International Journal of Automation and Computing 9(4), August 2012, 395-402
DOI: 10.1007/s11633-012-0660-2

Functional Verification of Signature Detection Architectures for High Speed Network Applications

M. Arun[1] A. Krishnan[2]

[1]Department of Electronics and Communication Engineering, Sri Krishna College of Engineering and Technology, Coimbatore 641008, India
[2]K. S. Rangasamy College of Technology, Tiruchengode 637215, India

Abstract: To meet the future internet traffic challenges, enhancement of hardware architectures related to network security has vital role where software security algorithms are incompatible with high speed in terms of Giga bits per second (Gbps). In this paper, we discuss signature detection technique (SDT) used in network intrusion detection system (NIDS). Design of most commonly used hardware based techniques for signature detection such as finite automata, discrete comparators, Knuth-Morris-Pratt (KMP) algorithm, content addressable memory (CAM) and Bloom filter are discussed. Two novel architectures, XOR based pre computation CAM (XPCAM) and multi stage look up technique (MSLT) Bloom filter architectures are proposed and implemented in third party field programmable gate array (FPGA), and area and power consumptions are compared. 10 Gbps network traffic generator (TNTG) is used to test the functionality and ensure the reliability of the proposed architectures. Our approach involves a unique combination of algorithmic and architectural techniques that outperform some of the current techniques in terms of performance, speed and power-efficiency.

Keywords: Signature detection, network intrusion detection system (NIDS), content addressable memory (CAM), Bloom filter, network security.

1 Introduction

Some content strings of Internet packet payload, also known as "signatures", imply network intrusion attempts. Signature based network intrusion detection system (NIDS) collects these signatures and scans the payload of the Internet packets for them in order to identify, deter and contain such malicious behaviors. A scalable and fast solution is needed to accommodate the largest signature set today and to sustain the real time processing of the high-speed network. This is very challenging especially for today′s high-speed networks with line speeds of 10 Giga bits per second (Gbps) and beyond. Software based NIDSs are not scalable to high-speeds[1, 2]. Hardware NIDSs have gained a lot of attention recently due to the intrinsic speed advantage over software systems.

In this paper, we present the implementation of signature detection technique (SDT) architectures in field programmable gate array (FPGA), which can provide 10 Gbps throughput using a single commodity FPGA. Available maximum speed of optical cable (OC) is 40 Gbps. Nowadays state-of-the-art FPGAs are compatible for high speed network applications of 40 Gbps or more[3]. Three different types of architectures, random access memory (RAM), content addressable memory (CAM) and Bloom filter, are selected for the study purpose. Two enhanced novel architectures, XOR based pre computation CAM (XPCAM) and multi stage look up technique (MSLT) Bloom filter are proposed and implemented in the FPGA.

The contributions of this paper are as follows:

1) Detailed hardware architectures XPCAM and MSLT Bloom filter are proposed.

2) Mathematical analyses are conducted to compare the performance of proposed architectures.

3) Proposed architectures are implemented using Virtex5 XC5VLX85 & back end tool and compared.

4) Proposed architectures are tested in 10 Gbps network traffic generator (TNTG).

The rest of the paper is organized as follows. Section 2 summarizes the related work on hardware-based signature detection techniques. Sections 3 and 4 define the problem of signature detection for NIDS using CAM and Bloom filter respectively, whereas Section 5 shows the proposed architectures. Section 6 shows implementation results of the proposed architectures. Section 7 describes the experience during the functional verification of the proposed architectures. Finally, Section 8 concludes the paper.

2 Related work

Software-based intrusion detection systems can only support modest throughput. On the other hand, hardware can easily adapt in NIDS application needs, achieving better performance with reasonable cost. In this section, we investigate various hardware-based solutions for string matching. FPGAs are more suitable, because they are reconfigurable; they provide hardware speed and exploit parallelism. This characteristic of reconfigurable devices allows updating or changing the rule set, adding new features, even changing systems architecture, without any hardware cost. One of the first attempts in string matching using FPGAs, presented in 1993 by Pryor et al.[4] Most researchers designed signature detection architectures based on regular expressions (NFAs and DFAs)[5−8]. This is a low cost solution, but does not achieve very high performance. A widely used technique to increase sharing and reduce designs cost is the use of pre-decoding, which was applied to both regular expression and CAM-like approaches[9−11]. Pre-decoding has been recently introduced and used by several research

Manuscript received March 27, 2011; revised November 2, 2011

groups. It is based on the idea that incoming data are predecoded in centralized decoders, so that each unique char acter is matched only once. A more efficient and very low cost approach was presented by Dharmapurikar et al.[12] who implemented Bloom filters to perform string matching.

In the past decade, much research on energy reduction has focused on the circuit and technology domains[13]. Several works on reducing CAM power consumption have focused on reducing match-line power[14,15]. Another approach is to use Bloom filter. Bloom filter is a space efficient, randomized data structure for representing a signature set and supporting set membership queries. Bloom filter summarizes the contents (files, keywords or objects) into a compact form that is orders of magnitude smaller than the original collection. The reduction in size may help to improve system performance and conserve system resources such as network bandwidth, memory capacity and disk space. We have seen quite few network system designs, prototypes and real deployments that utilize Bloom filters for content representation[16].

