International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 3, No. 1, March 2014, pp. 31~38
ISSN: 2089-4864

FPGA Based Firewall using Embedded Processor for Vulnarability Packet Detection

Mohamed Yousuf Hasan, Poornima V.P, Sujendran S, Karthikraja D
Department of Electronics and Communication Engineering, C. Abdul Hakeem College of Engineering and Technology, Vellore, India

Article Info

Article history:

Received Jan 6, 2014
Revised Feb 15, 2014
Accepted Feb 28, 2014

Keyword:

Algorithmic attacks
Embedded system
Memory gap
Network security
Virus detection

ABSTRACT

This paper describes the design of high performance packet filtering firewall using embedded system. An FPGA (field programmable gate array) platform has been used for implementation and analysing the network firewall. It is capable of accepting real time changes. This network security application has an ability to perform powerful protection against unwanted data packets such as virus attack, spam in e-mails, hackers, worms, spyware unauthorized contents. However the firewalls don't address the difficulty of unwanted data packets intrusion. The ultimate aim of this work is to create a systematic way of approach for unwanted packets discard in a network system. We use a specially trained algorithms such as Wu-manber algorithms (high performance, multi-pattern matching), bloom filter algorithm (space efficient data structure for testing an element in the set.Our design is mainly based on machine learning and artificial intelligence. This gives a high efficiency, improved performance and high ability of packet detection with less contribution of time in an effective way.

Corresponding Author:

Poornima.V.P
Department of Electronics and Communication Engineering,
C. Abdul Hakeem College of Engineering and Technology,
Vellore, India.
Email: poornima2810@gmail.com

1. INTRODUCTION

NETWORK security is very important for every end users to protect their pc against virus, spyware, worms. They may visit malicious websites or hackers may gain entry to their computers and use them as zombie computers to attack others.In order to over come these flaws firewall was introduced by developers. This firewall reduces the rate of external attacks successfully. How ever for latest methodologies firewall fails to protect (i.e., firewall codes are easily bared by using zombic computers. So to make a FPGA (field programmable gate array) based processor on Wu-Manber algorithm [3]. It is a modified version of Boyer-Moore algorithm [4] which is used to search for multiple patterns. And bloom filter algorithm [5] determines wheather an element is a non- member of a set using hash functions and bit vector and this is a space efficient algorithm.

In order to improve speed of this process and to accept real time changes there are six modes of operation [2]. Here in addition to processor memory content addressable memory (CAM) is used to increase the speed of packet matching.The whole device is implemented on an FPGA device.

Our approach is to make an FPGA (field programmable gate array)based embedded processor for vulnarabile packets such as virus attack, spam in e-mails, hackers, worms, spyware unauthorized contents.our work is to do the processor in a cost efficient manner with improved performance and an ability to generate extreme protection to the system.

