A Fast Multi-pattern Matching Algorithm for Deep Packet Inspection on a Network Processor

Jia Ni[1], Chuang Lin[1], Zhen Chen[1,2] and Peter Ungsunan[1]
Department of Computer Science[1], Research Institute of Information Technology[2], Tsinghua University, Beijing
{jni, clin, zhenchen, hongsunan}@csnet1.cs.tsinghua.edu.cn

Abstract

Deep Packet Inspection (DPI) is a critical function in network security applications such as Firewalls and Intrusion Detection Systems (IDS). Signature based scanners used in DPI apply multi-pattern matching algorithms to check whether the packet payload or flow content contains a specified signature in a signature set. Existing multi-pattern matching algorithms sacrifice memory space to achieve better performance. In this paper a novel fast multi-pattern matching algorithm, the Hash Boyer-Moore (HBM) Algorithm, is presented, which reduces the memory footprint of the heuristic table using a hash function and adds another heuristic table to reduce the false-positive ratio. Analyses and simulations show HBM offers higher speed and lower memory cost than some existing algorithms. The HBM algorithm was implemented on the Intel IXP 2400 Network Processor (NP) platform and experiments show suitable performance results in a Gigabit Ethernet LAN environment.

1. Introduction

With the increasing type and number of malicious attacks in the Internet[1-3], network security appliances such as Firewalls and IDS systems[4,5] need an effective tool to detect such attacks. Deep Packet Inspection (DPI) is a promising method of detecting these attacks and is already widely used. DPI usually consists of several functions, such as regular expression matching[6] and signature based scanning. Signature based scanning has many practical uses[8-10], the main principles of which are to check packet payloads, maintain flow states and scan flow streams to find specified signatures. This takes much more time than application protocol analysis[11] and usually becomes the bottleneck of the system. Thus, the design of high performance signature based scanners is very important in high-speed network security applications.

The essence of a signature based scanner is a multi-pattern matching algorithm. Based on a certain set of string patterns, such as worm code signatures[7], the scanner checks each byte of the packet payload to find out whether it contains one of these patterns (In this paper, the terms "signature" and "pattern" are used interchangeably).

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 describes the HBM algorithm. Section 4 gives some theoretical analyses and simulation results compared to some existing algorithms. Section 5 presents the test environment and experimental results on the Intel IXP 2400 Network Processor (NP) platform. Section 6 summarizes the paper and presents conclusions.

2. Related Works

Developing an efficient multi-pattern matching algorithm is still a difficult issue in research. There are three kinds of algorithms usually used to tackle this problem: (1) the Bloom Filter algorithm and its extensions; (2) the Aho-Corasick (AC) algorithm and its extensions; and (3) the Boyer-Moore Algorithm (BM) and its extensions.

B. Bloom proposed the Bloom Filter algorithm [8,18-20,24], which is widely implemented in hardware, like FPGA which contains multiple hash function blocks and paralleled memory accesses. Aho and Corasick proposed the AC algorithm[12], which has proven linear performance, making it suitable for searching a large set of signatures. The AC algorithm and its extensions deal well with regular expression matching, but they are not optimal for fixed pattern matching like worm scanning for its large number of states and frequent I/O operation compared to BM and its extensions.

Boyer and Moore proposed the classical single-pattern matching BM algorithm[13]. However, it cannot be transplanted to multi-pattern matching trivially, in which cases, some extensions of the BM algorithm are proposed, such as the WM algorithm[21], the Setwise BMH algorithm[22], or the AC-BM algorithm[23].These popular extensions change some features of the traditional algorithm to adapt to the multi-pattern case, which is mainly done on two points: (1) Several characters are combined into a block as one comparison unit; (2) the false-positive case is allowed and exact matching is needed.

The WM algorithm combines several characters into one block and only checks the rightmost block of each substring in the input stream from a heuristic skip table (so-called Table Delta 1). In this way, the heuristic skip table would be large. Setwise BMH and AC-BM use the trie and tree structure to form Table Delta 1 in order to decrease the memory cost, but additional I/O operations are needed. Meanwhile, the three algorithms above only use the Delta 1 table but omit the second skip table in BM (so-called Table Delta 2). The RSI[17] algorithm analyzes the relationship between the rightmost two blocks and reduces the false-positive ratio, but it uses more memory and still does not compare information from all the blocks.

