximation errors, and does not allow for merging information from multiple points in the network into network-wide total traffic statistics. Here, we present Distributed Probabilistic Counting (DPC), an algorithm to obtain duplicate-insensitive distributed per-flow traffic statistics based on a probabilistic counting technique. DPC is structurally simple, very fast, and highly parallelizable, and therefore allows for efficient implementations in software and hardware. At the same time it provides very accurate traffic statistics, as we demonstrate based on both artificial and real-world traffic data.

**Digital Object Identifier** 10.4230/OASIcs.KiVS.2011.133

## 1 Introduction

In this paper, we consider the problem of measuring *network-wide flow sizes* in high-speed network environments. We are interested in determining how many distinct packets of each given flow traversed a network during some time period. These data are relevant, for example, in the context of network usage monitoring, network layout optimization, or data traffic accounting. Gathering such statistics is not straightforward, though, if the considered network has a non-trivial topology: if there is no single, central point in the network that will "see" all the traffic, it is necessary to take measurements at multiple different points to obtain a complete picture. As packets may enter and leave the network at different points, and may take different paths through it, a packet may traverse multiple measurement points. Then, the question arises how to merge the individual routers' measurements without counting packets that have been observed at multiple routers more than once.

The reasons that make this problem so challenging are twofold. First, solving it requires distributed duplicate-insensitive counting, which is algorithmically challenging: if router $A$ observed 500 packets of flow $x$, and router $B$ saw 800 packets of this flow, how can we determine how many *distinct* packets of this flow have traversed the network? Did all of the 500 packets at $A$ also traverse $B$, or did $B$ just see a subset of them? Or are these two

* The work described herein was done while Alexander Marold was with the University of Düsseldorf.

17th GI/ITG Conference on Communication in Distributed Systems (KiVS'11).
Editors: Norbert Luttenberger, Hagen Peters; pp. 133–144

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

sets of packets completely disjoint, so that there were a total of 1,300 distinct packets of flow $x$ the network? Second, it must be taken into account that traffic measurements in high-speed network environments are a highly performance-critical task. For a single 40 Gbps link, the available time for processing a minimum-size IP packet is only 12 ns. Switching and routing equipment with a potentially large number of high-speed ports cannot provide enough processing power and memory bandwidth to maintain exact per-flow statistics—at least not at reasonable cost [14, 21, 29, 22]. As Gilder's law states, network bandwidth increases more rapidly than processing power, so the situation is getting worse and worse.

Here, we propose a technique which is able to perform distributed, duplicate-insensitive traffic accounting with extremely low effort per processed packet. Our technique—termed Distributed Probabilistic Counting (DPC)—uses probabilistic techniques to obtain approximate flow sizes. To this end, DPC builds upon ideas developed in our Probabilistic Multiplicity Counting (PMC) algorithm first described in [22]. However, while PMC allows for high performance traffic accounting at a single measurement point with very high accuracy, it makes heavy use of random numbers and, as a result, does not allow to merge traffic statistics from multiple routers in a duplicate-insensitive manner. DPC avoids these random decisions entirely and is able to perform duplicate-insensitive merging of distributed measurements. This ability is the key distinguishing feature of DPC.

The probabilistic approach employed in DPC deliberately waives perfect accuracy and accepts an (adjustable) estimation error. In turn, it becomes possible to perform distributed duplicate-insensitive measurements with minimal computational and storage requirements: DPC is able to record information on a passing-by packet by setting only one single bit in a bit field. This operation can be performed in constant time, without loops or conditional branches in the code, and with write-only memory access. It is therefore ideally suited for pipelined and parallelized implementation, and can also easily and efficiently be realized in hardware. We will first focus on a basic DPC variant which can count the *number* of packets, and then also discuss how to measure the total *size* of the packets in a flow.

The remainder of this paper is structured as follows. We first discuss related work in the area of traffic monitoring in Sec. 2. Subsequently, we derive the Distributed Probabilistic Counting algorithm step-by-step from its basic building blocks in Sec. 3. We evaluate DPC with both artificial network traffic patterns and based on real-world network traces in Sec. 4, before we conclude this paper with a summary in Sec. 5.

## 2 Related Work

Due to its high practical relevance, the topic of flow measurement has attracted a lot of attention in recent years, and many approaches have been proposed. However, they typically cannot be used for duplicate-insensitive multipoint measurements. This fundamental constrain applies to deterministic techniques like [27, 26, 23], and likewise to packet sampling based network usage data generation and collection approaches like Cisco's NetFlow [2], IPFIX [29], sFlow [18], Sampled NetFlow [3], Sketch-Guided Sampling [20] or ANLS [15]. The well-known "Sample-and-Hold" approach [9] and its successors [25, 19, 6] reduce the storage overhead at a single measurement point, but can again not easily be generalized to duplicate-insensitive multipoint measurements. The same holds true for hash-based and (pseudo-)randomized traffic accounting techniques including Spectral Bloom Filters [4], Count-Min Sketches [5], Counter Braids [23], MRSCBF [7], and also for our own prior work PMC [22].

