A Survey On Architectural Design Of Bloom Filter For Signature Detection

Manjula.G[1], Brindha.P[2]
[1]*PG Student,* [2]*Assistant Professor*

Abstract

Computer networks face number of problems like threats from hackers, viruses and other malwares, as the speed of the network increases rapidly. The usual software based signature Detection System provides lesser support to the host system as the network speed increases. Thus hardware implementations of such signature detection system can have significantly superior throughput using Bloom filters are started budding. Bloom filter offers a mechanism to store and inspect a large number of intrusions efficiently when implemented with Field Programmable Gate Array (FPGA) technology. This paper provides a survey of bloom filter with its properties and different architectures for the design of an efficient Counting Bloom Filter for signature detection technique.

1. Introduction

A signature detection scheme refers to the detection of network intrusions that arrive to the host system through network cable. These intrusions can be monitored using a signature matching scheme with the help of a maintaining a database. A network intrusion detection system (NIDS) attempts to detect malicious activity such as denial of service attack, port scans or tries to crack into computers by monitoring network traffic. NIDS monitors all incoming packets and tries to find suspicious patterns known as signatures or malwares. These are decided by a network administrator at the time of configuration and deployment of the network intrusion detection system based on the security and network policies of the organization. Since the number of threats and network speed increases every day, the conventional software based NID system such as Snort utilizes more processing time; hence it is difficult to cope with both protection and higher data rates. As a result custom hardware implementation of network intrusion detection started emerging, which can have significantly higher throughput and lesser time. For implementing this hardware based string matching technique, Bloom filters are used. Bloom filter offers a mechanism to investigate for a large number of strings efficiently and concurrently when implemented with Field Programmable Gate Array (FPGA) technology.

The concept of Bloom Filter [1] was introduced by Burton H. Bloom in 1970. A Bloom Filter is a probabilistic data structure that is used in characterizing large data to a set and in performing membership queries i.e. to test whether an element is a member of the set or not, in a space efficient manner. The general applications of bloom filter include Spell Checkers, Longest Prefix Matching, and Refining Web Search Results. Bloom filter plays a vital role in the field of network security and database management applications.

The paper aims to survey the design of counting bloom filter in various ways for the application of hardware based network intrusion detection system. The rest of the paper is organized as follows. Section 2 describes the operation of Standard Bloom filter and its properties. Literature survey regarding the different architectural design of Counting Bloom filter are discussed in section 3.Power consumption analysis for various architectures are done in section 4. Finally, Conclusion and future remarks are mentioned in section 5.

2. Standard Bloom Filter

A Bloom filter consists of set of hash functions h1,h2,…hk used to calculate different hash values and bit array of m bits that are initially set to 0.Bloom filter operates in two modes namely programming and querying mode. In programming mode, each element xi produces different hash values {h1(xi),..hk(xi)}and are stored randomly in the bit vector. In querying mode, a similar approach is followed as during the storage of elements. The

same k hash functions are considered over the element named y.We check for the bits hi(y) for i=1,2,….k to be equal to 1. If all bits are 1, the element was probably in the set. If one or more of these bits are still 0, the element is certainly not in the set. The probability of enabling same bit vector for different elements are possible, hence removing one element from the set disturbs the other.

