A Capability-Based Hybrid CPU/GPU Pattern Matching Algorithm for Deep Packet Inspection

Yi-Shan Lin[1], Chun-Liang Lee[2*], Yaw-Chung Chen[1]

[1] Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan.
[2] Department of Computer Science and Information Engineering, School of Electrical and Computer Engineering, College of Engineering, Chang Gung University, Taoyuan 33302, Taiwan.

* Corresponding author. Tel: +886-3-2118800 ext. 5196; email: cllee@mail.cgu.edu.tw
Manuscript submitted October 13, 2015; accepted December 26, 2015.
doi: 10.17706/ijcce.2016.5.5.321-330

Abstract: Network applications have been developed quickly during recent years, and communications between these applications involve a large quantity of data transfer through high speed networks. Deep packet inspection (DPI) becomes indispensable to ensure network application-aware security. One of the DPI services is the signature-based network intrusion detection system (NIDS), in which the implementation on software platforms has become a trend due to the advantages of high programmability and low cost. Recently, the graphic processing units (GPU) is commonly used to accelerate the packet processing because of its superior parallel processing power. Since delivering all packets to GPU causes high data transfer latency and consequently restricts the overall performance, our previous study proposed a mechanism, HPMA, to reduce the effect of transfer bottleneck and achieve higher processing speed. In this paper, we introduce an enhancement of HPMA, a capability-based hybrid CPU/GPU pattern matching algorithm (CHPMA). A preliminary experiment shows that the CHPMA not only performs as efficient as the HPMA in most cases, but also obtains higher performance gain than the HPMA under unfavorable conditions.

Key words: Network security, pattern matching algorithm, deep packet inspection (DPI), intrusion detection system (IDS), general-purpose graphics processing unit (GPGPU), compute unified device architecture (CUDA).

1. Introduction

Nowadays computer networks are widely used for human communication. Since a substantial quantity of data is transmitted during communication, network security is thus highly concerned. 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 completely ensure security, especially from the application layer perspective [1]. Hence, deep packet inspection (DPI) has become the focus in the application-aware security.

In DPI, the network intrusion detection systems (NIDSs) have been developed to examine the incoming packets. The signature-based NIDS is one kind of NIDS that determines whether the packet payloads contain malicious contents, which are called “signatures” or “patterns” in the NIDSs terminology. When these patterns are detected, the system either generates alert messages or drops the packet in order to protect other systems in the network. In a signature-based NIDS, the pattern matching algorithm is the core and time-consuming process for inspecting the packets. Literatures [2] indicated that it consumes approximately 70% execution time of the system; thus, the pattern matching algorithm dominates the NIDSs performance.

Pattern matching algorithms can be implemented by software or hardware. Implementations based on special-purpose hardware such as field programmable gate array (FPGA) [3], content addressable memory (CAM) [4] and application specific integrated circuit (ASIC) [5] were commonly used. Depends on the high-degree-parallelism of hardware devices, fast matching speeds can be achieved; however, the ASIC design and manufacture are expensive, and the FPGA programing is difficult [6]. On the other hand, software-based implementations on general-purpose processors (GPPs) [7] such as Intel x86 processors, provide lower cost and more scalability than hardware-based implementations. Thus, this study focuses on the software-based algorithm design.

In terms of software implementations, traditional approaches in which all incoming packets are inspected by central processing units (CPUs) have become inadequate for satisfying the required processing speed in high speed networks. Taking advantage of other processing units is one solution for acceleration. Graphical processing units (GPUs), which have parallel processing power superior to CPUs, are suitable candidates to provide higher processing speed. Nevertheless, sending all packets directly to GPU may not achieve the optimal efficiency due to the bottleneck associated with data transfer between CPU and GPU via peripheral component interconnect express (PCIe) channels. Therefore, collaboration between CPU and GPU becomes a new direction for NIDS performance improvement.

