A Recent Survey on Bloom Filters in Network Intrusion Detection Systems

K.Saravanan [#1], Dr.A.Senthil kumar [*2], J.S.Dolian [#3]

[#1&3] Department of Electronics & Communication,
Karunya University, Cbe, India.

[#2]Department of Electronics & Instrumentation,
Kongu Engineering College, Perundurai,, India.

Abstract: Computer networks are prone to hacking, viruses and other malware; a Network Intrusion Detection System (NIDS) is needed to protect the end-user machines from threats. An effective NIDS is therefore a network security system capable of protecting the end user machines well before a threat or intruder affects. NIDS requires a space efficient data base for detection of threats in high speed conditions. A bloom filter is a space efficient randomized data structure for representing a set in order to support membership queries. These Bloom filters allow false positive results (FPR) but the space saving capability often outweigh this drawback provided the probability of FPR is controlled. Research is being done to reduce FPR by modifying the structure of bloom filters and enabling it to operate in the increasing network speeds, thus variant bloom filters are being introduced. The aim of this paper is to survey the ways in which Bloom filters have been used and modified to be used in high speed Network Intrusion Detection Systems with their merits and demerits.

Keywords—**Bloom filter, false positive ratio (FPR), hash function, threats, Network Intrusion Detection System (NIDS).**

I. INTRODUCTION

To identify the predefined signatures in network streams software based detection techniques are commonly used. However, the software based techniques cannot keep up with the speeds that network bandwidth increases. Hence, hardware based systems started to emerge. Hence it is mandatory to scan network packets bit by bit to determine predefined signatures for viruses and worms. Finite Automata (FA) in field programmable gate arrays (FPGAs) is used to reach very high speeds. Hardware reconfiguration is required for adding and deleting strings, which is too expensive. The simplest approach is a ternary-content addressable memory (TCAM), which stores all the strings. Searches can be very fast, but the high cost makes TCAM infeasible for large signature sets [1].

For pattern matching Bloom filters are used. They are hash-based structures which have a certain degree of accuracy for considerable savings in memory. To represent a set of elements and to perform membership queries Bloom filters are created, and they are adopted for pattern matching simply by constructing filters according to a set of signatures. The advantages are the compact representation that is typical of Bloom filters and a remarkable reduction of the amount of traffic handled by the slow path, which result in a general performance improvement and scalability of IDSs.

In this paper we give a recent survey on Different types of Bloom filters used for Network Intrusion Detection system to benefit the research community to analyse and develop an efficient bloom filter which can have a prominent role in network security, each having its own merits and demerits. The details of standard Bloom filter, Counting Bloom filter, Pipelined Bloom filter and two-tier Bloom filter is explained below.

II. STANDARD BLOOM FILTER

The Bloom filter is used to test whether an element is a member of a set and is a space-efficient probabilistic data structure, false negatives are not possible but false positives are possible. Elements can only be added to the set and cannot be removed. The probability of false positives increases when more elements are added to the set.

An empty Bloom filter is a bit array of *m* bits, all set to 0. There must also be *k* different hash functions defined, each of which maps or hashes some set element to one of the *m* array positions with a uniform random distribution. To add an element, feed it to each of the *k* hash functions to get *k* array positions. Set the bits at all these positions to 1.

The requirement of designing *k* different independent hash functions can be prohibitive for large *k*. For a good hash function with a wide output, there should be little if any correlation between different bit-fields of such a hash, so this type of hash can be used to generate multiple "different" hash functions by slicing its output into multiple bit fields. Alternatively, one can pass *k* different initial values (such as 0,1, ,,, k-1) to a hash function that takes an initial value, or add these values to the key. For larger *m* and/or *k*, independence among the hash functions can be relaxed with negligible increase in false positive rate. Removing an element from this simple Bloom filter is impossible. The element maps to *k* bits, and although setting any one of these *k* bits to

zero suffices to remove it, this has the side effect of removing any other elements that map on-to that bit, and we have no way of determining whether any such elements have been added [4]. Such removal would introduce a possibility for false negatives, which are not allowed.

