Estimators Also Need Shared Values to Grow Together

Erez Tsidon
Technion, Qualcomm
erezts@tx.techion.ac.il

Iddo Hanniel
Technion
ihanniel@techion.ac.il

Isaac Keslassy
Technion
isaac@ee.techion.ac.il

***Abstract*—Network management applications require large numbers of counters in order to collect traffic characteristics for each network flow. However, these counters often barely fit into on-chip SRAM memories. Past papers have proposed using counter estimators instead, thus trading off counter precision for a lower number of bits. But these estimators do not achieve optimal estimation error, and cannot always scale to arbitrary counter values.**

In this paper, we introduce the CEDAR algorithm for decoupling the counter estimators from their estimation values, which are quantized into estimation levels and shared among many estimators. These decoupled and shared estimation values enable us to easily adjust them without needing to go through all the counters. We demonstrate how our CEDAR scheme achieves the min-max relative error, i.e., can guarantee the best possible relative error over the entire counter scale. We also explain how to use dynamic adaptive estimation values in order to support counter up-scaling and adjust the estimation error depending on the current maximal counter.

Finally we implement CEDAR on FPGA and explain how it can run at line rate. We further analyze its performance and size requirements.

I. Introduction

A. Background

In many high-end networking devices, the amount of *on-chip memory needed by counters* keeps growing and starts reaching its limits. This is because the needed number of counters in networking devices keeps increasing with the number of flows. It also keeps growing with the number of measurement-based applications, since each application can require different types of counters per flow. In addition, not only the number of counters, but also the counter sizes keep increasing with the line rates, since line rates directly determine the maximum counter value.

This memory limit on the number of counters can directly affect many algorithms for networking devices, since these often rely on the assumption that packet counts and flow rates are easily available. For instance, the ability to easily measure and compare flows lies at the basis of many fairness, scheduling, flow control, admission control, load-balancing, shaping and policing algorithms.

As a simplified example, consider a current high-speed multi-stage router [1]–[3], e.g. with 256 input and output ports and 4 priority classes. Then a middle-stage element sees $256^2 \cdot 4 \approx 262,000$ unicast router flows. Keeping only three counters per flow (e.g., the number of packets of the flow in the switch element, and its current arrival and departure rates) with 16 bits per counter would yield a needed memory size of 48 bits per flow, i.e. a total of 12 Mb $= 1.5$ MB. After implementing all the other functions in the middle-stage element, such as packet switching and scheduling control, this can fit into available commodity on-chip SRAM memory (barely). However, in next-generation routers with $1,024$ subports, the needed memory would scale to 24 MB—well beyond commodity SRAM sizes [4].

Per-flow counters are needed by many types of network management applications. For instance, flow control mechanisms such as AF-QCN [5], network traffic anomaly detection algorithms [6] and congestion control applications such as FRED [7] require per-flow counters for measuring flow capacities and flow rates.

In this paper, we consider networking device applications for which counter *estimates* are sufficient, even if the estimates have small errors, as long as all those estimates can fit on commodity on-chip SRAM memory and can be accessed in real time. For instance, a flow control mechanism similar to AF-QCN [5] could use rate estimates for each flow to obtain quick convergence to fairness. Taken to the extreme, these algorithms may in fact only depend on the order of magnitude of the estimated flow rate. For example, a differentiated service algorithm may provide a different service to flows with estimated rates of about 10Kbps, 100Kbps or 1Mbps.

This paper is about enabling many per-flow counters in a scalable way, both for packet counts and packet rates. We do so by replacing *exact counters* with *shared counter estimates*. These estimates will enable us to trade off a decreased needed amount of memory with an increased error probability in the counter estimation. In addition, using the estimator sharing, we provide ways to smoothly *change the counter scale*, and therefore adapt the estimation error to the traffic size in a dynamic way. This is particularly useful for estimates of window-based average *rates*, since they rely on counters that keep increasing and therefore need to adapt over several orders of magnitude.

B. Alternative Solutions and Related Work

The following alternative solutions attempt to use less bits per counter.

A first approach is to *keep less counters*. A straightforward solution adopted by high-end device designers against the

memory size problem is to avoid keeping an on-chip counter for each flow, and instead keep a counter *per pre-determined logical flow aggregate*. For instance, keep a counter for all packets with a certain priority, or for all packets coming from a given input port. However, this might prevent the use of several algorithms. For instance, upon congestion, this prevents the use of fairness algorithms based on the flow rates, such as the flow control and differentiated-service algorithms previously mentioned. Another way to keep less counters is to only keep counters for heavy hitters [8], [9]. However, while this can help in many applications, our goal is to provide a counter estimate for *all* flows.

A second approach is to only *keep counter estimates for active flows* using a structure based on a *CBF* (Counting Bloom Filter) [10], [11]. For instance, [12], [13] introduce the CBF-based Counter Braids structure to achieve near-entropy compression. However, the Counter Braids algorithms are not designed for *online* counter estimation, which is assumed in this paper. More generally, CBF-based structures perform most efficiently when most counter values are zero, which is not necessarily the case. In practice, when most counter values are zero, CBF can be combined with our estimators to provide even better error rates for the same amount of memory: instead of estimating flow counters, our estimators would simply *estimate the CBF counters*.

Likewise, a related algorithm, BRICK [14], restricts the counter sum in order to compress counters. This requirement assumes a flow distribution that keeps most of the counter values at zero. In this paper we attempt to avoid such an assumption.

A third approach is to use *DRAM*, either by combining small on-chip SRAM counters with large off-chip DRAM counters [15]–[18], or by only using off-chip DRAM counters with potentially a small queue in SRAM [19]–[21]. This DRAM-based approach provides exact counters instead of counter estimates. It can also obtain cheaper solutions by relying on DRAM rather than SRAM. Unfortunately, because it relies on off-chip DRAMs with longer access latencies, this approach cannot satisfy arbitrary real-time counter reads at line rate. For instance, it may need to restrict the pattern of the counter queries and/or add a lookahead queue of unpredictable length. In addition, this approach forces the designer to deal with the buffer management policy for the off-line DRAM in addition to the SRAM that is used in other applications, which may be a barrier to its use.

A fourth approach is to use *counter approximations* instead of real values, as in the Approximate-Counting [22], SAC [23] and DISCO [24] algorithms. In these methods each counter is replaced with an estimator that requires significantly less bits (e.g. 12 bits vs. 32). However, each method restricts its estimation value distribution to predefined values. For example, SAC restricts its estimation values to be of the form $a2^b$. We will show that these estimation values are not necessarily optimal, and provide the optimal values. In addition, some of these methods (e.g. DISCO) do not support up-scaling procedures, thus the maximal possible estimated value must be known ahead of time, resulting in a large estimation error over the entire counter scale. We propose a method to adjust the estimation error to the current maximal counter value.

