Gigabit Network Intrusion Detection System Using Extended Bloom Filter in Reconfigurable Hardware

Akshay Eldho Jose and T. Gireeshkumar

Abstract Network intrusion detection system collects information from network and identifies all the possible existing network security threats. Software based detection systems are common but are not good enough for the current network security requirements. Present day network intrusion detection needs wire-level data transfer to avoid the inefficiency in pattern matching process. Hardware based solutions like field programmable gate array which is known for its high processing capability can easily solve these issues. This paper implements a hardware based gigabit intrusion detection system using extended Bloom filter concepts. The paper presents a solution to reduce the high error rate of Bloom Filter by introducing a Reference Vector to the work and evaluates its performance. The reference vector verifies the Bloom filter output for any possible false positive results and reduces the error rate in the system.

Keywords Network intrusion detection · Field programmable gate array · Extended bloom filter · Reference vector

1 Introduction

Network security refers to the protection of various resources from all kind of malicious activities and ensures safety to network and data. It implements the security policies and analyzes various threats and stops it from entering the network. Network security consist of many layers such as firewall and network

A.E. Jose (✉) · T. Gireeshkumar
TIFAC CORE in Cyber Security, Amrita Vishwa Vidyapeetham,
Coimbatore, Tamil Nadu, India
e-mail: akshayeldhojose@gmail.com

T. Gireeshkumar
e-mail: gireeshkumart@gmail.com

© Springer India 2016

S.C. Satapathy et al. (eds.), *Proceedings of the Second International Conference on Computer and Communication Technologies*, Advances in Intelligent Systems and Computing 379, DOI 10.1007/978-81-322-2517-1_2

intrusion detection systems. Firewalls are used to block the access between the networks but it does not study the traffic nor alert the administrator. An intrusion detection system is capable of studying the traffic patterns and compares it against the known attack patterns. It has the capability to inspect the packet contents deeply and protects against network threats.

Software based network intrusion detections are common which can be implemented easily. Snort is an example of open source light weight intrusion detection system [1] which uses signatures to compare thousands of attack patterns. But if the network falls in gigabit speed, it will be difficult for the software to support the system. Hardware solutions solve these issues by converting the rules into Hardware description language. Field programmable gate array is one such hardware which is designed to be configured by a Hardware description language. It consists of many high speed logic blocks capable of parallel processing to produce high performance gain. The general functional model consists of three sections: buffering, packet analyzer and rule matching section. The gigabit network is connected to hardware through Ethernet interfaces. The packets are queued in buffer section to balance between hardware and network. Packets are forwarded to the packet analyzer where it extracts the information from packet. These information are then compared against a set of rules in rule matcher. The alerting and logging mechanism works according to the output of rule matcher.

In this paper, a gigabit network intrusion detection system is designed based on Bloom Filter, to identify the attack patterns in the network. The paper efficiently designs Bloom filter algorithm for string matching engine (SME). Bloom filter test the participation of an element from a group of elements. In this approach, the elements in a group are hashed with multiple hash functions and are stored in the memory. This stored information can be used to check whether a given element belongs to the group or not. The major advantage in Bloom filter is the constant amount of memory needed to store the hashed values irrespective of length of the input element. Also, the amount of computation needed for hash functions and memory lookups are constant thereby making process faster. The designs are based on the concepts of spectral Bloom filters [2] which is an extended version of naive Bloom filter [3] and optimized version of counting Bloom filter [4]. They solve some of the problems such as multi-set query, insertion and deletion to Bloom in real-time which are not possible with original Bloom Filters. In spectral Bloom Filter [2], instead of bit vectors, an array of counters was implemented and is incremented or decremented according to corresponding insertions or deletions.

The rest of the paper is organized as follows: In Sect. 2 the background and related works are given; In Sect. 3, the design implementation of the work and description of the design; In Sect. 4, results and discussions of the experiment and Sect. 5 concludes the paper.

2 Related Works

The high speed Intrusion Detection System is an area of high opportunity, especially in hardware based NIDS. The increase in data speed has led to the requirement of dedicated hardware components and its improvement is necessary for a near perfect product. Roesch [1], developed a tool in 1999 called Snort, that can rapidly find the possible holes in network security. Sidhu and Prasanna [5] focused on methods that could convert the regular expression faster to hardware circuitry. They skipped the conversion to deterministic finite automaton (DFA), directly compiled regular expression to nondeterministic finite automaton (NFA). This implementation was extended in the works of Hutchings, Brad and Franklin [6] in 2002 which proposed a high speed Network Intrusion Detection system. The system extracted regular expression from snort with the help of java code which was then processed by the Xilinx software to create the FPGA input. They proposed an automated compiler that could convert regular expression automatically.