We propose the HBM algorithm, an extension of the BM algorithm, with the following main contributions: (1) Table Delta 1 is organized in a hash mode allowing collision to reduce the memory footprint; (2) Table Delta 2 is constructed to reduce false-positives; (3) Skipping is allowed at any position to improve the algorithm's efficiency.

3. HBM Algorithm Description

The HBM algorithm consists of two parts. One is the off-line initialization which constructs Table Delta 1 and Delta 2 depending on the signature set. The other is the run-time procedure to check whether the input string stream contains a certain signature.

3.1. Symbol and Assumption

Let S indicate the signature set, n indicate the set size, L indicate the length of each signature (assume that all the signatures have the same length), and b indicate the block size. Thus, $S = \{S_0, S_1, \cdots, S_{n-1}\}$, where k-th signature is $S_k = s_{k,0}s_{k,1}...s_{k,L-1}$ .Let $y = h(a_0a_1a_2...a_{b-1})$ indicate the hash function, which is a mapping from a block of b characters to a value ranged in $[0, m-1]$. Let m indicate the size of Table Delta 1, where $D_1(i)$ is i-th item. The size of Table Delta 2 is fixed to $L-b+1$, where $D_2(t)$ is the t-th item. The input string is represented as *str(0)*, *str(1)*,…,where *str(i)* is i-th character. All the subscripts start at 0 in this paper.

A "comparison" is defined as the fetching of a character block from an input string and checking its value in Table Delta 1. If the value satisfies some condition, the comparison "succeeds", otherwise it "fails".

A "check" is defined as the fetching of a substring with the same signature length from the input string and testing whether it is a certain signature. If each comparison succeeds, then this check "succeeds", otherwise it "fails".

3.2. Off-line Initialization

The off-line initialization can be divided into three steps: (1) Construct Table Delta 1; (2) Construct Table Delta 2; (3) Construct the hash table for storing signatures (Table SHT). Before describing the algorithm in detail, an example is given to illustrate the principle of HBM.

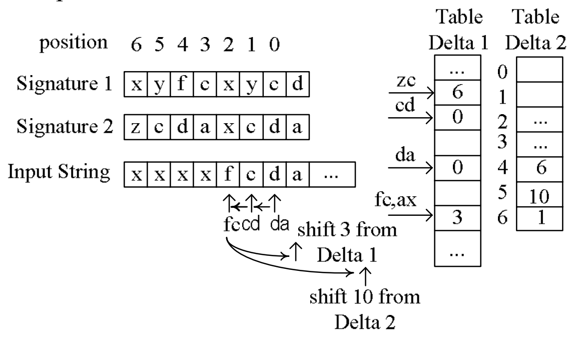


Figure 1. An example of the HBM algorithm

Each block in a signature has a hash entry in Delta 1, which represents the position of the rightmost appearance of the block. The position is counted from the right end. In Figure 1, the block size is 2 and the some entries in Delta 1 are shown. (For hash collisions such as with blocks "fc" and "ax", the same entry is shared, which contains the smaller value.) We make the following observations:

Observation 1: The Delta 1 value of the rightmost block of a signature should be 0, the 2nd block should less than or equal to 1, the 3rd should be less than or equal to 2, and so on.

Therefore, the results of a comparison can be determined by Observation 1. If the comparison succeeds, the next block to the left is compared. If the comparison fails, the pointer can skip some distance in order to align the block with its rightmost appearance in signatures. That is,

Observation 2: When current check fails, the comparison pointer can at least skip a distance of the value indexed in Table Delta 1.

Meanwhile, the occurrence of a failure indicates the success of former comparisons. In Figure 1, the failure happens at block "fc", so substring "cda" may "appear" at the end of a signature. If that certain signature contains a substring "\$cda" at a left position ('\$' is an arbitrary char), but not "fcda", the comparison pointer can skip a distance in order to make it align with "\$cda". Table Delta 2 stores such skipping distances.

Observation 3: When the current check fails, the comparison pointer can at least skip a distance of the value indexed in Table Delta 2.

If all the comparisons in the current check succeed, it only means that the substring may be a signature because the condition in Observation 1 is only a necessary condition. Therefore,

Observation 4: When current check succeeds, an exact match is needed to confirm that the substring is a real signature to avoid false-positives.

