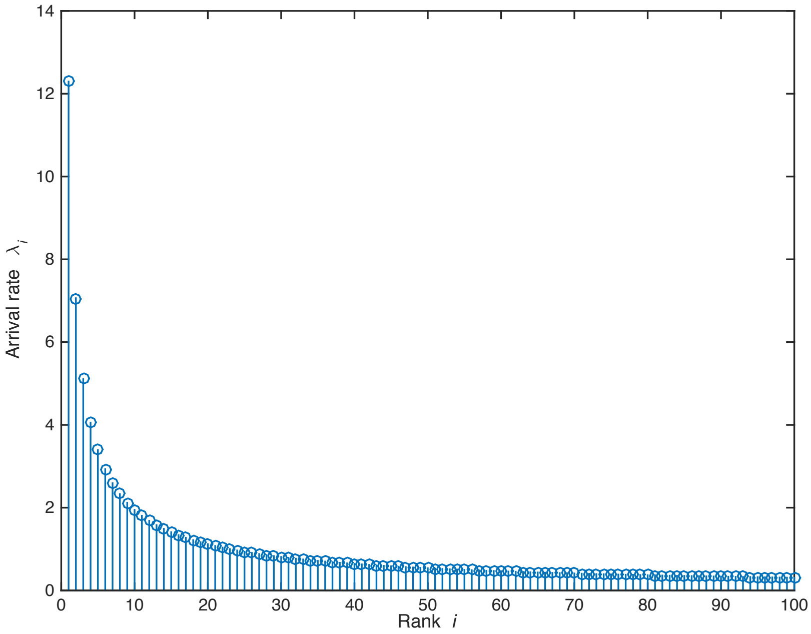
Ubinet master – Multimedia Networking Module
Exercises: Content storage

Exercise 1: Optimal cache dimensioning

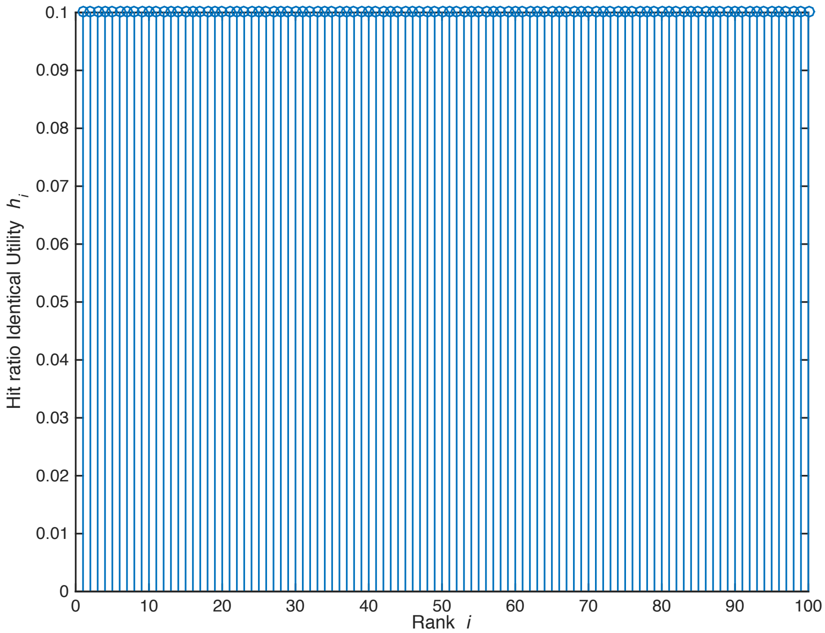
Let's assume that a CDN uses Time-To-Live (TTL) memories of size B = 10 video files to catch a catalog of $N=100$ files with the same size. The arrival rates λ_i (related to the popularity of the file i) are known for the catalog files and it follows the following Zipf-like Law of parameter s=0.8 and total number of arrival per second $\Lambda = 100$:

$$\lambda_i = \Lambda\, i^{-s} / \sum^{N}_{n=1} n^{-s} \text{ video requests / s.}$$



1. In a first moment, the CDN assumes that all the files have same cache utility function $U_i\,(h_i)$ depending on the *hitting probability or ratio* (equivalent to the average proportion along time that the file i is cached) h_i. In other words, it does not consider that the file popularity λ_i has any impact on its profit. Under these assumptions, for each content i, which value should be used as *TTL timer* t_i and what is the *hitting ratio* h_i ? Are all the *TTL timer* t_i equal? Are all the *hitting ratio* h_i equal? Why?

   Ans: Since video popularity has not impact on the cache utility, all the files have the same importance and all of them should be cached with the same probability h_i = B/N = 10/100 = 0.1.



However, the optimal TTL time for each file still depends on the individual request arrival rate of each file at the cache. Then, for non-reset TTL cache,

