Fast Pattern Matching Using Dynamically Partitioned AC-DFAs

Chun-Liang Lee^{a,*}, Bo-En Cheng^a

Department 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: cllee@mail.cgu.edu.tw

Abstract

Pattern matching is a critical component of deep packet inspection (DPI). To provide fast matching speed, deterministic finite automata constructed with the Aho-Corasic algorithm (AC-DFA) have been widely used. The matching speed of an AC-DFA is highly affected by the memory size used to store the DFA. In this paper, we propose a fast pattern matching algorithm using partitioned AC-DFAs. Most pattern matching algorithms using AC-DFAs partition an AC-DFA statically. Our proposed algorithm, however, dynamically partitions an AC-DFA according to inspected packet payloads. Simulation results show that our proposed algorithm achieves a higher matching speed (from 15% to 176%) than other two pattern matching algorithms that use partitioned AC-DFAs.

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

1. Introduction

Security is an important issue in today's Internet. Traditional firewalls provide basic protection by examining packet headers. To deal with advanced attacks, network intrusion detection systems (NIDSs) have been widely deployed⁽¹⁾. NIDSs can be classified into two categories: anomaly-based and signature-based. Anomaly-based NIDSs detect abnormal behaviors by monitoring and analyzing network activities⁽²⁾. Signature-based NIDSs scan packet payloads to determine if packets contain malicious contents, which are also called patterns or signatures. Since it is hard to define abnormal behaviors, signature-based NIDSs has the advantage of precisely detecting known attacks, and thus they have been studied extensively in the literature. In this paper, we also focus on signature-based NIDSs.

Pattern matching is a key factor influencing the

performance of an NIDS since it consumes a significant portion of system execution time^(3,4). Pattern matching algorithms can be implemented with hardware or software. Hardware-based implementation can achieve high speed via special-purposed devices⁽⁵⁻⁷⁾, but generally requires more time and cost to develop. In contrast, software-based implementation provides high flexibility and programmability⁽⁸⁻¹³⁾. Thus, in this paper, we focus on pattern matching algorithms for software-based NIDSs.

Software-based NIDSs can use graphics processing units (GPUs) or general-purpose central processing units (CPUs) to execute pattern matching tasks. Although the computing power of GPU has increased rapidly, GPU-based pattern matching algorithms consume significantly more energy and require higher cost compared with CPU-based ones. Moreover, most CPUs now supports single-instruction multiple-data (SIMD) instructions that can be used to speed up pattern matching. Our proposed algorithm uses a strategy similar to that used in the head-body matching (HBM) algorithm⁽¹²⁾ and the flexible head-body matching (FHBM) algorithm⁽¹³⁾. First, with a given pattern set, our proposed algorithm builds a deterministic finite automaton (DFA) using the Aho-Corasic (AC) algorithm⁽¹⁴⁾ (hereafter referred to as AC-DFA). Then, the AC-DFA is partitioned into two parts: head and body. Both the HBM and FHBM algorithms statically partition an AC-DFA. Our proposed algorithm, however, dynamically partitions an AC-DFA according to the numbers of accesses of all states, and thus can provide a higher matching speed than other algorithms.

The remainder of this paper is structured as follows. In Section 2, we summarize the related work in the literature. In Section 3, we describe our proposed algorithm in detail. Experimental results are presented and discussed in Section 4. Finally, Section 5 concludes the paper.

2. Related Work

The AC algorithm is one of the most well-known pattern matching algorithms. This algorithm constructs a DFA for finding all occurrences of a given pattern set in an input text. The input is inspected byte by byte in one pass. Thus, the AC algorithm is insensitive to pattern sets as well as the content being inspected. Because of its deterministic worst-case performance, the AC algorithm has attracted much attention, with a large number of researchers searching for ways to reduce its significant memory requirement. Tuck *et al.*⁽¹⁵⁾ found that most entries in a state transition table do not store valid transitions. They used a bitmap and variable-length list of transitions to successfully reduce required memory and to produce better throughput.

Liu *et al.*⁽¹⁶⁾ proposed a general DFA model called DFA with extended character-set (DFA/EC), which can reduce the number of states by removing part of each state and incorporating it with the next input character. In addition, they proposed a method to encode the complementary state into a single bit for reducing the size of the transition table. Therefore, inspecting each byte in packet payloads requires only one memory access.

