Counter Braids: A Novel Counter Architecture for Per-Flow Measurement

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

Andrea Montanari
Departments of EE and Stats
Stanford University
montanar@stanford.edu

Balaji Prabhakar
Departments of EE and CS
Stanford University
balaji@stanford.edu

Sarang Dharmapurikar
Nuova Systems, Inc
San Jose, California
sarang@nuovasystems.com

Abdul Kabbani
Department of EE
Stanford University
akabbani@stanford.edu

ABSTRACT

Fine-grained network measurement requires routers and switches to update large arrays of counters at very high link speed (e.g. 40 Gbps). A naive algorithm needs an infeasible amount of SRAM to store both the counters and a flow-to-counter association rule, so that arriving packets can update corresponding counters at link speed. This has made accurate per-flow measurement complex and expensive, and motivated approximate methods that detect and measure only the large flows.

This paper revisits the problem of accurate per-flow measurement. We present a counter architecture, called Counter Braids, inspired by sparse random graph codes. In a nutshell, Counter Braids "compresses while counting". It solves the central problems (counter space and flow-to-counter association) of per-flow measurement by "braiding" a hierarchy of counters with random graphs. Braiding results in drastic space reduction by sharing counters among flows; and using random graphs generated on-the-fly with hash functions avoids the storage of flow-to-counter association.

The Counter Braids architecture is optimal (albeit with a complex decoder) as it achieves the maximum compression rate asymptotically. For implementation, we present a low-complexity message passing decoding algorithm, which can recover flow sizes with essentially *zero error*. Evaluation on Internet traces demonstrates that almost all flow sizes are recovered exactly with only a few bits of counter space per flow.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operations - Network Monitoring; E.1 [**Data Structures**]

SIGMETRICS'08, June 2–6, 2008, Annapolis, Maryland, USA.
Copyright 2008 ACM 978-1-60558-005-0/08/06 ...$5.00.

General Terms

Measurement, Algorithms, Theory, Performance

Keywords

Statistics Counters, Network Measurement, Message Passing Algorithms

1. INTRODUCTION

There is an increasing need for fine-grained network measurement to aid the management of large networks [14]. Network measurement consists of counting the size of a logical entity called "flow", at an interface such as a router. A flow is a sequence of packets that satisfy a common set of rules. For instance, packets with the same source (destination) address constitute a flow. Measuring flows of this type gives the volume of upload (download) by a user and is useful for accounting and billing purposes. Measuring flows with a specific flow 5-tuple in the packet header gives more detailed information such as routing distribution and types of traffic in the network. Such information can help greatly with traffic engineering and bandwidth provisioning. Flows can also be defined by packet classification. For example, ICMP Echo packets used for network attacks form a flow. Measuring such flows is useful during and after an attack for anomaly detection and network forensics.

Currently there exists no large-scale statistics counter architecture that is both cheap and accurate. This is mainly due to the lack of affordable high-density high-bandwidth memory devices. To illustrate the problem, the processing time for a 64-byte packet at a 40-Gbps OC-768 link is 12 ns. This requires memories with access time much smaller than that of commercially available DRAM (whose access time is tens of nsec), and makes it necessary to employ SRAMs. However, due to their low density, large SRAMs are expensive and difficult to implement on-chip. It is, therefore, essential to find a counter architecture that minimizes *memory space.* There are two main components of the total space requirement:

1. Counter space. Assuming that a million distinct flows are observed in an interval[1] and using one 64-bit counter

[1] Our OC-48 (2.5 Gbps) trace data show that are about $900,000$ distinct flow 5-tuples in a 5-minute interval. On 40-Gbps links, there can easily be an excess of a million dis-

per flow (a standard vendor practice [20]), 8 MB of SRAM is needed for counter space alone.

2. Flow-to-counter association rule. The set of active flows varies over time, and the flow-to-counter association rule needs to be dynamically constructed. For a small number of flows, a content-addressable-memory (CAM) is used in most applications. However, the high power consumption and heat dissipation of CAMs forbid their use in realistic scenarios, and SRAM hash tables are used to store the flow-to-counter association rule. This requires at least another 10 MB of SRAM.

The large space requirement not only considerably increases the cost of line cards, but also hinders a compact layout of chips due to the low density of SRAM.

1.1 Previous Approaches

The wide applicability and inherent difficulty of designing statistics counters have attracted the attention of the research community. There are two main approaches: (i) Exact counting using a hybrid SRAM-DRAM architecture, and (ii) approximate counting by exploiting the heavy-tail nature of flow size distribution. We review these approaches below.

Exact counting. Shah et. al. [22] proposed and analyzed a hybrid architecture, taking the first step towards an implementable large-scale counter array. The architecture consists of shallow counters in fast SRAM and deep counters in slow DRAM. The challenge is to find a simple algorithm for updating the DRAM counters so that no SRAM counter overflows in between two DRAM updates. The algorithm analyzed in [22] was subsequently improved by Ramabhadran and Varghese [20] and Zhao et. al. [23]. This reduced the algorithm complexity, making it feasible to use a small SRAM with 5 bits per flow to count flow sizes in packets (not bytes). However, all the papers above suffer from the following drawbacks: (i) deep (typically 64 bits per flow) off-chip DRAM counters are needed, (ii) costly SRAM-to-DRAM updates are required, and (iii) the flow-to-counter association problem is assumed to be solved using a CAM or a hash table. In particular, they do not address the flow-to-counter association problem.

Approximate counting. To keep cost acceptable, practical solutions from the industry and academic research either sacrifice the accuracy or limit the scope of measurement. For example, Cisco's Netflow [1] counts both 5-tuples and per-prefix flows based on sampling, which introduces a significant 9% relative error even for large flows and more errors for smaller flows [12]. Juniper Networks introduced filter-based accounting [2] to count a limited set of flows predefined manually by operators. The "sample-and-hold" solution proposed by Estan and Varghese in [12], while achieving high accuracy, measures only flows that occupy more than 0.1% of the total bandwidth. Estan and Varghese's approach introduced the idea of exploiting the heavy-tail flow size distribution: since a few large flows bring most of the data, it is feasible to quickly identify these large flows and measure their sizes only.

1.2 Our Approach

The main contribution of this paper is an SRAM-only large-scale counter architecture with the following features:

1. Flow-to-counter association using a small number (e.g. 3) of hash functions.

2. Incremental compression of flow sizes as packets arrive; only a small number (e.g. 3) of counters are accessed at each packet arrival.

3. Asymptotic optimality. We have proved in [17] that Counter Braids (CB), with an optimal (but NP-hard) decoder, has an asymptotic compression rate matching the information theoretic limit. The result is surprising since CB forms a restrictive family of compressors.

4. A linear-complexity message passing decoding algorithm that recovers all flow sizes from compressed counts with essentially *zero error*. Total space in CB needed for exact recovery is close to the optimal compression of flow sizes.

5. The message passing algorithm is analyzable, enabling the choice of design parameters for different hardware requirement.

Remark: We note that CB has the disadvantage of not supporting instantaneous queries of flow sizes. All flow sizes are decoded together at the end of a measurement epoch. We plan to address this problem in future work.

Informal description. Counter Braids is a hierarchy of counters braided via random graphs in tandem. Figure 1(a) shows a naive counter architecture that stores five flow sizes in counters of equal depth, which has to exceed the size of the largest flow. Each bit in a counter is shown as a circle. The least significant bit (LSB) is the one closest to the flow node. Filled circles represent a 1, and unfilled circles a 0. This structure leads to an enormous wastage of space because the majority of flows are small.

Figure 1(b) shows CB for storing the same flow sizes. It is worth noting that: (i) CB has fewer "more significant bits" and they are shared among all flows, and (ii) the exact flow sizes can be obtained by "decoding" the bit patten stored in CB. A comparison of the two figures clearly shows a great reduction in space.

