# TEE Time at P4—Performance Analysis of Trusted Execution Environments for Packet Processing

Manuel Simon, Sebastian Warter, Sebastian Gallenmüller, and Georg Carle
Chair of Network Architectures and Services, Technical University of Munich, Germany
{simonm|gallenmu|carle}@net.in.tum.de, sebastian.warter@tum.de

***Abstract*—Modern computer networks, such as 5G/6G networks, require high-performance, low-latency, and secure packet processing while ensuring data confidentiality in cloud environments. Trusted Execution Environments (TEEs) address these security requirements and provide encrypted memory areas that protect sensitive data from untrusted cloud providers. This paper presents a performance analysis of TEE technologies, specifically Intel SGX and AMD SEV-SNP, in the context of software-based user-space packet processing with DPDK and the P4 language. We evaluate two architectural approaches: (1) integrating TEEs as external processing modules implemented with SGX and (2) executing the entire P4 pipeline inside a TEE using AMD-SEV. Our analysis examines computational and I/O overhead across different CPU architectures. The results show the trade-offs between TEE designs, implementations, and performance, demonstrating that AMD SEV-SNP offers better scalability with lower performance penalties compared to Intel SGX.**

***Index Terms*—TEE, SGX, SEV-SNP, P4, Packet Processing**

## I. Introduction

Modern computer networks, i.e., 5G/6G, aim for high-performance, low-latency, secure, and highly customizable end-to-end connections. This trend shifts functionality into the network using cloud-based network functions (NFs). However, moving functionality and data to third parties requires trust to guarantee the desired execution and protect sensitive data. For instance, monitoring and intrusion detection may involve the analysis of IP addresses of known entities or even include analyzing the payload. Sensitive user information must be protected from the third party. Moreover, administrators want to ensure that program code and functionality are not modified (maliciously) by the cloud provider. These problems are tackled by Trusted Execution Environments (TEE), offering encrypted memory in which the keys are only accessible by the underlying hardware itself. Therefore, CPUs offering TEE abstract the underlying secure execution from users. This way, the operator only has to trust CPU manufacturers but not cloud providers. Different implementations of TEEs exist; the most prominent include Intel's Software Guard Extensions (SGX) and AMD's Secure Encrypted Virtualization with Secure Nested Paging (SEV-SNP).

We analyze the performance of the different approaches and compare the use cases w.r.t. software packet processing. For that, we investigate T4P4S [1], a P4 software switch based on DPDK. P4 [2] is a programming language for data planes of software-defined networks. It brings the advantage of a high-level, domain-specific language to build high-performance NFs in a target-independent way. T4P4S translates the P4 programs to DPDK code, allowing the execution on commodity, general purpose hardware. Its implementation in software makes T4P4S a suitable choice for execution in the cloud. We will investigate two different modes: P4 provides the option of using external, non-P4 functionality, which we can use to define the API between common packet processing and the execution of sensitive parts inside the TEE. Alternatively, the whole packet processing pipeline may lay inside the TEE.

Our contributions are: Implementation of a P4 user-space software pipeline inside/next to a TEE, comparison of different TEE designs and implementations, detailed performance analysis of TEEs and implementations for user-space packet processing, and a performance model for I/O overhead.

## II. Background

*a) P4 [2]:* is a programming language for data planes. P4 supports hardware and software targets. So-called "*externs*" add non-P4, target-specific functionality. P4 offers a programmable pipeline to introduce new protocols. Our study utilizes the open-source P4 software target T4P4S [1] to investigate packet processing with and without TEEs. T4P4S is based on DPDK and, therefore, offers high performance.

*b) DPDK:* is a framework for high-performance packet processing. Its performance relies on: (1) packet reception via polling of batches, avoiding costly interrupts; and (2) running entirely in user space to bypass the kernel network stack. Direct Memory Access (DMA) for packet I/O causes issues when used with trusted execution (cf. Sec. II-0e). DPDK offers drivers that bind the NIC to the kernel while copying every packet to user space, i.e., XDP sockets (cf. Sec. II-0f); the copy operations increase processing costs. In return, DPDK programs can still run without required modifications if the user space drivers cannot be used.

