Chair of Network Architectures and Services
School of Computation, Information and Technology
Technical University of Munich

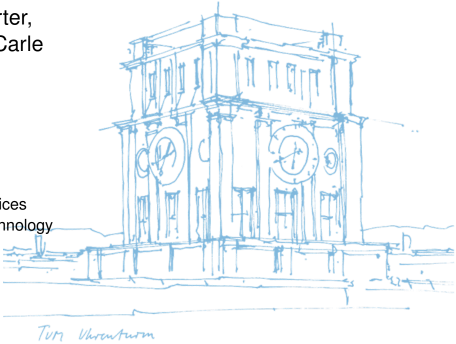
TEE Time at P4—Performance Analysis of Trusted Execution Environments for Packet Processing

Manuel Simon, Sebastian Warter,
Sebastian Gallenmüller, Georg Carle

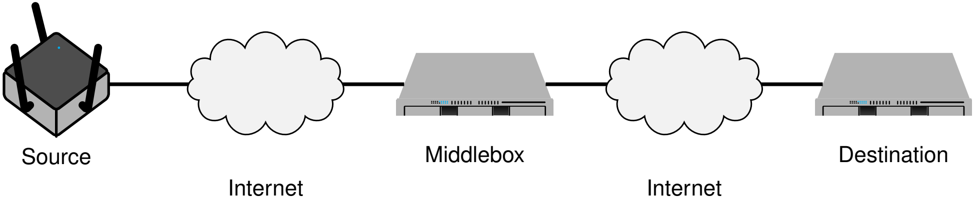
Wednesday 25th June, 2025

IEEE NetSoft 2025

Chair of Network Architectures and Services
School of Computation, Information and Technology
Technical University of Munich

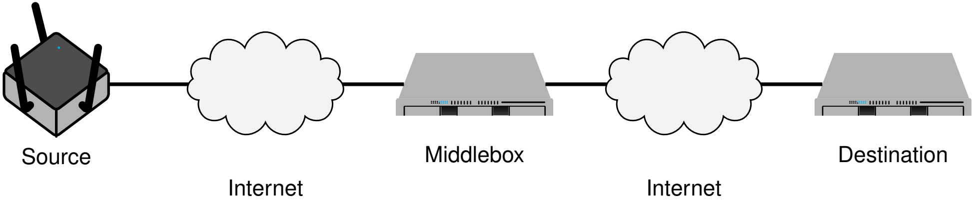


Motivation



Introduction

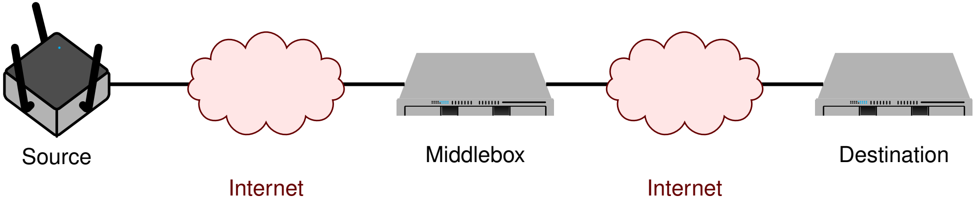
Motivation



What about confidentiality?

Introduction

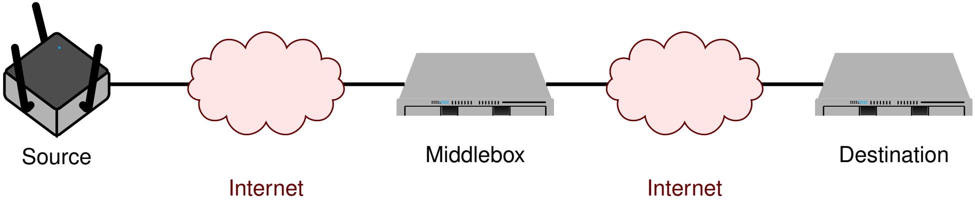
Motivation



What about confidentiality?

