PIPSEA: A Practical IPsec Gateway on Embedded APUs

Jungho Park[†*] Wookeun Jung[†] Gangwon Jo[†*] Ilkoo Lee[†] Jaejin Lee[†]

[†]Center for Manycore Programming
Department of Computer Science and Engineering
Seoul National University, Seoul 08826, Korea

[*]ManyCoreSoft Co., Ltd.
Room 308, Building 138, Seoul National University, Seoul 08826, Korea

{jungho, wookeun, gangwon, ilkoo}@aces.snu.ac.kr, jaejin@snu.ac.kr
http://aces.snu.ac.kr

ABSTRACT

Accelerated Processing Unit (APU) is a heterogeneous multicore processor that contains general-purpose CPU cores and a GPU in a single chip. It also supports Heterogeneous System Architecture (HSA) that provides coherent physically-shared memory between the CPU and the GPU. In this paper, we present the design and implementation of a high-performance IPsec gateway using a low-cost commodity embedded APU. The HSA supported by the APUs eliminates the data copy overhead between the CPU and the GPU, which is unavoidable in the previous discrete GPU approaches. The gateway is implemented in OpenCL to exploit the GPU and uses zero-copy packet I/O APIs in DPDK. The IPsec gateway handles the real-world network traffic where each packet has a different workload. The proposed packet scheduling algorithm significantly improves GPU utilization for such traffic. It works not only for APUs but also for discrete GPUs. With three CPU cores and one GPU in the APU, the IPsec gateway achieves a throughput of 10.36 Gbps with an average latency of 2.79 ms to perform AES-CBC+HMAC-SHA1 for incoming packets of 1024 bytes.

CCS Concepts

•**Security and privacy → Network security;**

Keywords

IPsec; APU; GPU; Cryptography; Heterogeneous computing; OpenCL

CCS'16, October 24-28, 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2976749.2978329

1. INTRODUCTION

The Internet Protocol Security (IPsec) [20] has been widely used to secure applications using the Internet (e.g., e-mail, file transfer, telnet, and web-based applications), VPN (Virtual Private Networks) tunnels, and end-to-end communications between two hosts. It is a protocol suite for securing the IP traffic using cryptographic methods. It provides data origin authentication, data confidentiality, data integrity, and anti-reply protection at the IP layer.

Since IPsec is based on compute-intensive crypto algorithms, IPsec processing requires higher performance to achieve real-time packet processing as network traffic increases. Thus, network systems have used special-purpose hardware units, such as ASICs [11, 15], FPGAs [6, 7, 19], and network processors [24, 25, 28] to accelerate network packet processing.

Even though special-purpose hardware-based cryptographic network systems achieve the desired high performance, they have several disadvantages: high verification/validation/building cost, high revision cost, and difficult programming. Instead, software-based network systems on commodity processors are considered as an alternate solution. They provide cost-effective packet processing, high flexibility, and high programmability. However, software-based network packet processing suffers from its low performance. To address this issue, GPGPUs (General-Purpose computing on Graphics Processing Units) have been considered as an effective solution [12, 18, 22, 23, 26, 30, 33] because of their strong computation capability exploiting massive parallelism.

In this paper, we present the design and implementation of a high-performance IPsec gateway, called PIPSEA, using a resource-constrained embedded APU (Accelerated Processing Unit) [1]. The APU contains at least two and at most four general-purpose CPU cores and a GPU in a single chip. The APU supports HSA (Heterogeneous System Architecture) [3] that provides coherent physical shared memory between the CPU cores and the GPU. IPsec protocols are implemented in OpenCL [21], and the GPU performs IPsec processing. Packet I/O in the IPsec gateway is based on DPDK [2].

We also present an IPSec packet scheduling algorithm that fully exploits a common GPU architecture and improves GPU utilization significantly. The proposed packet schedul-

ing algorithm conservatively assumes each incoming packet has a different IP connection, i.e., a different workload. To achieve high performance for a mix of diverse packets, it is essential to exploit the full compute capability of a GPU using such a packet scheduling algorithm. It works well not only with APUs but also with high-end discrete GPUs (dGPUs).

To the best of our knowledge, PIPSEA is the first practical IPsec solution using an embedded APU. PIPSEA has the following advantages over previous approaches that use high-end dGPUs:

- Since PIPSEA uses a resource-constrained embedded APU, its cost effectiveness is much higher than that of previous high-end dGPU solutions. Our IPsec gateway provides enough throughput to a small-sized network. It provides up to 10 Gbps connectivity (e.g., OC-192 or 10 GbE).
- Because of the HSA and our packet scheduling algorithm, the packet round-trip latency of PIPSEA is better than or comparable to that of previous high-end dGPU approaches.
- Our IPsec gateway has a novel packet scheduling technique to improve GPU utilization. It considers real-world IP traffic where each packet has a different workload, and completely removes control-flow divergence due to different workloads.
- Our IPsec gateway is scalable. A higher-throughput IPsec gateway can be built with multiple moderate-throughput IPsec gateways (e.g., using channel bonding). This also provides high availability. Using the gateway composed of multiple moderate gateways with a failover feature helps to improve availability.

With three CPU cores and one GPU in the APU, the IPsec gateway achieves a throughput of 10.36 Gbps with an average latency of 2.79 ms to perform AES-CBC+HMAC-SHA1 for incoming packets of 1024 bytes. With an average latency of 3.71 ms, it achieves a throughput of 10.66 Gbps for incoming packets of random lengths. For a random mix of six crypto algorithms, it achieves a throughput of 17.42 Gbps with an average latency of 3.92 ms for incoming packets of 1280 bytes.

The rest of the paper is organized as follows. The next section discusses related work. Section 2 describes the background to understand our implementation. Section 3 presents the design and implementation of PIPSEA. Section 4 evaluates the performance of PIPSEA and discuss the cost effectiveness of our solution. Finally, Section 6 concludes the paper.

2. BACKGROUND

We briefly introduce the Accelerated Processing Unit (APU) and Heterogeneous System Architecture (HSA)[3] in this section. We also briefly describe the OpenCL programming model and GPU architectures in the context of OpenCL.

2.1 APU and HSA

Many processor vendors including Intel, AMD, NVIDIA, and ARM have released heterogeneous multicore processors where a GPU is integrated in the same chip. For example,

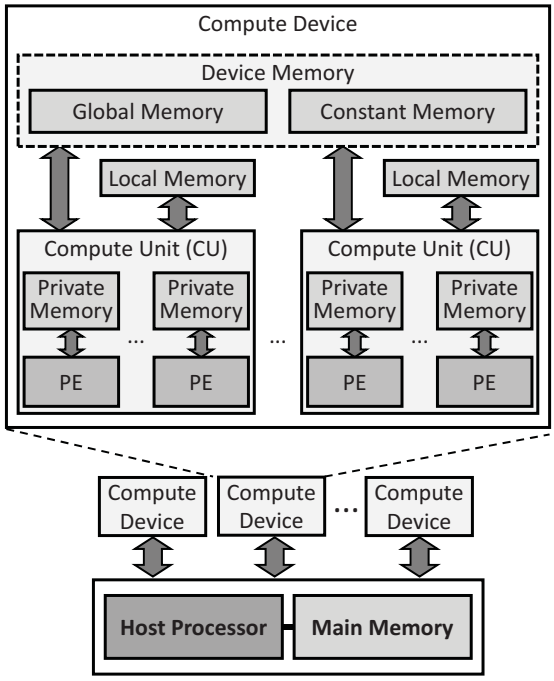


Figure 1: OpenCL platform model.

Intel's general-purpose Skylake processor integrates multiple x86 CPU cores and its HD graphics processor in a single chip. AMD's APU also has multiple x86 CPU cores and its Radeon GPU in a single chip.

However, the way of programming the integrated GPU (iGPU) in a heterogeneous multicore processor has been the same as that of the legacy GPGPU that uses a discrete GPU (dGPU). Thus, to perform computation on the iGPU or dGPU, data in the main memory (the CPU side) must be copied to the GPU-side memory if necessary. Moreover data coherence/consistency must be explicitly managed. This is because there is no real physically shared memory between the CPU and the GPU. To overcome such inefficiencies, the HSA has been proposed by the HSA foundation that is supported by many vendors, such as ARM, AMD, Qualcomm, TI, Samsung, etc.

The major goal of the HSA is to provide an efficient integrated environment for different kinds of computing devices such as CPUs and GPUs to system designers and software programmers. The legacy GPGPU (i.e., dGPU) has the most significant data transfer and coherence/consistency management overhead between the CPU and the GPU because each of them has a separate memory and the memories are connected via PCI-E bus. The iGPU has less overhead than the dGPU because the CPU-side memory and the GPU-side memory are connected by a data bus. The HSA enabled heterogeneous multicore processors provide a unified memory that allows coherent data sharing across all processors in the chip.

