Scaling the Performance of Network Intrusion Detection with Many-core Processors

Jaehyun Nam*, Muhammad Jamshed, Byungkwon Choi, Dongsu Han, and KyoungSoo Park

Graduate School of Information Security*, Department of Electrical Engineering
Korea Advanced Institute of Science and Technology (KAIST)
{namjh, ajamshed, cbkbrad}@kaist.ac.kr, {dongsuh, kyoungsoo}@ee.kaist.ac.kr

ABSTRACT

In this work, we present a highly scalable network intrusion detection system on many-core processors. To maximize the NIDS performance, we take advantage of the underlying hardware and adhere to four design principles: shared-nothing architecture, computation offloading, lightweight data structure, and flow offloading. Through the experimental results, we find that our design choices can significantly improve the NIDS performance (79 Gbps with 1514B synthetic packets). We believe that our design decisions can be easily extended to other many-core processors and programmable NICs.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*

General Terms

Performance

Keywords

many-core, network intrusion detection system, parallel, offloading

1. INTRODUCTION

Network intrusion detection systems (NIDSes) are widely deployed to detect malicious activities in a given network. As network bandwidth at the Internet edge rises, the need for a high performance NIDS is getting critical.

Current commercial NIDSes use FPGA and ASIC hardware for analyzing traffic at high rates. While these NIDSes deliver impressive monitoring throughputs, it is often challenging to configure or update such systems across varying networking environments. On the flip side, IDSes based on commodity-computing hardware, such as multi-core processors and GPUs, provide high flexibility as well as low cost. Moreover such hardware features have helped developers write efficient IDS systems that now achieve processing performances that are comparable with commercialized hardware solutions. However, NIDSes based on such heterogeneous systems (GPU-based) [5,6] do have some drawbacks: it is normally difficult to program a SIMD module on a GPU such that it delivers peak performance while keeping the power consumption and processing latencies in check.

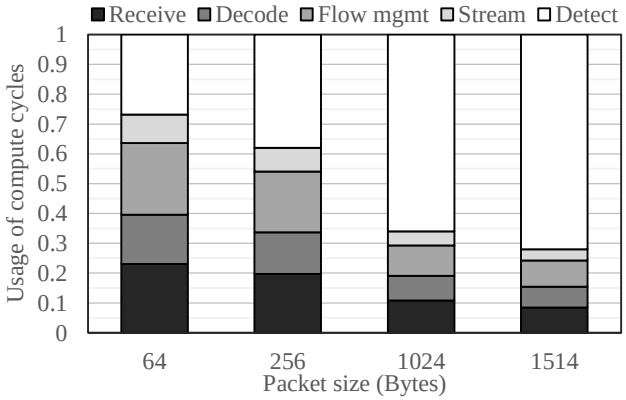

Figure 1: CPU usage breakdown of Suricata modules over various packet size

Recent advancements in system-on-chip many-core processors (MCPs) seem to have resolved most of the issues mentioned above. MCPs host tens to hundreds of processing cores, allowing highly flexible general-purpose computations. With massively parallel computation capacity, they can significantly improve the performance of a NIDS ported on a MCP.

In this work, we build a high-performance NIDS on many-core processors. Our design principle is to balance the workload across many cores to achieve high parallelism while minimizing the per-packet processing overhead by exploiting the underlying hardware. We enforce four design choices: shared-nothing architecture for high parallelism, computational offloading to programmable network interface cards (NICs), using lightweight data structures, and offloading flows to host system's CPU. We develop our system on a TILE-Gx72 many-core processor [3].

2. MOTIVATION

Pattern matching is typically the major performance bottleneck of signature-based NIDSes. However our microbenchmark experiments reveal that massively parallelizing the execution of pattern matching routines diminishes the performance drop since the DPI workload to analyze incoming traffic gets distributed to all cores. In fact, as shown in Figure 1, the per-packet overhead to process NIDS *metadata* takes up a large portion of overall processing cycles: the

detect module represents the pattern matching module while the other modules represent per-packet operations (based on packet headers). Per-packet operations require the same amount of processing cycles regardless of packet size. However, as the packet size decreases, the portion of per-packet operations becomes critical. In this work, we thus focus on improving the performance of processing metadata with hardware-level features of many-core processors.

3. DESIGN

The basic ideas of our NIDS are parallelizing pattern matching as much as possible by giving the most computation cycles to the performance-critical operation and reducing the overhead of per-packet operation as much as possible by employing available hardware resource to offload per-packet operations from regular processing cores.