Instead of working in device level, our approach concentrates on pre-computation techniques. Bloom filter based MSLT architecture and pre-computation based CAM using XOR gates are proposed where performance is improved by reducing the number of comparisons. Proposed architectures are described and implemented in third party FPGA. FPGA-based platforms can exploit the fact that the NIDS rules change relatively infrequently, and use reconfiguration to reduce implementation cost. In addition, FPGA-based systems can exploit parallelism in order to achieve satisfactory processing throughput.

3 Content addressable memory

A content-addressable memory (CAM) is a critical device for applications involving communication networks, Local area network bridges/switches, databases, lookup tables, and tag directories, due to its high-speed data search capability. Fig. 1 shows the memory organization of the pre computation based (PB)-CAM architecture proposed by Lin et al.[17], which consists of data memory, parameter memory, and parameter extractor, where $k \ll n$. To reduce massive comparison operations for data searches, the operation is divided into two parts. In the first part, the parameter extractor extracts a parameter from the input data, which is then compared to parameters stored in parallel in the parameter memory. If no match is returned in the first part, it means that the input data mismatch the data related to the stored parameter. Otherwise, the data related to those stored parameters have to be compared in the second part. It should be noted that although the first part must access the entire parameter memory, the parameter memory is far smaller than that of the CAM. Moreover, since comparisons made in the first part have already filtered out the unmatched data, the second part only needs to compare the data that match with the first part. The PB-CAM exploits this characteristic to reduce the comparison operations, thereby saving power. Therefore, the parameter extractor of the PB-CAM is critical, because it determines the number of comparison operations in the second part.

As we stated previously, the parameter extractor plays a significant role since this circuit determines the number of comparison operations required in the second part. Therefore, the design goal of the parameter extractor is to filter out as many unmatched data as possible to minimize the required number of comparison operations in the second part. The ones-count function was adapted to perform parameter extraction in [17]. For ones count approach, with an n bit data length, there are $n+1$ type of one′s count (from 0 ones to n ones count). Further, it is necessary to add an extra type of one′s count to indicate the availability of stored data. Therefore, the minimal bit length of the parameter is equal to $\log(n+2)$. The parameter extractor for the ones-count approach is implemented with full adders as shown in Fig. 2, where FA denotes full adder, INCR4 denotes 4 bit increment. In Section 2.2, we will use a 14 bit example to illustrate the ones-count PB-CAM system and discuss the disadvantages by mathematical analysis. For a 16 bit length input data, all the input data contain 2^{16} numbers, and the number of input data related to the same parameter for ones count approach is ${}_{16}\mathrm{C}_r$, where r is a type of one′s-count (from 0 to 15 one′s-counts). Then we can compute the average probability that the parameter occurs. The average probability can be determined by

$$\text{Average probability} = \frac{{}_{16}\mathrm{C}_r}{2^{16}} \tag{1}$$

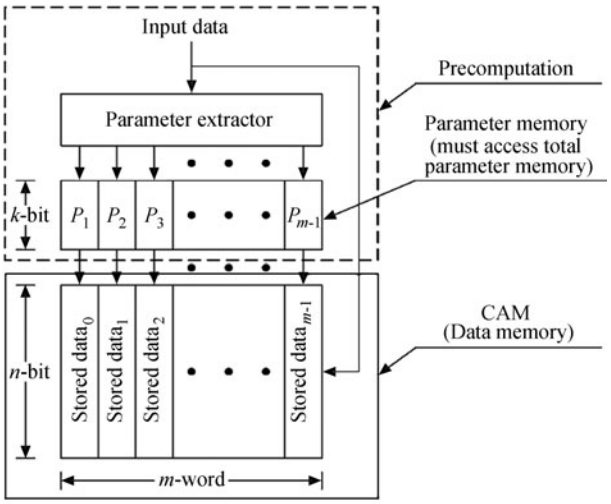

Fig. 1 PB-CAM Architecture

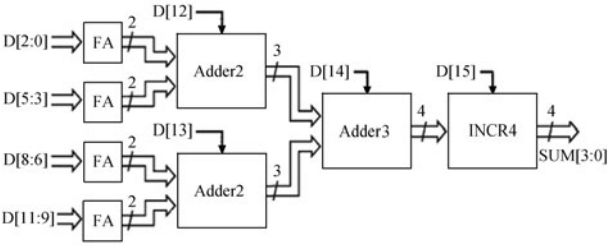

Fig. 2 One′s count parameter extractor

Table 1 lists the number of data related to the same parameter and their average probabilities for the input data that is 16 bit in length. Note that with conventional CAMs, the comparison circuit must compare all

stored data, whereas with the ones-count PB-CAMs, a large amount of unmatched data can be initially filtered out, reducing comparison operations for minimum power consumption in some cases. However, the average probabilities of some parameters, such as 0, 1, 2, 3, 13, 14, 15, and 16 are less than 1 %. In Table 1, we can see that parameters with over 2000 comparison operations range between 4 and 12. However, the summation of the average probabilities for these parameters is close to 92 %. Although the number of comparison operations required for ones-count PB-CAMs is fewer than that of conventional CAMs, ones-count PB-CAMs fail to reduce the number of comparison operations in the second part when the parameter value is between 5 and 9, thereby consuming a large amount of power. As can be seen in Fig. 3, random input patterns for the ones-count approach demonstrate the Gaussian distribution characteristic. Note that the Gaussian distribution will limit any further reduction of the comparison operations in PB-CAMs.

