BRICK: A Novel Exact Active Statistics Counter Architecture

Nan Hua\textsuperscript{1}, Bill Lin\textsuperscript{2}, Jun (Jim) Xu\textsuperscript{1}, Haiquan (Chuck) Zhao\textsuperscript{1}

\textsuperscript{1}Georgia Institute of Technology
\textsuperscript{2}University of California, San Diego

*Supported in part by CNS-0519745, CNS-0626979, CNS-0716423, CAREER Award ANI-0238315, and a gift from CISCO
Outline

• Motivation and current approaches

• Our approach

• Performance evaluation

• Conclusion
Motivation

• Routers need to maintain very large arrays of per-flow statistics counters at wirespeed
  
  – Needed for various network measurement, router management, traffic engineering, and data streaming applications

  – Millions of counters are needed for per-flow measurements

  – Large counters are needed (e.g. 64 bits) for worst-case counts during a measurement epoch

  – At 40 Gb/s, just 8 ns update time
Passive vs. Active Counters

• Passive counters:
  – For collection of traffic statistics that are analyzed “offline”, counters just need to be updated in wirespeed, **but** full counter values generally do not need to be read frequently (say not until the end of a measurement epoch)

• Active counters:
  – However, a number of applications require the maintenance of **active counters**, in which values may need to be read as frequently as they are incremented, typically on a per packet basis
  – e.g. in many data streaming applications, on each packet arrival, values need to be read from some counters to decide on actions that need to be taken
Naïve Approach

• Store full counters in SRAM, which supports both passive and active counter applications

• Problem: Prohibitively Expensive
  – e.g. 1 million flows x 64-bits = 64 Mbits = 8 MB of SRAM
  – Number of flows increasing with line rates
Hybrid SRAM-DRAM Architectures
(Shah’02, Ramabhadran’03, Roeder’04, Zhao’06)

• Basic idea
  – Store full counters in DRAM (64-bits)
  – Keep say a 5-bit SRAM counter, one per flow
  – Wirespeed increments on 5-bit SRAM counters
  – “Flush” SRAM counters to DRAM before they “overflow”
  – Once “flushed”, SRAM counter won’t overflow again for at least say another $2^5 = 32$ (or $2^b$ in general) cycles

• Problem: Passive Only
  – Can only read counter values at DRAM speed (e.g. 50 ns << wirespeed)
Interleaved DRAM Architectures
(Lin and Xu, HotMetrics’08)

• Basic idea
  – Exploit the fact that modern DRAMs have many internal memory banks (e.g. Rambus XDR has 16 internal banks per memory chip)
  – New memory transaction can be initiated say every 4ns if to a different (internal) memory bank, even though memory latency is much higher
  – Therefore, wirespeed counter updates can be achieved

• Problem: Still Passive Only
  – Worst-case counter read time too high
Counter Braids
(Lu et al, Sigmetrics’08)

- Inspired by the construction of LDPC codes
  - Counter updates performed on an encoded structure called a “counter braid”
  - Counter values can be viewed as a linear transformation of flow counts
  - However, counter braids are “more passive” than SRAM-DRAM or DRAM architectures – to find out the size of a single flow, one needs to decode all flow counts in a lengthy decoding process

- Problem: Also Passive Only
Approximate Counters

- Generally based on the approximate counting idea by Morris (1978)
  - Idea is to “probabilistically” increment a counter based on the current counter value
  - Small number of bits can be used (e.g. 5 bits per counter), and hence can be stored in SRAM for active retrieval
  - However, approximate counting in general has a very large error margin when the number of bits used is small (e.g. well over 100% error) – not acceptable in many applications

- **Problem: Large Errors Possible**
Summary

• Naïve “brute-force” SRAM approach
  – Too expensive

• SRAM-DRAM, DRAM, and counter braid approaches
  – Passive counting applications only

• Approximate methods
  – Not sufficiently accurate
Our Approach

Main observations

- The total number of increments during a measurement epoch is bounded by $M$ cycles (e.g. $M = 16$ million cycles)
- Therefore, the sum of all $N$ counters is also bounded by $M$ (e.g. $N = 1$ million counters)

\[
\sum_{i=1}^{N} C_i \leq M
\]

