High-Speed Pattern Matching Algorithm with CPU/GPU Cooperation for Network Intrusion Detection Systems

Chun-Liang Lee^{a,b,*}, Guan-Zhang Chen^a, Kai-Ping Lu^a

^aDepartment of Computer Science and Information Engineering, School of Electrical and Computer Engineering, College of Engineering, Chang Gung University, Kwei-Shan, Tao-Yuan 33302, Taiwan

^bStroke Center and Department of Neurology, Chang Gung Memorial Hospital, Linkou, New Taipei 20401, Taiwan

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

Abstract

Network intrusion detection systems (NIDSs) have been widely deployed in the Internet to protect Internet-enabled devices from malicious attacks by performing deep packet inspection (DPI). Pattern matching plays an important role in DPI, and consumes a significant portion of system execution time for NIDSs. In this paper, we propose a high-speed pattern matching algorithm with CPU/GPU cooperation. Incoming packets are first inspected by the CPU to quickly filter out suspicious packets that may contain malicious patterns. Then the GPU, which has superior parallel computing power, takes over to determine if a suspicious packet does contain malicious patterns. In addition, in our proposed algorithm, the GPU does not have to inspect the entire payload of a packet, but instead can skip the partial packet payload that has been inspected by the CPU. Through the cooperation between a CPU and GPU, our proposed algorithm can achieve higher pattern matching speeds than other algorithms. Simulation results show that even in the case that all packets contain malicious patterns, our proposed algorithm can achieve a matching speed of 15 Gbps.

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

1. Introduction

The number of Internet-enabled devices has increased significantly during recent years. This trend makes network security an important issue in today's Internet. Network intrusion detection systems (NIDSs)⁽¹⁾ provide protection against malicious attacks by performing deep packet

inspection (DPI), which scans each incoming packet to determine whether it contains malicious content, defined as patterns or signatures. More specifically, DPI executes a pattern matching algorithm to determine if a packet contains any patterns in a pre-defined pattern set.

According to the literature, pattern matching is a time-consuming task, and also a key factor influencing the performance of an NIDS^(2,3). Pattern matching algorithms can be divided into two categories: hardware-based algorithms and software-based algorithms. Hardware-based algorithms⁽⁴⁻⁶⁾ utilize special-purposed devices such as field programmable gate arrays (FPGAs), application-specific integrated circuits (ASICs), and content addressable memory (CAM) to achieve high matching speeds. However, this type of algorithms generally requires more time and cost to develop. In contrast, software-based algorithms⁽⁷⁻¹²⁾ utilize general purpose processors such as central processing units (CPUs) or graphics processing units (GPUs) for pattern matching, and provide high flexibility and programmability. Thus, in this paper, we focus on software-based pattern matching algorithms.

Compared to CPUs, GPUs have superior parallel computing power, and thus have attracted much attention in recent years as an alternative for applications with high computational demands. A number of pattern matching algorithms using GPUs can be found in the literature. Most existing algorithms use only GPUs for executing pattern matching. CPUs only take in charge of transferring packets to GPUs. Although GPUs offer better parallel computing power than CPUs, CPUs still are capable of contributing computing power to the pattern matching task. In our previous work⁽⁹⁾, we proposed a hybrid CPU/GPU pattern-matching algorithm (HPMA), which can achieve higher matching speeds than other algorithms with only

CPU or only GPUs. In this paper, we propose an improved version of the HPMA. The key idea of our proposed algorithm is to reduce the amount of data that a GPU has to process by enhancing the cooperation between a CPU and GPU, and thus our proposed algorithm can achieve higher matching speeds than the HPMA.

The remainder of this paper is organized 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

Pattern matching algorithms can be categorized into two kinds: single-pattern and multi-pattern matching. Single-pattern matching algorithms search one pattern at a time, while multi-pattern matching algorithms can search multiple patterns. Since DPI has to examine a packet to determine whether it contains any patterns in a pattern set, multi-pattern matching algorithms are more suitable for DPI. The Aho-Corasick (AC) algorithm⁽¹³⁾ is one of the most well-known multi-pattern matching algorithms. This algorithm constructs a deterministic finite automaton (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. Snort⁽¹⁴⁾, a free and open source NIDS, adopted the AC algorithm for pattern matching.

Since the pattern matching algorithm presented in this paper is an improved version of the HPMA, we briefly review the HPMA here. The HPMA decomposes the pattern matching task into two parts: pre-filtering and full pattern matching. All incoming packets are initially processed by a pre-filtering algorithm executed by the CPU. The pre-filtering algorithm was designed to be able to quickly identify non-malicious packets (i.e., packets that do not contain any patterns). Packets that are not non-malicious are called suspicious packets, since these packets may (but not necessarily) contain patterns. Suspicious packets are transferring by the CPU to the GPU for full pattern matching using the AC algorithm. The pre-filtering algorithm utilizes four tiny tables to filter out suspicious packets. According to the results shown in the paper, the required pre-filtering memory size were between 13.2 and 21.8 KB, making it possible for most memory access tasks to be performed using CPU caches, thereby enhancing pre-filter speed.

Fig. 1. Architecture of our proposed algorithm.

3. Proposed Pattern Matching Algorithm

As mentioned previously, the HPMA initially inspects a packet with the pre-filter algorithm, which can only determine if the packet is suspicious. Suspicious packets should be transferred to the GPU for full pattern matching. If the suspicious packet ratio (i.e., the number of suspicious packets to the number of all packets) is too high, the pre-filtering algorithm executed by the CPU may become the throughput bottleneck of the system. More specifically, all packets have to be inspected by the pre-filtering algorithm, but a large proportion of packets should be transferred to the GPU for further inspection, which means that the pre-filtering algorithm falls to reduce the GPU workload. Thus, in this paper, we aim to enhance the contribution made by the pre-filtering algorithm.