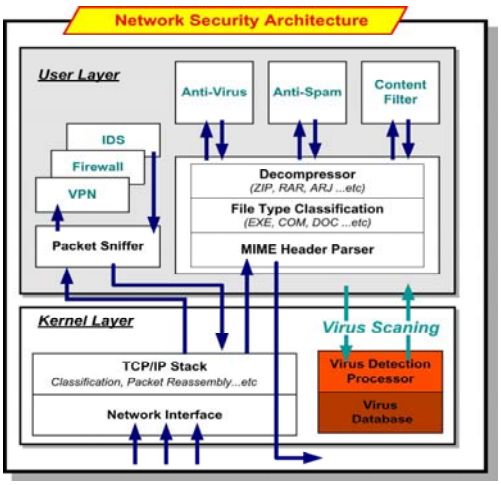

Figure 1. Network security architecture

2. LITRATURE SURVEY

The purpose of pattern matching is to check whether a textcontains at least one of a given set of patterns. There are many algorithms [3]-[5] and accompanying hardware accelerators for fast pattern matching. Its performance is not affected by the size of a given pattern set (the sum of all pattern lengths), but it requires asignificant amount of memory due to state explosion. In 2005, Lin Tan introduced a bit-split method [6] by splitting an 8-bit character into four 2-bit characters to construct the automaton. Their state machines are smaller than the original, and they have fewer fan-out states for each transaction. However, the bit-split method reads several memory blocks in parallel when matching patterns. Thus, it can only be implemented by on-chip memory because of its high memory read port requirements. Piti Piyachon and Yan Luo extended this concept [7] to the Intel IXP2855 network processor. In contrast, heuristic approaches are based on the Boyer-Moore algorithm [4], which was introduced in 1977. Its key feature is the shift value, which shifts the algorithm's search window for multiple characters when it encounters a mismatch. The search window is a range of text exactly fetched by pattern matching algorithms for each examination. This algorithm performs better because it makes fewer comparisons than the naïve pattern-matching algorithm. At runtime, the Boyer-Moore algorithm uses a pattern pointer to locate a candidate position by assuming that a desired pattern exists at this position. The algorithm then shifts its search window to the right of this pattern. By default, desired patterns can exist in any position of a text; therefore, all positions in a text are candidate positions and must be examined. If the string of search windows does not appear in the pattern, the algorithm can shift the pattern pointer to the right and skip multiple characters from the candidate position to the end of the pattern without making comparisons. Based on this concept, Wu and Manber (WM) [3] modified the Boyer-Moore algorithm [4] to search for multiple patterns. The WM algorithm is widely used in many applications, including Unix tools such as agrep and glimpse. However, the performance of both of these algorithms is bounded by the pattern length.Software-based Bloom filters [5] were first described in 1970. These filters can determine whether an element is a non-member of a given set in a constant amount of time using several hash functions and a bit vector. The Bloom filter method is exceptionally space-efficient. In a typical case, the filter rate for 30 000 patterns reaches 90% and requires only 34.76kB of memory. Although the Bloom filter rejects a non-member in constant time, it does not guarantee that an element is in a given set. False-positive problems necessitate a secondary method to verify the match. In brief, the Bloom filter does not perform pattern matching individually, except with an exact-matching method.

In 2004, Sarang et al. [4] presented a pattern-matching processor based on Bloom filters. They used multiple Bloom filters to check different-length prefixes of the pattern in parallel. This design needs 32 memory read ports because it uses 32 hash functions. However, most commonly used memory modules only have two ports: a read port and a write port. To lower the memory read port requirements, they divided a bit vector into several smaller vectors implemented by 140-block RAM of FPGA. The total memory size, then, is 70kB for 10 038 patterns. The design also includes an analyzer that isolates falsepositives. The performance of this design can reach 2.46Gb/s.Some design [7]-[10] take advantage of field-programmable gate array's (FPGA's) ability to be reconfigured to improve performance. Some of these designs are even based on non-deterministic finite automata (NFA) to handle complex regular expressions. These methods provide high throughput, but the maximum number of patterns they support is limited by the FPGA comparators. The Xilinx Virtex2-8000 FPGAs only support about 781 ClamAV rules.

Sourdi *et al.* implemented a perfect hashing function [10] on an FPGA to remove redundant memory accesses caused by address collisions. Some designs [11], [12] have used content-addressable memory (CAM) to improve engine filtering rates and to store the entire pattern database in a large external memory.We re-encode the shift table and Bloom filter to merge them into the same space, the shift-signature table. The new table not only maintains the shift value of properties but also avoids reducing the filter rate for a large scale pattern set. In the same space, 32 kB SRAM in this case, the filter rate of our approach is improved from 10.9% to 6.9% compared to the Bloom filter.

The remaining of this paper is as follows. Section II shows the literature survey. Section III shows the details of detail of machine learning algorithm. Section IV describes the details of the network processor. In Section V shows the simulation and results and present our conclusion in section VI.

3. MACHINE LEARNING ALGORITHM

In this section we describe in detail about Wu-Manber and Bloom Filter algorithm.