*c) TEEs:* guarantee, according to Sabt et al. [3], the "authenticity of the executed code, the integrity of runtime states [...], and the confidentiality of its code, data and runtime states [...]." Customers running code in a TEE need only trust the executed code and CPU manufacturer—not the hardware operator or hypervisor. To ensure the defined properties, TEEs encrypt the memory, using protected secrets on the CPU. Examples of TEEs include Intel SGX and AMD SEV. For packet processing, different use cases are possible: calculations on encrypted traffic, as secure network gateways, or as a privacy-preserving monitor of (encrypted) traffic.

Table I: Comparison of TEE implementations

| | Intel SGX | AMD SEV-SNP |
|---|---|---|
| Type | User space | VM |
| Mem. encryption/integrity | ✓ / ✓ | ✓ / ✓ |
| Overhead | Context switches | swiotlb |
| Architecture | split (secure enclave) | all in secure VM |
| Requires refactoring | ✓ | ✗ |

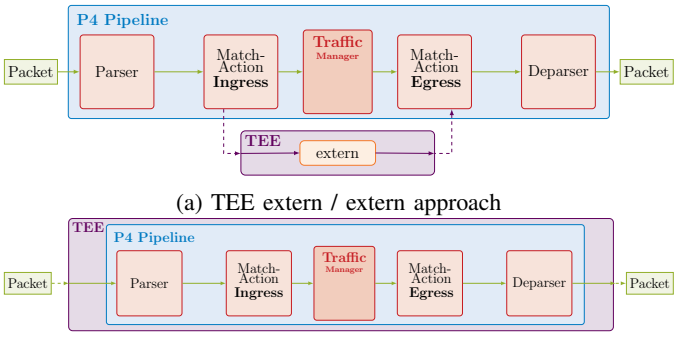
*d) Intel SGX:* implements a TEE by extending the Instruction Set Architecture. SGX allows the creation of so-called enclaves that are separated from the regular user space. Memory in the enclave is encrypted. Running programs can verify their integrity via a secure hash using a web service. SGX programs must be modified, requiring a split into untrusted and trusted parts, running in regular user space or the enclave. Since memory in the enclave is limited, only the parts of the software that require trust will run there. We did not investigate Intel TDX, a more recent TEE implementation, as we did not have access to a machine with TDX support.

*e) AMD SEV:* isolates the hypervisor from guest VMs and encrypts their memory. The AMD Secure Processor manages the access to the involved keys. Neither the hypervisor nor other VMs can access the data of the TEE VM. Hypervisor and guest kernel must be adapted to handle encrypted pages. To facilitate data sharing, memory pages can be marked as unencrypted. To transfer packets into trusted VMs, "bounce buffers" are used, which transfer all data from a temporary, unencrypted memory area—used for DMA operations by the NIC—to the encrypted memory area of the trusted VM. This transparent process allows the use of existing kernel drivers inside the VM without modifications. However, it introduces additional latency as all data must be copied. AMD proposed SEV Trusted-I/O (SEV-TIO) [4], which allows direct DMA operations on private and trusted memory pages, eliminating the detour through bounce buffers. This feature increases performance and mitigates attacks against the memory encryption [5]. Table I lists the features of the two investigated TEEs.

*f) XDP Framework:* Shared memory pages cannot be created inside an AMD SEV VM from user space. We can use the eXpress Data Path (XDP) [6] kernel hook and `AF_XDP` sockets as workaround. The hook is called early in the Linux network stack, and inside, extended Berkeley Packet Filters (eBPF) can control packet (pre-)processing. The eBPF VM is a register machine that runs verified code, and the NF can redirect packets to an `AF_XDP` socket for further processing outside eBPF. This way, packets from kernel space, where the shared pages are located, can be transferred to a user space application, i.e., DPDK. In copy mode, packets are duplicated and made accessible by DPDK early in the networking stack, thus avoiding costly kernel execution. Without SEV-TIO, in total, two copies (bounce buffers, XDP copy) are required to use DPDK inside the TEE. [3]

## III. Related Work

*LightBox* [7] is an SGX-enabled implementation for secure middleboxes. It offers flow state management and secures packet payloads and metadata. LightBox uses a complex setup




(a) TEE extern / extern approach

(b) P4 pipeline in TEE / secure P4 pipeline

Figure 1: TEE positions for a P4 pipeline; packet path in *green*, data path in *purple*; dashed paths may involve copies

