aci avances en ciencias e ingenierías

Artículo/Article
Sección/Section C
Vol. 13, nro. 2
ID: 2142

A high granularity approach to network packet processing for latency-tolerant applications with CUDA (Corvyd)

Daniel Barrett[1] and Maria Pantoja[1]

[1]CalPoly, San Luis Obispo CA 94332, USA
* Corresponding author/ Autor principal: mpanto01@calpoly.edu

Estudio de alta granularidad para el procesamiento de paquetes de red en aplicaciones con tolerancia a la latencia en CUDA (Corvyd)

Abstract

Currently, practical network packet processing used for Intrusion Detection Systems/ Intrusion Prevention Systems (IDS/IPS) tend to belong to one of two disjoint categories: software-only implementations running on general-purpose CPUs, or highly specialized network hardware implementations using ASICs or FPGAs for the most common functions, general-purpose CPUs for the rest. These approaches try to maximize the performance and minimize the cost, but neither system, when implemented effectively, is affordable to any clients other than those at the well-funded enterprise level. In this paper, we aim to improve the performance of affordable network packet processing in heterogeneous systems with consumer Graphics Processing Units (GPUs) hardware by optimizing latency-tolerant packet processing operations, notably IDS, to obtain maximum throughput required by such systems in networks sophisticated enough to demand a dedicated IDS/IPS system, but not enough to justify the high cost of cutting-edge specialized hardware. In particular, this project investigated increasing the granularity of OSI layer-based packet batching over that of previous batching approaches. We demonstrate that highly granular GPU-enabled packet processing is generally impractical, compared with existing methods, by implementing our own solution that we call Corvyd, a heterogeneous real-time packet processing engine.

Keywords: Networks, CUDA, Intrusion Detection Systems

Resumen

En este artículo, estamos interesados en investigar el procesamiento por lotes basado en capas para la inspección de paquetes de red en paralelo. Estudios anteriores de inspección de paquetes basada en GPU [1], [2] y [3] lograron ganancias de rendimiento a través de algunas innovaciones, los más importantes se basaron en la latencia de acceso a la memoria ocultando [4] que no es monopolizado por los sistemas GPU. Los sistemas de procesamiento de paquetes en cualquier hardware también deben usar alguna forma de bypass del kernel para evitar la sobrecarga asociada con las pilas de red de propósito general [4] y [6]. Estos estudios producen un rendimiento dramáticamente mejor que sus predecesores. Por el contrario, el enfoque de este proyecto se limita estrictamente a un conjunto de mejoras algorítmicas. El procesamiento por lotes es

Editado por / Edited by:
Dennis Cazar

Recibido / Received:
01/06/2021

Aceptado / Accepted:
07/09/2021

Publicado en línea / Published online:
15/12/2021

DOI: https://doi.org/10.18272/aci.v13i2.2142

común a cualquier implementación paralela de alto rendimiento. Este proyecto propone explorar un procesamiento más granular basado en capas que divide los paquetes a través de múltiples rondas de programación para maximizar la homogeneidad del lote y minimizar la divergencia de la GPU. Esto aumentará significativamente la sobrecarga para procesar un solo paquete, además de aumentar la latencia a medida que se requerirá más almacenamiento en búfer, pero este enfoque tiene el potencial de mejorar el rendimiento en cargas de trabajo de IP altamente erráticas, donde el trabajo previo ha favorecido fuertemente las cargas de trabajo uniformes como ejemplos de PoC minimizados para representar sistemas que probablemente tendrían un bajo rendimiento en el campo [4],[5], y [6].

Palabras clave: redes, CUDA, Sistemas de Detección de Intrusiones

INTRODUCTION

In this paper we are interested in investigating strictly layer-based batching for highly parallel deep packet inspection. Previous approaches to GPU-based packet inspection [1,2,3] (both shallow and deep) achieved performance gains through a few key innovations, the most significant of which were based on memory access latency hiding [4,5], which is not monopolized by GPU systems. Competitive packet processing systems on any hardware must also use some form of kernel bypass to avoid the overhead associated with general purpose network stacks [4,5]. These key approaches yield dramatically better performance over their predecessors. In contrast, this project's approach is very tightly.

constrained to a small set of algorithmic improvements. Batching is common to any high-throughput parallel implementation. Whereas prior work has tended to batch whole packets together for parallel processing, this project proposes to explore more granular, layer-based processing which splits packets up through multiple scheduling rounds in order to maximize batch homogeneity and minimize GPU warp divergence. This will significantly increase the overhead for processing a single packet, in addition to increasing latency as more buffering will be required, but this approach has potential to improve performance on highly erratic IP workloads, where prior work has strongly favored uniform workloads as minimized PoC examples to represent systems that would likely underperform in the field [4-6].