C. Our Contributions

In this paper we introduce a counter architecture, called CEDAR (Counter Estimation Decoupling for Approximate Rates), which decouples counter values from flow symbols. The goal of CEDAR is to provide *real-time scalable counter estimates*. CEDAR gathers estimation values into one array with a relatively small number of entries. Flow symbols are used as pointers into this estimation array. Since the number of estimation array entries is low compared to the number of flow entries (e.g. hundreds vs. millions), we show that the additional required memory is negligible.

Using this estimator array, we have the freedom to choose any estimation values. We exploit this advantage by determining the exact estimators that minimize the maximal relative error over the entire counter scale. That is, for any counter value, CEDAR can guarantee that the relative error does not exceed a predetermined guaranteed value, which is proved to be minimal. This estimation is particularly useful for applications that require fast reading and writing of flow counters with a predetermined relative estimation error that is independent of the counter value.

Then, to obtain counter estimators without a predetermined maximum, we introduce an online up-scaling scheme. This scheme is useful when there is no advance knowledge about the flow distribution, and therefore the maximal counter value cannot be predicted. Our up-scaling algorithm adjusts the relative error of the entire counter scale to the current maximal counter value. In this way the estimation error at any time is uniquely derived from the stream behavior.

In order to evaluate our algorithms, we use Internet traces and compare the estimation error of CEDAR versus SAC [23] and DISCO [24]. We show how our algorithm can more efficiently estimate counter values given the same amount of memory. We also show how our up-scaling mechanism keeps the max-relative-error adjusted to the counter scale.

Finally, we demonstrate the CEDAR implementation on FPGA and analyze its performance and size requirements.

II. The CEDAR Architecture

A. Architecture

As shown in Figure 1, the CEDAR architecture is based on two arrays. First, a flow array F of size N that contains pointers into the estimator array A for each of the flows F_j, $0 \leq j \leq N-1$. Second, an estimation array A of size L that contains estimator values A_i, $0 \leq i \leq L-1$.

For instance, in Figure 1, the first flow F_0 points to A_1 (the pointer to index 1 is denoted $p(1)$), and the value of estimator A_1 is 1.2. Therefore the estimator value for flow F_0 is 1.2.

Assuming that L is a power of 2, the number of bits needed for each of the pointers in F is simply $\log_2 L$. We also assume that each estimator value in A uses q bits.

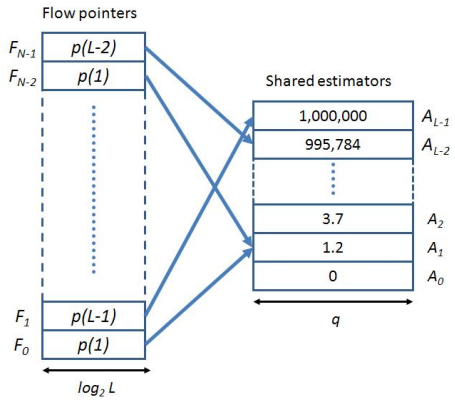


Fig. 1. CEDAR structure. The flow pointers on the left point to shared estimators on the right. For example, the estimator for flow F_0 is $A_1 = 1.2$.

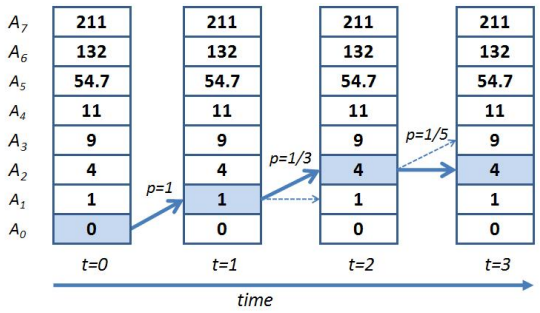


Fig. 2. Update mechanism of the CEDAR algorithm upon consecutive packet arrivals for a given flow. The flow counter estimate is initialized at $A_0 = 0$. After the first arrival, the difference between A_0 and A_1 is 1, therefore the update step is deterministic with probability $p = 1/1 = 1$ and the new estimate is $A_1 = 1$. After the second arrival, $A_2 - A_1 = 3$, therefore the flow pointer is incremented with probability $1/3$ to $A_2 = 4$. At the third arrival, $A_3 - A_2 = 5$, therefore the increment probability is $1/5$. Here it is not incremented.

B. Conversion to the CEDAR Structure

We show now that *any counter estimation method* that represents each estimate independently using a fixed number of bits can be converted easily into a CEDAR structure with asymptotically negligible effect on the estimation error. This applies for instance to the SAC [23] and DISCO [24] counter estimation methods.

Theorem 1: Any counter estimation method using a fixed number of bits per counter estimate can be converted into the CEDAR structure with the same estimation error, and an asymptotically negligible overhead as $N \to \infty$.

Proof: Consider an estimation method that uses q bits per counter for counter estimation. Thus, it can have a maximum of 2^q distinct symbols. If we build from these symbols an estimation array of size $L = 2^q$, then we can use the counter symbols as pointers into this estimator array and run the original algorithm without any change. Therefore we get a CEDAR structure that keeps the same estimation error. In addition the structure overhead is the additional contribution of the estimator array, i.e. $\frac{q \cdot 2^q}{N \cdot q} = \frac{2^q}{N} \xrightarrow{N \to \infty} 0$. ■

C. Performance Measures

Throughout this paper we only consider unit increments that represent packet arrivals. Note that the estimation error with variable increments larger than one (e.g. when measuring bytes) is actually lower given the same scale, because it enables more precise scale jumps, and therefore using unit increments results in a worst-case analysis.

We would like to evaluate our counter estimates using their *relative error* rather than their absolute error. That is, a difference of 10 between the real counter and our estimate may account as a huge error if the real value is 20, and as a negligible error if the real value is 100000. Thus, we would like to normalize the error by the counter value.

Formally, we denote by $T(l)$ the random variable that represents the required amount of traffic for a certain flow to point to estimator A_l, i.e. its *hitting time*. As in [23], [24], assuming our algorithms yield unbiased estimators and assuming $\mathbb{E}[T(l)] = A_l$, we will define the *relative error* (or coefficient of variation) as

$$\Delta[T(l)] = \sqrt{\frac{\mathrm{Var}[T(l)]}{(\mathbb{E}[T(l)])^2}} = \frac{\sigma[T(l)]}{\mathbb{E}[T(l)]}$$

D. Static CEDAR Algorithm

We start by introducing the static version of CEDAR, which uses a static and pre-determined estimator array A. Later, in Section IV, we will introduce the dynamic version of CEDAR that enables scaling the estimator array.