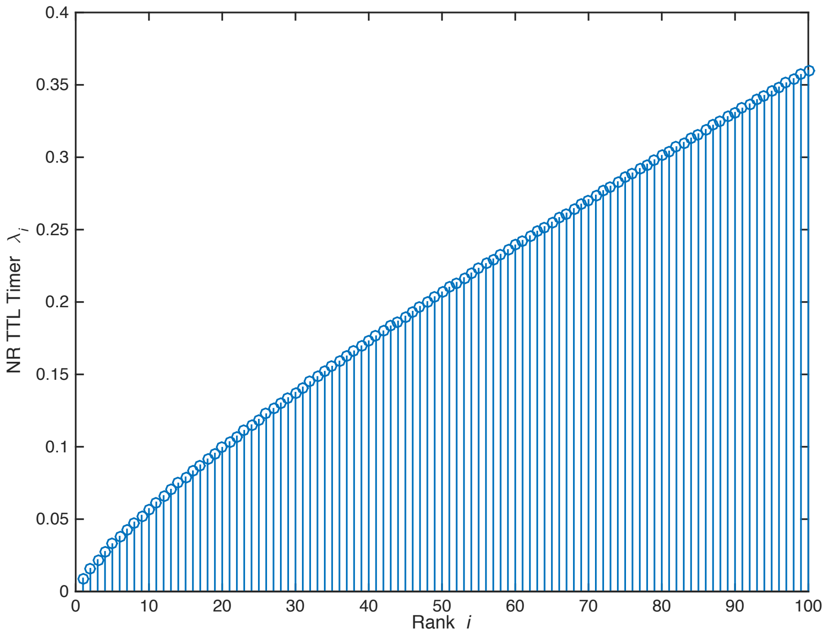
$$t_i = \frac{B}{\lambda_i(N-B)}$$

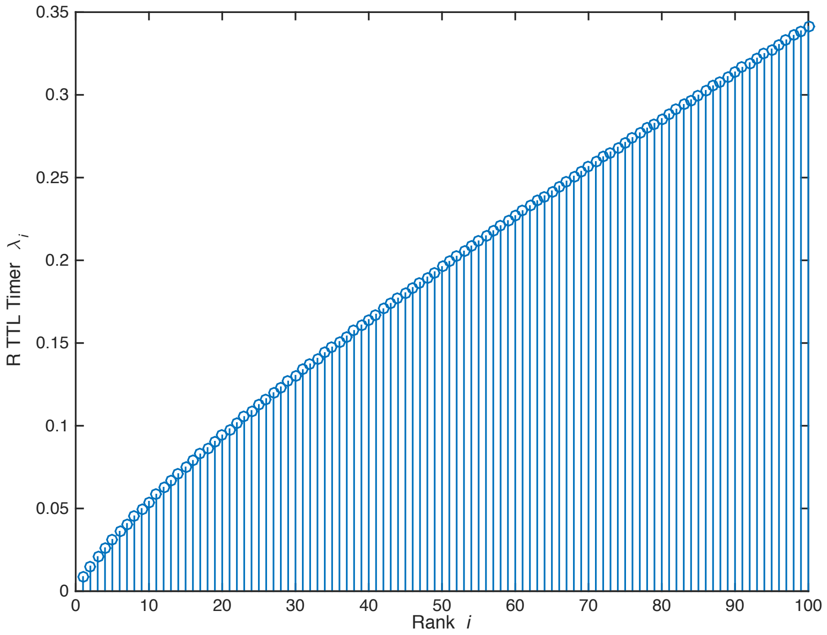
and for reset TTL cache

$$t_i = -\frac{1}{\lambda_i}\log\left(1-\frac{B}{N}\right).$$

Rank i	**Hit Ratio (*h_i*)**	**Non reset TTL timer (*t_i*)**	**Reset TTL timer (*t_i*)**
1	0.1	9.0 ms	8.6 ms
2	0.1	15.7 ms	14.9 ms
2	0.1	21.8 ms	20.6 ms
4	0.1	27.4 ms	25.0 ms
5	0.1	32.7 ms	31.0 ms
100	0.1	359.8 ms	341.2 ms

We see that timers increase with the file rank to compensate the decreasing popularity of the video: for example, for the least popular video, we will set up the timer to approx. 40 times larger than the most popular video timer, what intuitively does not seem very clever.



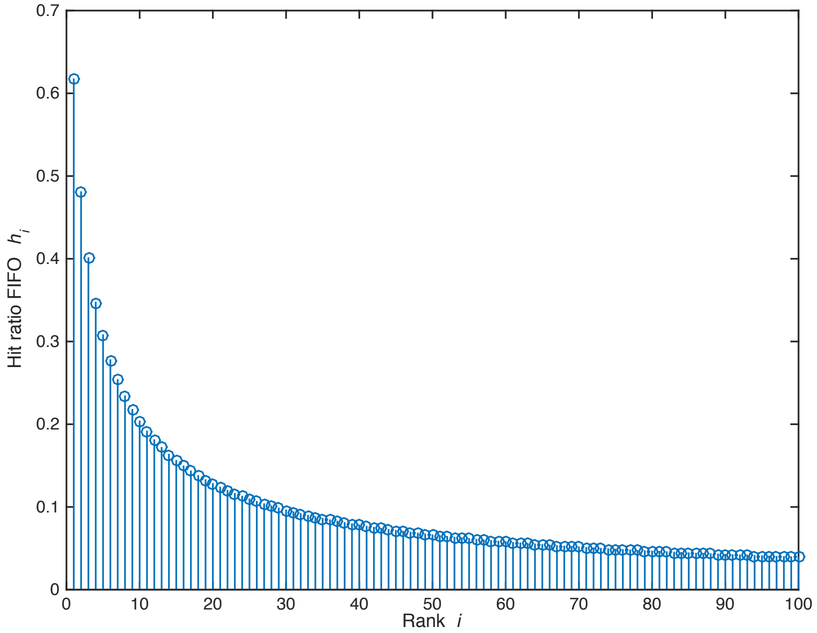


2. Now the CDN opts for using the most widely used replacement-based caching FIFO and LRU policies. These policies can be implemented as a *non-reset TTL* cache with *characteristic time* T_{FIFO} = 131 *ms* and a reset *TTL* cache with T_{LRU} = 117.5 *ms*, respectively. Compute for each content *i* and for both policies*,* what is the *hitting probability or ratio* (equivalent to the average proportion along time of the file *i* that is actually cached) h_i ? Are all the *TTL timer* t_i equal? Are all the *hitting ratio* h_i equal? Why?

Ans: The hit ratio for the FIFO policy is computed as:

$$h_i = 1 - 1/(1 + \lambda_i T)$$

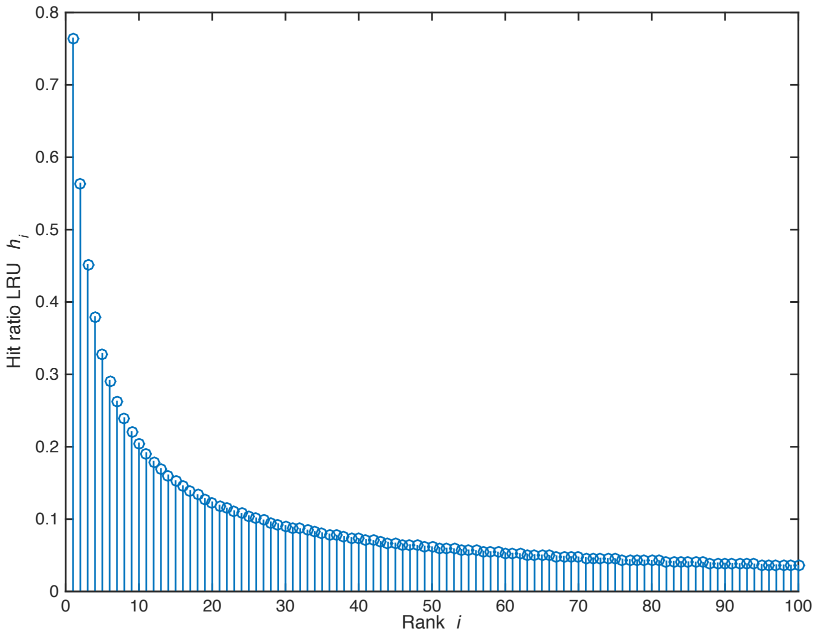
If we plot the results, we obtain the next figure, where the hit ratio h_1 of the most popular content (i=1) is equal to 0.6169 and the hit ratio h_{100} of the least popular content (i=100) is equal to 0.0389.



