International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869, Volume-1, Issue-10, December 2013

Virus Detection Processor using Pipeline Architecture for Embedded Network Security

P.Kokila, Dr.S.Kavitha

Abstract— **Network security has always been an important issue for data communication. End users are vulnerable to virus attacks, spam, and Trojan horses. Contemporary network security applications generally require the ability to perform powerful pattern matching to protect against attacks such as viruses and spam.**

Traditional hardware solutions are intended for firewall routers. However, the solutions in the literature for firewalls are not scalable, and they do not address the difficulty of an antivirus with an ever-larger pattern set. Because the firewall routers only focus on the application layer of the OSI model and they require high transmission speed. To achieve the high speed transmission, a pattern matching processor (also called as virus detection processor) is developed.

The aim of the work is to provide a systematic virus detection hardware solution for network security for embedded systems. Instead of placing entire matching patterns on a chip, the solution is a two-phase dictionary-based antivirus processor that works by condensing as much of the important filtering information as possible onto a chip and infrequently accessing off-chip data to make the matching mechanism scalable to large pattern sets.

In the first stage, the filtering engine can filter out the data using a merged shift table and in the second stage the unsafe datas can be filter out by the exact-matching engine. The filtering engine has two algorithms: one is Wu-manber and another one is Bloom filter. By using these two algorithms filter rate can be increased and reduces the memory gap.

Index Terms— **Virus, Pipeline, Network security, Memory gap and Embedded System**

I. Introduction

Network-on-Chip (NoC) is an approach to design the communication subsystem between IP cores in a System-on-chip (SoC). SoC refers to integrate all components of a computer or other electronic system into a single IC and IP core is a reusable unit of logic, cell, or chip layout design. The main areas related to NoC are security, routing and quality of service.

In the current scenario network security is the main cause in NoC. The issues like network intrusion, spam, viruses, etc. are a threat for the security of the networks in the internet.

A. Worm

A WORM is a program or algorithm that replicates itself over a computer network and usually performs malicious actions, such as using up the computer's resources and possibly shutting the system down. It does not need to attach itself to an existing program. Worms almost always cause at least some harm to the network, even if only by consuming bandwidth.

Worms spread by exploiting vulnerabilities in operating systems. Vendors with security problems supply regular security updates and if these are installed to a machine then the majority of worms are unable to spread to it. If vulnerability is disclosed before the security patch released by the vendor, a zero-day attack is possible.

Users need to be wary of opening unexpected email and should not run attached files or programs, or visit web sites that are linked to such emails. However some of the worms are increased with the growth and efficiency of phishing attacks, it remains possible to trick the end-user into running malicious code. Anti-virus and anti-spyware software are helpful, but must be kept up-to-date with new pattern files at least every few days.

B. Virus

A computer virus is a computer program that can replicate itself and spread from one computer to another. The term "virus" is also commonly used to refer other types of malware but not limited to adware and spyware programs which do not have a reproductive ability. Viruses are sometimes confused with worms and Trojan horses, which are technically different. Viruses may harm a computer system's data or performance. Some viruses and other malware have symptoms noticeable to the computer user. Some viruses do nothing beyond reproducing themselves. The code for a virus is hidden within an existing program and when that program is launched, the virus inserts copies of itself into other programs on the system to infect them.

C. C. Firewall router

When a new connection is established, the firewall router scans the connection and forwards these packets to the host after confirming that the connection is secure. Because firewall routers focus on the application layer of the OSI model, they must reassemble incoming packets to restore the original connection and examine them through different application parsers to guarantee a secure network environment. For instance, suppose a user searches for information on web pages and then tries to download a compressed file from a web server. In this case, the firewall router might initially deny some connections from the firewall based on the target's IP address and the connection port. Then, the firewall router would monitor the content of the web pages to prevent the user from accessing any page that

Manuscript received December 17, 2013.

P.Kokila, M.E. Student, Embedded System Technologies, Nandha Engineering College, Erode, TamilNadu, India.