In the static CEDAR algorithm, we make two simplifying assumptions on the fixed CEDAR structure. First, we assume that the estimator values A_i are defined in ascending order, with $A_0 = 0$. We denote the differences between two successive estimators as $D_i = A_{i+1} - A_i$, where $0 \le i \le L-2$, and further assume that $D_i \ge 1$ for any i. For instance, in Figure 1, $D_0 = 1.2 - 0 = 1.2$ and $D_1 = 3.7 - 1.2 = 2.5$.

The static CEDAR algorithm works as follows. At the start, all flow counters point to $A_0 = 0$. Then, at each arrival of a packet from flow j, which corresponds to a unit increment of its real counter value, the flow's pointer F_j is incremented by one with probability $\frac{1}{D_{F_j}} = \frac{1}{A_{F_j+1} - A_{F_j}}$. In other words, if the flow pointer first points to estimated value A_i (where $i = F_j$), then it either starts pointing to A_{i+1} with probability $1/D_{F_j}$, or continues pointing to A_i with probability $1 - 1/D_{F_j}$. Note that since $D_i \ge 1$ for any i, these probabilities are well defined.

Figure 2 further illustrates an example of such update sequence in the static CEDAR algorithm.

E. Unbiasedness and Traffic Expectation

Given our definition of the static CEDAR algorithm, we first show that the expectation of random variable $T(l)$ equals the estimated value A_l. This means that the expected hitting time of A_l is exactly A_l, i.e. the expected number of arrivals needed to reach some estimated value is exactly this estimated value.

Theorem 2: The hitting time $T(l)$ of estimator A_l satisfies the following property: $\mathbb{E}[T(l)] = A_l$

Proof: We can divide $T(l)$ into l i.i.d geometric random variables $G(p_i)$ with parameter $p_i = \frac{1}{D_i}, i = [0, ..., l-1]$, therefore

$$\mathbb{E}[T(l)] = \mathbb{E}\left[\sum_{i=0}^{l-1} G(\frac{1}{D_i})\right] = \sum_{i=0}^{l-1} \mathbb{E}\left[G(\frac{1}{D_i})\right] = \sum_{i=0}^{l-1} D_i = A_l$$

■

In addition, regardless of the exact values used within estimation array A, the CEDAR estimation is unbiased, as long as we assume that the estimation scale is unbounded (or that its maximum is never reached). The unbiasedness proof of CEDAR is similar to the proof of Theorem 1 in DISCO [24], therefore we do not repeat it.

Property 1: CEDAR estimators are unbiased.

III. Relative Error Minimization

A. Min-Max Relative Error

A network device designer will want to obtain a guarantee that *the counter estimates have good performance no matter how large the counters are*. For instance, such a guarantee may be that the relative error is below 5% over the whole scale of counters.

Unlike former approaches that restrict the counter estimators to specific values, the flexibility of the CEDAR structure enables us to analytically determine and implement the *optimal estimation array* that achieves such a guarantee. In the remainder of this section, we first determine the values of this optimal estimation array in Theorem 3, then prove that it is indeed optimal in Theorem 4.

More specifically, given an estimation array size L and a fixed maximal estimator value A_{L-1}, we want to determine the set of estimator values $\{A_0, ..., A_{L-1}\}$ that minimizes the maximum relative error over all estimators, i.e. achieves $\min_A \max_l \Delta[T(l)]$. In other words, this set should minimize the guaranteed error δ such that for all l, $\Delta[T(l)] \leq \delta$.

As we saw in Theorem 2, $T(l)$ is a sum of l i.i.d geometric random variables, therefore

$$\Delta[T(l)] = \sqrt{\frac{\mathrm{Var}[T(l)]}{(E[T(l)])^2}} = \sqrt{\frac{\sum_{i=0}^{l-1} \frac{1-1/D_i}{1/D_i^2}}{\left(\sum_{i=0}^{l-1} D_i\right)^2}}$$

and we can rewrite the condition $\Delta[T(l)] \leq \delta$ for all l as

$$\forall l : H_l(D_0, ..., D_l) \stackrel{\triangle}{=} \sum_{i=0}^{l} \left(D_i^2 - D_i\right) - \delta^2 \left(\sum_{i=0}^{l} D_i\right)^2 \leq 0 \tag{1}$$

We later show in Theorem 4 that in order to achieve such min-max relative error, all estimators must reach an equal relative error δ, that is,

$$\forall l : \Delta[T(l)] = \delta,$$

which we can rewrite as $\forall l : H_l(D_0, ..., D_l) = 0$ (also denoted as $H_l = 0$ for simplicity).

We now first prove in the following theorem that we achieve such an *equal-relative-error* estimation array iff the estimation values are given by the following recursive equation,

$$D_l = \frac{1 + 2\delta^2 \sum_{i=0}^{l-1} D_i}{1 - \delta^2}, \tag{2}$$

with $D_0 = \frac{1}{1-\delta^2}$, before proving in Theorem 4 that we indeed need to achieve this equal-relative-error to provide an optimal guarantee.

Theorem 3: Given a target relative error δ, and the $L-1$ constraints: $\forall l \in [0, L-2], H_l(D_0, ..., D_l) = 0$, then the estimation value increments D_l are necessarily given by the recursive function in Equation (2).

Proof: First, we highlight the recursive nature of the H_l functions. Extracting D_l from H_l yields:

$$\begin{aligned} H_l =& \sum_{i=0}^{l-1} D_i^2 + D_l^2 - \sum_{i=0}^{l-1} D_i - D_l \\ & - \left((\sum_{i=0}^{l-1} D_i)^2 + 2D_l \sum_{i=0}^{l-1} D_i + D_l^2\right)\delta^2 \\ =& (1-\delta^2)D_l^2 - (2\delta^2 \sum_{i=0}^{l-1} D_i + 1)D_l \\ & + \underbrace{\sum_{i=0}^{l-1} D_i^2 - \sum_{i=0}^{l-1} D_i - (\sum_{i=0}^{l-1} D_i^2)^2\delta^2}_{H_{l-1}} = 0 \end{aligned} \tag{3}$$

Assigning $H_{l-1} = 0$ immediately gives Equation (2). ■

The following theorem proves that minimizing the maximal relative error δ for a given A_{L-1} is equivalent to maximizing A_{L-1} for a given maximal relative error δ. More significantly, it demonstrates that both are equivalent to equalizing all relative errors, therefore yielding the recursive equation obtained above in (Theorem 3).