We proposed a hybrid CPU/GPU pattern matching algorithm (HPMA) [8] in our previous study. The HPMA first filtered the incoming packets by the CPU (called as "pre-filtering") to reduce the effect of transfer bottleneck. The filtered packets were suspected to contain malicious content and subsequently sent to the GPU for full pattern matching. The required data structure was sufficiently small to enable storing in CPU caches. This not only enables faster pre-filtering but also provides high filtration ratio to significantly lessen the GPU latency. For performance comparison with the HPMA, other two cases were implemented: full pattern matching directly in CPU (denoted as "CPU-only") and full pattern matching directly in GPU (denoted as "GPU-only"). The experimental result showed that the HPMA outperformed the CPU-only and GPU-only full pattern matching algorithms.

Even the HPMA achieved better efficiency, the overall performance may be limited under certain unfavorable conditions. To overcome this limitation, we propose a capability-based hybrid CPU/GPU pattern matching algorithm (CHPMA), which is an enhancement of the HPMA method. The CHPMA first estimates the CPU and GPU processing capability in the system, and then self-select the processing mode in runtime based on the estimation.

The rest of this paper is organized as follows: Section 2 presents previous studies on pattern matching algorithms and related methods applied to NIDSs. Section 3 illustrates the proposed CHPMA. Section 4 demonstrates the preliminary experimental result. Finally, the conclusion and future work are described in Section 5.

2. Related Work

Several pattern matching algorithms have been proposed, which can be divided into single-pattern matching and multi-pattern matching. The Knuth-Morris-Pratt (KMP) [9] and the Aho-Corasick (AC) [10] are respectively widely-used single-pattern matching and multi-pattern matching algorithms.

Software-based implementations have been focused on implementing with GPUs. Vasiliadis *et al*. [11] implemented an NIDS named Gnort, which was a modification of Snort. In Gnort, the AC algorithm was applied on a GPU platform to which the packets were transferred for full pattern matching; the matched results were subsequently returned to the CPU. The results showed that the maximum throughput was 2.3 Gbps in the synthetic traffic, and the processing speed was two times faster than Snort in real traffic. Moreover, Vasiliadis *et al*. [12] designed another parallel processing architecture named MIDeA that optimized the parallelism of the network interface card (NIC), CPU and GPU in order to upgrade the processing performance. MIDeA was implemented on off-the-shelf equipment and reached a throughput of 5.2 Gbps with real traffic input.

Although these solutions with GPUs improve the efficiency, there is still a restriction of data transfer bottleneck via the PCIe channels. Literature [13] mentioned that the GPU latency, including kernel launch latency and data transfer latency, should be considered in the implementations. The data transfer latency in particular may cause performance bottlenecks, implying that allocating all tasks to GPU does not always yield the optimal efficiency despite its powerful processing capability. Therefore, collaboration between CPU and GPU may offer the solution for the optimal efficiency.

Literature [14] was a collaboration method with CPU and GPU. They used the binary integer linear programming method to generate and optimize a sub-pattern set from the original patterns. The sub-pattern set was loaded to the GPU device memory which is used by filtering; the filtered results were then transferred to the CPU, which proceeds with full pattern matching. Nevertheless, this paper only described how to generate the sub-pattern set and did not provide any performance results. Besides, the data transfer bottleneck between the CPU and GPU may affect the overall performance, because all incoming packets were sent to the GPU in this literature.

We focused on a novel cooperation method between CPU and GPU. According to literature [13], it described that the CPU performance may be degraded with the computation and memory-intensive operations such as pattern matching algorithm; GPUs processing power may be limited with the data transfer bottleneck via the PCIe channel. Hence, we took these factors into consideration and designed a proper balancing allocation for more improvement; that was our previous work HPMA [8]. The following two aspects were emphasized for this design.

- CPU reduces the packets transferred to GPU with rapid pre-filtering and small data structure is required (lessen the computation and memory-intensive latency).
- GPU inspects the pre-filtered-out packets from CPU with full pattern matching (lessen the data transfer latency via PCIe channel).