Figure 1 depicts the architecture of our proposed algorithm. Incoming packets are stored in the main memory, indicated as host memory in the figure, and inspected by the pre-filtering algorithm. If a packet is detected that may contain malicious patterns, it is transferred to the memory used by the GPU, indicated as device memory. Different from the HPMA, our proposed algorithm transfers not only the suspicious packet, but also the information about the suspicious pattern found by the HPMA. More specifically, the HPMA determines if a packet is suspicious using four tiny tables, which store a set of subpatterns generated from the original pattern set. For each pattern p_i in the pattern set, there must be one element f_j in the subpattern set such that f_j is a substring of p_i . Thus, in addition to the packet, our proposed algorithm also transfers an offset value indicating the beginning location of the possible pattern to the GPU. When the GPU performs the full pattern matching algorithm, it does not need to inspect the packet from the first byte, but from the offset value sent by the pre-filtering algorithm, since the pre-filtering algorithm can guarantee that the content before the offset value contains no patterns.

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
Main memory	Type	DDR3
	Size	32 GB
	Frequency	1,600 MHz
GPU	Brand/Model	EVGA/NVIDIA GTX 780 Ti
	Number of cores	2,880
	Memory	3 GB GDDR5
NIC	Brand/Model	Intel/X540-T2

Table 2. Simulation parameters.

Parameter	Value
Packet length	1,500 bytes
Blocks per grid	15
Threads per block	512
Concurrent processes	5

Matching results are stored in the device memory, and finally be transferred back to the host memory in a batch to reduce the transferring time.

Since the offset value is only two bytes long, our proposed algorithm does not incur a significant overhead on transferring packets from the CPU to the GPU. Compared with the HPMA, our proposed algorithm can reduce the amount of data that the GPU have to process, and thus can achieve higher matching speeds than the HPMA.

4. Experimental Results and Discussion

Table 1 shows the hardware configuration used in our experiments. The simulation parameters are listed in Table 2. We used five concurrent processes to execute the pre-filtering algorithm. For a fair comparison, the AC algorithm also used five processes to process packets. The operating system was 64-bit Ubuntu 14.04 (kernel version 4.4.0). The pattern set from Snort⁽¹⁴⁾ was used for performance evaluation. The number of patterns is 1,288. A randomly chosen pattern inserted at a random packet position served as an intrusive packet.

A comparison of the CPU and GPU versions of the AC algorithm, the HPMA and our proposed algorithm throughputs for different intrusive packet percentages are presented in Fig. 2. When the percentage of intrusive

Fig. 2. Throughput value plotted against intrusive packet percentage.

Fig. 3. Percentage of data in packets inspected by the GPU plotted against intrusive packet percentage.

packets was between 0% and 30%, both the HPMA and our proposed algorithm achieved a throughput of 20 Gbps, which is the maximum speed that can be generated with Intel X540-T2 NIC. The CPU version of the AC algorithm had the lowest throughput among all algorithms. This is because the state transition tables used by the AC algorithm consumed too much memory, leading to longer memory access time for performing state transitions. The GPU version of the AC algorithm achieved nearly three times higher throughput values than the CPU version of the AC algorithm. This can be explained by the superior parallel computing power offered by GPU.

When 40% or more of the packets were intrusive, HPMA throughput started to decline. In contrast, the throughput of our proposed algorithm remained at 20 Gbps for intrusive packet percentages between 0% and 60%. Recall that all packets should be inspected by the pre-filtering algorithm executed by the CPU. For high

intrusive packet percentages, it means that many packets were suspicious packets and should be transferred to the GPU for further inspection. In other words, the pre-filtering algorithm would fail to achieve its original design goal, since it could not reduce enough number of packets that should be processed by the GPU. In the HPMA, the GPU had to inspect every packet from the beginning. In contrast, our proposed algorithm could utilize offset values sent by the pre-filtering algorithm to reduce the amount of data that the GPU should inspect. As a result, the pre-filtering algorithm not only filtered out suspicious packets, but also diminished the GPU workload.

Fig. 3 shows the percentages of data in packets inspected by the GPU for different percentages of intrusive packets. For the HPMA, percentage of data in packets that should be inspected by the GPU was slightly larger than intrusive packet percentage. This is because we used a fixed packet length. In addition, a packet that does not contain any patterns might be determined as a suspicious packet by the pre-filtering algorithm. For our proposed algorithm, depending on the offset values sent by the pre-filtering algorithm, the GPU could skip partial data of a packet. This explains why our proposed algorithm achieved higher throughput values than the HPMA.

5. Conclusions

In this paper, we proposed a high-speed pattern matching algorithm with CPU/GPU cooperation. Each packet is initially inspected by a pre-filtering algorithm to quickly determine whether it may contain malicious patterns. Suspicious packets are then transferred to the GPU for full pattern matching. Our proposed algorithm can reduce the amount of data that the GPU has to inspect. According to our results, our proposed algorithm achieved 20 Gbps of matching speed when intrusive packet percentages were 60% or less.

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 107-2221-E-182-029) 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) 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.
- (3) 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.
- (4) 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.
- (5) D.F. Bacon, R. Rabbah, and S. Shukla : “FPGA Programming of the Masses”, *Communications of the ACM*, Vol. 56, pp. 56-63, 2013.
- (6) O. Erdem : “Tree-Based String Pattern Matching on FPGAs”, *Computers & Electrical Engineering*, Vol. 49, pp. 117–133, 2016.
- (7) 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.
- (8) A. Bremner-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.
- (9) 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.
- (10) 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.
- (11) 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.

- (12) 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.
- (13) 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.
- (14) Snort.Org, Available Online: <http://www.snort.org> (accessed on 1 November 2018).