Theorem 4: Given the $L-1$ constraints: $\forall l \in [0, L-2], H_l \leq 0$, the following claims are equivalent:

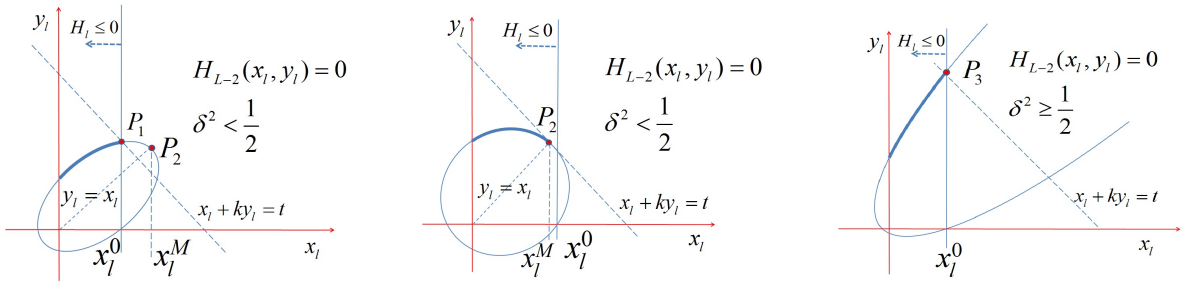
(i) *Maximizing* A_{L-1}: $A_{L-1} = \sum_{i=0}^{L-2} D_i$ is maximal.
(ii) *Equalizing all relative errors*: $\forall l \in [0, L-2], H_l = 0$.
(iii) *Minimizing* δ: δ is the minimal guaranteed relative error.

Proof: We start by proving that (i)⇒(ii). Assuming $\sum_{i=0}^{L-2} D_i$ is maximal, the $(L-1)^{th}$ equality, $H_{L-2} = 0$, is given by the following lemma:

Lemma 1: Assume that $\forall l \in [0, L-2], H_l \leq 0$. If $A_{L-1} = \sum_{i=0}^{L-2} D_i$ is maximal then $H_{L-2} = 0$

Proof Outline: We assume by contradiction that there is a set $\{D_0, ..., D_{L-2}\}$ that satisfies all $L-1$ constraints and achieves maximal $\sum_{i=0}^{L-2} D_i$ but $H_{L-2} < 0$. In [25], we show that under these conditions we can increase D_{L-2} by some small ϵ while still satisfying the constraints, and therefore the assumed sum is not maximal. ■

We now prove that $H_{L-2} = 0 \Rightarrow H_{L-3} = 0$, and more generally, by induction, $H_l = 0 \Rightarrow H_{l-1} = 0$, i.e. $H_l = 0$ for all l. We assume by contradiction that $H_{L-3} < 0$. We begin



(a) Ellipse case, option 1: intersects with $H_l \le 0$ constraint to the left of the tangency point P_2. The maximum for $x_l + ky_l$ is attained at the intersection point P_1.

(b) Ellipse case, option 2: intersects with $H_l \le 0$ constraint to the right of the tangency point P_2. The maximum for $x_l + ky_l$ is therefore attained at P_2.

(c) Parabola / Hyperbola case: always intersect with $H_l \le 0$ constraint at the upper branch of the curve. The maximum for $x_l + ky_l$ is attained at the intersection point P_3.

Fig. 3. Geometric view of Theorem 4 based on quadratic curves: the quadratic curves represent the $H_{L-2} = 0$ constraint (according to the δ value) while the vertical line $x_l = x_l^0$ represents the $H_l \le 0$ constraint.

by assigning $l = L-3$ and follow the following steps in the proof:

1) We show that if $H_l < 0$ then the only way to achieve maximal $\sum_{i=0}^{L-2} D_i$ is when $H_{l-1} < 0$ and $D_l = D_{l+1}$.
2) For $l > 0$, we set $l \leftarrow l-1$ and return to 1)
3) When $l = 0$ we get $\forall i, j : D_i = D_j$. We will refute this result by comparing it to the values we got in Equation (2), which achieve a larger $\sum_{i=0}^{L-2} D_i$.

We wish to provide an intuition for the first recursion cycle where $l = L-3$, before outlining the proof for the remaining recursion cycles. Assuming $\sum_{i=0}^{L-2} D_i$ is maximal, Lemma 1 gives us the $(L-1)^{th}$ equality: $H_{L-2} = 0$. In a similar manner to the way we extracted D_{l-1} in Equation (3), we extract both D_{L-2} and D_{L-3} from H_{L-2}, resulting in:

$$\begin{aligned} H_{L-2} =& (1-\delta^2)D_{L-3}^2 - 2\delta^2 D_{L-2}D_{L-3} \\ &+ (1-\delta^2)D_{L-2}^2 - \left(2\delta^2 \sum_{i=0}^{L-4} D_i + 1\right) D_{L-3} \\ &- \left(2\delta^2 \sum_{i=0}^{L-4} D_i + 1\right) D_{L-2} + H_{L-4} = 0 \end{aligned} \quad (4)$$

Interestingly, we can use a *geometric approach* to this optimization. Let us set $l = L-3$ and denote the following symbols: $x_l = D_l; y_l = D_{l+1}; A = 1-\delta^2; B = -2\delta^2; C_1 = -(1 + 2\delta^2 \sum_{i=0}^{l-1} D_i)$. We can rewrite Equation (4) as:

$$Ax_l^2 + Bx_l y_l + Ay_l^2 + C_1 x_l + C_1 y_l + H_{l-1} = 0 \quad (5)$$

where $A > 0, B < 0, C_1 < 0$, and $H_{l-1} \le 0$. *This equation defines a rotated quadratic curve in the (x_l, y_l) plane:* an *ellipse* when $\delta < \sqrt{0.5}$, a *parabola* when $\delta = \sqrt{0.5}$, and a *hyperbola* when $\delta > \sqrt{0.5}$ (see Figure 3)[1]. We now consider the constraint $H_l \le 0$ (i.e., $H_{L-3} \le 0$) in the (x_l, y_l) plane using our new notations. Note that from Equation (3), we can see that H_{L-3} can be computed from H_{L-2} by assigning $D_{L-2} = 0$. In our new notation this means setting $y_l = 0$. We get $Ax_l^2 + C_1 x_l + H_{l-1} \le 0$, implying that

$$x_l \le x_l^0 = \frac{-C_1 + \sqrt{C_1^2 - 4AH_{l-1}}}{2A} \quad (6)$$

That is, as shown in Figure 3, the solution range is on the quadratic curve from the left side of the vertical line $x_l = x_l^0$. In the hyperbola/parabola case the vertical line always intersects with the upper branch of the quadratic curve at P_3 since the curve is unbounded in the x_l axis. Therefore, $x_l + ky_l = t$ gets its maximal value at the intersection point P_3 (in the specific case we are analyzing, of the first recursion cycle, $k = 1$).