The hit ratio for the LRU policy is computed as:

$$h_i = 1 - e^{-\lambda_i T}$$

If we plot the results, we obtain the next figure, where the hit ratio h_1 of the most popular content (i=1) is equal to 0.7641 and the hit ratio h_{100} of the least popular content (i=100) is equal to 0.0356.



Rank i	**FIFO Hit Ratio (h_i)**	**LRU Hit Ratio (h_i)**	**FIFO TTL timer (t_i)**	**LRU TTL timer (t_i)**
1	0.6169	0.7641	131 ms	117.5 ms
2	0.4805	0.5637	131 ms	117.5 ms
2	0.4007	0.4510	131 ms	117.5 ms
4	0.3469	0.3790	131 ms	117.5 ms
5	0.3076	0.3287	131 ms	117.5 ms
100	0.0388	0.0356	131 ms	117.5 ms

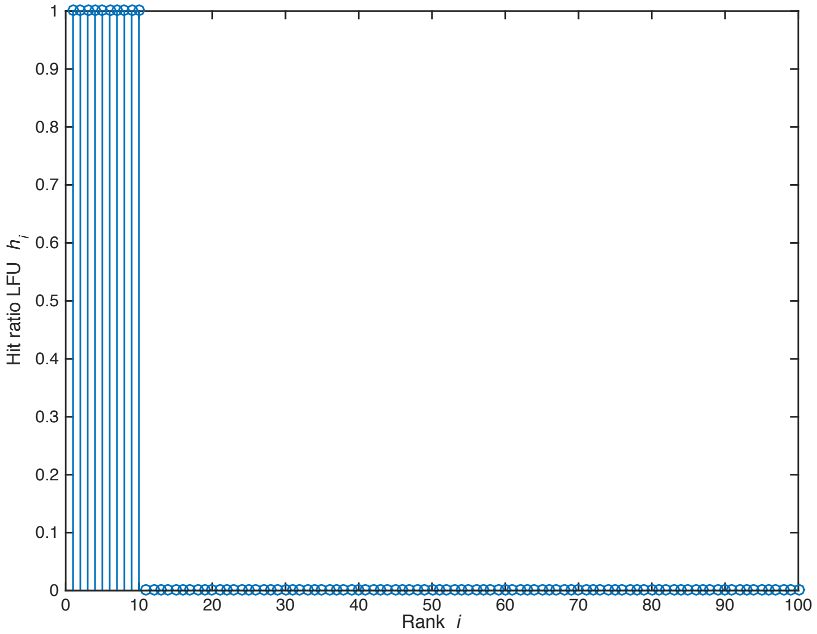
3. The CDN now decides to consider a linear utility $U_i (h_i) = \lambda_i \cdot h_i$, which is equivalent to use a LFU policy. Compute, for each content *i* in the CDN catalog *N,* which value should be used as *TTL timer* t_i and what is the *hitting ratio* h_i ?

   Ans: For the *hit probabilities* h_i, we obtain (not from derivative of L, intuitively):

   $h_i = 1$, $i = 1$ ….10 *$h_i = 0$, $i = 11$ ….100*

And for the *timers* t_i (regardless *non-reset or reset TTL*)

$t_i = \infty$, i =1 ….10 *$t_i = 0$, i =11 ….100*



Rank i	**LFU Hit Ratio (*h_i*)**	**LFU TTL timer (*t_i*)**
1	1	∞ ms
2	1	∞ ms
2	1	∞ ms
4	1	∞ ms
5	1	∞ ms
100	0	0 ms

4. The CDN now decides to consider a logarithmic utility $U_i(h_i) = \lambda_i \cdot log(h_i)$, which is equivalent to use a *proportionally fair policy*. Compute, for each content *i* in the CDN catalog *N,* which value should be used as *TTL timer* t_i and what is the *hitting ratio* h_i?

   Ans: For the *hit probabilities* h_i, we obtain

   $h_i = (\lambda_i / \sum_j \lambda_j) \cdot B$

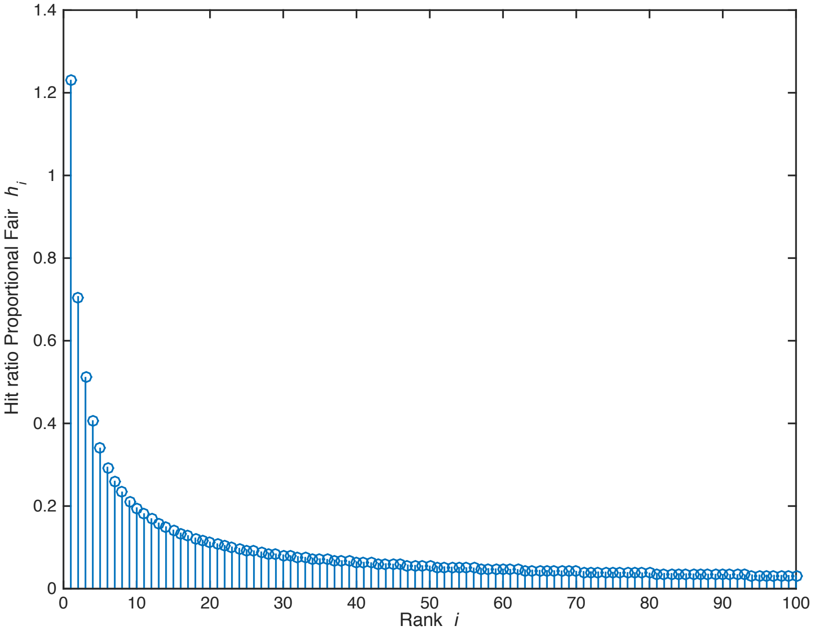
   And the *timers* t_i are computed as usual from $h_i = U_i'^{-1}(\alpha)$

