THE YES-NO BLOOM FILTER: REPRESENTING SETS WITH FEWER FALSE POSITIVES

Alexei Vernitski

THE MODEL

THE STANDARD BLOOM FILTER

The Bloom filter is a compressed way of representing a set, with some false positives but no false negatives. Each element of a set is assumed to be (or is associated with) a Boolean array of a fixed length n and with a fixed number r of bits set to 1 (later we shall refer to this array as the element's ID). The set is represented by an array which is a disjunction of its elements. When the membership of a set is queried using a Bloom filter, an element is assumed to be in the set if it is less than or equal to the Bloom filter (where the comparison is performed bitwise).

The advantages of the Bloom filter are that the calculations needed to form a Bloom filter and to test membership are easy to program and very fast to perform. One obvious disadvantage of the Bloom filter is that its simplicity gives rise to a large number of false positives; this is why a number of improved models has been suggested.

THE YES-NO BLOOM FILTER: GENERAL IDEA

We propose a new way of representing a set. It is based on the Bloom filters, and it has some false positives but no false negatives. Our model has considerably fewer false positives than the Bloom filter, as will be demonstrated below on a number of experiments.

The main insight behind our model is that usually not only the elements belonging to the set, but, more generally, all the elements which will ever be queried on their membership in this set are known in advance, at the stage when a representation of the set is formed. (In our experiments, we shall consider one such scenario: packet forwarding in information-centric networks.)

In this context, let us introduce some notation. Let U be the universe of all elements which might or might not be the elements of some particular set; let $S \subseteq U$ be a set we are representing, and let $T \subseteq U \setminus S$ be all the elements which do not belong to set S and whose membership in S is likely to be tested. What we have said in the previous paragraph is that when we form a representation of S, normally we know not only S, but also T; this is useful because typically, T is much smaller than $U \setminus S$.

Thus, what we shall do is propose a model in which we build a representation of set S which not only describes the elements of S, but also has some additional flexibility which can be used to avoid actively the false positives which might result from querying the membership of the

elements of T (as opposed to other elements of $U \setminus S$, whose membership is never going to be queried, and, therefore, we are not afraid of getting false positives from them).

This is a novel idea which somehow was overlooked in previous research, despite its obvious usefulness.

THE YES-NO BLOOM FILTER: SPECIFIC MODEL

We propose the following specific model implementing the above idea. We have chosen it because, like with the standard Bloom filter, the calculations needed to form a representation of a set and to test membership are easy to program and very fast to perform. (Without going into detail, this is so because the representation of a set in our model consists of two or more Bloom filters, and the fast performance is inherited from them.)

Let us describe our model. As with the standard Bloom filter, we assume that the representation of a set consists, in total, of n bits. These n bits are split into the following parts:

- A Bloom filter of length p, to which we shall refer as the YES-FILTER;
- A certain number (namely, m) Bloom filters of length q, to which we shall refer as the NO-FILTERs.

We shall refer to the above construction as a yes-no Bloom filter.

To describe informally how a yes-no Bloom filter represents a set S, considered together with some elements $T \subseteq U \setminus S$ whose membership in S can be queried, we can say that the yes-filter is chosen so that it represents set S, whereas the no-filters are chosen so that the union of all the m sets represented by them is equal to set T.

Now let us define formally how the yes-no Bloom filter works. Each element $e \in U$ is assumed to be (or is associated with) a Bloom filter of length p and m Bloom filters of length q; thus, e has a structure replicating the structure of a yes-no Bloom filter; for convenience, we shall refer to these parts of e as the yes-filter and the no-filters of e (whereas all these filters together shall be referred to as the element's ID). When membership of an element e$\in U$ in a set S represented by a yes-no Bloom filter β is queried, e is assumed to be in S if

- e's yes-filter is less than or equal to β's yes-filter (where the comparison is performed bitwise; that is, each bit of e's yes-filter is less than or equal to the corresponding bit of β's yes-filter) and
- none of e's no-filters is less than or equal to the corresponding β's no-filter (that is, in each of e's no-filters there is a bit which is greater than the corresponding bit of the corresponding no-filter in β).

The yes-no Bloom filter is a generalisation of the standard Bloom filter: indeed, when $m = 0$, the yes-no Bloom filter turns into the Bloom filter of length n. When $m > 0$, the length of the yes-filter is reduced (compared with the case $m = 0$), perhaps producing an increased number of false positives; however, at the same time, the no-filters reduce the number of false positives.