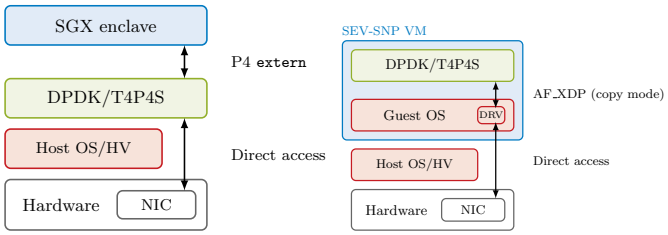
with custom virtual network interfaces, whereas our solution uses standard technologies. *OFTinSGX* [8] runs the OpenFlow rules of Open vSwitch inside an SGX enclave. Therefore, it isolates and secures the existing tables and rules. However, it cannot handle encrypted data flows. Our P4-based solution is more flexible, allowing arbitrary protocols and advanced processing. *ShieldBox* [9] leverages SGX enclaves to create secure containers. It uses Click [10] and SCONE to run NFs in the enclave. *SafeBricks* [11] similarly enables secure NF execution inside SGX enclaves. It uses DPDK for I/O and shared buffers to communicate with the NF in the enclave without the need for an additional copy. It splits the DPDK processing part from the trusted function. SafeBricks uses NetBricks [12] to program NFs; our solution relies on the target-independent P4. *rkt-io* [13] is a framework to run applications inside Intel SGX having a direct userspace network I/O stack within the TEE. They provide a POSIX-compatible `socket()` API. The modified DPDK version of rkt-io runs inside the SGX enclave to provide network access. We do not want to rely on highly customized and specialized solutions but investigate common, easy-to-adapt DPDK processing approaches. Li et al. [14] proposed a kernel module allowing hardware access to DPDK from inside an SEV VM. Their implementation partly relies on bounce buffers. Our approach uses existing kernel technologies instead.

We create and investigate DPDK-based applications inside TEEs, i.e., in secure pipeline mode, without adapting the application. Avoiding custom solutions sacrifices performance. However, the investigated application can be used inside TEEs or outside, or potentially in a more performant way using the upcoming SEV-TIO, without significant modifications. We rely on the established and multi-target P4 language instead of specialized frameworks to program the NFs. The results of our study show the bottlenecks when using standard solutions.

## IV. Design

### A. TEE extern next to the P4 pipeline

In the *extern approach*, the standard, fast packet processing is defined in P4, where trust is unnecessary. Inside the P4 pipeline, a well-defined API can be used to call the "extern", hard-coded functions of the TEE (cf. Fig. 1a). The TEE holds




(a) SGX enclave in P4 extern (b) P4 pipeline in SEV-SNP

Figure 2: TEE implementations for a P4 software Pipeline

secrets and can selectively use (encrypted) header fields or payload for processing.

*Use cases:* include trusted computation on secret data, while normal packet processing, including routing, is not part of the trusted area. As it follows the typical application split of SGX applications, the separated TEE module can be used to analyze privacy-concerned (meta-)data, i.e., access patterns of IP addresses. This way, the application in the TEE can be fed traffic data. Inside the TEE, the traffic is monitored to detect suspicious or malicious traffic, i.e., (DOS) attacks. The trusted application can return required actions, e.g., blocking users or IP addresses. The secret data cannot be accessed from outside the TEE to ensure privacy.

### B. P4 Pipeline inside the TEE

The *secure pipeline* puts the whole P4 pipeline into the TEE (cf. Fig. 1b). The whole packet processing, including access and modification to all header fields, is secured in the trusted environment. Packets are copied from/into the TEE and processed as a whole. Moreover, packets can potentially be altered before or after the pipeline during the exchange with the NIC since the access to the NIC is still untrusted. Using SEV-TIO would provide a remedy: fetching the packets from the NIC would be possible from inside the TEE, preventing untrusted modification and additional copies.

*Use cases:* include trustworthy forwarding and routing. This approach secures the whole packet processing pipeline. In addition to the extern approach, unencrypted header data, e.g., IP addresses and ports, is also protected. These header fields may influence the control flow of the packet processing. However, there is no split so that the whole pipeline can access all information. In the case of privacy-enhanced monitoring—in contrast to the extern approach—the monitoring state, including potential user data is accessible by the whole packet processing pipeline, reducing isolation.