Introduction

Motivation

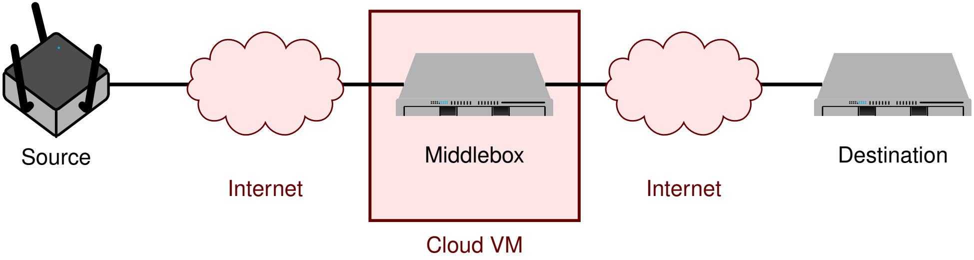


What about confidentiality?

- We don't trust the Internet → Encrypt message

Introduction

Motivation

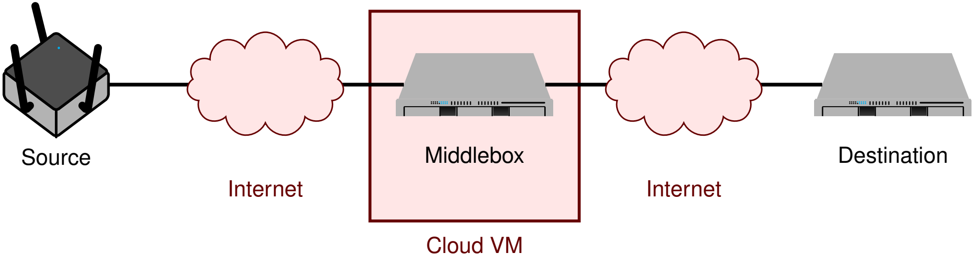


What about confidentiality?

- We don't trust the Internet $\rightarrow$ Encrypt message
- We don't trust the cloud provider $\rightarrow$?

Introduction

Motivation



What about confidentiality?

- We don't trust the Internet → Encrypt message
- We don't trust the cloud provider → ?
- We don't trust "ourselves" (privacy laws, customer trust) → ?

Only trust the CPU manufacturer using TEEs:

- CPU encrypts memory with not-accessible key
- CPU can attest that the correct code is running

Implementation of TEEs:

	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	✓	×

Other implementations (not covered): Intel TDX, ARM TrustZone

Problem statement:

- We want to use a standardized, high-level language for packet processing
- We want to use TEE for confidential processing
- We want to compare TEE implementations for packet processing

⇒ We integrate TEEs into the P4 pipeline, a high-level language for packet processing

Contribution:

⇒ **Two** designs/implementations for TEEs on P4 devices
⇒ Common framework/use case to compare TEE implementations
⇒ We use DPDK-based T4P4S for the implementation and analysis

P4 [1]:

- High-level language to program data plane
- Nowadays also targeting the end-hosts with Portable NIC Architecture (PNA)
- So-called `extern`s allow the integration of target-dependent, non-P4 functionality

T4P4S [10]:

- Open-source software P4 target transpiling P4 code to DPDK code

DPDK:

- High-performance packet processing framework
- Runs in *user space* and bypasses the Linux Networking Stack
- Polls batches of packet from NIC using DMA

Related Work

- **LightBox[1]:** SGX-enabled implementation of secure middleboxes; uses complex setup with custom virtual network interfaces
- **ShieldBox[2]:** creates secure containers leveraging SGX enclaves; built on Click [4] and SCONE
- **SafeBricks[3]:** secures NF execution inside SGX enclaves; splits DPDK architecture in trusted and untrusted parts; shared buffer for communication; built on NetBricks [6]
- **rkt-io[4]:** runs customized DPDK inside Intel SGX with direct userpace network I/O stack; provides POSIX socket API
- **Bridge the Future[5]:** kernel module allowing hardware access from DPDK inside an AMD-SEV VM

⇒ Our approach integrates TEE execution in the established P4 programming language

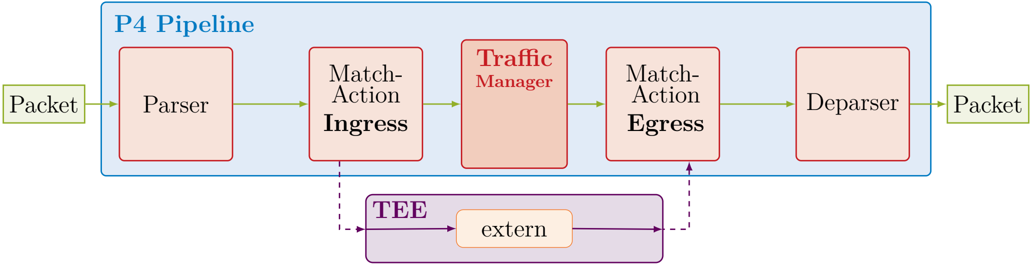
⇒ Our solution (i.e., pipeline approach) does not rely on custom solutions and allows replacement with upcoming technologies