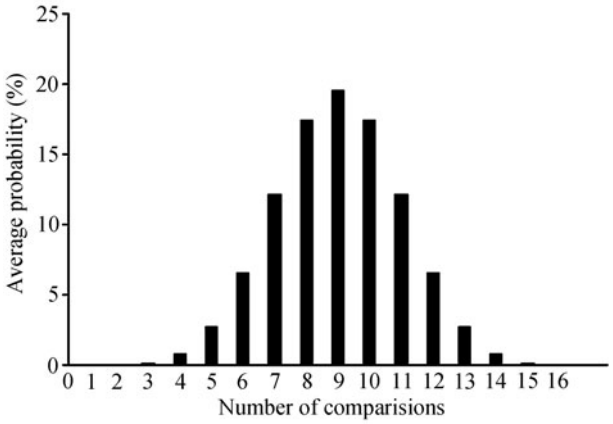

Fig. 3 Gaussian distribution

Table 1 Number of comparisons and average probabilities for the ones count approach

Parameter		Number of comparisons	Average probability
0000	0	1	0.001525879
0001	1	16	0.024414063
0010	2	120	0.183105469
0011	3	560	0.854492188
0100	4	1820	2.777099609
0101	5	4368	6.665039063
0110	6	8008	12.21923828
0111	7	11440	17.45605469
1000	8	12870	19.63806152
1001	9	11440	17.45605469
1010	10	8008	12.21923828
1011	11	4368	6.665039063
1100	12	1820	2.777099609
1101	13	560	0.854492188
1110	14	120	0.183105469
1111	15	16	0.024414063
Valid	16	1	0.001525879

4 Bloom filter

Bloom filter was formulated by Bloom[18] and is used widely today for different purposes including web caching, intrusion detection and content based routing. The theory behind Bloom filters is described in this section. Given a string X, the Bloom filter computes k hash functions producing hash values ranging from 1 to m. It then sets k bits in an m bit long vector at the addresses corresponding to the k hash values. The same procedure is repeated for all the members of the set. This process is called "programming" of the filter illustrated in Fig. 4. The query process is similar to programming, where a string whose membership is to be verified is input to the filter. The Bloom filter generates k hash values using the same hash functions it used to program the filter. However, finding an unset bit certainly implies that the string does not belong to the set, since if it did then all the k bits would definitely have been set when the Bloom filter was programmed with that string. This explains the presence of false positives in this scheme, and the absence of any false negatives. The concept is illustrated in Figs. 5 and 6. The false positive rate, f, is

$$f = (1 - \mathrm{e}^{-\frac{nk}{m}})^k \tag{2}$$

where, n is the number of strings programmed into the Bloom filter. The value of f can be reduced by choosing appropriate values of m and k for a given size of the member set, n. It is clear that the value of m needs to be quite large compared to the size of the string set, i.e., n. Also, for a given ratio of m/n, the false positive probability can be reduced by increasing the number of hash functions k. In the optimal case, when false positive probability is minimized with respect to k, we get the following relation:

$$f = \frac{m}{n} \ln 2. \tag{3}$$

This corresponds to a false positive probability of:

$$f = (\frac{1}{2})^k. \tag{4}$$

The ratio m/n can be interpreted as the average number of bits consumed by a single member of the set.

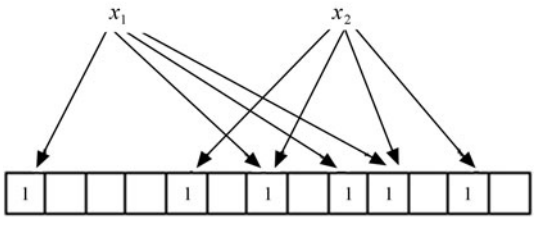

Fig. 4 Programming multiple strings in the Bloom filter

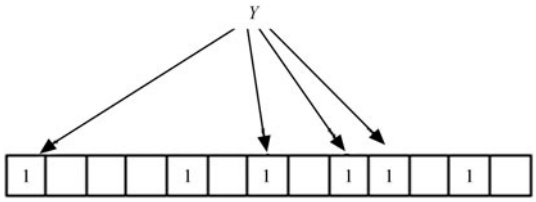

Fig. 5 Querying a Bloom filter with a string

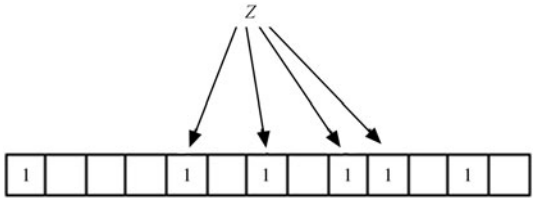

Fig. 6 False positives

A block diagram of a typical Bloom filter is illustrated in Fig. 7, where HF denotes hash function. Given a string X, which is a member of the signature set, a Bloom filter computes k many hash values on the input X and d which are uniformly distributed between 1 to number of hash functions, k. Then it uses these hash values as index to the m-bit long lookup vector. It sets the bits corresponding to the index given by the hash values computed. It repeats this procedure for each member of the signature set.