2.2 OpenCL and GPU Architectures

OpenCL (Open Computing Language) and NVIDIA CUDA [27] are the most widely used programming models for GPGPUs. OpenCL provides a common abstraction layer to the programmer across different architectures, such

as multicore CPUs, GPUs, Intel Xeon Phi coprocessors, FPGAs, and DSPs. This is a major advantage of using OpenCL over CUDA. Since the GPU in PIPSEA is programmed with OpenCL, our solution runs on different devices from different vendors. Note that NVIDIA GPUs also support OpenCL.

OpenCL platform model. As shown in Figure 1, the OpenCL platform model consists of a host processor connected to one or more *compute devices* (e.g., GPUs). Each compute device contains one or more *compute units* (CUs) and has compute device memory that is not visible to other compute devices. The compute device memory consists of global memory and read-only constant memory. Each CU contains one or more *processing elements* (PEs) and local memory. A PE is a processor and has its own private memory. PEs in the same CU share the local memory, and the local memory is not visible to other CUs. One of the general-purpose CPU cores in the APU becomes the host processor in PIPSEA.

OpenCL execution model. An OpenCL application consists of a *host program* and *OpenCL programs* both of which are based on C. The OpenCL program is a set of *kernels*. A kernel performs computation on the PEs within a compute device. The host program executes on the host processor in parallel with kernels and coordinates kernel executions.

An execution instance of a kernel is called a *work-item*. A work-item has a unique ID and can be treated as a thread. One or more work-items are grouped in a *work-group*. A work-group has also a unique ID. Before a kernel is launched by the host program, the host program defines the total number of work-items, the total number of work-groups, and the work-group size of the kernel. The OpenCL runtime distributes the kernel workload to CUs in the target device. The granularity of the distribution is a work-group. Work-items in a work-group execute *concurrently* in an SPMD (Single Program, Multiple Data) manner on the PEs of the associated CU. That is, they are context switched.

GPU architectures. OpenCL's compute device architecture is an abstraction of a common GPU architecture. For example, a CU and a PE correspond to a Streaming Multiprocessor (SM) and Stream Processor (SP) of NVIDIA GPUs, respectively. A CU is typically called a *GPU core*. The GPU in PIPSEA contains 8 GPU cores, and each core contains 64 PEs. The GPU also has the same memory architecture as that of OpenCL: global, constant, local, and private. In general, the access latencies of local and private memory are much lower than those of global and constant memory.

Hardware context switch. To hide long latency instructions such as global memory accesses, GPUs provide a hardware context-switch mechanism on CUs. The context switch unit on a CU is a *warp* (or *wavefront*) that is a set of work-items in a single work-group. Thus, a single work-group may consist of multiple warps running concurrently (i.e., context switched) on the same CU. The size of a warp is typically 32 for NVIDIA GPUs and 64 for AMD GPUs.

Work-group scheduling. An OpenCL kernel is typically executed by a number of work-groups that is larger than the number of CUs in the GPU. The hardware scheduler in the GPU dynamically determines which work-group is assigned to which CU [8]. A work-group that has a smaller ID value typically has a priority over an idle CU.

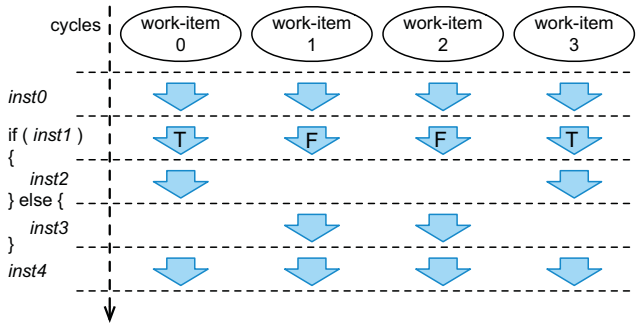


Figure 2: Control-flow divergence.

Control-flow divergence. When a warp is running on a CU, all work-items in the warp share a single program counter and execute the same instruction in a SIMD (Single Instruction, Multiple Data). This implies that all the work-items in the warp should take the same execution path. If some work-items in a warp take a different execution path due to a conditional branch, instructions in the path are predicated on the branch condition; if the condition is false, the instruction is suppressed, even though all the work-items follow the same execution path. This phenomenon in a GPU is called control-flow divergence. However, if the condition of a branch is evaluated to the same value for all work-items in a warp, control-flow divergence does not occur.

Figure 2 shows an example of control-flow divergence. There is a conditional branch due to an `if` statement. All the work-items in the warp take the same execution path but, because of the branch, *inst2* and *inst3* are predicated. Thus, it takes total 5 cycles. However, if the branch condition evaluates to the same value for all the work-items, there is no control-flow divergence, and the execution time is 4 cycles.

3. DESIGN AND IMPLEMENTATION

In this section, we describe the design and implementation of PIPSEA on an embedded APU. In the design, we use an embedded APU with at least two and at most three general-purpose CPU cores and one GPU, which is cost and power constrained. To reflect the reality of network traffic, we conservatively assume that each packet in the incoming network traffic belongs to a possibly different connection. That is, each packet has a different source IP address and a different destination IP address. This in turn implies that each packet needs to be handled with a different crypto algorithm and a different cipher (or authentication) key.

3.1 Overall Organization

Figure 3 shows the overall organization of PIPSEA. It is a multi-threaded program running in the user space and implemented on top of DPDK (Data Plane Development Kit)[2] and the OpenCL runtime[21].

The IPsec gateway consists of two threads: *packet I/O thread* and *IPsec thread*. Each thread is pinned to and running on a different CPU core. The Packet I/O thread has two modules: *receiver module* and *sender module*. The receiver module receives a bulk of packets from the RX queue in the NIC. The size of the bulk depends on the NIC. The maximum is 32 or 64 in general. After looking up the se-

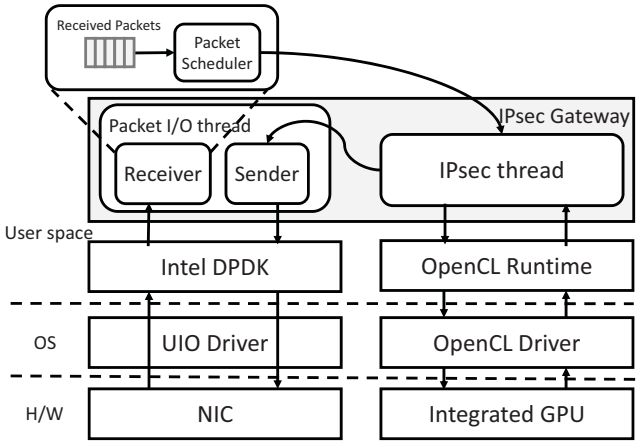


Figure 3: The organization of PIPSEA.

curity association database (SAD) and the security policy database (SPD), the *packet scheduler* in the receiver module groups the received packets into multiple batches by their crypto algorithms and lengths.

The batches of the packets from the packet scheduler are the input to the IPsec thread. The IPsec thread runs the OpenCL host program. It repeatedly launches an OpenCL kernel that performs IPsec packet processing including adding/modifying headers, encryption, decryption, and authentication. When a kernel execution is completed, the IPsec thread enqueues the processed packets to the *completion queue*. The *sender module* in the packet I/O thread sends the packets in the completion queue to the TX queue in the NIC.

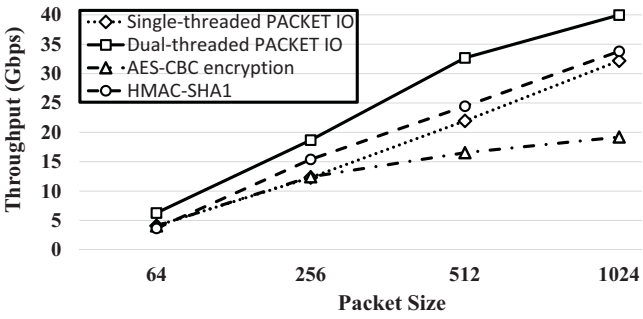


Figure 4: Throughput comparison of packet I/O and crypto algorithms.

Run-to-completion vs. pipeline. *Run-to-completion* and *pipeline* models are two widely used network packet processing models for multicore CPUs. While the latter assigns a different task to each core to process a packet, the former makes each core process a single packet from the beginning to the end including packet I/O.

PIPSEA is based on a pipeline model: the packet I/O thread performs packet I/O and the IPsec thread manages the GPU to process IPsec protocols. Since IPsec processing requires heavy computation, and the dedicated cores to the IPsec processing do not perform packet I/O, the pipeline model is better than the run-to-completion model for our case.