1.3 Related Theoretical Literature

Compressed Sensing. The idea of Counter Braids is thematically related to *compressed sensing* [6, 11], whose central innovation is summarized by the following quote:

Since we can "throw away" most of our data and still be able to reconstruct the original with no perceptual loss (as we do with ubiquitous sound, image and data compression formats,)why can't we **directly measure** *the part that will not end up being "thrown away"?* [11]

For the network measurement problem, we obtain a vector of counter values, **c**, via CB, from the flow sizes **f**. If **f** has a small entropy, the vector **c** occupies much less space than **f**; it constitutes "the part (of **f**) that will not end up being thrown away." An off-chip decoding algorithm then recovers **f** from **c**. While Compressed Sensing and CB are

tinct flow 5-tuples in a short observation interval. Or, for measuring the frequency of prefix accesses, one needs about 500,000 counters, which is the current size of IPv4 routing tables [20]. Future routers may easily support more than a million prefixes.

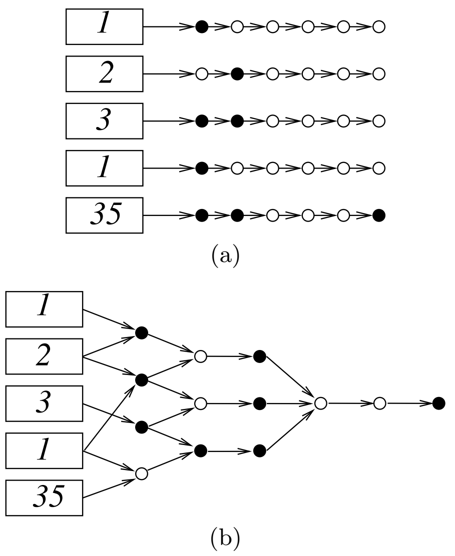

Figure 1: (a) A simple counter structure. (b) Counter Braids. (filled circle = 1, unfilled circle = 0).

thematically related, they are methodologically quite different: Compressed Sensing computes random linear transformations of the data and uses LP (linear programming) reconstruction methods; whereas CB uses a multi-layered non-linear structure and a message passing reconstruction algorithm.

Sparse random graph codes. Counter Braids is methodologically inspired by the theory of low-density parity check (LDPC) codes[13, 21]. See also related literatures on Tornado codes[18] and Fountain codes[4]. From the information theoretic perspective, the design of an efficient counting scheme and a good flow size estimation is equivalent to the design of an efficient *compressor*, or a *source code* [8]. However, the network measurement problem imposes a stringent constraint on such a code: each time the size of a flow changes (because a new packet arrives), a small number of operations must be sufficient to update the compressed information. This is not the case with standard source codes (such as the Lempel-Ziv algorithm), where changing a single bit in the source stream may completely alter the compressed version. We find that the class of source codes dual to LDPC codes [5] work well under this constraint; using features of these codes makes CB a good "incremental compressor."

There is a problem in using the design of LDPC codes for network measurement: with the heavy-tailed distribution, the flow sizes are a priori unbounded. In the channel coding language, this is equivalent to using a countable but infinite input alphabet. As a result, new ideas are developed for proving the achievability of optimal asymptotic compression rate. The full proof is contained in [17] and we state the theorem in the appendix for completeness.

The large alphabet size also makes iterative message passing decoding algorithms [15], such as Belief Propagation, highly complex to implement, as BP passes probabilities rather than numbers. In this paper, we present a novel message passing decoding algorithm of low complexity that is easy to implement. The sub-optimality of the message passing algorithm naturally requires more counter space than the information theoretic limit. We characterize the minimum space required for zero asymptotic decoding error using "density evolution" [21]. The space requirement can be further optimized with respect to the number of layers in Counter Braids, and the degree distribution of each layer. The optimized space is close to the information theoretic limit, enabling CB to fit into small SRAM.

Count-Min Sketch. Like Counter Braids, the Count-Min sketch [7] for data stream applications is also a random hash-based structure. With Count-Min, each flow hashes to and updates d counters; the minimum value of the d counters is retrieved as the flow estimate. The Count-Min sketch provides probabilistic guarantees for the estimation error: with at least $1 - \delta$ probability, the estimation error is less than $\epsilon|\mathbf{f}|_1$, where $|\mathbf{f}|_1$ is the sum of all flow sizes. To have small δ and ϵ, the number of counters needs to be large.

The Count-Min sketch is different from Counter Braids in the following ways: (a) There is no "braiding" of counters, hence no compression. (b) The estimation algorithm for the Count-Min sketch is one-step, whereas it is iterative for CB. In fact, comparing the Count-Min algorithm to our reconstruction algorithm on a one-layer CB, it is easy to see that the estimate by Count-Min is exactly the estimate after *the first* iteration of our algorithm. Thus, CB performs at least as well as the Count-Min algorithm.[2] (c) Our reconstruction algorithm *detects errors.* That is, it can distinguish the flows whose sizes are incorrectly estimated, and produce an upper and lower bound of the true value; whereas the Count-Min sketch only guarantees an over-estimate. (d) CB needs to decode all the flow sizes at once, unlike the Count-Min algorithm which can estimate a single flow size. Thus, Count-Min is better at handling online queries than CB.

Structurally related to Counter Braids (random hashing of flows into counters and a recovery algorithm) is the work of Kumar et. al. [16]. The goal of that work is to estimate the *flow size distribution* and not the actual flow sizes, which is our aim.

In Section 2, we define the goals of this paper and outline our solution methodology. Section 3 describes the Counter Braids architecture. The message passing decoding algorithm is described in Section 4 and analyzed in Section 5. Section 6 explores the choice of parameters for multi-layer CB. The algorithm is evaluated using traces in Section 7. We discuss implementation issues in Section 8 and outline further work in Section 9.

2. PROBLEM FORMULATION

We divide time into measurement epochs (e.g. 5 minutes). The objective is to count the number of packets per flow for all active flows within a measurement epoch. We do not deal with the byte-counting problem in this paper due to space limitation, but there is no constraint in using Counter Braids for byte-counting.

Goals: As mentioned in Section 1, the main problems we wish to address are: (i) compacting (or eliminating) the space used by flow-to-counter association rule, and (ii) saving counter space and incrementally compressing the counts.

[2] This is similar to the benefit of Turbo codes over conventional soft-decision decoding algorithms and illustrates the power of the "Turbo principle."

Additionally, we would like (iii) a low-complexity algorithm to reconstruct flow sizes at the end of a measurement epoch.

Solution methodology: Corresponding to the goals, we (i) use a small number of hash functions, (ii) braid the counters, and (iii) use a linear-complexity message-passing algorithm to reconstruct flow sizes. In particular, by using a small number of hash functions, we *eliminate* the need for storing a flow-to-counter association rule.

Performance measures:
(1) Space: measured in number of *bits per flow* occupied by counters. We denote it by r (to suggest *compression rate* as in the information theory literature.) Note that the number of counters is *not* the correct measure of compression rate; rather, it is the number of bits.
(2) Reconstruction error: measured as the fraction of flows whose reconstructed value is *different* from the true value:

$$\mathrm{P}_{\mathrm{err}} \equiv \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{\widehat{f_i} \neq f_i\}\,,$$

where n is the total number of flows, $\widehat{f_i}$ is the estimated size of flow i and f_i the true size. $\mathbb{I}$ is the indicator function, which returns 1 if the expression in the bracket is true and 0 otherwise. We chose this metric since we want exact reconstruction.
(3) Average error magnitude: defined as the ratio of the sum of absolute errors and the number of errors:

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

It measures how big an error is when an error has occurred.