Bloom Filter from Bloom [3] was mainly used for checking the string and database applications. Broder and Mitzenmacher [7] uncovered the various applications of Bloom filters in networking, its modern variants, and the mathematical basis behind them. The main advantage of Bloom filter is that it takes only constant amount of memory to hash each of the elements irrespective of its length and also the computation involved in detecting an element is constant. Dharmapurikar and Krishnamurthy [8] in 2003 proposed a technique with Bloom filter to verify the membership of the queries. The focus was to implement the fast string matching in hardware based Intrusion Detection System. The design consist of bloom filters corresponding to each string length which ranges from a minimum value to size of window. In 2004 [9] they analyzed its performance against the finite automata solutions. Universal Hash function known as H3, is found to be suitable for implementation of hash table in hardware from the study conducted by Ramakrishna et al. [10].

Song and Lockwood [11] in 2005 suggested a method for long string matching to reduce the supported signature length. Three bit extended Bloom filter were chosen because of its scalability and fast incremental updating ability. Fan et al. [4] proposed an extension to bloom filter that could insert and delete from Bloom vector in real time. Cohen and Matias [2] optimized the work for multiset query and introduced two new algorithms. Pontarelli and Bianchi [12] proposed a system where instead of purely distributing the packets across the modules they grouped similar traffic packets and dispatched it to differently capable blocks. The design mainly used the header information to classify it into different categories and each module of hardware processes the disjoint rule sets. They followed a shift and compare architecture which was presented by Baker and Prasanna [13].

3 Design and Implementations

Let a string be processed by some multiple hash functions and they result in some values less than the size of the vector to which hash function map the string. Those values are set as bit positions in vectors. The query procedure is same as programming where the strings are checked for membership. The bit in corresponding values of vector is compared against the hash values of the queried string and if at least one value is found different then it is declared as a non member. The false positive probability f is given by [4]

$$f = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k} \approx \left(1 - e^{-\frac{nk}{m}}\right)^{k}$$

where m is the size of the vector with n members and k hash functions. In optimal case, for a given value of m and n, we get number of hash functions with minimum false positivity,

$$k = \left(\frac{m}{n}\ln 2\right).$$

The extended Bloom filter replaces the vector with array of counters. The counter corresponding to hash value increases when strings are inserted and decreased when deleted. When an item is queried or deleted it checks only the minimum counter value which should be greater than one if it existed in Bloom. When an item is inserted to Bloom filter, minimum counter is increased which is equivalent to increasing all the counters. The hash function implemented in this paper is the universal hash function, H3. This class of function are found suitable for hardware implementations [14] since they are bound to the limited memory resources. For any bit string X with bits represented as

$$X = <x_1, x_2, x_3, \ldots, x_b>$$

the ith hash function over X is calculated as,

$$h_i = d_{i1} \cdot x_1 \oplus d_{i2} \cdot x_2 \oplus d_{i3} \cdot x_3 \oplus \cdots \oplus d_{ib} \cdot x_b$$

where ‘.’ is a bitwise AND operator and ‘$\oplus$’ is a bitwise XOR operator. d_{ij} is a predetermined random number in the range [0 … $m - 1$].

A second bloom vector of same size known as the reference vector is introduced in this paper. The resultant hash values of multiple hash functions from previous vector are hashed together to get a single value and is set in the reference bloom vector. The reference vector is considered only when there is a chance of false positivity. The reference vector is ignored when the first vector finds the item as a non member. This can further reduce the false positivity. The hash function uses simple *XOR* of previous hash function values to increase the lookup speed.

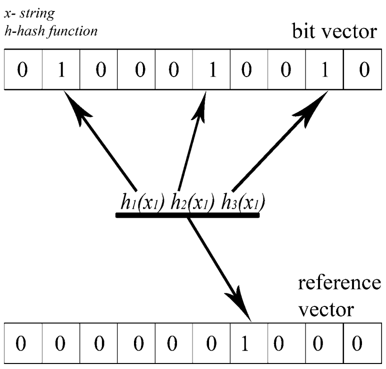


Fig. 1 Reference vector

The Fig. 1 shows the implementation of reference vector in Bloom filter and the hash function is given as

