Software-Defined Hardware-Assisted Isolation for Trusted Next-Generation IoT Systems

Filippos-George Kolimbianakis
Hellenic Mediterranean University
Estavromenos, 71410 Iraklio, Crete
Greece
filipposkolibian@yahoo.gr

George Kornaros
Hellenic Mediterranean University
Estavromenos, 71410 Iraklio, Crete
Greece
kornaros@hmu.gr

ABSTRACT

To mitigate[1] cybersecurity threats at the edge of the network in Internet-of Things (IoT) domains, recently, the use of networking technologies such as SDN-NFV has been proposed. Intelligent and dynamic security policy enforcement methodologies become increasingly important to bring more cautious in network communications for IoT services and applications which naturally embed traditional security and privacy risks, such as service hijacking, DDoS attack, denial service, IP spoofing, man-in-the-middle. To extend such frameworks, in this work we present a software-defined protection-oriented hardware technique to support physical isolation of memory compartments and of hardware devices such as DMAs and accelerators inside modern Systems-on-Chip (SoCs), not only at the edge but also at the IoT high-end accelerator-rich devices. In addition to network functions commonly supported in software-defined environments, we describe innovative lightweight software-controlled hardware mechanisms for enhancing IoT ecosystem security by design.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; *Embedded Hardware* • **Security and Privacy** → Security in hardware

KEYWORDS

Software-defined trusted interconnect, hardware isolated memory compartments, lightweight hardware protection, secure NFV, trusted IoT hardware functions

SAC'22, April 25 –April 29, 2022, Brno, Czech Republic

DOI: xx.xxxx/xxx_x

1 INTRODUCTION

Nowadays, the proliferation of trusted execution environments (TEEs) to protect sensitive code and data has resulted in most CPU vendors to roll out their TEEs (e.g., ARM TrustZone, Intel SGX, and AMD SEV) to create a secure execution environment, commonly referred to as an *enclave*. Trusted execution of application is a major concern from IoT environments to cloud infrastructures, from small resource-constrained devices to MPSoCs. Additionally, it is a great challenge to realize Network Functions Virtualization (NFV), i.e., specialized network services such as firewalling, routing and load balancing on standard computing platforms using virtualization. Cloud companies have realized this and have investigated other types of technologies and techniques, resulting in the adoption of containers and DevOps models to create new services faster and more dynamically, as well as in the adoption of FPGA-based reconfigurable computing to gain competitive edge by accelerating network functions[18][30][31]. Chip manufacturers and hardware integrators have brought up programmable platforms based on specialized network processors, which induces fast-paced development of specialized dynamically reconfigurable network applications. Napatech brought FPGA-based NICs. Marvell offers Xelerated processor for fiber access application as their so called Metro Ethernet Application. Xilinx has released the SDNet approach which supports SDN functionality through programmable data plane hardware. An FPGA-based NFV platform can be used as a promising platform where we need both a high degree of flexibility and also a high degree of performance, for instance via supporting network encryption and network compression using the FPGA-based NFV framework. In IoT domain, NFV technologies enable on-demand deployment of virtual in-network security functions, such as virtual firewall, intrusion detection, authentication, channel protection, with a rising trend to provide the defense mechanisms and threat countermeasures requested by security methodologies [25].

1.1 Trusting NFV Computations

Network Function Virtualization (NFV) has been proposed to move into the Cloud to leverage the manageability, performance and cost advantages of cloud computing. However, this runs the risk of exposing private data, as the hosting infrastructure is not under the control of the user of the cloud resources. To tackle these privacy risks, work has been done to augment cloud applications and middleboxes to operate on encrypted data to reduce the amount of sensitive data that the hosting cloud service has access to [12][13]. By operating on encrypted data, a virtualized middlebox run in a cloud service would not have access to the plaintext data that it is processing, meaning that any malicious tenant or service provider would not be able to steal any of this data.

Edge computing and IoT are deeply interrelated not only for improving latencies and data moving costs, but also to tackle denial of services (DoS) in IoT environment, a daunting problem due to its scale, complexity and frequency[14][16]. Moreover, NFV computations are moving to the edge thus promising to improve quality-of-service for complex IoT-driven applications. By being able to perform trusted computing, useful network services such as NAT and QoS can be provided without the application provider exposing sensitive data about the application. To achieve this, SGX has also been incorporated into a larger NFV network by establishing encrypted channels between the SGX enclaves in the edge applications with the organization's internal or cloud hosted NFV network[15].

To enable high-performance software packet processing in NFV, prior works proposed optimization techniques on hardware using accelerators either in look-aside of modern server CPUs or in-line by fully offloading data plane to hardware. Essentially, proposals involve offloading the network functions processing to hardware accelerators such as GPU, FPGA, System-on-Chip and Network Processor implementations [2][17][18]. Recently, for instance, to support higher throughput (40 Gbps) and lower latency (10 μs) over CPU-only implementation in NFV environments, DHL [3] is proposed to allow running multiple concurrent software network functions (NFs) with distinct accelerator functions on the FPGA simultaneously.