One of the benefits of the run-to-completion model is that packet I/O can be scaled over multiple CPU cores. Like the run-to-completion model, we may scale packet I/O over multiple CPU cores using multiple packet I/O threads. However, this may become a drawback in our case because the GPU can process only one chunk of packets at a time. When multiple chunks of packets are produced simultaneously by multiple packet I/O threads, each chunk should be processed by the GPU in a round-robin manner. This may lead to experiencing unacceptable high packet round-trip latencies.

As long as the throughput of single-threaded packet I/O is higher than the throughput of the IPsec processing on the GPU, exploiting multiple threads for the packet I/O is an overkill. Figure 4 compares the packet I/O-only throughput (including packet scheduling) and the throughput of crypto algorithms on the GPU. However, the single-threaded packet I/O throughput is lower than the best throughput of the GPU (at HMAC-SHA1). This implies that the single-threaded packet I/O is not powerful enough to achieve the maximum throughput of the IPsec gateway.

Therefore, we propose two models for PIPSEA: *single-threaded* and *dual-threaded* packet I/O models. The single-threaded packet I/O model is more resource-constrained and works when the APU has only two CPU cores. When there are two CPU cores available for packet I/O, we use the dual-threaded packet I/O model, where one thread handles the receiver module and the other handles the sender module. As shown Figure 4, the dual-threaded packet I/O throughput is always higher than the best throughput of the GPU. Thus, dual-threaded packet I/O is powerful enough to achieve the maximum throughput.

Packet I/O implementation. Fast packet I/O is a fundamental building block of network packet processing. Han *et al.* [12] and Kim *et al.* [22] report that the Linux network stack is inefficient because of unnecessary protocol handling and memory management overheads inside the Linux kernel. To overcome the inefficiency of the Linux network stack, a high performance user-level packet I/O framework such as DPDK [2] is necessary.

DPDK provides user-level zero-copy packet I/O API functions to eliminate data copy overhead between the network stack in the kernel space and the application in the user space. It also provides several useful libraries and drivers including a memory manager, buffer manager, queue manager, and optimized poll mode drivers to help develop high performance network applications. For instance, to improve performance by reducing TLB misses, its memory management exploits huge pages (2MB) for memory objects, such as memory pools, packet buffers, and rings.

Our IPsec gateway implementation is based on DPDK. We use the zero-copy packet I/O API functions. All queues and bins in PIPSEA are implemented with the lock-free rings provided by DPDK. It provides single/multi-producer, single/multi-consumer lock-free rings for not only high performance but also ease of development.

Instead of directly using the DPDK API functions, we implement a wrapper function for each DPDK API function used in the gateway. Thus, our implementation is not tightly coupled with DPDK. Another user-level packet I/O framework can be easily combined with our solution by just modifying the wrapper functions.

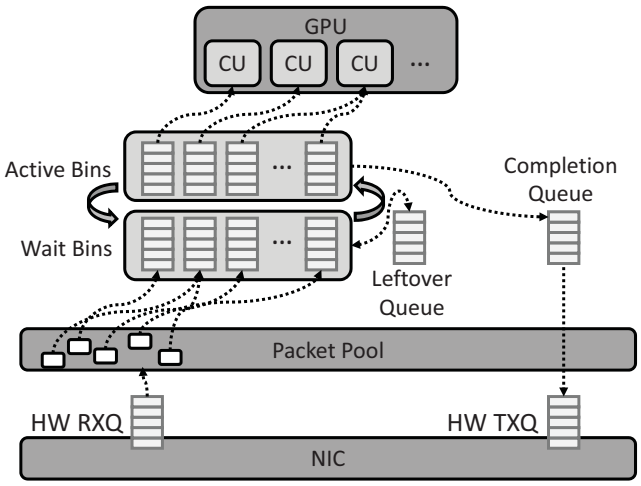


Figure 5: Queues and bins in PIPSEA.

3.2 Queues and Bins

While most of previous packet processing studies with dGPUs have made an effort to reduce data copy overhead between the CPU and the GPU, it is not a concern anymore for us because of the HSA supported by the APUs. The CPU cores and the GPU physically share the coherent main memory in the HSA.

Figure 5 shows the queues and bins used in PIPSEA. All packets from the NIC are stored in the packet pool managed by DPDK and allocated in the main memory. Accesses to packets are done using the same pointers to the packets on both sides of the CPU and the GPU because of the HSA. Thus, no packet copy occurs between the CPU and the GPU in our approach.

To efficiently pipeline packet scheduling and IPsec protocol processing, we implement two sets of bins: *active* and *wait*. Using these bins, we implement an efficient double buffering mechanism to overlap the computation on the GPU and the communication between the packet I/O thread and the GPU.

As mentioned before, the packet scheduler groups the received packets in to the wait bins by their crypto algorithms and lengths. While the packet scheduler fills in the wait bins, the OpenCL kernel on the GPU processes the packets in the active bins. After the GPU finishes processing the active bins, the packets in the active bins are enqueued to the completion queue by the IPsec thread.

To reduce the degree of packet reordering, the processed packets are enqueued to the completion queue in the same order as they were received.

Then, the IPsec thread switches the roles of the active bins and the wait bins. The OpenCL kernel on the GPU starts to process the new packets in the active bins, and the packet scheduler fills in the wait bins with new received packets. Our IPsec gateway continuously repeats this process. If a received packet cannot be scheduled in the wait bins because there is no appropriate bin for the packet, it is inserted to the *leftover queue* and will be scheduled in the next turn.

3.3 Scheduling Packets

The OpenCL kernel that performs IPsec processing on the GPU consumes a chunk of multiple packets (i.e., the packets in the active bins) at each kernel launch. The role of the packet scheduler in the packet I/O thread is to produce and schedule a chunk of packets to achieve the best utilization of the GPU hardware resources.

The crypto algorithm of each packet is determined by its source and destination IP addresses. The packet scheduler performs this task by looking up the SAD and SPD that are managed by the IKE (Internet Key Exchange) protocol. We assume that the SAD and SPD are predefined because the implementation of the IKE protocol is beyond the scope of this paper.

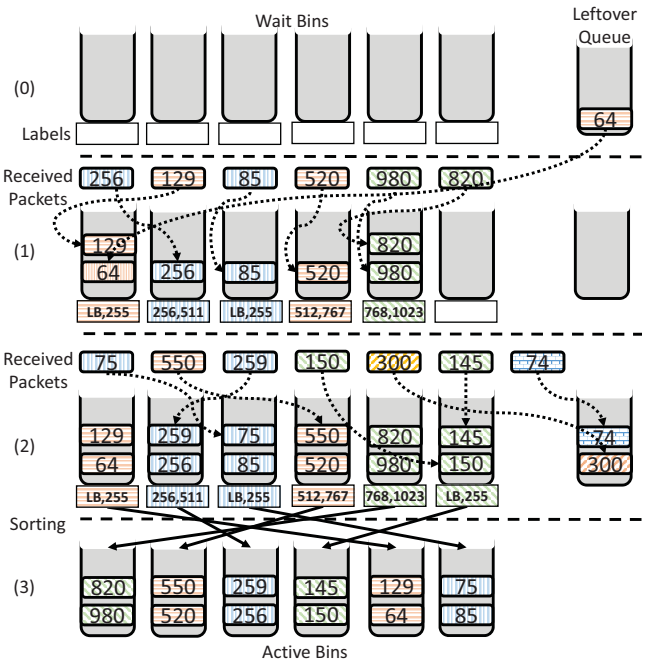


Figure 6: A packet scheduling example.

Packet scheduling algorithm. Figure 6 shows a snapshot produced by our packet scheduling algorithm. A different pattern or color in a packet indicates a different crypto algorithm. The number in a packet shows its length. When the packet scheduler starts to fill in the wait bins (step 0 in Figure 6), all the wait bins are empty and not labeled yet. The leftover queue contains some packets that could not be scheduled in the previous turn.

In our packet scheduling algorithm, the bin size is a multiple of the warp size in the GPU. The packets in an active bin will be grouped together and become a work-group when the OpenCL kernel launches to the GPU. Each work-item in a work-group processes a packet.

When the packet scheduler categorizes the packets in to the wait bins by their crypto algorithms and lengths, it categorizes the packets in the leftover queue first, and then moves on to the received packets (step 1 and 2 in Figure 6).

When a packet is processed by the packet scheduler, it searches for a bin labeled with the packet's algorithm and length range. For example, labels for the crypto algorithms include AES-CBC-encryption, AES-CBC-decryption, HMAC-SHA1, AES-CBC-encryption+HMAC-SHA1, AES-CBC-decryption+HMAC-SHA1, etc. In addition, labels for the length ranges include $[LB, 255]$, $[256, 511]$, $[512, 767]$,

[768, 1023], [1024, 1279], [1280, UB], where LB and UB are the lower and upper bounds of packet lengths in IPsec.