Dr.S.Kavitha, Professor, Nandha Engineering College, Erode, TamilNadu, India.

connects to malware links or inappropriate content, based on content filters. When the user wants to download a compressed file, to ensure that the file is not infected, the firewall router must decompress this file and check it using anti-virus programs.

In summary, firewall routers require several time-consuming steps to provide a secure connection. However, even under numerous security constrictions, firewall routers are still required to provide high-speed transmission. Fortunately, most security-guaranteed programs use rule-based designs. Therefore, in this project it is tried to develop a pattern matching processor to accelerate the detection speed. It is called as virus detection processor because the database size it supports has reached the antivirus software level and is far greater than those of previous works.

II. EXISTING METHOD – A LITERATURE SURVEY

A. *Automata-based architecture*

Most automata-based approaches are based on the algorithm proposed by Aho and Corasick in 1975 called AC algorithm which describes a linear-time algorithm for multi-pattern search with a large finite-state machine. Aho and Corasick proposed an algorithm for concurrently matching multiple strings. Aho–Corasick (AC) algorithm used the structure of a finite automation that accepts all strings in the set. The automation processes the input characters individually and tracks partially matching patterns. The AC algorithm has proven linear performance, making it suitable for searching a large set of rule. Two different implementations exist for Snort, implemented by Mike Fisk and Marc Norton, respectively. This work tested both implementations and employed the latter in the present experiments because of its superior performance. However, the Norton implementation requires considerably more. Its performance is not affected by the size of a given pattern set (the sum of all pattern lengths), but it requires a significant amount of memory due to state explosion

B. *Filtering based architecture*

Filtering based approach is also known as heuristic based approach which is mainly based on Boyer-Moore algorithm and Bloom filters. The Boyer–Moore algorithm [Boyer and Moore 1977] is widely used because of its efficiency in single-pattern matching problems. This algorithm uses two heuristics to reduce the number of character comparisons required for pattern matching. Both heuristics are triggered by mismatches. The first heuristic is commonly referred to as the bad character heuristic, works as follows: if the search pattern contains the mismatching character, the pattern is shifted so that the mismatching character is aligned with the rightmost position at which it appears in the pattern. Meanwhile, if the mismatching character does not appear in the search pattern, the pattern is shifted so that the first character in the pattern is one position later than that of the mismatching character in the given text. The second heuristic, commonly referred to as the good suffixes heuristic, works as follows: if a mismatch is found in the middle of the pattern, the search pattern is shifted to the next occurrence of the suffix in the pattern. The Boyer–Moore algorithm was designed for searching for a single pattern from a given text and performs well in this role. However, the current implementation of Boyer–Moore in Snort is not efficient in seeking multiple patterns from given payloads.

In Sarang et al. (2004) 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 70 kB for 10038 patterns. The design also includes an analyzer that isolates false positives. The performance of this design can reach 2.46 Gb/s.

C. *Bit Split method*

In 2005, Lin Tan introduced a bit-split method 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 to the Intel IXP2855 network processor. For increasingly large pattern sets, an IBM team implemented an optimized AC algorithm on the cell processor, and they discovered that the memory gap was the bottleneck. As a result, they modified the algorithm and used DMA to reduce the effect on the memory system.

Some designs take advantage of field-programmable gate array's (FPGA's) ability to be reconfigured to improve performance. Some of the 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. In 2004, to support an unlimited pattern count, Cho presented the idea of a two-phase architecture that implements a front-end filter with an FPGA and stores its full pattern database in a large memory.

Later, Sourdi implemented a perfect hashing function on an FPGA to remove redundant memory accesses caused by address collisions. Some designs have used content-addressable memory (CAM) to improve engine filtering rates and to store the entire pattern database in a large external memory. Since then, pattern-matching designs have tended to use a two-phase architecture, in which one phase finds suspicious positions and the other phase precisely identifies patterns.

