Deep Packet/Flow Analysis using GPUs

Qian Gong, Wenji Wu, Phil DeMar
Core Computing Division, Fermilab
PO Box 500, MS-368, Batavia, IL 60510, USA
{qgong, wenji, demar}@fnal.gov

***Abstract*—Deep packet inspection (DPI) faces severe performance challenges in high-speed networks (40/100 GE) as it requires a large amount of raw computing power and high I/O throughputs. Recently, researchers have tentatively used GPUs to address the above issues and boost the performance of DPI. Typically, DPI applications involve highly complex operations in both per-packet and per-flow data level, often in real-time. The parallel architecture of GPUs fits exceptionally well for per-packet network traffic processing. However, for stateful network protocols such as TCP, their data stream need to be reconstructed in a per-flow level to deliver a consistent content analysis. Since the flow-centric operations are naturally anti-parallel and often require large memory space for buffering out-of-sequence packets, they can be problematic for GPUs, whose memory is normally limited to several gigabytes. In this work, we present a highly efficient GPU-based deep packet/flow analysis framework. The proposed design includes a purely GPU-implemented flow tracking and TCP stream reassembly. Instead of buffering and waiting for TCP packets to become in sequence, our framework process the packets in batch and uses a deterministic finite automaton (DFA) with prefix-/suffix- tree method to detect patterns across out-of-sequence packets that happen to be located in different batches. Evaluation shows that our code can reassemble and forward tens of millions of packets per second and conduct a stateful signature-based deep packet inspection at 55 Gbit/s using an NVIDIA K40 GPU.**

I. Introduction

Deep packet analysis examines packet contents to search for the transport of sensitive information, signs of suspicious attacks, and signatures of different network applications. In order to perform a complete evaluation on the network status and safeguard the security environment, the inspection has to be performed at both per-packet and per-flow data levels. For the latter, packets belonging to the same flow need to be buffered, reassembled in sequence, and inspected in aggregate. Viewed at a high level, real-time DPI requires significant computing power and high I/O bandwidth. As the 40/100 GE network technologies come into play, keeping up with the inspection of full packet becomes quite difficult.

Special-purpose hardware devices such as FPGAs, ASICs, TCAM, and NPUs normally provide better performance than a functionally equivalent implementation in software. However, these systems have poor programmability and are difficult to extend. Software implementation on commodity hardwares can be an alternative solution for DPI applications. Recently, GPUs have shown great potentials in accelerating computationally intensive tasks in both scientific and engineering fields. Compared to multicore CPUs, GPU's ample memory bandwidth and native data-parallel execution mode are a better fit for the network I/O throughout-intensive requirements and scale better for packet analysis applications.

Nevertheless, most GPU-based traffic analysis tools are limited to per-packet level data parallelism and heavily rely on CPU to tackle stateful operations such as TCP flow state management and stream reassembly [2, 3]. A notable exception is GASPP [1], which pairs consecutive packets by hashing their TCP sequences. When any thread encounters a TCP sequence hole, GSAPP marks all subsequence packets of the same flow as *out-of-sequence* and buffers them until a new batch of traffic is received. Although this *packet buffering* mechanism is common in current per-flow level IDSs, it is very resource-intensive and vulnerable to denial-of-service attack triggered by intentionally sending legitimate mis-ordered traffic. Moreover, we argue the hash-based stream reassembly approach presented in GASPP is memory-consuming and degrades the performance in the case of hash collision.

Below, we present our effort towards designing a highly efficient GPU-based packet analysis framework, which provides both per-packet and per-flow level DPI capability. This framework fully utilizes the parallelism offered by multi-queue network interface cards (NICs), multicore architectures, and GPUs, to meet the challenges at both the packet capture and traffic analysis levels. Network flows are tracked and TCP streams reassembled on GPUs. Instead of buffering and waiting for TCP packets to become in sequence, incoming packets are processed in batches. Packets that belong to the same batch are aggregated and aligned according to their protocols and TCP sequences, with pattern matching states between consecutive batches connected by combining an Aho-Corasick (AC) DFA with a suffix tree [5]. Below, we will describe our pipeline architecture, packet processing mechanism, and preliminary results.

II. Design Architecture and Mechanism

The high-level design of our GPU-based DPI system is shown in Fig. 1. It includes a multi-parallel network traffic analysis architecture. Packets are captured using the WireCAP [4], a *zero-copy* packet capture engine, and are scaled to multi-queue NICs and multicore systems. WireCAP introduces an advanced buffer-pool mechanism to handle short-term packet overload situations on the NIC's ring buffer, as well as an advanced load balance mechanism to handle long-term load imbalance situations. These advanced mechanisms eliminate packet loss vulnerabilities inherent in conventional packet engines, such as PF_RING [6].

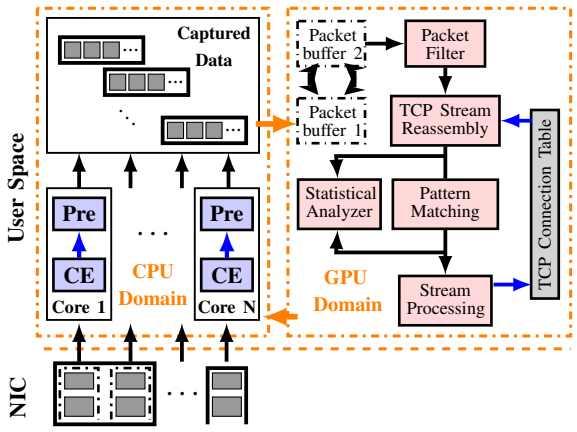

Fig. 1. System Architecture. (CE: capture engine; Pre: pre-processor).