The statement of asymptotic optimality in the appendix yields that it is possible to keep *space* equal to the flow-size entropy, and have *reconstruction error* going to 0 as the number of flows goes to infinity.

Both analysis (Section 5) and simulations (Section 7) show that with our low-complexity message passing decoding algorithm, we can keep *space* close to the flow-size entropy and obtain essentially *zero reconstruction error.* In addition, the algorithm offers a gracious degradation of error when space is further reduced, even below the flow-size entropy. Although reconstruction error becomes significant, average error magnitude remains small, which means that most flow-size estimates are close to their true values.

3. OUR SOLUTION

The overall architecture of our solution is shown in Figure 2. Each arriving packet updates Counter Braids in on-chip SRAM. This constitutes the encoding stage if we view measurement as compression. At the end of a measurement epoch, the content of Counter Braids, i.e., the compressed counts, are transferred to an offline processing unit, such as a PC. A reconstruction algorithm then recovers the list of <flow ID, size> pairs.

We describe CB in Section 3.1 and specify the mapping that solves the flow-to-counter association problem in Section 3.2. We describe the updating scheme, or the on-chip encoding algorithm, in Section 3.3, leaving the description of the reconstruction algorithm to Section 4.

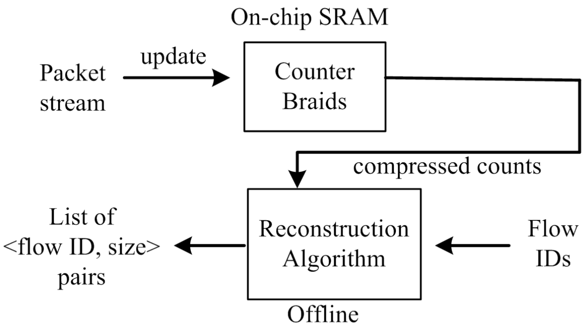

Figure 2: System Diagram.

3.1 Counter Braids

Counter Braids has a *layered* structure. The l-th layer has m_l counters with a depth of d_l bits. Let the total number of layers be L. In practice, $L = 2$ is usually sufficient as will be shown in Section 6. Figure 3 illustrates the case where $L = 2$. For a complete description of the structure, we leave L as a parameter.

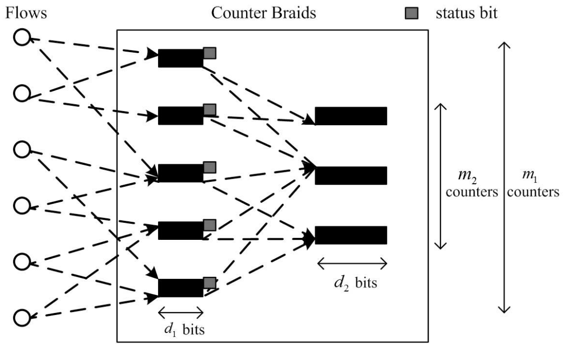

Figure 3: Two-layer Counter Braids with two hash functions and status bits.

We will show in later sections that we can use a decreasing number of counters in each layer of CB, and still be able to recover the flow sizes correctly. The idea is that given a heavy-tail distribution for flow sizes, the more significant bits in the counters are poorly utilized; since braiding allows more significant bits to be shared among all flows, a reduced number of counters in the higher layers suffice.

Figure 3 also shows an optional feature of CB, the status bits. A status bit is an additional bit on a first-layer counter. It is set to 1 after the corresponding counter first overflows. Counter Braids without status bits is theoretically sufficient: the asymptotic optimality result in the appendix is shown *without* status bits, assuming a high-complexity optimal decoder. However, in practice we use a low-complexity message passing decoder, and the particular shape of the network traffic distribution is better exploited with status bits.

Status bits occupy additional space, but provide useful information to the message-passing decoder so that the number of second-layer counters can be further reduced, yielding a favorable tradeoff in space. Status bits are taken into account when computing the total space; in particular, it figures in the performance measure, r, "space in number of bits per flow." In CB with more than two layers, every layer except the last will have counters with status bits.

3.2 The Random (Hash) Mappings

We use the same random mapping in two settings: (i) between flows and the first-layer counters, and (ii) between two consecutive layers of counters. The dashed arrows in Figure 3 illustrate both (i) and (ii) (which is between the first and second layer counters.)

Consider the random mapping between flows and the layer-1 counters. For each flow ID, we apply k pseudo-random hash functions with a common range $\{0, \cdots, m_1 - 1\}$, where m_1 is the number of counters in layer 1, as illustrated in Figure 3 (with $k = 2$.) The mapping has the following features:

1. It is dynamically constructed for a varying set of active flows, by applying hash functions to flow IDs. In other words, no memory space is needed to describe the mapping explicitly.

   The storage for the flow-to-counter association is simply the size of description of the k hash functions and does not increase with the number of flows n.

2. The number of hash functions k is set to a small constant (e.g. 3). This allows counters to be updated with only a small number of operations at a packet arrival.

Remark. Note that the mapping does not have any special structure. In particular, it is not bijective. This necessitates the use of a reconstruction algorithm to recover the flow sizes. Using $k > 1$ adds redundancy to the mapping and makes recovery possible. However, the random mapping does *more* than simplifying the flow-to-counter association. In fact, it **performs the compression** of flow sizes into counter values and reduces counter space.

Next consider the random mapping between two consecutive layers of counters. For each counter location (in the range $\{0, \cdots, m_l - 1\}$) in the l-th layer, we apply k hash functions to obtain the corresponding $(l+1)$-th layer counter locations (in the range $\{0, \cdots, m_{l+1} - 1\}$). It is illustrated in Figure 3 with $k = 2$. The use of hash functions enables us to implement the mapping without extra circuits in the hardware; and the random mapping further *compresses* the counts in layer-2 counters.

3.3 Encoding: The Updating Algorithm

The initialization and update procedures of a two-layer Counter Braids with 2 hash functions at each layer are specified in Exhibit 1. The procedures include both the generation of random mapping using hash functions and the updating scheme. When a packet arrives, *both* counters its flow label hashes into are incremented. And when a counter in layer 1 overflows, both counters in layer 2 it hashes into are incremented by 1, like a carry-over. The overflowing counter is reset to 0 and the corresponding status bit is set to 1.

It is evident from the exhibit that the amount of updating required is very small. Yet after each update, the counters store a compressed version of the most up-to-date flow sizes. The incremental nature of this compression algorithm is made possible with the use of random sparse linear codes, which we shall further exploit at the reconstruction stage.

Exhibit 1: The Update Algorithm

```
Initialize
    for layer $l = 1$ to 2
        for counter $i = 1$ to $m_l$
            $counters[l][i] = 0$

Update
    Upon the arrival of a packet $pkt$
        $idx1$ = hash-function1($pkt$);
        $idx2$ = hash-function2($pkt$);
        $counters[1][idx1] = counter[1][idx1] + 1$;
        $counters[1][idx2] = counter[1][idx2] + 1$;
        if $counters[1][idx1]$ overflows,
            Update second-layer counters ($idx1$);
        if $counters[1][idx2]$ overflows,
            Update second-layer counters ($idx2$)

Update second-layer counters ($idx$)
    $statusbit[1][idx] = 1$;
    $idx3$ = hash-function3($idx$);
    $idx4$ = hash-function4($idx$);
    $counters[2][idx3] = counter[2][idx3] + 1$;
    $counters[2][idx4] = counter[2][idx4] + 1$
```

The update of the second-layer counters can be pipelined. It can be executed together with the next update of the first-layer counters. In general, pipelining can be used for CB with multiple layers.