3.1. Wu-Manber Algorithm

The Wu-Manber algorithm is a high-performance, multi-pattern matching algorithm based on the Boyer-Moore algorithm.It builds three tables in the pre-processing stage: a shift table, a hash table and a prefix table. The Wu-Manber algorithm is an exact-matching algorithm, but its shift table is an efficient filtering structure. The shift table is an extension of the bad-character concept in the Boyer-Moore algorithm, but they are not identical. The matching flow is shown in (a). The matching flow matches patterns from the tail of the minimum pattern in the pattern set, and it takes a block B of characters from the text instead of taking them one-by-one. The shift table gives a shift value that skips several characters without comparing after a mismatch. After the shift table finds a candidate position, the Wu-Manber algorithm enters the exact-matching phase and is accelerated by the hash table and the prefix table. Therefore, its best performance is 0(BN/*m*)for the given text with length and the pattern set, which has a minimum length of *m*. The performance of the Wu-Manber algorithm is not proportional to the size of the pattern set directly, but it is strongly dependent on the minimum length of the pattern in the pattern set. The minimum length of the pattern dominates the maximum shift distance (*m*-B+1) in its shift table. However, the Wu-Manber algorithm is still one of the algorithms with the best performance in the average case.For the pattern set {erst, ever, there} shown in (d), the maximum shift value is three characters for and B=2 and *m*=4 .The related shift table,

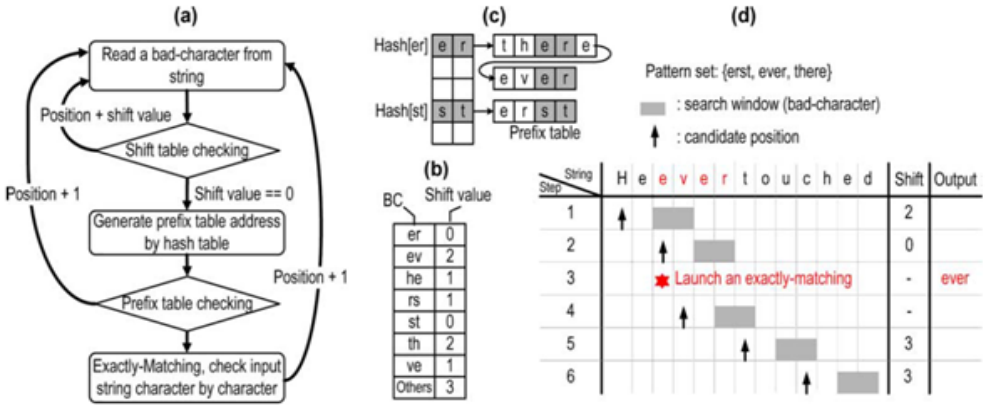

Figure 2. Wu-Manber Algorithm

Wu-Manber matching process. (a) Matching flow, (b) shift table, (c) hash table+prefix table, (d) matching process. Hash table and prefix are shown in (b) and (c). The Wu-Manber algorithm scans patterns from the head of a text, but it compares the tails of the shortest patterns. In step 1, the arrow indicates to a candidate position that a wanted pattern probably exists, but the search window (gray bar) is actually the character it fetches for comparison. According to shift[ev]=2, the arrow and search window are shifted right by two characters. Then, theWu-Manber algorithm finds a candidate position in step 2 due to shift[er]=0. Consequently, it checks the prefix table and hash table to perform an exact-matching and then outputs the “ever” in step 3. After completing the exact match, theWu-Manber algorithm returns to the shifting phase, and it shifts the search window to the right by one character to find the next candidate position in step 4. The algorithm keeps shifting the search window until touching the end of the string in step 6.

```
--*//*WU MANBER ALGORITHM.
if inp_pattern(j to j+1)=temp1(1 to 2) then
shf_out <= shf_val(1);
if shf_out=0 then
j <= j + 1;
present_seq <= inp_pattern(j to j+1);
next_seq <= inp_pattern(j+1 to j+2);
end if;
elsif inp_pattern(j to j+1)=temp2(1 to 2) then
shf_out <= shf_val(2);
--if ev_data(1 to 2)=" " then
if shf_out=2 then
j <= j + 2;
present_seq <= inp_pattern(j to j+1);
next_seq <= inp_pattern(j+2 to j+3);
th := er_data(2);
if inp_pattern(j to j+3)=th(1 to 4) then
output <= inp_pattern(j to j+3);
end if;
end if;
elsif inp_pattern(j to j+1)=temp3(1 to 2) then
shf_out <= shf_val(3);
--if he_data(1 to 2)=" " then
if shf_out=1 then
j <= j + 1;
present_seq <= inp_pattern(j to j+1);
next_seq <= inp_pattern(j+1 to j+2);
end if;
```