The HPMA outperformed the CPU-only and GPU-only algorithms by 3.4 and 2.7 times respectively with input traffic of 1460-byte random payloads, indicating that the collaboration between the CPU and GPU can upgrade the processing speed. However, the HPMA achieved better efficiency, but the following prerequisites should be satisfied: (1) the CPU processing capability is not too weak, and (2) the intrusive percentage of incoming packets does not reach to 100%. Or the overall processing speed would be restricted by the CPU and even get worse than the GPU-only method, since the former made the pre-filtering too slow, the latter caused redundant operations rather than directly delivering packets to the GPU.

3. Capability-Based Hybrid CPU/GPU Pattern Matching Algorithm

In this section, our CHPMA is introduced. Fig. 1 illustrates CHPMA architecture, which consists of two phases: benchmarking phase and runtime phase, which are further described in the following subsections.

3.1 The Benchmarking Phase

The benchmarking phase is a preprocessing phase to estimate the CPU/GPU processing capability in the system. Here, two components are created in this phase:

- CPU/GPU processing capability estimation.
- Theoretical filtration ratio calculation.

The “theoretical filtration ratio” (denoted as r_T) is defined as a threshold to determine whether the CPU pre-filter is enabled (denoted as “CG-hybrid mode”) or not (denoted as “GPU-direct mode”).

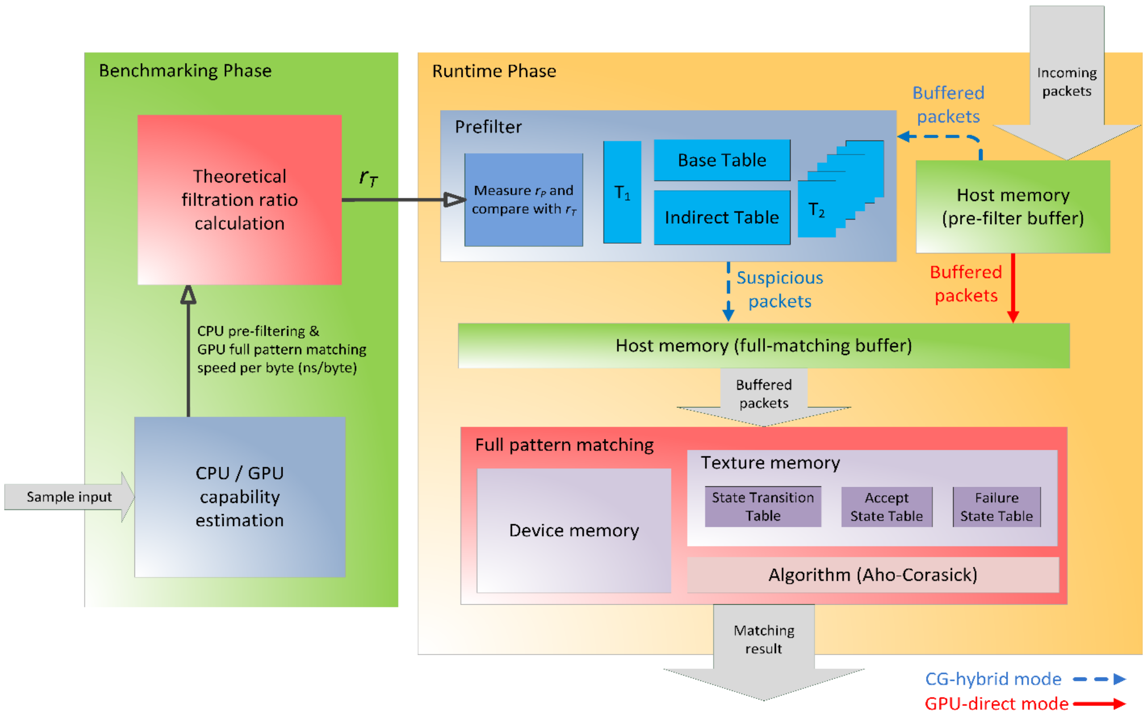


Fig. 1. CHPMA architecture.