1.2 TEEs

Trusted execution environments (TEEs) commonly are based on hardware support[29], while some, via using SGX enclaves store authentication credentials and execute cryptographic operations for network elements. SGX enclaves rely on a trusted computing base (TCB) of code and data loaded at enclave creation time, processor firmware and processor hardware. Program execution within an enclave is transparent to the underlying operating system and other mutually distrusting enclaves on the platform. Enclaves operate in a dedicated memory area called the Enclave Page Cache, a range of DRAM that cannot be accessed by system software or peripherals [1]. In the cloud/server side recent works such as ShieldBox[19] focuses on securing the Click modular architecture on the untrusted hardware with support of AES GCM cipher into ShieldBox and construction of secure packet processing chains. Additionally, in the client side, the Endbox[20] and Keystone[21] propose memory isolation via guarding by trusted hardware features and a programmable layer underneath an untrusted OS for a trusted execution environment. Additionally, TEEs involve memory constraints[43], and moving from an untrusted domain to a trusted domain adds overhead, as evidenced by study conducted at TEE performance characteristics and found TEE-based functionality to be costly to call and execute. The impact in terms of efficiency and energy dictates the need for more lightweight methods for IoT devices.

To tackle secure deployment of accelerator functions in accelerator-rich Systems-on-Chip (SoCs), not only at the edge but also at the IoT high-end devices, e.g., gateways, we present software-defined protection-oriented hardware mechanisms to support physical isolation of memory compartments and of hardware devices such as DMAs and accelerators in terms of their control and data path.

The rest of this paper is organized as follows: we further motivate our architecture by investigating the risks of modern high-end devices and attempts by prior research in Section II. Section III presents our proposed architecture and prototype implementations, in Section IV we present our experimental results, in Section V we discuss and assess our proposal, and we conclude in Section VI.

2 SDN TRUSTED PLATFORMS AND ATTACK MODELS

Attackers increasingly investigate S/W and H/W systems and particularly SDN strengths and weaknesses and try to maximize exploits based on vulnerabilities. For example, zero-day attacks refer to attacks committed based on newly discovered vulnerabilities [4]. From a security perspective, Man-in-the-Middle (MiM) attacks can possibly occur between controller and its switches where a switch can be compromised or controlled by an intruder. SDN architecture tries to approach this problem by dedicating a special connection between the controller and switches in a separate physical and logic subnet from the rest of switch ports. However, that does not eliminate completely the possibility of compromising the communication between the controller and its switches.

Additionally, Trusted Platform Modules (TPMs) have been proposed to provide limited software isolation using

remote attestation to prove that the software and operating system are in a secure state. However, TPMs cannot protect against access from hardware or side channels, such as DMA attacks. User-space processes are prevented from accessing protected memory regions by a memory controller known as Memory Management Unit (MMU). However, drivers that communicate to PCI/PCIe devices can directly access any physical memory address space by using Direct Memory Access (DMA) operations. Since the physical address space includes the whole system memory, a malicious driver could potentially read and write the entire memory of the system.

DMA has been the focus of security researchers for some years, because it allows to access the memory of a system through certain external interfaces while bypassing the operating system and any software-based security restrictions. Afterwards, the memory dump can be analyzed for sensitive data like passwords. However, attackers can also write to the memory pages of the running system and this way modify the system's properties or work-flow on-the-fly. Further, malware that uses DMA to infiltrate an operating system can be detected using techniques such as those presented by Stewin and Bystrov[5].

IOMMU is a memory management unit that maps I/O bus addresses to physical memory addresses for all DMA memory transactions on the bus[41][42]. The role of the IOMMU is similar to that of a traditional MMU: i) it translates the memory I/O address range of one device to the corresponding real physical address, and ii) it prevents any unauthorized access from one device address space to another. Even though the IOMMU establishes memory barriers among different device address spaces, which parts of the memory should be assigned to which devices remains an open research problem.

However, recent studies [6] reveal that devices interact deeply with the device driver, and with other parts of the operating system. In addition to user processes which use the system-call interface, device firmware which communicates over the IOMMU kernel shared-memory interface can stimulate complex vulnerable behavior. Layers involving communication stacks and memory allocators rarely are hardened against malicious devices. Various techniques try to limit the effectiveness of attackers in injecting kernel pointers, but leaves open a number of data fields, including data pointers, that could leave the system exposed to further vulnerabilities.

Moreover, allocating DMA buffers at sub-page (byte-level) granularity for NFs or other devices creates OS vulnerabilities since IOMMU protection works at page-level granularity[7]. Additionally, operating systems (OSes) commonly implement deferred protection for performance reasons. This means that the OS performs IOTLB invalidations asynchronously instead of invalidating the IOTLB on each *dma unmap* system call. The impact is the creation of a time window (of vulnerability) where a DMA buffer remains accessible to the device firmware after a *dma unmap* call of the buffer returns. For example, after an incoming packet successfully completes firewall inspection, a malicious NIC can modify the packet into a malicious one[8]. Likewise, an unmapped buffer may get reused by the OS, exposing the device to arbitrary sensitive data.