The lack of techniques to merge traffic statistics from multiple measurement points into a network-wide per-flow total has previously been recognized as a problem. Nevertheless,

there is only one existing approach that is able to fill this gap: hash-based sampling [11]. This technique builds upon existing sampling techniques as employed in, e. g., Sampled NetFlow. These standard techniques either record every $n$-th packet passing by, or they randomize the process and sample each packet with probability $1/n$. Based on the sampled subset of packets at a measurement point, the total amount of data is estimated. However, since only a (random) subset of the packets is considered, duplicate-insensitive merging of observations is not possible: different measurement points will draw different sample subsets of the passing-by packets, so one cannot reliably decide if two measurement points have seen entirely different sets of packets, or whether the same set of packets passed by and the nodes have just drawn different sample subsets from it.

Hash-based sampling overcomes this problem by using a hash value over the packet contents as a basis for the sampling decision. For instance, a packet might be sampled if its hash value modulo $n$ is zero, which results in one out of $n$ packets being sampled in expectation. This has the benefit that a packet will either be sampled at all measurement points it traverses, or at none of them, so that sampled packet sets from distinct measurement points can be compared. However, this approach exhibits a number of drawbacks. First, it shares the well-known granularity drawbacks of all sampling-based approaches [8, 17, 1, 9]. Moreover, in order to allow for duplicate-insensitive merging, it is necessary to collect and store additional payload data for each sampled packet, which implies significant storage requirements at the measurement points. Our proposed algorithm DPC is structurally so simple that it can process each packet, so sampling is not required. It does also not need to collect any per-packet information—and still allows for duplicate-insensitive merging of multiple measurement points' data.

## 3 The Distributed Probabilistic Counting Algorithm

Distributed Probabilistic Counting (DPC) builds upon the concept of FM sketches [10] to allow for efficient duplicate-insensitive counting. On this basis, it is able to process a traversing packet in $O(1)$ time. In the course of this section, we will first recapitulate the essential prerequisites for understanding DPC, which in particular includes a brief outline of the key ideas behind FM sketches. Subsequently, we discuss how distributed network flow size measurement can be accomplished on this basis.

Before we dive into the details of the algorithms, though, it is helpful to concretize the notion of a "flow" that we are going to use throughout this paper. By a flow we denote a subset of the packets in the network that are characterized by some common criterion. Each packet belongs to exactly one flow. For instance, a flow might consist of all the packets originating from the same source address, belonging to the same protocol, or traveling between the same pair of communicating hosts or subnets. The techniques proposed in this paper are independent from the specific criterion used to characterize the flows, as long as it is possible to determine the flow from the packet with little effort. For most practical purposes, determining the flow ID of a given packet does not incur more effort than to extract one or two header fields. This is trivial in both software and hardware. In the following, we may therefore assume that the flow ID of each processed packet is known.

### 3.1 FM sketches

FM sketches have been introduced by Flajolet and Martin to estimate the number of distinct elements in a multiset. Determining this number exactly requires $\Omega(n \log n)$ time for a multiset of size $n$; FM sketches provide a good estimate in linear time, i. e., with constant

effort per element. Before we step into the details of their application to network flow measurement, we briefly summarize the key concepts of FM sketches; a much more detailed description and analysis can be found in Flajolet and Martin's original publication [10].

An FM sketch is based upon a bit field $S = s_1, \ldots, s_w$, $w \geq 1$ and a hash function $h_1$ with geometrically distributed positive integer output, where the probability that $h_1(x) = i$ ($i \geq 1$) for any randomly picked element $x$ equals $P(h_1(x) = i) = 2^{-i}$. The bit field $S$ is initialized to zero. When an FM sketch is used to estimate the cardinality of a multiset $M$, each element $x \in M$ is hashed using $h_1$. Each output of $h_1$ is interpreted as an index in $S$, and the corresponding bit $s_{h_1(x)}$ is set to one (regardless of its current value). This leads to a bit pattern emerging in $S$, which, due to the geometric distribution of $h_1$, will typically have many 1-bits on the left hand side and many 0-bits on the right hand side.

Flajolet and Martin found that a good estimate for the number of distinct elements can be obtained from the length of the uninterrupted, initial sequence of ones in $S$, i. e., from

$$Z(S) := \min\{i \in \mathbb{N}_0 \mid s_{i+1} = 0\}. \tag{1}$$

The estimate $C$ is calculated from $Z$ as

$$C = \frac{2^Z}{\varphi} \qquad \text{where} \qquad \varphi \approx 0.77351. \tag{2}$$

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, essentially trading off accuracy against memory. The respective technique is called Probabilistic Counting with Stochastic Averaging (PCSA) in [10]. With PCSA, each element or packet is first mapped to one of the sketches by using a uniformly distributed hash function $h_2$, and is then added to this (and only this) sketch.