Thus, the principle of the HBM algorithm is to follow the four Observations mentioned above.

```
Initialize Table Delta 1:
1. $\forall x, D_1(x) = L - b + 1$
2. for $i := 0$ to $n-1$
     for $j := 0$ to $L-b$
       $x = h(s_{ij} s_{i(j+1)} \dots s_{i(j+b)})$
       if $D_1(x) > L - b - j$
       then
         $D_1(x) \leftarrow L - b - j$
       end if;
     end loop $j$;
   end loop $i$;
```

```
Initialize Table Delta 2:
1. Initialize Table Delta 1
2. $\forall t$, $D_2(t) = Infinite\_max$
3. for $i := 0$ to $n-1$
     for $t := 0$ to $L-b$
       $d \leftarrow L-b+1-rpr(S_i,t)$
       If $D_2(t) > d$
       then $D_2(t) \leftarrow d$
       end if;
     end loop $t$;
   end loop $i$;
```

Figure 2. The construction of Table Delta 1 and 2

Figure 2 shows the algorithm of constructing Table Delta 1 and Delta 2. As is shown in Observation 1 and 2, values in Delta 1 present not only the positions of the rightmost appearance of the block but also the skip distances.

The construction of Table Delta 1 differs from the BM algorithm in that: (1) The comparison unit is a block of characters instead of a single character; (2) The entry in Table Delta 1 is determined by the hash value of the block instead of the character itself; (3) Table Delta 1 allows hash collision. Once it happens, the smallest of the skip distances is stored there.

The construction of Table Delta 2 is also shown in Figure 2. It is similar to the BM algorithm, but differs in calculating the function *y=rpr()*. "*rpr*" means the rightmost plausible reoccurrence of a substring, which has the same meaning as in BM. Function $rpr(S_i, t)$ represents the rpr value at position t in signature S_i. First, we will define the "extension" of signatures and provide the definition of "unify".

Every signature pattern is preceded with the wild character "\$" in front of the first character, i.e., $pat = a_0 a_1 a_2 \dots a_{L-1}$ would be represented as $pat = \dots\$\$\$\$ a_0 a_1 a_2 \dots a_{L-1}$, and the Delta 1 value of each block including character "\$" would be 0 ($D_1 = 0$).

Definition of "unify": A substring in the signature pattern $substr = c_0 c_1 \cdots c_n$ is called unified, iff condition (*) is satisfied.

$$\begin{cases} D_1(c_{i-b+1} c_{i-b+2} \cdots c_{n-1}) \le 0, \dots, D_1(c_1 c_2 \cdots c_{b-1}) \le n-b \\ D_1(c_0 c_1 \cdots c_{b-1}) > n-b+1 \text{ or } c_0 = \$ \end{cases} \quad (*)$$

Here the term "unify" is derived from the BM algorithm with some changes. According to the definition above, a unified substring has the same characteristic as the tail substring in the signature. In other words, the certain substring is unified with the tail of the signature string. Function *rpr()* calculates the position of the rightmost "unified" substring. In signature S_i, if $substr = c_0 c_1 \cdots c_n$ is the rightmost unified substring, the result of function $rpr(S_i, t)$ is the position of character c_1 in S_i. Therefore, the value in Table Delta 2 is:

$$D_2(t) = \min_{\forall S_i \in S} \{L - b + 1 - rpr(S_i, t)\} \qquad (**)$$

For the same reason as in the BM algorithm, the Delta 2 value of the rightmost block position would never be used, which can be defined as 1 manually in the consideration of easy implementation. In other words, the last value of Table Delta 2 is 1 and is always equal to or less than the value of any entry in Table Delta 1 and no influence is taken into the algorithm.

3.3. Run-time Procedure

The run-time procedure of the HBM algorithm is rather easy. For each substring, mismatches are judged according to the condition $\overline{D_1(i)} < j$ from right to left, $\overline{D_1(i)}$ is the Delta 1 value of the block at position i (that is $\overline{D_1(i)} = D_1(h(str(i), str(i+1), \dots, str(i+b-1)))$). Once a mismatch is found, the pointer i skips a distance according to the maximum value in Table Delta 1 and Delta 2 until it reaches the end of the input stream. If there is no mismatch for a substring, an extra comparison is needed.

As is shown in Observation 4, an exact test is needed when a check succeeds. In order to reduce I/O operations, a hash table (called SHT) is used to store the signature set in Figure 3 where linked lists are used to solve the hash collision, so an exact test only needs to read the signatures with the same hash key and ignore the other ones.