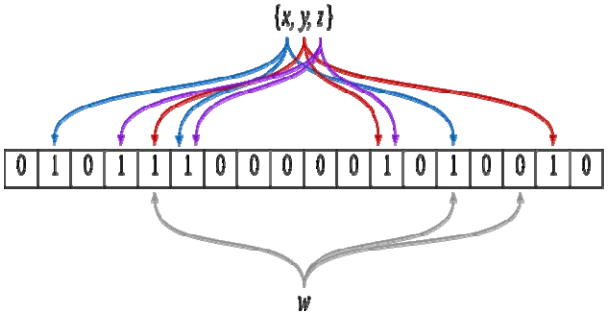

Fig.1 Standard Bloom Filter

An example of Bloom filter is shown in Fig 1. This is representing the set $\{X, Y, Z\}$. The coloured arrows show the positions in the bit array that each set element is mapped to. For this figure m=18 and k=3. Removal of an element from a Bloom filter can be simulated by having a second Bloom filter that contains items that have been removed. However, false positives in the second filter become false negatives in the composite filter, which are not permitted.

This approach also limits the semantics of removal since re-adding a previously removed item is not possible. However it is often the case that all the keys are available but are expansive to enumerate. When the false positive rate gets too high, the filter can be regenerated. This should be a relatively rare event.

A. *Space and time advantages*

Bloom filters have a strong space advantage over other data structures for representing sets, such as self-balancing binary search trees [15], tries, hash tables or simple arrays or linked list of the entries. Most of these require storing at least the data items themselves, which can require anywhere from a small number of bit, for small integers, to an arbitrary number of bits, such as for strings. Linked structures incur an additional linear space overhead for pointers. A bloom filter with 1% error and an optimal value of k, on the other hand, requires only about 9.6 bits per element, regardless of the size of the elements. This advantage comes partly from its compactness, inherited from arrays, and partly from its probabilistic nature. If the number of potential values is small and many of them can be in the set, then the bloom filter is easily surpassed by the deterministic bit array, which requires only one bit for each potential element. Note also that hash tables gain a space and time advantage if they begin ignoring collisions and only store whether each bucket contains an entry, in this case, they have effectively become bloom filters with k=1.

Bloom filters also have the unusual time needed to either add items or to check whether an item is in the set is fixed constant, completely independent of the number of items already in the set. No other constant-space data structure has this property, but the average access time of sparse hash tables can make them faster in practice than some Bloom filters. In a hardware implementation, the bloom filter shines because its k lookups are independent and can be paralleled.

B. *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 one by a certain hash function during the insertion of an element is then

$$1 - \frac{1}{m} \qquad (1)$$

The probability that is not set by any of the hash function is [15]

$$(1 - \frac{1}{m})^k \qquad (2)$$

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

$$(1 - \frac{1}{m})^{kn} \qquad (3)$$

The probability that it is 1 is therefore

$$1 - (1 - \frac{1}{m})^{kn} \quad (4)$$

The membership of an element that is not in the set can be tested now. 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 given as

$$\left(1 - \left[1 - \frac{1}{m}\right]\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \quad (5)$$

This is not strictly correct as it assumes independence for the probabilities of each bit being set. Assuming it is a close approximation, then 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. For a given *m* and *n*, the value of *k* (the number of hash functions) that minimizes the probability [2] is

$$\frac{m}{n} \ln 2 \approx \frac{9m}{n} \approx 0.7 \frac{m}{n} \quad (6)$$

which gives the false positive probability of

$$2^{-k} \approx 0.68185^{m/n} \quad (7)$$

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) is

$$m = -\frac{n \ln p}{(\ln 2)^2} \quad (8)$$

This means that in order to maintain a fixed false positive probability, the length of a Bloom filter must grow linearly with the number of elements being filtered. While the above formula is asymptotic (i.e. applicable as *m*, $n \to \infty$), the agreement with *m* bits, *n* elements and *k* hash functions is at most

$$\left(1 - e^{-k(n+0.5)/(m-1)}\right)^k \quad (9)$$

So the asymptotic formula can be used if a penalty is paid for at most half an extra element and at most one fewer bit.

