Robust Counting Via Counter Braids: An Error-Resilient Network Measurement Architecture

Yi Lu
Department of EE
Stanford University
Stanford, CA 94305
yi.lu@stanford.edu

Balaji Prabhakar
Department of EE and CS
Stanford University
Stanford, CA 94305
balaji@stanford.edu

***Abstract*—A novel counter architecture, called Counter Braids, has recently been proposed for accurate per-flow measurement on high-speed links. Inspired by sparse random graph codes, Counter Braids solves two central problems of per-flow measurement: one-to-one flow-to-counter association and large amount of unused counter space. It eliminates the one-to-one association by randomly hashing a flow label to multiple counters and minimizes counter space by incrementally compressing counts as they accumulate. The random hash values are reproduced offline from a list of flow labels, with which flow sizes are decoded using a fast message passing algorithm.**

The decoding of Counter Braids introduces the problem of collecting *flow labels* active in a measurement epoch. An exact solution to this problem is expensive. This paper complements the previous proposal with an approximate flow label collection scheme and a novel error-resilient decoder that decodes despite missing flow labels.

The approximate flow label collection detects new flows with variable-length signature counting Bloom filters in SRAM, and stores flow labels in high-density DRAM. It provides a good trade-off between space and accuracy: more than 99 percent of the flows are captured with very little SRAM space. The decoding challenge posed by missing flow labels calls for a new algorithm as the original message passing decoder becomes error-prone. In terms of sparse random graph codes, the problem is equivalent to decoding with *graph deficiency*, a scenario beyond coding theory. The error-resilient decoder employs a *new* message passing algorithm that recovers most flow sizes exactly despite graph deficiency. Together, our solution achieves a 10-fold reduction in SRAM space compared to hash-table based implementations, as demonstrated with Internet trace evaluations.

I. Introduction

Per-flow network measurement is important for a variety of purposes including accounting, traffic engineering and network forensics. A "flow" is a logical entity defined as a sequence of packets satisfying a common set of rules. For instance, packets with a specific source-destination address pair constitute a flow. Measuring flows of this kind yields useful information about routing distribution and network usage patterns. Flows can also be defined by classification results. In this case, one packet can potentially belong to more than one flow and consequently contribute to more than one counter.

With a highly specific definition of a flow, (for instance, the usual flow-tuple including source and destination addresses, source and destination ports, and flow type), modern high-speed links witness millions of flows in mere minutes, as observed in the OC-48 CAIDA traces (see Section V-C). The abundance of flows necessitates the use of a large number of counters, and a large database of flow labels (an example of flow label: [255.255.01.32, 235.129.5.5, 11, 5, 0]) accessible at link speed to direct increments to the correct counter.

The problem is exacerbated by the lack of affordable high-density high-bandwidth memory devices. The acceptable per-packet memory access time on high-speed links is much smaller than that of commercially available DRAM (tens of ns), necessitating the use of SRAM. However, due to their low density, large SRAMs are expensive and difficult to implement on-chip.

There are two central problems of per-flow measurement:

Flow-to-counter association. One-to-one association between flow labels and counters is maintained, in order for an arriving packet to update the correct counter. The association must be retrievable at link speed and is usually implemented as a SRAM hash table, with a flow label and its corresponding counter included in the same row. Flow labels are lengthy (for instance, it is 13 bytes long for the flow tuple described above), and the storage of the association consumes large amount of SRAM space.

Counter space. Each flow is assigned a counter that can accommodate the largest flow in the network, regardless of the actual flow size, and most counter bits are wasted.

Unlike previous approaches [1][2], Counter Braids (CB), proposed in [3], avoids storing the one-to-one flow-to-counter association by applying multiple random hash functions to flow labels on the fly. Counter space is shared among all flows with "braiding" and flow sizes are incrementally compressed. Exact measurement of *all* flows is achieved by recovering flow sizes offline at the end of each measurement epoch. The linear-complexity message passing decoding algorithm recovers hundreds of thousands of flow sizes with vanishing error in mere seconds.

Figure 1 illustrates the overall architecture of CB and Figure 2 shows the schematic diagram of a two-layer CB and its decoding graphs. Here are outlines of its operations:

Counting in SRAM. A packet computes 3 hash functions on its flow label and increments the layer-1 counters it hashes to. If a layer-1 counter overflows, it computes 3 hash functions on its location and increments the layer-2 counters hashed to.

Decoding offline. *Given the complete list of flow labels*, we

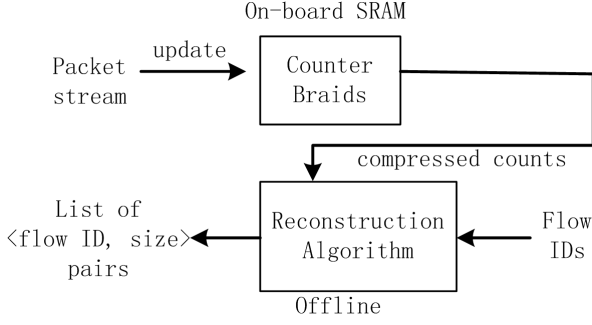

Fig. 1. System Diagram.

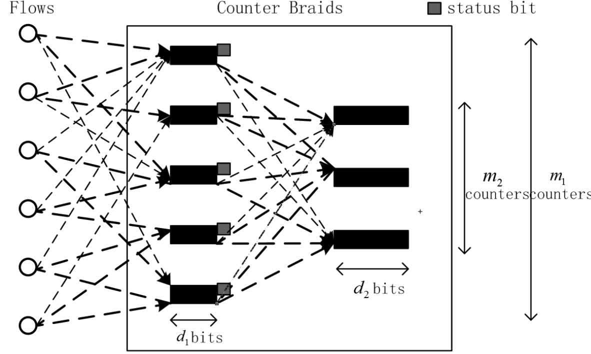

Fig. 2. Two-layer Counter Braids. The flow nodes (on the left) and the interconnecting graphs do not physically exist in SRAM. Each edge in the graph corresponds to a hash function computed on the fly by the entity (flow or layer-1 counter) on the left of the edge.

can reconstruct the interconnecting graphs in Figure 2 and use the message passing decoder [3] to obtain the flow sizes. Note that we are always able to reconstruct the graph between layer-1 and layer-2 counters without using the flow-label list.

