Bloom Filter Redux

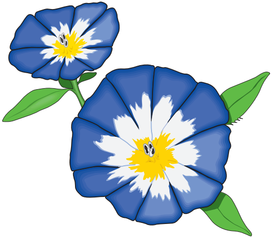
Matthias Vallentin   Gene Pang

CS 270
Combinatorial Algorithms
and Data Structures

UC Berkeley, Spring 2011

Inspiration

- Our **background**: network security, databases
  $\rightarrow$ We deal with massive data sets
- Lectures about **streaming algorithms** sparked our interest
  - Approximate set membership
  - Frequency estimation
- This **project**: explore and compare Bloom Filter variants



Bloom filters – What the Fl*wer?

Usage

When dealing with a **set** or **multiset** and space is an issue an, a **Bloom filter** (BF) may be tractable alternative.

- Synopsis data structure: substantially smaller than base data
- Price: only approximate answers
  - False Positives (FPs)
  - False Negatives (FNs)
- Applications
  - Dictionaries
  - Database joins
  - Networking (web caches, IP traceback, multicast, P2P overlays)
  - Blacklists (Google SafeBrowsing)

Outline

Bloom Filter

- Basic
- Counting
- Spectral
- Bitwise
- Stable
- A^2

Implementation

Evaluation

Outline

Bloom Filter
- Basic
- Counting
- Spectral
- Bitwise
- Stable
- A^2

Implementation

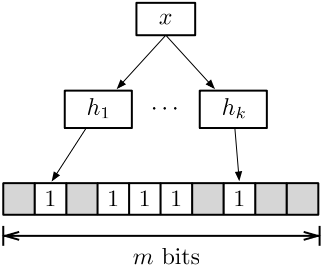
Evaluation

Terminology

- Universe U
- N distinct items
- k independent hash functions $h_1, \ldots, h_k$
- Vector V of m cells, i.e., $m = |V|$
- Set
  - $S = \{x_1, \ldots, x_n\}$ where $x_i \in U$ and $|S| = n$
- Multiset / Stream
  - $\mathcal{S} = \{x_1, \ldots, x_n\}$ where $x_i \in U$ and $|\mathcal{S}| = n$
  - $C_x = \left\{c_{h_1(x)}, \ldots, c_{h_k(x)}\right\}$ counters of x
  - f_x = multiplicity (frequency) of $x \in \mathcal{S}$
- Bloom filter estimate denoted by "hat"
  - $\widehat{S}$, $\widehat{\mathcal{S}}$, $\widehat{f}_x$, $\ldots$
- FP probability $\phi_P = \mathbb{P}\left[x \in \widehat{S} \,|\, x \notin S\right]$
- FN probability $\phi_N = \mathbb{P}\left[x \notin \widehat{S} \,|\, x \in S\right]$

Basic Bloom Filter

- By Burton Bloom in 1970 [Blo70]
- V has m single-bit cells
- k independent hash functions
- FPs but no FNs



add(x)

$$V[h_i(x)] = 1 \text{ for } i \in [k]$$

query(x)

$$\text{return } V[h_1(x)] == 1 \wedge \cdots \wedge V[h_k(x)] == 1$$

Bloom Error E_B

- **Bloom error** E_B: falsely report $x \in \widehat{S}$ although $x \notin S$
- Start with empty V, set k bits to 1. For a fixed cell i,

$$\mathbb{P}\left[V[i] = 0\right] = \left(1 - \frac{1}{m}\right)^k$$

- After n insertions,

$$\mathbb{P}\left[[V[i] = 1\right] = 1 - \left(1 - \frac{1}{m}\right)^{kn}$$

- Testing for membership involves hashing an item k times

$$\mathbb{P}\left[E_B\right] = \phi_P = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$$

Parameterization

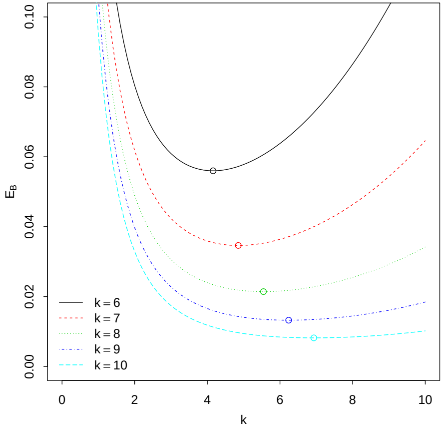
- Fix m and n. Then,

$$k^* = \operatorname*{argmin}_k \mathbb{P}\left[E_B\right] = \left\lfloor \frac{m}{n} \ln 2 \right\rfloor$$