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

$$C(S_1, \ldots, S_m) := m \cdot \frac{2^{\sum_{i=1}^{m} Z(S_i)/m}}{\varphi}. \tag{3}$$

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 $h_2$ selects one row, and a second, independent, geometrically distributed hash function $h_1$ picks one column. The 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 $0.78/\sqrt{m}$ [10]. Increasing $m$ thus results in a higher estimation accuracy. Note that increasing $m$ also increases the total number of 1-bits in the PCSA matrix for a given, fixed multiset; this trait will be important for understanding DPC's parameter tradeoffs discussed in our evaluation. For very small element counts in the order of $m$ or below, there are well-known initial inaccuracies. These can be partially alleviated by switching to a different evaluation method, based on "hit counting" [28], when $Z$ is small. For details, we would like to refer the reader to the respective discussion in [22].

FM sketches (and likewise PCSA matrices) can be merged to obtain the total number of distinct elements added to any of them by a simple bit-wise OR. Combining the FM sketch with all elements of set $A$ and the FM sketch with all elements of set $B$ using bit-wise OR produces a FM sketch that is identical to the sketch of set $A \cup B$. Elements present in both $A$ and $B$ will obviously not be counted twice, since the respective bit will always have value 1 in both sketches. This trait—termed duplicate insensitivity—is an essential key to the operation of DPC, and one key reason why FM sketches are an ideal basis for distributed traffic measurement.

## 3.2 FM sketches for flow accounting

We will now consider the case of measuring traffic data in a very much simplified case, where there is only one single flow $f$ in the network. Subsequently, we will then turn to the problem of handling an arbitrary number of flows in the network in parallel.

We consider the set $P_f$ of all packets from $f$. Of course, we can only measure traffic that has traversed at least one measurement point; we will therefore assume that there are enough measurement points in the network to observe each packet at least once (e. g., all routers or, for transit traffic, all entry/exit nodes). Consequently, each measurement point $m \in M$ has observed a subset of $P_{f,m} \subseteq P_f$, such that

$$P_f = \bigcup_{m \in M} P_{f,m}.$$

In general, the subsets $P_{f,m}$ will not be disjoint. Due to the duplicate insensitivity property of FM sketches discussed above, though, the overlap between the observed packet sets of distinct measurement points is not an issue if FM sketches are used to determine packet counts: the sketches from multiple measurement points can then be merged using bit-wise logical OR, and packets that have been recorded at multiple measurement points will only be counted once. Bringing this idea to reality, though, poses a number of challenges.

The first question that arises is what to use as the input for hash functions $h_1$ and $h_2$. There are two conflicting aims: one the one hand, distinct packets should yield different inputs to the hash functions—if this does not hold, the packets are essentially considered identical, and the duplicate insensitivity property of FM sketches will result in an underestimation of the flow size. From this perspective, it appears desirable to use the maximum available amount of information—that is, the complete packet—as the hash input. On the other hand, though, processing packets at the measurement points is highly performance critical. Therefore, it is reasonable to keep the amount of data that needs to be hashed as small as possible. There is clearly a tradeoff.

In PMC, the problems associated with deciding what to hash are overcome by introducing random decisions in the algorithm. This, however, comes at the cost of sacrificing the duplicate insensitivitiy property of FM sketches. We explicitly need duplicate insensitivity for distributed measurements in DPC, so the solution employed in PMC is not an option here. However, we can take the work in the context of hash-based sampling into account, where a similar tradeoff exists. It has been explored in that context by Henke et al. [12], who found that using a subset of the network and transport header fields as the hash function input is enough. They propose a solution which includes different transport header fields depending on the used transport layer protocol. In the context of DPC, this is undesirable, as it requires complex decisions and conditional branches in the algorithm, which contradicts DPC's aim to stay algorithmically very simple, in particular avoiding conditional branches. We therefore opt for a solution which includes a fixed subset of the bytes in each packet.

In order to investigate such a solution, we performed experiments to see how much data is necessary to distinguish packets. As the data basis we used 17 hours of real-world traffic captured in the network of FH Salzburg and made available in the MOME database [16]. This is the same data set that has also been used by Henke et al.. We extract varying amounts of data from the packets in this trace: the IP header (ignoring those fields that are modified by intermediate routers) plus an increasing number of bytes from the IP payload. We then determined in each case how many distinct packets resulted in identical extracted byte strings, and thus in identical inputs to $h_1$ and $h_2$.