III. COUNTING BLOOM FILTER

One property of Bloom filter is that it is not possible to delete a member stored into the filter. Deleting a particular entry requires that the corresponding *k* hashed bits in the bit vector be set to zero. This could disturb other members programmed into the filter which hash to any of these bits. In order to solve this problem, the idea of the *Counting Bloom Filters* was proposed in [1]. A Counting Bloom filter maintains a vector to counters corresponding to each bit in the bit-vector. Whenever a member is added to or deleted from the filter, the counters corresponding to the *k* hash values are incremented or decremented respectively. When a counter changes from 0 to 1, the corresponding bit in the bit-vector is cleared. It is important to note that the counters are changed only during addition and deletion of strings in a Bloom filter. For applications like network intrusion detection, these updates are relatively less frequent than the actual query process itself. Counters can be maintained in software and the bit corresponding to each counter is maintained in hardware. Thus, avoiding counter implementation in hardware, memory resources can be saved.

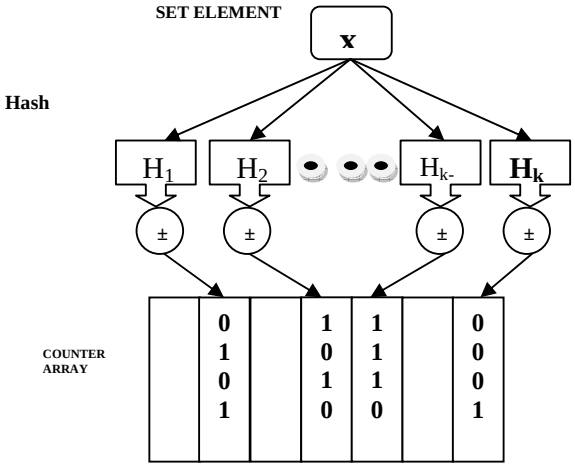

Fig 2 Counting Bloom Filter

In the figure 2 the operation of counting bloom filter is depicted as for each hash function separate counter is maintained increment/decrement operation.

IV. PIPELINED BLOOM FILTER

A pipelined Bloom filter [4] , as shown in Fig 2 consists of two groups of hash functions. The first stage always computes the hash values. By contrast, the second stage of hash functions only compute the hash values if in the first stage there is a match between the input and signature sought. The pipelined Bloom filter will have the same number of hash functions as per the same number of hash functions as the regular Bloom filters. A pipelined Bloom filter exploits the virus free nature of the network traffic at most of the time.

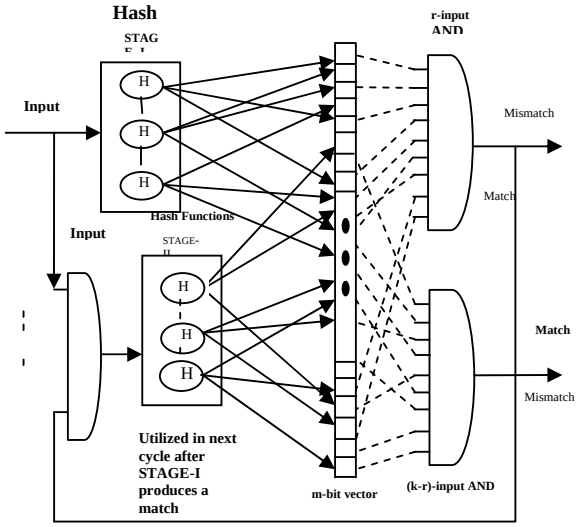

Fig 3 Pipelined Bloom filter.

At worst, it will operate like a regular Bloom filter, which uses all of the hash functions before making decision on the type of the input. Most of the time the first group of hash functions will result in a mismatch. The advantage of using a pipelined Bloom filter is if the first stage catches mismatch, there is no need to use the second stage in order to decide whether input string is a member of the signature set.