⇒ Our solution provides a framework/use case for performance comparison for TEE technologies

[1] Duan et al.: LightBox: Full-stack Protected Stateful Middlebox at Lightning Speed [2]
[2] Trach et al.: ShieldBox: Secure Middleboxes using Shielded Execution [9]
[3] Poddar et al.: SafeBricks: Shielding Network Functions in the Cloud [7]
[4] Thalheim et al.: rkt-io: a direct I/O stack for shielded execution [8]
[5] Li et al.: Bridge the Future: High-Performance Networks in Confidential VMs without Trusted I/O devices [5]

Design & Implementation

TEE as P4 extern



Design:

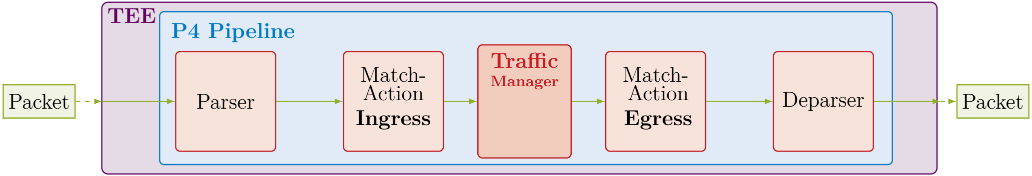
- Standard, fast packet processing defined in P4
- Hardcoded `extern` functions inside TEE
- \+ Well-defined API
- \+ Extern can be called for selected packets
- \- Normal packet processing, i.e., routing not protected

Use cases:

- Trusted computation on secret data
- Processing on privacy-concerned (meta-)data
- Trusted application can return required actions, without leaking private data

Design & Implementation

P4 Pipeline inside TEE – Secure Pipeline



Design:

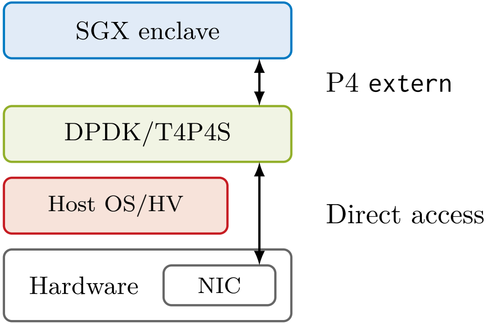
- Whole P4 pipeline inside TEE
- \+ Secures entire packet processing, all header accesses, and control flow
- \- Reduced isolation

Use cases:

- Trustworthy, secure routing
- Trustworthy packet processing

Design & Implementation

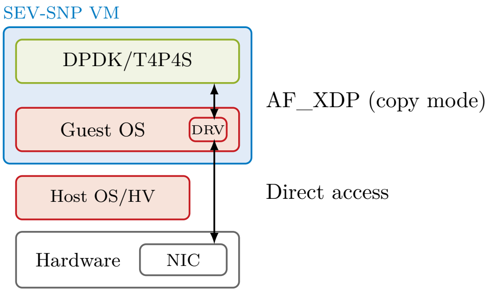
Extern using Intel SGX



- T4P4S/DPDK runs the typical way on bare-metal hardware
- T4P4S generates code for P4 pipeline
- Pre-defined `extern` code runs in SGX enclave, written in C
- Input/output fields are copied to/from enclave

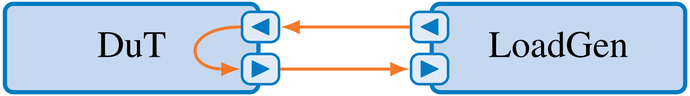
Design & Implementation

P4 Pipeline in AMD SEV-SNP



- T4P4S/DPDK runs inside Ubuntu SEV-SNP VM
- Bounce buffers (swiotlb) copy packets from unprotected DMA area
- `AF_XDP` socket in copy mode to transfer packet from kernel to user space
  - ⇒ *two* copies required
- I/O still unprotected
- SEV-TIO would guarantee protected and more performant I/O