- For k^*, $\mathbb{P}\left[E_B\right] = (0.619)^{m/n}$
- For a fixed $\phi_P = \mathbb{P}\left[E_B\right]$,

$$m = \left\lfloor -\frac{n \ln \phi_P}{(\ln 2)^2} \right\rfloor$$

$$\kappa = \left\lfloor -\frac{m}{\ln \phi_P} (\ln 2)^2 \right\rfloor$$



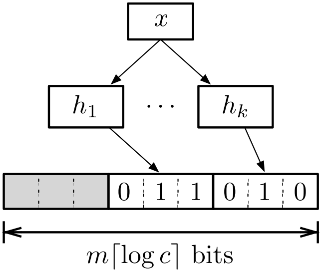
Definition

The **capacity** κ of a Bloom filter is the maximum number of items it can hold until a given ϕ_P can no longer be guaranteed. A Bloom filter is **full** when then number of added items exceeds κ.

Counting Bloom Filters [FCAB98]

Supporting Multisets

- V has m cells of width w
- Counters $c \in \{0, \ldots, 2^w - 1\}$
- Incrementing introduces FPs
- Decrementing introduces FNs
- Counter overflows



add(x)

$$++V[h_i(x)] \quad \forall i \in [k]$$

remove(x)

$$--V[h_i(x)] \quad \forall i \in [k]$$

count(x)

$$\min_{i \in [k]} \{V[h_i(x)]\}$$

Spectral Algorithms [CM03]

Minimum Selection (MS)

- Nothing fancy, we use it already for counting Bloom filters

$$m_x = \min_{i \in [k]} \{ V[h_i(x)] \}$$

- MS estimator: $\widehat{f}_x = m_x$
- **Claim 1**: $f_x \leq m_x$ and $\mathbb{P}[f_x \neq m_x] = E_B$

Minimum Increase (MI)

- When adding an item x, only increase the cell(s) with m_x
- **Claim 2**: $E_B^{MI} = O(E_B)$
- **Claim 3**: If x drawn uniformly from U, then

$$E_B^{MI} = \frac{E_B}{k}$$

Spectral Algorithms (cont'd)

Recurring Minimum (RM)

- Observation:
  - Items with high E_B less likely to have recurring minima
  - $\sim$20% of the items have a unique minimum
- Keep track of items with unique minimum in secondary Bloom filter V_2

add(x)

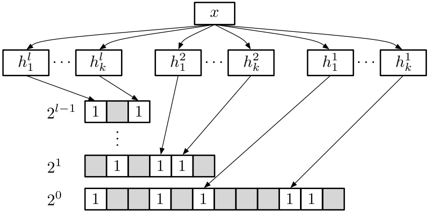
$++V[h_i(x)] \quad \forall i \in [k]$
$m_x \leftarrow \min_{i \in k} V[h_i(x)] \quad \forall i \in [k]$
if x has RM in V **then**
  return
end if
if $x \in V_2$ **then**
  $++V_2[h_i^2(x)] \quad \forall i \in [k_2]$
else
  $V_2[h_i^2(x)] += m_x \quad \forall i \in [k_2]$
end if

count(x)

$m_x \leftarrow \min_{i \in k} V[h_i(x)] \quad \forall i \in [k]$
if x has RM in V **then**
  return m_x
end if
if $x \in V_2$ **then**
  $m'_x \leftarrow \min_{i \in k_2} V[h_i^2(x)] \; \forall i \in [k_2]$
  return m'_x
else
  return m_x
end if

Bitwise Bloom Filter [LO07]

- l basic Bloom filters
- V_i has m_i cells of width w_i
- Counters $c \in \{0, \infty)$
- $\left\{h_j^i : j \in [k_i] \wedge i \in [l]\right\}$
- Both FPs and FNs
- Overflows only across items



add(x)

$i \leftarrow 0$
while $x \in V_i \wedge i < l$ **do**
  $V_i[h_j^i(x)] = 0 \quad \forall j \in [k_i]$
end while
$++V_i[h_j^i(x)] \quad \forall j \in [k_i]$

count(x)

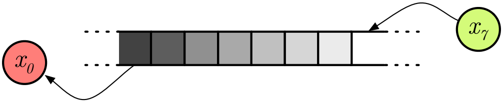
$c \leftarrow 0$
for $i \leftarrow 0$ **to** $l - 1$ **do**
  if $x \in V_i$ **then**
    $c \leftarrow c + 2^l$
  end if
end for
return c

Ageing

- Streaming data: Bloom filters fills up over time

$\rightarrow$ High number of FPs

- Can I haz **sliding window**?