V. TWO-TIER BLOOM FILTER

A two-tier Bloom filter design is proposed to reduce membership test time. Fig.,3 [14] shows the basic design of a single component containing hashing circuitry and a Bloom filter. The two tiers might exist in a hardware implementation of a Bloom filter, or between processor cache and main memory in a computer. An

on-chip Bloom filter can be implemented using fast SRAM, but it is limited in size to about *m* ¼ Mb. For applications with more than 131072 elements, the on-chip Bloom filter can be used as a cache for a large off-chip Bloom filter stored in the main memory of a system. The two-tier Bloom filter takes as input the element to be mapped into, or tested for membership, and outputs the *k* hash values and a single test output to indicate if the element being tested for was found in the on-chip cache Bloom filter. If the cache hit output false, then the computed hash values are used to test the external Bloom in the main memory. If the element is found in the external Bloom filter in the main hit causes the cache Bloom filter to learn the element. Thus the cache Bloom filter contains a subset of the elements mapped into the main memory Bloom filter.

Membership testing time (T_{test}) is a function of hashing time (t_{hash}), cache Bloom filter testing time (t_{cache}), main memory Bloom filter testing time (t_{main}) and probability of a successful membership test in the cache Bloom filter (P_{cache}). In order for the two-tier Bloom filter to have a smaller membership testing time than that of a single Bloom filter in main memory, $t_{cache} - P_{cache}\, t_{main} < 0$ must hold. The speed-up (S) of T_{test} is the ratio of the time required by the two-tier Bloom filter stored in the main memory. The speed- up expresses the relative reduction in membership testing time by using the two-tier Bloom filter.
Speed-up [4] is

$$S = \frac{Thash+Tmain}{Thash+Tcache+(1-Pcache)Tmain} \qquad (10)$$

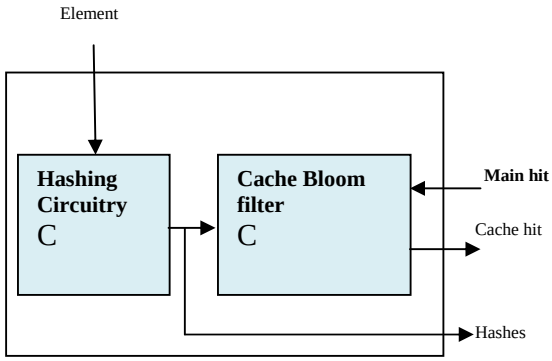

Fig. 4 Design of Two-tier Bloom filter

The specific target application for the two-tier Bloom filter is membership testing for a file system containing millions of files, each file with a unique identifier.

VI. Dynamic Bloom Filter

Dynamic Bloom filter [16] has been proposed as a method to implement Bloom filter in a scalable environment such as the size of a dataset is not known in advance. Bloom filter's major drawback is that the elements cannot be deleted from them once they are added and there is no certainty of the eventual dataset size. This is addressed in the Dynamic Bloom filters.

An alteration to Bloom filter is proposed with Dynamic Bloom filter by dynamically creating new filters on the fly as they are needed. DBF seems to be a logical addition to BF for a scalable environment just before the false positive rate of particular BF starts growing fast, switch to a new filter and store the old one. DBF handles inserts and lookups. DBF has serious scaling issues which seem to render it almost unusable. To rectify this, instead of creating a new filter of fixed size when needed, create a new filter that is twice the size of the last filter. The analysis [9] shows the false positive rate of Bloom filter and Dynamic Bloom filter.

Among these Bloom filters which we discussed earlier Dynamic Bloom filter has more merits, such variable length of data processing. Depends on the requirements the Dynamic Bloom filter can be adjusted for the desired performance. So it is appropriate to compare the performance of Standard Bloom filter and Dynamic Bloom filter based on the false positive probability. The graph in Fig.4 shows that the Dynamic Bloom filter has low false positive probability than Bloom filter.