The proposal of Counter Braids in [3] requires a complete flow label list at the decoding stage, hence introducing the problem of *flow label collection*. However, an exact flow label collection algorithm requires checking against a hash table at every packet arrival, which is costly in SRAM space.

A. Our Contributions

1. Space-efficient approximate flow label collection.
We propose to identify new flows using variable-length signature counting (VLSC) Bloom filters [4] in SRAM and store the flow labels in a simple list in off-chip high-density DRAM. Bloom filters with a total space of 0.6 MB are sufficient to capture all but 0.45% of flows for the OC-48 traces used for evaluation, and 80% of missing flows have fewer than 3 packets. This is in contrast to 28.4 MB for the exact collection, assuming a d-left [5] hash table is used with 1.5 times over-provisioning. The DRAM flow-label list is accessed only once for each flow, as the elimination of flow-to-counter association enables counting without accessing the flow labels.

Missing flow labels cause a problem to the original decoding algorithm, as the layer-1 decoding graph becomes incom-

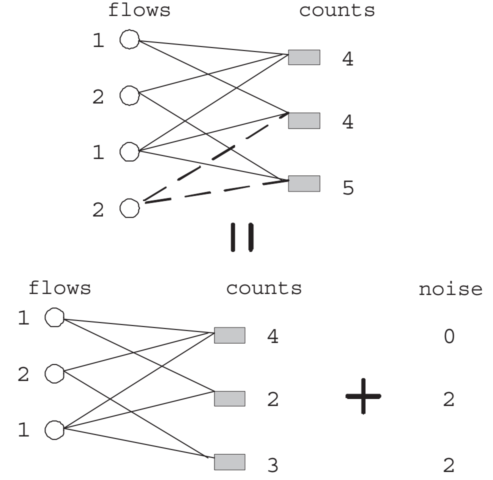

Fig. 3. Layer-1 decoding graph with missing flow labels. The top figure shows the actual flows and corresponding counts, with dashed lines indicating contribution from the missing flow. The bottom figure shows the actual decoding graph and noise in counters due to the missing flow.

plete. The graph deficiency results in extra counts in some unidentified layer -1 counters and we model them as *noise*, as illustrated in Figure 3. Formally, let the vector of flow sizes be $\mathbf{f}$, the adjacency matrix corresponding to the interconnecting graph be A, the vector of counter values be $\mathbf{c}$ and the noise be $\mathbf{n}$, the problem is to recover $\mathbf{f}$ from $\mathbf{c}$ where

$$\mathbf{c} = A\mathbf{f} + \mathbf{n}.$$

A crucial property of the original message passing algorithm in [3] is *anti-monotonicity*: it computes an upper bound of the flow sizes in odd iterations and a lower bound in even iterations. The algorithm aggressively selects the best bounds in both directions to quickly close the gap. The anti-monotonicity property is no longer valid with missing flow labels: a noisy counter *overestimates* the remaining flow sizes, hence it can produce an upper bound when a lower bound is expected. The aggressive optimality of the original algorithm makes it mistake an invalid bound for the best bound available, and results in massive propagation of errors.

2. Error-resilient message passing algorithm.
We propose a new layer-1 message passing algorithm to recover exact flow sizes with vanishing error, *despite missing flow labels*. Together with Bloom Filters, it achieves a 10-fold saving in space compared to d-left hash-table based solutions.

The new message passing algorithm makes a small, but crucial, modification to the original algorithm: it makes aggressive update on the upper bounds, but *conservative* on the lower bounds. The conservative update of flow sizes, on the other hand, *aggressively impedes* the propagation of invalid bounds, and restricts the effect of noise to a small set of flows (the set can be empty). Flows not in the set are decoded exactly, and for flows in the set, the algorithm is equivalent to taking the minimum of the overestimates, which keeps the errors small.

We note that the current solution does not measure all flows since a small fraction of flows are missing with the approximate flow label collection algorithm. However, the solution offers a superior trade-off between accuracy and space than other approximate counting solutions such as the sample-and-hold algorithm [2]. While the sample-and-hold algorithm counts the 1000 largest flows out of hundreds of thousands, our solution counts all but a small percentage (*e.g.*, 0.45%) of flows, most of which are small (*e.g.*, 80% of missing flows have fewer than 3 packets).

B. Theoretical Implications

Sparse random graph codes. Counter Braids[3] is methodologically inspired by the theory of low-density parity check codes [6][7]. In the communication scenario, the random graph of a code is pre-defined and stored in both encoders and decoders. With Counter Braids, however, the random graph is generated on the fly with hash functions at the encoding stage, and is stored for the decoding stage by collecting flow labels. Missing flow labels result in variable nodes being removed from the decoding graph, together with all edges connected to them, a problem we call *graph deficiency*. There can also be other forms of graph deficiency. For instance, network-wide collaborative measurement using Counter Braids can result in a *fraction* of edges missing for a variable, as a flow label is captured at some routers but not the other. We do not elaborate on the network-wide application in this paper.

The problem of graph deficiency raises interesting questions for the theory of sparse random graph codes: Can the remaining variable nodes be recovered with graph deficiency? And what is the level of deficiency that can be tolerated with a certain error rate? The error-resilient message passing algorithm proposed in this paper serves as a first step towards answering these questions.

Compressed sensing. Efficient algorithms for sparse signal recovery from a system of linear equations are of much interest to the compressed sensing community [8][9][10][11]. The problem of finding the sparsest solution to a system of underdetermined linear equations is in general NP-hard, but when the signal is sparse enough, it can be recovered with high probability in polynomial time.