- Although worst-case count can be $M$, the average count is much smaller (e.g. $M/N = 16$, then average counter size should be just $\log 16 = 4$ bits)
Our Approach (cont’d)

• To exploit the fact that most counters will be small, we propose a novel “Variable-Length Counter” representation called BRICK, which stands for Bucketized Rank-Indexed Counters

• Only dynamically increase counter size as necessary

• The result is an exact counter data structure that is small enough for SRAM storage, enabling both active and passive applications
Basic Idea

• Randomly bundle counters into buckets

Statistically, the sum of counter sizes per bucket should be similar
BRICK Wall Analogy

- Each row corresponds to a bucket

- Buckets should be *statically* sized to ensure a very low probability of overflow

- Then provide a small amount of extra storage to handle overflow cases
A Key Challenge and Our Approach

• The idea of variable-length data structures is not new, but *expensive pointers* are typically used to “chain” together different segments of a data structure.

• In the case of counters, these pointers are as or even more expensive than the counters themselves!

• Our key idea is a novel indexing method called **Rank Indexing**.
Rank Indexing

- How rank indexing works?
  - The location of the linked element is calculated by the "rank" operation, $\text{rank}(A, b)$, which returns the number of bits set in bitmap $A$ at or before position $b$
  - No need for explicit pointer storage!

<table>
<thead>
<tr>
<th>$A_3$</th>
<th>$A_2$</th>
<th>$A_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>00000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>11011</td>
</tr>
</tbody>
</table>

$C_1 = 64$
$C_2 = 31$
$C_3 = 30$
$C_4 = 2$
$C_5 = 379$
$C_6 = 4$
$C_7 = 3$
$C_8 = 2$

$\text{rank}(I_1, 5) = 2$ (it’s 2nd bit set in $I_1$)

$\text{rank}(I_2, 2)$

$A_3$ (data)
$I_2$ (index)
$A_2$ (data)
$I_1$ (index)
$A_1$ (data)

$C_1 = 64$
$C_2 = 31$
$C_3 = 30$
$C_4 = 2$
$C_5 = 379$
$C_6 = 4$
$C_7 = 3$
$C_8 = 2$
Rank Indexing

• **Key observation:** The *rank* operator can be efficiently implemented in modern 64-bit x86 processors

• Specifically, both Intel and AMD x86 processors provide a *popcount* instruction that returns the number of 1’s in a 64-bit word

• The *rank* operator can be implemented in just 2 instructions using a bitwise-AND instruction and the *popcount* instruction!
Dynamic Sizing

- Suppose we increment $C_2$, which requires dynamic expansion into $A_2$
- The update is performed by performing a variable shift operation in $A_2$, which is also efficiently implemented with x86 hardware instructions

\[ \text{rank}(I_1, 5) = 2 \]  
(it was 2\text{nd} bit set in $I_1$)

\[ \text{rank}(I_1, 5) = 3 \]  
(it’s now 3\text{rd} bit set in $I_1$)
Finding a Good Configuration

• We need to decide on the following for a good configuration
  – $k$: the number of counters in each bucket
  – $p$: the number of sub-arrays in each bucket $A_1 \ldots A_p$
  – $k_1 \ldots k_p$: the number of entries in each sub-array ($k_1 = k$)
  – $w_1 \ldots w_p$: the bit-width sub-array

• Given these configurations, we can decide on the probability of bucket overflow $P_f$ using a binomial distribution tail bound analysis
Tail Bound (I)

- Due to the total count constraint $\sum_{i=1}^{N} C_i \leq M$

  at most $\frac{M}{2^{w_1 + w_2 + \ldots + w_{d-1}}}$ (defined as $m_d$)

  counters would be expanded into the $d^{th}$ Array

- Translated into the language of balls and bins:
  - Throwing $m_d$ balls into $N$ bins
  - The capacity of each bin is only $k_d$.
  - Bound the probability that more than $J_d$ bins have more than $k_d$ balls
Tail Bound (II)

• Random Variable $X_i^{(m)}$ denotes the number of balls threw into $i^{th}$ bin, when there comes $m$ balls in total.