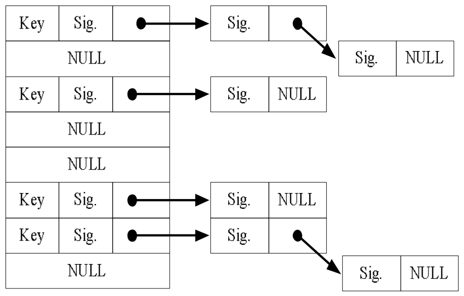


Figure 3. Hash table for storing signatures (SHT)

4. Performance Analysis

In the HBM algorithm, the criterion of performance is based on two parts. One is the average skip distance which represents the instances of comparison; the other is the false-positive ratio which represents the additional time consumption for exact matches. Thus, the time consumption for each character can be represented as follows:

$$\overline{T_{char}} = t_{comp} \cdot \overline{c_{char}} + \frac{t_{exact-comp} \cdot P_{false-positive}}{\overline{distance_{comp}}} = \frac{t_{comp} + t_{exact-comp} \cdot P_{false-positive}}{\overline{distance_{comp}}} \quad \textbf{(Eq. 1.)}$$

In Eq. 1, $\overline{T_{char}}$ indicates the time cost for dealing with each character, t_{comp} indicates the time for a comparison, $\overline{c_{char}}$ indicates the average number of comparisons dealing with a character, $t_{exact-comp}$ indicates the time to test for false-positives. $P_{false-positive}$ indicates the false-positive possibility of one check, $\overline{distance_{comp}}$ indicates the average skip distance after a comparison. t_{comp} and $t_{exact-comp}$ are constants determined by the data structure and memory I/O speed, so the false-positive ratio $P_{false-positive}$ and average skip distance $\overline{distance_{comp}}$ become the main evaluation of performance.

4.1. Average Skip Distance

In this section, the theoretical analysis will be explained first and a comparison of the HBM, WM and RSI algorithms will also be provided by simulation.

In the HBM algorithm, the two Delta tables are used to achieve higher efficiency, though it is much more likely to use values in Table Delta 1 than Delta 2. When a skipping happens, it has only less than a 1% chance to use the value in Table Delta 2 but more than a 99% chance to use Table Delta 1. Table Delta 2 is mainly used to reduce the false-positive ratio.

Considering that Delta 1 is used in most cases, we can simply approximate the average skip distance value in Table Delta 1 as the one in the whole HBM algorithm. If the hash function outputs results in a uniform distribution, the possibility of a block hitting a certain entry in Table Delta 1 is p=1/m. The average skip distance can be calculated by Eq. 2.

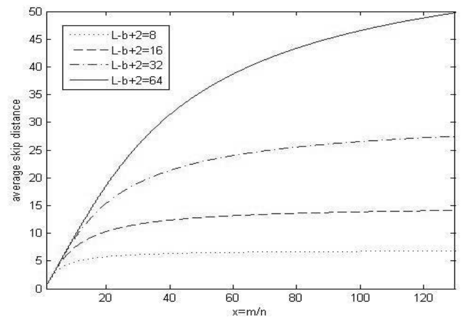
$$\overline{distance_{\text{Delta1}}} = \sum_{i=0}^{L-b} \sum_{j=0}^{n-1} (1-p)^{in+j} \cdot p \cdot i + (1-p)^{(L-b+1)n} (L-b+1) \quad \textbf{(Eq. 2.)}$$

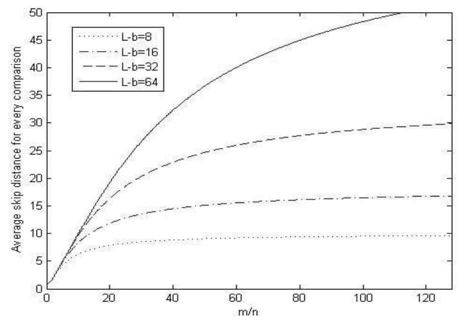
Let $a = (1-p)^n = (1-\frac{1}{m})^n \approx e^{-\frac{n}{m}}$, when m is a large number. Eq. 2 can be rewritten as Eq. 3

$$\overline{distance_{\text{Delta1}}} \approx \frac{a - a^{l-b+2}}{1-a} \approx \frac{e^{-\frac{n}{m}} - e^{-\frac{n}{m}(l-b+2)}}{1-e^{-\frac{n}{m}}} \quad \textbf{(Eq. 3.)}$$