Initially, the random packet payloads are inserted as sample input into the component of “CPU/GPU processing capability estimation”. Random packet payloads can be used because the time complexity of pre-filtering and AC algorithm only relates to the packet payload length. The estimation results contain two values: the total processing time of CPU pre-filtering (denoted as T_{CPU}) and the total processing time of GPU full pattern matching (denoted as T_{GPU}). These two values are defined with the following equations.

$$T_{\text{CPU}} = n \times t_{\text{CPU}} \tag{1}$$

$$T_{\text{GPU}} = n \times t_{\text{GPU}} \tag{2}$$

where n is the total number of payload bytes processed. The values t_{CPU} and t_{GPU} represent the average processing time per byte of T_{CPU} and T_{GPU}, respectively. Here, the t_{CPU} and t_{GPU} are regarded as the input of the component “Theoretical filtration ratio calculation”.

Once the values n, t_{CPU} and t_{GPU} are known, we can obtain the r_T by the following equation.

$$T_{C+G} = n \times t_{\text{CPU}} + r \times n \times t_{\text{GPU}} \tag{3}$$

where T_{C+G} represents the total processing time of using CG-hybrid mode, and r is the ratio of suspicious packets. If we want to satisfy the condition that using CG-hybrid mode preforms better than using GPU-direct mode, the processing time in CG-hybrid mode should be no more than that in GPU-direct mode. Therefore, it can be formulated as follows.

$$T_{C+G} \le T_{\text{GPU}} \tag{4}$$

From (1) to (4), we can derive

$$r \leq 1 - \frac{t_{\mathrm{CPU}}}{t_{\mathrm{GPU}}} \tag{5}$$

If the ratio r satisfies (5), it indicates that the choice of CG-hybrid mode is better; on the contrary, GPU-direct mode is a proper selection. Hence, the r_T is defined as

$$r_T = 1 - \frac{t_{\mathrm{CPU}}}{t_{\mathrm{GPU}}} \tag{6}$$

Finally, the r_T is outputted from the benchmarking phase.

3.2 The Runtime Phase

After the r_T is loaded from the previous phase, the runtime phase can start. Here, four components are created in this phase:

- Pre-filter buffer
- Pre-filter
- Full-matching buffer
- Full pattern matching

The pre-filter buffer is used to store the packets which are delivered batch by batch. First, the incoming packets are received and sent to this buffer; as soon as this buffer becomes full, these packets will be processed by the pre-filter or sent to the full-matching buffer directly according to the selected processing mode. If the current processing mode is CG-hybrid, the system performs CPU pre-filtering and delivers the filtered-out packets (suspicious packets) to the full-matching buffer. Meanwhile, the pre-filter measures the practical filtration ratio (denoted as r_P) and compares with the r_T to determine whether the processing mode is stayed in CG-hybrid or changed to GPU-direct. The definition of r_P is

$$r_P = \frac{N_{\mathrm{sus}}}{N_{\mathrm{all}}} \tag{7}$$

where N_{sus} is number of suspicious packets in the pre-filter buffer, and N_{all} is total number of packets in the pre-filter buffer. In case $r_P \leq r_T$, the system infers that during the time, the processing in CG-hybrid mode is still better than that in GPU-direct mode. Therefore, the system will stay in CG-hybrid mode; otherwise, the system infers that GPU-direct mode becomes better, so the processing mode is changed to GPU-direct. Thus, the next batch of incoming packets will be copied to the full-matching buffer directly.

As the packets in the pre-filter buffer are determined, the packets being further inspected are copied to the full-matching buffer. Up to this step, one processing cycle is finished; afterward the next batch of incoming packets is loaded into the pre-filter buffer to carry on the next processing cycle. While the full-matching buffer is full or the last packet is delivered in certain processing cycle, the packets in this buffer are delivered to GPU device memory; the system triggers the GPU full pattern matching. Finally, the matched result is outputted and copied back to the CPU host memory.