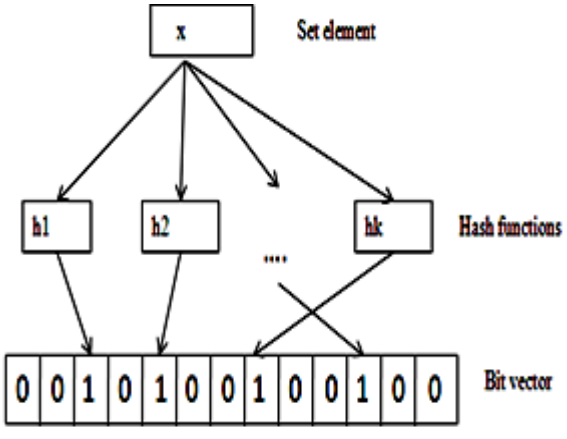

Fig 1: Programming the Bloom filter

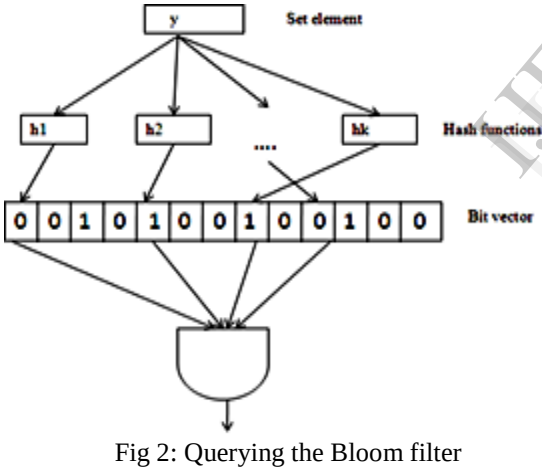

Fig 2: Querying the Bloom filter

2.1. Properties of Standard Bloom Filter

- Bloom filter is a space efficient data structure.
- The time required to query does not depend on the number of elements present in the set.
- The probability of false positives can be significantly lowered and a false negative does not exist.
- Bloom Filter allows in the approximation of intersection between two sets.
- If two Bloom Filters with same number of bits and same number of hash functions represent sets S1 and S2 respectively then a single Bloom Filter representing the union of these two sets can be obtained by taking OR of the two bit-vectors of the original Bloom filters.

2.2. False Positive Probability

The false positive probability rate of a Bloom filter and the optimal number of hash functions can be derived as follows. Let us assume that a hash function selects each array position with equal probability and m denotes the number of bits in the Bloom filter. While inserting an element into the filter, the probability that a certain bit is not enable to one by a hash function is 1-1/m.

Since there are k hash functions in the filter, the probability of any of them to be not set a specific bit to one is given by

$$(1-1/m)^k \qquad (1)$$

After the insertion of n elements into the filter, the probability that a given bit is still zero is given by

$$(1-1/m)^{kn} \qquad (2)$$

Accordingly the probability that the bit set to one is

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

For the membership test of an element, if all of the k array positions in the filter calculated by the hash functions are set to one, the Bloom filter declares that the element belongs to the set. The probability when the element is present in the set is given by

$$(1-(1-1/m)^{kn})^k \approx (1-e^{-kn/m})^k \qquad (4)$$

The probability of false positives can be reduced by minimizing $(1-e^{-kn/m})^k$ with respect to k. This is accomplished by taking the derivative and making equal to zero. The obtained result gives the false positive probability as

$$(1/2)^k \approx 0.6185^{m/n} \qquad (5)$$

An optimal number of hash functions is assumed to be k = (m/n) ln 2

3. Literature Review

This section performs the literature survey of Counting Bloom Filter and its implementation for the application of network intrusion detection. The

different architectural design for the regular CBF is performed.

3.1. Counting Bloom Filter

To overcome the drawback of Standard bloom filter, Counting bloom filter is introduced. A counting bloom filter [6] consists of m counters along with m bits for each elements present in the set. When an element is inserted, the corresponding counters are incremented; similarly when the element is deleted, the corresponding counters are decremented. The value of the counter gives the number of elements hashed to it. Since each counter size is limited, the n-bit counter will overflow if it reaches a value of 2^n. If the bit size for the counter of CBF is *c* and the total memory size is *m*, CBF can use only *m/c*-sized memory space to register *n* elements. Therefore, the false positive of CBF is defined as:

$$F_{pCBF} = (1-e^{-knc/m})^{k} \qquad (6)$$

The major advantage of Counting Bloom Filter is that deletion of data in the data structure is possible.i.e; it allows storage of elements and their removal without any collision. The memory utilization is much less compared to the standard bloom filter. At the same time, it has a limitation of having higher false positive rate.