If successful, the packet scheduler inserts the packet in the bin found. Otherwise, it inserts the packet in an empty bin and labels the bin with the packet's algorithm and length range. If there is no empty bin available, the packet is inserted to the leftover queue. As a result, each bin is filled with the packets of the same algorithm and the same length range.

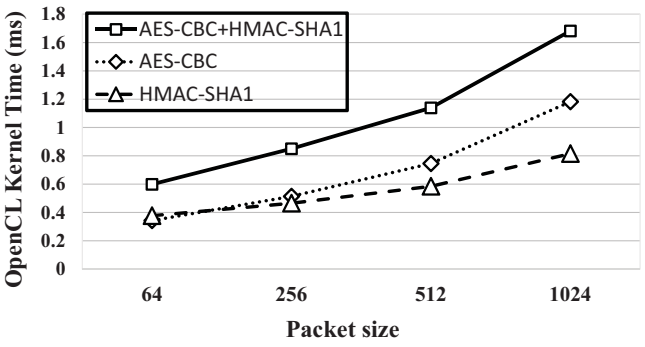


Figure 7: Average OpenCL kernel execution time for 2048 packets.

At the time of the OpenCL kernel completion, the IPsec thread switches the roles of the active bins and the wait bins. Then, it launches the OpenCL kernel to process the active bins. Before the OpenCL kernel launch, it sorts the active bins in a decreasing order by their estimated GPU processing time to help the GPU hardware scheduler (step 3 in Figure 6).

The bin processing time is estimated by the bin's crypto algorithm and packet length range. After measuring IPsec processing time on the GPU (i.e., OpenCL kernel execution time) with different packet sizes and crypto algorithms, we use the result to estimate processing time for each bin. Figure 7 shows a result of measuring IPsec processing time on the GPU with different packet sizes and crypto algorithms. The OpenCL kernel execution time is proportional to the packet length varying from 64B to 1024B.

Zhang *et al.* [32] propose a software solution to avoid control flow divergence. Similar to PIPSEA, their solution groups GPU threads to avoid control flow divergence. However, it does not provide a method to resolve the load imbalance between GPU threads. It neither sorts the groups nor uses bins to achieve the load balance.

3.4 Effects of Packet Scheduling

As a result of our packet scheduling algorithm, each bin is filled with the packets of the same crypto algorithm and the same length range. In addition, bins are sorted in a decreasing order by their estimated processing time on the GPU. Note that the bin size is a multiple of the warp size in the GPU, and each bin becomes a work-group of the OpenCL kernel.

Avoiding control-flow divergence. If packets in a work-group execute different crypto algorithms, they will take different execution paths. As a result, significant control-flow divergence occurs. As mentioned in Section 2.2, the control-flow divergence is one of major performance factors in GPU programming[4, 27].

By grouping incoming packets of the same crypto algorithm in to a work-group together, we fully eliminate control-flow divergence caused by different crypto algorithms. In addition, by grouping incoming packets of similar lengths in to the same work-group, we minimize control-flow divergence caused by conditional branches (e.g., loop exit conditions) on the packet length in a crypto algorithm.

Avoiding inter-work-group load imbalance. As mentioned in Section 2.2, the GPU hardware dynamically assigns work-groups to idle CUs in the GPU, and a work-group that has a smaller ID value has a priority over an idle CU. By sorting bins in a decreasing order of their estimated processing time, the first bin in the sorted order maps to the work-group of the smallest ID when the OpenCL kernel launches. This implements a variation of the LPTF (Longest Processing Time First) algorithm. It avoids the load imbalance on CUs caused by different amounts of work between different work-groups.

Optimality consideration. Assigning work-groups to multiple CUs can be mapped to a multiprocessor scheduling problem. Note that our scheduling approach achieves a near-optimal solution. Since this problem is known to be NP-complete[9], we use an algorithm that gives a near-optimal solution to boost the speed.

LPTF sorts the tasks by its processing time and assigns them to the processor with the least amount of assigned tasks. Similarly, our packet scheduler sorts the bins by its estimated processing time, and the hardware scheduler assigns them to idle CUs one by one. LPTF is known to achieve an upper bound less than 4/3 of the minimum processing time [10].

Avoiding intra-work-group load imbalance. On the other hand, if a work-group consists of packets with multiple lengths, the processing time of the work-group is determined by the longest packet in the batch because the processing time of a crypto algorithm is typically proportional to the packet length (Figure 7). By grouping incoming packets of similar lengths in to the same work-group, we minimize such load imbalance between work-items in a work-group.

Packet reordering problem. Packet reordering occurs naturally by the network environment, such as route change, parallelism within a router, packet retransmission, etc. The packet reordering problem may adversely affect the performance and efficiency of the packet destination that needs to correct the order of packets.

Our implementation makes some packets reordered because the packets enqueued to the leftover queue are processed and transmitted later than their peers. However, we expect that the degree of reordering is not significant because our implementation preferentially schedules the packets in the leftover queue in the next OpenCL kernel launch. We observe that only less than 0.0001% of packets are reordered using 128 bins.

3.5 OpenCL Kernel Optimization

Assuming that packet I/O is fast enough, the OpenCL kernel execution time determines the throughput of IPsec gateway and the round-trip latencies of IPsec packets. Thus, we optimize the OpenCL kernel as much as possible considering important performance factors of GPUs, such as global memory access coalescing, control-flow divergence,

APU system	
Processor	AMD RX-421BD APU (4 x86 cores, 8 GPU cores)
RAM	2 x 8 GB (DDR4 2133 MHz)
NIC	Intel X710DX4 (4-port 10GbE)
OS	CentOS 6.5
Software	AMD OpenCL SDK 3.0 Intel DPDK 2.1
dGPU system	
Processor	2 x Intel Xeon CPU E5-2680 v3 (total 24 x86 cores)
GPU	2 x AMD FirePro S9100 or 2 x AMD Radeon R9 Nano
RAM	16 x 16 GB (DDR4 2133 MHz)
NIC	2 x Intel X710DX4 (total 8-port 10GbE)
OS	CentOS 6.5
Software	AMD OpenCL SDK 3.0 Intel DPDK 2.1

Table 1: System configuration.

occupancy, vectorized accesses to the global memory, and exploiting the local memory[4, 27].

Our packet scheduling algorithm enables PIPSEA to implement all IPsec crypto algorithms in one monolithic OpenCL kernel that is launched repeatedly by the IPsec thread. It performs all functions of IPsec including packet header processing, encryption, decryption, and authentication. Each work-item in the kernel handles a packet. Each work-group handles the packets contained in a bin.

For cipher algorithms, we focus on AES-CBC that is one of the most widely used cipher algorithm for IPsec. We also implement HMAC-SHA1, a commonly used authentication algorithm in IPsec. Other crypto algorithms in IPsec, such as AES-CTR, 3DES, AES-GCM, HMAC-SHA256, etc. can be easily added to the OpenCL kernel.

3.6 Tuning Parameters

Number of bins and bin size. The number of wait bins *times* the bin size is the total number of packets that are processed by one OpenCL kernel launch (i.e., chunk size). This significantly affects the packet round-trip latency and the throughput. Thus, both the number of the bins and the bin size are tuning parameters.

Granularity of packet length ranges. The granularity of the packet length range is also a tuning parameter, and it is closely related to the number of bins. The finer the length range, the more the reduction of the intra-work-group load imbalance and control-flow divergence. However, fine-grained length range increases the number of bins and gives more bin searching overhead to the packet scheduler. Moreover, if we fix the number of bins available, the fine-grained length range makes more packets to go to the leftover queue.

4. EVALUATION

This section describes the performance evaluation result of PIPSEA. In addition, we discuss the cost effectiveness of our proposal.

4.1 Methodology

System configuration. PIPSEA is built with an x86-based AMD embedded APU processor, 16 GB of RAM, and one Intel NIC with four 10GbE ports. Since PIPSEA runs also on dGPUs as long as they support the HSA, we run PIPSEA on two types of dGPUs: a professional high-end dGPU (AMD FirePro S9100) and a gaming GPU (AMD Radeon R9 Nano). Table 1 summarizes the specification of the system.

Packet generation. To evaluate PIPSEA, we use a DPDK-based packet generator. It is directly connected to PIPSEA via the four-port 10GbE NIC. Its role is both the source and the sink of packets. It produces up to 40 Gbps network traffic that contains packets with various types of lengths, such as fixed lengths, uniform random lengths, and lengths that follow a specific network traffic distribution pattern. For the packets that follow a specific network traffic distribution pattern, we choose IMIX (Internet Packet Mix) that resembles the real-world traffic in the distribution of packet lengths and defined by Intel. It consists of 61.22% of 60-byte packets, 23.47% of 536-byte packets, and 15.31% of 1360-byte packets [14].

