A Fast and Scalable Multi-Pattern Matching Algorithm for Intrusion Detection Systems

Chun-Liang Lee^a, Chung-Yuan Huang^{a,*}, Kai-Ping Lu^a, Jhao-Han Chen^a

^aDepartment of Computer Science and Information Engineering, School of Electrical and Computer Engineering,
College of Engineering, Chang Gung University, No. 259, Wen-Hwa 1st Road,
Kwei-Shan, Tao-Yuan 33302, Taiwan

*Corresponding Author: gscott@mail.cgu.edu.tw

Abstract

In order to protect networks from attacks, network intrusion detection systems (NIDS) have been widely deployed. These devices scan incoming packets to detect malicious content according to the predefined patterns. It is time consuming for NIDS to inspect each packet to check if it contains any patterns. In this paper, we propose a scalable and high-performance pattern matching algorithm. The key idea behind the proposed algorithm is to build a small and adjustable lookup table which can be completely stored in the on-chip memory of a network processor, and reduce the probability of accessing the external memory. Since the latency of one on-chip memory access is far smaller than that of one external memory access, the time required to inspect a packet can be greatly reduced. Simulation results show that the proposed algorithm is significantly better than the compared algorithm in terms of speed and scalability.

Keywords: network security, intrusion detection, pattern matching, deep packet inspection.

1. Introduction

Network security is an important issue that has been studied extensively. Conventional network security systems such as firewalls can restrict network traffic if the packet headers contain abnormal IP addresses, ports, and protocols. However, only examining the headers does not ensure security, especially from the application layer perspective^(1,2). Hence, network intrusion detection system (NIDS) has been developed to provide better security protection.

For each incoming packet, an NIDS scans the packet

payload to determine if it contains any pre-defined patterns (or signatures). If any patterns are identified, the system generates alert messages or drops the packet to protect other network devices. Pattern matching is a time-consuming task in an NIDS. Studies have indicated that it consumes up to 70% of the system's execution time⁽³⁻⁶⁾. Therefore, the pattern matching performance is crucial to an NIDS.

Many pattern matching algorithms have been proposed for NIDS, which can be implemented by software or hardware. Hardware-based implementations use special-purpose hardware, such as content addressable memory (CAM)^(7,8) and application specific integrated circuit (ASIC), to achieve high matching speeds by exploring the high level of parallelism available on devices. The major drawbacks with the hardware-based implementations are time-consuming and expensive in production. In contrast, software-based implementations provide better flexibility and programmability than hardware-based implementations, but fail to provide satisfactory performance.

In recent years, network processors (NPs) have emerged as a successful platform that has the performance and flexibility at the same time. Due to the limited on-chip memory (L1 cache) in a network processor, which is generally a few KBs, the required data structures of most existing pattern matching algorithms have to be accessed from the external memory (L2 memory) frequently. Since the latency of one L2 memory access is far larger than that of one L1 cache access, frequently accessing the external memory significantly degrades the throughput of a pattern matching algorithm. Sheu et al. proposed a hierarchical multi-pattern matching algorithm (HMA), which can reduce the amount of external memory accesses by constructing

small index tables for frequently accessed data⁽⁹⁾. However, the format of tables generated by the HMA is fixed, which makes the HMA unable to achieve good performance on different hardware architectures. Therefore, in this paper, we propose a scalable multi-pattern matching algorithm for intrusion detection systems. Similar to the HMA, the first-tier lookup table, which is also the most frequently accessed table, is small enough to be kept in the L1 cache. Most importantly, the size of the first-tier lookup table proposed in this paper can be adjusted according to the L1 cache size. Therefore, our proposed algorithm can provide fast matching speed in different hardware architectures.

The rest of this paper is organized as follows: Section 2 presents previous studies on pattern matching algorithms. Section 3 presents the proposed pattern matching algorithm. Section 4 presents the experimental results, a performance comparison, and a discussion. Finally, Section 5 concludes the paper.

2. Related Work

Numerous pattern matching algorithms have been proposed in the literature. These algorithms can be divided into two categories: single-pattern matching and multi-pattern matching. For single-pattern matching, each pattern is searched individually in a given packet. The Knuth-Morris-Pratt (KMP)⁽¹⁰⁾ and Boyer-Moore algorithms⁽¹¹⁾ are two of the most widely used single-pattern matching algorithms. For multi-pattern matching, more than one pattern (a pattern set) can be found simultaneously in a given packet. The Aho-Corasick (AC)⁽¹²⁾ and Wu-Manber (WM) algorithms⁽¹³⁾ are well-known multi-pattern matching algorithms.