## V. Implementation

*a) P4 extern with SGX:* Due to the split architecture, we implement the P4 TEE extern using Intel SGX (cf. Fig. 2a). Theoretically, externs can be implemented in AMD SEV; however, running a VM for just a single operation may not justify the overhead. Furthermore, the processing logic must be highly adapted to fit the communication model with a VM.

As only the extern runs in a TEE, we can run T4P4S the common way, directly on the CPU cores. T4P4S utilizes DPDK to directly access the NIC using DMA. The P4 pipeline is generated by T4P4S out of the P4 program. The extern code which runs inside the SGX enclave is written in C. The input fields and the output are copied from/to the enclave. T4P4S has to be adapted/extended to run the extern.

*b) Secure P4 pipeline in SEV-SNP VM:* To implement the whole P4 packet processing pipeline in a TEE, we use SEV-SNP Linux VMs, shown in Fig. 2b. We run a patched Ubuntu inside the VM with the required SEV extensions for the Linux kernel. It is not possible for T4P4S/DPDK to directly access the NIC, even if the NIC is exclusively bound to the VM (cf. Sections II-0e & II-0f). Therefore, we used the Linux kernel driver of the guest system to overcome that issue and built an `AF_XDP` socket that T4P4S/DPDK can access. Our experiments showed that it is required to run the `AF_XDP` socket in the copy mode (by setting a specific flag), leading to an additional copy of every packet. Then, T4P4S/DPDK can run without further required modifications, with the cost of additional copies by the bounce buffer and the `AF_XDP` socket. However, as no modifications to the code are required, the application can easily be used in an untrusted environment as well. Additionally, it may be used in a TIO environment without XDP later, probably performing better.

## VI. Performance Model

We model I/O overhead and performance based on $r_{rx}$ (ingress of DuT) and $r_{tx}$ (egress of DuT). To build our model, we define, $r_{max}$: the maximum packet rate transmitted, $n$: number of packets received in a batch, $t_{rx}(n)$, $t_p(n)$, $t_{tx}(n)$: time to receive/process/sent all $n$ packets of a batch, $t_b(n)$: total time to handle all $n$ packets. We make the following assumptions: (i) the packets have a fixed size, (ii) the number of packets in a batch is constant but, depending on the packet rate, may be lower than the maximum batch size, (iii) packets are only lost if more packets arrive than can be processed and the batch size is already maximized, (iv) times to receive, process, and send a batch depends linearly on the number of packets (factor $b$); additional constant per-batch overhead (constant $a$) is possible. Based on the assumptions, we model:

$$t_b(n) = t_{rx}(n) + t_p(n) + t_{tx}(n) \quad t_{rx}(n) = a_{rx} + n \cdot b_{rx}$$
$$t_p(n) = a_p + n \cdot b_p \quad t_{tx}(n) = a_{tx} + n \cdot b_{tx}$$

To approximate parameters $a_{rx}, a_{tx}, b_{rx}, b_{tx}$ we conducted measurements (1500-byte packets) to determine $n$ by using different $r_{rx}$ using linear-least squares. Using the model, measured values for $r_{tx}$ can be split into the I/O and processing times. On our setup, we measure these times for different $r_{rx}$ and apply the model to calculate the general shares of I/O and processing time for the packets. We provide all model parameters for all experiments on GitHub [15].

## VII. Evaluation

*Scenario:* Our two-host topology consists of a Device under Test (DuT) and a load generator (LoadGen). The DuT runs T4P4S with our extensions for TEEs. The P4 programs forward all incoming packets and emulate an operation on

Table II: Setup configurations

| Setup | CPU | NIC |
|---|---|---|
| *A* | Intel Xeon Gold 6421N (1.8 GHz) | Intel E810 (100 Gbit/s) |
| *B* | Intel Xeon Gold 6312U (2.4 GHz) | Intel E810 (100 Gbit/s) |
| *C* | AMD EPYC 9354 (3.25 GHz) | Intel E810 (100 Gbit/s) |
| *D* | AMD EPYC 7543 (2.8 GHz) | Intel E810 (100 Gbit/s) |