3.2. Bloom Filter Algorithm

A Bloom filter is a space-efficient data structure used to test whether an element exists in a given set. This algorithm is composed of different hash functions and a long vector of bits.Initially, all bits are set to 0 at the preprocessing stage. To add an element, the Bloom filter hashes the element by these hash functions and gets positions of its vector. The Bloom filter then sets the bits at these positions to 1. The value of a vector that only contains an element is called the signature of an element. To check the membership of a particular element, the Bloom filter hashes this element by the same hash functions at run time, and it also generates positions of the vector. If all of these bits are set to 1, this query is claimed to be positive, otherwise it is claimed to be negative. The output of the Bloom filter can be a false positive but never a false negative. Therefore, some pattern matching algorithms based on the Bloom filter must operate with an extra exact-matching algorithm. However, the Bloom filter still features the following advantages: 1) it is a space-efficient data structure; 2) the computing time of the Bloom filter is scaled linearly with the number of patterns; and 3) the Bloom filter is:

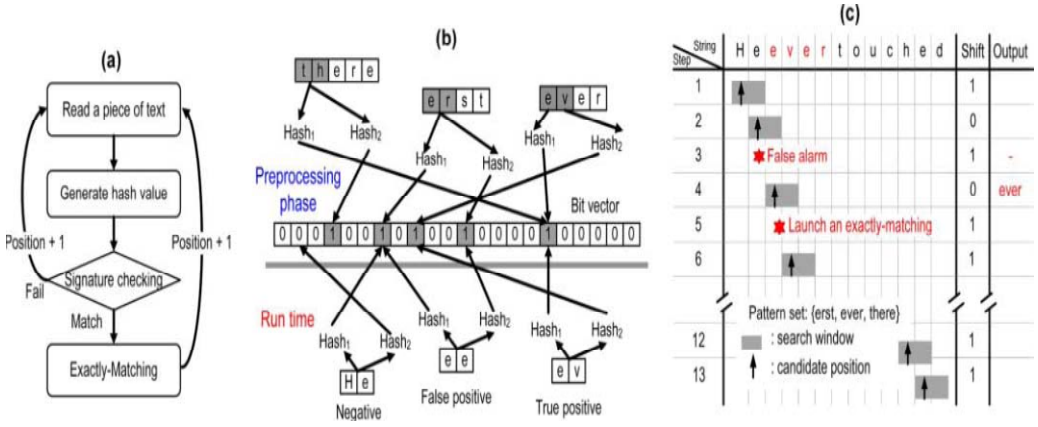

Figure 3. Bloom filter

Bloom filter matching process. (a) Matching flow; (b) bit-vector building; (c) matching process independent of its pattern length (a) describes a typical flow of pattern matching by Bloom filters. This algorithm fetches the prefix of a pattern from the text and hashes it to generate a signature. Then, this algorithm checks whether the signature exists in the bit vector.If the answer is yes, it shifts the search window to the right by one character for each comparison and repeats the above step to filter out safe data until it finds a candidate position and launches exact-matching (b) shows how a Bloom filter builds its bit vector for a pattern set {erst, ever, there} for two given hash functions. The filter only hashes all of the pattern prefixes at the preprocessing stage. Multiple patterns setting the same position of the bit vector are allowed.

```
--*//* BLOOM FILTER ALGORITHM.
if inp_pattern(j to j+1)="he" then
h1_h2 := h_t(2) and h_t(7);
if h1_h2='0' then
shf_out <= 1;
j <= j + 1;
present_seq <= inp_pattern(j to j+1);
next_seq <= inp_pattern(j+1 to j+2);
end if;
elsif inp_pattern(j to j+1)="ee" then
h1_h2 := h_t(12) and h_t(7);
if h1_h2='1' then
shf_out <= 0;
j <= j + 1;
present_seq <= inp_pattern(j to j+1);
next_seq <= inp_pattern(j+1 to j+2);
end if;
elsif inp_pattern(j to j+1)="ev" then
h1_h2 := h_t(9) and h_t(17);
if h1_h2='1' then
shf_out <= 0;
j <= j + 1;
present_seq <= inp_pattern(j to j+1);
next_seq <= inp_pattern(j+1 to j+2);
if inp_pattern(j to j+3)=p2(1 to 4) then
output <= inp_pattern(j to j+3);
end if;
end if;
else
h1_h2 := '0';
shf_out <= 1;
j <= j + 1;
present_seq <= inp_pattern(j to j+1);
next_seq <= inp_pattern(j+1 to j+2);
end if;
h1h2 <= h1_h2;
end if;
```