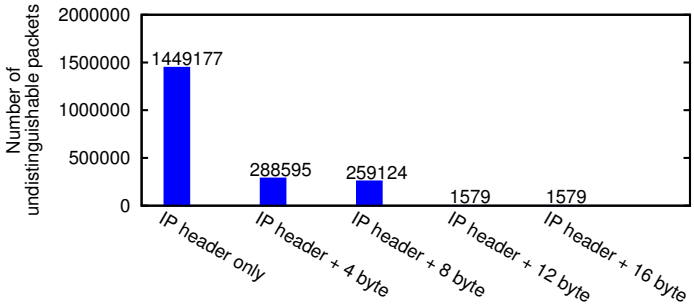



**Figure 1** Distinguishability of packets based on varying amounts of IP payload.

Figure 1 shows how many of the packets in the MOME Salzburg traces were indistinguishable from a previously seen packet, based on comparing the non-volatile fields of the IP header plus a varying initial fraction of the IP payload (0–16 bytes). The traces contain a total of ca. 12.6 million packets. Using the IP header alone is clearly insufficient, as this leads to huge numbers of non-distinguishable packets. As expected, when more bytes from the IP payload are taken into account, the accuracy increases. The steps to using 4, 8, and 12 bytes of IP payload in addition to the IP header result in significant improvements. Then, however, the number of undistinguishable packets has become very low already, and increasing the amount of data taken into account further to 16 bytes does not yield further benefits. Therefore, we use the IP header plus the first 12 bytes of IP payload as the input to hash functions $h_1$ and $h_2$ in the following.

## 3.3 Measuring all flows in parallel

So far, we have discussed how an FM sketch can be used to measure the number of packets in an individual flow. This alone, however, is of limited utility: even if each router used an FM sketch in the way discussed above for each individual flow, this would require to locate the FM sketch for the respective flow in the router's memory whenever a packet is to be recorded. This in turn would require using some lookup data structure to find the corresponding FM sketch for each processed packet—which clearly incurs far too much overhead.

We therefore take up the idea of *virtual matrices* from PMC [22]: there, the information about individual flows is not stored explicitly and separately. Instead, all bits describing all flows are mapped to one single, unstructured bit field $B$ of size $l$ bits. The parameter $l$ can be chosen arbitrarily and is typically in the order of a few megabits. This mapping is performed using another (uniform) hash function $H$, which maps a tuple of a flow ID plus the row and column coordinates of a PCSA matrix entry to one of the bits in $B$, i. e.,

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

where $\mathcal{F}$ is the set of possible flow IDs and $m$ and $w$ are the dimensions of the virtual PCSA matrices. This is visualized in Figure 2. If the bit at row $i$ and column $j$ in the virtual PCSA matrix of flow $f$ is to be accessed, $H(f, i, j)$ yields the corresponding index in $B$.

Recording information about packets in a router using DPC consequently works as follows: initially, all positions in $B$ are set to zero. Whenever a packet from flow $f$ is encountered, it is hashed using hash functions $h_1$ and $h_2$ (using the IP header plus 12 bytes of IP payload, as discussed above). This yields the row and column indices—denoted by $i$ and $j$, respectively—in $f$'s virtual matrix. Then, $H(f, i, j)$ is evaluated and the resulting bit in $B$ is set to one.

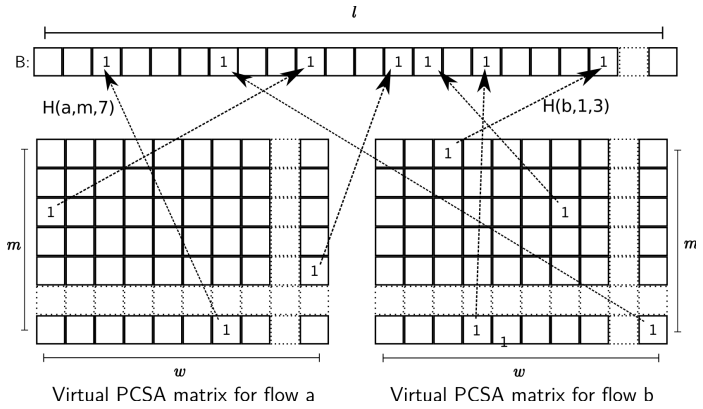



**Figure 2** Mapping of virtual matrix entries to $B$ [22].

All these operations can be implemented very efficiently in both software and hardware. In particular, if hash functions are chosen carefully, implementations without conditional branches are possible, which is ideal for using the algorithm on pipelined CPU designs. Moreover, DPC inherits the write-only property of FM sketches: counting a packet with DPC does not require reading from memory—instead, only one single bit needs to be written to $B$. The address of this bit can be determined from the packet alone, without looking up any information from memory. In particular, there is no need for a complex lookup data structure. This, too, makes the algorithm a perfect match for highly performance critical tasks like network traffic monitoring: in contrast to other approaches like, e. g., per-flow counters in combination with hash-based sampling, DPC's per-packet effort is constant.