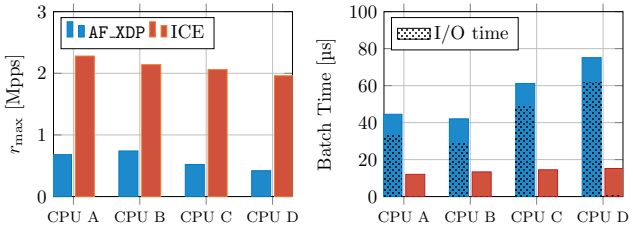



Figure 3: Maximum packet rates ($r_{\max}$), batch processing times ($t_b(32)$) *without* TEE, and I/O times ($t_{rx}(32) + t_{tx}(32)$)

an encrypted header field that XORs (decrypt), increments (operation), and XORs (encrypt) a 4 B field. This happens inside the enclave for SGX experiments. The load generator utilizes MoonGen [16] to generate constant bitrate traffic with a default packet size of 1500 B, while measuring the corresponding latencies and throughputs.

*Configuration:* Table II lists the four investigated DuT setups. The DuT runs Ubuntu Jammy with AMD's SEV-SNP host kernel for the SEV-SNP hypervisor or else AMD's SEV-SNP guest kernel based on Linux 6.7.0. All four Setups *A*–*D* were investigated using the same kernel and AMD's SEV-SNP QEMU v8.2.0 to eliminate any impact of version differences on measurement results. We assign 1 GB hugepages for DPDK: 48 GB for the host system, and 32 GB for VMs, if used. The TX/RX ring sizes are set to 1024. Experiments involving XDP drivers are configured for busy polling according to the DPDK documentation, with a `busy_budget` of half the batch size.

### A. Baseline

First, we determine the baseline performance of a P4 forwarder to calculate the overhead of the different TEE implementations. Our baseline measurements use the four listed CPUs, with the "normal", so-called ICE driver in polling mode and the `AF_XDP` driver in copy mode.

*Throughput:* Fig. 3 shows the maximum packet rates for a baseline forwarder (bare-metal, single core). The forwarding rates are slightly higher for the Intel CPUs (2.28/2.14 Mpps, for *A*/*B*), than for the AMD CPUs (2.06/1.96 Mpps, for *C*/*D*) using the ICE driver. Rates significantly decrease on `AF_XDP` due to the additional copies. For the Intel CPUs, the forwarding rates decrease to 0.68 Mpps (29.8 % of the ICE driver) for CPU *A*, and 0.74 Mpps (34.6 %) for CPU *B*. Again, the AMD CPUs perform worse, in absolute and relative terms. Using the `AF_XDP` driver, $r_{\max}$ reduces to 0.52 Mpps (25.2 %) for Setup *C*, and 0.42 Mpps (21.4 %) for Setup *D*. Notably, no correlation between throughput and CPU clock rates is observable.

*Time:* We measured the batch processing times and the overhead for generated packet rates between 0.1–3.0 Mpps to calculate the model parameters $a_{tx}$, $a_{rx}$, $a_p$ and $b_{tx}$, $b_{rx}$, $b_p$. Applying these, we calculate the I/O overhead for the optimal case (batch size $n = n_{max} = 32$).

Using the model, the batch processing times and the I/O overhead can be calculated (cf. Fig. 3). While the processing timesare nearly constant in all cases, the I/O overhead is more significant for `AF_XDP`. The XDP I/O overhead is relatively lower for Intel than for AMD CPUs. Single-packet I/O takes between 800–1000 ns ($b_{rx} + b_{tx}$) on Intel and >1400 ns on AMD CPUs. The DPDK driver is more efficient, and due to the small number, it is hard to measure the exact share, but the relative difference is about 100 to 150 times.

### B. Overhead of TEE

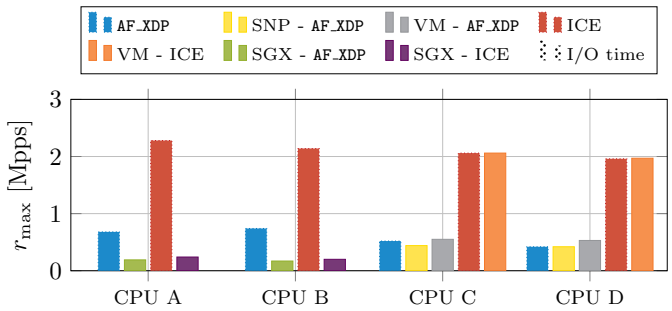
After determining the baseline, we can now measure the overhead of the TEEs. For AMD CPUs, we additionally compare it with a VM setup without SEV-SNP to see the overhead produced by each part of it.