The decoding of CB can be viewed as *recursive* sparse signal recovery from systems of underdetermined linear equations. In particular, we consider the case where the sparsity is linear in the signal dimension. The original decoder in [3] is hence a sparse signal recovery algorithm with complexity almost *linear* in the signal sparsity. With missing flow labels, an unknown subset of the linear equations becomes linear inequalities, or equivalently, polluted with noise that is correlated and of unbounded size. Instead of recovering a good approximation of the signal [12], we seek to recover the signal *exactly* despite the noise with the error-resilient message passing algorithm, which can be interesting to sparse signal recovery in general scenarios.

The rest of the paper is organized as follows. We present the space-efficient flow label collection algorithm in Section II. Section III describes the error-resilient message passing decoder with missing flow labels, and Section IV analyzes the algorithm using density evolution. We evaluate the performance of the VLSC Bloom filter, the decoder and the overall architecture in Section V, and conclude in Section VI.

II. Space-efficient Flow Label Collection

A. Problem

For each measurement epoch, we want to collect the labels of all flows that have contributed to the counts in Counter Braids. Memory resource constraints prevent us from recording the label for each packets, which would alone require hundreds of Megabytes for a 5-minute trace. Ideally we want to identify a new flow when it starts, and record one label only for each flow.

A TCP flow starting in the current epoch is easy to identify by capturing its “SYN” packet, a part of the handshake process. The collection of flow labels of TCP flows starting in *previous* epochs and *non-TCP* flows is more difficult, and requires the *detection* of a new flow.

One exact, but expensive, solution is using an SRAM hash table to maintain a list of active flow labels: a packet with no flow label in the hash table signals a new flow arrival. However, the hash-table solution requires keeping the entire flow label, which is 13 bytes for a 5-tuple. Moreover, even a hash table using the very efficient d-left load balancing needs approximately 1.5 times over-provisioning to reduce hash table collisions.

B. Approximate Solution with VLSC Bloom Filters

We propose to use a VLSC Bloom filter [4] for flow label collection. A Bloom filter [13] is a fingerprinting device that supports approximate set membership queries. Instead of storing a 13-byte unique ID for each element in the set, it stores a binary vector, which is a “XOR” sum of a low-dimension projection of all IDs in the set, hence reducing space drastically.

While the basic version of Bloom filter works for a static set, the VLSC Bloom filter deals with a *dynamic* set, whose membership varies over time. Its probabilistic nature makes it produce false positives and false negatives: it may declare that an element is in the set when it is not, or it may declare that it is not in the set when it is. However, by sizing the Bloom filter appropriately, both errors can be made small enough for many applications [14][15]. We defer to the Appendix an outline of the operations of the basic and the VLSC Bloom filters, and refer the reader to [13] and [4] for more details.

Figure 4 shows a flowchart for flow label collection with Bloom filters. Note that the path to flow label collection is separate from that to Counter Braids, hence the two paths can be implemented in parallel.

All flow labels collected are stored in a simple list in the DRAM: a new flow label is appended at the end of the list. The

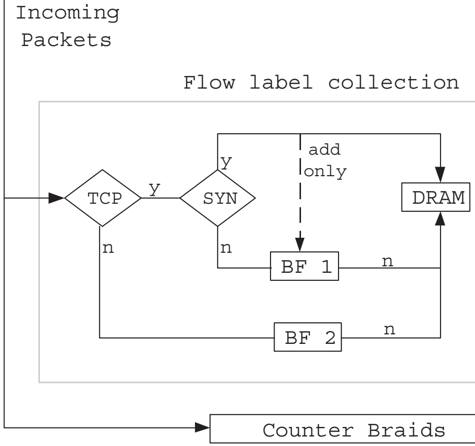

Fig. 4. Flow label collection using VLSC Bloom filters.

access time of DRAM poses no problem since flow arrivals are much less frequent than packet arrivals [16]. In fact, a single-port DRAM suffices since only write operations take place during a measurement epoch. At the end of an epoch, the DRAM content is flushed to software together with the CB content.

Figure 4 shows two VLSC Bloom filters, of which "BF 1" is used to identify TCP flows continuing from the previous epoch, and "BF 2" is used to identify non-TCP flows. The filter "BF 1" is only active for the first $15 - 30$ seconds of the epoch since most continuing flows send at least one packet during this time. We disregard the flows that send no packet during this period, as they constitute a very small fraction of the flows. In contrast, "BF 2" is active throughout the measurement epoch.

There are three paths to the DRAM, corresponding to the three different flow types and flow label collection processes. We describe them from top to bottom. The first path corresponds to a TCP "SYN" packet, belonging to a TCP flow starting in the current epoch. The flow label of the "SYN" packet is recorded in DRAM directly, without consulting the Bloom filters. The dashed line labelled "add only" inserts the flow label into the Bloom filter "BF 1", when the filter is active, so that further packets of the same flow will not cause duplicated labels in the DRAM.

The second path corresponds to a TCP packet that is not "SYN". This is to collect labels of TCP flows starting in previous epochs. If the packet does *not* find its signature in "BF 1", its flow label is recorded in the DRAM, and also inserted in "BF 1" to prevent duplicated labels. Missing flow labels result from *all* packets of a flow being false positives in "BF 1", hence *a long flow is more likely to have its flow label recorded*. A duplicated flow label results from *each* false negative packet. We do not consider duplication a problem so long as the DRAM is not excessively accessed. The duplication is removed before decoding in software.

In addition, when a TCP flow terminates, a "FIN" packet is detected, and its flow label is *deleted* from "BF 1". The deletion operation is possible since we are using the VLSC Bloom filter. This operation is not shown in Figure 4 to keep the figure simple.

The third path corresponds to a non-TCP packet. The operations are the same as that of the second path, except for deletion. Since non-TCP flows sends no "FIN" packet, deletion is not allowed, and we instead *age* the bits in the Bloom filter to keep the membership up to date. Missing flow labels and duplication result from the same principle.

As shown in Section V-A, the tradeoff between space and accuracy with the VLSC Bloom filter is favorable. With 16 bits per flow, more than 99.5% of all flows are captured, in contrast to 156 bits per flow for an exact collection. In addition, almost *all* long flows are captured and most missing flows are small.