The packet generator also produces packets that have various types of source and destination IP addresses. They include fixed addresses and uniform random addresses. By exploiting this feature, we evaluate PIPSEA with packets that require randomly-mixed crypto algorithms for IPsec processing.

Throughput and latency measurements. The packet generator provides a feature to measure the round-trip latency of a packet. After generating packets with time stamps, it calculates the round-trip latency of the packets when it receives them and stores the log in a file. We measure the latency of PIPSEA using this feature.

PIPSEA performs both IPsec inbound and outbound processing for decapsulation and encapsulation of ESPs (Encapsulation Security Payloads), AHs (Authentication Headers), and both. While the IPsec encapsulation increases the packet size, the decapsulation decreases the packet size. To be clear on the throughput metric, we measure the incoming throughput, not the outgoing throughput of PIPSEA. In addition, the *packet size* means the length of the ethernet frame in bytes, including the frame check sequence (32-bit CRC). In the IPsec decapsulation, the packet size is defined by the length of the plain packet excluding IPsec headers and paddings.

RFC 2544 [5] describes benchmarking methodologies for network interconnect devices. We faithfully follow RFC 2544 to measure the throughput and the latency of PIPSEA.

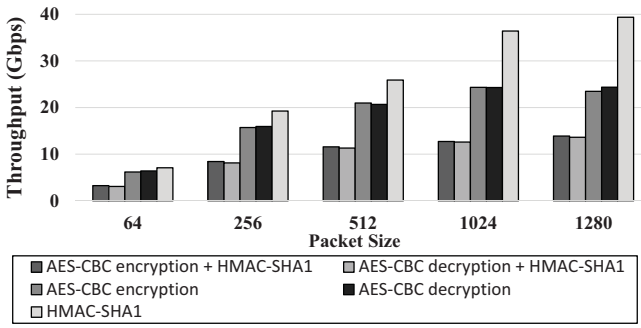


Figure 8: The throughput of the crypto algorithms with pre-generated packets in main memory.

Crypto algorithms used. We implement some of the most widely used IPsec crypto algorithms in the OpenCL kernel. We implement AES-128-CBC encryption/decryption and HMAC-SHA1.

Figure 8 shows the throughput of the crypto algorithms on the embedded APU without Packet I/O and IPsec header processing. The crypto algorithms process pre-generated packets stored in main memory, and the packets do not include any headers. Note that the packet size is just the payload size. Thus, the result shows the maximum throughput that can be achieved for each algorithm with the embedded APU. For example, the achievable throughput is up to 13.87 for AES-CBC encryption + HMAC-SHA1 and up to 39.37 Gbps for HMAC-SHA1 with 1280-byte packets.

In all cases, the larger the packet, the higher the throughput. This is because the OpenCL kernel launch overhead is relatively small for large packets. For example, the overhead for 64-byte packets is about 30% of the total AES-CBC encryption processing time. For 1024-byte packets, it is just 7%.

4.2 Performance of PIPSEA

We measure the performance PIPSEA for six combinations of crypto algorithms and various packet sizes including 64B, 256B, 512B, 1024B, 1280B, uniform random lengths(64 $\sim$ 1280), and IMIX. The six combinations include AES-CBC+HMAC-SHA1 (ENC/DEC), AES-CBC (ENC/DEC), and HMAC-SHA1 (ENC/DEC), where ENC and DEC stand for IPsec encapsulation and decapsulation, respectively. Each incoming packet has a different source IP address and a different destination IP address. Thus, each packet requires a different IPsec crypto algorithm because its crypto algorithm is determined by its source and destination IP addresses.

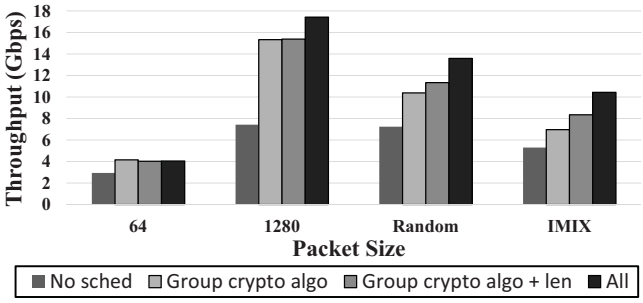


Figure 9: Effects of packet scheduling.

Effects of the packet scheduling algorithm. Figure 9 shows the effect of the proposed packet scheduling technique. Labels No sched, Group crypto algo, Group crypto algo + len, and All stand for no packet scheduling, grouping only by crypto algorithms into the bins, grouping by crypto algorithms and packet lengths, and fully applying the proposed scheduling algorithm including bin sorting, respectively. Packets of the random mix of the six crypto algorithms is the input, and the granularity of packet length ranges is 256.

In this evaluation, a chunk of 4096 packets and 64 bins are used. The chunk size is the number of packets processed in one OpenCL kernel launch. The chunk size is equal to the number of total work-items that execute the kernel. In addition, the number of bins is equal to the number of work-groups that execute the kernel. Thus, the work-group size is determined by the chunk size divided by the number of bins. We vary the size of packets.

For small packets, the packet scheduling effect is not significant because load imbalance due to different algorithms is not significant. In addition, the packet scheduling overhead is relatively large compared to the packet processing time.

The effect of each step in PIPSEA packet scheduling is apparent for Random and IMIX. For IMIX, the throughput with the packet scheduling is almost twice bigger than that without the scheduling. The proposed packet scheduling algorithm improves the throughput by 1.87 and 1.96 times for Random and IMIX, respectively.

To see the effectiveness of the bin sorting, we compare Group crypto algo + len and All in Figure 9. The bin sorting optimization improves the throughput by 1.19 and 1.24 times for Random and IMIX, respectively.

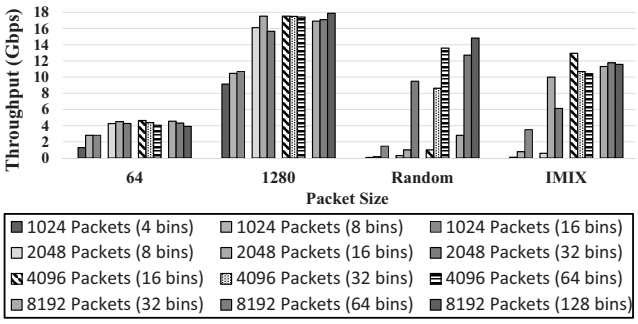


Figure 10: The throughput of PIPSEA with different chunk sizes and numbers of bins.

Effects of the tuning parameters. We evaluate PIPSEA with different tuning parameters described in Section 3.6, such as chunk size and the number of wait bins.

Figure 10 shows the throughput of PIPSEA with different chunk sizes and numbers of bins. A packet is processed by one of the six IPsec crypto algorithm combinations at random. The granularity of packet length ranges is 256B. The experiment is performed with the dual-threaded packet I/O model described in Section 3 .

For 64- and 1280-byte fixed-length packets, we see that the throughput is saturated with a chunk of 2048 packets and 16 bins. With fixed-length packets, since there are variations only in crypto algorithms, the chunk size is the more important performance factor than the number of bins.

For both Random and IMIX packets, the throughput is proportional to the chunk size with a fixed number of bins. In addition, since the possible number of combinations between different packet length ranges and different algorithms is much bigger than 8 bins, the throughput is very low with 4 and 8 bins. For Random packets, we see that the throughput is proportional to the number of bins with a fixed chunk size.

The highest throughput for Random is 14.82 Gbps with a chunk of 8192 packets and 128 bins. The second highest for Random is 13.58 Gbps with a chunk of 4096 packets and 64 bins. Thus, we may choose 8192 as the chunk size and 128 as the number of bins for PIPSEA as long as it does not cause long packet latencies. However, a large chunk size may

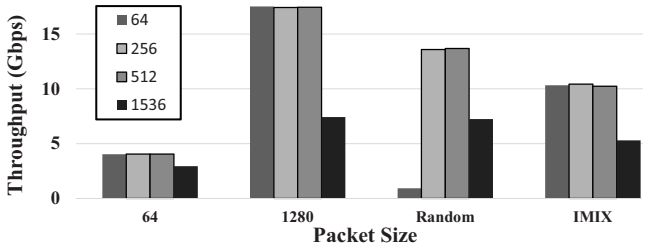


Figure 11: Throughput with different granularities of packet length ranges.

lengthen the OpenCL kernel execution time and adversely affect the latency. Since the average latency of the former is 5.38ms and that of the latter is 3.92ms, a chunk of 4096 packets and 64 bins is a better option.