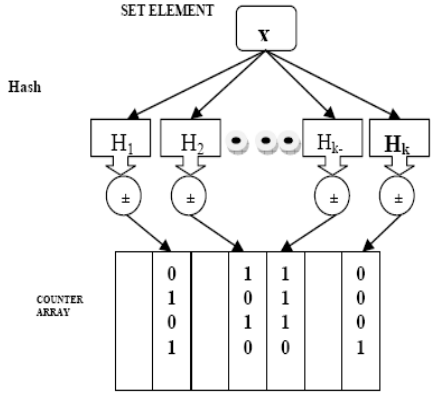

Fig 3: Counting Bloom filter

3.2. Counting Bloom Filter implementation for signature detection

Safi et.al [4] performs the comparison of two implementations namely Low power CBF with the SRAM based counting bloom filter where S-CBF uses an SRAM array of counts along with the up/down counter, zero comparator and a small controller. In this implementation a small controller coordinates the entire sequence of actions and the updates are performed as read-modify-write sequences which can be depicted as follows: 1) the count is read from the SRAM; 2) it is adjusted using the counter; and 3) it is written back to the SRAM. The probe operation is employed as a read from the SRAM, and the comparison with zero can be done using the zero comparator. The major drawbacks with S-CBF are that it suffers in both delay and energy as updates require two SRAM accesses per operation.

Safi et.al describes another Low power CBF that does not require the actual count values of the counter rather it cares only whether a count is "zero" or "nonzero". The Low power CBF performs three operations namely:1) increment count (INC); 2) decrement count (DEC); and 3) test if the count is zero (PROBE). The first two operations are performed during the updating phase of the set whereas PROBE is done at the time of query mode. This structure consists of an array of up/down LFSRs together with zero detectors. The use of up/down LFSRs offer a better delay and power than other synchronous up/down counters provided with the same length of count sequence.

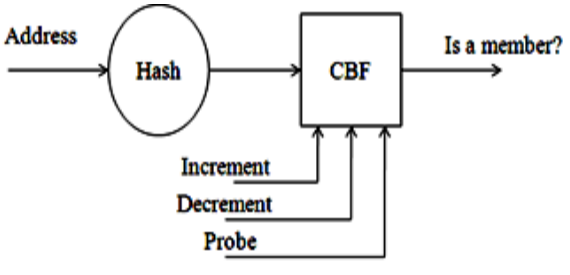

Fig 4: CBF for membership test.

Safi et.at explored that Low power CBF is superior when compared to S-CBF in terms of both delay and speed along with the drawback of more expense in terms of area.

3.2.1. K-stage Pipelined Bloom Filter Architecture

In the network intrusion detection application due to very low rate of malicious traffic there is no need to calculate all the hash functions to obtain the

result of non membership. To exploit this pipelined architecture is introduced to the existing structure. In the Pipelined architecture of bloom filter, an enable input is used. For first hash function it is set to one and for the remaining k-hash functions the output of the previous value is given. This architecture as shown in fig 5 consists of several groups of hash functions that are utilized in different stages. The first stage always computes the hash values. The second and further stages are used only if there is a match in the previous stage [7]. The main advantage of using a pipelined Bloom filter structure is if the first stage produces a mismatch there is no need to compute the second stage because a bloom filter never produces a false negatives. This saves the power consumed by pipelined bloom filter as compared to the standard bloom filter. But the drawback occurs at the power saving ratio which diminishes when there are high no of matches in the first stage and second stage is utilized more. To remove this problem the fully Pipelined architecture of Bloom filters [5] is introduced.

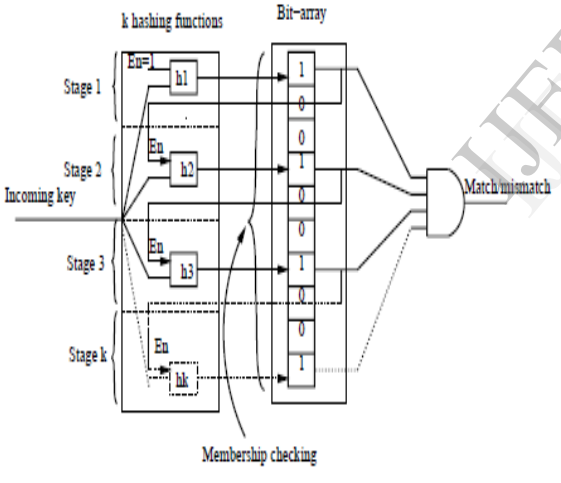

Fig 5: k-stage pipelined bloom filter Architecture.