III. Error-resilient Message Passing Decoder

We propose a novel error-resilient layer-1 decoding algorithm to work with the approximate flow label collection. The new algorithm introduces a crucial modification to the algorithm in [3] to decode even when some flow labels are missing. It retains the features of the original algorithm such as linear complexity, low memory requirement for compression with countable alphabets, and exact decoding of most flows. We do not include in this paper the upper layer algorithms as they remain the same, and refer the reader to [3].

A. Algorithm

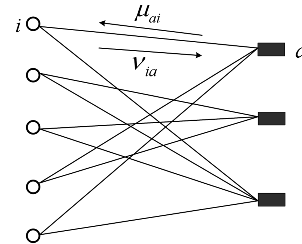

Fig. 5. Message passing on a bipartite graph with flow nodes (circles) and counter nodes (rectangles.)

Consider the layer-1 decoding graph illustrated in Figure 5. It is a bipartite graph with flow nodes on the left and counter nodes on the right. The flow nodes are obtained from the list of flow labels collected. An edge connects flow i and counter a if one of the hash functions maps flow i to counter a. Let $\mathbf{f}$ denote the vector of flow sizes whose labels are *collected*, and let $\mathbf{c}$ denote the counter values.

$$c_a = \sum_{i \in \partial a} f_i + n_a,$$

where ∂a denotes the flows adjacent to counter a on the graph and n_a is a non-negative integer corresponding to counts due to missing flows. The problems is to estimate $\mathbf{f}$ from $\mathbf{c}$.

Message passing algorithms are iterative. In the tth iteration messages are passed from all counter nodes to all flow nodes and then back in the reverse direction. A message goes from counter a to flow i (denoted by μ_{ai}) and vice versa (denoted

by ν_{ia}) only if nodes a and i are neighbors (connected by an edge) on the bipartite graph.

The error-resilient message passing algorithm is listed in Exhibit 1. The messages start from the flow nodes and are initialized to 0. At a flow (counter) node, an outgoing message is computed for each edge ia (ai), based on incoming messages on all edges except ia (ai). The interpretation of the messages is as follows: μ_{ai} conveys counter a's guess of flow i's size based on the information it received from neighboring flows *other than flow* i. Conversely, ν_{ia} is the guess by flow i of its own size, based on the information it received from neighboring counters *other than counter* a.

Exhibit 1: The Error-Resilient Message Passing Algorithm

1: **Initialize**
2: min = minimum possible flow size;
3: $\nu_{ia}(0) = 0 \quad \forall i$ and $\forall a$;
4: $c_a = a^{th}$ counter value

5: **Iterations**
6: for iteration number $t = 1$ to T
7: $\mu_{ai}(t) = \max\left\{\left(c_a - \sum_{j \neq i} \nu_{ja}(t-1)\right), min\right\}$;
8: $\nu_{ia}(t) = \min_{b \neq a} \mu_{bi}(t)$

9: **Final Estimate**
10: $\widehat{f}_i(T) = \min_a\{\mu_{ai}(T)\}$

B. Error-Resilience

First consider the noiseless situation. It is easy to verify that the messages $\mu_{ai}(t)$ and $\nu_{ia}(t)$ are upper bounds of the flow size f_i when t is odd and lower bounds when t is even. The algorithm performs best when the *tightest* bound is selected at the flow node at each iteration, that is, choosing the minimum of incoming upper bounds when t is odd, or maximum of incoming lower bounds when t is even.

This optimal strategy, however, is *error-prone*. With non-negative noise in the counter values, a counter node can send an upper bound when a lower bound is expected, and vice versa. We call such a message a false lower (upper) bound. The optimal strategy will lead the flow nodes to choose a false lower (upper) bound at the first opportunity, hence propagating errors aggressively.

The error-resilient algorithm, on the other hand, resists error by choosing the most conservative lower bound. The chance of error propagation is drastically reduced: A false lower bound is chosen only if *all* incoming lower bounds are false. This is in contrast to the error-prone strategy where a false lower bound is chosen so long as *one* of the incoming lower bounds is false.

Remark. The choice of upper bounds remains aggressive so that the gap can close. We choose to be conservative with the choice of lower bounds since the non-negative nature of the noise makes false bounds originate in even iterations, that is, with lower bounds. It is more effective to reduce the error propagation earlier in the iterations.

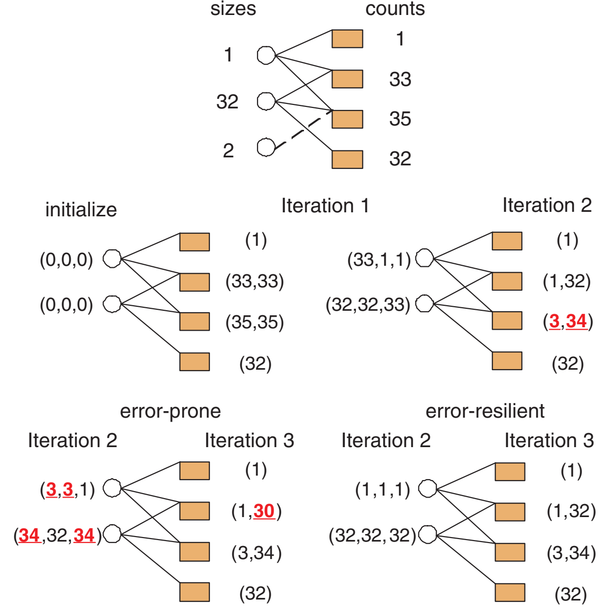

Fig. 6. Illustration of error-resilience. Numbers in the topmost figure are true flow sizes and counter values, with the dashed line indicating the missing flow. In an iteration, numbers next to a node are messages on its outgoing edges, ordered from top edge to bottom edge. Each iteration involves messages going from counters to flows and back from flows to counters. The last two figures compare ν_{ia} updates of iteration 2 and μ_{ai} updates of iteration 3 between the error-prone and error-resilient algorithms. False lower (upper) bounds are underlined.

Toy example. Figure 6 shows the evolution of messages over 3 iterations on a toy example. This example is trivial since each flow node has one counter that registers exactly its value. It is shown simply as an illustration for error-resilience.

