Bloom Filters

References

A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Mathematics*, vol. 1 no. 4, pp. 485-509, 2004.

Li Fan, Pei Cao, Jussara Almeida, Andrei Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Transactions on Networking*, Vol. 8, No. 3, June 2000.

- Origin of counting Bloom filters

Origin and applications

- Randomized data structure introduced by Burton Bloom [CACM 1970]
  - It represents a set for membership queries, with false positives
  - Probability of false positive can be controlled by design parameters
  - When space efficiency is important, a Bloom filter may be used if the effect of false positives can be mitigated.
- First applications in dictionaries and databases

First application in networking: distributed cache (2000)

Proxy 1
Cache 1
Summary 2
Summary 3

Proxy 2
Cache 2
Summary 1
Summary 3

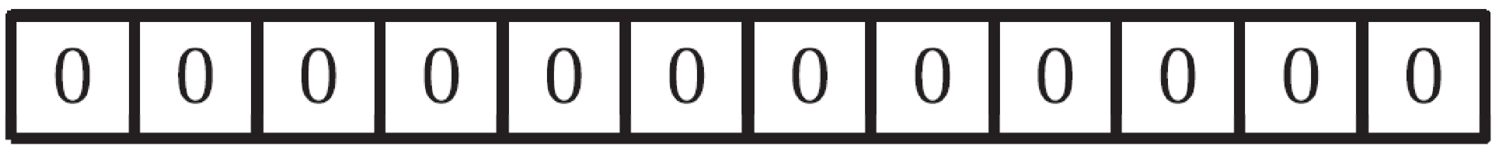
Proxy 3
Cache 3
Summary 1
Summary 2

- Numerous applications in networking since 2000

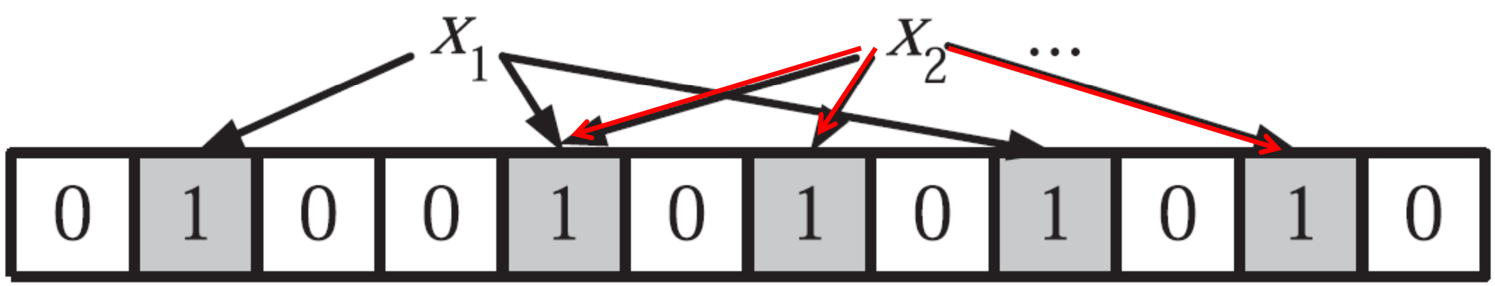
Standard Bloom Filter

- A Bloom filter is an *array of m bits* representing a set $S = \{ x_1, x_2, \ldots, x_n\}$ of *n* elements
  - Array set to 0 initially
- *k* independent hash functions $h_1, \ldots, h_k$ with range $\{1, 2, \ldots, m\}$
  - Assume that each hash function maps each item in the universe to a random number *uniformly* over the range $\{1, 2, \ldots, m\}$
- For each element x in S, the bit $h_i(x)$ in the array is set to 1, for $1 \leq i \leq k$,
  - A bit in the array may be set to 1 multiple times for different elements