Yang and Prasanna⁽¹²⁾ found that AC-DFA throughput can significantly degrade for large pattern sets and input streams with high match ratios. To solve the problem caused by memory access overhead, they have proposed a head-body finite automaton (HBFA) and an HBM algorithm. The HBFA consists of a head DFA (H-DFA) and body nondeterministic finite automaton (B-NFA). The H-DFA remains the same structure as an AC-DFA, but only states close to the start state are included. Thus, the number of states in the H-DFA is much less than that in the original AC-DFA, and the average access probability of a state is higher. The B-NFA is stored in a data structure that can be used to execute the matching process by the SIMD operations. Their test results indicate that the HBM algorithm can achieve up to 7 times higher throughput than the AC algorithm.

In the HBM algorithm, an AC-DFA is partitioned according to a pre-defined depth value. Lee *et al.*⁽¹³⁾ found that even in cases where a good depth value is selected, the HBM algorithm may still fail to achieve good throughput due to the way it partitions the AC-DFA. They proposed a pattern matching algorithm called flexible head-body matching (FHBM) algorithm that partitions head and body parts based on head size (i.e., the number of states in the H-DFA). The FHBM algorithm can construct more efficient

Fig. 1. AC-DFA for the example pattern set.

HBFA, resulting in higher throughput compared to the HBM algorithm.

3. Proposed Pattern Matching Algorithm

Given pattern set $S = \{\text{accord, ace, ache, acute, bad, bed, in, inner}\}$, the AC-DFA can be constructed as shown in Fig. 1. A circle represents a state, and a double circle represents a matching state. If a matching state is reached, it indicates that at least one pattern has been found. Fig. 1 omits all backward transitions for clarity. The number beside each state will be explained later.

Assume a maximum head size of 10 states. According to the HBM algorithm, the AC-DFA head and body parts after partitioning are shown in Fig. 2. Since the HBM algorithm partitions the AC-DFA based on the pre-defined depth, only states at depths 0, 1, and 2 can be included in the head part due to the limited head size. The FHBM algorithm, however, can put more states in the head part to fully utilize the head size. Fig. 3 shows the head and body parts after partitioning using the FHBM algorithm. The key idea of our proposed algorithm is to select appropriate states for inclusion in the head part based on the access probability of a state rather than the depth of a state. More specifically, the higher probability a state will be accessed, the higher priority the state will be included in the head part. Since it is unlikely that contents in packet payloads are randomly distributed symbols, and network attacks tend to generate intrusive packets to target hosts in a short period of time, we were motivated to propose our algorithm, which is described below.

Recall that the basic idea behind the HBM algorithm is to keep the states near the start state (i.e., state 1 in Fig. 1)

Fig. 2. Head and body parts for the example AC-DFA using the HBM algorithm.

Fig. 3. Head and body parts for the example AC-DFA using the FHBM algorithm.

in the head part for efficient access since state transition tables used in H-DFA can be looked up quickly. The implicit assumption of the HBM algorithm is that the lower depth a state is at, the higher probability the state will be accessed. However, this assumption is not true unless contents in packet payloads are randomly distributed symbols. In our proposed algorithm, for each state, we use an additional variable to count the number of accesses for the state in a period of time. Then, our proposed algorithm partitions the AC-DFA according to the head size and the access counters. A greedy algorithm can be used to achieve the following two goals:

- (a) The number of states in the head part is as close as possible to the pre-defined head size.
- (b) The summation of the counter values for all states in the head part is maximum.

We use an example to illustrate how our proposed algorithm works. Let the number beside a state in Fig. 1 be the number of accesses for the state in a period of time. Our proposed algorithm starts from the start state (i.e., state 1).

Fig. 4. Head and body parts for the example AC-DFA using our proposed algorithm.

Table 1. Hardware configuration for the experiments.

Component		Specification
Main board	Brand/Model	SuperMicro/X9DRG-QF
CPU	Brand/Model	Intel/Xeon E5-2620 v2
	Number of cores	6
	Frequency	2.1 GHz
	L1 cache size	192 KB
	L2 cache size	1,536 KB
	L3 cache size	15,360 KB
Main memory	Type	DDR3
	Size	48 GB
	Frequency	1,600 MHz

Since the maximum head size is 10 states, state 1 can be added to the head part, and more states are allowed to be added. There are three candidate states (i.e., states 2, 3, 4) to be considered. Since the access counter of state 2 is the largest among these three states, state 2 is selected to be added to the head part. The head part now contains 2 states, and eight more states can be added. Candidate states are states 3, 4, and 5 since state 5 can be reached via state 2. Following the same rule, state 5 has the largest access counter, and thus it is added to the head part. Then, states 11, 10, 12, 9, 17, 3, and 4 are added to the head part sequentially. Fig. 4 gives the final head and body parts after partitioning using our proposed algorithm. It is easy to see that our proposed algorithm can partition an AC-DFA into head and body parts which satisfy the two goals mentioned earlier. We can also see that the proposed algorithm is simple and thus can partition an AC-DFA very quickly.