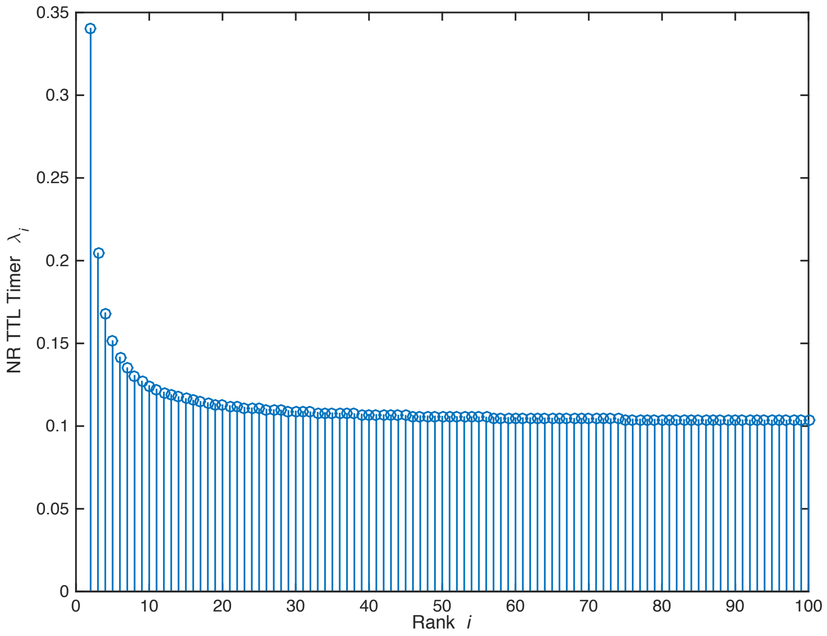
   a. *for non-reset TTL cache:*

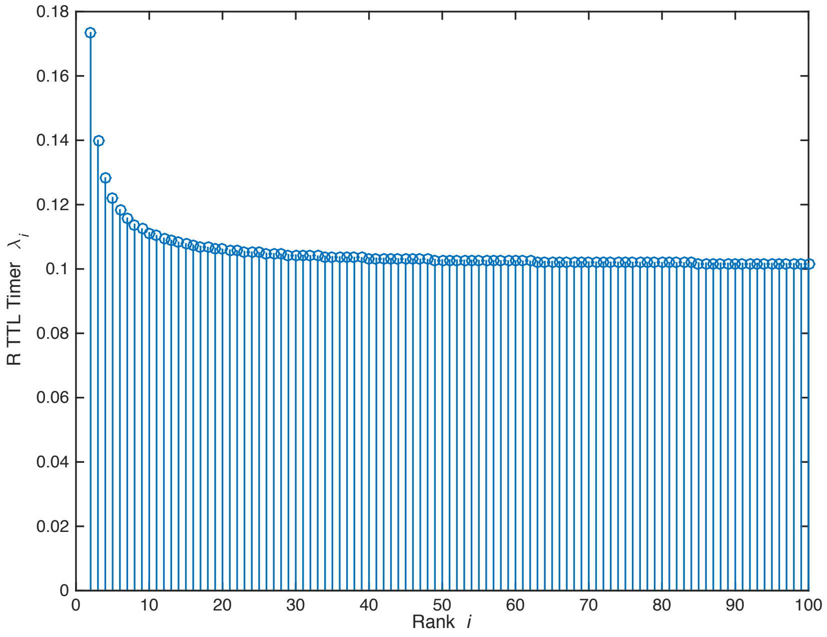
   $$t_i = -\frac{1}{\lambda_i}\left(1 - \frac{1}{1 - U_i'^{-1}(\alpha)}\right)$$

   b. *for reset TTL cache:*

$$t_i = -\frac{1}{\lambda_i} \log\left(1 - U_i'^{-1}(\alpha)\right)$$







Rank i	**Hit Ratio (h_i)**	**Non-reset TTL timer (t_i)**	**Reset TTL timer (t_i)**
1	*1.2293 > 1*!	∞ ms	∞ ms
2	0.7061	340.2 ms	173.4 ms
2	0.5105	204.3 ms	139.9 ms

4	0.4055	168.2 ms	128.2 ms
5	0.3392	151.3 ms	122.1 ms
100	0.0309	103.2 ms	101.6 ms

5. The CDN now decides to consider a negative inverse utility $U_i(h_i) = -\lambda_i/h_i$, which is equivalent to use a *minimum potential delay fairness*. Compute, for each content *i* in the CDN catalog *N,* which value should be used as *TTL timer* t_i and what is the *hitting ratio* h_i?

Ans: For the *hit probabilities* h_i, we obtain

$h_i = (\sqrt{\lambda_i} / \sum_j \sqrt{\lambda_j}) \cdot B$

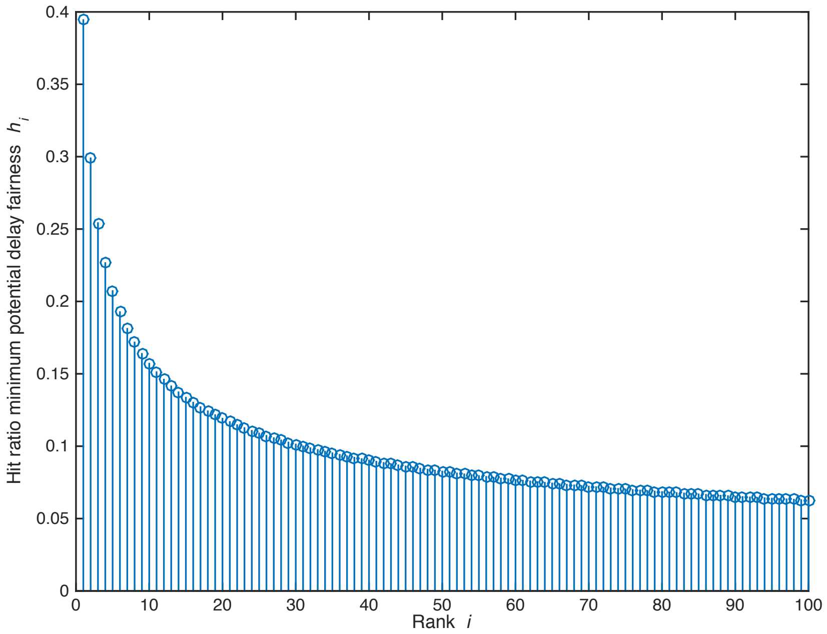
And the *timers* t_i are computed as usual from $h_i = U_i'^{-1}(\alpha)$