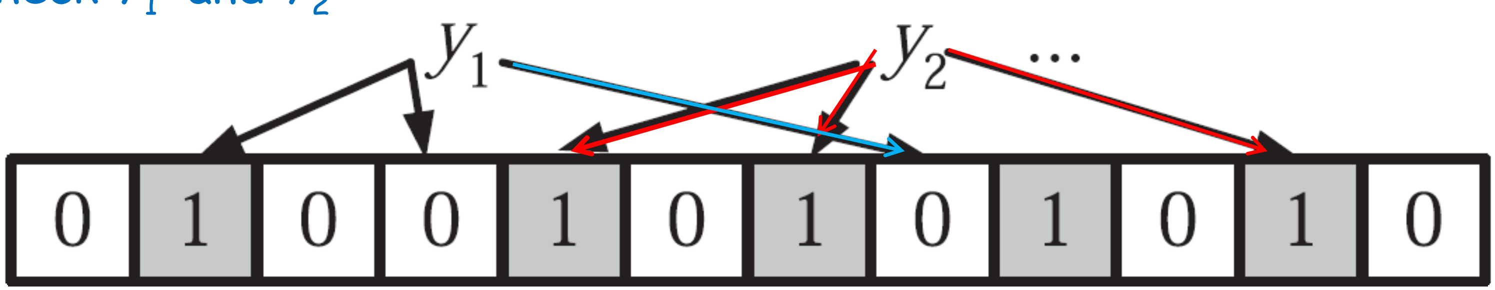
A Bloom filter example
(three hash functions)



Insert X_1 and X_2



Check Y_1 and Y_2



Standard Bloom Filter (cont.)

- To check membership of y in S, check whether $h_i(y)$, $1 \leq i \leq k$, are all set to 1
  - If not, y is definitely not in S
  - Else, we conclude that y is in S, but sometimes this conclusion is wrong (false positive)
- For many applications, false positives are acceptable as long as the probability of a false positive is small enough
- We will assume that $kn < m$

False positive probability

- After all members of S have been hashed to a Bloom filter, the probability that a specific bit is still 0 is

$$p' = (1 - \frac{1}{m})^{kn} \simeq e^{-kn/m} = p$$

- For a non member, it may be found to be a member of S (all of its k bits are nonzero) with *false positive probability*

$$(1 - p')^k \simeq (1 - p)^k$$

False positive probability (cont.)