4. Experimental Results

Table 1 shows the hardware configuration used in our experiments. Source codes were compiled using GCC 4.8.4.

Table 2. Pattern set statistics.

Item	Value
Number of patterns	1,000
Number of characters	10,910
Number of depths	95
Number of states	8,483

Table 3. Pattern length distribution.

Pattern length	Number of patterns
1-5	222
6-10	361
11-15	195
16-20	87
20-	135

Table 4. Number of states at different depth ranges.

Depths	Number of states
≤ 1	128
≤ 2	636
≤ 3	1,306
≤ 4	2,078
≤ 5	2,859
≤ 6	3,581
≤ 7	4,230
≤ 8	4,806
≤ 9	5,310
≤ 10	5,732

The operating system was 64-bit Ubuntu 14.04 (kernel version 4.4.0). The pattern set from Snort⁽¹⁷⁾ was used for performance evaluation. The pattern set statistics and the pattern length distribution are listed in Tables 2 and 3, respectively. The input data stream was generated by inserting randomly chosen patterns from the pattern set into a plain text which was an HTML-formatted King James Bible. The match ratio of the generated input data stream was 0.01. For a given pattern set, the match ratio of an input data stream refers to the proportion of the length of malicious content to the length of the input data stream. A substring in the input data stream is considered malicious if it is a significant prefix string of any pattern in the pattern set. A prefix string is significant if it covers a significant part (e.g., > 80%) of the full string. Since the HBM, FHBM, and proposed algorithms were designed for multi-core platforms, we created four threads, with each performing its own pattern matching task using the shared HBFA. All data represent averages from 10 simulations.

Fig. 5. Throughput value plotted against head size.

Table 5. Depth information of the head part (head size = 1200 states).

Algorithm	Minimum depth	Maximum depth
HBM	2	2
FHBM	2	3
Proposed	1	28

A comparison of the HBM, FHBM and our proposed algorithm throughputs for different head sizes is presented in Fig. 5. We can see that throughput value increased with head size for both the FHBM and our proposed algorithms. This is because that both algorithms can fully utilize the head size. In contrast, the HBM algorithm partitions an AC-DFA into head and body parts according to a pre-defined head size to determine the maximum depth of states that can be added into the head part. Since the combined number of states at depths 0, 1 and 2 was 636 (see Table 4), only the states at depths 0 and 1 could be moved to the head part when the head size were 500 and 600. This explains why the throughput values were identical for the head sizes of 500 and 600 states. Similar situation can be observed for head sizes between 700 to 1,200 states since only the states at depths 0, 1 and 2 could be moved to the head part.

Compared with the HBM and FHBM algorithms, our proposed algorithm achieved higher throughput values for all head sizes. For a head size of 1,200 states, our proposed algorithm achieved up to 589 MB/s, which was 84% and 15% higher than the HBM (321 MB/s) and FHBM algorithms (512 MB/s), respectively. Although both the FHBM and our proposed algorithms can fully utilize the head size, our proposed algorithm uses a different way to partition an AC-DFA by considering the numbers of accesses for all states. Table 5 lists the depth information

for all states in the head part after partitioning when the head size was 1,200 states. The minimum/maximum depth is the smallest/largest depth value among all leaf states in the head part. Since the HBM algorithm partitions an AC-DFA based on depth, the minimum and maximum depth values were identical. The FHBM algorithm could add states at depth larger than the HBM algorithm. This explains why the maximum depth value of the FHBM algorithm was larger than the minimum depth value by one. Our proposed algorithm, however, partitions an AC-DFA according to the numbers of accesses of all states. Thus, the minimum depth and maximum depth values did not have any relationship, while the other two algorithms did. In this experiment, the minimum depth and maximum depth values were 1 and 28, respectively.