In the ellipse case there are two options. Let $x_l = x_l^M$ be the vertical that passes through P_2, the point of the ellipse that is tangent to the line $x_l + ky_l = t$ (see Figure 3). In option 1, $x_l^0 < x_l^M$ and the maximal value is attained at P_1, whereas in option 2, $x_l^M < x_l^0$ and the maximal value is attained at the tangency point P_2. In [25] we prove that P_2 is also the intersection point between the ellipse and $y_l = x_l$.

The outline of the proof remainder goes as follows. We treat $(D_0, ..., D_{L-4})$ as constants in the geometric plane. Thus, according to our assumption that $\sum_{i=0}^{L-2} D_i$ is maximal, $D_{L-3} + D_{L-2} = x_l + y_l$ should be maximal too in each one of the three geometric options. We will show that due to our contradiction assumption that $H_{L-3} < 0$ the only valid choice is option 2 of the ellipse case since in the other options the maximum for $D_{L-3} + D_{L-2}$ is attained where $H_{L-3} = 0$. In this option the maximum for $D_{L-3} + D_{L-2}$ hits the tangency point P_2 at which $D_{L-3} = D_{L-2}$. We will plug this result back into Equation (4), extract D_{L-4} from the summation and build again the same quadratic curves. This time we treat $(D_0, ..., D_{L-5})$ as constants and maximize $D_{L-4} + D_{L-3} + D_{L-2} = D_{L-4} + 2 \cdot D_{L-3}$, i.e. maximize $x_l + 2y_l$ where $l = L-4$. In the kth step of the recursion, we treat $(D_0, ..., D_{L-k-3})$ as constants and the maximization will be on $x_l + ky_l$ where $l = L-k-2$. Note that this general maximization is the step that we develop in Figure 3.

[1]On how to classify a quadratic curve, see for example [26] or any basic textbook on curves and surfaces such as [27].

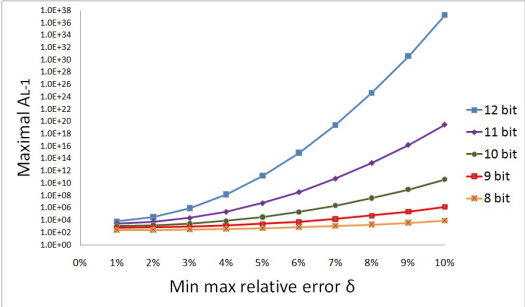


Fig. 4. The maximal estimator value A_{L-1}, as a function of the minimal max-relative-error δ, for several bit widths $\log_2(L)$.

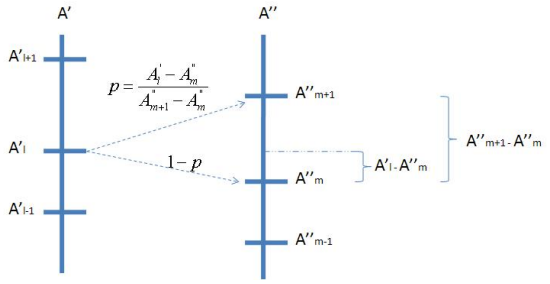


Fig. 5. CEDAR scaling. A' is a CEDAR estimation array that matches a relative error δ_0. A'' is a CEDAR array that matches a relative error $\delta_0+\delta_{\text{step}}$, therefore it has a higher maximal value. Each flow pointer that points to an estimator in A' is converted to one of the two closest estimators in A'', according to probability p.

Proceeding with the recursion steps till $l = 0$ will finally yield $\forall i, j : D_i = D_j$, which will be used to refute the contradiction assumption. Due to space limitations, we present the full proof details in [25].

We prove now that (ii)⇒(i). Assuming for all l $H_l(D_0, ..., D_l) = 0$, we get the recursive formula from Equation (2). This formula defines $L-1$ linear equations that can be represented as a triangular matrix, i.e. they are linearly independent. This means there exists a solution and it is unique.

Now we show that (iii) $\iff$ (i). Denote by $M = A_{L-1}$ the maximal value of $\sum_{i=0}^{L-2} D_i$ for a given δ. Equation (2) defines a function $M(\delta) = \sum_{i=0}^{L-2} D_i(\delta)$. It is easy to see from Equation (2) that $M(\delta)$ is a strictly increasing function of δ (since the D_ls increase as δ increases). Therefore, there exists an inverse function $\delta(M)$, which for any given value A_{L-1}, returns δ such that $M(\delta) = A_{L-1}$. This is the minimal δ since a smaller δ will not satisfy the relative error constraints, whereas a larger δ will not be minimal. ■

B. Capacity Region of Static CEDAR

Figure 4 illustrates the maximal possible guaranteed region for CEDAR, as provided by the min-max optimization. It plots the maximal estimator value as a function of the min-max relative-error, given several possible bit widths, as indicated in Equation (2). For instance, given $\log_2 L = 10$ bits, i.e. $L = 2^{10} = 1024$ estimators, we can achieve a min-max relative error of $\delta = 5\%$ using a maximal estimator value of $A_{L-1} = 32 \cdot 10^3$, and an error of $\delta = 10\%$ using $A_{L-1} = 35 \cdot 10^9$.

Alternatively, each plot also defines the lowest possible relative error guarantee given a counter scale, as proved in Theorem 4. For instance, given $A_{L-1} = 32 \cdot 10^3$, it is not possible to provide a better guarantee than $\delta = 5\%$.

Figure 4 also shows how an increase in the number of bits, i.e. in the available memory resources, leads to an increase in the maximal counter value and/or a decrease in the guaranteed relative error.

IV. Dynamic up-scale

A. Motivation

We now introduce the full CEDAR algorithm that can *dynamically adjust* to the counter scale. This algorithm is used to support an unlimited counter value, yet also provide a good relative error at any given time. The strength of our algorithm is the fact that we control the up-scaling by adjusting the relative error rather than changing the maximal estimation value. In this way the error of the entire counter scale is always under control and can be fine-tuned by the user.

In practice, we start with the best estimation array that matches an initial maximal relative error δ_0. This estimation array is unique, as we proved in Theorem 4. As more and more packets are streaming in and the maximal flow counter is going to exceed our maximal estimation value, we set a new relative error of $\delta_{\text{new}} = \delta_0 + \delta_{\text{step}}$ and build a new matching estimation array. We further proceed in the same way for the next iterations.