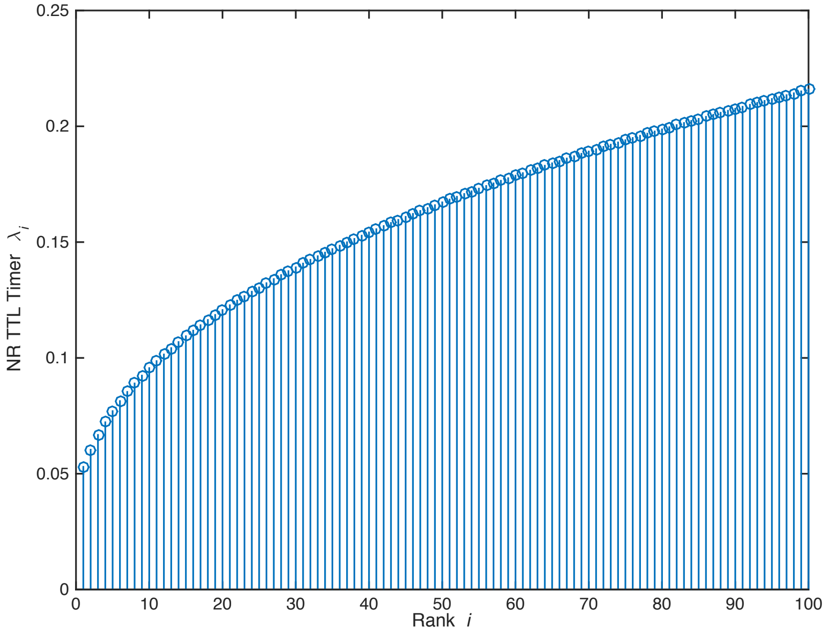
a. *for non-reset TTL cache:*

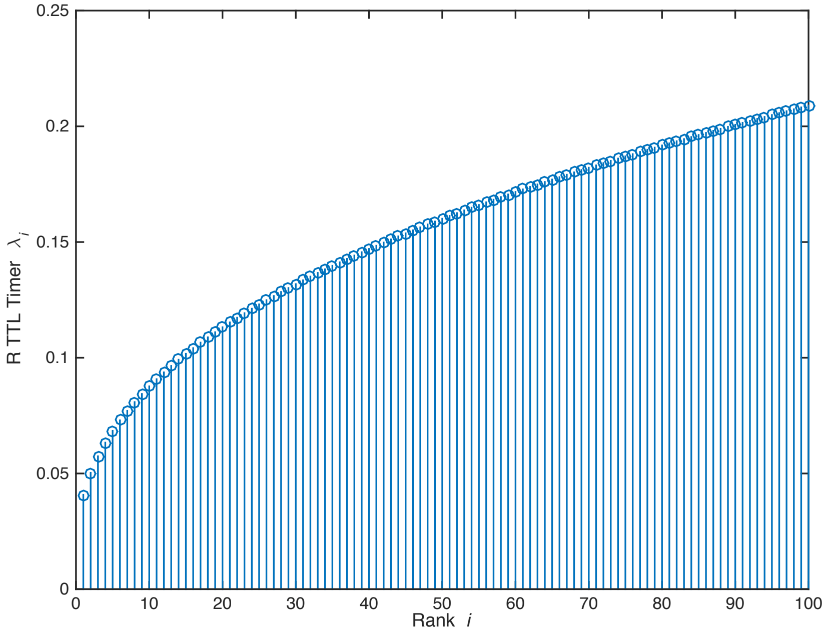
$$t_i = -\frac{1}{\lambda_i}\Big(1 - \frac{1}{1 - U_i'^{-1}(\alpha)}\Big)$$

b. *for reset TTL cache:*

$$t_i = -\frac{1}{\lambda_i}\log\Big(1 - U_i'^{-1}(\alpha)\Big)$$







Rank i	**Hit Ratio (h_i)**	**Non-reset TTL timer (t_i)**	**Reset TTL timer (t_i)**
1	0.3943	52.0 ms	40.8 ms
2	0.2988	60.4 ms	50.3 ms
2	0.2541	66.7 ms	57.4 ms

4	0.2265	72.2 ms	63.3 ms
5	0.2072	77.1 ms	68.4 ms
100	0.0625	215.9 ms	209.0 ms

6. The CDN now decides to consider as global utility the smallest hit ratio, which is equivalent to use a *max min fairness*. Compute, for each content *i* in the CDN catalog *N,* which value should be used as *TTL timer* t_i and what is the *hitting ratio* h_i?

   Ans: The results are the same as in the identical utility functions case. Note that, again we neglect the popularities.

Exercise 2: Bloom filters

1. Express the false positive probability ***p*** of a Bloom filter with the number of elements ***n***, the number of bits ***m***, and the number of hash functions ***k***.

   Ans:

$$p=\left(1-\left[1-\frac{1}{m}\right]^{kn}\right)^{k}\sim\left(1-e^{-kn/m}\right)^{k}$$

   Demonstration in next pages
   (see also Wikipedia entry: https://en.wikipedia.org/wiki/Bloom_filter)

2. Find the expression of ***k*** which minimizes the false positive probability, all other parameters given, and express the corresponding value of ***p***.

   Ans:

$$\frac{\partial p(k)}{\partial k}=0\rightarrow k^{*}=\frac{m}{n}\ln 2$$

$$p(k^{*})=\left(1-e^{-k^{*}n/m}\right)^{k^{*}}=\left(1-e^{-\frac{m}{n}\ln 2\frac{n}{m}}\right)^{\frac{m}{n}\ln 2}\rightarrow \ln p=-\frac{m}{n}(\ln 2)^{2}\rightarrow \boldsymbol{p=2^{-k^{*}}}$$

   Demonstration in next pages

3. Dimensioning the Bloom filter: Consider a typical example where a single server is likely to see *n* = 40 million objects and we are willing to tolerate a false positive probability of 0.1%. What are the minimum value of *m* and the corresponding value of *k* to satisfy these constraints?

   Ans:

$$m=-\frac{n\ln p}{(\ln 2)^{2}}\text{= 5.7510e+08 = }\textbf{575.10 million}$$

$$k^{*}=\frac{m}{n}\ln 2\text{ = 9.9658 → }\textbf{10 approx}.$$

PFE | Joint ISP ^{net} Content Gap for Video

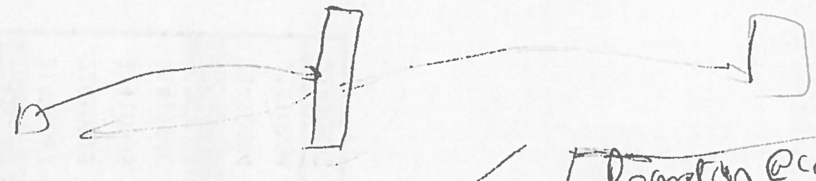
J I N O V E

Jeudi matin → { doodle → Demain MultiEdge

- check all acronymes AQM, MP, HAS, VoD, CC,

- imprimer le cours Ubinet

PFE Ubinet



formation @CNAM

- French Course

-

- LTE → possibilités

- SDN : basics

- SDN : in LTE

TD Ubinet :