At iteration 2, counter 3 sends false lower bounds "3" and "34". The false bounds are propagated by the error-prone algorithm as it aggressively chooses the largest (seemingly best) lower bounds, which results in a false upper bound "30" by counter 2 at the next iteration. The error-resilient algorithm, however, chooses the most conservative lower bounds, and remains unaffected by the false bounds.

The flow estimates for both algorithms at each iteration are listed in Table I, with f_i^p denoting the error-prone estimate of the i-th flow and f_i^r the error-resilient estimate. In this particular example, all flow sizes are reconstructed correctly with the error-resilient algorithm, but incorrectly with the error-prone one. The false bounds are highlighted in bold.

iteration	f_1^p	f_2^p	f_1^r	f_2^r
0	0	0	0	0
1	1	32	1	32
2	**3**	**34**	1	32
3	1	**30**	1	32

TABLE I
COMPARISON OF FLOW ESTIMATES BY THE ERROR-PRONE AND ERROR-RESILIENT ALGORITHMS.

IV. Analysis of Error-resilient Decoding

The sparse random decoding graph allows us to analyze the asymptotic performance of the algorithm using the *density evolution* principle [7]. Density evolution exploits the *locally tree-like* property of sparse random graphs as the system size goes to infinity: for any finite iterations of the message passing algorithm, all incoming messages at any node are *independent* and a recursive function can be written for the evolution of reconstruction error. Due to space constraint, we shall be content to state the main theorems and highlight interesting results.

Let $\beta = m/n$, where m is the number of counters and n is the number of flows, including those with missing labels. Let each flow node connect to k uniformly sampled counter nodes. We obtain the asymptotic equation by keeping β constant and letting n and m go to infinity. Let s be the proportion of missing flow labels.

Let

$$\rho_{\gamma,1}(x) = \sum_{i=1}^{\infty} \frac{e^{-\gamma}(\gamma x)^{i-1}}{(i-1)!},$$

$$\rho_{\gamma,2}(x) = \sum_{i=2}^{\infty} \frac{e^{-\gamma}(\gamma x)^{i-1}}{(i-1)!}$$

where $\gamma = (1-s)nk/m = (1-s)k/\beta$ is the average degree of a counter node in the decoding graph, that is, with only *collected* flow labels. The probability that a edge originates from a counter node of degree i converges to a Poisson distribution as $n \to \infty$, and $\rho_{\gamma,1}(x)$ is the generating function for the Poisson distribution. The quantity $\rho_{\gamma,2}(x)$ is defined for convenience of expression.

Assume that we are given the flow size distribution and let

$$\epsilon = \mathbb{P}(f_i > min).$$

Recall that min is the minimum value of flow sizes.

Let

$$f(\gamma, x) = \epsilon\{1 - [\rho_{\gamma,1}(1 - [1 - \rho_{\gamma,1}(1-x)]^{k-1})]^{k-1}\},$$

and

$$\gamma^* \equiv \sup\{\gamma \in \mathbf{R} : x = f(\gamma, x) \text{ has no solution } \forall x \in (0,1]\}.$$

Theorem 1: **Threshold with Intact Flow Labels.** If $s = 0$, we have

$$\beta^* \equiv \frac{m}{n} = \frac{k}{\gamma^*}$$

such that in the large n limit
(i) If $\beta > \beta^*$, $\widehat{\mathbf{f}}(2t) \uparrow \mathbf{f}$ and $\widehat{\mathbf{f}}(2t+1) \downarrow \mathbf{f}$.
(ii) If $\beta < \beta^*$, there exists a positive proportion of flows such that $\widehat{f}_i(2t) < \widehat{f}_i(2t+1)$ for all t. Thus, some flows are not correctly reconstructed.

Remark. The threshold with intact flow labels is a good indication of actual performance with small fraction of flow labels missing, which we shall demonstrate in Figure 7 using simulation data. We are interested in small fraction of missing flows since we want to measure most of the flows and this region yields good space-accuracy tradeoff for Bloom filters.

Next, we are interested in the region called "error floor", which is the error probability for β above the threshold. Formally, let $p_{k,s}(\beta)$ be the probability that a flow *with label* is incorrectly decoded as $n \to \infty$, with k hash functions and a fraction of s missing flow labels. Let $\beta^*_{k,s}$ be the corresponding threshold, *i.e.*, the discontinuity point of $p_{k,s}(\beta)$. The error floor is the curve $p_{k,s}(\beta)$ for $\beta > \beta^*_{k,s}$.

Let $c = 1 - \exp(-\frac{s\gamma}{1-s})$ be the proportion of noisy counters. Let

$$f_i(\gamma, x) = h_{\gamma,i}(1 - (1 - g_{\gamma,i}(x^{k-1}))^{k-1}, x^{k-1}),$$

where

$$\begin{aligned} g_{\gamma,1}(x) &= (1-c)\rho_{\gamma,2}(1-x), \\ h_{\gamma,1}(x,y) &= (1-c)\rho_{\gamma,2}(1-x) + c\rho_{\gamma,2}(y), \\ g_{\gamma,2}(x) &= 1 - \rho_{\gamma,1}(1-x), \\ h_{\gamma,2}(x,y) &\equiv h_{\gamma,2}(x) = (1-c)(1-\rho_{\gamma,1}(1-x)) + c. \end{aligned}$$

and

$$\begin{aligned} l_{k,s}(\beta) &\equiv \inf\{x \in [0,1] : x = f_1(\gamma, x)\}, \\ u_{k,s}(\beta) &\equiv \inf\{x \in [0,1] : x = f_2(\gamma, x)\}, \end{aligned}$$

Theorem 2: **Asymptotic Error Floor.** For $\beta > \beta^*_{k,s}$,

$$l_{k,s}(\beta) < p_{k,s}(\beta) < u_{k,s}(\beta).$$

Remark. With intact flow labels, the asymptotic error floor is 0. With missing flow labels, the asymptotic error floor is positive for a finite β due to the propagation of error from the noisy counters to a small neighborhood.