3.2.2. Fully Pipelined Bloom Filter Architecture

Pipelined architecture of standard bloom filter reduces the false positive probability because first stage utilizes more no of hash function thus leading to increase in the probability of mismatch. Hence second stage is not utilized but more number of hash functions consumes more power which becomes the drawback. In fully pipelined architecture [5] number of stages equals to the number of hash functions and each stage has only one hash function as shown in fig 6. The programming mode is same as in case of standard Bloom Filter. In query mode, test string is progressed to next stage only when a previous hash function produces a match. Thus fully pipelined architecture removes the above drawback as each of its stage has only one hash function which has same false positive probability as the standard bloom filter.

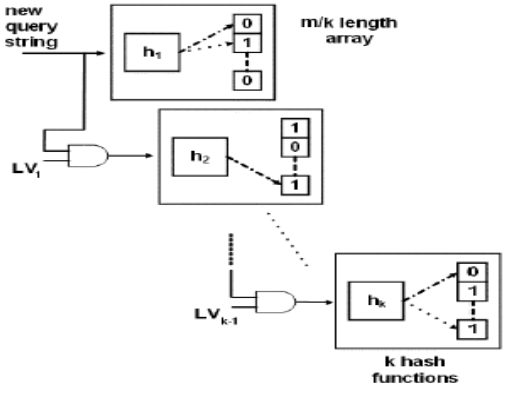

Fig 6: Fully Pipelined Bloom Filter Architecture.

3.2.3. Parallel Pipelined Bloom Filter Architecture

In the previous architecture, when a query string is evaluated on one of the pipeline stages, the rest of the stages remain 'idle'. The Parallel-Pipelined design [9] simultaneously evaluates multiple query strings using the multiple hash functions to speed up the overall function. Fig 8 shows the overall design of this architecture where k different query strings can be evaluated on each hash module. In the next clock cycle, a query string in each pipeline register is sent to the next pipeline register. To concurrently perform multiple queries, additional pipeline registers are used. Additionally, a counter which has log2k size is also designed to control the overall process. When this counter notifies the end of filtering stages or every query string is decided not to be a member, the counter is made reset and new query strings are evaluated. Therefore, the k query strings can be evaluated in k or less number of cycles in this design. Consequently, at most there exists k times improvement in throughput compared to the regular Bloom filter.

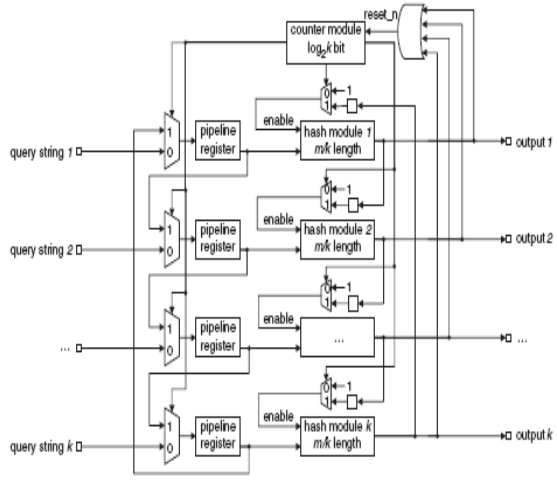

Fig 7: Parallel-Pipelined Bloom Filter Architecture.

3.2.4. Parallel Processing Architecture

Standard bloom filter architecture can effectively represent the items with a single element but it cannot support the programming and querying of items that have multiple elements. To perform the operation on multiple elements items Parallel Bloom filter architecture is developed [2] .In this architecture a number of string inputs can be tested at the same time. Hash values for all of the test strings are tested by checking particular bit locations in m bit look up array as in case of standard bloom filter. Bit location values in look up array are ANDED separately for different hash values of different test string. The advantage of parallel architecture [8] is that the speed of operation with multiple inputs is high. The drawback that occurs with this architecture is area requirement is more thus consumes much power and area.