All of these designs provide more than 1-Gb/s performance, and some support even more than 10 Gb/s. However, with increasing pattern sets, it becomes more difficult to implement these designs in on-chip memory or dedicated circuits. Some designs attempt to store pattern sets in external memory, which is typically implemented by SDRAM or DDR, for their space requirements. Although DRAM technology has greatly improved over the last few decades, DRAM-based memories still require initial cycles before pumping out their first non-consecutive data. The gain only appears in consecutive readings of various sectors. A non-consecutive read operation of DDR memory still typically costs 25 - 40 ns, compared to the 1 - 3 ns working cycle of existing processors. Unfortunately, most pattern matching designs have irregular access to their memories. Thus, even though the kernels of these works are well designed, their performances are slowed dramatically by these long memory access processes especially for filtering-based designs. For this reason, improving the filter rate and overlapping the access time are the two major trends. Some designs try to use caches to overlap the access time.

III. PROPOSED METHOD

A. *Two phase architecture*

The work has focused on algorithms and has even developed specialized circuits to increase the scanning speed. However, these works have not considered the interactions between algorithms and memory hierarchy. Because the number of attacks is increasing, pattern-matching processors require external memory to support an unlimited pattern set. This method makes the memory system the bottleneck. However, when the pattern set is already intractably large, a perfect solution is unattainable. A more realistic goal is to provide high performance in most cases while still performing reasonably well in the worst case. With an eye toward high performance, updatability, unlimited pattern sets and low memory requirements, it is to present a two-phase architecture (Filtering Engine and Exact Matching Engine) that uses off-chip memory to support a large pattern set.

The two-phase pattern-matching architecture mostly comprising the filtering engine and the exact-matching engine. The filtering engine is a front-end module responsible for filtering out secure data efficiently and indicating to candidate positions that patterns possibly exist at the first stage. The overall performance depends on the filtering engine. It has separate memory for storing significant information. Here 32KB of on-chip memory is sed for the ClamAV virus database, which contained more than 30,000 virus codes.

Initially, a pattern pointer is assigned to point to the start of the given text at the filtering stage. Suppose the pattern matching processor examines the text from left to right. The filtering engine fetches a piece of text from the text buffer according to the pattern pointer and checks it by a shift-signature table. If the position indicated by the pattern pointer is not a candidate position, then the filtering engine skips this piece of text and shifts the pattern pointer right multiple characters to continue to check the next position. Filtering engine has two classical filtering algorithms for pattern matching to improve the filter rate.

1. Wu-Manber Algorithm
2. Bloom Filter Algorithm

If both layers are missing their filter, the processor enters the exact-matching phase. After an alarm caused by the filtering engine, the exact-matching engine precisely verifies this alarm by retrieving a trie structure. This structure divides a pattern into multiple sub-patterns and systematically verifies it. Only a few unsaved data need to be checked precisely by the exact-matching engine in the second stage.

IV. VIRUS SIGNATURE AND DETECTION TECHNIQUES

A. *Pattern Sets*

A set of patterns, mostly used to include or exclude certain files. The individual patterns support if and unless attributes to specify that the element should only be used if or unless a given condition is met. In the given set of k patterns, {P1, P2… Pk}, If k=1, then it is a single-pattern matching problem. When k>1, we have a multiple-pattern matching problem. In this paper, k is usually a large number (e.g., thousands). Given a packet of length n, the goal is to report all the matching patterns in the packet. The two major types of attack in the pattern sets are typical and extreme case attacks. The two types of typical attacks are 1. Deep search attack and 2. Sub-pattern attack.

B. *Deep Search Attack*

The Figure 4.1 shows the deep search attacks which is the most common and the easiest to build. This type of attack's text is constructed by patterns in the given pattern set, and therefore, these texts frequently cause the filtering engine to launch alarms and require verification of the exact-matching engine. Also it illustrates a typical attack text "barrow" built by the pattern set {bar, row, ed}.

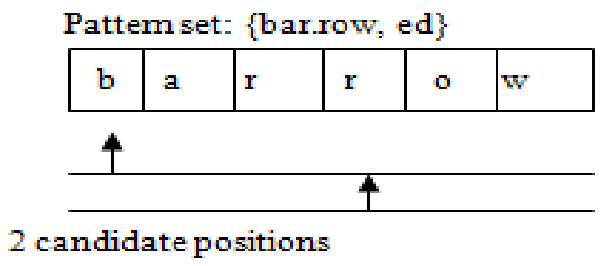


Figure 4.1 Deep Search attack

candidate positions cause exact-matching to take a long time to traverse its data structure. For the ClamAV, the average pattern length is 64 characters; thus, attack texts constructed with this method have a 1/64 chance of launching an exact match for each examination.