While one attack approach involves circumventing all the hardware protection mechanisms currently available, other options involve attempting to disable them. For instance, an attacker can modify hardware-level data structures (e.g., Interrupt Descriptor Table) or configuration tables (e.g., DMA remapping table) to turn off IOMMU. Another approach aims at making the illusion of a non-parsable DMA remapping table. To achieve this, an attacker has to set a zero length for such a table during boot time. Another class of attacks aim at modifying the metadata exposed by I/O controllers to mislead the IOMMU. One such an attack is described by Sang et al. [22], where, the attackers could map two I/O devices into the same physical memory range by impersonating a PCI Express device with a legacy PCI device.

The software side of peripheral DMA interfaces is not implemented by carefully hardened kernel system-call code, tested by decades of malicious attacks and fuzzing, but by thousands of device drivers that have been designed around historic mutual trust, hardware convenience, and performance maximization. Unlike most system-call interfaces, many key data structures shared between the kernel and peripherals are via shared memory, e.g., descriptor rings, rather than register passing and selected copy avoidance. Prior work has suggested that shared-memory interfaces are particularly vulnerable to race conditions and other unsafe interactions. For instance, attacking the ring-buffer code to modify mappings of a custom IOMMU in Apple iOS has been recently reported [9]. The SoC includes a rather large TLB and therefore, changes in IO-Space may not immediately be reflected until the entries are evicted from the cache. Additionally, even if the device provides “lockdown” regions, to which subsequent modifications are prohibited, by mapping in physical addresses corresponding to the aforementioned hardware registers, we can proceed to read their contents directly from IO-Space. This, in turn, reveals the physical ranges encoded in the “lockdown registers”.

Extensive research has been done to address thread and memory isolation in bare-metal real-time systems [23][24]. The proposed compartmentalization is the countermeasure to the attacker model involves exploiting memory corruption vulnerabilities that leads to compromising one of the software modules and thus taking over the whole system,

including both the software stack and its associated physical and networking components. However, MINION [23] and MBED uVisor [24] are daunting tasks for developers restricting the design of compartmentalization of code, data and peripherals.

Our attack model does not address timing attacks and memory physical attacks e.g., attacks on DRAM cannot be prevented because there is no Memory Encryption Engine to encrypt code or data before being transferred in and out to memory. We also ignore the threat of implementation bugs in the secure hardware application, and side-channels on the FPGA accelerators that may unintentionally compromise the security of the system. We assume that hardware primitives, functions, or accelerators are secure.

3 DESIGN OF SOFTWARE-DEFINED ON-CHIP INTERCONNECT

Contrary to recent research works focusing in protection-unaware performance extensions and in automating of software-defined frameworks for NFV nodes [3] and in accelerating of crypto-functions (via ISA extensions in HPC processors or via hardware cores in lightweight IoT devices), we propose several mechanisms in this section, to enable stronger security and trusted deployment of network functions while using hardware accelerators.

3.1 Dynamic Memory Mapping through using Isolated Programming

The proposed technique provides data isolation via combining software programmable data plane partitioning and access rights thus supporting multiple software NFs to use accelerator modules simultaneously without interfering each other. The Software-Defined Interconnect (SDI) block controls and sets specific boundaries for the memory region i.e., the scratchpad memory compartment, which memory region, a master core such as an accelerator can access. As Figure 1 shows, the SDI can also be configured to access any memory region and even directly the system memory, depending on the system model supported. This software-defined interconnect block enables dynamic memory mapping through a configuration port which can be programmed at runtime. The configuration is register-based and allows 16 discrete regions of minimum size of 1024 32-bit words (or 4096 bytes); this size is defined at design time and can be tailored to system needs accordingly.

The functionality of the SDI block is invasive in terms of supervising and controlling incoming addresses. An accelerator module is free to access any address space (commonly contiguous), but the generated memory addresses are updated to conform to the valid address space, as assigned by the control firmware running in the SWCtrl block (see Figure 1). The SDI block forwards an incoming transaction to the physical memory with a new address that uses only the lowest N+12 bits of the original incoming address, where the width of n is configurable. The high order bits (32-N-12) are filled with the contents of the *allocation* register inside SDI, which is prior configured by the SWCtrl firmware. This *allocation* register can be either hardwired or configured by the control firmware.