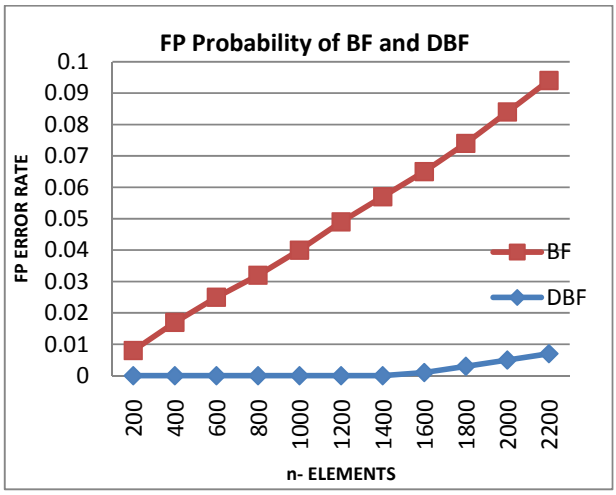

Fig. 5 FP probability of BF and DBF

VII. CONCLUSIONS

A Bloom filter is a space-efficient representation of a set or a list that handles membership queries, hence it can be used for the variety of Network applications mainly in detecting intrusions in NIDS. Each has the advantages and disadvantages, but it depends on the requirements. Wherever a list is used, and space is at premium, consider using a Bloom filter if the effect of false positives can be mitigated.

REFERENCES

[1] Gianni Antichi, Domenico Ficara, Stefano Giordano, Gregario Procissi and Fabio, "Counting Bloom Filters for Pattern Matching and Anti-Evasion at the wire Speed" IEEE Network, January/February 2009.

[2] A.Broder and M.Mitzenmacher, "Network Applications of Bloom Filter: A survey", Internet mathematics, vol.1, no.4, pp.485-509, July 2003.

[3] Domenico Ficara, Stefano Giordano, Gregario Procissi, Fabio Vitucci, "Multilayer Compressed Counting Bloom Filters" Proc., INFOCOM '08, Apr.2008.

[4] Yu Hua, Bin Xiao, "A Multi-attribute Data Structure with Parallel Bloom Filters For Network Services"

[5] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, John Lockwood, "Deep Packet Inspection Using Parallel Bloom Filters" IEEE Micro,vol.24, no.1, 2004, pp.52-61.

[6] Sarang Dharmapurikar, Praveen Krishnamurthy, David E.Taylor, "Longest Prefix Matching Using Bloom Filters", IEEE/ACM Transactions on Networking, vol.14, Issue.2, 2006, pp. 397-409.

[7] Ilhan Kaya, Taskin Kocak, "Energy-Efficient Pipelined Bloom Filters For Network Intrusion Detection" IEEE Conference on Communications ICC '06, vol.5, pp.2382-2387.

[8] Mahmood Ahmadi, Stephen Wong, "Modified Collision Packet Classification Using Counting Bloom Filter In Tuple Space" Proc., PDCN '07.

[9] Deke Guo, Jie Wu, Honghui Chen, Xueshan Luo "Theory And Network Application Of Dynamic Bloom Filters" INFOCOM '06, April 2006, pp.1-12.

[10] M. Mitzenmacher, "Compressed Bloom Filters", IEEE/ACM transactions on networking, vol.10, no.5, pp.604-612, October, 2002.

[11] H. Song et al., "Fast Hash Table Lookup Using Extended Bloom Filter: An aid to Network Processing", Proc.2005 Conf.Apps., Tech., Architecture Protocols Comp.Commun., New York, NY, 2005, pp. 181-192.

[12] M. Nourani and P.Katta, "Bloom Filter Accelerator for String Matching", Proc. 16th Int'l.Conf.Comp.Commun.Net., pp. 185-190.

[13] N.S. Artan and H.J Chao, "Multi-Packet Signature Detection Using Prefix Bloom Filters", Proc.IEEE GLOBECOM, vol.3, 2005, pp. 1811-16.R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, "High-speed digital-to-RF converter," U.S. Patent 5 668 842, Sept. 16, 1997.

[14] M. Jimeno, K.J. Christensen and A. Roginsky, "Two-tier Bloom Filter To Achieve Faster Membership Testing", Electronics Letters, vol.44, No.7, 27th March 2008.

[15] Benoit Donnet, Bruno Baynat, Timur Friedman, " Improving retouched Bloom Filter For Trading off Selected False Positives Against False Negatives", Journals, Computer networks, vol.54, 2010, pp.3373-3387.