The AC algorithm uses a finite state automaton to perform pattern matching, and has a time complexity of $O(n)$, where n is the length of the scanned text, regardless of the number of patterns and the lengths of patterns. The major drawback of the AC algorithm is that it requires a huge amount of memory to store the transition matrixes of the states. Tuck et al. proposed a modified version of AC algorithm, which uses a compressed pointer methodology to reduce the memory requirement⁽¹⁴⁾. According to the experimental results shown in their paper, the size of required memory can be reduced by 98% (from 53.1 MB to 1.09MB). However, it is still too large to store in the L1 cache of a network processor.

The HMA uses hierarchical tables to improve the

memory efficiency⁽⁹⁾. A small and simple first-tier table with 256 entries is designed to inspect each byte of a packet. The operation performed in the first-tier table is simple and deterministic. That is, once the currently inspected byte fails to get a match in the first-tier matching, this byte is guaranteed to be safe, and the next byte can be inspected. Otherwise, the L2 memory will be accessed to make sure if this byte is part of a pattern. To reduce the possibility of accessing the L2 memory, the authors proposed the frequent common-code searching (FCS) algorithm, which finds a minimum set of significant codes, denoted by F , to represent the pattern set. More specifically, for each pattern p , at least one character of p occurs in F . In addition, the cluster balancing strategy (CBS) was also proposed to reduce the time spent in the second-tier tables.

The major drawback of the HMA is that the format of tables is fixed. As a result, even though network processors are equipped with larger L1 caches, the HMA cannot fully utilize the L1 cache. This motivates us to design a scalable pattern matching algorithm for a broad variety of hardware architectures.

3. Proposed Multi-Pattern Matching Algorithm

3.1 Construction of the Required Lookup Tables

In our proposed algorithm, the first-tier table (H^1) can be adjusted according to the L1 cache size. The key idea of the proposed algorithm is to add one small and adjustable field in H^1 . The additional field is used to further reduce the possibility of accessing the L2 memory. In the HMA, once the currently inspected byte is matched in H^1 , the second-tier table (H^2), which is stored in the L2 memory, will be accessed to perform the exact match. However, in our proposed algorithm, the additional field is checked before accessing H^2 . The procedure of building the additional field is described as follows.

Firstly, a set of frequent common-codes F is generated as in the HMA. Then, for each element x in F , the next n bytes following x in all pattern in F are extracted to form a set N_x . An 8-bit vector M with exactly m bits set to 1 is used as a mask to extract m bits from all elements of N_x . As a result, the value of extracted bits ranges from 0 to 2^m-1 . Finally, a 2^m -bit bit vector VBV is generated by set the i -th bit to 1 if i belongs to the values of extracted bits.

Fig. 1. Example of VBV generation.

Figure 1 gives an example to explain how VBVs are generated. Suppose that the pattern set is {"aCount", "able", "ate"}. According to the HMA, the frequent common-code set is {'a'} since 'a' appears in all patterns. Thus, N_a is {'C', 'b', 't'}. Assuming that ASCII encoding is used and M is 11100000, which means m is 3, the first three bits will be extracted from each element in N_a , which include 010, 011, and 011. Because m is 3, the length of VBV will be 2^3 bits. In addition, the second (i.e., $(010)_2=2$) and the third (i.e., $(011)_2=3$) bit of VBV are set to 1. The VBV is stored in the entry of 'a' in H^1 .

The value of m is adjustable, and the length of VBV is determined by m . Thus, the size of H^1 can be controlled by m . It is obvious that the longer the VBV is, the more L2 memory accesses can be reduced. The value of m can be set according to the L1 cache size.

3.2 Pattern Matching Algorithm

Given a packet payload T , each byte is scanned sequentially. Suppose that the t -th byte of T ($T[t]$) is being

scanned. The $T[t]$ -th entry of H^1 is checked to see if $T[t]$ belongs to the frequent common-code set. If not, the scanning of $T[t]$ is completed, and can proceed to scan the next byte $T[t+1]$. If $T[t]$ -th entry of H^1 is not null, this means $T[t]$ belongs to the frequent common-code set. Before accessing H^1 as the HMA does, our proposed algorithm reads $T[t+1]$ and extracts m bits using M . Then, the extracted bits are treated as an index to check if the corresponding bit in VBV is equal to 1. If so, then H^2 will be accessed as the HMA does. Otherwise, the scanning of $T[t]$ is completed, and proceed to scan the next byte $T[t+1]$.