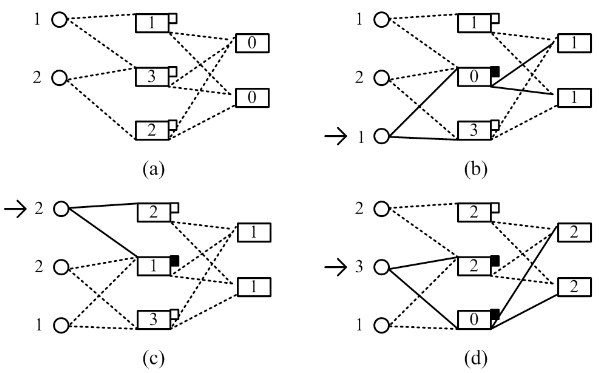

Figure 4: A toy example for updating. Numbers next to flow nodes are current flow sizes. Dotted lines indicate hash functions. Thick lines indicate hash functions being computed by an arriving packet. The flow with an arriving packet is indicated by an arrow.

Figure 4 illustrates the updating algorithm with a toy example. (a) shows the initial state of CB with two flows. In (b), a new flow arrives, bringing the first packet; a layer-1 counter overflows and updates two layer-2 counters. In (c), a packet of an existing flow arrives and no overflow occurs. In (d), another packet of an existing flow arrives and another layer-1 counter overflows.

4. MESSAGE PASSING DECODER

The sparsity of the random graphs[3] in CB opens the way to using low-complexity message passing algorithms for reconstruction of flow sizes, but the design of such an algorithm is not obvious. In the case of LDPC codes, message passing decoding algorithms hold the promise of approaching capacity with unprecedentedly low complexity. However, the algorithms used in coding, such as Belief Propagation, have increasing memory requirement as the alphabet size grows, since BP passes probability distributions instead of single numbers. We develop a novel message passing algorithm that is simple to implement on countable alphabets.

4.1 One Layer

Consider the random mapping between flows and the first-layer counters. It is a bipartite graph with flow nodes on the left and counter nodes on the right, as shown in Figure 5. An edge connects flow i and counter a if one of the k hash functions maps flow i to counter a. The vector $\mathbf{f}$ denotes flow sizes and $\mathbf{c}$ denotes counter values.

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

where ∂a denotes all the flows that hash into counter a. The problem is to estimate $\mathbf{f}$ given $\mathbf{c}$.

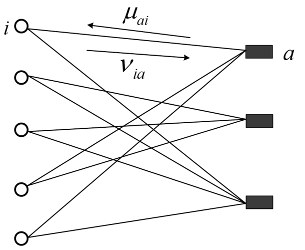

Figure 5: Message passing on a bipartite graph with flow nodes (circles) and counter nodes (rectangles.)

Message passing algorithms are iterative. In the t^{th} iteration messages are passed from all flow nodes to all counter nodes and then back in the reverse direction. A message goes from flow i to counter a (denoted by ν_{ia}) and vice versa (denoted by μ_{ai}) only if nodes i and a are neighbors (connected by an edge) on the bipartite graph.

Our algorithm is described in Exhibit 2. The messages $\nu_{ia}(0)$ are initialized to 0, although any initial value less than the minimum flow size, *min*, will work just as well. 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*.

Remark 1. Since $\nu_{ia}(0) = 0$,

$$\mu_{ai}(1) = c_a \quad \text{and} \quad \widehat{f}_i(1) = \min_a \{c_a\},$$

which is precisely the estimate of the Count-Min algorithm. Thus, the estimate of Count-Min is the estimate of our message-passing algorithm after the *first iteration.*

Exhibit 2: The Message Passing Decoding Algorithm

```
1:  Initialize
2:     $min$ = minimum 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:        $\nu_{ia}(t) = \begin{cases} \min_{b \neq a} \mu_{bi}(t) & \text{if } t \text{ is odd,} \\ \max_{b \neq a} \mu_{bi}(t) & \text{if } t \text{ is even.} \end{cases}$

9:  Final Estimate
10:    $\widehat{f}_i(T) = \begin{cases} \min_a\{\mu_{ai}(T)\} & \text{if } T \text{ is odd,} \\ \max_a\{\mu_{ai}(T)\} & \text{if } T \text{ is even.} \end{cases}$
```

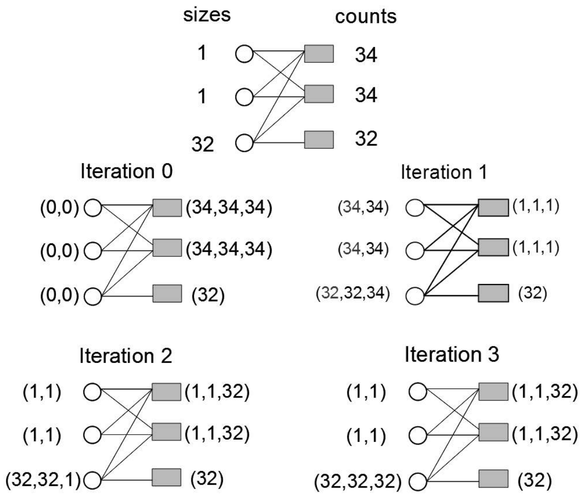

Figure 6: The decoding algorithm over 4 iterations. Numbers in the topmost figure are true flow sizes and counter values. In an iteration, numbers next to a node are messages on its outgoing edges, from top to bottom. Each iteration involves messages going from flows to counters and back from counters to flows.

Remark 2. The distinction between odd and even iterations at line 8 and 10 is due to the "anti-monotonicity property" of the message-passing algorithm, to be discussed in Section 5.

Remark 3. It turns out that the algorithm remains unchanged if the minimum or maximum at line 8 is over *all* incoming messages, that is,

$$\nu_{ia}(t) = \begin{cases} \min_b \mu_{bi}(t) & \text{if } t \text{ is odd,} \\ \max_b \mu_{bi}(t) & \text{if } t \text{ is even.} \end{cases}$$

The change will save some computations in implementation. The proof of this fact and ensuing analytical consequences is deferred to forthcoming publications. In this paper, we stick to the algorithm in Exhibit 2.

[3] Each random mapping in CB is a random bipartite graph with edges generated by the k hash functions. It is sparse because the number of edges is *linear* in the number of nodes, as opposed to quadratic for a complete bipartite graph.

Toy example. Figure 6 shows the evolution of messages over 4 iterations on a toy example. In this particular example, all flow sizes are reconstructed correctly. Note that we are using different degrees at some flow nodes. In general, this gives potentially better performance than all flow nodes having the same degree, but we will stick to the latter in this paper for its ease of implementation.

The flow estimates at each iteration are listed in Table 1. All messages converge in 4 iterations and the estimates at Iteration 1 (second column) is the Count-Min estimate.

iteration	0	1	2	3	4
$\widehat{f}_1$	0	34	1	1	1
$\widehat{f}_2$	0	34	1	1	1
$\widehat{f}_3$	0	32	32	32	32

Table 1: Flow estimates at each iteration. All messages converge after Iteration 3.

4.2 Multi-layer

Multi-layer Counter Braids are decoded recursively, one layer at a time. It is *conceptually* helpful to construct a new set of flows $\mathbf{f}_l$ for layer-l counters based on the counter values at layer $(l-1)$. The presence of status bits affects the definition of $\mathbf{f}_l$.

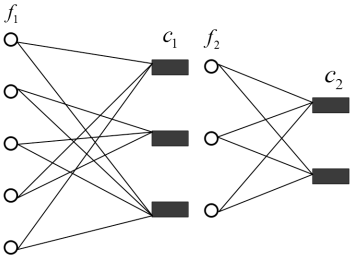

Figure 7: Without status bits, flows in $\mathbf{f}_2$ have a one-to-one map to all counter in $\mathbf{c}_1$.