3.1 Shared-nothing architecture

Earlier versions of NIDSes adopt pipelining architecture that has a few fundamental limitations. For instance, it is hard to decide how many cores need to be assigned for each module due to various incoming workloads. In addition, the pipelining architecture can increase inter-core communication and lock contention, which are crucial to achieve high performance. On the other hand, our NIDS adopts the shared-nothing architecture where each thread is running a separate NIDS engine and pinned to a tile. Furthermore, we eliminate any shared data structures (e.g., flow table) between NIDS engines. Thus, the shared-nothing architecture ensures high scalability unlike the pipelining architecture.

3.2 Optimizations

Computation offloading to programmable NICs: Recently, many NICs [1] provide programmable features. To optimize per-packet operations, we employ the feature of programmable NICs (specifically, mPIPE packet I/O engine in a TILE-Gx72 processor) to offload per-packet operations (e.g., decoding and flow hash computation) from regular processing cores. Once packets come to the NICs, the NICs preprocess some of per-packet operations, and each NIDS engine directly exploits the processed result. Thus, we can save the compute cycles and memory accesses from tiles.

Lightweight data structure: To further reduce the overhead of per-packet operations, we simplify packet metadata structure. NIDSes support diverse packet I/O engines and protocols so that the size of packet metadata structure becomes huge. We divide the data structures into two groups: frequently used fields and rarely used fields. Then, we extract the latter from the data structure. If the rarely used fields are required, they are dynamically allocated on demand. With the lightweight data structure, we can reduce the overall number of cache misses.

Flow offloading to host-side CPU: Once the ingress traffic exceeds the NIDS processing capacity on the many-core processor, subsequent packets should be dropped, resulting in missing chances to detect some of malicious activities. To mitigate this problem, we exploit a PCIe module on the many-core processor (specifically TRIO module in a TILE-Gx72 processor). The basic idea is to dynamically offload subsequent flows to host-side CPU for analysis when the many-core processor faces a high workload. To maximize offloading throughput, we exploit three techniques: developing a zero-copying offloading module, increasing PCIe queues to reduce the contention across main cores, and offloading packets in a batch to reduce per-packet PCIe transfer overhead.

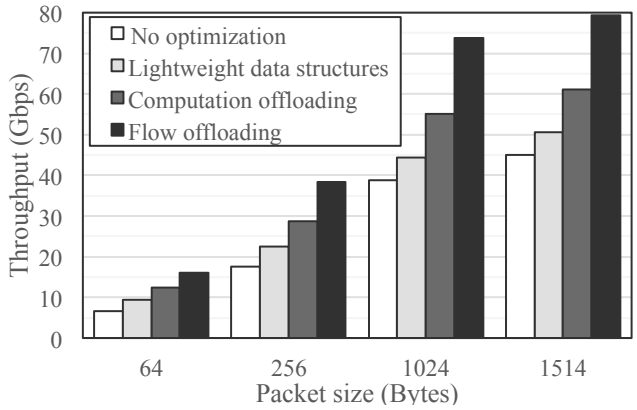

Figure 2: Breakdown of performance improvement by each technique

4. EXPERIMENTAL RESULT

We install a many-core TILE-Gx72 processor on a machine with dual Intel E5-2690 CPUs (octacore, 2.90 GHz, 20 MB L3 cache) with 32 GB of RAM. We run our NIDS extended from a TILE-optimized Suricata [2] on the TILE processor and CPU-based Kargus [5] on the host side. Each NIDS is configured with 2,435 HTTP rules from the Snort 2.9.2.1 ruleset. We generate specific-size packets containing random payloads using our packet generator developed on PSIO [4].

Figure 2 shows the performance breakdown of the three key techniques under synthetic HTTP traffic. The overall performance ranges from 16 to 79 Gbps depending on the packet size. Computation offloading and metadata reduction achieve 33% (1514B packets) to 88% (64B packets) improvements and CPU-side flow offloading achieves 32% additional improvement on average. Through the results, we find that reducing the per-packet operations significantly improves the overall NIDS performance, and we gain noticeable performance benefits by utilizing the host resources.

5. REFERENCES

[1] AdvancedIO - Programmable Network Interface Cards. `http://www.advancedio.com/products/function/programmable-network-interface-solutions`.

[2] EZ-Chip - Security and IDS/IPS solution. `http://www.tilera.com/Solutions/?ezchip=561`.

[3] TILE-Gx Processor Family. `http://www.tilera.com/products/processors/TILE-Gx_Family`.

[4] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: a gpu-accelerated software router. In *Proceedings of the ACM SIGCOMM*, 2010.

[5] M. A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park. Kargus: a highly-scalable software-based intrusion detection system. In *Proceedings of the ACM Conference on Computer and Communications Security(CCS)*, 2012.

[6] G. Vasiliadis, M. Polychronakis, and S. Ioannidis. Midea: a multi-parallel intrusion detection architecture. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2011.