Constructions and Applications for Accurate Counting of the Bloom Filter False Positive Free Zone

<table>
<thead>
<tr>
<th>Ori Rottenstreich</th>
<th>Pedro Reviriego</th>
<th>Ely Porat</th>
<th>S. Muthukrishnan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technion</td>
<td>Uni. Carlos III de Madrid</td>
<td>Bar Ilan</td>
<td>Rutgers Uni.</td>
</tr>
</tbody>
</table>

ACM Symposium on SDN Research (SOSR), March 3, 2020
Set representation: Support queries of the form: Is $Flow\ y \in set\ S$?

Flow size estimation:
How many observed packets of $Flow\ y$?

- Requirements for data structure:
  - Space efficient
  - Fast (Update, Query)

Can tasks be supported accurately?
Set Representation - Naïve Solutions

Is $y \in S$?

- $O(|S|)$ – Searching in a list
- $O(\log(|S|))$ – Searching in a sorted list
- $O(1)$?
  - Tradeoff: Errors occur with low probability

- Two possible errors
  - False Positives - $y \notin S$ but the answer is $y \in S$.
  - False Negatives - $y \in S$ but the answer is $y \notin S$. 

Set $S$
(Special Flows)
Bloom Filters (Bloom, 1970)

- Initialization: Array of $m$ zero bits

  \[
  \begin{array}{cccccccccccccc}
  0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  \end{array}
  \]

- Insertion: Each of the $|S|$ elements is hashed $k$ times, the corresponding bits are set

  \[
  S=\{x,y\}
  \]

- Query: Hashing the element, checking that all $k$ bits are set

- No false negatives

- False positive rate (probability) $\text{FPR} \approx (0.6185)^{m/|S|}$
  - Controlled by the memory allocation but always positive
  - Can we completely avoid false positives?

---

- Bloom Filters (Bloom, 1970)
The Bloom filter principal:
Wherever a set is used and space is a concern, consider using a Bloom filter if the effect of false positives can be mitigated

- Cache/Memory Framework
- Packet Classification
- Intrusion Detection
- Routing
- Accounting
- Beyond networking: Spell Checking, DNA Classification

Can be found in
- Google's web browser Chrome
- Google's database system BigTable
- Facebook's distributed storage system Cassandra
- Mellanox's IB Switch System
- Blockchain systems: Bitcoin and Ethereum
Application example:
In Packet Bloom filters

Multicast addressing

- No states in the routers
- Finite universe of possible links, short paths
- Path = Set of links
- Forwarding decision based on a membership query

Rome $ightarrow$ Milan
Milan $ightarrow$ Zurich
Milan $ightarrow$ Munich

Link ID:
- Rome→Milan: 0 1 0 0 0 1 0
- Milan→Zurich: 1 0 0 0 0 1 0
- Milan→Munich: 0 1 0 0 0 0 1

Packet header

Bloom filter:

False positive: a packet is forwarded on an extra link

Can cause infinite loops!

Application example: Blockchain Technology

Light Clients in Bitcoin and Ethereum

- Interested in a small subset of accounts (addresses)
- A full client holds a Bloom filter of the addresses,
  Only relevant traffic is forwarded to the light client
- False positive: Redundant forwarded traffic

- Finite universe: The set of all active addresses
- Typically small sets of accounts in a light client

Avoiding False positives

- Only possible when the universe of elements is finite
- We define conditions, under which the filter is guaranteed to avoid false positives
  - Requirements:
    - The size of $S$ is at most $d$
    - The elements inserted are from $U = \{1, \ldots, n\}$
  - Boundaries of the False Positive Free Zone

False positive free zone:
For a given memory size $m$, smaller universe size $n$ allows more elements in a set $d$
Intuition for the False Positive Free Zone

- Input:
  - Universe \( U = \{1, \ldots, n\} \)
  - No false positives for \(|S| \leq d\)

- Carefully design the hash function (selected bits for each element) so that:
  - Given any set of size at most \( d \):
    - Every element not in the set maps to at least one bit of 0
  - False positives cannot occur

- The existing construction has memory complexity of \( O(d^2 \log n) \)
- Cannot scale well for allowing large maximal set size \( d \)
Outline