• The fail probability is

$$\Pr\left[ \sum_{j=1}^h \mathbb{1}_{\{X_j^{(m)} > c\}} > J \right]$$

($J$ is the number of full-size buckets pre-allocated)
(now we forgot “d”, since the calculation is the same for each level. For convenience, we use $c$ to denote $k_d$)

• We could “estimate” fail probability by this way:
  – The overflow probability from one bin is roughly

$$\epsilon = \text{Binotail}_{k,m/N}(c)$$

($\text{Binotail}$ is tail probability of Binomial distribution)

  – Then the total fail probability would be roughly

$$\delta = \text{Binotail}_{h,\epsilon}(J)$$

  – This calculation is not strict! since Random Variable $X_i^{(m)}s$ are correlated under the constraint (although weakly)
Tail Bound (III)

• How to “de-correlate” the weakly correlated $X_i^{(m)}$?
• Construct Random Variables $Y_i^{(m)}$, $i=1,...,h$, which is i.i.d random variables with Binomial distribution $(k,m/N)$.
• It could be proved that:

$$E[f(X_1^{(m)},...,X_h^{(m)})] \leq 2E[f(Y_1^{(m)},...,Y_h^{(m)})]$$

where $f$ is an nonnegative and increasing function.
• Then, we could use the following increasing indicator function to get the bound

$$f(x_1,...,x_h) = 1 \left\{ \sum_{i=1}^{h} 1_{\{x_i > c\}} > J \right\}$$
Numerical Results

• Sub-counter array sizing and per-counter storage for \( k = 64 \) and \( P_f = 10^{-10} \)

(a) Sizing of sub-counter arrays.

<table>
<thead>
<tr>
<th>( p )</th>
<th>( k_2 )</th>
<th>( k_3 )</th>
<th>( k_4 )</th>
<th>( k_5 )</th>
<th>( w_1 )</th>
<th>( w_2 )</th>
<th>( w_3 )</th>
<th>( w_4 )</th>
<th>( w_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>15</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>( \lg \frac{M}{N} + 3 )</td>
<td>4</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>( \lg \frac{M}{N} + 2 )</td>
<td>2</td>
<td>4</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>10</td>
<td>3</td>
<td>1</td>
<td>( \lg \frac{M}{N} + 2 )</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

(b) Size of each sub-counter array = \( k_j \times w_j \) (in bits).

<table>
<thead>
<tr>
<th>( p )</th>
<th>( A_2 )</th>
<th>( A_3 )</th>
<th>( A_4 )</th>
<th>( A_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>( 15 \times 4 = 60 )</td>
<td>( 3 \times 13 = 39 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( 25 \times 2 = 50 )</td>
<td>( 10 \times 4 = 40 )</td>
<td>( 2 \times 12 = 24 )</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>( 25 \times 2 = 50 )</td>
<td>( 10 \times 3 = 30 )</td>
<td>( 3 \times 4 = 12 )</td>
<td>( 1 \times 9 = 9 )</td>
</tr>
</tbody>
</table>

(c) Storage per counter.

<table>
<thead>
<tr>
<th>( p = 3 )</th>
<th>( p = 4 )</th>
<th>( p = 5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lg \frac{M}{N} + 6.05 )</td>
<td>( \lg \frac{M}{N} + 5.66 )</td>
<td>( \lg \frac{M}{N} + 5.50 )</td>
</tr>
</tbody>
</table>
Effects of Larger Buckets

- Bucket size $k = 64$ works well, amenable to 64-bit processor instructions
Simulation of Real Traces

- USC (18.9 million packets, 1.1 million flows) and UNC traces (32.6 million packets, 1.24 million flows)

### Percentage of full-size buckets

<table>
<thead>
<tr>
<th>Trace</th>
<th>$h$</th>
<th>$J$</th>
<th>$\frac{J}{h}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>USC</td>
<td>17.3K</td>
<td>111</td>
<td>0.60%</td>
</tr>
<tr>
<td>UNC</td>
<td>19.5K</td>
<td>104</td>
<td>0.57%</td>
</tr>
</tbody>
</table>
Concluding Remarks

• Proposed an efficient variable-length counter data structure called BRICK that can implement exact statistics counters

• Avoids explicit pointer storage by means of a novel rank indexing method

• Bucketization enables statistical multiplexing
Thank You