Fig. 2 is the finite automaton for illustrating the mode transition depending on the theoretical filtration ratio (r_T) and practical filtration ratio (r_P). There are two states in this finite automaton: CG-hybrid mode and GPU-direct mode. CG-hybrid mode is the initial state when the system starts up. As the current state is CG-hybrid mode, the r_P is measured and compared with r_T. If the r_P is equal or smaller than r_T, the state stays in CG-hybrid mode; otherwise, the state transits to GPU-direct mode. The system stays in the state of GPU-direct during some processing cycles and then transits back to CG-hybrid. Here, the duration was set to twenty processing cycles. As the current state is GPU-direct mode, differ to CG-hybrid, the r_T is compared with the r_{P^0}. The r_{P^0} is the practical filtration ratio of 0% intrusive packets (random packets), which corresponds to Table 2 presented in the next section. If the r_T equals or lower than r_{P^0}, indicating the lack of CPU processing resource no matter what traffic is inputted, the state always stays in GPU-direct.

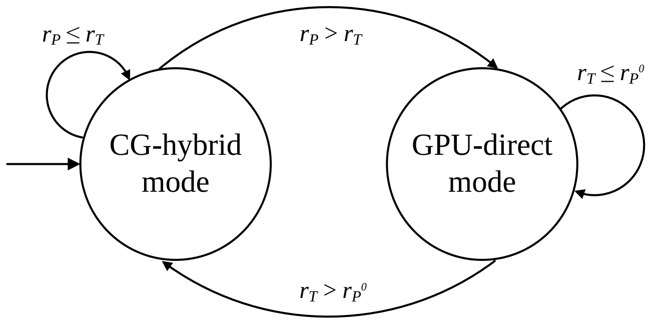


Fig. 2. The finite automaton of CHPMA.

4. Preliminary Evaluation

In this section, a preliminary evaluation for the CHPMA is presented. We constructed the CHPMA using NVIDIA's Compute Unified Device Architecture (CUDA) [15] to implement the full pattern matching algorithm and using OpenMP [16] to implement the pre-filtering algorithm. Both are in multithreaded fashion in order to achieve higher performance. Table 1 lists the hardware specification in the experiment, in which Intel Core i7-3770 CPU with 4 cores and NVIDIA GeForce GTX680 GPU with 1536 cores were used. Here, we only allocated 2 CPU cores to the pre-filtering in order to simulate the unfavorable condition of low CPU processing capability. For the CUDA parameter settings, the number of blocks and the number of threads per block were set to 128 and 64 respectively. We prepared 6 intrusive traffic sets, which the intrusive packet percentages are 0%, 20%, 40%, 60%, 80% and 100%. The input traffic sets with high intrusive percentages were to simulate the other unfavorable condition. The pattern set was extracted from the Snort rules 2008 [17]. The lengths of patterns in this set ranged from 1 to 208 bytes and 13.8 bytes in average. After the patterns were obtained, random payloads with lengths of 1460 bytes were generated. If a payload was determined to be intrusive, a randomly chosen pattern was inserted in an arbitrary location in this payload. For other configuration, the full matching buffer and the pre-filter buffer size were set to 512 MB and 1 MB, respectively.

Table 1. Hardware Specification in Preliminary Evaluation

Device	CPU	GPU
Specification	Intel Core i7-3770 Number of cores: 4 Clock rate: 3.40GHz Host memory: 8GB DDR3	NVIDIA GeForce GTX680 Number of cores: 1536 Clock rate: 1058 MHz Device memory: 2GB GDDR5

Table 2. The Actual Intrusive Percentage, Corresponding Practical and Theoretical Filtration Ratio