B. Algorithm

In order to support up-scaling we use two estimation arrays A' and A'' in a simple *ping-pong* way. We initialize A' with equal-error estimators that match an initial error of δ_0 according to Equation (2), and likewise initialize A'' with estimators that match $\delta_0 + \delta_{\text{step}}$. Therefore A'' also has an higher maximal estimator value than A'.

We start by using the static CEDAR algorithm with the A' array. Once the maximal flow pointer is getting close to the end of A' array such that $\max_j(F_j) \geq i_{thr}$, where i_{thr} is an arbitrary threshold value, we perform the following up-scale procedure: For j=0 to $N-1$ do

1) Set $l = F_j$,
2) Find m such that $A''_m \leq A'_l < A''_{m+1}$ (e.g. using binary search within A''),
3) Set $p = \frac{A'_l - A''_m}{A''_{m+1} - A''_m}$,
4) Set $F_j = m+1$ with probability p and $F_j = m$ with probability $1-p$.

Figure 5 depicts a single step of the up-scale procedure.

We run the up-scaling procedure in parallel with the real-time counter updates, i.e. *without stopping* the CEDAR update algorithm. We use the current-flow index f as follows: assuming a new packet arrival for flow F_i, if $i < f$ we use the new estimation array A'', and otherwise use A' (since the current flow f has not been up-scaled yet).

When the up-scaling procedure ends, we prepare A' for the next up-scaling event by filling it with new estimation values

that match a new error of additional δ_{step}, i.e. after a new up-scaling event the relative error is $\delta_0 + 2 \cdot \delta_{\text{step}}$ and after n events it is $\delta_0 + n \cdot \delta_{\text{step}}$.

C. CEDAR Up-Scale Unbiasedness

In the following theorem we show the unbiasedness of CEDAR up-scale algorithm. Since we know from the unbiasedness of static CEDAR that estimation updates are unbiased as long as the scale is maintained, the proof essentially shows that up-scaling does not introduce bias. As for static CEDAR, the result assumes that either the estimator scale is unbounded, or up-scaling is done such that the maximum estimator value is never reached.

Theorem 5: The CEDAR up-scaled estimators are unbiased.

Proof: In the up-scaling phase, let C denote the real number of packets for the flow, $\hat{C}'$ its estimation under the old scale, and $\hat{C}''$ its estimation under the new scale. Then we need to show that $\mathbb{E}\left(\hat{C}''\right) = \mathbb{E}\left(\hat{C}'\right) = C$. Using the law of iterated expectation $\mathbb{E}\left[\hat{C}''\right] = \mathbb{E}\left(\hat{C}''|\hat{C}'\right)$, it is therefore sufficient to show that $\mathbb{E}\left(\hat{C}''|\hat{C}'\right) = \hat{C}'$.

Consider now a random path of the counter estimator. As illustrated in Figure 5, denote by A'_l the random variable that represents the estimation value before the up-scaling. Also define A''_m on the new scale such that $A'_l \in [A''_m, A''_{m+1})$. Then we just need to show that for any such A'_l, $\mathbb{E}\left(\hat{C}''|\hat{C}' = A'_l\right) = A'_l$. Denote a Bernoulli random variable with parameter p as $B(p)$. Then we obtain:

$$\begin{aligned}
\mathbb{E}\left(\hat{C}''|\hat{C}' = A'_l\right) &= \mathbb{E}\Big(A''_m + (A''_{m+1} - A''_m) \\
&\qquad \cdot B\left(\frac{A'_l - A''_m}{A''_{m+1} - A''_m}\right)\Big) \\
&= A''_m + (A''_{m+1} - A''_m) \cdot \frac{A'_l - A''_m}{A''_{m+1} - A''_m} \\
&= A'_l
\end{aligned}$$

■

V. CEDAR Evaluation

We evaluate the performance of CEDAR on two real Internet packet traces [28], [29]. First, we verify the unbiasedness of the CEDAR estimator, and also check that its relative error is constant over the entire counter scale, as proved in Theorem 4.

In Figure 6 we show the CEDAR estimation results with a 4KB estimation array (12-bit estimators). The initial scale is set to a relative error of $\delta_0 = 1\%$ with steps of $\delta_{\text{step}} = 0.5\%$. The maximal value we reached is 10^6 which is equivalent to an error of 3%. As it shown in the figure, CEDAR keeps its unbiasedness over the entire scale as well as its relative error.

In Figure 7 we compare the CEDAR's relative error to SAC and DISCO with two different Internet traces. In Figures 7(a) and 7(b) we are using 12-bit estimators. CEDAR parameters are just the same as the previous experiment. For SAC we used a 8:4 bit allocation as recommended in [23], that is, 8 bits to the magnitude (which is denoted in [23] by $A[i]$) and 4 bits to the exponent (denoted by $mode[i]$). As DISCO introduces no up-scaling scheme we need to make some assumption on the maximal estimated value in order to compare it to SAC and CEDAR, we assume it can scale up to 32-bit counter. Therefore, we set the DISCO exponent parameter to $b = 1.0041$ which can reach the maximum real counter limit with no need to do up-scaling.

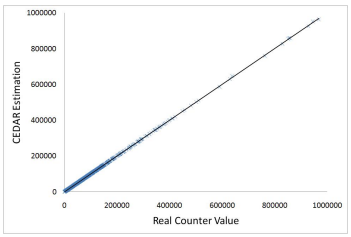
As shown in the figure SAC achieves the best error in small counter values, since for small values it can represent the exact counter value. However, from counter value of 10^3 and on we see that it performs the worst, reaching a relative error of above 5%. Regarding DISCO, it performs better than SAC, however it is bounded between 4% and 5%. As we can see, CEDAR keeps a near-constant relative error throughout the whole counter scale in spite of the fact that it performed 4 scaling steps. CEDAR relative error is lower by a third compared to those of DISCO and SAC. In Figure 7(c) we use trace [29] again but with 8-bit estimators. For CEDAR we use again $\delta_0 = 1\%$ and $\delta_{step} = 0.5\%$, for SAC we use a 5:3 bit allocation and for DISCO we use an exponent parameter of $b = 1.079$. As we can see again, CEDAR max relative error is better by third from both SAC and DISCO. Interestingly, we can see how SAC estimator restriction to the form $a2^b$ causes high oscillations in its relative error.

In order to check the error adaptation of the CEDAR up-scaling algorithm, we ran again trace [28] using 12-bit estimators but this time we stop the stream at three different points: after 1% of the trace, 10% and 100%. At every point we sample the counter estimations through the entire counter scale and measure the relative error. We compare the results to a static CEDAR, i.e. a CEDAR array that uses no up-scaling and relies on the maximal counter value from the beginning. Figure 8 shows the relative error results. We can see in Figure 8(a) that without the scaling algorithm the error is always 3% no matter what the real maximal counter value is. When using scaling, like in Figures 8(b) and 8(c), we can see that the error is adjusted to the current maximal counter (after 1% and 10% of the trace).