Figure 11 shows the effect of the granularity of packet length ranges. For this evaluation, the input is the random mix of the six crypto algorithms. The chunk size is 4096, and the number of bins is 64. We evaluate four different granularities of packet length ranges, 64, 256, 512 and 1536. The numbers of packet length ranges are 24, 6, 3 and 1 for the granularities of 64, 256, 512 and 1536, respectively.

For the fixed-length packets, the result shows that the packet scheduling overhead due to the fine-grained ranges is negligible. In addition, coarse-grained ranges adversely affect the throughput. For Random and IMIX, we see that too fine-grained or coarse-grained ranges adversely affect the throughput. For Random, the throughput with the granularity of 64 is much smaller than others because the number of bins is not enough such fine-grained length ranges. Overall, too fine-grained or coarse-grained length ranges are bad to achieve high throughput.

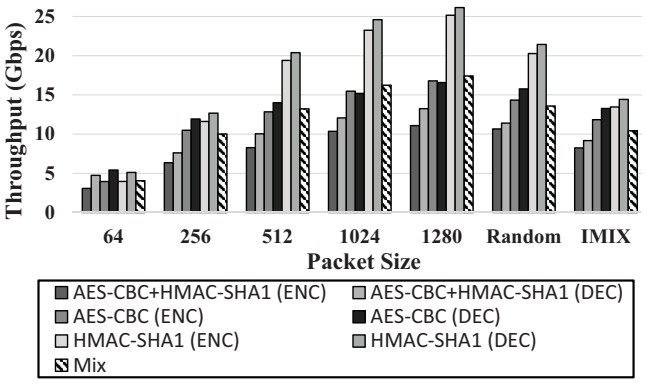


Figure 12: Throughput for different crypto algorithms.

Throughputs for different crypto algorithms. Figure 12 shows throughputs of different crypto algorithms with a chunk of 4096 packets and 64 bins. In Figure 12, we vary packet lengths. The bar labeled Mix stands for a random mix of the six different crypto algorithms.

In most of cases, the decapsulation throughput is higher than the encapsulation throughput because the throughput is measured over the incoming traffic, not outgoing traffic.

The throughput of AES-CBC+HMAC-SHA1(ENC) is 3.08 Gbps for 64-byte packets and 11.09 Gbps for 1280-byte packets. These values are smaller than those of AES-CBC+HMAC-SHA1 with the pre-generated packets stored in main memory due to the packet I/O overhead and the IPsec header handling overhead.

In all cases, the throughput of Mix is better than that of AES-CBC+HMAC-SHA1(ENC) that is the most time consuming crypto algorithm combination among the six combinations. This is because many crypto algorithms in Mix compete for the wait bins.

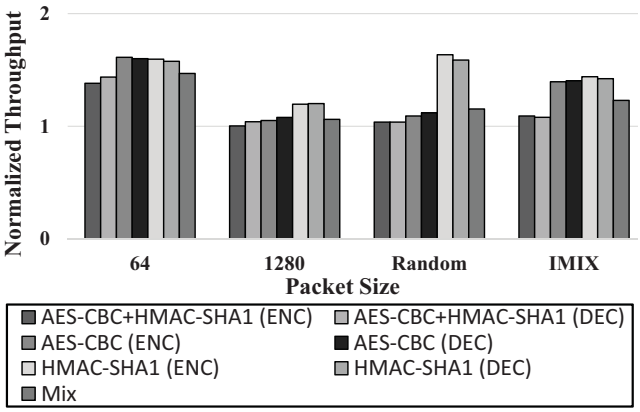
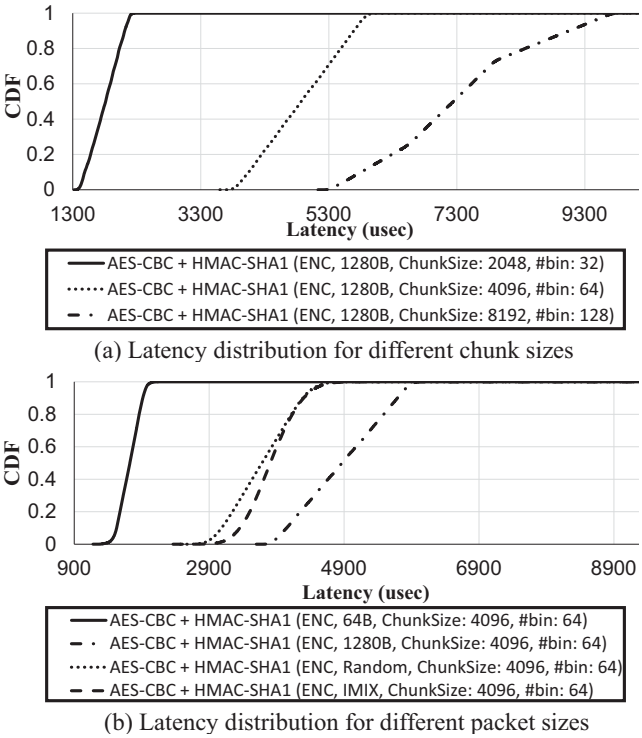


Figure 13: Comparison of the single-threaded and dual-threaded packet I/O models.



(a) Latency distribution for different chunk sizes

(b) Latency distribution for different packet sizes

Figure 14: Latency distribution.

Packet I/O models. PIPSEA provides two types of the packet I/O model: *single-threaded packet I/O* and *dual-threaded packet I/O*. Figure 13 shows the comparison between them. The granularity of the packet length range, the

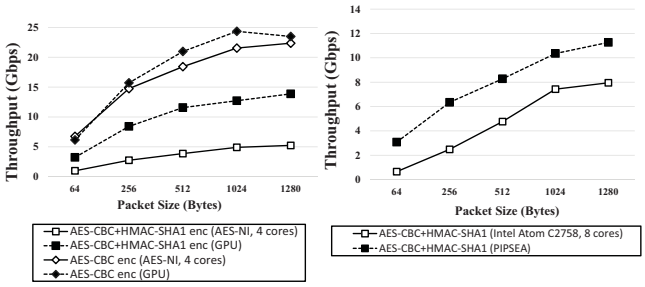
chunk size, and the number of bins used are 256, 4096, and 64, respectively.

In all cases, the throughput of the dual-threaded packet I/O is comparable to or better than that of the single-threaded packet I/O. For the most time consuming AES-CBC+HMAC-SHA1, single-threaded packet I/O is comparable to the dual-threaded packet I/O because the packet I/O overhead is relatively small compared to the IPsec processing time. On the other hand, for the least time consuming HMAC-SHA1, the throughput of the single-threaded packet I/O is much smaller than that of the dual-threaded packet I/O.

Latencies. Figure 14 shows the latency distribution of PIPSEA for AES-CBC+HMAC-SHA1(ENC). The input is 1280-byte packets. To measure the latency, we have offered the maximum incoming traffic rate that is same as the measured maximum throughput by following the rules in RFC 2544 [5]. In addition, the numbers in Figure 14 are conservatively measured. The latency includes the end-to-end latency of PIPSEA and the overhead of the software packet generator, such as packet I/O overhead and the per-packet timestamping overhead. To be more accurate, we need to obtain the end-to-end latency of PIPSEA excluding the packet generator overhead.

The latency distribution of PIPSEA spreads from hundreds of usec to thousands of usec due to the kernel execution time on the GPU. Figure 14 (a) shows that the latency increases as the chunk size is increases. The smaller the chunk size, the narrower the distribution because the smaller the number of packets, the shorter the kernel processing time on the GPU. The latencies for Random and IMIX shows a similar distribution to Figure 14 (b).

NBA [22] reports that their average latency of AES-CTR+HMAC-SHA1 for packets of 1024B is around 3.8ms when the incoming rate is 3 Gbps. Note that NBA uses two high-end dGPUs. The average latency of PIPSEA for AES-CBC+HMAC-SHA1 is 2.79 ms for packets of 1024B when the incoming rate is 10.36 Gbps.



(a) Throughput of AES-NI and GPU (b) Throughput of QAT and PIPSEA

Figure 15: Throughputs of AES-NI and Intel QuickAssist.

4.3 AES-NI and QuickAssist Technology

AES-NI (Advanced Encryption Standard-New Instruction) is a set of instructions to improve the performance of cryptography. Each CPU core in PIPSEA supports AES-NI. We implement AES-CBC and AES-CBC+HMAC-SHA1 using AES-NI, and compare the encryption performance of the two crypto algorithms using the four CPU cores with AES-NI and using only the GPU in the APU.

Figure 15 (a) shows that the performance of AES-CBC+HMAC-SHA1 using AES-NI is much worse than that of using the GPU because the GPU is much faster than the CPU cores to perform HMAC-SHA1. Because AES-NI is specialized for AES algorithms, it does not work well with HMAC-SHA1.