- Define

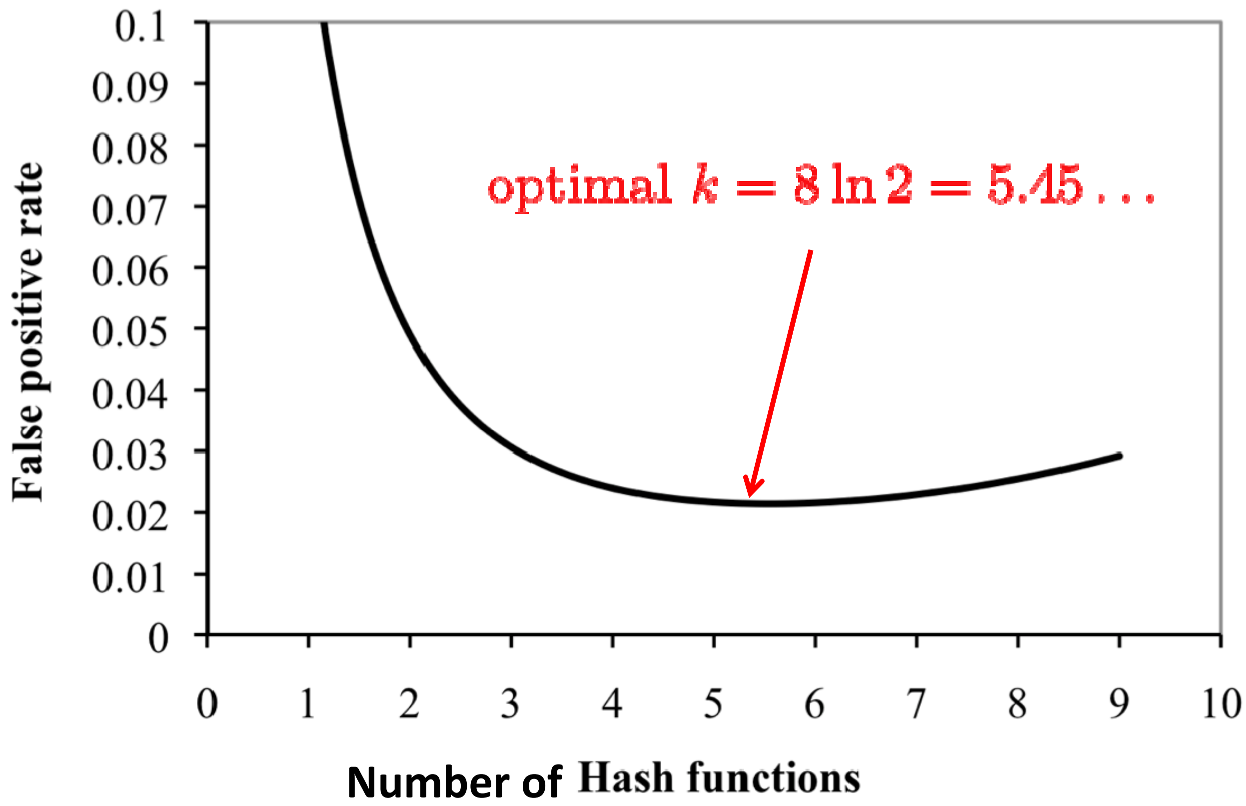
$$f' = (1 - p')^k = (1 - (1 - \frac{1}{m})^{kn})^k$$

$$f = (1 - p)^k = (1 - e^{-kn/m})^k$$

- Two competing forces as k increases
  - Larger k -> $(1 - p')^k$ is smaller for a fixed p'
  - Larger k -> p'= $(1 - 1/m)^{kn}$ is smaller -> 1-p' larger

False positive rate vs. *k*

Number of bits per member $\frac{m}{n} = 8$



Optimal number k from derivative

Rewrite f as

$$f = \exp(\ln(1-e^{-kn/m})^k) = \exp(k\ln(1-e^{-kn/m}))$$

Let $g = k\ln(1-e^{-kn/m})$

Minimizing g will minimize $f = \exp(g)$

$$\frac{\partial g}{\partial k} = \ln(1-e^{-kn/m}) + \frac{k}{1-e^{-kn/m}} \frac{\partial(1-e^{-kn/m})}{\partial k}$$

$$= \ln(1-e^{-kn/m}) + \frac{k}{1-e^{-kn/m}} \frac{n}{m} e^{-kn/m} = -\ln(2) + \ln(2) = 0$$

if we plug $k = (m/n)\ln 2$ which is optimal

(It is in fact a global optimum)

Optimal *k* from symmetry

- Alternatively, from $p = e^{-kn/m}$ we get

$$k = -\frac{m}{n}\ln(p)$$

From previous slide, we have

$$g = k\ln(1 - e^{-kn/m}) = -\frac{m}{n}\ln(p)\ln(1-p)$$

- From above, symmetry indicates that the minimum value for g occurs when p=1/2.

Thus

$$k_{opt} = -\frac{m}{n}\ln(1/2) = \frac{m}{n}\ln(2)$$

Optimal k from symmetry

using the precise probability of false positive

$$f' = (1 - p')^k = \exp(k \ln(1 - p'))$$

From $p' = (1 - 1/m)^{kn}$, solving for k

$$k = \frac{1}{n \ln(1 - 1/m)} \ln(p')$$

Let $g' = k \ln(1 - p')$ (in equation for f' above)

$$= \frac{1}{n \ln(1 - 1/m)} \ln(p') \ln(1 - p')$$

Using the precise probability of false positive to get optimal k (cont.)