The average skip distance after a check, $\overline{distance_{comp}}$, can be approximately calculated (see Figure 4(b)) as Eq. 4:

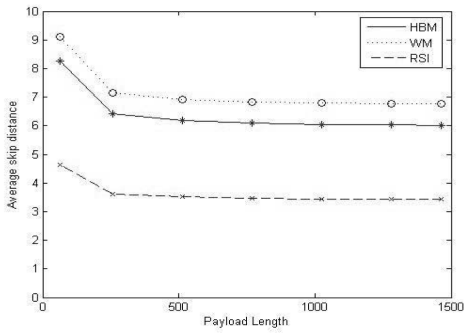
$$\overline{distance_{comp}} \approx \sum_{i=0}^{k} \frac{skip_i \cdot P(j=i)}{i+1} + 1 \cdot P(match) = \sum_{i=0}^{k} \frac{(1-a^{L-b+2-i}) \cdot a^{i+1} \cdot \prod_{t=0}^{i} (1-a)^t}{(1-a) \cdot (i+1)} + \prod_{t=0}^{k+1} (1-a)^t \quad \textbf{(Eq. 4.)}$$

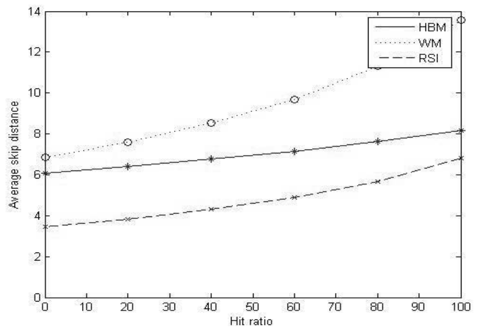




(a) Average skip distance in table Delta 1 (b) Average skip distance for each comparison

Figure 4. Average skip distance with m/n





(a)Distance with payload length (hit ratio=0) (b)Distance with hit ratio (payload length = 704)

Figure 5. Average skip distance of HBM, WM and RSI (Table Delta 1 size = 3840 entries)

It is clear that the average skipping distance is a function of m/n with parameters L and b whose relationship is shown in Figure 4(a), where the acceleration of skip distance decreases with the increase in size of Table Delta 1.

Figure 5 shows the simulation results which compare the average skip distance of the HBM, WM and RSI algorithms with 512 randomly signatures of 32 bytes fixed length. From Figure 5, the WM algorithm has the best skip distance and HBM is a little worse. But the false positive ratio of WM is not much higher than HBM and RSI as shown in Figure 7. In fact, an exact comparison takes hundreds of times longer than an operation that visits Table Delta 1, so WM may offer the worst performance in practical implementation.

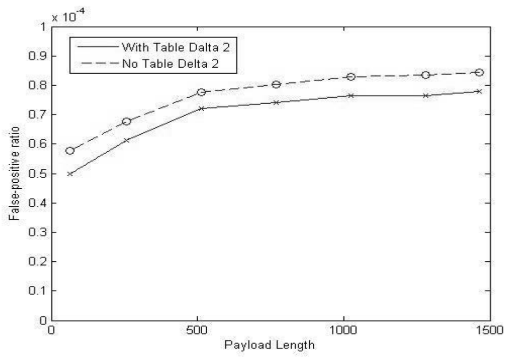
4.2. False-positive Ratio

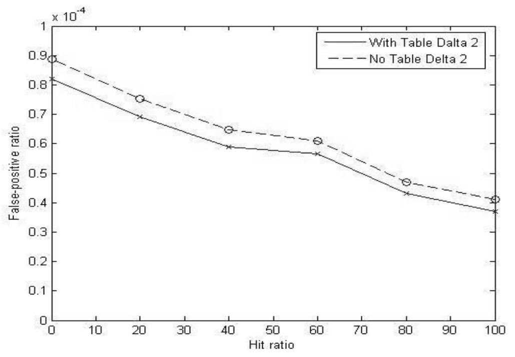
Unlike the AC algorithm, all of the extensions of the BM algorithm are subject to false-positives. The possibility of a false-positive in HBM can be calculated as Eq. 5.

$$P_{false-positive} = \prod_{j=1}^{L-b+1}[1-(1-\frac{1}{m})^{jn}]-\frac{n}{2^L} \qquad \textbf{(Eq. 5.)}$$

Table Delta 2 is used to reduce the false-positive ratio as shown in Figure 6. In most cases, the algorithm without Table Delta 2 will have a false-positive ratio about 10% higher, which means about 10% more false-positive testing.