$$h_1 = h_1(x) \oplus h_2(x) \oplus \cdots \oplus h_k(x)$$

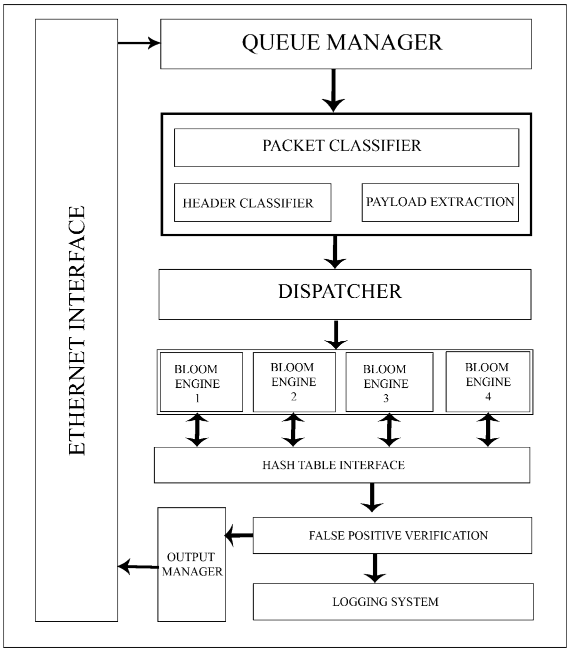
where x is the string and k is the number of hash functions in first vector.

3.1 Architectural Model

The architectural model of network intrusion detection based on reference Bloom Vector is shown in Fig. 2. It consists of an ethernet interface to which the network is connected. The queue manager record the packets and puts it in order. It allows synchronization between hardware and network. The packet classifier consists of header classifier and payload extractor. The header classifier extracts the header through the layers of protocol and the control is passed to the payload extractor. The information from payload extractor is given to the dispatcher unit which decides the distribution of the packet. It distributes the payload to one of the multiple bloom engines in such a way that the load is uniformly distributed. A Bloom engine consist of many Bloom filters, each having a different input window size. At each clock cycle, one byte is shifted in the window. The Bloom filter compares the resultant hash values of the input with the values stored in the hash table. The number of parallel bloom filters depends on the maximum length of the string to be compared and so window size is set from minimum to maximum length.

Multiple such Bloom engines are considered in order to increase the throughput of the system. These Bloom engines are connected to hash tables stored in the memory. Hash tables consist of stored hashes of attack patterns which are compared with hashed inputs in bloom filters. If the item is found to be a member, then the hash values are forwarded to False Positive Verifier. The reference vector is similar to a Bloom vector, except that the function intakes the Bloom filter hash values. The reference vector is already set with values corresponding to each string. If the

Fig. 2 Architectural model



comparison with the bloom vector gives a positive result with a certain error probability, the reference vector confirms the result, reducing the error rate. The alerting and logging systems are activated incase the output of reference vector is positive. The alerting mechanism notifies the administrator for the event and logging system saves the collected information to a file.

4 Results and Discussions

FPGA is the most feasible solution for wire-speed implementations. The design mainly focused on the minimum resource utilization in hardware. Different hardware implementable string matching algorithms were selected and analyzed, based on their complexities and features. Table 1 shows a brief comparison between

Table 1 Study of different String matching algorithms

Algorithm	Search method	Pre-processing	Search-time complexity
Brute force	Linear	No	$\mathcal{O}((n-m+1)m)$
Rabin–Karp	Hash	Yes	$\mathcal{O}((n-m+1)m)$
Morris Pratt	Heuristics based	Yes	$\mathcal{O}(m+n)$
Aho–Corasick	Automaton-based	Yes	$\mathcal{O}(m+z)$
Bloom filter	Hash	Yes	$\mathcal{O}(k)$

different hardware string matching algorithms. Bloom filter found to be the most compatible algorithm among them. It remembers only the flipping of bits in bit array and do not store any hashed keys. The selection of hash function, thus directly impacts the performance of the hardware. Five different non cryptographic hash functions were chosen for investigation. Universal Hash function, CRC 32, Pearson hash, Bit extraction hashing, XOR hashing were tested with uniformity test and avalanche test. Uniformity test analyze the distribution of hash values. Avalanche test checks whether a small change in input produce a large change in output. Considering the results from tests, H3 class of universal hash function was found better than the rest for hardware implementation.

The design is implemented in Xilinx Virtex II Pro with 4,176 Kbit block RAM [15]. Hash table were stored in SRAM with a total capacity of 4.5 Mbytes. Multiple engines with 20 distinct lengths can be scanned at a time. Despite the fact that the rate is kept at the same level as in other comparable works, the error rate is decreased to a small value. The system is designed in such a way that, the reference vector is inquired only after a positive outcome in bloom filter. The value $\frac{m}{n}$ gives the ratio between vector size and number of elements. The proper selection of m, n and k values can reduce the error probability.

The relationship between them clearly shows the size of bit vector (m) to be greater than the number of elements (n) and larger k values could reduce the errors. The Fig. 3 shows the comparison of false positive probability between old and new proposed system for different values of k with $\frac{m}{n} = 15$. The graph shows an enormous reduction in error probability making the system highly efficient.