C. *Sub-Pattern Attack*

The Figure 4.2 shows the sub-pattern attacks which is relatively rare, but it provides a higher-density attack text than the first. If the given pattern set includes a pattern that contains the prefix of another pattern, the attack texts built by these patterns provide more candidate positions in the same length. The patterns "heat" and "eat" in Figure 4.2 are an example. For the same length, the text launches many more alarms than the text in Figure 4.1

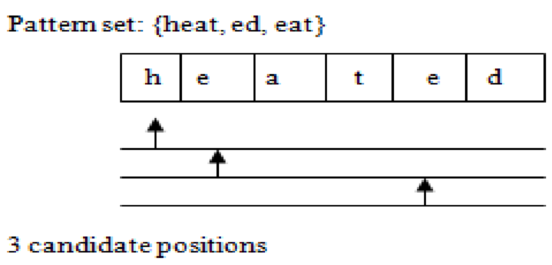


Figure 4.2 Sub-Pattern attack

D. *Extreme Case Attack*

The Figure 4.3 shows an extreme case that combines the first and second types: a pattern is constituted by a serial of characters “a”; the constitution of the attack text is also same as the pattern but is longer. As a result, the attack text in figure can be considered to constitute a multiple of this pattern. In addition, this pattern contains the prefix of itself. Two of these patterns can compose the third. Therefore, this attack text can be considered to be a special case that combines by the first and second types and causes exact-matching for every position of itself. This extreme case dramatically lowers the performance. Although it can be avoided by choosing the pattern set well, the first type of attack text can still be built easily if the pattern sets are known by the attackers. Thus, a two-phase architecture is vulnerable to algorithmic attacks

. Pattern Set: {aaa}

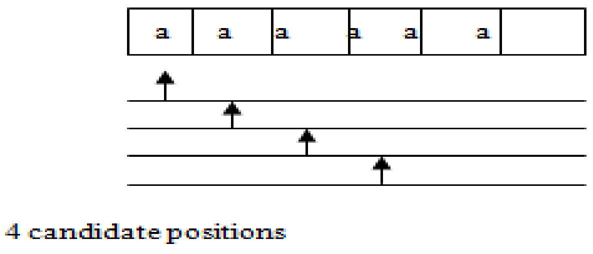


Figure 4.3 Extreme case attacks

E. *Virus Signatures*

A signature is a characteristic byte-pattern that is part of a certain virus or family of viruses. If a virus scanner finds such a pattern in a file, it notifies the user that the file is infected. The user can then delete, or (in some cases) "clean" or "heal" the infected file. Some viruses employ techniques that make detection by means of signatures difficult but probably not impossible. These viruses modify their code on each infection. That is, each infected file contains a different variant of the virus. Most modern antivirus programs try to find virus-patterns inside ordinary programs by scanning them for so-called virus signatures.

F. *Virus Detection*

The design considerations for a virus-detection engine in mobile devices are analyzed as follow. The system throughput should reach up to 1 Gbps for supporting real-time virus detection. The scalability of handling more than ten thousands patterns is required for versatile network protection. In addition, the system must be highly flexible to accommodate the rapidly increasing new virus patterns. Power consumption is the most important design consideration for mobile devices. The increasing virus pattern will greatly increase the power consumption and the cost of on-chip CAMs. The memory design is critical for dealing with the increasingly large virus database. For this purpose, special processor called virus detection processor with two phase architecture is used.

G. *Virus Detection Processor*

The virus-detection processor is to condense as much information on-chip as possible. Most of input data can be quickly scanned without further inspection. The entire virus scanning is split into two phases: fast on-chip filtering by the filtering engine, and the exactly- matching with some off-chip memory accesses. The filtering engine screens Impossible matches by consulting two TCAM lookup tables (named no-plane and yes-plane), which are used to perform two steps of the on-chip data scanning. There are six steps to be followed for detecting the viruses in which first two steps includes the Filtering Engine techniques and remaining steps includes the Exact Matching Engine techniques as shown in Figure 4.4.