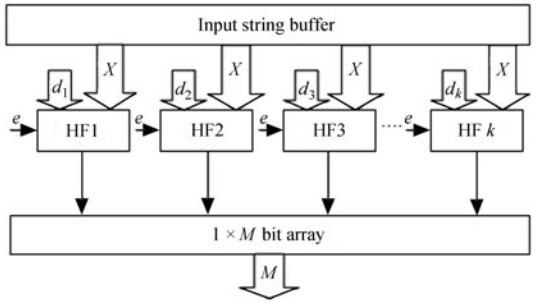

Fig. 7 Block diagram of typical Bloom filter

For an input string Y, Bloom filter computes k many hash values by utilizing the same hash functions used in programming of the Bloom filter. Bloom filter looks up the bit values located on the offsets (computed hash values) on the bit vector. If it finds any bit unset at those addresses, it declares the input string to be a nonmember of the signature set, which is called a mismatch. Otherwise, it finds all the bits are set, it concludes that input string may be a member of the signature set with a false positive probability, which is called a match.

5 Low power architectures

5.1 XOR based CAM

The key idea behind our method is to reduce the number of comparison operations by eliminating the Gaussian distribution explained in Section 3. For a 16 bit input data, if we can distribute the input data uniformly over the parameters, then the number of input data related to each parameter would be $2^{14}/16 = 4096$, and the maximum number of required comparison operations would be $2^{14}/16 = 4096$ for each case in the second part of the comparison process. Compared with the ones-count approach, this approach can reduce comparison operations by a minimum of 3912 and a maximum of 8774 (i.e., for parameter value from 4 to 12) for 92 % of the cases. Based on these observations, we propose a new parameter extractor called Block-XOR, which is shown in Fig. 8 to achieve the previous requirement. In our approach, we first partition the input data bit into several blocks, from which an output bit is computed using XOR logic operation for each of these blocks. The output bits are then combined to become the input parameter for the second part of the comparison process. To compare with the ones-count approach, we set the bit length of the parameter to $\log n$, where n is the bit length of the input data. Therefore, the number of blocks is $n/\log n$ in our approach. The selected signal is defined as

$$S = A_3A_2A_1A_0. \tag{5}$$

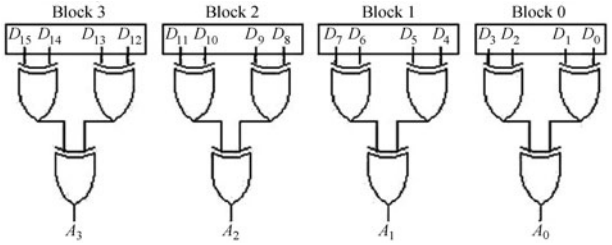

Fig. 8 XOR based pre-computation block

The concept of Block-XOR approach is to uniformly distribute the parameter over the input data. By the rule of product, the number of input data that results in the same parameter is $8 \times 8 \times 8 \times 8 = 4096$. Consequently, the average probability can be determined as $4096/2^{16} \times 100\,\% = 6.25\,\%$. Obviously, the concept of Block-XOR approach can reduce the comparison operations, hence minimize power consumption. Table 2 lists the number of input data that result in the same parameter for the proposed Block-XOR PB-CAM. As can be seen from Tables 1 and 2, in most cases, the proposed Block-XOR PB-CAM required far fewer comparison operations than the ones-count approach for parameter values between 4 and 12 illustrated in Fig. 9.

Table 2 Number of comparisons and average probabilities for the Block-XOR approach

Parameter		Number of comparisons	Average probability
0000	0	4096	6.25
0001	1	4096	6.25
0010	2	4096	6.25
0011	3	4096	6.25
0100	4	4096	6.25
0101	5	4096	6.25
0110	6	4096	6.25
0111	7	4096	6.25
1000	8	4096	6.25
1001	9	4096	6.25
1010	10	4096	6.25
1011	11	4096	6.25
1100	12	4096	6.25
1101	13	4096	6.25
1110	14	4096	6.25
1111	15	4096	6.25
Valid	16	4096	6.25