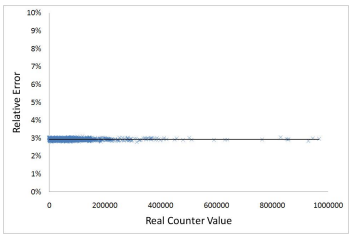
VI. FPGA Implementation

A. Implementation Description

In this section we examine a CEDAR implementation using Xilinx Virtex-5 FPGA. We use 192KB RAM for the flow pointer array and 16KB for the estimation array. Each estimator is 32-bit width. The estimation values (given by Equation (2)) are multiplied by 1000 in order to avoid float operations, yet keep a reasonable accuracy. The maximal design clock rate is 170MHz, therefore, assuming packets are accessed through a 32-bit bus with the same clock, our design may support streams of up to 5.4Gbps. Without any optimization our design uses 2500 1x6 LUT and 2500 flip-flops (roughly equivalent to 12K gates on ASIC design). In order to enable fast reading and writing operations we use DPR (Dual Port RAM) for all arrays, in this way both the CEDAR module and the client application can access any array at any time. The CEDAR module has

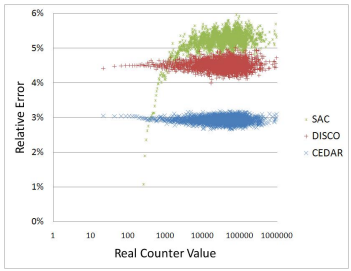


(a) CEDAR unbiased estimation vs. real counter values (Property 1).

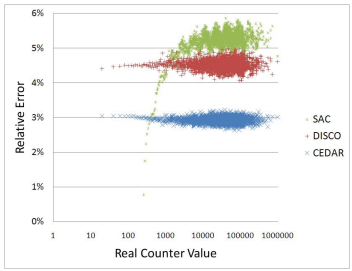


(b) CEDAR constant relative error (Theorems 3 and 4).

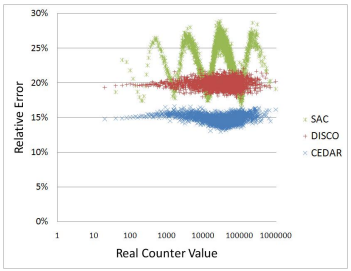
Fig. 6. CEDAR results on a real IP packet trace [28] using 12-bit estimators.



(a) Trace [28] counter estimation using 12-bit estimators.

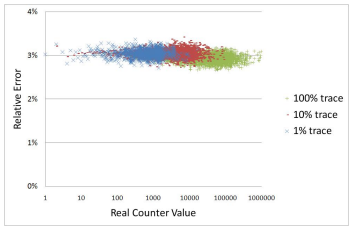


(b) Trace [29] counter estimation using 12-bit estimators.

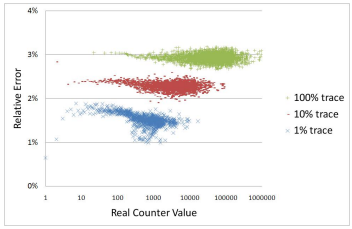


(c) Trace [29] counter estimation using 8-bit estimators (note the new plot scale).

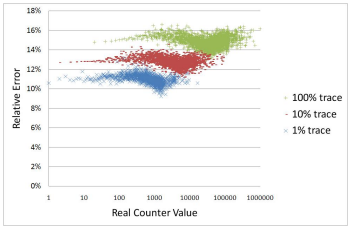
Fig. 7. Relative error comparison on real IP packet traces [28] and [29] using 12-bit and 8-bit estimators.



(a) Static CEDAR with 12-bit estimators. The maximal value is aimed at 10^6. The matching relative error is kept at 3% throughout the trace.



(b) CEDAR with up-scaling, using 12-bit estimators. The relative error is adjusted according to the current maximal counter value.



(c) CEDAR with up-scaling, using 8-bit estimators (note the new plot scale). The relative error is adjusted according to the current maximal counter value.

Fig. 8. Relative error comparison between static CEDAR and CEDAR with up-scaling, using trace [28].

an I/O register interface in order to program it with initial configurations, e.g. array sizes, and commands, e.g. start/stop.

Figure 9 depicts our implementation block scheme. The CEDAR module itself is built of a simple state machine. For any incoming packet we first fetch the flow pointer, then we fetch the two successive estimators and finally we decide whether to update or not the flow pointer. Both the flow pointer array and the estimation array are stored in a dual-port RAM in order to enable the client application fast reading of the flow counter estimation at any given time.

A second estimation RAM is drawn with a dashed line in order to support the up-scaling algorithm. The up-scaling algorithm is done by a dedicated SW application. In order to trigger this application, CEDAR generates an interrupt when the maximal flow index exceeds a programmable threshold. In addition, CEDAR has a *current-up-scaled-flow-index* register. This register is updated by the up-scaling application during its iteration through the flow pointer array. CEDAR uses this register in two manners: to decide which estimation array should be used for a specific flow (by comparing the flow pointer to that register) and to lock the updating of the current up-scaled flow.

B. Performance Analysis

We would like now to analyze the proposed model performance. As shown in the figure, we have 3 RAM accesses and sometimes even 4: one reading of the flow pointer, two readings of successive estimators and an optional writing of a new flow pointer. In our implementation we handle each packet separately. That is, we do not accept the next packet till we finish a full cycle of the CEDAR state machine. This imposes a delay between 2 successive incoming packets of at least 4 cycles. Typically, this is not a real problem since the packet minimal size is bounded by its header size which is usually longer than 4 words. So assuming we have a word access per cycle, our 4-cycle delay will not cause any stream halt.

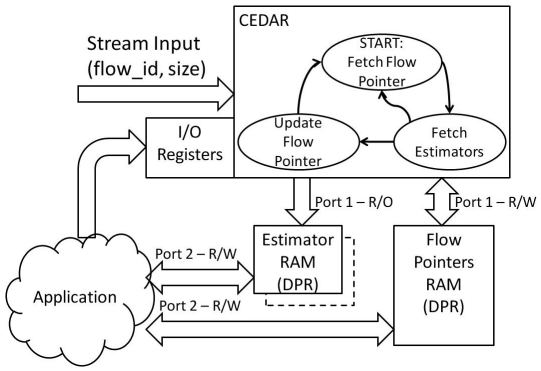


Fig. 9. CEDAR FPGA implementation block scheme. CEDAR is implemented as a three-state state machine. We use DPR (Dual-Port RAM) for the estimator array and for the flow pointers. Thus, both the CEDAR module and the CEDAR application can access arrays for fast updating and fast reading.