We use an example to explain how the pattern matching algorithm operates (Figure 2). Suppose that the pattern set is {"caT", "do.g", "lion", "a?oudad"}, the mask M is 11100000, and the value of m is 3. Obviously, the frequent common-code set F will be {'a', 'o'}. Following the procedure mentioned in Sect. 3.1, N_a is {'?', 'T'} and N_o is {'.', 'n', 'u'}. Then the VBV for 'a' and 'o' can be obtained. The *single-pid* field of H^1 is used to indicate if the common code is also a pattern. Since there is no one-byte pattern, all *single-pid* fields are set to 0. In addition, the *fid* field of H^1 is used to point to the corresponding H^2 if the entry belong to the frequent common-code set. Otherwise, it is set to 0 (or null). Given that the packet payload T ="ascaT", the details of pattern matching are listed below:

(a) $T[1]$ is 'a'. Since $H^1['a'].fid$ is not 0, this means 'a' is a frequent common code. Thus, $T[2]$ (i.e., 's'=01110011) is read and its first three bits is then extracted, which is 011 or 3 in decimal. The 3rd least significant bit of $H^1['a'].VBV$ (i.e., 0000110) is 0. The inspection of $T[1]$ is completed.

Fig. 2. Example of H^1 and H^2 constructed by the proposed algorithm.

(b) $T[2]$ is 's'. Since $H^1['s'].fid$ is 0, the inspection of $T[2]$ is completed.

(c) $T[3]$ is 'c'. Since $H^1['c'].fid$ is 0, the inspection of $T[3]$ is completed.

(d) $T[4]$ is 'a'. Since $H^1['a'].fid$ is 1, $T[5]$ (i.e., 'T'=01010100) is read and its first three bits is then extracted, which is 010 or 2 in decimal. The 2nd least significant bit of $H^1['a'].VBV$ (i.e., 0000110) is 1. $H^1['a'].fid$ is used to access the corresponding H^2 , and find a match.

(e) $T[5]$ is 'T'. Since $H^1['T'].fid$ is 0, the inspection of $T[5]$ is completed.

Through the above example, we can see that the proposed algorithm can avoid to access the L2 memory when processing $T[1]$ while the HMA cannot.

4. Experimental Results

In this section, we compare the matching speed of the proposed algorithm with that of the HMA. The pattern set used to evaluate the performance of pattern matching algorithms was extracted from Snort⁽¹⁵⁾, an open-source NIDS. Table 1 lists the statistics of the pattern set. The number of patterns extracted from Snort is 2,000. In order to study the impact of the size of pattern set on the matching speed, we generated nine pattern sets with different number of patterns, ranging from 200 to 1800. The patterns in each pattern set were randomly selected from the original pattern set, and the pattern length distribution of each pattern set is identical to Table 1.

We assume that the time required to execute a RISC instruction is one cycle, and so is to perform one L1 cache access. In addition, it requires more time to perform one L2 memory access, which is set to 100 cycles.

In order to evaluate the performance of algorithms for different types of traffic, two models of packet were used. In the first packet model, which will be denoted by Model 1 hereafter, each packet has a fixed length of 640 bytes and contains α patterns, which are randomly selected from the pattern set and inserted into random positions in the packet. In the second packet model, which will be denoted by Model 2 hereafter, all packets are 4 bytes long, and the lengths of patterns are thus no greater than 4 bytes. This

Table 1. The pattern length distribution.

Pattern length	≤ 4	≤ 8	≤ 12	≤ 16	> 16
Percentage (%)	12	28	38	47	53

Fig. 3. Average matching time with different pattern set sizes using the Model 1 traffic.

model is used to evaluate the best and the worst performance of algorithms.

Figure 3 shows the average number of cycles required to process one byte for the HMA and our proposed algorithm using the Model 1 traffic. Each result was obtained by processing one million packets. We can see that our proposed algorithm performs better than the HMA does for both $\alpha = 0$ and $\alpha = 4$. When $\alpha = 0$, no packet contains any patterns. As a result, both our proposed algorithm and the HMA can achieve their best performance. However, our proposed algorithm can provide 37% better performance than the HMA in average. In the case of $\alpha = 4$, our proposed algorithm still outperforms the HMA by 15%. As the value of α increases, the performance improvement with our proposed algorithm will decrease. This is because more L2 memory accesses are required, which mitigates the benefit brought by the VBV. Practically, most packets do not contain any pattern. Thus, our proposed algorithm can achieve significant performance improvement in practical uses.