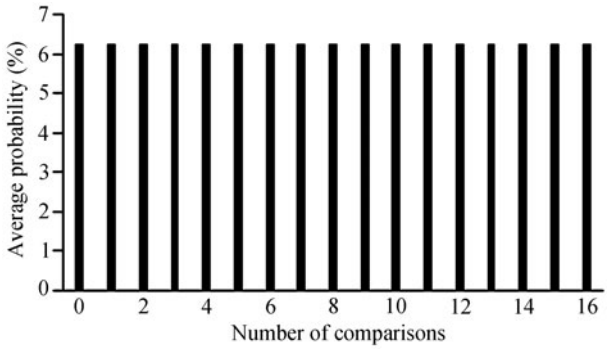

Fig. 9 Uniform distribution

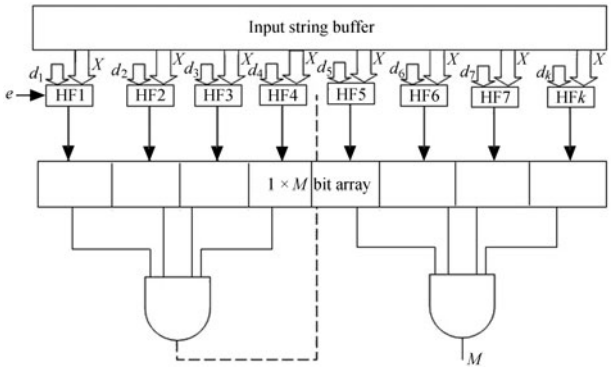

Fig. 10 Bloom filter architecture (hash per stage $r = k/2$)

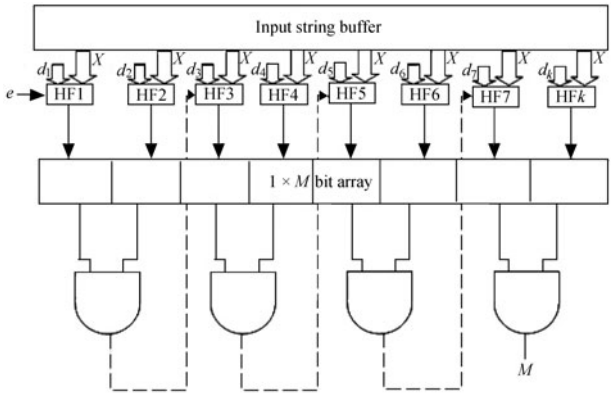

Fig. 11 Bloom filter architecture (hash per stage $r = k/4$)

5.2 MSLT based Bloom filter

A Bloom filter never produces false negatives, which means if it decides that an input is a nonmember, then the input certainly does not belong to the signature set. However, it may produce false positives. It may conclude that the input is a member of the signature set, although in reality the input may not be a member of the set. Following the analysis of [12], the false positive probability f is calculated by (2). In order to minimize the false positive probability, the value of $m$must be quite larger than n. For a fixed value of m/n, k must be large enough such that f gets minimized. Since the number of hash functions in Bloom filters is large to reduce the false positive probability, it is intuitive that their total power consumption is large. During the programming phase of the Bloom filter, not much can be done to reduce the power consumption; otherwise Bloom filter will produce many false positives. However, while performing lookups over the Bloom filter, the number of hash functions used to produce a decision can be reduced significantly. The architecture to support such a lookup operation for a multi-hashing scheme is illustrated in Fig. 10 where number of hash functions per stage (r) is $k/2$. We have introduced low power Bloom filter architectures where $r = k/4$ is illustrated in Fig. 11. If match is achieved in the first stage itself then three fourth of the hash calculations are minimized when half of the hash calculations are reduced. In a similar fashion we have considered low power architecture with $k/8$ for power analysis. $H3$ class of universal hash function was used in the hash calculations of MSLT. Universal class of hash functions were first introduced by Carter and Wegman[19]. Given any string X, consisting of b bits, $X = \langle x_1, x_2, x_3, \cdots, x_b \rangle$ i-th hash function over the string X is defined as

$$h_i(x) = d_{i1} \cdot x_1 \text{ xor } d_{i2} \cdot x_2 \text{ xor } d_{i3} \cdot x_3 \text{ xor } \cdots \text{ xor } d_{ib} \cdot x_b \quad (6)$$

where $d_{ij}{}'$s are random coefficients uniformly distributed between 1 to size of the lookup vector, m, and x_k is the k-th bit of the input string. "$\cdot$" is a bit by bit AND operation, and xor is a logical exclusive OR operation.

5.3 Theoretical power analysis of MSLT architectures

A theoretical approach is followed to analyze and compare the power consumptions of the different lookup operations available through Bloom filter architectures presented in the Section 5.2. Power consumption (see (7)) of a Bloom filter when performing a regular lookup operation is a summation of the power consumptions of each of the hash function computations, i.e., P_{Hi}, plus the power consumed in accessing the memory for each hash value computed, i.e., P_Q, plus the power consumed by an AND gate.

$$P_{\mathrm{BF}_{reg}} = \sum_{i=1}^{k} (P_{Hi} + P_Q) + P_{\mathrm{AND}}. \quad (7)$$

Power consumption of an AND gate is ignored hereafter, since it is negligible compared to the power used by the hash functions. Hence, all of the k many hash functions are of type 8 bit $H3$ class of hash functions, so (7) becomes

$$P_{\mathrm{BF}_{reg}} = k(P_8 + P_Q) \quad (8)$$

The probability that a bit is still unset after all the signatures are programmed into the Bloom filter by using k-many independent hash functions is α.