To conclude, we get a simple implementation that can work with high stream bit-rates and enable fast reading by the client application. Assuming an ASIC solution is much faster than FPGA, our implementation can probably support up to tens of Gbps. In terms of size, our implementation requires in the worst case 12K gates, which is negligible when compared to the flow pointer RAM requirement.

VII. Conclusion

In this paper we proposed a simple counter estimation method called CEDAR. We proved that it can guarantee optimal estimation in the manner of *min-max-relative-error*. We proved that the minimal boundary is attained when all estimators have *equal-relative-error*. In addition, we introduced a simple up-scaling algorithm which enables us to adjust the relative error of the entire counter scale according to the current maximal counter without having any pre-knowledge about the flow distribution. We also proved the unbiasedness of the CEDAR up-scaling algorithm.

Then we demonstrated the CEDAR estimation using two different Internet traces. We showed its unbiasedness as well as relative error equality through the entire counter scale. We also showed that it achieves the min-max boundary, up-scales smoothly, and adjusts the relative error estimation to the maximal flow counter.

Last, we showed a simple CEDAR implementation on FPGA that enables the client application to perform a fast reading of counter estimation at any given time. Our implementation consumes a relatively small space for the CEDAR logic part and can support high bit rates. We synthesized our FPGA to support a bit stream of 5.4Gbps, and expect an equivalent ASIC implementation to support tens of Gbps.

Acknowledgement

The authors would like to thank William Backshi for the CEDAR implementation on FPGA, as well as Ran Manevich and Ori Rottenstreich for their helpful comments. The work was partly supported by the European Research Council Starting Grant n°210389, as well as the Loewengart Research Fund, an Intel research grant on Heterogeneous Computing, and the Hasso Plattner Center for Scalable Computing.

References

[1] Cisco, CRS-1 multishelf system. http://www.cisco.com/en/US/products/ps5842/

[2] Juniper, TX matrix plus. http://www.juniper.net/us/en/products-services/routing/t-tx-series/txmatrix-plus/

[3] Broadcom-Dune, FE600 fabric element. http://www.dunenetworks.com/webSite/Modules/News/NewsItem.aspx?pid=355n&id=89g

[4] F. Abel *et al.*, "Design issues in next-generation merchant switch fabrics," *IEEE/ACM Trans. on Networking*, vol. 15, no. 6, pp. 1603-1615, 2007.

[5] A. Kabbani *et al.*, "AF-QCN: Approximate fairness with quantized congestion notification for multi-tenanted data centers," *IEEE Hot Interconnects*, Mountain View, CA, Aug. 2010.

[6] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," *Internet Measurement Workshop*, Nov. 2002.

[7] D. Lin and R. Morris, Dynamics of Random Early Detection," *SIGCOMM Symp. on Comm. Architectures and Protocols*, 1996.

[8] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," *28th International Conference on Very Large Data Bases*, 2002.

[9] X. Dimitropoulos, P. Hurley and A. Kind, "Probabilistic Lossy Counting: An efficient algorithm for finding heavy hitters," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, January 2008.

[10] O. Rottenstreich and I. Keslassy, "The variable-increment counting Bloom filter," *IEEE Infocom*, Orlando, FL, March 2012.

[11] C. Estan, G. Varghese, "New directions in traffic measurement and accounting," *ACM Trans. on Computer Systems*, vol. 21, no. 3, 2003.

[12] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, A. Kabbani, "Counter braids: a novel counter architecture for per-flow measurement". *ACM SIGMETRICS*, pp. 121–132, 2008.

[13] Y. Lu and B. Prabhakar, "Robust counting via counter braids: an error-resilient network measurement architecture," *IEEE Infocom*, 2009.

[14] N. Hua, B. Lin, J. J. Xu, and H. C. Zhao, "Brick: A novel exact active statistics counter architecture," *ACM/IEEE ANCS*, 2008.

[15] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Maintaining statistics counters in router line cards," *IEEE Micro*, vol. 22, no. 1, pp. 76–81, 2002.

[16] S. Iyer, R. R. Kompella, and N. McKeown, "Designing packet buffers for router linecards," *IEEE/ACM Trans. on Networking*, vol. 16, no. 3, pp. 705–717, 2008.

[17] M. Roeder and B. Lin. "Maintaining exact statistics counters with a multilevel counter memory," *IEEE Globecom*, Dallas, TX, 2004.

[18] Q. Zhao, J. Xu, and Z. Liu. "Design of a novel statistics counter architecture with optimal space and time efficiency," *ACM SIGMETRICS*, 2006.

[19] B. Lin and J. Xu, "DRAM is plenty fast for wirespeed statistics counting," *ACM Perf. Eval. Review*, vol. 36, no. 2, pp. 45–50, Sep. 2008.

[20] B. Lin, J. Xu, N. Hua, H. Wang, H. Zhao, "A randomized interleaved dram architecture for the maintenance of exact statistics counters," *ACM SIGMETRICS*, Seattle, WA, June 2009.

[21] H. Zhao, H. Wang, B. Lin, and J. Xu, "Design and performance analysis of a dram-based statistics counter array architecture," *ANCS*, Princeton, NJ, Oct. 2009.

[22] P. Flajolet, "Approximate counting: a detailed analysis," *BIT*, vol. 25, pp. 113-134, 1985.

[23] R. Stanojevic, "Small active counters", *IEEE Infocom*, 2007.

[24] C. Hu, B Liu, and K Chen, "Discount counting," *IEEE ICNP*, 2009.

[25] E. Tsidon, I. Hanniel and I. Keslassy, "CEDAR: Counter-Estimation Decoupling for Approximate Rates," Tech. Report TR11-04, Comnet, Technion, Israel. http://webee.technion.ac.il/~isaac/papers.html

[26] E. W. Weisstein, "Quadratic curve," MathWorld. http://mathworld.wolfram.com/QuadraticCurve.html

[27] E. Cohen, R. F. Riesenfeld and G. Elber, "Geometric modeling with splines: an introduction", 2001.

[28] CAIDA Anonymized 2008 Internet Trace, equinix-chicago 2008-03-19 19:00-20:00 UTC, Direction A. http://www.caida.org/data/monitors/passive-equinix-chicago.xml

[29] CAIDA Anonymized 2008 Internet Trace, equinix-chicago 2008-03-19 19:00-20:00 UTC, Direction B.