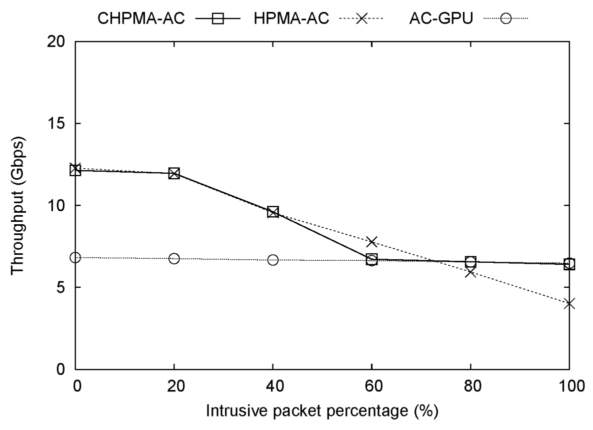
Actual intrusive percentage	Practical filtration ratio (r_P)	Theoretical filtration ratio (r_T)
0%	0.044 (r_{P^0})	0.52
20%	0.235	
40%	0.427	
60%	0.617	
80%	0.808	
100%	1.000	

Table 2 displays the result of our pre-filtering. The actual intrusive percentages and corresponding practical filtration ratios (r_P) are presented. The result shows that the pre-filter provided high-filtration ratio, in which the difference with actual percentages ranged only from 0% to 4.4%. Moreover, the estimated r_T in this condition from the benchmarking phase was 0.52.

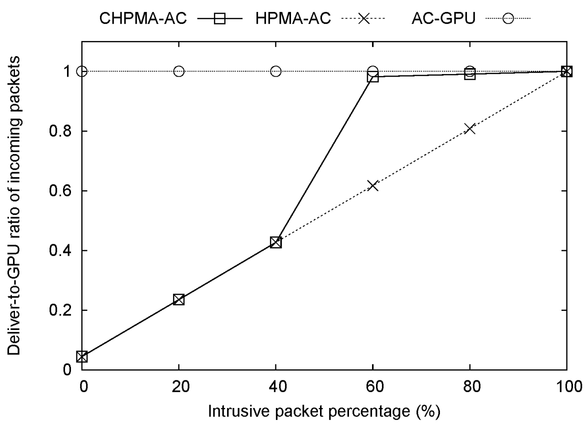
The preliminary experimental results are illustrated in Fig. 3(a) and Fig. 3(b). In these figures, the terms "CHPMA-AC" and "HPMA-AC" are respectively the matching methods of CHPMA and HPMA that both used the AC algorithm in full pattern matching. "AC-GPU" is the GPU-only matching method that directly inspects packets by the AC algorithm in the GPU.

Fig. 3(a) displays the performance result and comparison with the previous methods. The x-axis represents the input traffic set with different intrusive percentages, and the y-axis is the average throughputs performed in Gbps. AC-GPU kept the throughputs from 6.8 to 6.5 Gbps; HPMA-AC presented the throughputs in 12.1 Gbps between the intrusive percentages 0% and 20%, and the performance began to degrade after 20% since the data transfer bottleneck occurred when the number of GPU-inspected packets increased. Overall, HPMA-AC was more efficient than AC-GPU until the intrusive percentage 60%, in which the throughputs ranged from 12.1 Gbps to 7.8 Gbps. Nevertheless, it became worse while the intrusive percentage exceeded 60%; the performance degraded the most to 4.0 Gbps when intrusive percentage achieved 100%. The cause was not only the lack of CPU capability but also high percentage of intrusive packets. Note that when the intrusive percentage was up to 100%, that is, all incoming packets were intrusive and delivered to GPU, HPMA-AC must be no better than AC-GPU method due to the redundant operations rather than the packets processed by GPU directly.

On the other hand, CHPMA-AC adapted as HPMA-AC before intrusive percentage of 60% and adapted as AC-GPU otherwise. When the intrusive percentage was lower than 60%, CHPMA-AC performed as HPMA-AC, which also achieved from 12.1 Gbps to 7.8 Gbps; otherwise, CHPMA-AC performed as AC-GPU. This is because the r_T value was 0.52 that is lower than corresponding r_P in intrusive percentage 60% (0.617). Hence, CHPMA-AC selected GPU-direct mode, which reached 6.7 Gbps to 6.4 Gbps.



(a)



(b)

Fig. 3. (a) The performance result of CHPMA and comparison with previous methods. (b) The ratio of packets delivered to GPU in CHPMA and comparison with previous methods.