$$\alpha = \left(1 - \frac{1}{m}\right)^{kn} \approx \mathrm{e}^{-\frac{kn}{m}} \quad (9)$$

where $1/m$ represents any one of the m bits set by a single hash function operating on a single signature. Then

$(1-1/m)$ is the probability that the bit is unset after a single hash value computation with a single signature. For it to remain unset, it should not be set by any of the k-many hash functions each operating on all of the n-many signatures in the signature set. Consequently, the probability that any one of the bits is set is

$$(1-\alpha) = 1 - \mathrm{e}^{-\frac{kn}{m}}. \tag{10}$$

In order for the first stage to produce a match, the bits indexed by all r of the independent random hash functions should be set. So the match probability of the first stage is, represented as p,

$$p = \prod_{i=1}^{r}(1-\alpha) = (1-\alpha)^r \approx \left(1 - \mathrm{e}^{-\frac{kn}{m}}\right)^r. \tag{11}$$

Power consumption of MSLT Bloom filter architectures where $r = k/2$, $k/4$ and $k/8$ are given by

$$P_{\mathrm{BF}_{r=\frac{k}{2}}} = \frac{k}{2} \times (P_{H8} + P_Q) \times \left(1 + p^{\frac{k}{2}}\right) \tag{12}$$

$$P_{\mathrm{BF}_{r=\frac{k}{4}}} = \frac{k}{4} \times (P_{H8} + P_Q) \times \left(1 + p^{\frac{k}{4}} + p^{\frac{k}{2}} + p^{\frac{3k}{4}}\right) \tag{13}$$

$$P_{\mathrm{BF}_{r=\frac{k}{8}}} = \frac{k}{8} \times (P_{H8} + P_Q) \times \left(1 + p^{\frac{k}{8}} + p^{\frac{k}{4}} + p^{\frac{3k}{8}} 1 + p^{\frac{k}{2}} + p^{\frac{3k}{8}} + p^{\frac{3k}{4}} + p^{\frac{7k}{8}}\right). \tag{14}$$

The power saving ratio (PSR), in a single Bloom filter implemented based on the architectures presented functioning on two different lookup techniques can be calculated as

$$\mathrm{PSR} = \frac{\left(P_{\mathrm{BF}_{reg}} - P_{\mathrm{BF}_{r=\frac{k}{n}}}\right)}{P_{\mathrm{BF}_{reg}}}. \tag{15}$$

Using (15), with reference to the power consumption of BF_{reg}, PSR of $\mathrm{BF}_{r=k/2}$, $\mathrm{BF}_{r=k/4}$ and $\mathrm{BF}_{r=k/8}$ are calculated for various k values by considering the design specifications shown in Table 3. For 8 bit signature length (i) with m/n ratio as 24, power consumption $(P_{H8} + P_Q)$ of an 8 bit hash function was estimated by third party back end tool and substituted in (8).

Table 3 Design specifications

Parameters	Values
m/n ratio	21
Number of signatures, n	1024
Size of the m bit vector, m	21504
Width of the signature, i	8
$P_{H8} + P_Q$	0.801 μW

Observation says that increment in k increases the number of basic functional modules used in the design which increases device density. Obviously device density is directly proportional to power consumption, by the observation from Fig. 12, which cannot be compensated using parallel look up techniques proposed.

In this section, we have discussed about the proposed methods and their power consumption which is compared with the related work, theoretically. In Section 6, we will see the performance analysis on the proposed architectures where the speed and device utilization are considered.

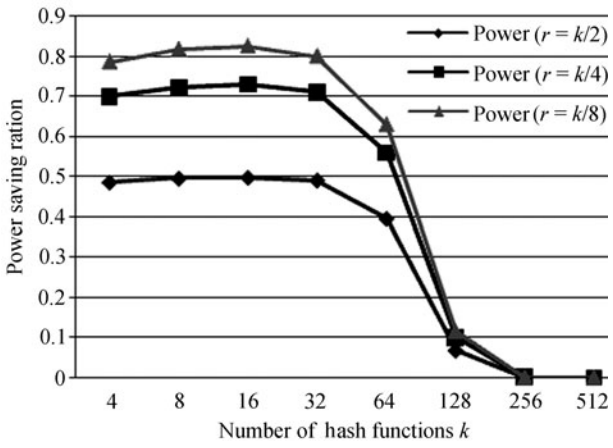

Fig. 12 Power saving ratio (PSR)

6 Implementation results

In this section, we present the results of hardware simulation implemented in Xilinx 10.1i. The simulation of all existing and proposed architectures were synthesized, placed, and routed on the Virtex5 XC5VLX85 chip where the package and speed are FF676 and –3, respectively. We use a new metric "Performance" defined to be the area cost divided by implementation speed as follows.

$$\mathrm{Performance} = \left(\frac{\mathrm{AreaCost}}{\mathrm{Speed}}\right) = \left(\frac{\frac{\mathrm{Signature}}{\mathrm{Slice}}}{\mathrm{Period}}\right). \tag{16}$$

Table 4 summarizes the performance comparison between related works and our design. In Table 4, LUT means look up table. We focus on the Signature/Slice metric. Our design has the best throughput in all approaches. We also can see that our design has higher throughput and performance than related works[17, 20–22]. Study of Table 4 says that performance proposed CAM and Bloom filter based architectures out performs the existing works. When we compare XPCAM and MSLT architectures, hash based Bloom filter MSLT architectures are suitable for applications based on their device utilization, performance and bandwidth. Even XPCAM architectures are immediate follower in the performance table; they are essential contributors for certain applications where data retrieval is unavoidable.