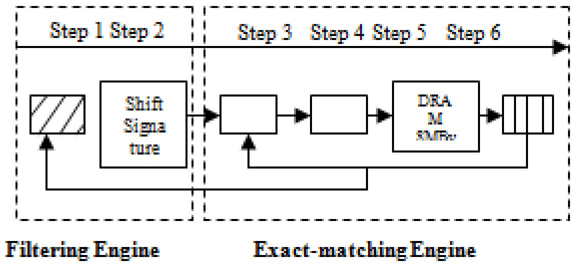


Figure 4.4 Execution Flow for Processor

The filtering engine and the exact matching engine have six steps for finding patterns. The six steps are as follows:

1. Search Window read.
2. Secure text filters.
3. Search window read.
4. Address generates.
5. Trie table read.
6. Sub-pattern compare.

Initially, a pattern pointer is assigned to point to the start of the given text at the filtering stage. Suppose the pattern matching processor examines the text from left to right. The filtering engine fetches a piece of text from the text buffer according to the pattern pointer and checks it by a shift-signature table. If the position indicated by the pattern pointer is not a candidate position, then the filtering engine skips this piece of text and shifts the pattern pointer right multiple characters to continue to check the next position.

The shift-signature table combines two data structures used by two different filtering algorithms, the Wu-Manber algorithm and the Bloom filter algorithm, and it provides two-layer filtering. If both layers are missing their filter, the processor enters the exact-matching phase. After an alarm caused by the filtering engine, the exact-matching engine precisely verifies this alarm by retrieving a trie structure. This structure divides a pattern into multiple sub-patterns and systematically verifies it. The exact-matching engine generally has four steps for each check. First, the exact-matching engine gets a slice of the text and hashes it to generate the trie address. Then, the exact-matching engine fetches the trie node from memory. This step causes a long latency due to the access time of the off-chip memory. Finally, the exact-matching engine compares the trie node with this slice. When this node is matched, the exact-matching engine repeatedly executes the above steps until it matches or misses a pattern. The pattern matching processor then backs out to the filtering engine to search for the next candidate.

V. FILTERING ENGINE TECHNIQUES

The filtering engine is a front-end module responsible for filtering out secure data efficiently and indicating to candidate positions that patterns possibly exist at the first stage. The performance of the filtering engine of a two-phase architecture can be ten times better than the exact-matching, attackers can forge specific strings that trigger the architecture to launch a large number of false alarms by its front-end module, causing it to busy itself verifying these alarms precisely by its back-end module. Designs that feature filters indicate that the action behind these filters is costly and necessary. In this work, the overall performance strongly depends on the filtering engine. Providing a high filter rate with limited space is the most important issue.

Filter Engines have individual memories for storing significant information. For cost reasons, only a small amount of significant information regarding the patterns can be stored in the filtering engine's on-chip memory.

In this case, a 32-kB on-chip memory is used for the ClamAV virus database, which contained more than 30000 virus codes and localized most of the computing inside the chip. The filtering engine techniques contain two algorithms as Wu-Manber algorithm and Bloom filter algorithm.

A. *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 preprocessing 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 Figure 3.5.

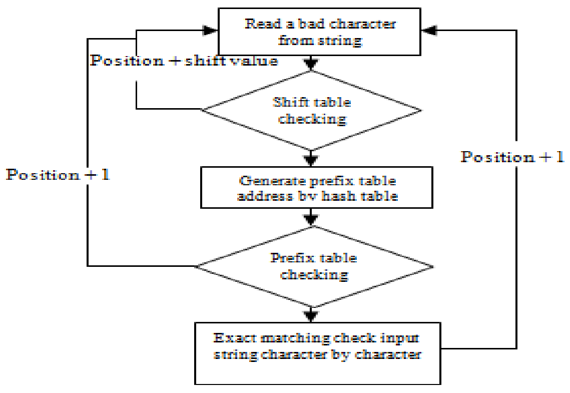

Read a bad character from string
Position + shift value
Shift table checking
Position + 1
Generate prefix table address by hash table
Position + 1
Prefix table checking
Exact matching check input string character by character

Figure 5.1 Matching Flow of Wu – Manber algorithm

The matching flow matches patterns from the tail of the minimum pattern in the pattern set, and it takes a block 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 O(BN/m) for the given text with length N 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.