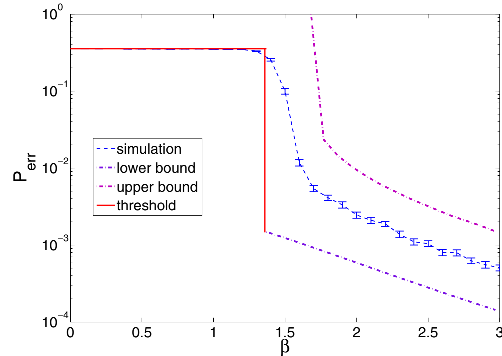

Fig. 7. Probability of reconstruction error for flows with labels ($\mathrm{P_{err}}$) against number of counters per flow (β). Threshold with intact flow labels and bounds on asymptotic error floor are shown with simulation data, with $k = 4$ and $s = 0.05$.

Figure 7 shows the predicted error curve with threshold of intact flow labels and bounds on asymptotic error floor for $k = 4$ and $s = 0.05$, together with simulation data.

A. Two Error Regions

With intact flow labels, the threshold divides the region with positive error from that with *zero* error; with missing flow labels, the threshold divides the region of *large-scale failure* from that of *error floor*, assuming only a small fraction of flow labels are missing[1]. The large-scale failure is due to an insufficient number of counters and decoding cannot get started, whereas the error floor results from a small neighborhood of the noisy counters.

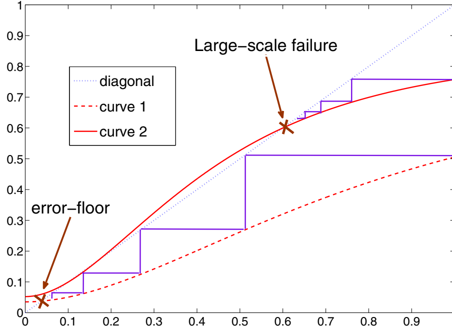

Fig. 8. Density evolution above and below threshold, resulting in error floor and large-scale failure respectively.

Figure 8 illustrates the decoding processes leading to the two error regions respectively. Curve 1 and curve 2 are two different mappings from a current reconstruction error probability to that after two iterations[2]. Hence, the error evolution is equivalent to a walk between a curve and the diagonal line, and the *fix point* to which the walk converges yields the final reconstruction error.

Curve 1 corresponds to β above the threshold. It intersects the diagonal only at one point close to the origin, labelled "error floor", and the walk converges to the intersection point. Curve 2 corresponds to β below the threshold. It intersects the diagonal at 3 points, and the walk converges to the intersection closer to 1, labelled "large-scale failure". As β increases from below the threshold to above, the curve moves from 2 to 1. The *threshold* corresponds to the β value where the curve intersects the diagonal at *two* points and the larger one is a tangential intersection. As the number of intersection points decreases, the *abrupt change* of the fixed point causes discontinuity in the final reconstruction error at the threshold.

B. Flow Node Degree and Missing Fraction

We investigate the effect of flow node degree and fraction of missing flow labels on the threshold and asymptotic error floor using Theorem 1 and 2. For this section, let the flow size distribution $\mathbb{P}(f_i > x) = x^{-1.5}$.

[1]When a large fraction of flow labels are missing, the error floor will be so high that it interferes with the threshold phenomenon. Experiments show that "large" means at least 20%, and can be much larger with increasing flow node degrees.

[2]The recursion is over two iterations instead of one because the mapping is different for odd and even iterations.

Table II shows the result for different flow node degrees k. The threshold with intact flow labels is denoted by β^*. The upper and lower bounds of asymptotic error floor are denoted by l and u respectively, and are computed for a missing fraction $s = 0.04$ and number of counters per flow $\beta = 2$. The threshold *increases* with k for $k > 3$, with the minimum achieved at $k = 3$. However, the error floor *decreases* as k increases. This is because of the symmetry between decoding and error propagation: *a good decoding graph is also a good error propagation graph.*

k	2	3	4	5	6
β^*	1.19	1.16	1.37	1.56	1.75
l	0.02	0.002	3×10^{-4}	6×10^{-5}	1×10^{-5}
u	0.3	0.03	0.004	0.001	3×10^{-4}

TABLE II
THRESHOLD (β^*), LOWER (l) AND UPPER (u) BOUNDS OF ASYMPTOTIC ERROR FLOOR FOR $2 \leq k \leq 7$. $\mathbb{P}(f_i > x) = x^{-1.5}$.

Table III shows the values of l and u for different fraction of missing labels, s, with $\beta = 2$ and $k = 4$. Both bounds increases polynomially in s with an exponent of 3, corresponding to the exponent $k - 1$ in the definition of $f_i(\gamma, x)$.

s	0.05	0.03	0.01	0.008
l	6×10^{-4}	1×10^{-4}	5×10^{-6}	3×10^{-6}
u	0.01	0.002	8×10^{-5}	3×10^{-5}

TABLE III
LOWER (l) AND UPPER (u) BOUNDS OF ASYMPTOTIC ERROR FLOOR FOR DIFFERENT VALUES OF s. $\mathbb{P}(f_i > x) = x^{-1.5}$.

V. EVALUATION

We evaluate the space-accuracy tradeoff of VLSC Bloom filters in Section V-A, and the performance of the error-resilient decoder in Section V-B. The combined performance of the entire system as shown in Figure 4 is presented in V-C.

A. VLSC Bloom filters

We simulate the performance of a single VLSC Bloom filter on a 5 million packet CAIDA trace collected at 9:10 am, Aug 14, 2002. There are a total of 168640 flows. The number of concurrently active flows reaches a maximum of 46953. Hence we size the Bloom filters for 47000 flows.

Space (bits per flow)		12	14	16
Missing ($\times 10^{-3}$)	Large	0.01	0	0
	Medium	4.6	2.5	1.2
	Small	28	11	5
	Overall	19	7.2	3.8
Duplication ($\times 10^{-2}$)		3.1	2.2	0.901

TABLE IV
FRACTION OF MISSING FLOW LABELS OF EACH CATEGORY AND FRACTION OF DUPLICATED FLOW LABELS.