$\rightarrow$ Too expensive to keep old data around

- Want: Bloom Filter behaving like a **FIFO**

Stable Bloom Filter [DR06]

- Basic Bloom filter with m fixed-width cells of size w
- Counters reflect age
  1. Decrement d cells before each insertion
  2. Adding an item x sets its counter to $2^w - 1$

add(x)

1: **for** $i \leftarrow 1$ **to** d **do**
2: Draw $\alpha \sim \mathrm{Unif}\{0, m-1\}$
3: $--V[\alpha]$
4: **end for**
5: $V[h_i(x)] = 2^w - 1 \quad \forall i \in [k]$

- **Stable** property: fraction of zeros will become fixed
- Bloom error when having reached the stable point

$$\phi_P = \left(1 - \left(\frac{1}{1 + \frac{1}{d(1/k - 1/m)}}\right)\right)$$

- Tweak parameters w, k, m, d to achieve the desired ϕ_P

A^2 Buffering [Yoo10]

- Two bit vectors V_1 and V_2 where $|V_1| = |V_2| = \frac{m}{2}$
- Swap both vectors when V_1 becomes full (reached κ_a)
- Bloom error:

$$\phi_{P_a} = 1 - \sqrt{1 - \phi_P}$$

- Optimal k_a and κ_a:

$$k_a^* = \left\lfloor -\log_2 \left(1 - \sqrt{1 - \phi_P}\right) \right\rfloor$$

$$\kappa_a^* = \left\lfloor \frac{m}{2k_a^*} \ln 2 \right\rfloor$$

add(x)

if $x \in V_1$ **then**
  return
end if
$V_1 \leftarrow V_1 \cup \{x\}$
if V_1 has not reached κ_a **then**
  return
end if
Flush V_2
Swap V_1 and V_2
$V_1 \leftarrow V_1 \cup \{x\}$

query(x)

return $x \in V_1 \vee x \in V_2$

Outline

Bloom Filter
- Basic
- Counting
- Spectral
- Bitwise
- Stable
- A^2

Implementation

Evaluation

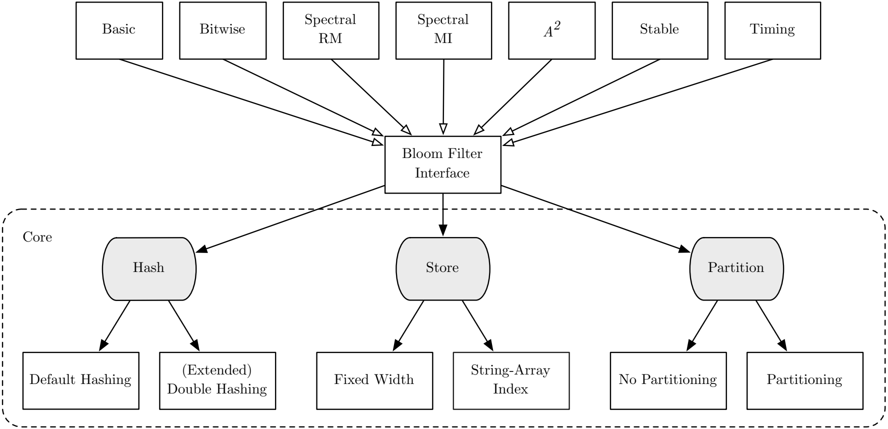
`libBf`: Bloom Filter Library in C++11

Implementation of 6 Bloom filters

1. A^2
2. Basic (+ counting)
3. Bitwise
4. Spectral (MI)
5. Spectral (RM)
6. Stable

- Policy-based design
  - **Hash**: computes hash values
  - **Store**: provides $O(1)$ random-access counter storage
  - **Partition**: maps hash values to cells
- Easy to use
  - Header-only
  - BSD-style license
  - Interface fully documented (Doxygen)
  - Available at `https://github.com/mavam/libBf`

`libBf`: Policy-Based Architecture



- Modular: cleanly layered
- Fast: static polymorphism (CRTP)
- Safe: fail early at compile time (type-traits, SFINAE)