(c) shows an example of the matching process. The arrows indicate the candidate positions. The gray bars represent the search window that the Bloom filter actually fetches for comparison. Both the candidate position and search window are aligned together.Thus, the Bloom filter scans and compares patterns from the head rather than the tail, like the Wu-Manber algorithm. In step 1, the filter hashes “He” and mismatches the signature with the bit vector.

The filter then shifts right 1 character and finds the next candidate position. For the search window “ee”, the Bloom filter matches the signature and then causes a false alarm to perform an exact-matching in steps 2 and 3. The filter then returns to the filtering stage and shifts one character to the right in step 4, which launches a true alarm for the pattern “ever”.Finally, the Bloom filter filters the rest of text and finds nothing.

4. NETWORK PROCESSOR

The overall layout of the embedded network firewall design is as shown Figure 4. As shown in the block diagram, the main modules in the embedded network firewall are Nios II 32 bit microprocessor module, Ethernet module, Content Addressable Memory (CAM) module, Netmask RAM module, Arbiter module and Network Firewall module (NFM). All of these modules tightly work together to achieve a powerful, flexible and easy to configure packet filtering firewall.At the beginning, when the system is powered up, the firewall module is in inactive mode as the Nios II has not been programmed yet to run any code or OS. Nios II Integrated Development Environment (IDE) development tool is used to upload essential applications including RTOS to run on the Nios II. The software initializes all of the modules in the design according to the selected mode of operation. When the Ethernet module finds any TCP/IP traffic targeted toward the Nios II, it interrupts the NFM. In the next step, the NFM extracts the necessary field (based on the initial operation mode) off the Ethernet frame and inquires the permission from the CAM module. If the CAM module finds the selected field in its memory block (match found), it gives the pass permission to the NFM and the NFM interrupts the Nios II for a receiving packet. In the case of an invalid packet, the NFM drops the packet and waits for the next interrupt from the Ethernet. After each Ethernet interrupt, the packet status is reported to the Nios II by the NFM for monitoring purposes.

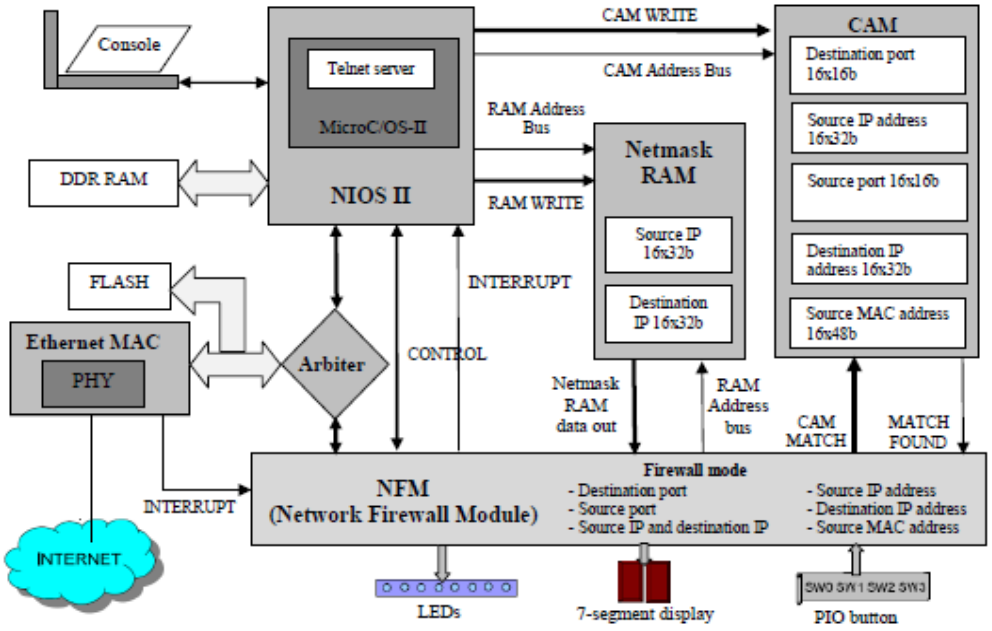

Figure 4. Embedded network firewall block diagram

The firewall operation mode and configuration can be changed any time on the fly by Telnetting to the Telnet server running on the Nios II.