- From previous slide

$$g' = \frac{1}{n \ln(1 - 1/m)} \ln(p') \ln(1 - p')$$

- By symmetry, g′ (also f′) minimized at p′=1/2

- Optimal *k* is

$$k'_{opt} = \frac{1}{n \ln(1 - 1/m)} \ln(p') = \frac{1}{n \ln(1 - 1/m)} \ln(1/2)$$

Optimal number of hash functions

- Using $k_{opt} = \frac{m}{n}\ln(2)$ the false positive rate is

$$(1-p)^{\frac{m}{n}\ln(2)} = (0.5)^{\frac{m}{n}\ln(2)} \simeq (0.6185)^{m/n}, \text{ where } \ln(2) = 0.6931$$

- In practice, k should be an integer. May choose an integer value smaller than k_{opt} to reduce hashing overhead

m/n denotes bits per entry

False positive rate

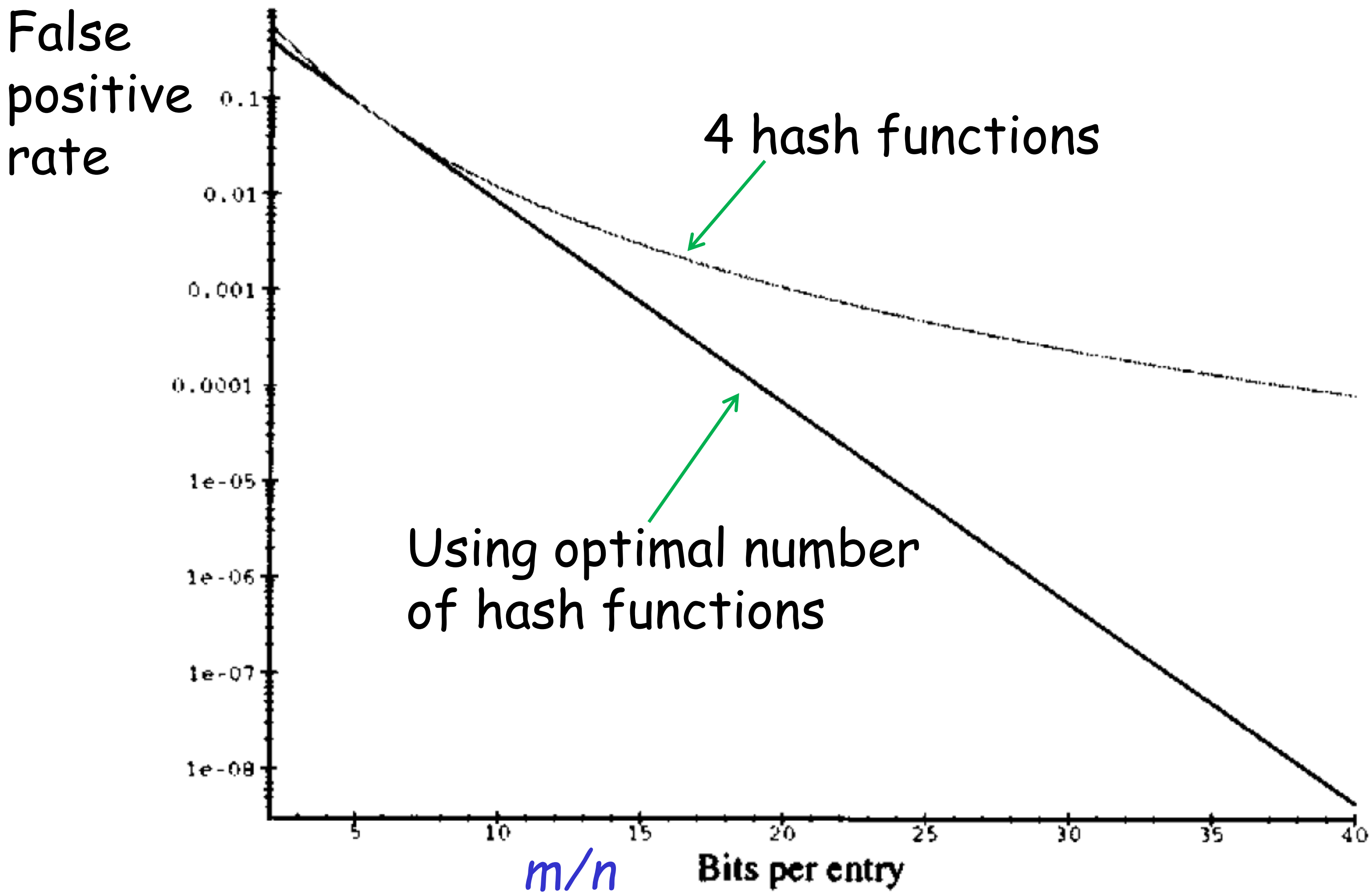
$$m/n = 6 \quad k = 4 \quad p_{\text{error}} = 0.0561$$