• Introduction to Bloom filter
• The false positive free zone
• Existing Scheme – EGH filter
• New Scalable Schemes – OLS filter and POL filter
• CM Sketch – Application for accurate flow size estimation
• Summary
Existing Scheme: The EGH Filter

- Combinatorial group testing technique
  - Based on Chinese Remainder Theorem
- Input:
  - Universe \( U = \{1, \ldots, n\} \)
  - At most \( d \) elements in the filter
- Select the \( k \) first primes 2, 3, 5, \ldots, \( p_k \) so that \( 2 \times 3 \times 5 \times \ldots \times p_k > n^d \)
- The EGH filter is \( 2 + 3 + 5 + \ldots + p_k \) bits long, composed of \( k \) blocks
- No false positives for \( |S| \leq d \)
- Memory Complexity of \( O(d^2 \log n) \)

\[
\begin{array}{cccccc}
2 & 3 & 5 & 7 \\
x=1 & 0 & 1 & 0 & 1 & 0 & 0 \\
x=9 & 0 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

EGH Filter Example

- $U = \{1, \ldots, n=48\}$, $d = 2$
- A 2-disjunct matrix with $n=48$ columns, $m=28$ lines
- $m = 28 = 2 + 3 + 5 + 7 + 11$ bits
- Simple five hash functions:
  - $h_1(x) = x \mod 2$,
  - $h_2(x) = x \mod 3 + 2$,
  - $h_3(x) = x \mod 5 + 5$,
  - $h_4(x) = x \mod 7 + 10$,
  - $h_5(x) = x \mod 11 + 17$
Scalability for Large Sets

- Memory Complexity of existing scheme $O(d^2 \log n)$
- Grows quadratically with maximal set size $d$
- Cannot scale well for representing large sets

- Larger sets can be useful, eg, for
  - Larger caches
  - Transaction pools for higher transaction rates
  - Encoding paths in networks of larger diameter

- Can the memory complexity scale better to allow larger sets?
- Potentially larger dependency in the universe size $n$
Based on Orthogonal Latin Square (OLS) Codes

- Previously used to detect and correct errors in memories
- Parity check matrix on which two elements share at most a parity bit

Latin square properties:

- $s \times s$ array
- Each symbol appears exactly once in each row and column
- In our case, symbols are 0, 1, 2, ..., s-1
- A pair of squares is called orthogonal if when superimposed imply all $s^2$ pairs

Examples for OLS

First scheme: OLS Filter

Additional matrices
First scheme: OLS Filter

- Based on Orthogonal Latin Square (OLS) Codes
  - Previously used to detect and correct errors in memories
  - Parity check matrix on which two elements share at most a parity bit
- Input:
  - Universe $U = \{1, \ldots, n\}$
  - At most $d$ elements in the filter
- Latin squares of size $\sqrt{n} \times \sqrt{n}$
- The filter is divided in $d+1$ groups of size $\sqrt{n}$
- Each group is based on a matrix: Two simple and additional orthogonal latin squares
- Modular construction on $d$, more parity groups can be added to increase $d$
- No false positives for $|S| \leq d$
- Memory Complexity of $(d+1) \cdot \sqrt{n}$
- Scales linearly with maximal set size $d$
OLS Example:
Universe size \( n = 25 \) (\( \sqrt{n} = 5 \)),
Maximal Set size \( d = 3 \)

- Universe size \( n \), for each element (column) a single bit of 1 in each group
- No false positives for \( |S| \leq d \)
- Filter length = \((d+1)\cdot\sqrt{n}\)
For every element a single bit of 1 in each group, a total of \( d+1 \) bits of 1

Two columns cannot share more than a single one

Given a set of size \(|S| \leq d\), among the \( d+1 \) bits of an element not in the set at least one if not covered by the set elements
Second scheme: POL Filter

Based on Polynomials of degree $t-1$
  - Assumption: $t \sqrt{n} = n^{1/t}$ is a prime number
  - Coefficients belong to $[0, t \sqrt{n} - 1]$

Input:
  - Universe $U = \{1, \ldots, n\}$
  - At most $d$ elements in the filter