Bloom filter and CAM based architectures explained in previous sections are implemented in third party FPGA and their device utilization details are summarized. Input signature length is considered as 16 bit for all the architectures. Due to the reduction in IOBs, PB CAM results better than RAM. XPCAM consumes less power and space than previous cases with the help of XOR based pre computation block. Comparisons show that number of bits is reduced in BF_{reg} and further reduced in MSLT by the proposed lookup technique. Functionality of proposed architectures is verified using the 10 Gbps network traffic generator (TNTG)–Test Environment.

Table 4 Implementation results

Design	Device	Size of the signature	No. of signatures	Slice	No. of registers	No. of LUT	Signature Slice	Period	Throughput Gbps	Performance
Proposed	Virtex5LX85T	32	16028	15531	15943	48767	1.03	5.10	6.27	0.20
XPCAM		16	16028	7221	15640	15736	2.22	3.42	4.68	0.65
OCCAM[17]		32	16028	37740	22945	5784	0.42	5.80	5.52	0.07
		16	16028	20500	20145	22254	0.78	5.70	2.81	0.14
Proposed		32	16028	7635	6550	30387	2.10	3.45	9.28	0.61
MSLT		16	16028	5626	9690	15375	2.85	3.42	4.68	0.83
DSLT[20]		32	16028	17239	8775	42632	0.93	4.60	6.96	0.20
		16	16028	8852	13758	28574	1.81	4.75	3.37	0.38
Discrete comparator[21]	Virtex26000	32	2457	23843	–	–	0.10	4.10	7.80	0.03
Sharing prefix[22]	Virtex26001	32	8003	10309	–	–	0.78	12.4	2.59	0.06

7 Testing

Fig. 13 shows the TNTG–Test Environment used to verify the proposed SDT architectures. This network has been modified slightly from the one first developed for the 1998 evaluation[23].

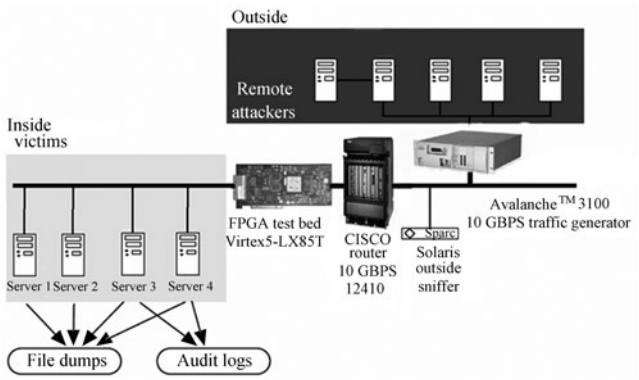

Fig. 13 TNTG–Test Environment

Test setup generates and captures live traffic at the rate of 10 Gbps which is seen between a victim base and the Internet. Background traffic is generated that simulates hundreds of programmers, secretaries, managers, and other types of users running common UNIX and Windows NT programs.

At the same time, attacks are launched against the Cisco router (12410) and the four primary victim systems (light grey box) running with different server operating systems.

The proposed MSLT Bloom filter and XPCAM architectures are described in Verilog HDL and implemented in the inbuilt Virtex5 LX85T FPGA accommodated in the test bench. Proposed architectures′ functionality is verified using the TNTG–Test Environment.

8 Conclusions

Network intrusion detection system is a system which can detect network attacks resulted from worms and viruses on the Internet. An efficient signature detection architecture plays an important role in NIDS. Hardware based NIDS are an alternative to meet the in line wire speed in terms of Gbps for network applications. In this work, we propose pre-processing based architectures to improve the performance of the SDTs. The design of the proposed architecture and its implementation in FPGA are described in detail. Our simulation results show that the proposed architecture performs better than all the existing approaches in terms of the throughput and the device utilization. We also tested a proof-of-concept system with 10 Gbps traffic. This work focuses on detection of known attacks as signatures contained in a single packet.

The proposed architectures may not suitable to detect unknown attacks. As future work, we would like to further develop this approach to detect unknown attacks and extend with application specific integrated circuit (ASIC) design.

References

[1] D. J. Day, Z. X. Zhao. Protecting against address space layout randomization (ASLR) compromises and return-to-Libc attacks using network intrusion detection systems. *International Journal of Automation and Computing*, vol. 8, no. 4, pp. 472–483, 2011.

[2] S. S. S. Sindhu, S. Geetha, M. Marikannan, A. Kannan. A neuro-genetic based short-term forecasting framework for network intrusion prediction system. *International Journal of Automation and Computing*, vol. 6, no. 4, pp. 406–414, 2009.