Table 2 shows the simulation results using the Model 2 traffic. Recall that the length of packets in this model is fixed at 4 bytes. Thus, all possible packets (i.e., 2^{32} packets) can be generated to thoroughly evaluate the performance of both algorithms. For the best case, inspecting one byte only needs one L1 cache access and

Table 2. Average matching time (in cycles) using the Model 2 traffic.

	HMA	Proposed algorithm
Worst case	439	439
Best case	7	7
Average case	28.85	12.61

some simple operations, which leads to a total of 7 cycles. It is obvious that there is no difference between both algorithms in the best case. As for the worst case, both algorithms need to access the L2 memory for reading the H^2 table. Our proposed algorithm needs to deal with the extra comparison with the VBV, and thus requires more time than the HMA. However, in the average case, our proposed algorithm can reduce 56% of execution time. This indicates that the VBV can effectively reduce the possibility of accessing the L2 memory.

5. Conclusions

In this paper, we proposed a multi-pattern matching algorithm for intrusion detection systems. Using a small and adjustable lookup table to store frequently accessed data, the proposed algorithm can reduce the possibility of accessing the external memory, and significantly improve the matching performance. Simulation results show that the proposed algorithm can reduce the matching time by up to 37% as compared with the HMA. Most importantly, the proposed algorithm is easy to implement and is applicable to different hardware architectures.

Acknowledgment

This work was supported in part by the High Speed Intelligent Communication (HSIC) research center, Chang Gung University, Taiwan, and by grants from the Ministry of Science and Technology of Taiwan (NSC-100-2221-E-182-050) and the Chang Gung Memorial Hospital (BMRP 942).

References

- (1) M. Handley, V. Paxson and C. Kreibich : "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-end Protocol Semantics," in Proceedings of the 9th USENIX Security Symposium, pp. 2-18, 2001.
- (2) C. Kruegel, F. Valeur, G. Vigna and R. Kemmerer : "Stateful Intrusion Detection for High-speed Networks," in Proceedings of IEEE Symposium on Security and Privacy, pp. 285-293, 2002.
- (3) S. Antonatos, K. G. Anagnostakis, and E. P. Markatos : "Generating Realistic Workloads for Network Intrusion Detection Systems," in Proceedings of ACM Workshop on Software and Performance, pp. 207-215, 2004.
- (4) J. B. Cabrera, J. Gosar, W. Lee and R. K. Mehra : "On the Statistical Distribution of Processing Times in Network Intrusion Detection," in Proceedings of IEEE Conference on Decision and Control, Vol. 1, pp. 75-80, Dec. 2004.
- (5) M. Roesch : "Snort – Lightweight Intrusion Detection for Networks," in Proceedings of the 13th Systems Administration Conference, pp. 229-238, 1999.
- (6) T. Sato and M. Fukase : "Reconfigurable Hardware Implementation of Host-based IDS," in Proceedings of the 9th Asia-Pacific Conference on Communication, pp. 849-853, Sept. 2003.
- (7) C. R. Meiners, J. Patel, E. Norige, E. Torng, and A. X. Liu : "Fast Regular Expression Matching Using Small TCAMs for Network Intrusion Detection and Prevention Systems," in Proceedings of the 19th USENIX Security Symposium, 2010.
- (8) I. Sourdis and D. Pnevmatikatos : "Pre-decoded CAMs for Efficient and High-Speed NIDS Pattern Matching," in Proceedings of the 12th Annual IEEE Symposium on Field Programmable Custom Computing Machines, 2004.
- (9) T.F. Sheu, N.F. Huang and H.P. Lee : "Hierarchical Multi-pattern Matching Algorithm for Network Content Inspection," Information Sciences, Vol. 178, Issue 14, pp. 2880-2898, 2008.
- (10) D.E. Knuth, J. Morris, and V. Pratt : "Fast Pattern Matching in Strings," SIAM Journal on Computing, Vol. 6, No. 2, pp. 127-146, 1977.
- (11) 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.
- (12) A.V. Aho and M.J. Corasick : "Efficient String Matching: An Aid to Bibliographic Search," Communications of the ACM, Vol. 18, Issue 6, pp. 333-340, 1975.
- (13) R.S. Boyer and J.S. Moore : "A Fast String Searching Algorithm," Communications of the ACM, Vol. 20, Issue 10, pp. 762 - 772, 1977.
- (14) N. Tuck, T. Sherwood, B. Calder, and G. Varghese : "Deterministic Memory-efficient String Matching Algorithms for Intrusion Detection," in Proceedings of IEEE INFOCOM, Vol. 4, pp. 2628-2639, 2004.
- (15) Snort, <http://www.snort.org/>