Build-Your-Own Bloom Filter with `libBf`

1. Define a core type

```
typedef core<
    fixed_width<uint8_t, std::allocator<uint8_t>>
  , double_hashing<default_hasher, 42, 4711>
  , no_partitioning
> my_core;
```

2. Define a Bloom filter type

```
typedef basic<my_core> my_bloom_filter;
```

3. Instantiate with a core

```
my_bloom_filter bf({ 1 << 10, 5, 4 });
```

4. Use

```
bf.add("foo")
bf.add("foo")
bf.add('z')
bf.add(3.14159)
std::cout << bf.count("foo") << std::endl; // returns 2
```

The Bliss of C++11

- Type inference:

```
auto i = std::unordered_map<int, int>().begin();
decltype(i) j;
```

- Lambda functions:

```
[&](int i) -> bool { return i % 42; }
```

- Rvalue references:

```
template <typename Core>
bloom_filter(Core&& core) { ... }
bloom_filter bf({ 128, 5, 4 });
```

- Range-based `for` loops:

```
for (auto i : { 2, 4, 8, 16 })
    f(i * 2);
```

- Type traits for metaprogramming
- Beefed-up STL: RNGs, distributions, hashing,. . .

Outline

Bloom Filter

- Basic
- Counting
- Spectral
- Bitwise
- Stable
- A^2

Implementation

Evaluation

Evaluation

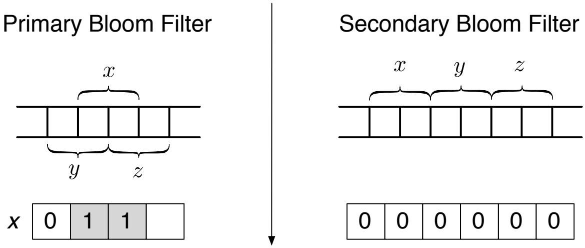
- Analyze correctness

$\rightarrow$ Recurring minimum (RM) seems to have a bug

- How does this garden variety of Bloom filters perform?

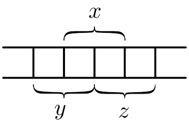
$\rightarrow$ Compare performance metrics (FP, FN, TP, TN) across BFs