[3] H. Shrikumar. 40Gbps de-layered silicon protocol engine for TCP record. In *Proceedings of Design, Automation and Test in Europe*, IEEE, Munich, Germany, pp. 1–6, 2006.

[4] D. V. Pryor, M. R. Thistle, N. Shirazi. Text searching on splash 2. In *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, IEEE, Napa, USA, pp. 172–177, 1993.

[5] R. Sidhu, V. K. Prasanna. Fast regular expression matching using FPGAs. In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE, Rohnert Park, USA, pp. 227–238, 2001.

[6] R. Franklin, D. Carver, B. L. Hutchings. Assisting network intrusion detection with reconfigurable hardware. In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE, Napa, USA, pp. 111–120, 2002.

[7] J. Moscola, J. Lockwood, R. P. Loui, M. Pachos. Implementation of a content-scanning module for an internet firewall. In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE, pp. 31–38, 2003.

[8] C. R. Clark, D. E. Schimmel. Efficient reconfigurable logic circuit for matching complex network intrusion detection patterns. *Lecture Notes in Computer Science, Springer*, vol. 2778, pp. 956–959, 2003.

[9] C. R. Clark, D. E. Schimmel. Scalable parallel pattern-matching for high-speed networks. In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE, pp. 249–257, 2004.

[10] Y. H. Cho, W. H. Mangione-Smith. Deep packet filter with dedicated logic and read only memories. In *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE, pp. 125–134, 2004.

[11] Z. K. Baker, V. K. Prasanna. Time and area efficient reconfigurable pattern matching on FPGAs. In *Proceedings of ACM International Symposium on Field-Programmable Gate Arrays*, ACM, Monterey, USA, pp. 223–232, 2004.

[12] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, J. W. Lockwood. Deep packet inspection using parallel bloom filters. *IEEE Micro*, vol. 24, no. 1, pp. 52–61, 2004.

[13] K. Pagiamtzis, A. Sheikholeslami. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[14] H. Miyatake, M. Tanaka, Y. Mori. A design for high-speed-low power CMOS fully parallel content-addressable memory macros. *IEEE Journal of Solid-State Circuits*, vol. 6, no. 6, pp. 956–968, 2001.

[15] I. Arsovski, A. Sheikholeslami. A mismatch-dependent power allocation technique for match-line sensing in content-addressable memories. *IEEE Journal of Solid-State Circuits*, vol. 38, no. 11, pp. 1958–1966, 2003.

[16] H. Cai, P. Ge, J. Wang. Applications of bloom filters in peer-to-peer systems: Issues and questions. In *Proceedings of International Conference on Networking, Architecture, and Storage*, IEEE, Chongqing, China, pp. 97–103, 2008.

[17] C. S. Lin, J. C. Chang, B. D. Liu. A low-power pre-computation-based fully parallel content-addressable memory. *IEEE Journal of Solid-State Circuits*, vol. 38, no. 4, pp. 654–662, 2003.

[18] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* , vol. 13, no. 7, pp. 422–426, 1970.

[19] J. L. Carter, M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, 1979.

[20] I. Kaya, T. Kocak. Low-power Bloom filter architecture for deep packet inspection. *IEEE Communications Letters*, vol. 10, no. 3, pp. 210–212, 2006.

[21] I. Sourdis, D. Pnevmatikatos. Fast, large-scale string match for a network intrusion detection system. In *Proceedings of International Conference on Field Programmable Logic and Applications*, pp. 880–889, 2003.

[22] B. L. Hutchings, R. Franklin, D. Carver. Assisting network intrusion detection with reconfigurable hardware. In *Proceedings of IEEE Symposium on Field-Programmable Custom Computer*, IEEE, Napa, USA, pp. 111–120, 2006.

[23] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, M. A. Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of DARPA Information Survivability Conference and Exposition*, IEEE, Hilton Head, USA, vol. 2, pp. 12–26, 2000.

M. Arun received the B. Eng. degree in electrical and electronics engineering from Thiagarajar College of Engineering, Madurai, India in 2002, and M. Eng. degree in VLSI design from Anna University, Chennai, India in 2004 and he is a Ph. D. candidate at Anna University, Chennai, India. He is currently an associate professor in the Department of Electronics Engineering at Sri Krishna College of Engineering, Coimbatore, India. He has published over 8 papers in international journals and 3 in technical conferences. He is a member of IEEE an ISTE.

His research interests include low power VLSI, high performance computer networks, and quantum reversible logic.

E-mail: aruninvlsi@gmail.com (Corresponding author)

A. Krishnan received his B. Sc. degree in physics from Madras University, Chennai India in 1963, B. Eng. degree in electrical engineering from College of Engineering, Chennai, India in 1966, M. Eng. in control systems from PSG College of Technology, Coimbatore, India and Ph. D. degree from Indian Institute of Technology, Kanpur, India in 1979. He is currently the dean at K. S. Rangasamy College of Technology, Tiruchengode, India. He has published over 160 papers in journals and technical conferences. He is a senior member of IEEE and ISTE.

His research interests include control systems and digital systems.

E-mail: a_krishnan26@hotmail.com