Computer Networks 54 (2010) 3373–3387

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Improving retouched Bloom filter for trading off selected false positives against false negatives

Benoit Donnet [a,*], Bruno Baynat [b,**], Timur Friedman [b,***]

[a] Université catholique de Louvain, Louvain-la-Neuve, Belgium
[b] UPMC Sorbonne Universités and CNRS, Paris, France

ARTICLE INFO

Article history:
Received 8 September 2009
Received in revised form 4 June 2010
Accepted 8 July 2010
Available online 15 July 2010
Responsible Editor: C. Westphal

Keywords:
Bloom filters
False positives
False negatives
Bit clearing
Measurement
Traceroute

ABSTRACT

Where distributed agents must share voluminous set membership information, Bloom filters provide a compact, though lossy, way for them to do so. Numerous recent networking papers have examined the trade-offs between the bandwidth consumed by the transmission of Bloom filters, and the error rate, which takes the form of false positives. This paper is about the retouched Bloom filter (RBF). An RBF is an extension that makes the Bloom filter more flexible by permitting the removal of false positives, at the expense of introducing false negatives, and that allows a controlled trade-off between the two. We analytically show that creating RBFs through a random process decreases the false positive rate in the same proportion as the false negative rate that is generated. We further provide some simple heuristics that decrease the false positive rate more than the corresponding increase in the false negative rate, when creating RBFs. These heuristics are more effective than the ones we have presented in prior work. We further demonstrate the advantages of an RBF over a Bloom filter in a distributed network topology measurement application. We finally discuss several networking applications that could benefit from RBFs instead of standard Bloom filters.

1. Introduction

Introduced in 1970 [1], it is just in the past decade that the *Bloom filter* has attracted attention from the networking research community [2]. A Bloom filter compactly encodes set information into a bit vector that can then be queried regarding set membership. A vector of all zeroes represents the empty set. To record a key as being in the set, hash it to obtain an index into the vector and set the bit at that position to one. You may use multiple hash functions, in which case you set several bits to one. To query if a key is in the set, check if all hash positions are set to one. Though the filter will occasionally return a false positive, erroneously claiming that a key belongs to the set, it will never return a false negative, erroneously claiming that a key does not belong. You may set the vector size and number of hash functions in light of the anticipated set size, to aim for a particular trade-off between the size of the bit vector and the false positive rate.

A prime appeal of the Bloom filter to networking researchers comes from the bandwidth efficiencies that it offers for the transmission of set membership information between networked hosts [3]. We ourselves have proposed their use for large-scale route tracing infrastructures [4]. Continuously running production systems of this sort include Archipelago [5], DIMES [6], RIPE TTM [7], iPlane [8], Gulliver [9], Ono [10], and TDMI [11]. They have in common the placement of agents across the Internet so as to obtain measurements from a variety of vantage points. If these agents are to coordinate their efforts, as Archipelago agents will do for their *scamper* measurements [12], then they must communicate between each other, either directly or

* Corresponding author. Tel.: +32 10 478718; fax: +32 10 450345.
** Corresponding author.
*** Corresponding author.
E-mail addresses: benoit.donnet@uclouvain.be (B. Donnet), bruno.baynat@lip6.fr (B. Baynat), timur.friedman@lip6.fr (T. Friedman).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.07.003

via a central server. Such communication is potentially bandwidth-hungry, which is a problem if the agents are expected to adhere to tight bandwidth constraints, as they are for Ono [13, Section 5.3].

This paper describes a way for distributed route tracing agents to coordinate to reduce their impact on end-hosts. For each trace that an agent makes, it records the IP address that it encounters one hop before the end host. We call this the *penultimate node*. The agent encodes the set of penultimate nodes in a Bloom filter, which it sends to other agents. Another agent consults the filter while tracing: if it encounters an address already seen by the first agent, it stops tracing that route and goes onto another one.

Using Bloom filters in this way allows a trade-off between filter size and coordination failures. A smaller filter saves on the bandwidth required for coordination. A larger filter reduces on average the rate of false positives, which stop route tracing before the penultimate node is reached and thereby deprive the system of potentially useful information. Without the Bloom filter, there would be no such flexibility. Replacing the filter with an explicit list of penultimate nodes would limit the infrastructure operator to one extreme end of this trade-off: high coordination bandwidth in exchange for no traces that stop too early. This, despite the fact that some low level of coordination failures might perhaps be tolerable.