Each element $y$ is defined by the polynomial for which

$$P_y(t \sqrt{n}) = \sum_{i=0}^{t-1} a_i \cdot (t \sqrt{n})^i = y$$

Each element $y$ is represented by the values of the polynomial modulo $t \sqrt{n}$ for

$$x \in [0, (t-1) \cdot d]$$

No false positives for $|S| \leq d$
Memory Complexity of $((t-1) \cdot d+1) \cdot t \sqrt{n}$
• Universe size $n = 7^3 = 343$, $t\sqrt{n} = 7$ for parameter $t=3$
• OLS filter length $19(d+1)$
• POL filter length $((t-1) \cdot d+1) \cdot t\sqrt{n} = (2d+1) \cdot 7 = 14d+7$

• For $d = 2$:
  ▪ Number of groups $((t-1) \cdot d+1) = ((3-1) \cdot 2+1) = 5$
  ▪ Each of $t\sqrt{n} = 7$ bits
  ▪ Filter of length $5 \cdot 7 = 35$ bits, five groups of 7 bits

• For each value $y$ among the $n=343$:
  ▪ Compute the polynomial $P_y(x)$ such that $y = P_y(t\sqrt{n} = 7) = a_0+a_1 \cdot 7+a_2 \cdot 7^2+a_3 \cdot 7^3+…$
  ▪ Compute vector of five groups based on values $P_y(x)$ for $x=0,1,2,3,4$

• Examples:
  ▪ For $y = 7 = t\sqrt{n}$, Polynomial $P_y(x) = x$
    $$(1000000 0100000 0010000 0001000 0000100)$$
  ▪ For $y = 50 = 7^2+1=(t\sqrt{n})^2+1$, Polynomial $P_y(x) = x^2+1$
    $$(0100000 0010000 0000010 0001000 0001000)$$
Memory Footprint

- Allows better scalability for larger sets (d)
- Results in more expensive dependency in universe size (n)

<table>
<thead>
<tr>
<th>filter</th>
<th>memory complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGH filter</td>
<td>$O(d^2 \cdot \log n)$</td>
</tr>
<tr>
<td>OLS filter</td>
<td>$(d + 1)\sqrt{n}$</td>
</tr>
<tr>
<td>POL filter</td>
<td>$((t - 1) \cdot d + 1) \cdot \sqrt{n}$</td>
</tr>
</tbody>
</table>

![Graphs showing memory footprint for different filters and maximal set sizes](image)
Counting Bloom Filters (CBFs)

- Bloom filters do not support deletions of elements. Simply resetting bits might cause false negatives.

- Counting Bloom filters - Storing array of counters instead of bits.
  - Insertion: Incrementing \( k \) counters by one
  - Deletion: Decrementing \( k \) counters by one
  - Query: Checking that \( k \) counters are positive

- The same false positive probability
- Require more memory – usually x4
- The false positive free zone applies also to Counting Bloom filters
Accurate Flow Size Estimation

Count Min Sketch [Cormode and Muthukrishnan, 2005]

• Flow size: Number of packets in a flow
• Increment counter in each appearance

Traditional flow size estimation:
  • Minimal value among counters mapped by the flow
  • Can suffer from overestimation

Accurate Count Min Sketch Idea:
  • Design your hash functions carefully based on the false positive free zone
  • If the number of measured flows is small, it is the only flow in at least one counter
  • Flow size estimation is accurate based on that counter, no overestimation
Accurate Flow Size Estimation
Count Min Sketch

- If we use a Bloom filter with a FPFZ of $d$ for the mappings of flows to counters:
  - Universe size $n$ refers to the number of potential flows
  - No error for active flows when the CMS has $d+1$ or less active flows
  - No error for non active flows when the CMS has $d$ or less active flows

- Total of $n=25$ potential flows with at most 10 flows of a non-zero size. Size of non-zero flows is uniformly distributed in $[1,100]$
Constructions of Accurate Bloom Filters and Accurate Count-Min Sketch

• Bloom filter constructions that avoid false positives
• Apply for a finite universe and scale for large sets
• Applications like: Routing, blockchain, distributed storage

• Accurate flow size estimation
  ▪ Avoiding overestimations in the well known Count-Min Sketch

Questions? Comments?

Ori Rottenstreich (Technion)
sites.google.com/site/orirottenstreich
Email: or@technion.ac.il

Thank you!