$$m/n = 8 \quad k = 6 \quad p_{\text{error}} = 0.0215$$

$$m/n = 12 \quad k = 8 \quad p_{\text{error}} = 0.00314$$

$$m/n = 16 \quad k = 11 \quad p_{\text{error}} = 0.000458$$

False positive rate vs. bits per entry



Standard Bloom Filter tricks

- Two Bloom filters representing sets S_1 and S_2 with the same number of bits and using the same hash functions.
  - A Bloom filter that represents the union of S_1 and S_2 can be obtained by taking the OR of the bit vectors
- A Bloom filter can be halved in size. Suppose the size is a power of 2.
  - Just OR the first and second halves of the bit vector
  - When hashing to do a lookup, the highest order bit is masked

Notation: OR denotes bitwise *or*

Counting Bloom filters

- Proposed by Fan et al. [2000] for distributed caching
- Every entry in a counting Bloom filter is a small counter (rather than a single bit).
  - When an item is inserted into the set, the corresponding counters are each incremented by 1
  - When an item is deleted from the set, the corresponding counters are each decremented by 1
- To avoid counter overflow, its size must be sufficiently large. It was found that 4 bits per counter are enough.

Counter overflow probability

- Consider a set of n elements, k hash functions, and m counters
  - C(i) is the count for the i^th counter

$$P[c(i) = j] = \binom{nk}{j}\left(\frac{1}{m}\right)^{j}\left(1-\frac{1}{m}\right)^{nk-j}$$

$$P[c(i) \geq j] \leq \binom{nk}{j}\frac{1}{m^{j}}$$

$$\leq \left(\frac{enk}{jm}\right)^{j} \quad \text{(a very loose upper bound)}$$

Counter overflow probability (cont.)

- Choose k such that $k \le m/n$ (ln 2)

  Then

$$P[c(i) \ge j] \le \left(\frac{enk}{jm}\right)^j \le \left(\frac{e \ln 2}{j}\right)^j$$

$$P[\max_{1 \le i \le m} c(i) \ge j] \le m \left(\frac{e \ln 2}{j}\right)^j \quad \text{for some } i$$

- Using 4 bits, each counter counts from 0 to 15

$$P[\max_{1 \le i \le m} c(i) \ge 16] \le m \times 1.37 \times 10^{-15}$$

Counter overflow consequences

- When a counter does overflow, it may be left at its maximum value.
- This can later cause a false negative *only if* eventually the counter goes down to 0 when it should have remain at nonzero.
- The expected time to this event is very large but is something we need to keep in mind for any application that does not allow false negatives

Conclusions

- Wherever a list or set is used, and space is at a premium, a Bloom filter may be used if the effect of false positives can be mitigated
  - No false negative
- With a counting Bloom filter, false negatives are possible, albeit highly unlikely

The End