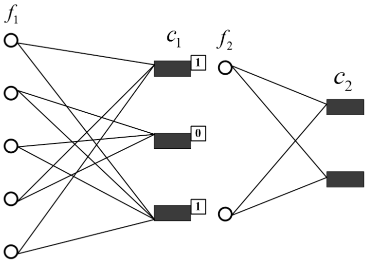

Figure 8: With status bits, flows in $\mathbf{f}_2$ have a one-to-one map to only counters that have overflown (whose status bits are set to 1).

Figure 7 illustrates the construction of $\mathbf{f}_2$ when there are no status bits. The vector $\mathbf{f}_2$ has a one-to-one map to counters in layer 1, and a flow size in $\mathbf{f}_2$ equals the number of times the corresponding counter has overflown, with the minimum value 0.

Figure 8 illustrates the construction of $\mathbf{f}_2$ when there are status bits. The vector $\mathbf{f}_2$ now has a one-to-one correspondence with only those counters in layer 1 that have *overflown;* that is, counters whose status bits are set to 1. The new flow size is still the number of times the corresponding counter overflows, but in this case, the minimum value is 1. It is clear from the figure that the use of status bits effectively reduces the number of flow nodes in layer 2. Hence, fewer counters are needed in layer 2 for good decodability. This reduction in counter space at layer 2 trades off with the additional space needed for status bits themselves! As we shall see in Section 6, when the number of layers in CB is small, the tradeoff favors the use of status bits.

The flow sizes are decoded recursively, starting from the topmost layer. For example, after decoding the layer-2 "flows," we add their sizes (the amount of overflow from layer-1 counters) to the values of layer-1 counters. We then use the new values of layer-1 counters to decode the flow sizes. Details of the algorithm are presented in Exhibit 3.

Exhibit 3: The Multi-layer Algorithm

```
for $l = L$ to 1
        construct the graph for $l_{th}$ layer
        as in Figure 7 if without status bits;
        as in Figure 8 if with status bits;
        decode $\mathbf{f}_l$ from $\mathbf{c}_l$ as in Exhibit 2
        $\mathbf{c}_{l-1} = \mathbf{c}_{l-1} + \mathbf{f}_l \times 2^{d_{l-1}}$
        where $d_{l-1}$ is the counter depth in bits
        at layer $(l-1)$
```

5. SINGLE-LAYER ANALYSIS

The decoding algorithm works one layer at a time; hence, we first analyze the single-layer message passing algorithm and determine its rate r and reconstruction error probability $\mathrm{P_{err}}$. This analysis lays the foundation for the design of multi-layer Counter Braids, to be presented in Section 6. Since all counters in layer 1 have the same depth d_1, a very relevant quantity for the analysis is the number of counters per flow:

$$\beta \equiv m/n,$$

where m is the number of counters and n is the number of flows. The compression rate in bits per flow is given by $r = \beta d_1$. The bipartite graph in Figure 5 will be the focus of study, as its properties determine the performance of the algorithm.

LEMMA 1. **Toggling Property.** *If $\nu_{ia}(t-1) \le f_i$ for every i and a, then $\mu_{ai}(t) \ge f_i$ and $\nu_{ia}(t) \ge f_i$. Conversely, if $\nu_{ia}(t-1) \ge f_i$ for every i and a, then $\mu_{ai}(t) \le f_i$ and $\nu_{ia}(t) \le f_i$.*

The proof of this lemma follows simply from the definition of ν and μ and is omitted.

LEMMA 2. **Anti-monotonicity Property.** *If ν and ν' are such that for every i and a, $\nu_{ia}(t-1) \le \nu'_{ia}(t-1) \le f_i$, then $\nu_{ia}(t) \ge \nu'_{ia}(t) \ge f_i$. Consequently, since $\widehat{\mathbf{f}}(0) = 0$, $\widehat{\mathbf{f}}(2t) \le \mathbf{f}$ component-wise and $\widehat{\mathbf{f}}(2t)$ is component-wise non-decreasing. Similarly $\widehat{\mathbf{f}}(2t+1) \ge \mathbf{f}$ and is component-wise non-increasing.*

Proof. It follows from line 7 of Exhibit 2 that, if $\nu_{ia}(t-1) \leq \nu'_{ia}(t-1) \leq f_i$, then $\mu_{ai}(t) \geq \mu'_{ai}(t) \geq f_i$.[4] From this and the definitions of ν and $\widehat{f}$ at lines 8 and 10 of Exhibit 2, the rest of the lemma follows. ■

The above lemmas give a powerful conclusion: The true value of the flow-size vector is sandwiched between monotonically increasing lower bounds and monotonically decreasing upper bounds. The question, therefore, is:

Convergence: When does the sandwich close? That is, under what conditions does the message passing algorithm converge?

We give two answers. The first is general, not requiring any knowledge of the flow-size distribution. The second uses the flow-size distribution, but gives a much better answer. Indeed, one obtains an exact threshold for the convergence of the algorithm: For $\beta > \beta^*$ the algorithm converges, and for $\beta < \beta^*$ it fails to converge (i.e. the sandwich does not close.)

5.1 Message Passing on Trees

DEFINITION 1. *A graph is a forest if for all nodes in the graph, there exists no path of non-vanishing length that starts and ends at the same node. In other words, the graph contains no loops. Such a graph is a tree if it is connected.*

FACT 1. *Consider a bipartite graph with n flow nodes and $m = \beta n$ counters nodes, where each flow node connects to k uniformly sampled counter nodes. It is a forest with high probability iff $\beta \geq k(k-1)$ [19].*

Assume the bipartite graph is a forest. Since the flow nodes have degree $k > 1$, the leaves of the trees have to be counter nodes.

THEOREM 1. *For any flow node i belonging to a tree component in the bipartite graph, the message passing algorithm converges to the correct flow estimates after a finite number of iterations. In other words, for every a, $\mu_{ai}(t)$, $\nu_{ia}(t)$ and $\widehat{f}_i(t)$ all coincide with f_i for all t large enough.*

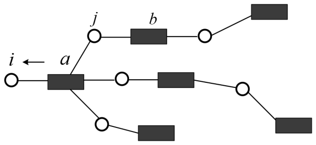

Figure 9: *The tree T_{ai} rooted at the directed edge $a \rightarrow i$. Its depth is $D_{ai} = 2$.*

Proof For simplicity we prove convergence for $\mu_{ai}(t)$, as the convergence of other quantities easily follows. Given the directed edge $a \rightarrow i$, consider the subtree T_{ai} rooted at $a \rightarrow i$ obtained by cutting all the counter nodes adjacent to i but a, cf. Figure 9. Clearly $\mu_{ai}(t)$ only depends on the counter values inside T_{ai}, and we restrict our attention to this subtree. Let D_{ai} denote the depth of T_{ai}. We shall prove by induction on D_{ai} that $\mu_{ai}(t) = f_i$ for any $t \geq D_{ai}$.

If $D_{ai} = 1$, this is trivially true: at any time $\mu_{ai}(t) = c_a$ and since $c_a = f_i$, the thesis follows.

Assume now that the thesis holds for all depths up to D, and consider $D_{ai} = D + 1$. Let j be one of the flows in T_{ai} that hashes to counter a, and let b denote one of the other counters to which it contributes, cf. Figure 9. Since the depth of the subtree T_{bj} is at most D, by the induction hypothesis, $\mu_{bj}(t) = f_j$ for any $t \geq D$. Consider now $t \geq D+1$. From the messages defined in Exhibit 2 and the previous observation, it follows that $\mu_{ai}(t) = c_a - \sum_{j \neq i} f_j = f_i$ as claimed. ■

Unfortunately, the use of the above theorem for CB requires $\beta \geq k(k-1)$, which leads to an enormous wastage of counters. We will now assume knowledge of the flow-size distribution and dramatically reduce β. We will work with sparse random graphs that are not forests, but rather they will have a *locally tree-like* structure.