5. SIMULATION AND RESULTS

Analyzes two scenarios of malicious attacks and provides two methods for keeping performance within a reasonable range. First, re-encoded the shift table to make it provide a bad-character heuristic feature and high filter rates for large pattern sets at the same time. Second, the proposed skip mechanism cushions the below to performance under algorithmic attacks. The proposed re-en-coding stage removes 95.42% of the exact-matching events in the front-end for normal cases, and the skip mechanism eliminates 95.8% of the external memory accesses for attack cases with just 32kB internal memory and 8 MB external memory.

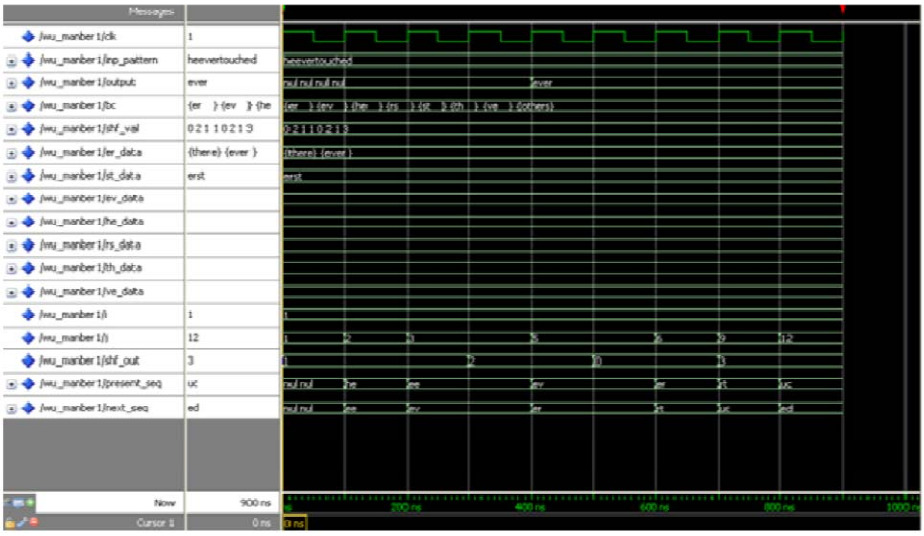

Figure 5. Filtering Engine: Wu Manber Algorithm

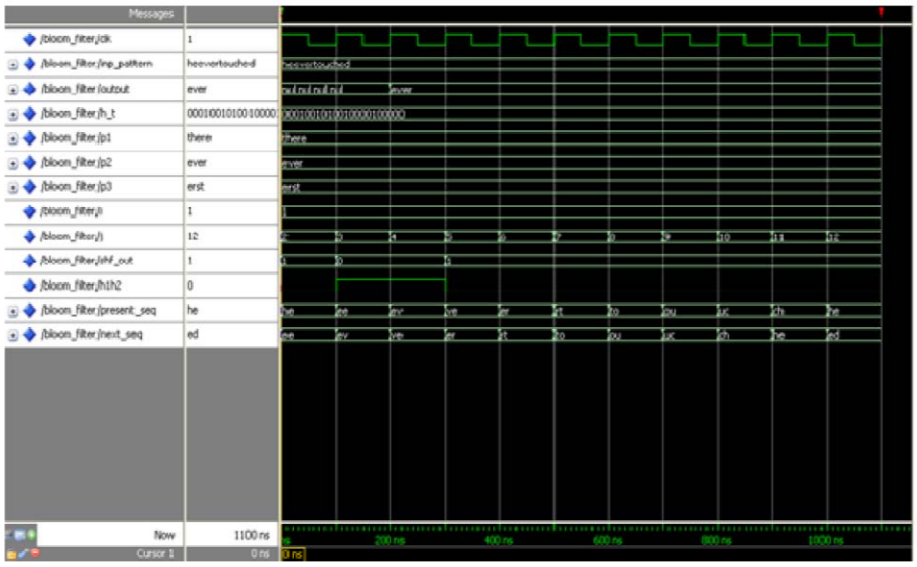

Figure 6. Filtering Engine: Bloom Filter Algorithm

6. CONCLUSION

Many previous designs have claimed to provide high performance, but the memory gap created by using external memory decreases performance because of the increasing size of virus databases. Furthermore, limited resources restrict the practicality of these algorithms for embedded network security systems. Two-phase heuristic algorithms are a solution with a trade-off between performance and cost due to an efficient filter table existing in internal memory however, their performance is easily threatened by malicious attacks.

REFERENCES

[1] A Scalable High-Performance Virus Detection Processor Against a Large Pattern Set for Embedded Network Security. Chieh-Jen Cheng, Chao-Ching Wang, Wei-Chun Ku, Tien-Fu Chen, Jinn-Shyan Wang. *IEEE transactions on very large scale integration systems*. 2012; 20(5).

[2] Embeded Network Firewall on FPGA. Raouf Ajami, Anh Dinh Department Of Electrical and Computer Engineering University Of Saskatchewan. Canada.

[3] S Wu, U Manber. A fast algorithm for multi-pattern searching. Univ. Arizona, Tucson, Report TR-94-17. 1994.

[4] RS Boyer, JS Moore. A fast string searching algorithm. *Commun. ACM*. 1977; 20: 762–772.
[5] BH Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*. 1970; 13: 422–426.
[6] L Tan, T Sherwood. *A high throughput string matching architecture for intrusion detection and prevention*. Proc. 32nd Annu. Int.Symp. Comput. Arch., 2005: 112–122.