Since the SDI block ensures that accelerators (faulty or malicious intellectual property (IP) cores) cannot access an illegal address space, i.e., a memory region not entitled to access, now it is important to protect the Central DMA block (CDMA) from potential attacks. To achieve this goal, a customized DMA is designed to replace the Xilinx's provided DMA block. The following prototype presents the block interface, which was synthesized using Xilinx HLS tools. The interface, after the common *source*, *destination* and *length* fields involves specific fields to allow the SWCtrl and the CPU to define four memory regions, thus strictly limiting data accesses. The SWCtrl uses a 32-bit magic number as the identifier (ID) field to specify the limit parameters, while the least significant eight-bit ID identifies the restrictions for the DMA issuer.

```
int CDMA(volatile int *memory,
  unsigned int source,
  unsigned int destination,
  unsigned int length,
  unsigned int base_address_source,
  unsigned int base_address_destination,
  unsigned int id,
  unsigned int source_limit_begin[4],
  unsigned int source_limit_end[4],
  unsigned int destination_limit_begin[4],
  unsigned int destination_limit_end[4])
```

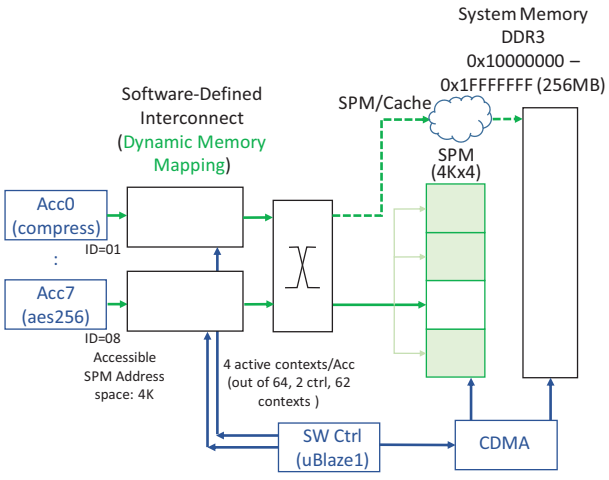

Figure 1. Organization of hardware functions accessing the appropriate SPM partition assigned as permitted by software-assisted dynamic memory mappers. Each SDI block is coupled with an Accelerator block, and allows assignment of four different contexts, i.e., dynamic assignment of one out of four memory compartments for the lifetime as the firmware running in MicroBlaze (uBlaze1) defines.

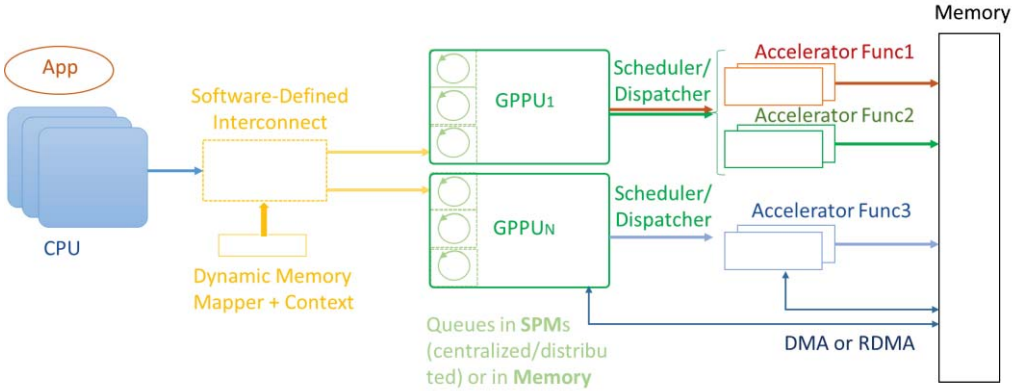

Figure 2. Packet-based dispatching of functions through a general packet-based interface unit (GPPU blocks) to act as a proxy, essentially dedicated to perform queue management and job dispatching to single or multiple hierarchical structured accelerators

The key idea is that a CPU is assigned a range of 64 IDs which determine the origin of the DMA operation commanded which must respect the pre-defined memory region boundaries. This condition holds for four regions of IDs, thus supporting four distinct CPUs. The CPU ID, in conjunction with the AXI on-chip protocol ID of the CPU issuer, are validated to subvert an adversary. Finally, a CPU can use number 255 as a special ID for debugging purposes, i.e., reading the DMA transfer process status.

3.2 Protecting Dispatching to Hardware Accelerators

With the potential growth in real-time and embedded systems in recent years, hardware schedulers have been proposed to circumvent the overhead incurred by dynamic scheduling algorithms (e.g., Earliest Deadline First (EDF)) and to distribute tasks on multiple processing cores, regardless of the internal architecture of the cores [32]. Run-time scheduling is especially of rising importance in view of massively heterogeneous systems with a sea of accelerators[33] [34]. The scheduler commonly delivers the data to the accelerators via an interconnect scheme, which should contain the entire description on the task to be performed, while DMA engines transfer the accelerator data from and to the shared system memory[33].

However, in these efforts and in other recent works [10][11][34], dispatching of jobs to accelerator units is tackled largely with the objective to improve processor performance efficiency and to help the OS for higher resource utilization, while neglecting or relying on software component (e.g., driver) to ensure protection. To enhance the security of deep neural network accelerators architecture against side-channel information leakage, dedicated encryption units for instruction and data path are proposed[36]; these solutions however increase the system complexity. Such complex systems, which also use packet-based communication as we do in our accelerator structure next, propose the simultaneous execution of original (target) DNN networks and the detect algorithm or network to detect adversary sample attacks[37]. Similarly, to prevent injecting malicious data to execute unauthorized code (e.g., buffer overflow), Dynamic Information Flow Tracking (DIFT) techniques attempt, even by using hardware co-processors[38][39], to mark insecure data and track their use during the execution.