5.2 Sparse Random Graph

It turns out that we are able to characterize the reconstruction error probability at t-th iteration of the algorithm more precisely. A nice observation enables us to use the idea of *density evolution*, developed in coding theory [21], to compute the error probability recursively in the large n limit. Due to space limitation, we are unable to fully describe the ideas of this section. We will be content to state the main theorem and make some useful remarks.

Consider a bipartite graph with n flow nodes and $m = \beta n$ counter nodes, where each flow node connects to k uniformly sampled counter nodes. Let

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

where $\gamma = nk/m$ is the average degree of a counter node. The degree distribution of a counter node converges to a Poisson distribution as $n \rightarrow \infty$, and $\rho_\gamma(x)$ is the generating function for the Poisson distribution.

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

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

and

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

THEOREM 2. **The Threshold.** *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.[5]

[4] Note that we implicitly assume that t is odd to be consistent with the definition of $\nu(\cdot)$ at line 8.

[5] In the event of $\widehat{f}_i(2t) < \widehat{f}_i(2t+1)$, we know that an error has occurred. Moreover, $\widehat{f}_i(2t)$ lower bounds and $\widehat{f}_i(2t+1)$ upper bounds the true value f_i.

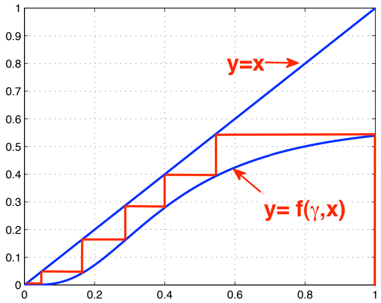

Figure 10: Density evolution as a walk between two curves.

Remark 1. The characterization of the threshold β^* largely depends on the *locally treelike* structure of the sparse random graph. More precisely, it means that the graph contains no finite-length loops as $n \to \infty$. Based on this, the *density evolution* principle recursively computes the error probability after a *finite* number of iterations, during which *all incoming messages at any node are independent.* With some observations specific to this algorithm, we obtain $f(\gamma, x)$ as the recursion.

Remark 2. The definition of γ^* can be understood visually using Figure 10. The recursive computation of error probability[6] corresponds to a walk between the curve $y = f(\gamma, x)$ and the line $y = x$, where two iterations (even and odd) correspond to one step. If $\gamma < \gamma^*$, $y = f(\gamma, x)$ is below $y = x$, and the walk continues all the way to 0, cf. Figure 10. This means that the reconstruction error is 0. If $\gamma > \gamma^*$, $y = f(\gamma, x)$ intersects $y = x$ at points above 0, and the walk ends at a non-zero intersection point. This means that there is a positive error for any number of iterations.

Remark 3. The minimum value of $\beta^* = \sqrt{\epsilon}$ can be obtained after optimizing over all degree distributions, including irregular ones. For the specific bipartite graph in CB, where flow nodes have regular degree k and counter nodes have Poisson distributed degrees, we obtain

$$\gamma^* = \frac{1}{\sqrt{\epsilon}}, \ \beta^* = 2\sqrt{\epsilon},$$

for $k = 2$. The values of γ^* and β^* for different k are listed in Table 2 for $\mathbb{P}(f_i > x) = x^{-1.5}$. The optimum value $\sqrt{\epsilon} = 0.595$ in this case. The value $k = 3$ achieves the lowest β^* among $2 \leq k \leq 7$, which is 18% more than the optimum.

k	2	3	4	5	6	7
γ^*	1.69	4.23	5.41	6.21	6.82	7.32
β^*	1.18	0.71	0.74	0.80	0.88	0.96

Table 2: Single-layer rate for $2 \leq k \leq 7$. $\mathbb{P}(f_i > x) = x^{-1.5}$.

6. MULTI-LAYER DESIGN

Given a specific flow size distribution (or an upper bound on the empirical tail distribution), we have a general algorithm that optimizes the number of bits per flow in Counter Braids over the following parameters: (1) number of layers, (2)number of hash functions in each layer, (3) depth of counters in each layer and (4) the use of status bits. We present below the results from the optimization.

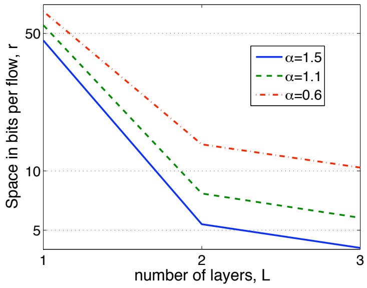

Figure 11: Optimized space against number of layers.

(i) Two layers are usually sufficient.
Figure 11 shows the decrease of total space (number of bits per flow) as the number of layers increases, for power-law distributions $\mathbb{P}(f_i > x) = x^{-\alpha}$ with $\alpha = 1.5$, 1.1 and 0.6. For distributions with relatively light tails, such as $\alpha = 1.5$ or 1.1, two layers accomplish the major part of space reduction; whereas for heavier tails, such as $\alpha = 0.6$, three layers help reduce space further.

Note that the distribution with $\alpha = 0.6$ has very heavy tails. For instance, the flow distributions from real Internet traces, such as those plotted in [16], has $\alpha \approx 2$. Hence two layers suffice for most network traffic.

(ii) 3 hash functions is optimal for two-layer CB.
We optimized total space over the number of hash functions in each layer for a two-layer CB. Using 3 hash functions in both layers achieves the minimum space. Fixing $k = 3$ and using the traffic distribution, we can find β^* according to Theorem 2. The number of counters in layer 1 is $m_1 = \beta^* n$, where n is the number of flows.

(iii) Layer-1 counter depth and number of layer-2 counters.
There is a tradeoff between the depth of layer-1 counters and the number of layer-2 counters, since shallow layer-1 counters overflow more often. For most network traffic with $\alpha \geq 1.1$, 4 or 5 bits in layer 1 suffice. For distributions with heavier tails, such as $\alpha = 1$, the optimal depth is 7 to 8 bits. Since layer-2 counters are much deeper than layer-1 counters, it is usually favorable to have at least one order fewer counters in layer 2.

(iv) Status bits are helpful.
We consider a two-layer CB and compare the optimized rate with and without status bits. Sizings that achieve the minimum rate with $\alpha = 1.5$ and maximum flow size 13 are summarized below. Here r denotes the total number of bits per flow. β_i denotes the number of counters per flow in the i-th layer. d_1 denotes the number of bits in the first layer, (in the two-layer case, $d_2 =$ maximum flow size $-d_1$). k_i denotes the number of hash functions in the i-th layer. CB with status bits achieves smaller total space, r. Similar results are observed with other values of α and maximum flow size.

[6] More precisely, it refers to the probability that an outgoing message is in error.

	r	β_1	β_2	d_1	k_1	k_2
status bit	4.13	0.71	0.065	4	3	3
no status bit	4.66	0.71	0.14	5	3	3

We summarize the above as the following **rules of thumb.**

1. Use a two-layer CB with status bits and 3 hash functions at each layer.
2. Empirically estimate (or guess based on historical data) the heavy-tail exponent α and the max flow size.
3. Compute β^* according to Theorem 2. Set $m_1 = \beta^* n$ and $m_2 = 0.1\beta^* n$.
4. Use 5-bit counters at layer 1 for $\alpha \geq 1.1$, and 8-bit counters for $\alpha < 1.1$. Use deep enough counters at layer 2 so that the largest flow is accommodated (in general, 64-bit counters at layer-2 are deep enough).