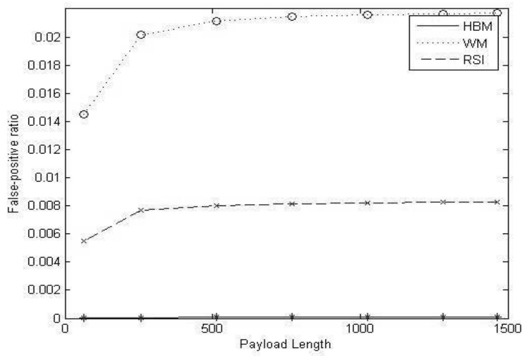
The HBM algorithm has a lower false-positive ratio, for it uses Observation 1 in the entire procedure while other existing algorithms only use it once or several times. Figure 7 presents the simulation results with the HBM, WM and RSI algorithms. From Figure 7 we can see that HBM has a much lower false-positive ratio than WM and RSI, so it is possible to provide much higher performance than the two existing algorithms.

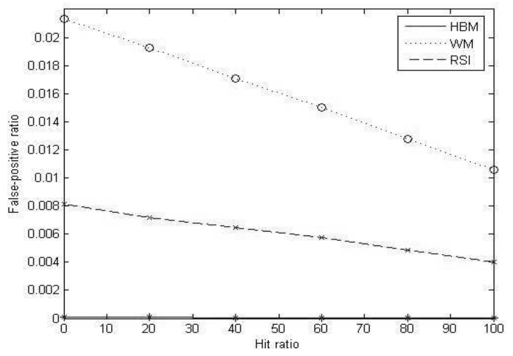




(a)False-positive with payload length (hit ratio=0) (b)False-positive with hit ratio (payload length = 704)

Figure 6. False-positive ratio with and without Table Delta 2 (Table Delta 1 size = 3840 entries)





(a)False-positive with payload length (hit ratio=0) (b)False-positive with hit ratio (payload length = 704)

Figure 7. False-positive ratio of three algorithm (Table Delta 1 size = 3840 entries)

5. Experiment

In our work, Network Processors[14-16] are chosen instead of General Purpose or ASIC Processors in virtue of its programming flexibility and specific architecture for processing network packets. Our scanner is implemented on Intel IXP 2400 platform.

The optimization of HBM is mainly focused on two points: (1) Design and assign function blocks in parallel to make full use of the multiple Micro Engines (MEs); (2) Store two heuristic tables and Table SHT in different memory types to fit the hierarchical memory structure. Compared to ALU operations, I/O operations are more likely to cause system bottlenecks. The HBM algorithm is very suitable for the hierarchical memory structure. Table 1 is allocated memory for 512 signatures 32 bytes long each.

Table 1. The memory allocation of the HBM algorithm in the IXP 2400

Context	Distribution	Size	Speed
Table Delta 1	On chip Local Memory	640 Long Word	Fast (several cycles)
Table Delta 2	On chip Next Neighbor Reg.	32 bytes	Fast (several cycles)
Original Signature Set	Order storing in DRAM	16K bytes	Slow (100MHz bus)
SHT	Hash link table in DRAM	32K bytes	Slow (100MHz bus)

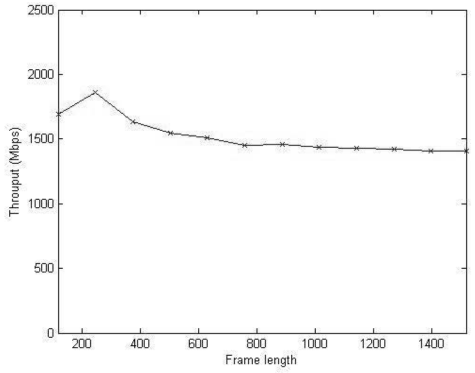
In our experiment, 512 signature patterns are generated randomly with a fixed length of 32 bytes. The length of an input Gigabit Ethernet frame is indicated by 118, 246, …, 1398, and 1518. The hit ratio is from 0% to 100%. The carrying signature is inserted randomly at any position of the payload. The throughput results are shown from Figure 8 to Figure 10.

Compared with the results in [9-10,18], it is demonstrated that there is a big performance leap in throughput and delay, which makes the HBM algorithm based scanner practical in Gigabit Ethernet.