Researchers have proposed the Bloom filter for so many networked applications precisely to allow them to enjoy this sort of flexible trade-off. However we can do even better, as this trade-off is more simplistic and limiting than it needs to be. Simplistic, because the false positive rate expresses an average based on an idealized query distribution in which all keys are equiprobable, whereas actual system performance depends on that distribution and the identities of those keys that cause the false positives. Limiting, because the system might tolerate false negatives, but the Bloom filter does not allow us to introduce false negatives into the trade-off.

This paper describes the *retouched Bloom filter* (RBF), a modification to the standard Bloom filter that allows us to remove selected false positives at the cost of introducing random false negatives. We create an RBF from a Bloom filter, as Section 3 describes, by selectively changing individual bits from 1 to 0, while the size of the filter and the query mechanism remain unchanged. As Section 3.1 shows analytically, if we create an RBF through a purely random process then we decrease the false positive rate, on average, in the same proportion as the false negative rate that we generate. Simple heuristic algorithms that we present in Section 3.1 do better than the random process and lower the false positive rate by a greater degree, on average, than the corresponding increase in the false negative rate. In addition, Section 4 provides mechanisms that allow us to selectively remove the most troublesome false positives, further improving performance when we take the query distribution into account.

The RBF algorithms require space that is at most a small constant multiple of the Bloom filter's vector size. Compared to the creation of a standard Bloom filter, the RBF algorithms also incur additional processing costs related to key removal. These costs are a constant multiple of a number of RBF parameters, such as the number of hash functions and the number of false positives to remove. The additional processing and storage requirements that are incurred when switching from Bloom filters to RBFs are restricted entirely to the locations at which the RBFs are created. There is strictly no addition to the critical resource in our networked scenario, which is the bandwidth consumed by communication between measurement points. At the receiver of the RBF, queries take place using exactly the same mechanism as for the Bloom filter, incurring no additional space or time complexity.

Compared to our previous work [4], which introduced the RBF, the algorithms in this paper are more effective. By more carefully tracking the quantities of false negatives generated and false positives removed at each step of an algorithm, we achieve a greater decrease, on average, in the false positive rate for a given increase in the false negative rate. Based on simulations, we demonstrate that our new improved algorithms perform between 10% and 20% better on average than the simple algorithms we previously proposed. The case study in Section 5 has been rerun using these improved algorithms.

The work that we present here is the first that subjects false negatives in a Bloom filter variant to either analytic or simulation studies. In particular, our work is the first to explicitly study the trade-off between false positives and false negatives and it is the first to consider the efficiency of the means employed for such a trade-off. Section 6 describes the extensive related work on Bloom filters [3,14–20]. Some of these extensions, such as the *anti-Bloom filter* [14] and the *generalized Bloom filter* [15], target the suppression of false positives, or the removal of bits in the vector in general. Some of them (such as the *variable Bloom filter* [19]) even provide a trade-off between the false positive rate and the false negative rate. Nevertheless, as we show, these variants differ significantly from the standard Bloom filter in that they either increase the memory cost (i.e., increase the size of the filter) or they modify the filter behavior when performing membership queries.

The remainder of this paper is organized as follows: Section 2 presents the standard Bloom filter, using notation introduced by Broder and Mitzenmacher [2]; Section 3 presents the RBF, and shows analytically that the reduction in the false positive rate is equal, on average, to the increase in the false negative rate even as random 1s in a Bloom filter are reset to 0s; Section 4 presents improved methods for selectively clearing 1s that are associated with the most troublesome false positives, and shows through simulations that they reduce the false positive rate by more, on average, than they increase the false negative rate; Section 5 describes the use of RBFs in a network measurement application; Section 6 discusses several Bloom filter variants, compares RBFs to them and discusses other networking usages of RBFs; finally, Section 7 summarizes the conclusions and future directions for this work.

2. Bloom filters

A *Bloom filter* [1] is a vector v of m bits that codes the membership of a subset $A = \{a_1, a_2, \ldots, a_n\}$ of n elements of