B. *Matching Process*

For the pattern set {erst, ever, there} the maximum shift value is three characters for B=2 and m=4. 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, the Wu-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, the Wu-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.

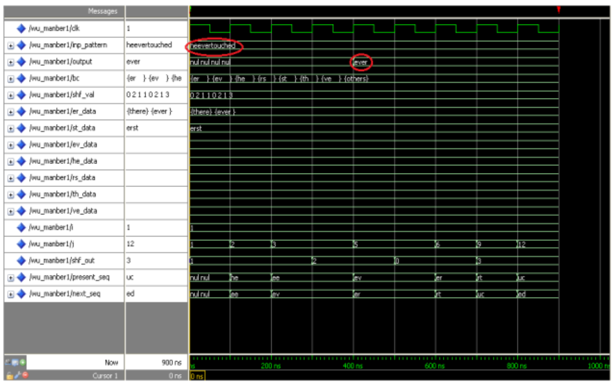
C. *Bloom filter Algorithm*

A Bloom filter is a space-efficient data structure used to test whether an element exists in a given set. This algorithm is com- posed 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 pat-tern 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 independent of its pattern length.

VI. SIMULATION RESULTS

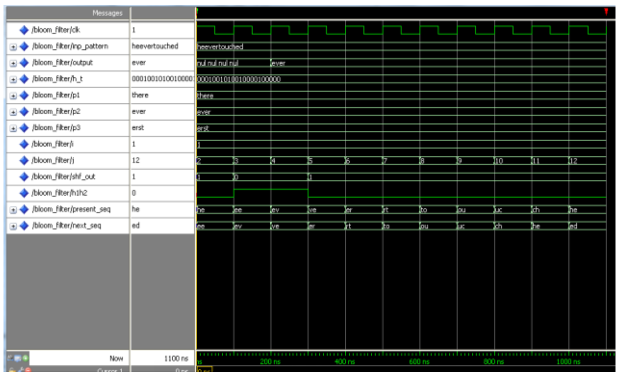
The Filtering Engine architecture is designed using Verilog HDL and simulation is done by using ModelSim6.2 is shown in Figure 6.1 and 6.2. The output detects the virus data "ever" from the given set of input by using proposed algorithm.



```
Messages
/wu_manber1/clk             1
/wu_manber1/inp_pattern     heevertouched
/wu_manber1/output          ever
/wu_manber1/bc              {er } {ev } {he
/wu_manber1/shf_val         0 2 1 1 0 2 1 3
/wu_manber1/er_data         {there} {ever }
/wu_manber1/st_data         erst
/wu_manber1/ev_data
/wu_manber1/he_data
/wu_manber1/rs_data
/wu_manber1/th_data
/wu_manber1/ve_data
/wu_manber1/i               1
/wu_manber1/j               12
/wu_manber1/shf_out         3
/wu_manber1/present_seq     uc
/wu_manber1/next_seq        ed
Now                         900 ns
Cursor 1                    0 ns
```

Figure 6.1 Simulated output of Wu-Manber Algorithm

The performance of the Wu-Manber algorithm does not increase with table size but rather remains almost unchanged. The experiment indicates that only 1.3 to1.7 characters can be shifted for each check during the run. However, the filtering engine based on the Wu-Manber algorithm still has fewer checks than the bloom filter because of its shifting feature and also because of the long access time of the external memory.



```
Messages
/bloom_filter/clk           1
/bloom_filter/inp_pattern   heevertouched
/bloom_filter/output        ever
/bloom_filter/h_t           0001001010010000
/bloom_filter/p1            there
/bloom_filter/p2            ever
/bloom_filter/p3            erst
/bloom_filter/i             1
/bloom_filter/j             12
/bloom_filter/shf_out       1
/bloom_filter/h1h2          0
/bloom_filter/present_seq   he
/bloom_filter/next_seq      ed
Now                         1100 ns
```