5. Conclusions

In this paper, we proposed an efficient pattern matching algorithm using partitioned AC-DFAs. Different from the HBM and FHBM algorithms, our proposed algorithm dynamically partitions an AC-DFA based on the numbers of accesses of all states, and thus can provide higher throughput than the other two algorithms. According to our results, when the head size is 1,200 states and the match ratio is 0.01, the proposed algorithm achieved up to 84% and 15% higher throughputs than the HBM and FHBM algorithms, respectively.

Acknowledgment

This work was supported in part by the High Speed Intelligent Communication (HSIC) Research Center of Chang Gung University, Taiwan, and by grants from the Ministry of Science and Technology of Taiwan (MOST 106-2221-E-182-017) and Chang Gung Memorial Hospital (BMRP 942).

References

- (1) A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B.D. Payne : “Evaluating Computer Intrusion Detection Systems: A Survey of Common Practices”, *ACM Computing Surveys*, Vol. 48, Article 12, 2015.
- (2) A. Avritzer, R. Tanikella, K. James, R.G. Cole, and E.J. Weyuker : “Monitoring for Security Intrusion Using Performance Signatures”, in *Proceedings of the 1st Joint WOSP/SIPEW International Conference on Performance Engineering*, pp. 93–104, 2010.
- (3) 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 the Conference on Decision and Control*, pp. 75-80, 2004.
- (4) S. Antonatos, K.G. Anagnostakis, and E.P. Markatos : “Generating Realistic Workloads for Network Intrusion Detection Systems”, *ACM SIGSOFT Software Engineering Notes*, Vol. 29, pp. 207-215, 2004.
- (5) H.-J. Tsai, K.-H. Yang, Y.-C. Peng, C.-C. Lin, Y.-H. Tsao, M.-F. Chang, and T.-F. Chen : “Energy-Efficient TCAM Search Engine Design Using Priority-Decision in Memory Technology”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 25, pp. 962-973, 2017.
- (6) D.F. Bacon, R. Rabbah, and S. Shukla : “FPGA Programming of the Masses”, *Communications of the ACM*, Vol. 56, pp. 56-63, 2013.
- (7) O. Erdem : “Tree-Based String Pattern Matching on FPGAs”, *Computers & Electrical Engineering*, Vol. 49, pp. 117–133, 2016.
- (8) Y.-S. Lin, C.-L. Lee, and Y.-C. Chen : “Length-Bounded Hybrid CPU/GPU Pattern Matching Algorithm for Deep Packet Inspection”, *Algorithms*, Vol. 10, Article Number 16, 2017.
- (9) A. Bremler-Barr, D. Hay, and Y. Koral : “CompactDFA: Scalable Pattern Matching Using Longest Prefix Match Solutions”, *IEEE/ACM Transaction on Networking*, Vol. 22, pp. 415-428, 2014.
- (10) C.-L. Lee, Y.-S. Lin, and Y.-C. Chen : “A Hybrid CPU/GPU Pattern-Matching Algorithm for Deep Packet Inspection”, *PLoS ONE* Vol. 10, e0139301, 2015.
- (11) Y.-S. Lin, C.-L. Lee, and Y.-C. Chen : “A Capability-Based Hybrid CPU/GPU Pattern Matching Algorithm for Deep Packet Inspection”, *International Journal of Computer and Communication Engineering*, Vol. 5, pp. 321-330, 2016.
- (12) Y.-H. Yang and V.K. Prasanna : “Robust and Scalable String Pattern Matching for Deep Packet Inspection on Multicore Processors”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, pp. 2283-2292, 2013.
- (13) C.-L. Lee and T.-H. Yang : “A Flexible Pattern-Matching Algorithm for Network Intrusion Detection Systems Using Multi-Core Processors”, *Algorithms*, Vol. 10, Article Number 58, 2017.
- (14) A.V. Aho and M.J. Corasick : “Efficient String

Matching: An Aid to Bibliographic Search”,
Communications of the ACM, Vol. 18, pp. 333-340
1975.

- (15) N. Tuck, T. Sherwood, B. Calder, and G. Varghese :
“Deterministic Memory-Efficient String Matching
Algorithms for Intrusion Detection”, in Proceedings of
IEEE INFOCOM, pp. 333-340, 2004.
- (16) Liu, C.; Pan, Y.; Chen, A.; Wu, J. A DFA with extended
character-set for fast deep packet inspection. IEEE
Transactions on Computers 2014, 63, 1925–1937.
- (17) Snort.Org, Available Online: <http://www.snort.org>
(accessed on 1 December 2017).