OVERVIEW CORVYD

Corvyd is a heterogeneous system, using host-based computation to read, structure, and output raw packet data. The GPU device is responsible for all intelligent packet processing. Each host stage uses custom concurrent queues for communication among threads. Wherever practical, each host section gets its own thread. A notable exception is the Device Dispatcher, which is simply a static interface to the kernel and would not benefit from its own parallel pipeline stage. The sections, appearing in fig.1, are as follows:

DOI: https://doi.org/10.18272/aci.v13i2.2142

Packet Batcher

The packet batcher is responsible for reading corvyd packet streams. The corvyd packets are sorted into batches, based on similarity between the packets. Each batch has an associated packet type, and contains only packets for which the last parsed layer is of that type. Processing status of a packet is tracked with the "layer" construct. Batches have a size fixed at launch. Once a batch has collected enough packets, it is sent to the Batch Scheduler for further processing. Note that this means that the worst-case memory complexity of the batcher is proportional to O(p), for which p is the number of unique packet types staged in the batcher, so processing highly diverse packet streams requires significant host memory.

Batch Scheduler

The scheduler is responsible for deciding which batches run when. In practice, it serves as a placeholder since the host-side processing jobs bottleneck around the scheduler, and intelligent GPU scheduling work is effectively done by batching packets in the first place, which is what the scheduler would be doing otherwise. In the current version, the scheduler is just responsible for calling the host wrapper for each GPU kernel as appropriate for the batch type.

DEVICE DISPATCHER

The device dispatcher is the host interface to GPU. It sets up, calls, and recovers results of GPU kernels, and returns the results to the scheduler to be passed on. As batches are very large sets of data not unlike graphical frames, yet do not share the precise regularity of size thereof, batches are handled one at a time.

PACKET SERIALIZER

The design of the OSI model [3] asserts independence of layers within a packet. Therefore, it is not guaranteed that two packets with a second layer of one type will have third layers that also match. Because of the design of batches as described above, after each layer is processed, packets must be re-sorted into new batches for which the last layers match. The serializer breaks batches into their original packets and streams those packets off to the batcher to be re-entered into the pipeline. If the last processed layer is the last one to be processed, then the processed packet is passed on for output.

DOI: https://doi.org/10.18272/aci.v13i2.2142

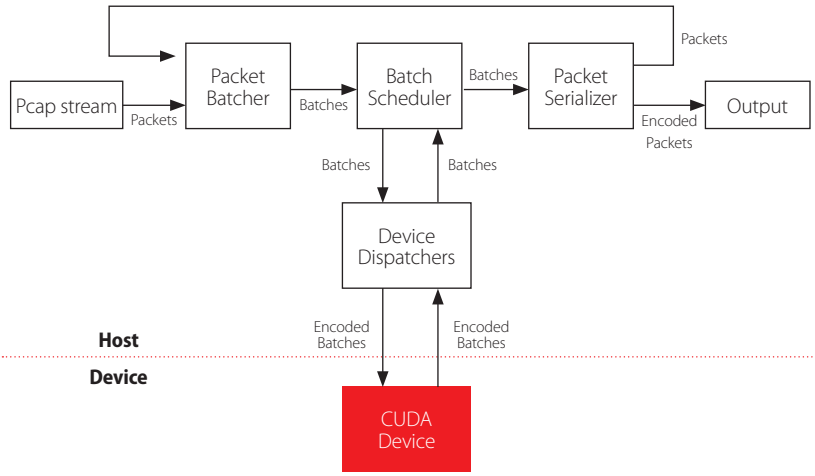

Figure 1. Data flow and structural diagram of the Corvyd processing pipeline.

Experiment

To evaluate its performance, corvyd was compared with Snort 3 [7], a CPU packet processing IDS/IPS system, tcpdump [8], a simple and lean traffic analyzer, and PacketShader [6], a landmark GPU packet processing system. It must be noted that not all of these systems serve precisely the same purpose, but their basic functions and approaches overlap enough to be considered a basic representation of existing tools. Each system was run and timed on a controlled set of pcap files, most importantly characterized by the inter-packet entropy; we hypothesized that our system would perform best with highly entropic workloads compared with other systems, which would perform best with relatively uniform pcaps. The test set consisted of 4 pcaps:

1. A small pcap (¡10 packets) to measure setup overhead.
2. A pcap of identical IP packets.
3. A pcap of a variety of IP packet types, for which each packet of the same type was identical
4. A pcap of real ambient home network traffic.

Solely for reasons of availability, our test system used a Ryzen 1700X CPU, 2x8GB 2133 MT/s DDR4 memory, and a Nvidia RTX 2070 Super GPU, and a Samsung 860 EVO SATA III SSD for all tests.