The Intel QuickAssist technology (QAT) is a hardware-assisted acceleration technology to boost performance of cryptography, compression and pattern matching applications. Using a sample QAT application provided by DPDK on an Intel embedded processor Atom C2758 with 8 cores, we compare its AES-CBC+HMAC-SHA1 performance with PIPSEA. Figure 15 (b) shows that PIPSEA outperforms Atom C2758 in all different packet sizes. The AES-CBC+HMAC-SHA1 throughput of Atom C2758 is up to 7.95 Gbps for packets of 1280B.

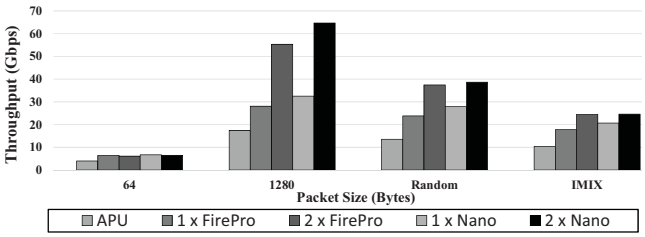


Figure 16: Throughputs of APU and dGPUs for the random mix of the six crypto algorithms.

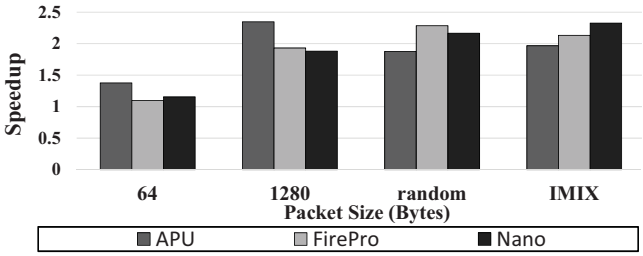


Figure 17: Speedups of the packet scheduling algorithm for the random mix of the six crypto algorithms on the APU and the dGPUs.

4.4 Discrete GPUs

Figure 16 shows the throughputs of the APU and two dGPUs for the random mix of the six crypto algorithms. 2xFirePro and 2xNano stand for the cases when dual dGPUs are used to run PIPSEA. Since the computing power of each dGPU is much larger than that of the APU, the throughputs of the dGPUs are much higher than that of the APU. For example, the throughput for the packets of 1280B with dual AMD Radeon R9 Nano GPUs is 64.73 Gbps. However, the cost effectiveness of the dGPUs is much lower than the APU (Table 2).

Figure 17 shows the effectiveness of the packet scheduling algorithm for the APU and the dGPUs. The speedup is obtained over the case when the scheduling algorithm described in Section 3.3 is not used. In this case, we just group the packets in their incoming order and offload them to the GPU. For example, the algorithm improves the throughput by 2.13 times for IMIX on the AMD Radeon R9 Nano

Category	**PIPSEA** w/ APU	**PIPSEA** w/ dGPU		Intel Atom w/ QAT	Intel QAT Chipset	Multicore CPU w/ AES-NI	Multicore CPU	NBA [22] w/ dGPU	Helion ESP w/ FPGA
HW Model	AMD APU RX-421BD	Intel Xeon E5-2620v3 + AMD FirePro S9100	Intel Xeon E5-2620v3 + AMD Radeon R9 Nano	Intel Atom C2758	Intel DH8950 PCH	Intel i7-5820K	Intel i7-5820K	Intel Xeon E5-2670 + NVIDIA GeForce GTX 680	Xilinx Artix-7
Type	Standalone	Standalone	Standalone	Standalone	Accelerator	Standalone	Standalone	Standalone	Accelerator
Price	$100.00	$2,944.00	$1,544.00	$258.00	$268.00	$439.00	$439.00	$4,404.00	$34.37
Performance Measurement	IPsec CBC+SHA1	IPsec CBC+SHA1	IPsec CBC+SHA1	IPsec CBC+SHA1	Unknown	In-memory CBC+SHA1	In-memory CBC+SHA1	IPsec CTR+SHA1	IPsec CBC+SHA1
Throughput	11.1 Gbps	36.5 Gbps	42 Gbps	7.9 Gbps	43 Gbps	12.3 Gbps	8.1 Gbps	32 Gbps	1.16 Gbps
Mbps/$	110.90	12.40	27.20	30.62	160.45	28.02	18.45	7.27	33.75
Power	35W	320W	270W	20W	20W	140W	140W	620W	-

Note:

1. We estimate the cost of a standalone system by the sum of the prices of CPUs, main memory and additional devices such as dGPUs. We assume that the price of the main memory is $50 (two 4GB DIMMs, $25 each).
2. We estimate the cost and the power consumption of an accelerator system by the price and the power consumption of the chip itself. Thus, its cost and power consumption may be underestimated.
3. We assume that PIPSEA w/ dGPU exploits the low-end Intel Xeon CPUs, because it only uses the dGPUs for the IPsec processing.
4. PIPSEA, Multicore CPU, NBA [22] w/ dGPU and Helion ESP Engine have programmability. The others exploit dedicated hardware for packet encryption/decryption and does not have programmability. The performance of Intel Atom w/ QAT (Intel QuickAssist Technology), Multicore CPU w/ AES-NI and Multicore CPU are measured by ourselves.
5. The performance numbers of Intel QAT chipset and Helion ESP Engine are obtained from the documents [16, 13].
6. The performance of NBA w/ dGPU is obtained from the its paper [22].
7. There is no information available about the power consumption of Helion ESP Engine.

Table 2: Cost-effectiveness comparison of existing IPsec solutions and PIPSEA.

GPU. The result indicates that the packet scheduling algorithm performs well not only for the APU, but also for the dGPUs.

4.5 Cost-effectiveness

Table 2 is the cost-effectiveness comparison of existing IPsec solutions and PIPSEA. Existing solutions are categorized in *standalone* and *accelerator* systems. An accelerator system needs an additional CPU, memory, etc. to operate it properly. In this table, the additional cost for the accelerator system is not included.

The cost effectiveness (Mbps/$) of PIPSEA is 110.90. That is much higher than any other existing solutions except the Intel QAT Chipset (DH8950 PCH) whose cost effectiveness is 160.45. However, it is an accelerator system and requires additional cost to operate the chipset properly. Moreover, Intel QAT chipset is inflexible, unlike PIPSEA.

The cost effectiveness of PIPSEA is about 15 times higher than NBA [22]. Because PIPSEA achieves good performance using a single embedded APU of four CPU cores and eight GPU cores that worths about US$50.

The energy efficiency of PIPSEA is also higher than other solutions. The power consumption of the embedded APU is only 35W. Although the power consumption of the Intel QAT accelerator (20W) is lower than the APU, an additional CPU is required to make the QAT accelerator run.

5. RELATED WORK

In this section, we classify the related work in two categories: hardware-accelerated IPsec and dGPU-based network security.

5.1 Hardware-accelerated IPsec

Many studies have been done to accelerate IPsec using special purpose hardware. Ha *et al.* [11] propose an ASIC design of IPsec hardware accelerator. Its estimated throughput is 200 Mbps. Hodjat *et al.* [15] propose a specialized cryptographic coprocessor to accelerate AES processing. The throughput of the AES coprocessor is 3.43 Gbps. Dandalis *et al.* [7], Chodowiec *et al.* [6], and Kakarountas *et al.* [19] implement IPsec solutions with FPGAs. The FPGA implementation of SHA-1 from Kakarountas *et al.* achieves a throughput up to 4.2 Gbps. Thoguluva *et al.* [28] implement IPsec on a mobile application processor SoC that contains a programmable security processor. The IPsec implementation offloads cryptographic algorithms to the security processor. Its performance is up to 8 Mbps. Meng *et al.* [25] propose a high-performance network processor solution for IPsec using Cavium OCTEON CN58XX. Their implementation achieves 20 Gbps with packets of 1024B.

5.2 Discrete GPU-based Network Security

Cryptographic network applications. A few studies including GASPP [30], PacketShader [12], NBA [22], and SSLShader [18] have been done to accelerate cryptographic network applications on dGPUs. GASPP [30], NBA [22], and PacketShader [12] are dGPU-based network packet processing frameworks. SSLShader is an SSL (Secure Socket Layer) acceleration solution that exploits dGPUs. NBA and PacketShader implement an IPsec protocol to show their packet processing performance on a dGPU. NBA implements AES-CTR and HMAC-SHA1, and PacketShader implements AES-CBC and HMAC-SHA1. GASPP [30] implements just a simple AES-CBC encryption application to encrypt all incoming packets.