*Throughput:* Again, we first investigate the influence of the TEE technologies on the maximum throughput. Fig. 4a depicts the $r_{max}$ for the different scenarios: bare-metal without TEEs (for comparison), SGX, VM without TEE, and SEV-SNP. Each experiment was performed with both drivers on a single CPU core with a packet size of 1500 B.

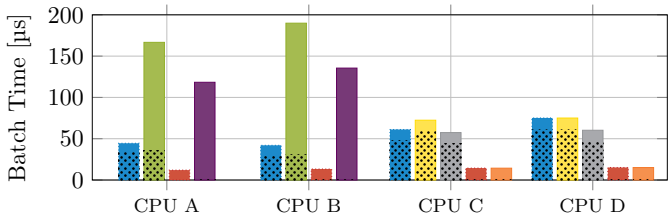
First, we look into the *extern* approach implemented using Intel SGX. There is a considerable performance drop for both investigated Intel CPUs when packets have to traverse the secure enclave. $r_{max}$ falls from 2.28 Mpps to 0.24 Mpps for CPU *A*, which is only 10.5 % of the baseline performance. A similar picture can be drawn for CPU *B*: there, $r_{max}$ decreases from 2.14 Mpps to 0.20 Mpps, which is 9.3 %. The performance for the `AF_XDP` is depicted only for completeness but not used in the extern approach.

Second, we investigate the *secure P4 pipeline* approach, using AMD SEV-SNP. We can only use the `AF_XDP` driver for that and depict the performance of the ICE driver only for completeness. For both CPUs, the performance is similar. Surprisingly, using a VM on Setup *C* improves $r_{max}$ from 0.52 Mpps to 0.55 Mpps, which is 105.7 % of the baseline. We speculate that the memory alignment of the VM is improved by chance, enhancing cache performance. Enabling SEV-SNP reduces $r_{max}$ to 0.44 Mpps, which is 84.6 % of the baseline, and 80.0 % of the vanilla VM. It is again similar to the other AMD CPU *D*: Baseline $r_{max}$ is 0.42 Mpps, 0.53 Mpps (126.2 %) for vanilla VM, and 0.42 Mpps for SEV-SNP (100 % or 79.2 %, respectively).

While the overhead for SGX is relatively high, the overhead of SEV-SNP is handy. For this solution, most overhead is introduced by the required `AF_XDP` driver. Comparing both solutions, the SEV-SNP achieves a higher $r_{max}$, 0.44/0.42 Mpps for SEV-SNP, compared to 0.24/0.22 Mpps for SGX, approximately half. The performance is better, even though all the packets have to be copied several times. On the other side, the input and output values of the enclave have to be copied as well. Additionally, context switches are required.




(a) Maximum packet rates ($r_{max}$)




(b) Batch processing times ($t_b(32)$), I/O times ($t_{rx}(32) + t_{tx}(32)$)

Figure 4: Different drivers with *and* without TEE

*Time:* We can again model the parameters for the I/O overhead of the batch times (cf. Fig. 4b). For experiments with a low $r_{max}$ (i.e., SGX), the data points to create the linear approximation functions are limited, as the system becomes overloaded quickly, resulting in maximum batch sizes.

Fig. 4b shows similar I/O overhead inside and outside of the SGX enclave. The additional workload arises from higher processing times due to the SGX enclave transition. For SEV-SNP, i.e., in Setup *C*, the processing share stays nearly the same for SEV-SNP as without TEE. However, the I/O costs increase for $t_b(32)$ between 44.54–59.44 µs (133.5 %) due to the additional copy. But, we can measure better performance for a VM than bare-metal, and we cannot calculate a significant difference for the CPU *D*. Thus, the overhead of SEV-SNP is minimal despite the additional copy. Comparing the per-packet processing times ($b_p$) for SEV (408/429 ns) and SGX (2249/1560 ns), we can, despite the decreased precision of the model, observe the overhead produced through the context switches between untrusted part and the secure enclave.