Spectral Bloom Filter RM Bug



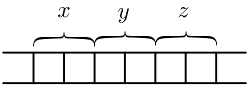
Spectral Bloom Filter RM Bug

Primary Bloom Filter



x	0	1	1	
y	1	2	1	

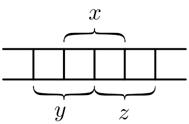
Secondary Bloom Filter



	0	0	0	0	0	0
y	0	0	1	1	0	0

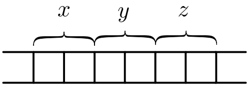
Spectral Bloom Filter RM Bug

Primary Bloom Filter



x	0	1	1	
y	1	2	1	
x	1	3	2	0

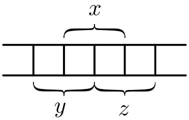
Secondary Bloom Filter



	0	0	0	0	0	0
y	0	0	1	1	0	0
x	2	2	1	1	0	0

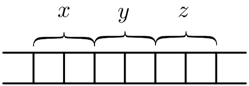
Spectral Bloom Filter RM Bug

Primary Bloom Filter



x	0	1	1	
y	1	2	1	
x	1	3	2	0
z	1	3	3	1

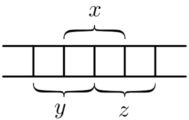
Secondary Bloom Filter



	0	0	0	0	0	0
y	0	0	1	1	0	0
x	2	2	1	1	0	0
z	2	2	1	1	1	1

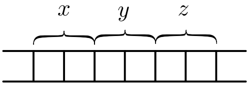
Spectral Bloom Filter RM Bug

Primary Bloom Filter



x	0	1	1	
y	1	2	1	
x	1	3	2	0
z	1	3	3	1
x	1	4	4	1

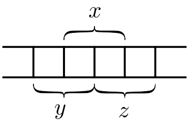
Secondary Bloom Filter



	0	0	0	0	0	0
y	0	0	1	1	0	0
x	2	2	1	1	0	0
z	2	2	1	1	1	1
	2	2	1	1	1	1

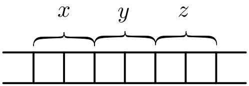
Spectral Bloom Filter RM Bug

Primary Bloom Filter



x	0	1	1	
y	1	2	1	
x	1	3	2	0
z	1	3	3	1
x	1	4	4	1
y	2	5	4	1

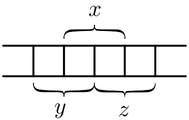
Secondary Bloom Filter



	0	0	0	0	0	0
y	0	0	1	1	0	0
x	2	2	1	1	0	0
z	2	2	1	1	1	1
	2	2	1	1	1	1
y	2	2	2	2	1	1

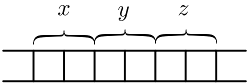
Spectral Bloom Filter RM Bug

Primary Bloom Filter



x	0	1	1	
y	1	2	1	
x	1	3	2	0
z	1	3	3	1
x	1	4	4	1
y	2	5	4	1
x	2	6	5	1

Secondary Bloom Filter



	0	0	0	0	0	0
y	0	0	1	1	0	0
x	2	2	1	1	0	0
z	2	2	1	1	1	1
	2	2	1	1	1	1
y	2	2	2	2	1	1
x	3	3	2	2	1	1

Bug

Item x was inserted 4 times, but spectral RM as in the paper reports 3, which is *not* an upper bound on the actual value.

Spectral Bloom Filter RM Bug

- Implications: Claim 1 does not hold for spectral RM.

$\rightarrow$ FNs *can* occur

- "Optimization:" keep track of items in 2^{nd} BF via 3^{rd} BF
- Equivalent to always looking in both BFs
- Not really an optimization

Experimentation

- Is it still possible to look up the 2^{nd} BF only for unique minimum?
- Let m_x^i be the count estimate of x in BF i
- We played with functions $g(m_x^1, m_x^2)$ to reduce FNs
- Our finding: significantly reduced FN rates for

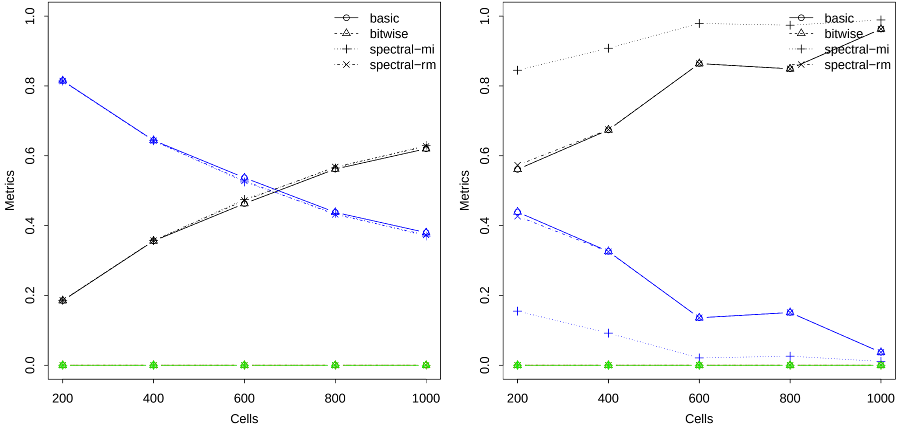
$$g(x,y) = \frac{x+y}{2}$$

$\rightarrow$ Performance: better FN rates, lookup only 20% of the time

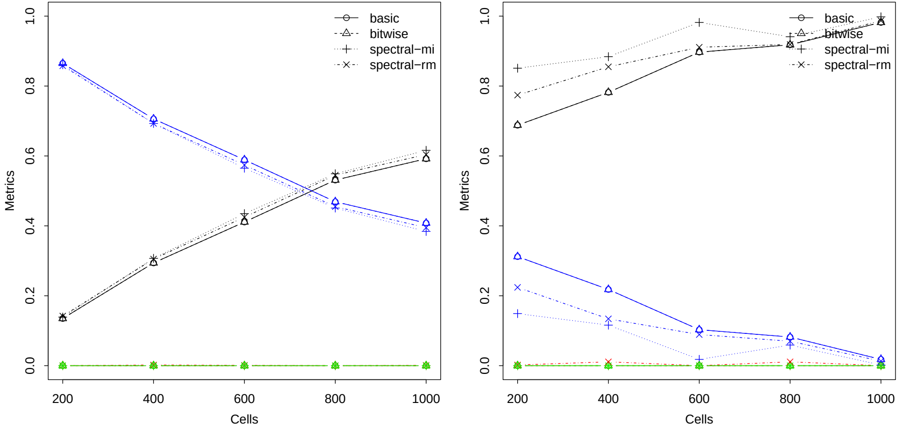
Performance Analysis

- Compare FP (blue), FN (red), TP (black), TN (green) rates as a function of space
- Very preliminary analysis
- Synthetic data from two discrete distributions
  - $\mathrm{Unif}\{0, 1000\}$ (left panel)
  - $\mathrm{Zeta}(1.5)$ (right panel)
- Fixed parameters: $w = 17$, $n = 1000$