7. EVALUATION

We evaluate the performance of Counter Braids using both randomly generated traces and real Internet traces.

In Section 7.1 we generate a random graph *and* a random set of flow sizes for each run of experiment. We use $n = 1000$ and are able to average the reconstruction error, $\mathrm{P_{err}}$, and the average error magnitude, E_m, over enough rounds so that their standard deviation is less than 1/10 of their magnitude.

In Section 7.2 we use 5-minute segments of two one-hour contiguous Internet traces and generate a random graph for each segment. We report results for the entire duration of two hours. The reconstruction error $\mathrm{P_{err}}$ is the total number of errors divided by the total number of flows, and the average error magnitude E_m measures how big the deviation from the actual flow size is provided an error has occurred.

7.1 Performance

First, we compare the performance of one-layer and two-layer CB. We use 1000 flows randomly generated from the distribution $\mathbb{P}(f_i > x) = x^{-1.5}$, whose entropy is a little less than 3 bits. We vary the total number of bits per flow in CB and compute $\mathrm{P_{err}}$ and E_m. In all experiments, we use CB with 3 hash functions. For the two-layer CB, we use 4-bit deep layer-1 counters with status bits. The results are shown in Figure 12.

The points labelled 1-layer and 2-layer threshold respectively are asymptotic threshold computed using density evolution. We observe that with 1000 flows, there is a sharp decrease in $\mathrm{P_{err}}$ around this asymptotic threshold. Indeed, the error is less than 1 in 1000 when the number of bits per flow is 1 bit above the asymptotic threshold. With a larger number of flows, the decrease around threshold is expected to be even sharper.

Similarly, once above the threshold, the average error magnitude for both 1-layer and 2-layer Counter Braids is close to 1, the minimum magnitude of an error. When below the threshold, the average error magnitude increases only linearly as the number of bits decreases. At 1 bit per flow, we have $40-50\%$ flows incorrectly decoded, but the average error magnitude is only about 5. This means that many flow estimates are not far from the true values.

Together, we see that the 2-layer CB has much better performance than the 1-layer CB with the same space. As we increase the number of layers, the asymptotic threshold will move closer to entropy. However, we observe that the 2-layer CB has already accomplished most of the gain.

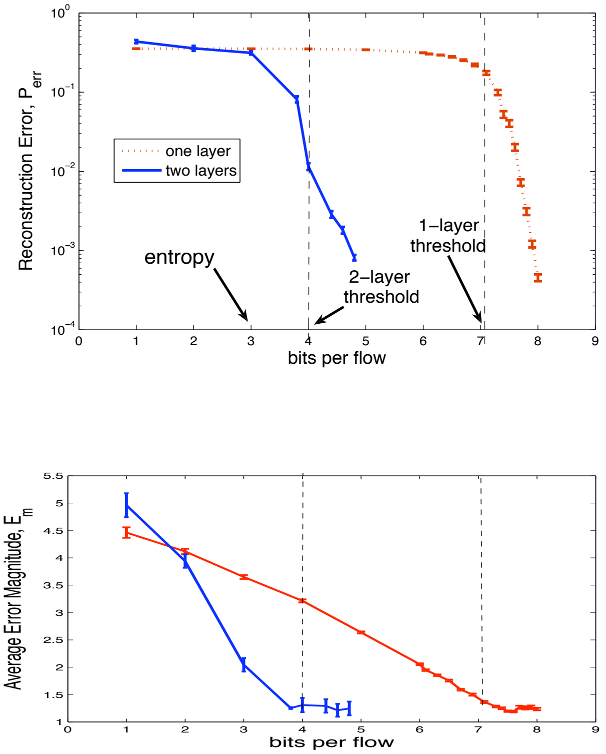

Figure 12: Performance over a varying number of bits per flow.

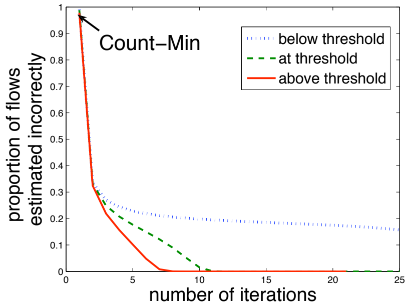

Figure 13: Performance over number of iterations. Note that $\mathrm{P_{err}}$ for a Count-Min sketch with the same space as CB is high.

Next, we investigate the number of iterations required to reconstruct the flows. Figure 13 shows the remaining proportion of incorrectly decoded flows as the number of iterations increases. The experiments are run for 1000 flows with the same distribution as above, on a one-layer Counter Braids. The number of bits per flow is chosen to be below, at and above the asymptotic threshold. As predicted by density evolution, $\mathrm{P_{err}}$ decreases exponentially and converges to 0 at or above the asymptotic threshold, and converges to a positive error when below threshold. In this experiment, 10 iterations are sufficient to recover most of the flows.

7.2 Trace Simulation

We use two OC-48 (2.5 Gbps) one-hour contiguous traces at a San Jose router. Trace 1 was collected on Wednesday, Jan 15, 2003, 10am to 11am, hence representative of weekday traffic. Trace 2 was collected on Thur Apr 24, 2003, 12am to 1am, hence representative of night-time traffic. We divide each trace into 12 5-minute segments, corresponding to a measurement epoch. Figure 14 plots the tail distribution ($\mathbb{P}(f_i > x)$) for all segments. Although the line rate is not high, the number of active flows is already significant. Each segment in trace 1 has approximately 0.9 million flows and 20 million packets, and each segment in trace 2 has approximately 0.7 million flows and 9 million packets. The statistics across different segments within one trace are similar.

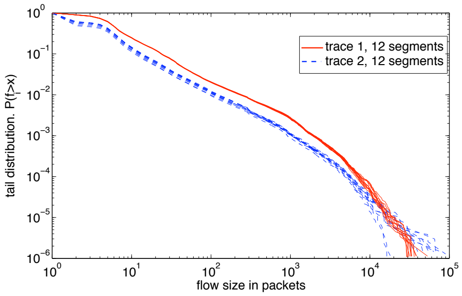

Figure 14: Tail distribution.

Trace 1 has heavier traffic than trace 2 and also a heavier tail. In fact, it is the heaviest trace we have encountered so far, and is much heavier than, for instance, traces plotted in [16]. The proportion of one-packet flows in trace 1 is only 0.11, similar to that of a power-law distribution with $\alpha = 0.17$. Flows with size larger than 10 packets are distributed similar to a power law with $\alpha \approx 1$.

We fix the same sizing of CB for all segments, mimicking the realistic scenario where traffic varies over time and CB is built in hardware. We present the proportion of flows in error $\mathrm{P}_{\mathrm{err}}$ and the average error magnitude E_m for both traces together. We vary the total number of bits in CB[7], denoted by B, and present the result in Table 3.

For all experiments, we use a two-layer CB with status bits, and 3 hash functions at both layers. The layer-1 counters are 8 bits deep and the layer-2 counters are 56 bits deep.

B (MB)	1.2	1.3	1.35	1.4
$\mathrm{P}_{\mathrm{err}}$	0.33	0.25	0.15	0
E_m	3	1.9	1.2	0

Table 3: Simulation results of counting 2 traces in 5-minute segments, on a fixed-size CB with total space B.

We observe a similar phenomenon as in Figure 12. As we underprovide space, the reconstruction error increases significantly. However, the error magnitude remains small. For these two traces, 1.4 MB is sufficient to count all flows correctly in 5-minute segments.

8. IMPLEMENTATION

8.1 On-Chip Updates

Each layer of CB can be built on a separate block of SRAM to enable pipelining. On pre-built memories, the counter depth is chosen to be an integer fraction of the word length, so as to maximize space usage. This constraint does not exist with custom-made memories.

