Bloom filters

Approximate membership data structures

Bloom filters [Bloom 1970]: generalities

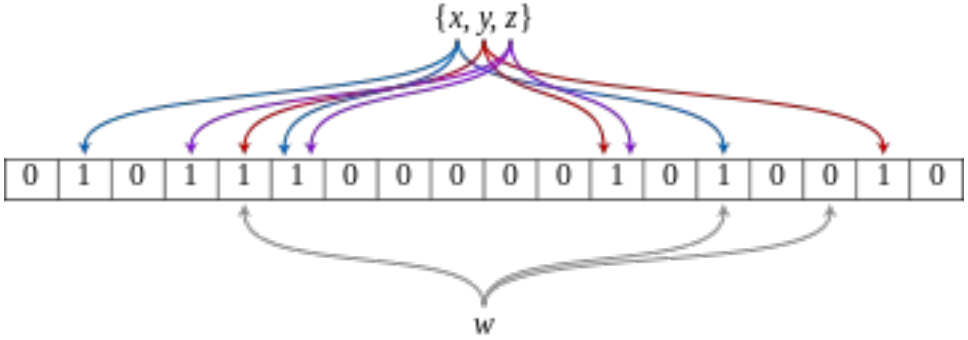
- *approximate membership data structure*: supports INSERT and MEMBER
- MEMBER only checks for the presence, no satellite data
- produces false positives (with controlled probability)

- cannot iterate over the elements of the set
- DELETE is not supported (in the basic variant)

- *very* space efficient, keys themselves are *not* stored
- *Example*: forbidden passwords

Bloom filter: how it works

- $\mathcal{U}$: universe of possible keys
- S : subset of keys, $|S| = n$
- m : size of allocated *bit array* B



- define k hash functions $h_1, \ldots, h_k$: $\mathcal{U} \rightarrow \{0, \ldots, m-1\}$

- INSERT(x): set $B[h_i(x)] = 1$ for all i
- MEMBER(x): check $B[h_i(x)] = 1$ for all i