## VIII. Discussion & Conclusion

We investigated two different approaches for TEE together with P4 pipelines. The *extern* approach builds a TEE next to the pipeline and relies on Intel SGX. The *secure pipeline* implements the whole P4 pipeline inside AMD SEV-SNP. We aimed not to build a custom solution but to use DPDK applications without required modifications to ensure their portability. However, this requires workarounds using bounce buffers and *AF_XDP* involving additional packet copies and negatively influencing the performance. Nevertheless, we showed that the *secure pipeline* using SEV-SNP offers higher throughputs. Using SGX enclaves for the *extern* approach drastically increases processing times for the required context switches. While the SGX extern can be optimized further, as related work shows, it will likely not scale as well as SEV-SNP.

Future work may investigate scenarios based on latency and multi-core scaling. The *secure pipeline* approach may be implemented using Intel TDX and compared to AMD SEV. SEV-TIO devices might increase the performance without fundamental changes to the application. A reevaluation will be worthwhile after first NICs with SEV-TIO become available.

## Acknowledgments

This work was supported by the EU's Horizon 2020 programme as part of the projects SLICES-PP (10107977) and GreenDIGIT (101131207), by the German Federal Ministry of Education and Research (BMBF) under the projects 6G-life (16KISK002) and 6G-ANNA (16KISK107), and by the German Research Foundation (HyperNIC, CA595/13-1).

## References

[1] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki, "T4P4S: A Target-independent Compiler for Protocol-independent Packet Processors," in *19th International Conference on High Performance Switching and Routing, HPSR, Bucharest, Romania*. IEEE, 2018.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *CCR*, vol. 44, no. 3, pp. 87–95, 2014.

[3] F. Parola, R. Procopio, R. Querio, and F. Risso, "Comparing User Space and In-Kernel Packet Processing for Edge Data Centers," *CCR*, vol. 53, no. 1, p. 14–29, Apr. 2023.

[4] AMD, "AMD SEV-TIO: Trusted I/O for Secure Encrypted Virtualization," 2023, Last accessed: 2025-01-27. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/developer/sev-tio-whitepaper.pdf

[5] M. Li, Y. Zhang, Z. Lin, and Y. Solihin, "Exploiting unprotected I/O operations in amd's secure encrypted virtualization," in *Security Symposium (USENIX Security), Santa Clara, CA, USA*. USENIX, 2019.

[6] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel," in *International Conference on emerging Networking EXperiments and Technologies (CoNEXT), Heraklion, Greece*. ACM, 2018.

[7] H. Duan, C. Wang, X. Yuan, Y. Zhou, Q. Wang, and K. Ren, "LightBox: Full-stack Protected Stateful Middlebox at Lightning Speed," in *Conference on Computer and Communications Security (CCS), London, UK*. ACM, 2019.

[8] J. Medina, N. Paladi, and P. Arlos, "Protecting OpenFlow using Intel SGX," in *Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Dallas, TX, USA*. IEEE, 2019.

[9] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "ShieldBox: Secure Middleboxes using Shielded Execution," in *Symposium on SDN Research (SOSR), Los Angeles, CA, USA*. ACM, 2018.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, p. 263–297, Aug. 2000.

[11] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "SafeBricks: Shielding Network Functions in the Cloud," in *Symposium on Networked Systems Design and Implementation (NSDI)*. Renton, WA: USENIX, 2018.

[12] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," in *Symposium on Operating Systems Design and Implementation (OSDI)*. Savannah, GA: USENIX, 2016.

[13] J. Thalheim, H. Unnibhavi, C. Priebe, P. Bhatotia, and P. Pietzuch, "rkt-io: A Direct I/O Stack for Shielded Execution," in *European Conference on Computer Systems (EuroSys)*. New York, NY, USA: ACM, 2021.

[14] M. Li, S. Srivastava, and M. Yan, "Bridge the Future: High-Performance Networks in Confidential VMs without Trusted I/O devices," *CoRR*, vol. abs/2403.03360, 2024.

[15] "GitHub: manuel-simon/netsoft-2025," Last accessed: 2025-04-29. [Online]. Available: https://github.com/manuel-simon/netsoft25-results

[16] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Internet Measurement Conference, IMC Tokyo, Japan*. ACM, 2015.