[7] S Dharmapurikar, P Krishnamurthy, TS Sproull. Deep packet inspection using parallel bloom filters. *IEEE Micro*. 2004; 24(1): 52–61.
[8] G Memik, SO Memik, WH Mangione-Smith. *Design and analysis of a layer seven network processor accelerator using reconfigurable logic.* Proc. 10th Annu. IEEE Symp. Field-Program. Custom Comput. Mach., 2002; 131–140.
[9] R Sidhu, VK Prasanna. *Fast regular expression matching using FPGAs.* Proc. 9th Annu. IEEE Symp. Field-Program. Custom Comput. Mach. 2001: 227–238.
[10] I Sourdis, D Pnevmatikatos, S Wong, S Vassiliadis. *A reconfigurable perfect-hashing scheme for packet inspection.* Proc. 15th Int. Conf. Field Program. Logic Appl., 2005; 644–647.
[11] F Yu, RH Katz, TV Lakshman. *Gigabit rate packet patternmatching using TCAM.* Proc. 12th IEEE Int. Conf. Netw. Protocols. 2004; 174–183.
[12] I Sourdis, D Pnevmatikatos. *Pre-decoded CAMs for efficient and high-speed NIDS pattern matching.* Proc. 12th Annu. IEEE Symp.Field-Program. Custom Comput. Mach., 2004; 258–267.

BIOPGRAPHIES OF AUTHORS

V. Mohamed Yousuf Hasan received his B.E. in Electronics and Communication Engineering and M.E. in Applied Electronics from Anna University in the year 2005, and 2008 respectively. He is working as a Assistant Professor (2005 - Till now) in the Department of Electronics and Communication Engineering at C. Abdul Hakeem College of Engineering and Technology, Vellore, India. His current research interests include NOC design, Firewall On Chip design, Deep Content Filtering He is a member of various professional societies like ISTE, IAENG, IACSIT etc.,

Poornima V. P. was born in Banglore, in1989. She received the B.E degrees in Electronics And Communication Engineering from Adhiparasakthi College Of Engg-Kalavai, in 2011. Currently doing M.E from C.Abdul Hakeem College Of Engineering, Melvisharam.

Sujendran S. He is the student of B.E Electronics And Communication Engineering Dept from from C.Abdul Hakeem College Of Engineering, Melvisharam.

Karthikraja D. He is the student of B.E Electronics And Communication Engineering Dept from from C.Abdul Hakeem College Of Engg, Melvisharam.