- false positives but no false negatives

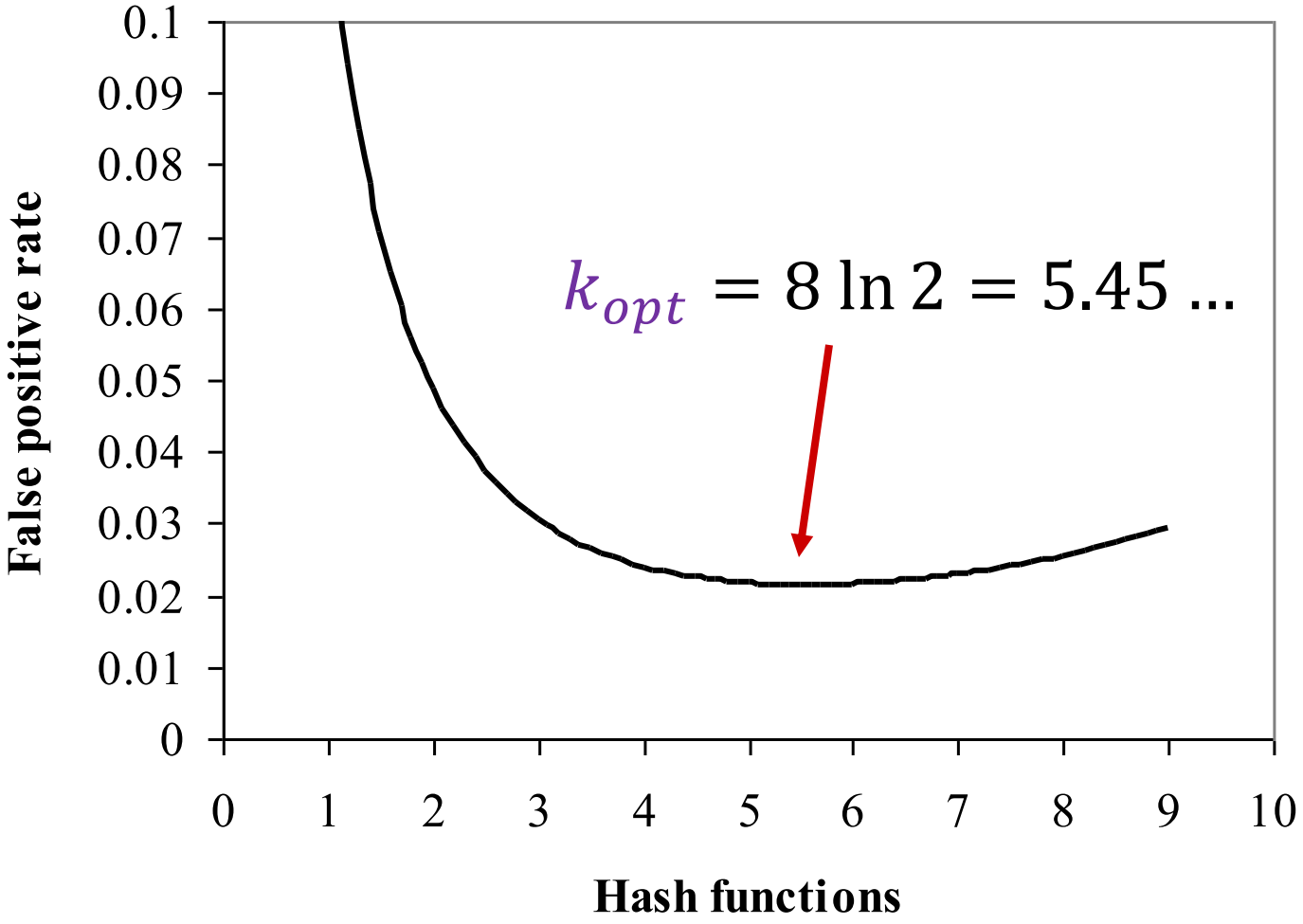
Bloom filters: analysis

- P[specific bit of filter is 0] $= (1 - 1/m)^{kn} \approx e^{-kn/m} \equiv p$
- P[false positive] $= (1 - p)^k = (1 - e^{-kn/m})^k$
- Optimal number of hash functions: $k_{opt} = \ln 2 \cdot \frac{m}{n} \approx 0.693 \cdot \frac{m}{n}$
- Therefore, for $k = k_{opt}$,

$$\text{P[false positive]} = 2^{-\ln 2 \cdot \frac{m}{n}} \approx 0.6185^{\frac{m}{n}}$$

- E.g. with 10 bits per element, P[false positive] is less than 1%
- To insure the FP rate ε: $m = \log_2 e \cdot n \cdot \log_2 \frac{1}{\varepsilon} \approx 1.44 \cdot n \cdot \log_2 \frac{1}{\varepsilon}$

Dependence on the nb of hash functs



$m/n = 8$

n elements
m bits
k hash functions

Lower bound on the size of approximate membership data structures (AMD)

- Bloom filter takes $1.44 \cdot \log \frac{1}{\varepsilon}$ bits per key, is this optimal?

- How many AMDs are there to store all sets of size n drawn from universe $\mathcal{U}$ with FPP ε?
- Each AMD specifies a set of size $\varepsilon|\mathcal{U}|$ (assuming $|\mathcal{U}|$ large) containing a set of size n
- Any set of size n should be covered, and the number of such sets is $\geq \binom{|\mathcal{U}|}{n} / \binom{\varepsilon|\mathcal{U}|}{n} \approx \left(\frac{1}{\varepsilon}\right)^n$
- $\Rightarrow$ each FPP must take $\geq n \cdot \log \frac{1}{\varepsilon}$ bits

Bloom filter: properties/operations

- For the optimal number of hash function, about a half of the bits is 1 [*immedate from the formula*]
- The Bloom filter for the union is the OR of the Bloom filters
- Is similar true for the intersection? [*explain*]
- If a Bloom filter is sparse, it is easy to halve its size