The result with the deletion operation is shown in Table IV. We omit the result with the aging operation as it is similar. We designate the longest 10% of the flows (which brings 80% of the traffic) as "long", and the shortest 60% as "short" and the rest as medium. The result is given in fraction of actual number of flow labels. For instance, with 12 bits per flow, 0.001% of the long flows are missing, as well as 0.465% of the medium flows and 2.8% of the short flows. Overall, 1.9% of the flows are missing. And we get 3.1%, or 5000, duplicated flow labels. With 16 bits per flow, more than 99.5% of all flows are captured, and most missing flows are small.

B. *Error-resilient Decoder*

Let $\tilde{n}$ be the number of flows with labels, f_i be the true size and $\widehat{f_i}$ be the estimated size of the i-th flow. We measure the following quantities:

1) Reconstruction Error, $\mathrm{P_{err}}$: Fraction of flows incorrectly decoded.

$$\mathrm{P_{err}} \equiv \frac{\sum_{i=1}^{n} \mathbb{I}(f_i \neq \widehat{f_i})}{\tilde{n}}.$$

2) Average error magnitude, E_m: The ratio of the sum of absolute errors and the number of errors. It measures how big an error is *when an error has occurred*.

$$E_m = \frac{\sum_i |f_i - \widehat{f_i}|}{\sum_i \mathbb{I}(f_i \neq \widehat{f_i})}.$$

To evaluate the decoding performance, we use 1000 flows and generate a random graph *and* a random set of flow sizes from the distribution $\mathbb{P}(f_i > x) = x^{-1.5}$ for each run of experiment. Missing flow labels are selected uniformly and discarded before decoding. The quantities of interest are averaged over enough runs so that their standard deviations are less than 10% of their respective values. We use synthetic traces of a known distribution to obtain a better understanding of the decoding property, isolated from other factors.

The reconstruction error, $\mathrm{P_{err}}$, is shown in log scale, while the error magnitude, E_m, is shown in linear scale. Figure 9 shows the performance with different flow node degrees k. As k increases, the threshold of $\mathrm{P_{err}}$ *increases* and the error floor *decreases*, both in agreement with Table II. The error magnitude E_m decreases linearly below the threshold, and stays close to 1, which is the minimum possible value, above the threshold.

Figure 10 shows the performance with different fraction of missing labels s. As s decreases, the error floor decreases, and s has no effect on the threshold, both in agreement with Table III. The error magnitudes for different s are indistinguishable as they share the same threshold.

C. *Trace Simulation with Flow Label Collection*

We use two OC-48 (2.5 Gbps) one-hour contiguous traces at a San Jose router. Trace 1 was collected on Wed Jan 15, 2003, 10 am to 11 am and Trace 2 was collected on Thur Apr 24, 2003, 12 am to 1 am. We divide each trace into 12 5-minute segments, each corresponding to a measurement epoch. Each segment in trace 1 contains about 0.9 million flows and 20 million packets, and each segment in trace 2 contains about 0.7 million flows and 9 million packets. The statistics across different segments within one trace are similar.

The same sizings for VLSC Bloom filters and Counter Braids are used for all segments, mimicking the realistic scenario where traffic varies over time and the devices are built in hardware. The TCP Bloom filter is 0.2 MB and the non-TCP Bloom filter is 0.4 MB. We use a two-layer CB with 3 hash functions at both layers. The layer-1 counters are 8 bits

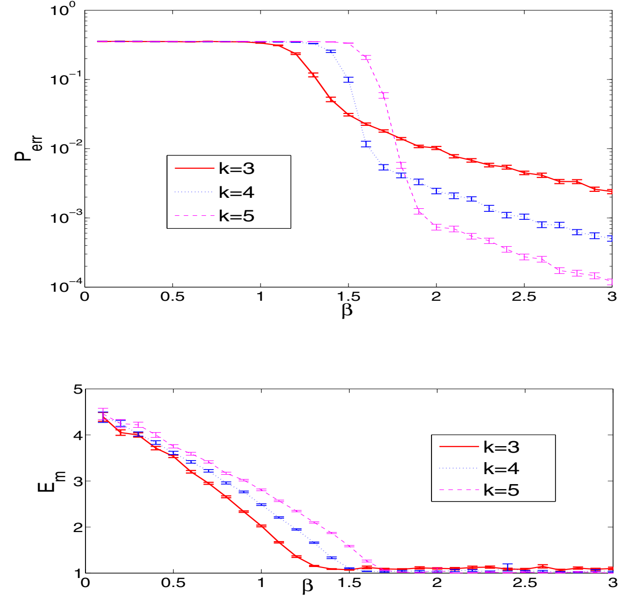

Fig. 9. Performance for different flow node degrees (k) against number of counters per flow (β), with 5% flow labels missing.

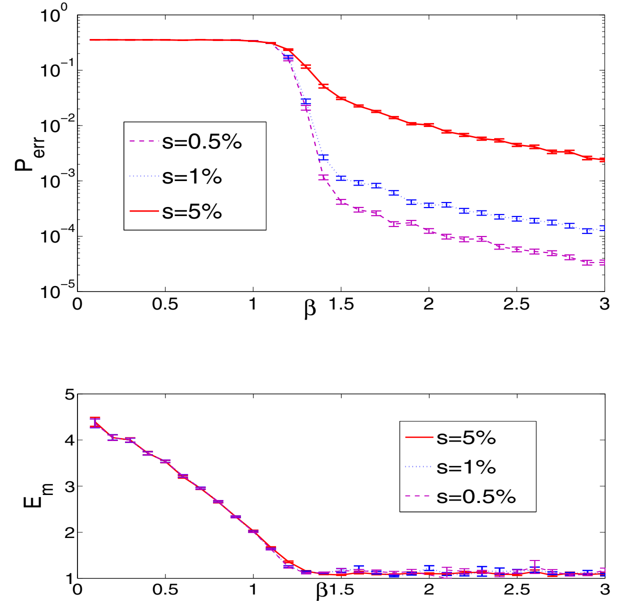

Fig. 10. Performance for different missing fraction (s) against number of counters per flow (β), with flow node degree $k = 3$.

deep and the layer-2 counters are 56 bits deep.