THE YES-NO BLOOM FILTER: IMPLEMENTING FLEXIBILITY

A number of choices need to be made to implement yes-no Bloom filters. As with the standard Bloom filters, one would expect that the parameters defining the structure of the yes-no Bloom filter (the values of n, m, p, q and the values of r separately for the yes-filter and for the no-

filters) would be decided upon in advance and not changed within any particular application. In the description of our experiments below, we suggest values of these parameters which seem to work well.

In addition to the static choice of these basic parameters, at the stage when the yes-no Bloom filter is built for a particular set S, considered together with elements $T \subseteq U \setminus S$ whose membership in S can be queried, one can build the yes-no Bloom filter in a number of ways; namely, one would aim to choose the no-filters in such a way that between them, they minimise the number of false positives. In principle, this looks like a complicated discrete optimisation problem; it is not unreasonable to conjecture that finding an exact solution to it (that is, the smallest possible number of false positives) is NP-hard. However, for our model to workable, we cannot afford to use time-expensive algorithms; this is why we suggest using a simple and fast greedy algorithm, and we show in the experiments below that it performs well.

Namely, here is how we are building yes-no Bloom filter β representing set S, considered together with elements $T \subseteq U \setminus S$ whose membership in S can be queried.

- Form the yes-filter of β a disjunction of the yes-filters of all elements $e \in S$.
- Assume that the elements of T are ordered in some linear order; proceeding through this list one element at a time, perform the following steps for each $f \in T$ individually.
- Proceeding through the list of no-filters, perform the following steps for each no-filter ν individually.
- Attempt to add the element f to the filter ν. Namely, form a disjunction ν' of ν and the corresponding no-filter of f; then check the following condition: that for each element $e \in S$ the corresponding filter of e is not less than or equal to ν'.
- If the condition is satisfied, replace the value of ν by ν' and proceed to the next element of T.
- If the condition is not satisfied and ν is not the last no-filter, proceed to the next no-filter.
- If the condition is not satisfied and ν is the last no-filter, proceed to the next element of T; we have not included f in any no-filter, and it will be a false positive.

EXPERIMENTS

The purpose of this section is to present examples showing by how much accuracy is improved is yes-no Bloom filters are used instead of standard Bloom filters.

The parameters used are as follows. The total length of a yes-no Bloom filter used is 256, which is the same as the length of the Bloom filter used for comparison. In yes-no Bloom filters, the number of no-filters is 2, the length of each no-filter is 32 bits, the number of set bits in no-filters is 3, the number of set bits in the yes-filter is 4. The traditional Bloom filter model is used with the number of set bits equal to 6, which is the optimal value of this parameter.

PACKET FORWARDING

In an information-centric network, to forward a packet from one node to another, one forms a set consisting of all the links of the path which the packet should take. At each node along the path, the set is inspected, and the packet is forwarded along the next link on the path, but not any other links adjacent to the node. Thus, the set of all links is U, the set of links along the path is S, and the set of all other links adjacent to the nodes in the path is T.

Let us stress a couple of technical aspects. We assume that each node operates in a very simple way: it always forwards a packet along all the links adjacent to it which, when queried, are assumed to belong to S. Note that links are directed (that is, a link from a node $n1$ to a node $n2$ is not the same as a link from a node $n2$ to a node $n1$); this arrangement is convenient for a number of reasons; in particular, it prevents backward movement of a packet along its path.