Ultimately, job dispatching can be achieved through both hardware and software components which minimize operating system's overheads and additionally, isolate dispatching process from user perturbations since a clear and reduced–base (size) interface is provided to the user. At the same time, buffer management (allocation and deallocation) is done via pre-defined buffers. DMA operations for commands and data are not controlled with a system-IOMMU, but via using a job manager-assistant, namely a general packet processing unit (GPPU) hardware controller. The GPPU controls one or even multiple different accelerators and, by abstracting the diversity of the functionality of these custom accelerators, it provides a simple API to the CPU and hides the accelerators details. This GPPU controller is extended and, collaboratively with the CPU's MMU, utilize pinned buffers which are supplied to the accelerator functions and at the same time the buffers are exposed to the virtual address space of the application.

As Figure 2 shows, a single SDI block is integrated in an architecture organization inspired by recent work [10], in the control path between the CPU and the GPPU dispatcher. Similar to Figure 1 organization, a firmware driven dynamic memory controller executes in a MicroBlaze soft-CPU and manages the SDI block. The memory address range assigned to each Dispatcher (at design time) is now isolated and even partitioned to discrete compartments. The firmware, active at Microblaze boot time, polices all incoming addresses from CPU_0 to comply to GPPU0 address space and all incoming addresses from CPU_1 to GPPU1 address space. A compromised kernel or user-level driver and application is

unable to access a job dispatcher allocated to another application on the host CPU, unless all applications and drivers inside the OS are under the control of the attacker.

4 RESULTS

An organization based on Figure 1 method to implement a software-defined interconnect on a ZYNQ7020 platform has an impact of two clock cycles per transaction due to re-generation of the target address. Even when realizing an AES128 accelerator function for encrypting network packet payloads under the SDI control and supervision, as Table I shows, the overhead of SDI is negligible since the latency for such a streaming processing is minimal compared to the delay of the accelerator.

TABLE I
LATENCY MEASUREMENTS OF IMPLEMENTING OF AES HARDWARE ACCELERATORS WITH SDI ON A ZYNQ7020 PLATFORM

Function	Description	Delay (in clock cycles @ 100MHz)
DMA xfer	4 KB transfer (DDR3→SPM)	3040
AES Enc	48 bytes	2775
AES Dec	48 bytes	3207

The integration of the SDI design, as indicated in Figure 2 for streaming-based accelerators, imposes almost zero-delay impact, based on the acceleration technique proposed in [10], because it is incorporated in the control channel. As shown in Figure 4, the data accelerators path is modified as well, to enable controlled isolation via an SDI data proxy. The data path basically dominates the acceleration processing in terms of time and energy consumption. Both the user application and the packet dispatching software manager (i.e., GPPU) are completely independent of the SDI Manager. The SDI Manager is responsible for configuring and physically compartmentalizing the memory regions for the queues and data partitions devoted to each accelerator. Although the GPPU hardware dispatches job requests and validates requests to access both accelerator and memory through queue allocation and usage control for the lifetime of the acceleration process, it is still susceptible to software attacks injecting invalid memory pointers for the data target. The SDI Data interconnect though physically defines a dedicated memory data region inside the off-chip DRAM (App Data Structures in Figure 4). As a result of the inaccessible memory address, the specific accelerator will return an error condition.

After implementing the design shown in Figure 2 with a Sobel and a matrix multiplication accelerator assigned to isolated packet dispatchers, we collected nearly same results as in [10] in terms of performance (with an average of 0.2 percent deviation), as shown in Figure 3, over the base version when scaling SOBEL filtering kernels and matrix multiplication kernels on the accelerators. Notice that as in the implementation in [10], we used minimal size jobs with the single and main objective to evaluate the dispatching throughput and not the actual accelerator throughput, since the latter mostly depends on the accelerator realization itself and the sustained memory bandwidth.

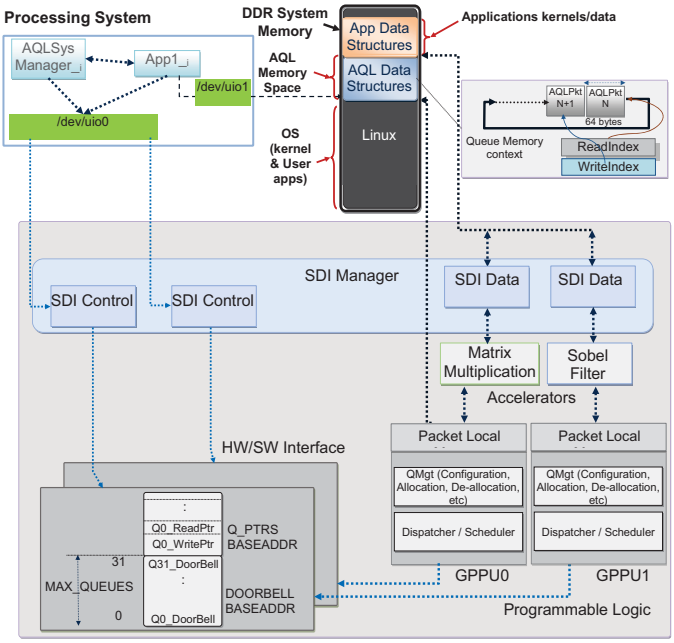

Figure 3. Securing memory compartments via using software-defined proxies for both control and data paths in packet- (i.e., GPPU-) based embedded accelerators.