Of course, it may (and will) occur that multiple positions from the same or different virtual matrices are mapped to the same bit in $B$. This must be taken into account when an estimate is extracted from a virtual matrix: it may happen that a value of one is read from bit position $H(f, i, j)$, even though the addressed entry $(i, j)$ in the virtual matrix of flow $f$ has never been set to one during the measurement period—simply because $H(f, i, j)$ is equal to $H(f', i', j')$ and the bit at position $(i', j')$ in the virtual matrix of flow $f'$ *has* been set to one. We call this a "false positive bit".[1]

In [22], we have argued that the probability that this happens is identical to the *fill rate* $p$ of bit field $B$, i. e., to the fraction of bits in $B$ that are set to one. The fill rate $p$ is trivial to determine from $B$, and can be used to adjust Flajolet and Martin's estimation formula accordingly. Due to space limitations it is not possible to repeat the reasoning in detail here, but the key result from [22] is that it suffices to substitute Flajolet and Martin's constant $\varphi$ in (2) or (3), respectively, by a value $\varphi_p$ which takes the bit field fill rate $p$ into account. This $\varphi_p$ can be obtained as the following limit

$$\varphi_p = \lim_{n\to\infty} 2^{E[Z(n,p)]}/n \tag{4}$$

where

$$E[Z(n,p)] = \sum_{k=1}^{w} k \cdot (q_k(n,p) - q_{k+1}(n,p)) \quad \text{with} \quad q_k(n,p) = \prod_{i=1}^{k} \left[1 - (1-2^{-i})^n \cdot (1-p)\right].$$

[1] Note that the inverse case of a "false negative bit" cannot occur, since bits are never reset to zero during one measurement interval.

The convergence in (4) is so fast that simply evaluating the formula for a sufficiently large value of $n$ (e. g., $n = 10^5$) suffices to obtain a practically usable value for $\varphi_p$.

Consequently, after a measurement period (e. g., one day) is over, the bit fields $B$ can be collected from all measurement points in the network. They can then be merged in a duplicate-insensitive manner by a bit-wise OR operation. For each flow $f$ of interest, it is then possible to extract the virtual PCSA matrix by iterating over the tuples $(f, i, j)$ for all virtual matrix entries $(i, j)$, evaluating $H$ for each of them, and thereby re-constructing $f$'s virtual PCSA matrix. This matrix can then be evaluated using Flajolet and Martin's methodology, with the modification of using $\varphi_p$ in the role of $\varphi$ as discussed above. The result is a duplicate-insensitive estimate for the total number of packets from flow $f$ observed in the entire network. By combining not all bit fields from all measurement points, but only some of them, it is of course likewise possible to obtain per-flow statistics for different parts of the network or for individual measurement points—based on the very same raw data.

## 3.4 Determining flow sizes instead of packet counts

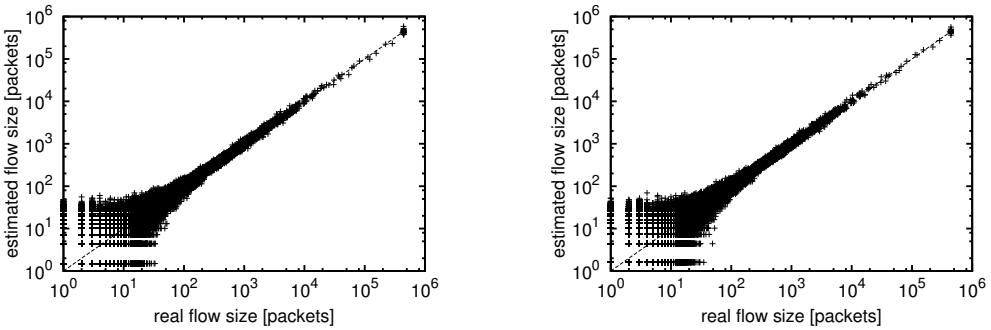
In some applications, including billing of customers of an ISP, it is not only interesting to count the number of packets in a flow. Instead, the total size of the flow in bytes is required. To solve this, [22] suggests to draw a random integer in the range $[0, \mathrm{MTU} - 1]$ for each packet. The packet is counted if and only if this random number is less than the size of the packet. This essantially leads to probabilistically counting "MTU equivalents" instead of packets. The flow size estimate is obtained by multiplying the estimate with the MTU.