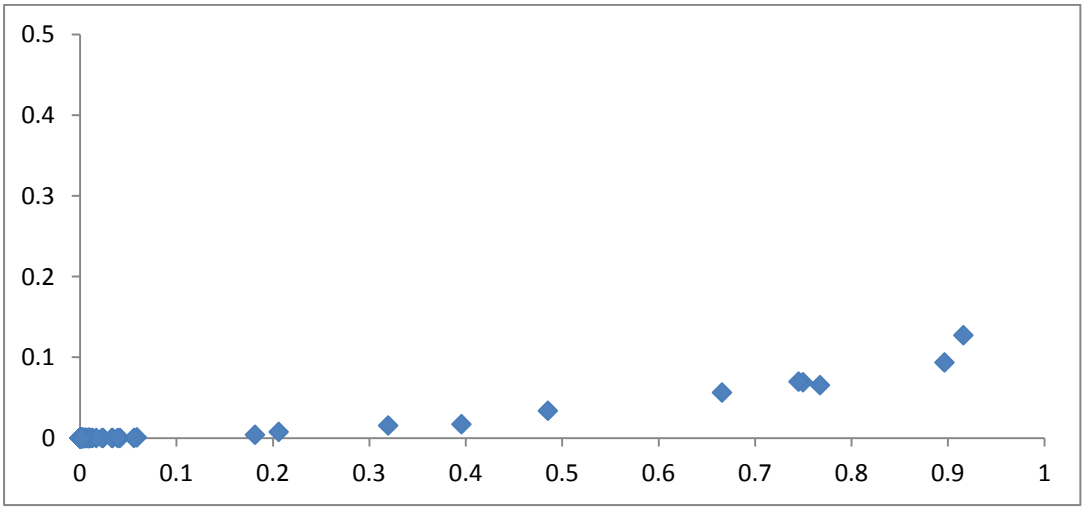
If set S is represented with some false positives, some nodes might forward the packet not only along the links in S, but also along some links in T. The additional traffic resulting from false positives wastes network resources and should be avoided. At each node, the packet should be processed in the fastest and easiest way possible; this is why Bloom filters are used. We suggest using yes-no Bloom filters instead of Bloom filters.

REAL-LIFE NETWORKS

Although we could presented experimented with artificially generated networks, we think it is better if we show how the performance of Bloom filters and yes-no Bloom filters compares on real-life example. This is why we used the networks from the Network Topology Zoo. This collection of networks consists of 261 real-life examples. Although they are not information-centric networks, we consider them as such for the sake of experiment (this is justified because we are interested not in the protocols used in a network, but in typical topologies of networks). In each of these networks, we choose a long realistic forwarding path and consider this path as set S, whereas all the other links adjacent to the path are considered as set T. For each of these pairs S, T we consider a number of random allocations of IDs for all the elements of S and T and calculate the number of false positives.

RESULTS

Our results are presented on the following diagram:



The horizontal axis is the expected number of false positives if we use the Bloom filter. The vertical axis is the expected number of false positives if we use the yes-no Bloom filter.

Let us comment on one example in detail. The top-right point corresponds to a 28-hop path in the network from the TataNld example[1]. If the Bloom filter is used for forwarding along this path (or a similar path), one can expect, approximately, 0.93 false positives (as calculated in our

[1] See it on the diagram http://www.topology-zoo.org/maps/TataNld.jpg

experiments); but if the yes-no Bloom filter is used, one can expect, approximately, 0.13 false positives. The meaning of this is that if we use Bloom filters, each packet will almost certainly be diverted to a wrong direction approximately at one point during its travel; however, if we use yes-no Bloom filters, this will only happen to approximately one out of eight packets.

Note that this diagram presents the important section of our results, but there are also some outliers, which are of a minor significance, but should be mentioned. On the left-hand side, there are many very small networks for which neither the Bloom filter nor, of course, the yes-no Bloom filter produce any false positives. They are not interesting for our comparison. On the diagram, they all are represented by the point with the coordinates 0,0.

On the right-hand side (beyond the boundaries of the diagram), there are two comparatively large networks, in which both the Bloom filters and the yes-no Bloom filters produce a relatively large number of false positives, and, therefore, perhaps one would want to consider using some more involved forwarding model (although, we are pleased to say, the yes-no Bloom filter still performs noticeably better than the Bloom filter). For completeness, here these two examples are presented in a table.

Example name	Path length	Bloom filter false positives	Yes-no Bloom filter false positives
UsCarrier	35	1.95	0.48
Kdl	58	15.87	11.27

GENERALISATION

In this paper, we have built a yes-no Bloom filter as a combination of Bloom filters. Instead of the Bloom filters, it is possible to use another way of representing sets as building blocks of such a combination. This might make sense in some applications.

CONCLUSION

We have proposed a novel way of representing sets based on Bloom filters, which we call the yes-no Bloom filter. It is as easy to compute and fast as the classic Bloom filter, but has a considerably smaller number of false positives. We have shown on computational experiments how the yes-no Bloom filters outperform the Bloom filters in the scenario of packet forwarding in information-centric networks.