Figure 6.2 Simulated output of Bloom Filter Algorithm

VII. 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 tradeoff between performance and cost due to an efficient filter table existing in internal memory; however, their performance is easily threatened by malicious attacks. This work analyzes two scenarios of malicious attacks and provides two methods for keeping performance within a reasonable range. First, the design 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.

REFERENCES

[1] Chieh-Jen Cheng, "A Scalable high-performance virus detection processor ssagainst a Large Pattern Set for Embedded Network Security", vol. 20, No. 5, May 2012.
[2] Fredkin.E, "Trie memory," Commun. ACM, vol. 3, pp. 490-499, 1960.
[3] Bloom B.H, "Space/time trade-offs in hash coding with allowable errors," Commun. ACM, vol. 13, pp. 422-426, 1970.
[4] Aho A.V and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," Commun. ACM, vol. 18, pp. 333-340, 1975.
[5] Boyer R.S and J. S. Moore, "A fast string searching algorithm," Commun. ACM, vol. 20, pp. 762-772, 1977.
[6] Wu .S and U. Manber, "A fast algorithm for multi-pattern searching," Univ. Arizona, Tucson, Report TR-94-17, 1994.
[7] Sidhu R. and V. K. Prasanna, "Fast regular expression matching using FPGAs," in Proc. 9th Annu. IEEE Symp. Field-Program. Custom Comput. March, 2001, pp. 227-238.
[8] Cho Y. H., S. Navab, and W. H. Mangione-Smith, "Specialized hard- ware for deep network packet filtering," in Proc. Reconfig. Comput.Going Mainstream, 12th Int. Conf. Field-Program. Logic Appl., 2002, pp. 452-461.
[9] Memik G., S. O. Memik, and W. H. Mangione-Smith, "Design and analysis of a layer seven network processor accelerator using reconfig- urable logic," in Proc. 10th Annu. IEEE Symp. Field-Program. Custom Comput. Mach., 2002, pp. 131-140.
[10] Micron Technology, Inc., Boise, ID, "256 MB DDR2 SDRAM datasheet," 2003.
[11] Clark C. R and D. E. Schimmel, "Scalable pattern matching for high speed networks," in Proc. 12th Annu. IEEE Symp. Field-Program. Custom Comput. Mach., 2004, pp. 249-257.
[12] Dharmapurikar S, P. Krishnamurthy, and T. S. Sproull, "Deep packet inspection using parallel bloom filters," IEEE Micro, vol. 24, no. 1, pp. 52-61, Jan. 2004.
[13] Liu R.-T., N.-F. Huang, C.-N. Kao, and C.-H. Chen, "A fast string-matching algorithm for network processor-based intrusion detection system," ACM Trans. Embed. Comput. Syst., vol. 3, pp. 614-633, 2004.
[14] Sourdis I. And D. Pnevmatikatos, "Pre-decoded CAMs for efficient and high-speed NIDS pattern matching," in Proc. 12th Annu. IEEE Symp.Field-Program. Custom Comput. Mach., 2004, pp. 258-267.
[15] Yu F, R. H. Katz, and T. V. Lakshman, "Gigabit rate packet pattern-matching using TCAM," in Proc. 12th IEEE Int. Conf. Netw. Protocols,2004, pp. 174-183.
[16] Piyachon P. and Y. Luo, "Efficient memory utilization on network pro- cessors for deep packet inspection," presented at the ACM/IEEE Symp. Arch. for Netw. Commun. Syst., San Jose, CA, 2006.
[17] Yi. S, B.-K. Kim, J. Oh, J. Jang, G. Kesidis, and C. R. Das, "Memory-efficient content filtering hardware for high-speed intrusion detection systems," presented at the ACM Symp. Appl. Comput., Seoul, Korea, 2007.