The different acceleration schemes, are free from an internal address translation scheme similar to an IOMMU, since the data are located in a reserved memory partition, shared by the accelerator and the OS, through using mmap() operation at initialization phase. Importantly, the SDI units, which are pre-programmed at design time, prevent defective or intentional unauthorized access to regions other than those designated.

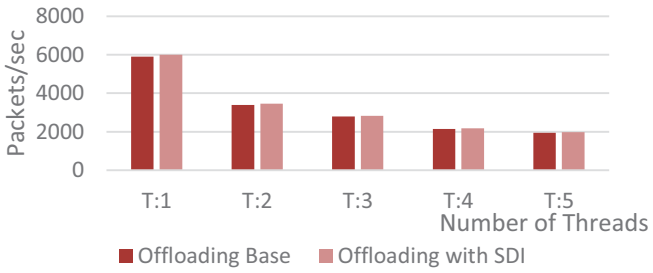

Figure 4. Performance gain of acceleration methods compared to base acceleration for scaling number of threads (T:1-5) when offloading kernels on Zedboard platform.

5 DISCUSSION

Even though recently proposed methods enhance channel protection management by dynamically provisioning the

necessary crypto-keys for NFV/SDN-enabled IoT M2M communications [27], security functions towards the edge of the network and inside modern complex IoT SoCs need careful design methods as well. To enable a robust architecture addressing the threat model and supporting all SDN planes presented in recent SDN models [35], counter-measures are required similar to above proposed techniques. Our techniques are inspired by the essential principle that to guarantee security, pre-made decision about the boundary between what is trusted and untrusted are required by design [21]. In conjunction, the hardware extensions controlling this boundary need a verified trusted firmware layer for customization on top.

Our system also covers assaults that can indirectly access off-chip and on-chip memory by putting malicious components in the device, similar to the threat model outlined to offer efficient extended protection for TEEs[43]: DMA assaults and cold boot attacks. By resetting the device and running a malicious memory-dumping kernel that may recover memory contents, cold boot attacks can get access to memory. Attackers can acquire physical memory access by attaching DMA-capable peripherals to devices without ever needing to exploit software flaws. At a high silicon cost, an IOMMU-enabled system can prevent such malfunctions or attacks. Our suggested method eliminates such costs either in hardware (i.e., IOMMUs) or software (e.g., reduced kernel design[43]), resulting in a viable energy-saving solution for IoT devices.

Similarly to Keystone attack model [21], even a compromised enclave, application and OS cannot bypass the isolation boundaries that our software-defined interconnect mandates. The firmware managing the SDI block and CDMA do not share any state with the host OS or the user application and hence are not exposed to controlled channel attacks. Instead of controlling the entity that manages the page tables, we control the path to transfer data and to process these data by a network function accelerator or by a processing entity in general. Even though we share the same initial principle as in Sanctum [28], to provide hardware-based primitives, we propose a more lightweight solution, very straightforward, with much lower complexity for the SoC hardware and firmware support.

6 CONCLUSIONS

In this paper we presented hardware-based techniques to secure hardware accelerator functions through utilizing a software-defined interconnect manager which supervises accelerator accesses to memory compartments guaranteeing isolation. Our evaluation results show that it only adds an insignificant overhead when processing accelerator data packets or control flow. The proposed technique either eliminates classes of attacks (physical and software attacks) and allows integration with existing techniques, while proving to be fairly simple and straightforward in terms of implementation and integration complexity.

For future work, we intend to control per process bindings of accelerator functions and investigate the concept of self-provisioning and secure updating process which allows for management policies whether an update is legal or not.

Further, we intend to combine DMA pointers checking with lightweight DIFT techniques[40] with applications in detecting control-flow exploits and malicious data leakage, which are primary security threats.

ACKNOWLEDGMENTS

The research leading to these results received funding from the European Union (EU) project AVANGARD (No 869986). The authors would also like to thank the reviewers for their valuable comments.

REFERENCES

[1] McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C.V., Shafi, H., Shanbhogue, V., Savagaonkar, U.R.: Innovative Instructions and Software Model for Isolated Execution. In: Proc. of 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. pp. 10:1–10:1. HASP '13, Jun. 2013.