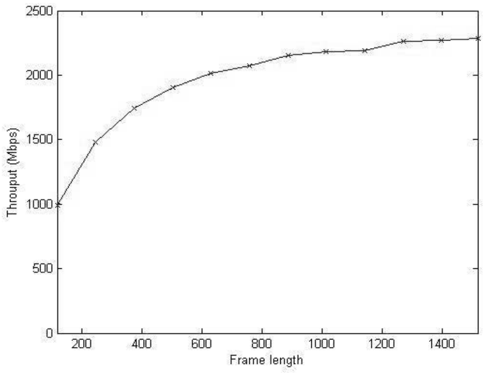
Figure 8 plots the variation of throughput performance with frame length and Figure 9 presents the variation of the throughput performance with hit ratio. Figure 10 shows a referential delay cycle at the media bus at 104MHz. From the experiment, the signature based scanner with the HBM algorithm has achieved the throughput of Gigabit Ethernet. Meanwhile, the throughput ratio is stable with the variation of frame length and hit ratio.

As the system needs to scan the entire payload by shifting each byte one by one, the frame forwarding rate is related to the size of the frame. In the worst case (i.e., hit ratio is 0%), the throughput comes down slowly (see Figure 8 (a)) as the size of the frame is increased.

In the simulation experiment, the testing for exact match is also executed on MEs. Once a string is matched and proved true, the scanner just stops and alarms without any more operations. Therefore, it takes fewer cycles when handling frames with signatures, especially long frames with high hit ratios. For example, in the case of worm scanning, the throughput of the system will not be impaired when a worm breaks out and the network is jammed with worm frames (see Figure 8(b) and Figure 9).



(a) Hit ratio is 0%



(b) Hit ratio is 100%

Figure 8. The variation of throughput with the frame length

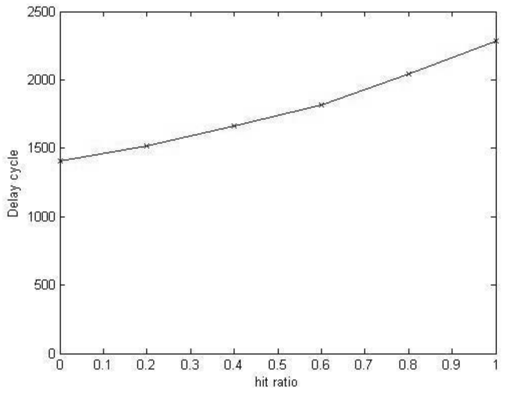


Figure 9. The variation of throughput with the hit ratio (frame length = 1518)

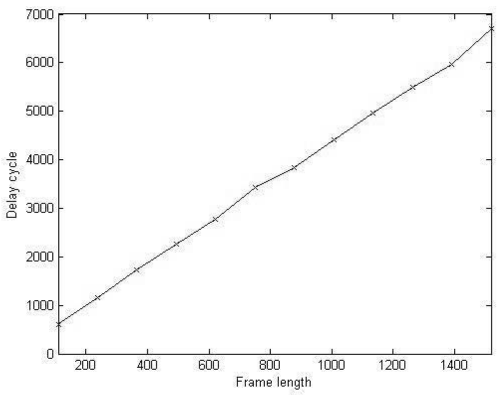


Figure 10. The variation of single frame delay with the frame length (hit ratio = 0)

6. Conclusion

Signature-based scanners are widely used in network security applications. Their critical function is the multi-pattern matching algorithm which is still an important issue in the high speed network security field. The HBM Algorithm, a novel fast multi-pattern matching algorithm, is presented in this paper to inherit the skipping characteristic of the Boyer-Moore algorithm when used in multi-pattern matching mode. It offers higher performance and takes less memory than other existing algorithms. This signature-based scanner with HBM was implemented on the Intel IXP 2400 Network Processor platform and its performance was tested. From these experiments, it has proven stable, offers high performance and meets the needs of Gigabit Ethernet.