This cannot be done in the same way for DPC, though: the random decision may lead to a packet being counted at one measurement point, but not at another. Duplicate-insensitive merging as discussed above is then not possible. However, following the spirit of hash-based sampling, we can substitute the random decision by a hash-based one, where instead of the random number a (uniform) hash value over the packet is used (with a hash function that is independent from $h_1$, $h_2$). A bit in $B$ is set if and only if the packet size exceeds this hash value modulo the MTU. Then, a packet will either lead to a bit in $B$ being set at all measurement points, or at none of them, avoiding the above discussed problem. We therefore obtain a variant of DPC that can perform distributed flow size measurements.

# 4 Evaluation

We evaluated DPC by simulating artificial network flows in a small and static topology on the one hand, and by applying it to traffic traces from a real university network on the other hand. This provides insights into the performance of the algorithm both under controlled conditions and when applied to real-world network traffic.

Let us first consider real-world network traffic. We use the freely available traces from FH Salzburg [16] that have been employed above in Sec. 3.2 already. These traces were captured at a mirror port of a 4 Mbps link using tcpdump and were anonymized with tcpdpriv. They contain TCP, UDP, and ICMP traffic without payload. These traces are from one single measurement point only; we can therefore use them to compare DPC's results to PMC, but we cannot assess DPC's distributed measurement capabilities. PMC is structurally similar to DPC, and has been shown to greatly outperform sampling-based traffic accounting and other bitfield-based techniques in [22]. Here, we use both PMC and DPC with a bit field size $l$ of 4 MBit, $m = 64$ rows per virtual matrix, and SHA1 for hashing (using some of the bits to construct $h_1$, others for $h_2$).




**(a)** PMC. **(b)** DPC.

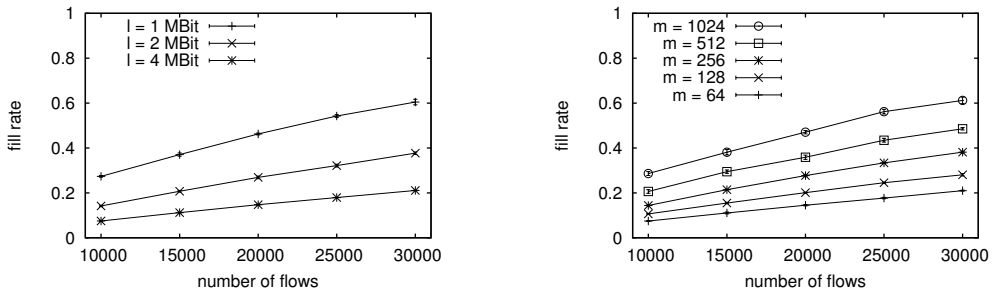
**Figure 3** PMC and DPC flow size estimation accuracy on the FH Salzburg traces.

In Figure 3, each data point represents one flow. The $x$ axes show the true flow sizes, the $y$ axes the corresponding estimates (note the log scales). For a perfect estimation, all points would therefore lie on the line of identity (here indicated as a dashed line). The subfigures show the results for DPC and PMC, respectively, if applied to the same network traffic.

Both algorithms produce accurate results at least for flows of non-negligible size; they do not differ in their accuracy. This becomes clear if we consider the standard error. When calculating the standard error in our setting, there is one caveat, though: large deviations between correct flow size and estimate are visible only on the left hand side of the plots. There, the absolute flow sizes are very small, so an estimation error of only a few packets already causes large relative deviations. This expected effect is due to the inherent inaccuracies of FM sketches for small estimates, which can be compensated only to a certain extent as discussed in Sec. 3.1. If not handled appropriately, these large relative deviations for very small flows dominate the standard error, and do not allow to appropriately judge the estimation quality for (practically much more relevant) flows of non-negligible size. We therefore consider the standard error for those flows which exceed a size of $m$ packets (as discussed in Sec 3.1, this is the point where the inaccuracies vanish). While the standard errors without this adjustment are 5.879 (for PMC) and 5.854 (for DPC) in this experiment, they drop to 0.1669 (for PMC) and 0.1673 (for DPC) when only flows of size $> m$ are taken into account. Whichever of the two variants is considered, it becomes clear that the accuracy differences between DPC and PMC observed here are vanishingly small and statistically not significant.

Since the central intention behind DPC is to allow for duplicate-insensitive measurements, we now turn towards a more detailed assessment of DPC in a distributed environment. To this end, we use simulated network traffic in a structurally simple, artificial network with multiple measurement points. This network consists of four routers A–D in a fully connected topology. Each of the four nodes runs one instance of DPC. We generated a varying number of flows in the network, with Pareto-distributed flow sizes (shape parameter $\alpha = 0.5$) to model the Internet traffic flow size distribution [24, 13]. We randomly picked one out of six possible routes for each flow, all with equal probability.[2] We then generated the packets for each flow (with random payload) and recorded them in all traversed DPC bit fields. Since the order and exact timing of packet arrivals at the router has no influence at all on

[2] The configured routes were: route 1: all packets A→D; route 2: all packets 3→4→2; route 3: 50 % of the packets 1→2, 50 % 1→3; route 4: 50 % of the packets 2→4, 50 % 3→4; route 5: 50 % of the packets 1→2, 50 % 3→4; route 6: all packets 1→2→3→4.