We need a list of flow labels to construct the first-layer graph for reconstruction. In cases where access frequencies for pre-fixes or filters are being collected, the flow nodes are simply the set of pre-fixes or filter criteria, which are the same across all measurement epochs. Hence no flow labels need to be collected or transferred.

In other cases where the flow labels are elements of a large space (e.g. flow 5-tuples), the labels need to be collected and transferred to the decoding unit. The method for collecting flow labels is application-specific, and may depend on the particular implementation of the application. We give the following suggestion for collecting flow 5-tuples in a specific scenario.

For TCP flows, a flow label can be written to a DRAM which maintains flow IDs when a flow is established; for example, when a "SYN" packet arrives. Since flows are established much less frequently than packet arrivals (approximately one in 40 packets causes a flow to be set up [10]), these memory accesses do not create a bottleneck. Flows that span boundaries of measurement epochs can be identified using a Bloom Filter[3].

Finally, we evaluated the algorithm by measuring flow sizes in packets. The algorithm can be used to measure flow sizes in bytes. Since most byte-counting is really the counting of byte-chunks (e.g. 32 or 64 byte-chunks), there is the question of choosing the "right granularity": a small value gives accurate counts but uses more space and vice versa. We are working on a nice approach to this problem and will report results in future publications.

8.2 Computation Cost of Decoder

We reconstruct the flow sizes using the iterative message passing algorithm in an offline unit. The decoding complexity is linear in the number of flows. Decoding CB with more than one layer imposes only a small additional cost, since the higher layers are $1-2$ orders smaller than the first layer. For example, decoding 1 million flows on a two-layer Counter Braids takes, on average, 15 seconds on a 2.6GHz machine.

9. CONCLUSION AND FURTHER WORK

We presented Counter Braids, a efficient minimum-space counter architecture, that solves large-scale network measurement problems such as per-flow and per-prefix counting. Counter Braids incrementally compresses the flow sizes as it counts and the message passing reconstruction algorithm recovers flow sizes almost perfectly. We minimize counter space with incremental compression, and solve the flow-to-counter association problem using random graphs. As shown from real trace simulations, we are able to count upto 1 million flows purely in SRAM and recover the exact flow sizes. We are currently implementing this in an FPGA to determine the actual memory usage and to better understand implementation issues.

[7] We are not using bits per flow here since the number of flows is different in different segments.

Several directions are open for further exploration. We mention two: (i) Since a flow passes through multiple routers, and since our algorithm is amenable to a distributed implementation, it will save counter space dramatically to combine the counts collected at different routers. (ii) Since our algorithm "degrades gracefully," in the sense that if the amount of space is less than the required amount, we can still recover many flows accurately and have errors of known size on a few, it is worth studying the graceful degradation formally as a "lossy compression" problem.

Acknowledgement: Support for OC-48 data collection is provided by DARPA, NSF, DHS, Cisco and CAIDA members. This work has been supported in part by NSF Grant Number 0653876, for which we are thankful. We also thank the Clean Slate Program at Stanford University, and the Stanford Graduate Fellowship program for supporting part of this work.

10. REFERENCES

[1] *http://www.cisco.com/warp/public/732/Tech/netflow.*

[2] Juniper networks solutions for network accounting. *www.juniper.net/techcenter/appnote/350003.html.*

[3] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Comm. ACM*, 13, July 1970.

[4] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, pages 56–67, 1998.

[5] G. Caire, S. Shamai, and S. Verdú. Noiseless data compression with low density parity check codes. In *DIMACS*, New York, 2004.

[6] E. Candès and T. Tao. Near optimal signal recovery from random projections and universal encoding strategies. *IEEE Trans. Inform. Theory*, 2004.

[7] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1), April 2005.

[8] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, New York, 1991.

[9] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Trans. Networking*, 1997.

[10] S. Dharmapurikar and V. Paxson. Robust tcp stream reassembly in the presence of adversaries. *14th USENIX Security Symposium*, 2005.

[11] D. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52(4), April 2006.

[12] C. Estan and G. Varghese. New directions in traffic measurement and accounting. *Proc. ACM SIGCOMM Internet Measurement Workshop*, pages 75–80, 2001.

[13] R. G. Gallager. *Low-Density Parity-Check Codes*. MIT Press, Cambridge, Massachussetts.

[14] M. Grossglauser and J. Rexford. Passive traffic measurement for ip operations. *The Internet as a Large-Scale Complex System*, 2002.

[15] F. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47:498–519, 2001.

[16] A. Kumar, M. Sung, J. J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. *Proceedings of ACM SIGMETRICS*, 2004.

[17] Y. Lu, A. Montanari, and B. Prabhakar. Detailed network measurements using sparse graph counters: The theory. Allerton Conference, September 2007.

[18] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. A. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proc. of STOC*, pages 150–159, 1997.

[19] M. Mézard and A. Montanari. *Constraint satisfaction networks in Physics and Computation.* In Preparation.

[20] S. Ramabhadran and G. Varghese. Efficient implementation of a statistics counter architecture. *Proc. ACM SIGMETRICS*, pages 261–271, 2003.

[21] T. Richardson and R. Urbanke. *Modern Coding Theory.* Cambridge University Press, 2007.

[22] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown. Analysis of a statistics counter architecture. *Proc. IEEE HotI 9.*

[23] Q. G. Zhao, J. J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. SIGMetrics/Performance, June 2006.

Appendix: Asymptotic Optimality

We state the result on asymptotic optimality without a proof. The complete proof can be found in [17].

We make two assumptions on the flow size distribution p:

1. It has *at most power-law tails.* By this we mean that $\mathbb{P}\{f_i \geq x\} \leq Ax^{-\epsilon}$ for some constant A and some $\epsilon > 0$. This is a valid assumption for network statistics [9].

2. It has *decreasing digit entropy.* Write f_i in its q-ary expansion $\sum_{a\geq 0} f_i(a)q^a$. Let $h_l = \sum_x \mathbb{P}(f_i(l) = x) \log_q \mathbb{P}(f_i(l) = x)$ be the q-ary entropy of $f_i(l)$. Then h_l is monotonically decreasing in l for any q large enough.

We call a distribution p with these two properties *admissible.* This class includes most cases of practical interest. For instance, any power-law distribution is admissible. The (binary) entropy of this distribution is denoted by $H_2(p) \equiv -\sum_x p(x) \log_2 p(x)$.

For this section only, we assume that all counters in CB have an equal depth of d bits. Let $q = 2^d$.

Definition 2. *We represent CB as a sparse graph G, with vertices consisting of n flows and a total of $m(n)$ counters in all layers. A sequence of Counters Braids $\{G_n\}$has* design rate *r if*

$$r = \lim_{n\to\infty} \frac{m(n)}{n} \log_2 q \,. \tag{1}$$

It is reliable *for the distribution p if there exists a sequence of reconstruction functions $\widehat{\mathsf{F}}_n \equiv \widehat{\mathsf{F}}_{G_n}$ such that*

$$\mathrm{P}_{\rm err}(G_n, \widehat{\mathsf{F}}_n) \equiv \mathbb{P}\{\widehat{\mathsf{F}}_n(\mathbf{c}) \neq \mathbf{f}\} \overset{n}{\to} 0 \,. \tag{2}$$

Here is the main theorem:

Theorem 3. *For any admissible input distribution p, and any rate $r > H_2(p)$ there exists a sequence of reliable sparse Counter Braids with asymptotic rate r.*

The theorem is satisfying as it shows that the CB architecture is fundamentally good in the information-theoretic sense. Despite being incremental and linear, it is as good as, for example, Huffman codes, at infinite blocklength.