We report results for the entire duration of two hours. The average proportion of missing flows is 0.45% for these traces, and 80% of the missing flows have fewer than 3 packets. The reconstruction error $\mathrm{P_{err}}$ is the total number of incorrect estimates divided by the total number of flows with labels, and the average error magnitude E_m is the sum of absolute deviation of all incorrect estimates divided by the total number of incorrect estimates.

Table V presents the results with different sizes for CB. The total space for VLSC Bloom filters and CB is denoted by B.

B (MB)	2.6	2.7	2.8	2.9	3.6
$\mathrm{P_{err}}$	0.68	0.32	0.003	0.00015	6.9×10^{-5}
E_m	3	1.9	1.2	1.1	1.2

TABLE V
SIMULATION RESULTS OF COUNTING 2 TRACES IN 5-MINUTE SEGMENTS, WITH TOTAL SPACE B FOR CB AND BLOOM FILTERS.

We observe the same phenomenon of threshold and error floor. As we underprovide space, the reconstruction error increases significantly. However, the error magnitude remains small. For these two traces, 2.9 MB is sufficient to collect all but 0.45% flow labels, with an additional 0.015% decoded incorrectly. In contrast, exact counting with a d-left load balanced hash table requires 21-byte wide rows, and 1.5 times over-provisioning, totaling 28.4 MB for 0.9 million flows. Our solution hence provides a 10-fold reduction in SRAM space.

VI. CONCLUSION

We proposed a space-efficient approximate flow label collection scheme using VLSC Bloom filters and an error-resilient message passing algorithm that decodes flows sizes with vanishing error, despite missing flow labels. This enables the deployment of Counter Braids in scenarios where the flow label collection is best-effort. The error-resilient message passing algorithm also has interesting theoretical implications for sparse random graph codes and compressed sensing.

Acknowledgement: Support for OC-48 data collection is provided by DARPA, NSF, DHS, Cisco and CAIDA members. Yi Lu is supported by the Cisco Stanford Graduate Fellowship.

REFERENCES

[1] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Analysis of a statistics counter architecture," in *HotI 9*, Stanford, August 2001.
[2] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *ACM SIGCOMM*, 2002.
[3] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter Braids: A novel counter architecture for per-flow measurement," in *SIGMETRICS*, 2008.
[4] Y. Lu, B. Prabhakar, and F. Bonomi, "Bloom filters: Design innovations and novel applications," in *Allerton Conference*, 2005.
[5] M. Mitzenmacher and B. Vocking, "The asymptotics of selecting the shortest of two, improved," in *Harvard CS*, 1999, pp. 326–327.
[6] R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, Massachussetts, 1963.
[7] T. Richardson and R. Urbanke, *Modern Coding Theory*, Cambridge University Press, 2008.
[8] E. Candès and T. Tao, "Near optimal signal recovery from random projections and universal encoding strategies," *IEEE Trans. Inform. Theory*, 2004.
[9] G. Cormode and S. Muthukrishnan, "Combinatorial algorithms for compressed sensing," in *SIROCCO*, 2006.
[10] A. Gilbert, M. Strauss, J. Tropp, and R. Vershynin, "Algorithmic linear dimension reduction in the l1 norm for sparse vectors.," *submitted*, 2006.
[11] P. Indyk, "Explicit constructions for compressed sensing of sparse signals," in *19th Symposium on Discrete Algorithms*, 2008.
[12] E. J. Candes, J. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Comm. Pure Appl. Math.*, pp. 1207–1223, 2006.
[13] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Comm. ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[14] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Transaction on Networking*, vol. 8, no. 3, pp. 281–293, 2000.
[15] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," in *ACM SIGMETRICS*, 2004.
[16] S. Dharmapurikar and V. Paxson, "Robust TCP stream reassembly in the presence of adversaries," *14th USENIX Security Symposium*, 2005.

Appendix

Basic Bloom filter

A basic Bloom Filter is a m-bit vector B. Let U denote the universe of elements whose set membership we want to decide. Let S be a subset of U. We shall say that an element $x \in U$ is "valid" if $x \in S$; else, we shall say that x is "invalid."

Available to us are k hash functions $h_1(\cdot), ..., h_k(\cdot)$ that each map an $x \in U$ to a randomly chosen element of the set $\{e_1, ..., e_m\}$, where e_i is an m-bit vector with only its ith bit set to 1. Let $h(x)$ be the logical OR of $h_1(x), ..., h_k(x)$. We refer to $h(x)$ as the "signature" of x. For two binary numbers, a and b, of equal length, $a \ll b$ denotes that b has a 1 in each location where a has a 1. A basic Bloom Filter has the following two operations.

Training. Given S and $h(\cdot)$, the vector B is set to the logical OR of $h(x_1), ..., h(x_n)$, $S = \{x_1, x_2, \ldots, x_n\}$. Equally, the bits corresponding to the signatures of elements in S are set to 1 in the bitmap vector B.

Querying. To determine whether a $y \in U$ belongs to S, compute $h(y)$. If $h(y) \ll B$, declare that $y \in S$, else declare that $y \notin S$. Clearly, the declaration that $y \notin S$ can never be false; however, the declaration that $y \in S$ can be false sometimes.

Variable-length signature counting Bloom filter

A counting Bloom filter [14] is a Bloom filter where a multi-bit counter replaces each single bit in the bitmap. We made the signature variable-length [4] so that it is more robust with counter overflows, and requires a smaller counter depth. Let $q \leq k$. We outline the operations below.

Insertion. Same as **training** of a basic Bloom filter.

Querying. If q bits of $h(y)$ are set to 1 in B, declare $y \in B$.

Recover. If querying returns positive, one random missing bit of $h(y)$ (if any) is set to 1.

Deletion. If the departure of an element from the set S is explicitly known, all k bits $h_1(x)$,. . .,$h_k(x)$ are set to 0.

Aging. If the departure is not known, obsolete elements are aged out by setting bits to 0 in a round-robin manner.