References

[1] Darrell M. Kienzle, Matthew C. Elder, “Recent worms: a survey and trends,” Proceedings of the 2003 ACM workshop on Rapid Malcode (WORM’2003), Pages: 1–10, October 2003.
[2] Nicholas Weaver, Vern Paxson, Stuart Staniford, Robert Cunningham, “A taxonomy of computer worms,” Proceedings of the 2003 ACM workshop on Rapid Malcode (WORM’2003), Pages: 11- 18, October 2003.
[3] D. Moore, C. Shannon, and J. Brown, “Code-Red: a case study on the spread and victims of an internet worm,” Proceedings of the Internet Measurement Workshop 2002, Marseille France, November 2002.
[4] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time,” Computer Networks, 31(23-24), pp. 2435-2463, 14 Dec. 1999. Also see www.bro-ids.org.
[5] M. Roesch, “Snort: Lightweight intrusion Systems Administration Conference,” 1999. Also see www.snort.org.
[6] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley and J. Turner., “Algorithms to accelerate Multiple Regular Expression Matching for Deep Packet Inspection,” ACM SIGCOMM Computer Communication Review, Volume: 36 , Issue: 4, October 2006.
[7] J. Newsome, B. Karp and D. Song, “Polygraph: Automatically Generating Signatures for. Polymorphic Worms,” in Proceedings of the IEEE Symposium on Security and. Privacy (Oakland 2005), May 2005.
[8] Bharath Madhusudan and John W. Lockwood, “A Hardware-Accelerated System for Real-Time Worm Detection,” IEEE Micro, Volume 25, Issue 1, January 2005.
[9] Zhen Chen and Chuang Lin et al., “AntiWorm NPU-based Parallel Bloom filters for TCP/IP Content Processing in Giga-Ethernet LAN,” IEEE proceeding of Local Computer Network, the First Workshop on Network Security (WoNS 2005), Sydney, Australia, November 15, 2005.
[10] Zhen Chen and Chuang Lin et al., “AntiWorm NPU-based Parallel Bloom filters in Giga-Ethernet LAN,” IEEE International Conference on Communications (ICC 2006), Network Security and Information Assurance Symposium, 2006.
[11] H. Dreger, A. Feldmann, M. Mai, V. Paxson and R. Sommer, “Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection,” Proc. USENIX Security Symposium, August 2006.
[12] A.V. Aho and M. J. Corasick, “Fast pattern matching: an aid to bibliographic search,” Communication of ACM, Vol. 18, No. 6, June 1975.
[13] Robert S. Boyer and J Strother Moore, “A Fast String Searching Algorithm,” Communication of ACM, Vol. 20, No. 10, 1977.
[14] Bill Carlson, Intel Internet Exchange Architecture Intel Network Processors, Intel Press, 2003.
[15] Erik J. Johnson and Aaron R. Kunze, IXP2400/2800 Programming: The Complete Microengine Coding Guide, Intel Press, 2003.
[16] Uday R. Naik and Prashant R. Chandra, Designing High-Performance Networking Application, Intel Press, 2004.
[17] Bo Xu, Xin Zhou and Jun Li, “Recursive Shift Indexing: A Fast Multi-Pattern String Matching Algorithm,” Proc. of the 4th International Conference on Applied Cryptography and Network Security (ACNS 2006), 2006.
[18] Rong-Tai Liu, Nen-Fu Huang, Chia-Nan Kao, Chih-Hao Chen, Chi-Chieh Chou, “A Fast Pattern-Match Engine for Network Processor-based Network Intrusion Detection System,” International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 1, 2004.
[19] B. Bloom, “Space/Time Trade-Offs in Hash Coding with Allowable Errors,” Comm. ACM, vol. 13, no. 7, May 1970, pp. 422-426.
[20] Sarang Dharmapurikar, Praveen Krishnamurthy, T.S. Sproull and J. W. Lockwood, “Deep packet inspection using parallel bloom filters,” IEEE Micro, Volume 24, Issue 1, Pages:52 – 61, Jan.-Feb. 2004.
[21] S. Wu and U. Manber, “A fast algorithm for multi-pattern searching,” Technical Report TR-94-17, Department of Computer Science, University of Arizona, 1994.
[22] M. Fisk and G. Varghese, “An analysis of fast string matching applied to content-based forwarding and intrusion detection,” Technical Report CS2001-0670, University of California at San Diego, 2002.
[23] C. J. Coit, S. Staniford, and J. McAlerney, “Towards faster pattern matching for intrusion detection, or exceeding the speed of snort,” Proc. of the 2nd DARPA Information Survivability Conference and Exposition (DISCEX II), 2002.
[24] H. Song and J. Lockwood, “Multi-pattern signature matching for hardware network intrusion detection systems,” Global Telecommunications Conference, 2005. GLOBECOM’05. IEEE Volume 3, Page(s):5, Nov., 2005.