[2] E. Kaljic, A. Maric, P. Njemcevic, and M. Hadzalic, "A Survey on Data Plane Flexibility and Programmability in Software-Defined Networking", IEEE Access, Apr. 2019, [Online]. Available: doi: 0.1109/ACCESS.2019.2910140

[3] X. Li, X. Wang, F. Liu and H. Xu, "DHL: Enabling Flexible Software Network Functions with FPGA Acceleration," 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, 2018, pp. 1-11.

[4] K. Rowan, V. Kotronis and P. Smith, "OpenFlow: a security analysis", In Proceedings of the 8th Workshop on Secure Network Protocols (NPSec), ICNP, Oct. 2013.

[5] P. Stewin and I. Bystrov, "Understanding DMA Malware", In Proceedings of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, vol. 7591, pp. 21-41, Jul. 2013.

[6] A. T. Markettos, C. Rothwell, B. F. Gutstein, A. Pearce, P. G. Neumann, S. W. Moore, R. N. M. Watson, "Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals", In Proceedings of the Network and Distributed Systems Security Symposium (NDSS), Feb. 2019.

[7] A. Markuze, A. Morrison, and D. Tsafrir, "True IOMMU Protection from DMA Attacks: When Copy is Faster than Zero Copy", In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16), pp. 249-262.

[8] S. Boyd-Wickizer and N. Zeldovich, "Tolerating Malicious Device Drivers in Linux", In USENIX Annual Technical Conference (ATC), pp. 117–130, 2010.

[9] G. Beniamini, "Over the air – vol. 2, pt. 3: Exploiting the Wi-Fi stack on Apple devices," Oct. 2017. [Online]. Available: https://googleprojectzero:blogspot:co:uk/2017/10/over-air-vol-2-pt-3-exploiting-wi-fi:html

[10] G. Kornaros and M. Coppola, "Enabling Efficient Job Dispatching in Accelerator-Extended Heterogeneous Systems with Unified Address Space," In 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2018, pp. 180-188, doi: 10.1109/CAHPC.2018.8645945.

[11] O. Tomoutzoglou, D. Mbakoyiannis, G. Kornaros and M. Coppola, "Efficient Job Offloading in Heterogeneous Systems through Hardware-assisted Packet-based Dispatching and User-level Runtime

Infrastructure," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol 39, no. 5, pp. 1017-1030, May 2020, doi: 10.1109/TCAD.2019.2907912

[12] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: securely outsourcing middleboxes to the cloud", In 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), pages 255{273, 2016.

[13] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic", In ACM SIGCOMM Computer Communication Review, volume 45, pages 213-226. ACM, 2015.

[14] K. Bhardwaj, M.-W. Shih, P. Agarwal, A. Gavrilovska, T. Kim, and K. Schwan, "Fast, scalable and secure onloading of edge functions using airbox". In Edge Computing (SEC), IEEE/ACM Symposium on, pages 14-27. IEEE, 2016.

[15] K. Bhardwaj, J. C. Miranda, and A. Gavrilovska, "Towards IoT-DDoS prevention using edge computing," in Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge 18), 2018.

[16] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, ''Edge computing for autonomous driving: Opportunities and challenges,'' Proc. IEEE, vol. 107, no. 8, pp. 1697–1716, Aug. 2019, doi: 10.1109/jproc.2019. 2915983

[17] Y. Go, M. A. Jamshed, Y.G. Moon, C. Hwang, and K.S. Park, "APUNet: Revitalizing GPU as Packet Processing Accelerator", In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17), 2017.

[18] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware", In Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM'16), 2016.

[19] B. Trach et al., "ShieldBox: Secure Middleboxes using Shielded Execution," in ACM SOSR'18, 2018.

[20] Goltzsche et al., "ENDBOX: Scalable Middlebox Functions Using Client-Side Trusted Execution", in 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2018.

[21] D. Lee et al., "Keystone: An Open Framework for Architecting Trusted Execution Environments", In 15th European Conference on Computer Systems (EuroSys '20), 2020, https://doi.org/10.1145/3342195.3387532

[22] F. L. Sang, É. Lacombe, V. Nicomette and Y. Deswarte, "Exploiting an I/OMMU vulnerability," In 5th International Conference on Malicious and Unwanted Software, Nancy, Lorraine, 2010, pp. 7-14, doi: 10.1109/MALWARE.2010.5665798

[23] C. H. Kim, T. Kim, H. Choi, Z. Gu, B. Lee, X. Zhang, and D. Xu, "Securing real-time microcontroller systems through customized memory view switching," in Network and Distributed Systems Security Symp. (NDSS), 2018.

[24] The mbed OS uVisor. https://www.mbed.com/en/technologies/security/uvisor/

[25] S. N. M. García, A. M. Zarca, J. L. Hernández-Ramos, J. B. Bernabé, and A. S. Gómez, ''Enforcing behavioral profiles through software-defined networks in the industrial Internet of Things,'' Appl. Sci., vol. 9, no. 21, p. 4576, Oct. 2019.

[26] S. Ziegler, A. Skarmeta, J. Bernal, E. Kim, and S. Bianchi, "Anastacia: Advanced networked agents for security and trust assessment in CPS IoT architectures," in 2017 Global Internet of Things Summit (GIoTS), Jun. 2017, pp. 1–6.