Among others, GASPP introduces a packet scheduling optimization to reduce control-flow divergence and load imbalance between GPU threads. It relies on sorting packets in the GPU side, the overhead of which is not negligible. In addition, control-flow divergence due to different workloads may not be completely removed in GASPP. On the other hand, our IPsec packet scheduling algorithm completely removes control-flow divergence due to different workloads and provides a near optimal load balancing algorithm between GPU cores in addition to between GPU threads. Note that load balancing between GPU cores is not considered in GASPP. Moreover, packet scheduling occurs on the CPU

side in parallel with the kernel execution on the GPU side. Thus, our packet scheduling overhead is fully hidden.

Other applications. Besides cryptographic network applications, dGPUs have been used to accelerate other network-security-related applications. Gnort [29], MIDeA [31], and Kargus [17] use dGPUs to provide high-performance network intrusion detection.

Overall differences. To the best of our knowledge, PIPSEA is the first IPsec solution using an embedded APU. Most of the previous GPU-based approaches are designed for multiple high-end multicore CPUs and multiple high-end dGPUs. They exploit many CPU cores to maximize the performance of packet processing. It is hard to directly adapt these approaches to an embedded APU because it has a very restricted number of CPU cores (2–4 cores). Moreover, all previous GPU-based approaches are implemented in CUDA [27]. Consequently, they work only for NVIDIA GPUs. Although the performance of previous solutions is higher than that of our solution, the cost effectiveness of PIPSEA is much better than theirs.

Most of previous GPU-based IPsec approaches do not seriously consider real-world network traffic, where each incoming packet potentially has a different length and requires a different crypto algorithm. If we consider such network traffic, control-flow divergence and load imbalance in the GPU become more severe and adversely affects performance. Our packet scheduling algorithm handles such network traffic and avoids the control-flow divergence and load imbalance.

6. CONCLUSIONS

In this paper, we present the design and implementation of a high-performance IPsec gateway, PIPSEA, using a low-cost commodity embedded APU. PIPSEA is implemented using DPDK and OpenCL. IPsec protocols are implemented in OpenCL to offload IPsec processing on the on-chip GPU. We propose a packet scheduling algorithm that fully exploits a common GPU architecture and improves GPU utilization significantly. It avoids control-flow divergence and load imbalance caused by different workload between packets. We evaluate PIPSEA with different tuning parameters and a variety of input traffics.

With three CPU cores and one GPU in the APU, the IPsec gateway achieves a throughput of 10.36 Gbps with an average latency of 2.79 ms to perform AES-CBC+HMAC-SHA1 for incoming packets of 1024 bytes. With an average latency of 3.71 ms, PIPSEA achieves a throughput of 10.66 Gbps for packets of random lengths. For a random mix of six crypto algorithms, it achieves a throughput of 17.42 Gbps with an average latency of 3.92 ms for packets of 1280 bytes.

The cost effectiveness of our gateway is much higher than any other existing IPsec solutions that have full programmability. The HSA supported by the APU eliminates the data copy overhead between the CPU cores and the on-chip GPU. As a result, HSA enables such a cheap embedded APU to achieve high performance.

Since PIPSEA spends at most three CPU cores and one on-chip GPU on IPsec processing, adding more CPU cores to implement different pipeline stages, such as packet routing and intrusion detection, enables PIPSEA to become a part of a more powerful network system.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2013R1A3A2003664). ICT at Seoul National University provided research facilities for this study.

7. REFERENCES

[1] APUs-Accerlated Processing Units. Website. http://www.amd.com/en-us/innovations/software-technologies/apu/.

[2] DPDK: Data Plane Development Kit. Website. http://www.dpdk.org.

[3] Heterogeneous System Architecture. Website. http://www.hsafoundation.com.

[4] AMD. OpenCL Optimization Guide. Website. http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/opencl-optimization-guide/.

[5] S. Bradner and J. McQuaid. Benchmarking Methodology for Network Interconnect Devices. RFC 2544 (Informational), Mar. 1999. Updated by RFCs 6201, 6815.

[6] P. Chodowiec, K. Gaj, P. Bellows, and B. Schott. Experimental Testing of the Gigabit IPSec-Compliant Implementations of Rijndael and Triple DES Using SLAAC-1V FPGA Accelerator Board. In *Proceedings of the 4th International Conference on Information Security*, ISC '01, pages 220–234, 2001.

[7] A. Dandalis and V. K. Prasanna. An Adaptive Cryptographic Engine for Internet Protocol Security Architectures. volume 9, pages 333–353, July 2004.

[8] T. T. Dao, J. Kim, S. Seo, B. Egger, and J. Lee. A Performance Model for GPUs with Caches. *Parallel and Distributed Systems, IEEE Transactions on*, 26(7):1800–1813, July 2015.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[10] R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[11] C.-S. Ha, J. H. Lee, D. S. Leem, M.-S. Park, and B.-Y. Choi. ASIC design of IPSec hardware accelerator for network security. In *Proceedings of the 2004 IEEE Asia-Pacific Conference on Advanced System Integrated Circuits*, pages 168–171, Aug 2004.

[12] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-accelerated Software Router. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 195–206, 2010.

[13] Helion Technology Limited. IPsec solutions. Website. http://www.heliontech.com/ipsec.htm.

[14] A. Hoban. Using Intel AES New Instructions and PCLMULQDQ to Significantly Improve IPSec Performance on Linux. *White paper*, 2010.

[15] A. Hodjat, P. Schaumont, and I. Verbauwhede. Architectural Design Features of a Programmable High Throughput AES Coprocessor. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)*

Volume 2 - Volume 2, ITCC '04, pages 498–, 2004.

[16] Intel Corporation. Integrated Cryptographic and Compression Accelerators on Intel Architecture Platforms. 2013. http://intel.ly/1NF6xFq.

[17] 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 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 317–328, 2012.

[18] K. Jang, S. Han, S. Han, S. Moon, and K. Park. SSLShader: Cheap SSL Acceleration with Commodity Processors. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 1–14, 2011.

[19] A. P. Kakarountas, H. Michail, A. Milidonis, C. E. Goutis, and G. Theodoridis. High-Speed FPGA Implementation of Secure Hash Algorithm for IPSec and VPN Applications. *The Journal of Supercomputing*, 37(2):179–195, 2006.

[20] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard), November 1998. Obsoleted by RFC 4301, updated by RFC 3168.

[21] Khronos Group. *OpenCL 2.0 Specification*. Khronos Group, November 2013.

[22] J. Kim, K. Jang, K. Lee, S. Ma, J. Shim, and S. Moon. NBA (Network Balancing Act): A High-performance Packet Processing Framework for Heterogeneous Processors. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 22:1–22:14, 2015.

[23] Y. Li, D. Zhang, A. X. Liu, and J. Zheng. GAMT: A Fast and Scalable IP Lookup Engine for GPU-based Software Routers. In *Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '13, pages 1–12, 2013.

[24] Y. Liu, D. Xu, W. Song, and Z. Mu. Design and Implementation of High Performance IPSec Applications with Multi-Core Processors. In *Proceedings of the 2008 International Seminar on Future Information Technology and Management Engineering*, FITME '08, pages 595–598, Nov 2008.

[25] J. Meng, X. Chen, Z. Chen, C. Lin, B. Mu, and L. Ruan. Towards High-performance IPsec on Cavium OCTEON Platform. In *Proceedings of the Second International Conference on Trusted Systems*, INTRUST'10, pages 37–46, 2011.

[26] S. Mu, X. Zhang, N. Zhang, J. Lu, Y. S. Deng, and S. Zhang. IP Routing Processing with Graphic Processors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 93–98, Leuven, Belgium, 2010. European Design and Automation Association.

[27] NVIDIA. *CUDA C Programming Guide*. NVIDIA, July 2013.

[28] J. Thoguluva, A. Raghunathan, and S. T. Chakradhar. Efficient Software Architecture for IPSec Acceleration Using a Programmable Security Processor. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08, pages 1148–1153, 2008.

[29] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis. Gnort: High Performance Network Intrusion Detection Using Graphics Processors. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, RAID '08, pages 116–134, 2008.

[30] G. Vasiliadis, L. Koromilas, M. Polychronakis, and S. Ioannidis. GASPP: A GPU-accelerated Stateful Packet Processing Framework. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 321–332, 2014.

[31] G. Vasiliadis, M. Polychronakis, and S. Ioannidis. MIDeA: A Multi-parallel Intrusion Detection Architecture. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 297–308, 2011.

[32] E. Z. Zhang, Y. Jiang, Z. Guo, K. Tian, and X. Shen. On-the-fly Elimination of Dynamic Irregularities for GPU Computing. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 369–380, New York, NY, USA, 2011. ACM.

[33] J. Zhao, X. Zhang, X. Wang, and X. Xue. Achieving O(1) IP Lookup on GPU-based Software Routers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 429–430, 2010.