Incoming packets are transferred to GPU in batches to reduce PCIe transaction overhead. We allocate a separate buffer for each capture thread to collect packets and a queue containing a number of empty buffer chunks of the same size. Whenever a buffer gets full, the host thread exchanges it with an empty chunk in the queue and continue to load new packets. At the GPU side, we employ a double buffering scheme to pipeline the inter-device memory transfer and the GPU execution. While the GPU is performing deep analysis for the packets in one buffer, the CPU copies newly arrived packets to another.

Transferred traffic first goes through a packet filter. The filter is a program with a set of rules to decide what packets will be processed. For the packet analysis, payloads of the TCP traffic have to be reconstructed and processed in-sequence, although some stateless protocols such as UDP can be executed in a per-packet data-parallel mode. The stream reassembly function uses a few basic GPU operations, such as *sort* and *prefix-sum*, to aggregate packets belonging to the same TCP flow. Packets are sorted by their IP addresses, TCP ports, and TCP sequences. The result is a list of indices of the next in-order packets. Payloads of TCP packets are then concatenated by a normalizer, which marks the overlaps and updates the next packet array if encountering duplicate. Traffic aggregated by the flow reassembler can be sent to a header analyzer as well to further learn the traffic distribution.

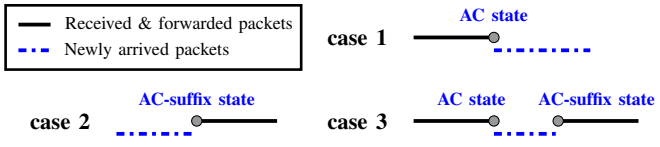

Fig. 2. Pattern matching over out-of-sequence TCP segments.

Here we implemented a signature-based intrusion detection. To detect patterns that happen to cross TCP segments in different batches, we scan each incoming traffic stream twice, one through an AC automaton and one through a AC-suffix automaton. Our DPI framework uses a hash table stored at GPU for keeping the state of TCP connections. For each in-sequence TCP segment, the sequence number of its lastly received packet and the final matching states are stored. Hash table collisions are handled via a linked list. As illustrated in Fig. 2, when a new batch of traffic is received and reassembled, the stream processing module searches the records in the connection table, and the matching process of pre-established connections will continue from the previously stored states. The stream processor is executed again at the end of each batch operation to update the stored matching states and flow connections records.

TABLE I
RAW PROCESSING THROUGHPUT

	GPU	CPU
TCP Reassembly	552.28 Gbit/s	4.55 Gbit/s (Libnids)
Pattern Matching	62.53 Gbit/s	0.27 Gbit/s (Snort)

III. THROUGHPUT PERFORMANCE

We compare the throughputs of our GPU code to two widely adopted IDSs—Snort [8] and Libnids [7], and to a CPU-implemented multithreading AC-suffix-tree code. The experiments were conducted using a server equipped with an Intel E5-2650 v3 CPU and an NVIDIA K40 GPU. Table I shows the raw processing throughputs of two core traffic processing applications—TCP stream reassembly and pattern matching; Fig. 3 displays the comparison of end-to-end GPU and CPU processing throughputs, with and without PCIe data transfer. The results suggest that our raw GPU applications can be hundreds of times faster than an equivalent single-threading CPU code; when the overhead of data transfer is considered, our code is still over 15 times faster than a multithreading application using 8 CPU cores.

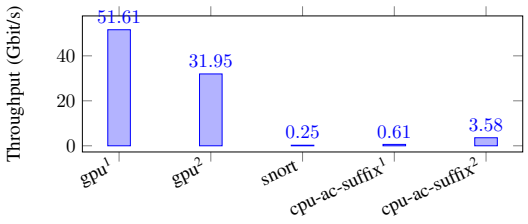

Fig. 3. Comparison of end-to-end throughputs. gpu^2 and gpu^1 measure the throughputs with and without data transfer; $cpu\text{-}ac\text{-}suffix^1$ uses one CPU thread while $cpu\text{-}ac\text{-}suffix^2$ uses eight.

IV. CONCLUSION AND FUTURE WORKS

To conclude, we present the first (to the best of our knowledge) purely GPU-implemented DPI code that tracks the flow states on GPU and fully addresses the patterns across any out-of-sequence packets. The next work will be extending the current intrusion detection capacity by 1) fusing the information learned from the header and payload of traffics; 2) implementing a regular expression matching engine for out-of-sequence packets without requiring packet buffering.

REFERENCES

[1] Vasiliadis, Giorgos, et al. "GASPP: A GPU-Accelerated Stateful Packet Processing Framework." USENIX Annual Technical Conference. 2014.

[2] Jamshed, Muhammad Asim, et al. "Kargus: a highly-scalable software-based intrusion detection system." Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012.

[3] Vasiliadis, Giorgos, Michalis Polychronakis, and Sotiris Ioannidis. "MIDeA: a multi-parallel intrusion detection architecture." Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011.

[4] Wu, Wenji, and Phil DeMar. "Wirecap: a novel packet capture engine for commodity NICs in high-speed networks." Proceedings of the 2014 Conference on Internet Measurement Conference. ACM, 2014.

[5] Chen, Xinming, et al. "AC-suffix-tree: Buffer free string matching on out-of-sequence packets." Architectures for Networking and Communications Systems (ANCS), 2011 Seventh ACM/IEEE Symposium on. IEEE, 2011.

[6] Deri, Luca. "Improving passive packet capture: Beyond device polling." Proceedings of SANE. Vol. 2004. 2004.

[7] Libnids, accessed on July 26, 2017. [Online]. Available: http://libnids.sourceforge.net.

[8] Roesch, Martin. "Snort: Lightweight intrusion detection for networks." Lisa. Vol. 99. No. 1. 1999.