**(a)** Varying $l$ (fixed $m = 64$). **(b)** Varying $m$ (fixed $l = 4\,\text{MBit}$).

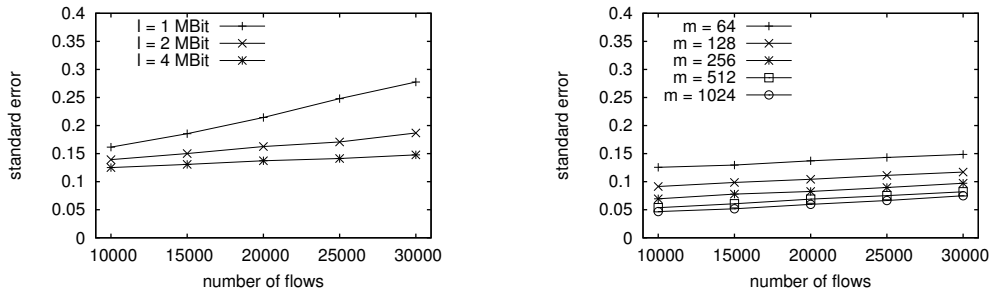
**Figure 4** Impact of the number of flows on the fill rate, for varying $l$ and $m$.

the resulting bit field, it is not necessary to employ a full-scale network simulator for these experiments. At each node, the result was a DPC bit field, containing information about all packets encountered by the respective router. The bit fields from all routers in the topology were subsequently combined for evaluation. Based on this combined bit field, the flow size for each flow was estimated.

The fill rate $p$ plays a central role for the estimation accuracy of DPC. The more bits in $B$ are set, the more false positive bits will occur, which in turn increases the estimation error. We are therefore interested in how fast $p$ grows with an increasing number of recorded flows, and performed a corresponding evaluation. Figure 4 shows the results for different values of $m$ and different bit field sizes $l$, as the number of flows increases. The figure includes 95 % confidence intervals; the error bars are so small that they are barely visible, though. As one would expect, using a larger bit field results in lower fill rates.

Figure 5 shows that lower fill rates directly translate into more accurate estimates, if other parameters remain unchanged. It shows the resulting standard errors for the same parameter combinations as above (for the reasons stated before, only flows of size $> m$ are taken into account; this figure does not include error bars, since the $y$ values already *are* a quantification of the error). These insights imply that it is possible to increase the accuracy by simply increasing the size of the available memory—or by decreasing the measurement interval duration, as a lower total number of sampled packets, too, is an obvious means to reduce the fill rate of the bit field. Note that the used bit field size has no influence on the computational effort for processing one packet, so the accuracy can be increased without incurring higher processing effort. This trait stands in sharp contrast to sampling-based techniques, where a higher fraction of packets need to be sampled to achieve higher accuracy.

The impact of the number of rows in the virtual PCSA matrix (parameter $m$) is slightly more complex. As $m$ inherently limits the estimation accuracy of the underlying FM sketches, it is, at some point, necessary to increase $m$ to achieve higher accuracies. Just like using a larger bit field, increasing $m$ does not increase the effort for processing a packet. As stated in Sec. 3.1, though, a higher value of $m$ results in a higher number of 1-bits per virtual matrix, and thus in turn increases the fill rate. This effect is visible in Figure 4b. It causes a higher number of false positive bits and thus reduces the accuracy benefits obtained by increasing $m$ in the first place. Despite this inherent tradeoff, finding a good value for $m$ is not too difficult: as Figures 4 and 5 show, the impact of this parameter is again just as one would expect, and all degradations are very graceful. Depending on the accuracy requirements, values of $m$ in the range between 32 and 1024 seem reasonable in practice, where for higher values it is advisable to use correspondingly larger bit fields (or shorter measurement intervals).




**(a)** Varying $l$ (fixed $m = 64$). **(b)** Varying $m$ (fixed $l = 4\,\text{MBit}$).

**Figure 5** Impact of the number of flows on the standard error, for varying $l$ and $m$.

## 5 Conclusion

In this paper, we have introduced Distributed Probabilistic Counting (DPC), an algorithm for obtaining flow size estimates in an entire network. With DPC, routers in the network can independently measure the network traffic during a measurement period. Their local observations—represented in an unstructured bit field—can then be merged in a duplicate-insensitive manner to yield statistics for the network in total. DPC is based on probabilistic counting techniques to perform this task with minimal effort per processed packet. We have shown that DPC yields accurate flow sizes estimates using both simulated network traffic patterns and real-world traffic traces.