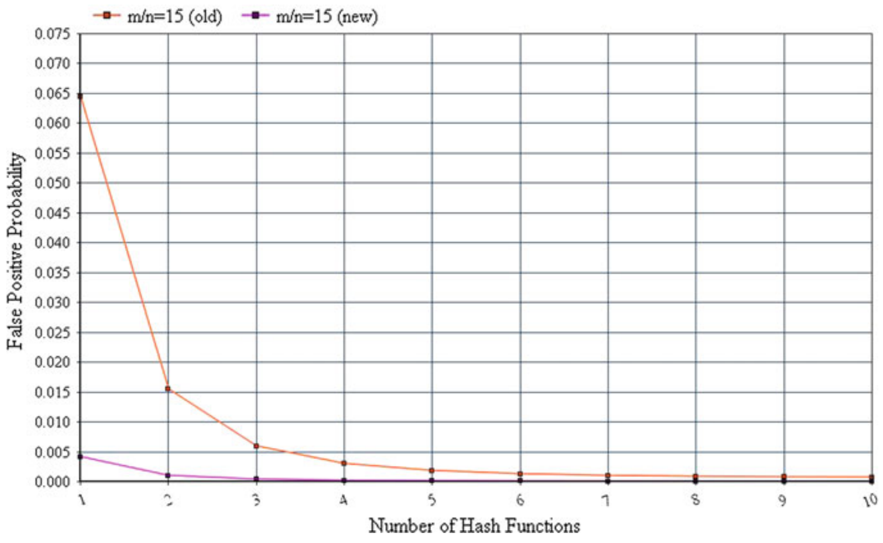


Fig. 3 Comparison of new error probability for various k values with $\frac{m}{n} = 15$

5 Conclusion

The paper proposed a network intrusion detection system in hardware platform to meet the current network requirements. The system adopted Bloom filter algorithm along with H3 hash function for the generation of bit vector. This paper presents an architecture for fast string matching with the help of multiple bloom engines. Further the design consists of a reference vector, which is meant to reduce the error rate in the system. An analysis of the trade-offs between number of hash functions and false positive probability has been presented. The FPGA based implementation is performed with the help of Xilinx Virtex Pro II FPGA board to support gigabit speed.

References

1. Roesch, Martin, et al.: Snort: lightweight intrusion detection for networks. LISA **99**, 229–238 (1999)
2. Cohen, S., Matias, Y.: Spectral bloom filters. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 241–252. ACM (2003)
3. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
4. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. In: *IEEE/ACM Transactions on Networking (TON)* 8(3):281–293 (2000)
5. Sidhu, R., Prasanna, V.K.: Fast regular expression matching using FPGAs. In: The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM'01, pp. 227–238. IEEE (2001)
6. Hutchings, B.L., Franklin, R., Carver, D.: Assisting network intrusion detection with reconfigurable hardware. In: *Proceedings 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. pp. 111–120. IEEE (2002)
7. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: a survey. Internet Math **1**(4), 485–509 (2004)
8. Dharmapurikar, S., Krishnamurthy, P., Sproull, T., Lockwood, J.: Deep packet inspection using parallel bloom filters. In: *Proceedings of 11th Symposium on High Performance Interconnects*. pp. 44–51. IEEE (2003)
9. Dharmapurikar, S., Attig, M., Lockwood, J.: Design and implementation of a string matching system for network intrusion detection using FPGA-based bloom filters. In: IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM04) (2004)
10. Ramakrishna, M., Fu, E., Bahcekapili, E.: A performance study of hashing functions for hardware applications. In: Proceedings of International Conference on Computing and Information, pp. 1621–1636 (1994)
11. Song, H., Lockwood, J.W.: Efficient packet classification for network intrusion detection using FPGA. In: *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pp. 238–245. ACM (2005)
12. Pontarelli, S., Bianchi, G., Teofili, S.: Traffic-aware design of a high-speed FPGA network intrusion detection system. Trans. Comput. IEEE **62**(11), 2322–2334 (2013)
13. Baker, Z.K., Prasanna, V.K.: Automatic synthesis of efficient intrusion detection systems on FPGAs. In: Field Programmable Logic and Application, pp. 311–321. Springer, Berlin (2004)

14. Hua, N., Norige, E., Kumar, S., Lynch, B.: Non-crypto hardware hash functions for high performance networking ASICs. In: *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, pp. 156–166. IEEE Computer Society (2011)
15. Xilinx Inc. Virtex-II Pro and Virtex-II Pro X platform FPGAs: Complete data sheet (2004)