[27] I. Farris, J. Bernabe, N. Toumi, D. Garcia-Carrillo, T. Taleb, A. Skarmeta, and B. Sahlin., "Towards Provisioning of SDN/NFV-based Security Enablers for Integrated Protection of IoT Systems," in IEEE Conference on Standards for Communications and Networking (CSCN-2017), 2017.

[28] V. Costan, I. Lebedev, S. Devadas, "Sanctum: Minimal Hardware Extensions for Strong Software Isolation", in Proc of 25th USENIX Security Symposium, pp 857-874, 2016.

[29] P. Maene, J. Gotzfried, R. d. Clercq, T. Muller, F. Freiling, and I. Verbauwhede, "Hardware-Based Trusted Computing Architectures for Isolation and Attestation", IEEE Transaction on Computers, vol. 67, no.3, Mar 2018, pp. 361-374.

[30] P. Shantharama, A. S. Thyagaturu and M. Reisslein, "Hardware-Accelerated Platforms and Infrastructures for Network Functions: A Survey of Enabling Technologies and Research Studies," in IEEE Access, vol. 8, pp. 132021-132085, 2020, doi: 10.1109/ACCESS.2020.3008250

[31] G. S. Niemiec, L. M. S. Batista, A. E. Schaeffer-Filho and G. L. Nazar, "A Survey on FPGA Support for the Feasible Execution of Virtualized Network Functions," in IEEE Communications Surveys & Tutorials, vol. 22, no. 1, pp. 504-525, 2020, doi: 10.1109/COMST.2019.2943690

[32] D. Derafshi , A. Norollah , M. Khosroanjam, and H. Beitollahi, "HRHS: A High-Performance Real-Time Hardware Scheduler", in IEEE Transactions on Parallel and distributed Systems, vol. 31, no/ 4, Apr 2020, pp. 897-908, doi: 10.1109/TPDS.2019.2952136

[33] K. Hegde, A. Srivastava, and R. Agrawal, "HTS: A Hardware Task Scheduler for Heterogeneous Systems", online: https://arxiv.org/abs/1907.00271, 2019.

[34] S. M. Lee, E. N. Ri Ko and S. E. Lee, "A Hardware Scheduler for Multicore Block Cipher Processor", 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), 2016, pp. 750-754, doi: 10.1109/PDP.2016.59

[35] Open Networking Foundation, "Threat Analysis for the SDN Architecture", ver 1.0, Jul 2016, TR-530, online: https://opennetworking.org/wp-content/uploads/2014/10/Threat_Analysis_for_the_SDN_Architecture.pdf.

[36] X. Wang, R. Hou, Y. Zhu, J. Zhang, and D. Meng, "NPUFort: a secure architecture of DNN accelerator against model inversion attack", In Proceedings of the 16th ACM International Conference on Computing Frontiers (CF '19), pp. 190–196, doi:10.1145/3310273.3323070

[37] X. Wang, R. Hou, B. Zhao, F. Yuan, J. Zhang, D. Meng, and X. Qian, "DNNGuard: An Elastic Heterogeneous DNN Accelerator Architecture against Adversarial Attacks", In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20), pp. 19–34, DOI:https://doi.org/10.1145/3373376.3378532

[38] H. Kannan, M. Dalton, and C. Kozyrakis, "Decoupling dynamic information flow tracking with a dedicated coprocessor," in Proceedings of DSN, Jun. 2009, pp. 105–114.

[39] L. Piccolboni, G. Di Guglielmo and L. P. Carloni, "PAGURUS: Low-Overhead Dynamic Information Flow Tracking on Loosely Coupled Accelerators," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 11, pp. 2685-2696, Nov. 2018, doi: 10.1109/TCAD.2018.2857321

[40] D. Townley, K. N. Khasawneh, D. Ponomarev, N. Abu-Ghazaleh, and L. Yu, "LATCH: A Locality-Aware Taint Checker", In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52), 2019, pp. 969–982, doi:10.1145/3352460.3358327

[41] G. Kornaros, K. Harteros, I. Christoforakis and M. Astrinaki, "I/O virtualization utilizing an efficient hardware system-level Memory Management Unit," 2014 International Symposium on System-on-Chip (SoC), 2014, pp. 1-4, doi: 10.1109/ISSOC.2014.6972448

[42] G. Kornaros, M. D. Grammatikakis and M. Coppola, "Towards Full Virtualization of Heterogeneous NoC-based Multicore Embedded Architectures," 2012 IEEE 15th International Conference on Computational Science and Engineering, 2012, pp. 345-352, doi: 10.1109/ICCSE.2012.55

[43] S. Zhao, Q. Zhang, Y. Qin, W. Feng, and D. Feng, "Minimal kernel: An operating system architecture for TEE to resist board level physical attacks," in 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019). Chaoyang District, Beijing: USENIX Association, Sep. 2019, pp. 105–120.