Note that the curves of HPMA-AC and AC-GPU intersected between the intrusive percentages 60% and 80%. An error between the r_T and the intersection caused CHPMA-AC to become worse than HPMA-AC in intrusive percentage 60%. The reason is that the full matching buffer was not filled with the last batch of incoming packets, which were directly delivered to the GPU. This made the average processing time per byte in GPU increased, so the practical filtration threshold resulted was thus right shifted. After the intrusive percentages increased to 80%, CHPMA-AC performed efficient than HPMA-AC, especially 1.6 times faster in the case of intrusive percentage 100%. On the other hand, the throughput of CHPMA-AC was slightly lower (about 0.1 Gbps) than that of AC-GPU in this case, since the processing mode was changed back from GPU-direct to CG-hybrid in a few processing cycles. Namely, the small number of packets were processed in CG-hybrid mode, so a slight performance degradation occurred.

Fig. 3(b) displays the ratio of packets delivered to GPU and comparison with the previous methods. The x-axis represents the input traffic sets with different intrusive percentages, and the y-axis is the proportion of packet number delivered to GPU to total packet number. Each deliver-to-GPU ratio of incoming packets in AC-GPU was always 1, because all incoming packets were sent to the GPU whatever the input intrusive percentage was. The resulted deliver-to-GPU ratios in HPMA-AC presented linearly, which corresponded with the practical filtration ratio in Table 2. In terms of CHPMA-AC, CG-hybrid mode was selected when the intrusive percentage within 40%, so the resulted deliver-to-GPU ratios in CHPMA-AC were identical with HPMA-AC method. On the other hand, when the intrusive percentage became larger than 40%, more packets were sent to GPU directly without pre-filtering; therefore, the resulted deliver-to-GPU ratios increased.

Note that in CHPMA-AC, the processing mode was switched back to CG-hybrid in certain processing cycles except the case of intrusive percentage 100%. A few packets (about 1% to 2% of incoming packets) were thus processed by the pre-filter and part of those packets were not filtered-out. Hence, the resulted deliver-to-GPU ratios with intrusive percentages 80% and 60% were not 1.

5. Conclusion and Future Work

This study proposes the CHPMA that is an improvement of our previous work HPMA implemented on CPU/GPU platform. The CHPMA first estimates CPU/GPU processing capability, and in runtime the system can automatically select one of the processing modes, CG-hybrid and GPU-direct, depending on the estimation. We had a preliminary evaluation that the throughput and the deliver-to-GPU ratio results are presented. Moreover, the previous methods are also compared. The result shows that the CHPMA was capable of achieving better efficiency than the HPMA in handling the high-intrusive-percentage traffic, and performed as efficient as the HPMA otherwise. Since the input traffic sets were generated with fixed intrusive percentage of packets here, the CHPMA should be further enhanced to deal with the traffic with varied intrusive percentages. The theoretical filtration ratio accuracy and the processing mode selection can be also improved. Besides, the deliver-to-GPU ratios of incoming packets can be analyzed. In our future work, we will have further design toward these factors based on the CHPMA, being more dynamic to achieve higher efficiency.

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-101-2221-E-009-004-MY3, MOST-102-2221-E-182-034 and MOST-104-2221-E-182-005) and Chang Gung Memorial Hospital (BMRP 942).

References

[1] Paxson, V. (1999). Bro: A system for detecting network intruders in real-time. *Computer Networks, 31(23)*, 2435-2463.

[2] Cabrera, J. B., Gosar, J., Lee, W., & Mehra, R. K. (2004). On the statistical distribution of processing times

in network intrusion detection. *Proceedings of IEEE Conference on Decision and Control*: *Vol. 1* (pp. 75-80).

[3] Aitra, A., Najjar, W., & Bhuyan, L. (2007). Compiling PCRE to FPGA for accelerating Snort IDS. *Proceedings of ACM/IEEE Symposium on Architecture for Networking and Communications Systems* (pp. 127-136).