DOI: https://doi.org/10.18272/aci.v13i2.2142

RESULTS

Given the lack of system-specific optimization, corvyd showed some limited viability compared with other packet processing systems. As seen in table 1, corvyd has a large minimum runtime, resulting from the forced GPU transfers which are inappropriate for small packet processing. For packet captures of larger sizes, corvyd runtimes scale better with complexity than snort3, but much worse than the almost solely size-dependent runtimes of tcpdump. Snort3 does sophisticated processing on a single main thread, so harsh complexity scaling is to be expected. Tcpdump on the other hand does minimal processing and packet dissection and is much more comparable to corvyd. The performance of tcpdump, however, excels far beyond that of the other programs. As others have found [4], [5], the inclusion of a GPU coprocessor in packet processing tasks yields its most significant speedups from memory access latency hiding and the simple addition of hardware. Yet it does not obviously overcome the basic limitations of increased complexity and increased memory overhead associated with heterogeneous computing, however well implemented. Finally, even our version of pure CPU packet processing was much faster than our GPU version, also scaling better with complexity.

Table 1. Performance, in seconds, of various tested programs on various described test workloads

Program	Tiny pcap	Simple pcap	Moderate pcap	Complex pcap
corvyd	0.201	2.02	2.42	2.94
corvyd (CPU only)	0.004	0.40	0.44	0.49
snort3	0.108	2.94	3.15	4.8
tcpdump	0.004	0.43	0.5	0.51

FUTURE WORK

The current state of corvyd is that of a humble proof of concept. It lacks optimization, and does not define an optimal platform. While the harsh contrast between our own GPU and CPU results (table 1) convinces us that the GPU implementation would likely fail even if done optimally, we cannot know for sure until a more competent development team furthers the project. In particular, the host pipeline is cumbersome and memory-intensive. Despite being conceived to run on a weaker CPU with access to a GPU, it is still very CPU-demanding, and sometimes CPU-limited. While performance with an ample host system would be interesting, it was not pursued for reasons of cost and practicality.

CONCLUSION

It is clear that corvyd offers no benefit over traditional CPU-based packet processing. While some of this is explained by the relative crudeness of the implementation, our findings on corvyd time complexity scaling, GPU vs. CPU, corroborate the findings of the G-Opt[4] and APUNet [5] teams, which indicated that, in both performance and cost, CPU packet

DOI: https://doi.org/10.18272/aci.v13i2.2142

processing in general applications reigns supreme. It is the opinion of the authors that even in a medium-sized business where snort3 on a desktop may be too simple and a dedicated flow processor may be too expensive, it is likely better to persist on the snort or suricata [8], [7], [9] box than to repurpose GPUs for a hybrid IPS/IDS system.

AUTHORS' CONTRIBUTIONS

Conceptualization, D.B. and M.P.; Methodology, D.B. and M.P.; Investigation, D.B. and M.P. ; Writing – Original Draft, D.B. and M.P.; Writing – Review & Editing, D.B. and M.P.; Supervision, D.B. and M.P.

CONFLICTS OF INTERESTS

All authors declare that they have no conflicts of interest.

DOI: https://doi.org/10.18272/aci.v13i2.2142

aci avances en ciencias e ingenierías

REFERENCES

[1] Project, T.S. (2020). "snort user manual 2.9.16". https://www.snort.org/documents/snort-users-manual, [Online; accessed 24-April-2020].

[2] Cisco Systems: The Cisco Flow Processor (2014). Cisco's Next Generation NetworkProcessor Solution Overview. https://www.cisco.com/c/en/us/products/collateral/routers/asr-1000-series-aggregation-services-routers/solution_overview_c22-448936.html/ [Online; accessed 19-January2020].

[3] ISO/IEC JTC 1 (1996). "ISO 35.100.01: Open systems interconnection in general"

[4] Vasiliadis, G., Koromilas, L. (2014). GASPP: A GPU-accelerated stateful packet processingframework. USENIX ATC'14.

[5] Go, Y., Jamshed, M.A., Moon, Y., Hwang, C., Park, K. (2017). APUNet: Revitalizing GPUas packet processing accelerator. USENIX NSDI'17.

[6] Han, S., Jang, K., Park, K., Moon, S. (2010). Packetshader: a GPU-accelerated softwarerouter. SIGCOMM'10.

[7] Kalia, D. Zhou, M.K., Andersen, D.G. (2015). Raising the bar for using gpus in softwarepacket processing. Usenix.

[8] Group, T.T. (2020). Tcpdump and libpcap.

[9] Vokorokos, L., Bala´˘z, A., Mado˘s, B. (2012). Intrusion detection architecture utilizing graphics processors. Acta Informatica Pragensia 1, 50–59. doi: https://doi.org/10.18267/j.aip.5

DOI: https://doi.org/10.18272/aci.v13i2.2142