- Relire Eurecom + q° base CDN

- Exos Bloom

- Manip

$\forall e, i, \quad h_i(e) \rightsquigarrow U_{[1 \dots m]}$

$\forall l$, $\Pr\{T[l] = 0\} = ?$ after inserting the n elements of S ?

or $\Pr\{h_i(e) \neq l\} = 1 - \frac{1}{m}$

$\rightarrow \Pr\{T[l] = 0\} = \left(1 - \frac{1}{m}\right)^{kn}$ after n elements and k probes

$= 1 - \Pr\{T[l] = 1\}$

False positive probability for an elemente : $\Pr\{\exists e' : T[h_i(e')] = 1, \forall i = 1 \dots k\} \underset{\uparrow}{=} \left(\Pr\{T[h_i(e')] = 1\}\right)^k$

assuming independent proba that each $T[l]$ takes value 1

that is $p = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \simeq \left(1 - e^{-\frac{kn}{m}}\right)^k$

For given n and m, best k : $\frac{dp}{dk} = \dots = 0 \Rightarrow k \simeq \frac{m}{n} \ln(2)$

$\Rightarrow \boxed{p \simeq 2^{-k} \simeq 2^{-\frac{m}{n}\ln(2)}}$ key equation to size the Bloom filter

To understand its space efficiency, it is instructive to compare the general Bloom filter with its special case when $k = 1$. If $k = 1$, then in order to keep the false positive rate sufficiently low, a small fraction of bits should be set, which means the array must be very large and contain long runs of zeros. The information content of the array relative to its size is low. The generalized Bloom filter (k greater than 1) allows many more bits to be set while still maintaining a low false positive rate; if the parameters (k and m) are chosen well, about half of the bits will be set,[5] and these will be apparently random, minimizing redundancy and maximizing information content.

Probability of false positives

Assume that a hash function selects each array position with equal probability. If m is the number of bits in the array, the probability that a certain bit is not set to 1 by a certain hash function during the insertion of an element is

$$1 - \frac{1}{m}.$$

If k is the number of hash functions, the probability that the bit is not set to 1 by any of the hash functions is