[4] Sourdis, I., & Pnevmatikatos, D. (2004). Pre-decoded CAMs for efficient and high-speed NIDS pattern matching. *IEEE International Symposium on Field-Programmable Custom Computing Machines* (pp. 258-267).

[5] Liu, R. T., Huang, N. F., Chen, C. H., & Kao, C. N. (2004). A fast string-matching algorithm for network processor-based intrusion detection system. *ACM Transactions on Embedded Computing System*, *3(3)*, 614-633.

[6] Bacon, D. F., Rabbah, R., & Shukla, S. (2013). FPGA programming of the masses. *Communications of the ACM*, *56(4)*, 56-63.

[7] Vallentin, M., Sommer, R., Lee, J., Leres, C., Paxson, V., & Tierney, B. (2007). The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. *International Symposium on Recent Advances in Intrusion Detection* (pp. 107-126).

[8] Lee, C. L., Lin, Y. S., & Chen, Y. C. (October 2015). A hybrid CPU/GPU pattern matching algorithm for deep packet inspection. Retrieved October 10, 2015, from http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0139301

[9] Knuth, D. E., Morris, J., & Pratt, V. (1977). Fast pattern matching in strings. *SIAM Journal on Computing, 6(2)*, 127-146.

[10] Aho, A. V., & Corasick, M. J. (1975). Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, *18(6)*, 333-340.

[11] Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E. P., & Iasnnidis, S. (2008). Gnort: High performance network intrusion detection using graphics processors. *International Symposium on Recent Advances in Intrusion Detection* (pp. 116-134).

[12] Vasiliadis, G., Polychronakis, M., & Ioannidis, S. (2011). MIDeA: A multi-parallel intrusion detection architecture. *ACM Conference on Computer and Communication Security* (pp. 297-308).

[13] Han, S., Jang, K., Park, K., & Moon, S. (2011). PacketShader: A GPU-accelerated software router. *ACM SIGCOMM Computer Communication Review*, *41(4)*, 195-206.

[14] Wu, C., Yin, J., Cai, Z., Zhu, E., & Cheng, J. (2009). An efficient pre-filtering mechanism for parallel intrusion detection based on many-core GPU. *Security Technology* (pp. 298-305). Springer Berlin Heidelberg.

[15] Nvidia Corporation. (2015). NVIDIA CUDA C Programming Guide. Retrieved October 10, 2015, from http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[16] OpenMP. (2015). Retrieved October 10, 2015, from http://openmp.org

[17] Snort.Org. (2015). Retrieved October 10, 2015, from http://www.snort.org

Yi-Shan Lin was born in Taipei, Taiwan, in 1985. She received the M.S. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2010. She is currently pursuing the Ph.D. degree with the Institute of Computer Science and Engineering, National Chiao Tung University, Hsinchu, Taiwan. Her research interests include deep packet inspection, software-defined networking and artificial intelligence.

Chun-Liang Lee received the M.S. and Ph.D. degrees in computer science and information engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1997 and 2001, respectively. From 2002 to 2006, he was with the Telecommunication Laboratories, Chunghwa Telecom Co. Ltd. Since 2006, he has been an assistant professor of computer science and information engineering at Chang Gung University, Taoyuan. His research interests include the design and analysis of network protocols, quality of service in the Internet, and packet classification algorithms.

Yaw-Chung Chen received the B.S. degree from National Chiao Tung University, Hsinchu, Taiwan, the M.S. degree from Texas A&M University, Kingsville, Texas, USA, and the Ph.D. degree from Northwestern University, Evanston, Illinois, USA. In 1986, he joined AT & T Bell Laboratories, where he worked on various exploratory projects. He joined National Chiao Tung University, Hsinchu, Taiwan as an associate professor in 1990. He is currently a professor in the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. He is also the director of Information Industry Institute/National Chiao Tung University Joint Research Center. His research interests include M2M/D2D communications, media streaming, and P2P systems. He is a senior member of IEEE and a member of ACM.