## References


1 Baek-Young Choi and Supratik Bhattacharyya. Observations on Cisco sampled NetFlow. *SIGMETRICS Performance Evaluation Review*, 33(3):18–23, 2005.

2 Cisco Systems. NetFlow. http://www.cisco.com/web/go/netflow.

3 Cisco Systems. Sampled NetFlow. http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/12s_sanf.html.

4 Saar Cohen and Yossi Matias. Spectral bloom filters. In *SIGMOD '03: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 241–252, June 2003.

5 Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55:29–38, 2004.

6 Xenofontas Dimitropoulos, Paul Hurley, and Andreas Kind. Probabilistic lossy counting: An efficient algorithm for finding heavy hitters. *SIGCOMM Computer Communications Review*, 38(1), 2008.

7 Ruan Donghua, Lin Chuang, Chen Zhen, Ni Jia, and Peter D. Ungsunan. Handling high speed traffic measurement using network processors. In *ICCT '06: Proceedings of the International Conference on Communication Technology*, pages 1–5, November 2006.

8 Nick Duffield, Carsten Lund, and Mikkel Thorup. Charging from sampled network usage. In *IMW '01*, pages 245–256.

9 Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems*, 21(3):270–313, 2003.

10 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, October 1985.

**11** Christian Henke, Carsten Schmoll, and Tanja Zseby. Empirical evaluation of hash functions for multipoint measurements. *SIGCOMM Computer Comm. Review*, 38(3):39–50, 2008.

**12** Christian Henke, Carsten Schmoll, and Tanja Zseby. Evaluation of header field entropy for hash-based packet selection. In *PAM '08: Proceedings of the Passive and Active Measurement Conference*, pages 82–91, April 2008.

**13** Félix Hernández-Campos, J. S. Marron, Gennady Samorodnitsky, and F. D. Smith. Variable heavy tails in Internet traffic. *Elsevier Performance Evaluation*, 58(2+3):261–261, 2004.

**14** Nicolas Hohn and Darryl Veitch. Inverting sampled traffic. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, pages 222–233, October 2003.

**15** Chengchen Hu, Sheng Wang, Jia Tian, Bin Liu, Yu Cheng, and Yan Chen. Accurate and efficient traffic monitoring using adaptive non-linear sampling method. In *INFOCOM '08: Proceedings of the 27th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 26–30, April 2008.

**16** Information Society Technologies (IST) Programme of the European Union. Traffic measurement database MOME. http://www.ist-mome.org/, March 2010.

**17** InMon Corp. sFlow accuracy & billing. http://www.inmon.com/pdf/sFlowBilling.pdf.

**18** InMon Corp. sFlow version 5. http://sflow.org/sflow_version_5.txt.

**19** Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems*, 28(1):51–55, 2003.

**20** Abhishek Kumar and Jun Xu. Sketch guided sampling – using on-line estimates of flow size for adaptive data collection. In *INFOCOM '06: Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies*, April 2006.

**21** Abishek Kumar, Jun Xu, and Jia Wang. Space-code bloom filter for efficient per-flow traffic measurement. *IEEE Journal on Selected Areas of Communications*, 24(12):2327–2339, December 2006.

**22** Peter Lieven and Björn Scheuermann. High-speed per-flow traffic measurement with probabilistic multiplicity counting. In *INFOCOM '10: Proceedings of the 29th Annual Joint Conference of the IEEE Computer and Communications Societies*, March 2010.

**23** Yi Lu, Andrea Montanari, Balaji Prabhakar, Sarang Dharmapurikar, and Abdul Kabbani. Counter braids: A novel counter architecture for per-flow measurement. In *SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 121–132, June 2008.

**24** Bruce A. Mah. An empirical model of HTTP network traffic. In *INFOCOM '97: Proceedings of the 16th Annual Joint Conference of the IEEE Computer and Communications Societies*, April 1997.

**25** Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Data Bases*, pages 346–357, August 2002.

**26** Sriram Ramabhadran and George Varghese. Efficient implementation of a statistics counter architecture. In *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, June 2003.

**27** Devavrat Shah, Sundar Iyer, Balaji Prabhakar, and Nick McKeown. Analysis of a statistics counter architecture. In *HOTI '01: Proceedings of the 9th Symposium on High Performance Interconnects*, August 2001.

**28** Kyu young Whang, Brad T. Vander-Zanden, and Howard M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems*, 15(2):208–229, June 1990.

**29** Tanja Zseby, Maurizio Molina, Nick Duffield, Saverio Niccolini, and Frederic Raspall. Sampling and filtering techniques for IP packet selection. RFC 5475, March 2009.