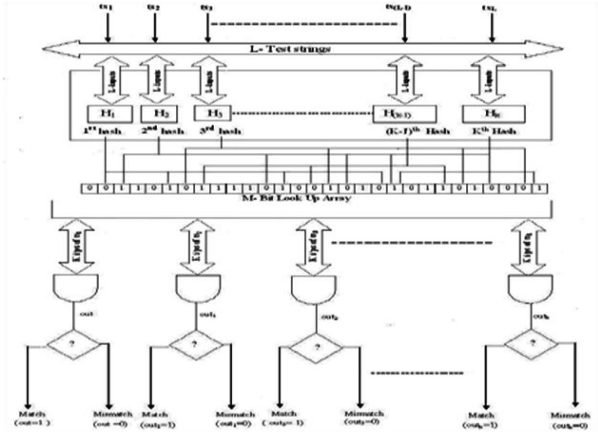

Fig 8: Parallel Processing Architecture

4. Power Consumption Analysis

The analysis of the above architectures can be done with the reference parameter as power consumption. The Power consumption of standard bloom filter is given by,

$$P_{standard}=\sum_{i=1}^{k}(P_H+P_L)+P_R \quad (7)$$

where P_H is the power dissipation by a hash function and P_L is the power required to perform a lookup operation on the array. The term $P_H + P_L$ represents the power consumption on a hash operation.P_R is the power consumption to hold an input query.

The power consumption of the Fully Pipelined architecture is given by,

$$P_{Fullypipelined}=\sum_{i=1}^{k}(P_H+P_L)S^{i-1}+P_R \quad (8)$$

where S is the probability for a query string to pass a matching stage.

The power consumption of the k-stage pipelined architecture is given by,

$$P_{K\,stage\,Pipelined}= P_{H1}+P_{L1}+P_{R1}(P_{H2}+P_{L2})+P_{R1}P_{R2}(P_{H3}+P_{L3}) +\ldots\ldots+P_{R1}P_{R2}\ldots\ldots P_{RK-1}(P_{HK}+P_{LK}) \quad (9)$$

The power consumption of the Parallel pipelined CBF architecture is given by,

$$P_{Parallel\,Pipelined}=\sum_{i=1}^{k}(P_H+P_L)S^{i-1}+KP_R+ KP_r+2KP_{mux}+P_{cnt} \quad (10)$$

where P_r is the power consumption on the 1-bit output buffer for each hash function module. P_{mux} is the power consumption on the 2-input multiplexer and P_{cnt} is the power consumption on the counter.

5. Conclusion and Future Work

In this paper the concept of Bloom filter are over viewed. The detailed explanation of design of Counting Bloom filter for the application of Network Intrusion Detection System in various papers has been analyzed. In the future while comparing to the existing methods, we are interested in developing an efficient counting bloom filter design for the application of network intrusion detection system.

Acknowledgement

The authors acknowledge the contributions of the students, faculty of Velalar College of Engineering and Technology for helping in the design of test circuitry, and for tool support. The authors also thank the anonymous reviewers for their thoughtful comments that helped to improve this paper**.** The authors would like to thank the anonymous reviewers for their constructive critique from which this paper greatly benefited.

References

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

[2] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood, "Deep packet inspection using parallel Bloom filters,"IEEE Micro,vol. 24, no.1,pp52-61, 2004.

[3] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese,"An Improved Construction for Counting Bloom Filters," in *14th* Annual European Symposium on Algorithms, pp. 684–695, 2006.

[4] Elham Safi, Andreas Moshovos and Andreas Veneris," L-CBF: A Low-Power, Fast Counting Bloom Filter Architecture" ., IEEE transactions on very large scale integration (VLSI) systems, vol. 16, no. 6, June 2008.

[5] Michael Paynter and Taskin Kocak,"Fully Pipelined Bloom Filter Architecture", IEEE Communications Letters Vol.12 No. 11, pp. 855-857, November 2008.

[6] 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.

[7] Mahmood Ahmadi, Stephan Wong," *K*-Stage Pipelined Bloom Filter for Packet Classification", International Conference on Computational Science and Engineering,IEEE Computer society,,2009.

[8] Bin Xio & YuHua "using Parallel Bloom Filters for multi attribute Representation on Network Services"IEEE Transaction on Parallel and Distributed Systems, vol.21, n0.1, January 2010.

[9**]** Deokho Kim, Doohwan Oh and Won W. Ro," Design of power-efficient parallel pipelined Bloom filter", Electronics Letters, 29th March 2012, Vol. 48 No. 7