Bloom filters: applications

- Bloom filters are very easy to implement
- apply to the streaming mode
- Used e.g. for
  - spell-checkers (in early UNIX-systems)
  - unsuitable passwords, "approximate" unsuitable passwords (Manber&Wu 1994)
  - online applications (traffic monitoring, …)
  - distributed databases
  - malicious sites in Google Chrome
  - read articles in publishing systems (Medium)
  - Google Bigtable, Apache HBase, Bitcoin, bioinformatics, …
- Sometimes (when the set of possible queries is predefined) it is possible to store the set of false positives in a separate data structure

Cuckoo filters

filters via Cuckoo hashing

General idea: use *fingerprints*

- Given ε, pick a hash function

$$f: \mathcal{U} \longrightarrow [0..2^{\log\frac{1}{\varepsilon}}-1] \quad \leftarrow \textit{fingerprints}$$

- $P[f(x) = f(y)] = \frac{1}{2^{\log\frac{1}{\varepsilon}}} = \varepsilon$ (collision probability)

Filters via MPHF

- Given a set $S \subset \mathcal{U}$, build an MPHF $h: S \longrightarrow [0..n-1]$
- Build an array F of fingerprints: $F[h(x)] = f(x)$

$$\overbrace{n \cdot \log \frac{1}{\varepsilon}}^{F} + \text{<size of MPFR>}$$

- Space: $n \cdot \log \frac{1}{\varepsilon} +$ <size of MPFR>
- *lower bound*: $\text{size of MPFR} \geq 1.44n$
- S must be static, insertions/deletions are not supported

Cuckoo filter: ideas

- Use Cuckoo hash table (e.g. (2,4)-table) instead of MPHF
- Supports insertions and deletions (assuming no collision)!
- *Problem*: How to move a fingerprint? i.e. how to know its alternative bucket?

Cuckoo filter: ideas

- Use Cuckoo hash table (e.g. (2,4)-table) instead of MPHF

- *Problem*: How to move a fingerprint? i.e. how to know its alternative bucket?
- Let $|T| = 2^t$

$$h_1 : S \to [0..2^t - 1],$$

$$h_2 : [0..2^{\log\frac{1}{\varepsilon}} - 1] \to [0..2^t - 1]$$

$$\text{location } 1 : h_1(x)$$

$$\text{location } 2 : h_1(x) \oplus h_2(f(x))$$

- Alternative location of a fingerprint α at location i is

$$i \oplus h_2(\alpha)$$

Remarks

- Deletions are supported!
- Two locations of a key are not fully independent. E.g. two keys sharing the same bucket and the same fingerprint have the same alternative location. (⇒ store *multisets* in b-element buckets)
- *Practical*: Cuckoo vs. Bloom: for small false positive rate ($< 3\%$) and $b = 4$, Cuckoo filter achieves the same performance as Bloom with smaller space

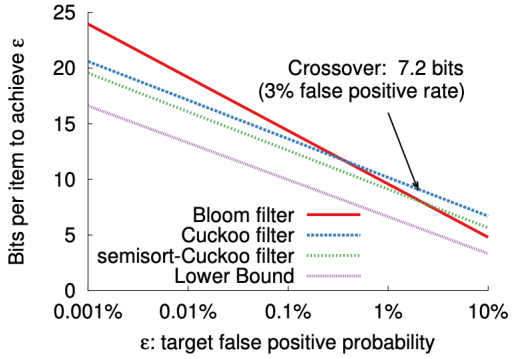


Figure 4: False positive rate vs. space cost per element. For low false positive rates (< 3%), cuckoo filters require fewer bits per element than the space-optimized Bloom filters. The load factors to calculate space cost of cuckoo filters are obtained empirically.

[Fan et al. Cuckoo filter: practically better than Bloom, CoNEXT 2014]