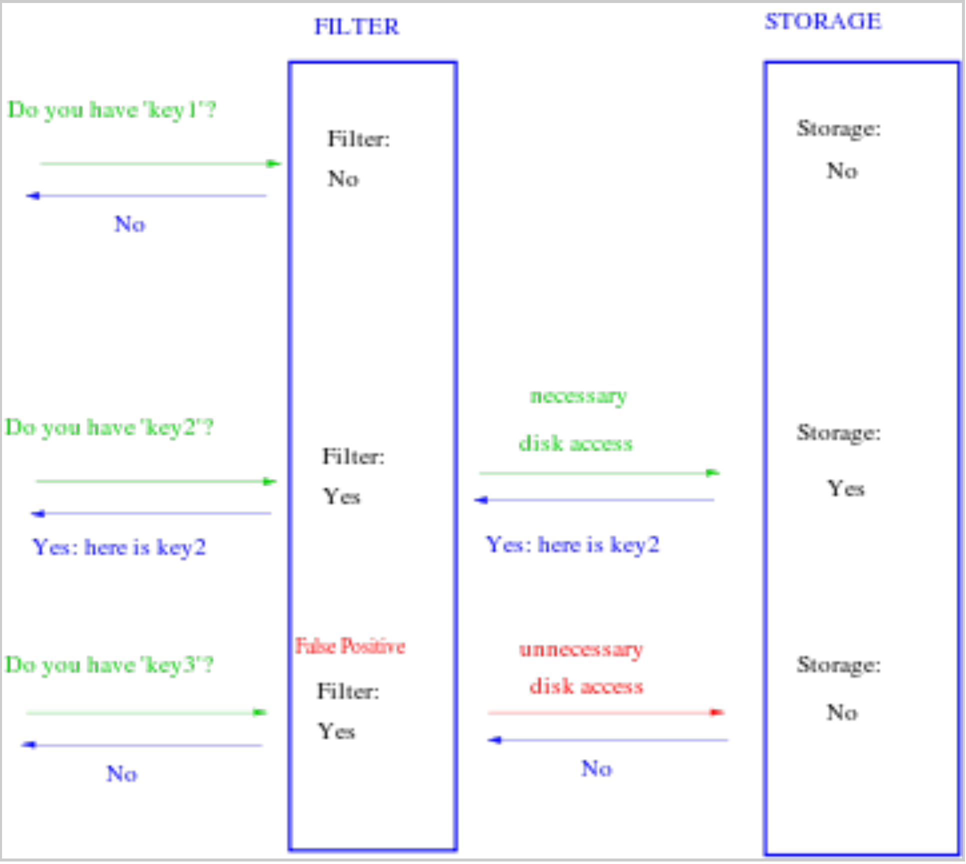
$$\left(1 - \frac{1}{m}\right)^k.$$

If we have inserted n elements, the probability that a certain bit is still 0 is

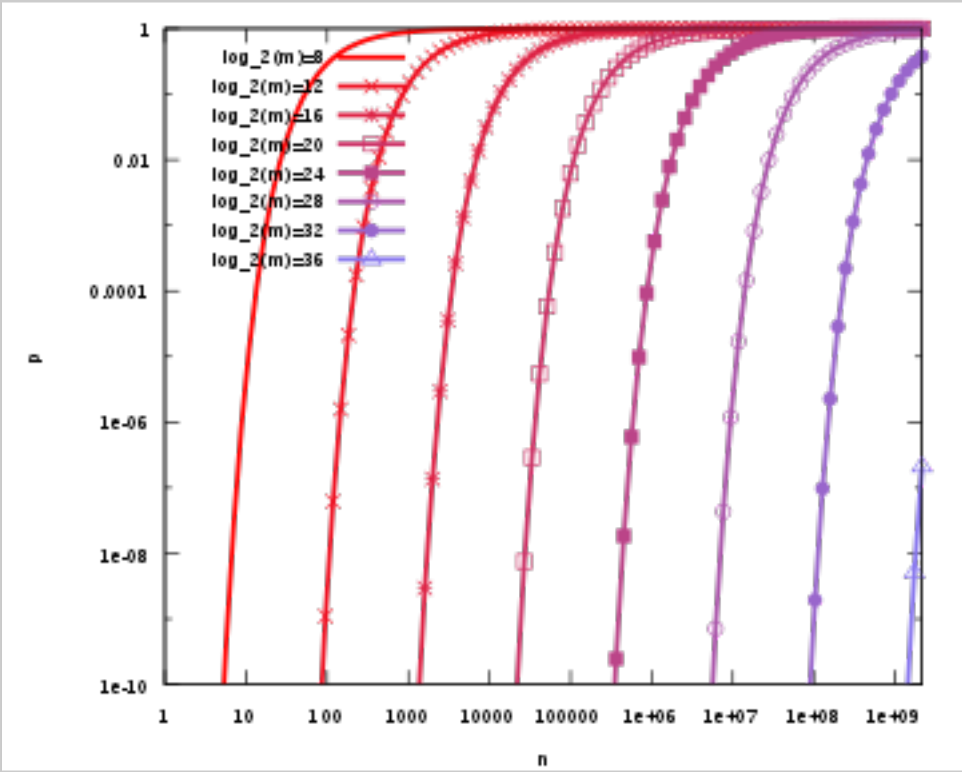
$$\left(1 - \frac{1}{m}\right)^{kn};$$

the probability that it is 1 is therefore

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



Bloom filter used to speed up answers in a key-value storage system. Values are stored on a disk which has slow access times. Bloom filter decisions are much faster. However some unnecessary disk accesses are made when the filter reports a positive (in order to weed out the false positives). Overall answer speed is better with the Bloom filter than without the Bloom filter. Use of a Bloom filter for this purpose, however, does increase memory usage.



The false positive probability p as a function of number of elements n in the filter and the filter size m. An optimal number of hash functions $k = (m/n) \ln 2$ has been assumed.

Now test membership of an element that is not in the set. Each of the k array positions computed by the hash functions is 1 with a probability as above. The probability of all of them being 1, which would cause the algorithm to erroneously claim that the element is in the set, is often given as

$$\left(1-\left[1-\frac{1}{m}\right]^{kn}\right)^k \approx \left(1-e^{-kn/m}\right)^k.$$

This is not strictly correct as it assumes independence for the probabilities of each bit being set. However, assuming it is a close approximation we have that the probability of false positives decreases as *m* (the number of bits in the array) increases, and increases as *n* (the number of inserted elements) increases.

An alternative analysis arriving at the same approximation without the assumption of independence is given by Mitzenmacher and Upfal.[6] After all *n* items have been added to the Bloom filter, let *q* be the fraction of the *m* bits that are set to 0. (That is, the number of bits still set to 0 is *qm*.) Then, when testing membership of an element not in the set, for the array position given by any of the *k* hash functions, the probability that the bit is found set to 1 is $1-q$. So the probability that all *k* hash functions find their bit set to 1 is $(1-q)^k$. Further, the expected value of *q* is the probability that a given array position is left untouched by each of the *k* hash functions for each of the *n* items, which is (as above)

$$E[q] = \left(1-\frac{1}{m}\right)^{kn}.$$

It is possible to prove, without the independence assumption, that *q* is very strongly concentrated around its expected value. In particular, from the Azuma–Hoeffding inequality, they prove that[7]

$$\Pr(|q-E[q]| \ge \frac{\lambda}{m}) \le 2\exp(-2\lambda^2/m)$$

Because of this, we can say that the exact probability of false positives is

$$\sum_t \Pr(q=t)(1-t)^k \approx (1-E[q])^k = \left(1-\left[1-\frac{1}{m}\right]^{kn}\right)^k \approx \left(1-e^{-kn/m}\right)^k$$

as before.

Optimal number of hash functions

For a given *m* and *n*, the value of *k* (the number of hash functions) that minimizes the false positive probability is

$$k = \frac{m}{n}\ln 2,$$

which gives

$$2^{-k} \approx 0.6185^{m/n}.$$

The required number of bits *m*, given *n* (the number of inserted elements) and a desired false positive probability *p* (and assuming the optimal value of *k* is used) can be computed by substituting the optimal value of *k* in the probability expression above:

$$p = \left(1-e^{-(\frac{m}{n}\ln 2)\frac{n}{m}}\right)^{\frac{m}{n}\ln 2}$$

which can be simplified to:

$$\ln p = -\frac{m}{n}\left(\ln 2\right)^2.$$

This results in:

$$m = -\frac{n \ln p}{(\ln 2)^2}.$$

This means that for a given false positive probability *p*, the length of a Bloom filter *m* is proportionate to the number of elements being filtered *n*.[8] While the above formula is asymptotic (i.e. applicable as $m,n \to \infty$), the agreement with finite values of *m*,*n* is also quite good; the false positive probability for a finite Bloom filter with *m* bits, *n* elements, and *k* hash functions is at most

$$(1 - e^{-k(n+0.5)/(m-1)})^k.$$

So we can use the asymptotic formula if we pay a penalty for at most half an extra element and at most one fewer bit.[9]

Approximating the number of items in a Bloom filter

Swamidass & Baldi (2007) showed that the number of items in a Bloom filter can be approximated with the following formula,

$$n^* = -\frac{m}{k}\ln\left[1 - \frac{X}{m}\right],$$

where n^* is an estimate of the number of items in the filter, *m* is the length (size) of the filter, *k* is the number of hash functions, and *X* is the number of bits set to one.

The union and intersection of sets

Bloom filters are a way of compactly representing a set of items. It is common to try to compute the size of the intersection or union between two sets. Bloom filters can be used to approximate the size of the intersection and union of two sets. Swamidass & Baldi (2007) showed that for two Bloom filters of length m, their counts, respectively can be estimated as

$$n(A^*) = -\frac{m}{k}\ln\left[1 - \frac{n(A)}{m}\right]$$

and

$$n(B^*) = -\frac{m}{k}\ln\left[1 - \frac{n(B)}{m}\right].$$

The size of their union can be estimated as

$$n(A^* \cup B^*) = -\frac{m}{k}\ln\left[1 - \frac{n(A \cup B)}{m}\right],$$

False positive probability (cont.)

