Lecture 8 — October 1

Lecturer: John Byers *BOSTON UNIVERSITY* *Scribe: Ilir Çapuni*

In this lecture we reviewed the basic idea of Bloom filters and gave more precise definition and analysis of the false positive probability. We considered a simple application where Bloom filters are used [2].

We also gave a motivating discussion of yet another example of Bloom filters - IP traceback [3].

8.1 Bloom filters

We recall the general idea of Bloom filters as a method how to represent a bag of integers with few bits.

Let n be a number of integers, m the number of bits and k the number of hash functions.

Formally, a Bloom filter is a bit vector v of m bits with k independent hash functions $h_1(x), \ldots, h_k(x)$.

We assume that $m < n$. In particular $kn < m$. What are optimal design settings values for k once we know how many integer we want to store in some storage of m bits? The goal of our design is to minimize false positive probability. The false positive probability is easy to calculate. The probability that one hash functions misses one bit is $(1 - 1/m)$. Assuming that hash functions are independent and random, after all of them have "done their job", the probability that this particular bit is till zero is

$$\left(1 - \frac{1}{m}\right)^{kn}.$$

Finally, false positive probability is

$$Pr[\text{All } k \text{ bits are 1}] = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k}.$$

We now recall one very useful fact: $(1 - 1/x)^y \approx e^{-y/x}$. Using this fact, we have

$$Pr[\text{ False positive }] = \left(1 - e^{-kn/m}\right)^{k}.$$

Now, let us minimize the false positive probability as a function of number of hash functions k.

We consider the function $f(x) = \left(1 - e^{-xn/m}\right)^{x}$. Since ln is monotonic and continous function, finding the minima of f is the same of finding the minima $\ln f(x)$ but more easier.

Let $g(x) = \ln f(x) = x(1 - e^{-xn/m})$. We find $g'(x) = \ln\left(1 - e^{-xn/m}\right) + \frac{xn}{m} \cdot \frac{e^{-xn/m}}{1 - e^{-xn/m}}$.

We find that $g'(x) = 0$ if $x = m/n \ln 2$. For this particular value, $g''(x) > 0$ and furthermore, g is monotonic and positive, so x is a global minimum of g.

 x here is a real number and our k is an integer, so we can take a floor of x to be our k.

The setting we presented above balances zeroes and ones. It makes them approximately half and half.

Using aforementioned, some reasonable values for m, n and k are:

1. $m/n = 6$, $k = 4$, $Pr[$ False positive $] = 0.05$
2. $m/n = 8$, $k = 6$, $Pr[$ False positive $] = 0.02$

8.1.1 A Sample Application

We now consider the case of a huge set of URLs and we denote it U. Let $n = |U|$. Let each URL be 800 bits long. In order to represent U, we need $800n$ bits.

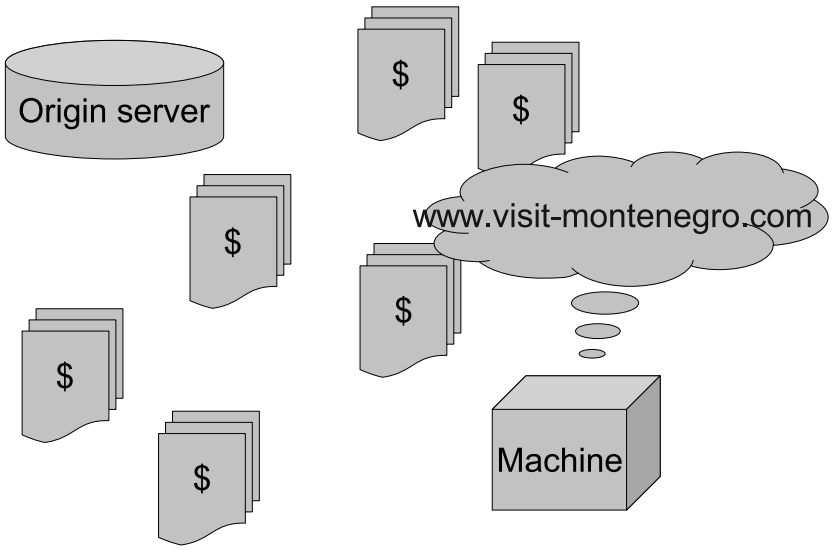

Figure 8.1. A simple shared caching structure

Now consider the following basic caching structure from *Fan et al.* where there is a group of caches each containing a set of documents indexed by URLs. When the machine asks to retrieve a web document, it asks first the local cache and in the case of a cache miss it will consult the distributed cache. This protocol is called Internet cache protocol and comes from the early days of the web.

This simple calculation shows that this is very inefficient.

Suppose that cache consists of N servers and that the average cache hit ratio is H. With these parameters, each cache needs to handle $(N-1)(1-H)R$ inquiries from neighboring caches. On the network, there will be $N(N-1)(1-H)R$ ICP inquiries.

One idea to fix is to have caches periodically distribute the summary of the contents changes. In addition it should take advantage of hierarchy to compress the namespace. A good idea how to distribute changes is to take note of differences that occurred. In addition to all these possible fixes, we can also use Bloom filters: each cache represents its documents set $D_i(t)$ at the moment t as a Bloom filter. Filters then are shared to the distributed cache. Now, each URL which takes used to take hundred of bits now is stored with 6 or 8 bits.

Problem. There is a problem here. Namely, Bloom filter do not have deletion.

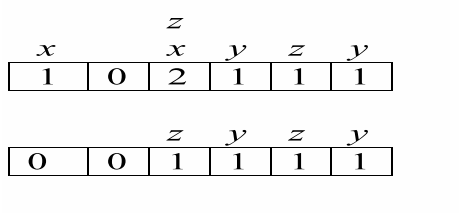

Figure 8.2. Counting Bloom filter with 3 elements inserted. On the bottom: after the deletion of x.

A fix for this problem is *Counting Bloom filter*: as we insert/delete stuff in/from the filter, we increment/decrement values on it. So, now instead of bits we are using integers from the set $\{0, 1, \ldots, c\}$ where c is the capacity of each cell. Say, we can use 4 bits for a cell as a counter.

8.1.2 Remarks

There has been a lot of works in Bloom filters. New concepts like *Spectral Bloom filters*, *Space-code Bloom filter*, *Bloomier Filter*... have been introduced.

8.2 IP Traceback

Not only good packets travel through internet. When a packet has been considered as malicious by the receiver, it is often useful to trace back the route it came from and to identify the origin. Victim wants to figure out the path (which could be a graph) that this packet took.

A naive idea is to let routers store each packet for some period of time. Then, the victimized computer can launch querying routers above it if they have seen this packet and depending on their answer this process will continue. However, routers are usually limited

with space so instead of storing the whole packet, we will store only its digest (important fields) using Bloom filter, trading certainty for efficiency and space. Now, a router when asked if it has seen a packet x, it could respond with NO or YES whereas false positive answers are possible. Also, it is very dangerous to store packets since that to be a reason for a router to be attacked.

Bibliography

[1] B. Bloom. "*Space/time trade-offs in hash coding with allowable errors*," Communications of the ACM, 13(7):422-426, 1970.

[2] L. Fan, P. Cao, J. Almeida and A. Z. Broder, "*Summary Cache: A Scalable Wide-area Cache Sharing Protocol*," in Proceedings of ACM SIGCOMM '98.

[3] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, W. Strayer, "Hash-Based IP Traceback," in Proceedings of ACM SIGCOMM '01.