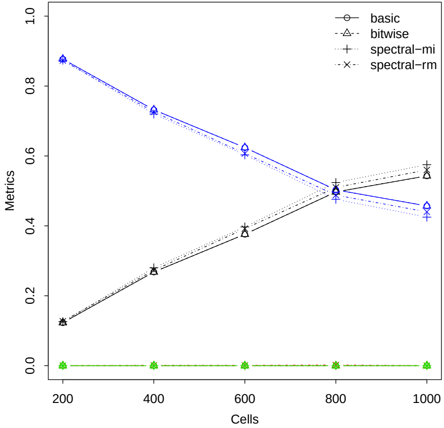
Metrics for $k = 2$ and $w = 17$

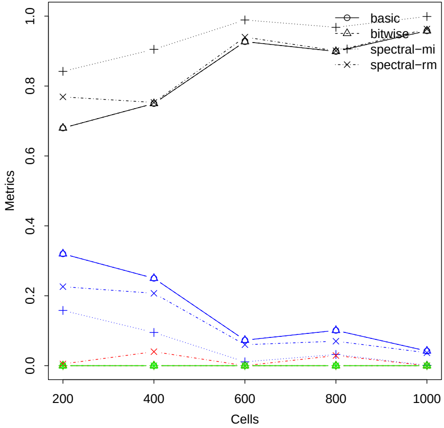


Metrics for $k = 3$ and $w = 17$

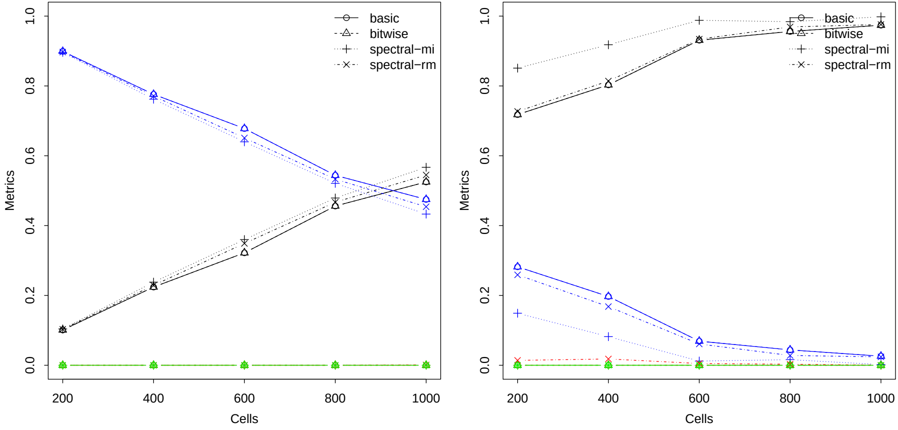


Metrics for $k = 4$ and $w = 17$





Metrics for $k = 5$ and $w = 17$



Summary

- Studied a variety of different Bloom filter types
- Implemented and published `libBf`, a C++11 Bloom filter library
- Started to study the trade-offs in the parameter space
- Next steps: more rigorous performance measurements needed

References I

Burton H. Bloom.
Space/Time Trade-offs in Hash Coding with Allowable Errors.
Commun. ACM, 13:422–426, July 1970.

Saar Cohen and Yossi Matias.
Spectral Bloom Filters.
In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 241–252, New York, NY, USA, 2003. ACM.

Fan Deng and Davood Rafiei.
Approximately Detecting Duplicates for Streaming Data using Stable Bloom Filters.
In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 25–36, New York, NY, USA, 2006. ACM.

References II

Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder.
Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol.
In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '98, pages 254–265, New York, NY, USA, 1998. ACM.

Ashwin Lall and Mitsunori Ogihara.
The Bitwise Bloom Filter.
Technical Report TR-2007-927, University of Rochester, November 2007.

MyungKeun Yoon.
Aging bloom filter with two active buffers for dynamic sets.
IEEE Trans. Knowl. Data Eng., 22(1):134–138, 2010.

Backup Slides

Bloom Filter Halving