- Define

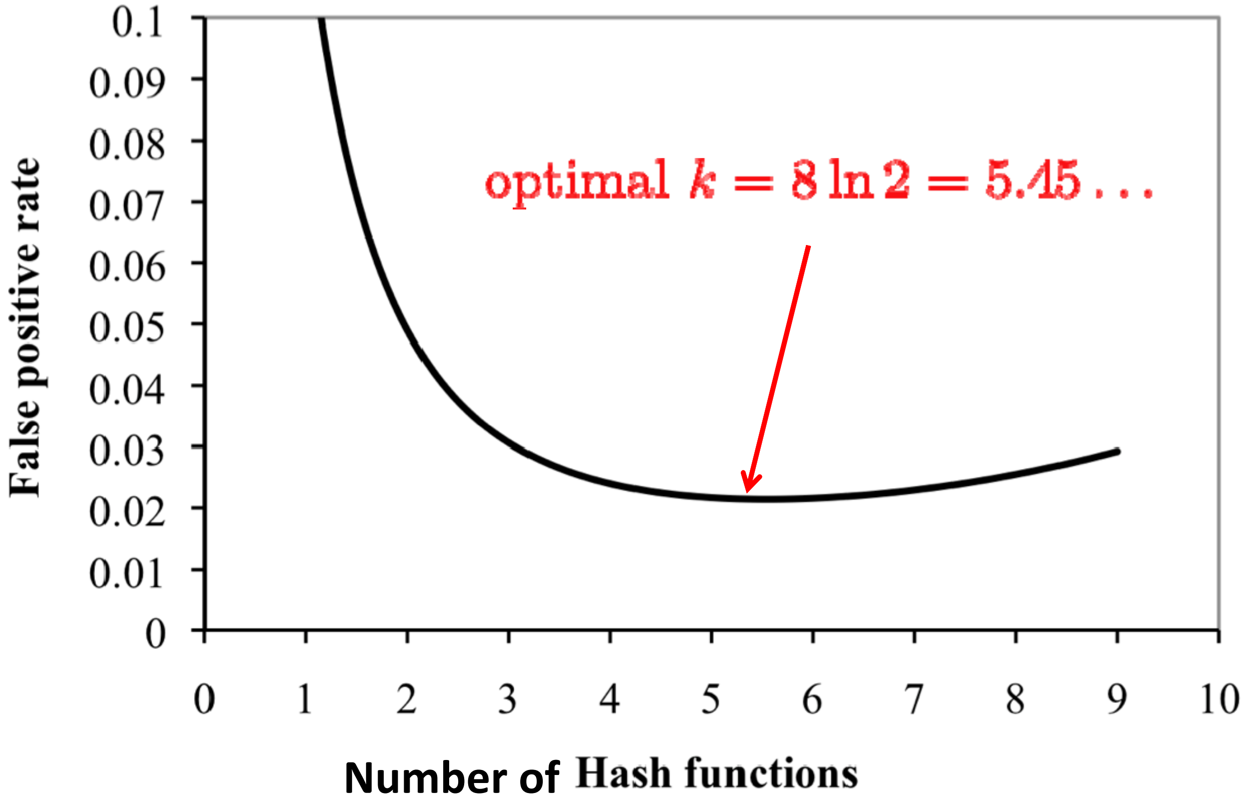
$$f' = (1 - p')^k = (1 - (1 - \frac{1}{m})^{kn})^k$$

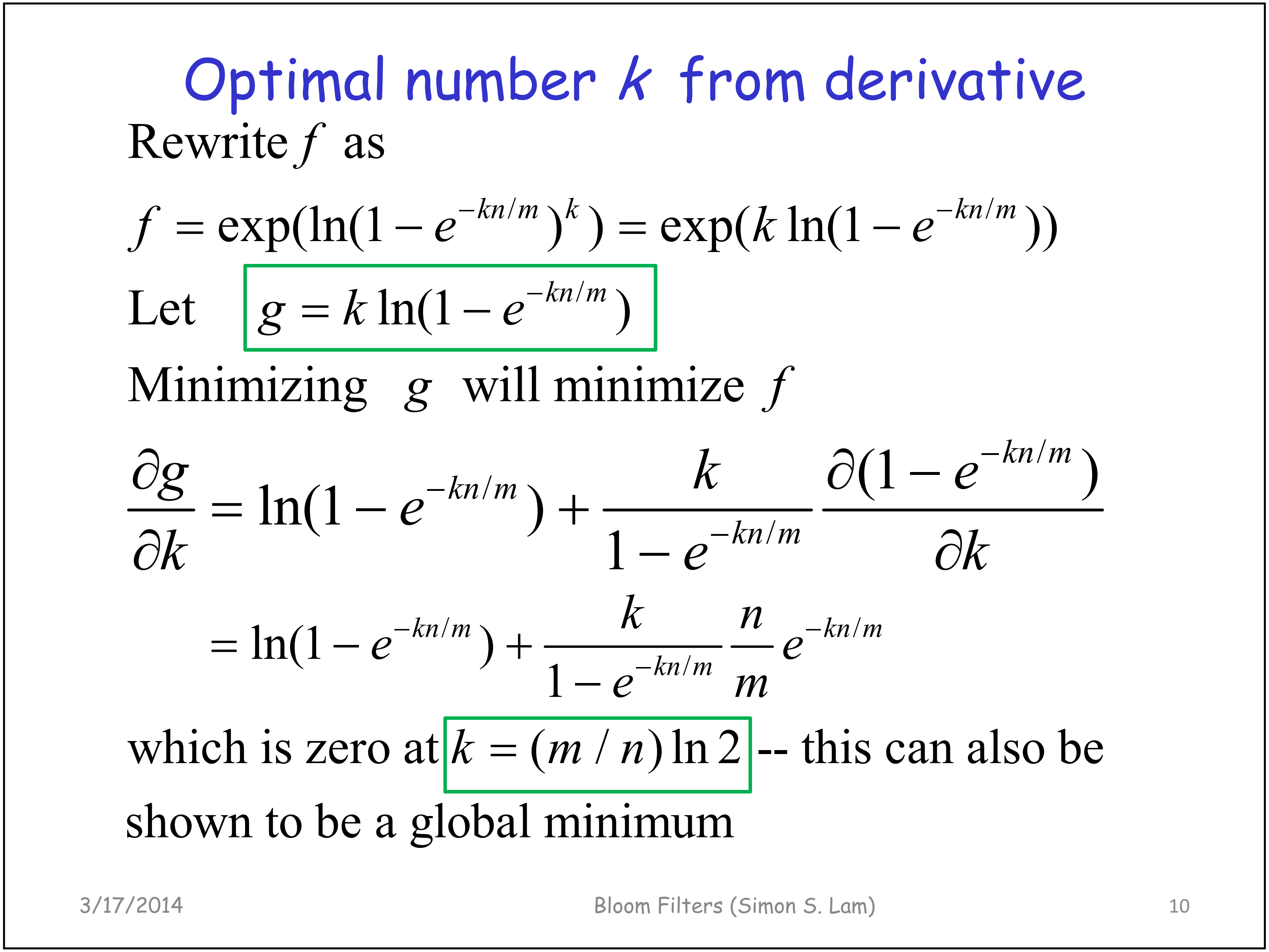
$$f = (1 - p)^k = (1 - e^{-kn/m})^k$$

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

Minimizing g will minimize f

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

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

which is zero at $\boxed{k = (m / n) \ln 2}$ -- this can also be shown to be a global minimum