Setup



Configuration

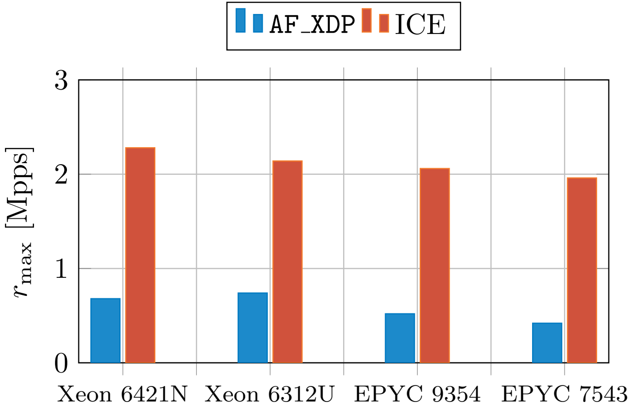
- *CPU*:
  - Intel Xeon Gold 6421N (1.8 GHz)
  - Intel Xeon Gold 6312U (2.4 GHz)
  - AMD EPYC 9354 (3.25 GHz)
  - AMD EPYC 7543 (2.8 GHz)
- *NIC*: Intel E810 (100 Gbit/s)
- *OS*: Ubuntu Jammy (with AMD SEV-SNP kernel extensions)

Scenario

- DuT runs T4P4S with TEE extensions
- *XOR* to emulate en-/decrypt of 4 Byte header field as secured operation
- LoadGen runs MoonGen [3] for traffic generation and measurements
- CBR traffic with 1500 Byte packet size
- Performance model used to calculate I/O shares, more details in paper

Evaluation

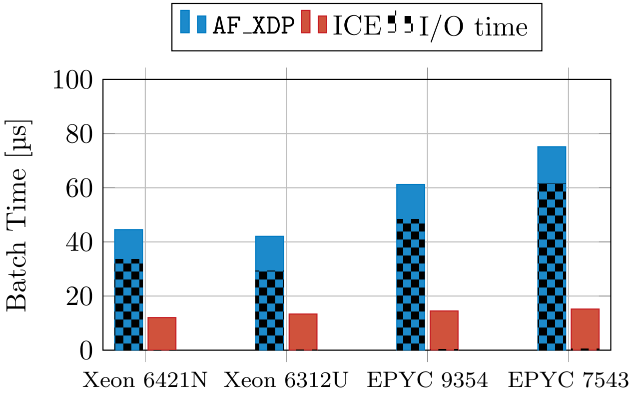
Baseline — Throughput



Baseline: Simple forwarder *without* TEE

- Intel CPUs with slightly higher throughput
- **AF_XDP** reduces throughput by 65 %–82 %

Baseline — I/O Overhead

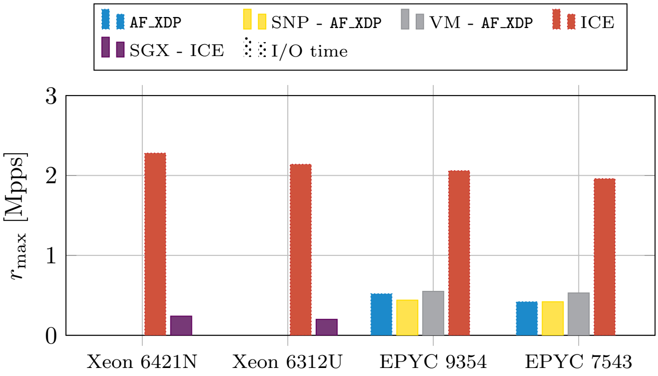


Baseline:

- Using performance model and different packet rates, we calculate I/O shares of processing times
- Overhead of **AF_XDP** lays mostly in I/O operations due to additional copies
- Processing times similar for both drivers
- I/O overhead bigger for Intel CPUs

Evaluation

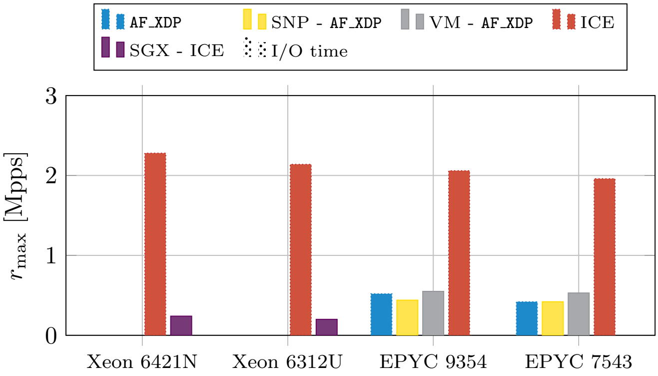
TEE implementations — Throughput



Extern approach/Intel SGX:

- **SGX** decreases performance by 90 %

TEE implementations — Throughput



Extern approach/Intel SGX:

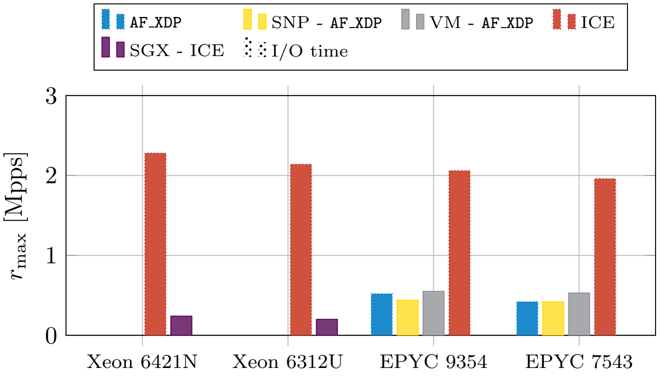
- **SGX** decreases performance by 90 %

Secure pipeline/AMD SEV-SNP:

- **AF_XDP VM** offers similar performance to **bare-metal AF_XDP**

Evaluation

TEE implementations — Throughput



Extern approach/Intel SGX:

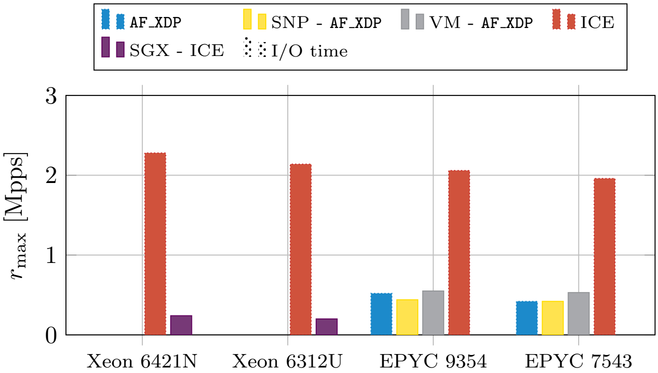
- **SGX** decreases performance by 90 %

Secure pipeline/AMD SEV-SNP:

- **AF_XDP VM** offers similar performance to **bare-metal AF_XDP**
- **SNP** reduces performance by 0 %–15 % compared to **bare-metal AF_XDP**

Evaluation

TEE implementations — Throughput



Extern approach/Intel SGX:

- **SGX** decreases performance by 90 %

Secure pipeline/AMD SEV-SNP:

- **AF_XDP VM** offers similar performance to **bare-metal AF_XDP**
- **SNP** reduces performance by 0 %–15 % compared to **bare-metal AF_XDP**

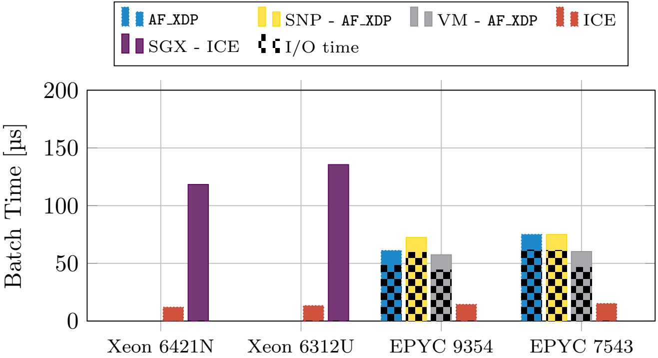
Comparison:

- Secure pipeline using *SEV-SNP* allows for approx. double the throughput than the extern approach using *SGX* (0.44/0.42 Mpps compared to 0.24/0.22 Mpps)
- However, both solutions perform worse than **bare-metal ICE driver** solutions

⇒ SEV-TIO and PCI-TDISP would increase performance significantly

Evaluation

TEE implementations — Overhead



Extern approach/Intel SGX:

- **SGX** enclave transition produces overhead (i.e., context switches and copy of data from/to enclave)

Secure pipeline/AMD SEV-SNP:

- **SNP** has slightly higher I/O overhead, due to additional (second) copy of packet

Conclusion

Contributions:

- Implemented two approaches for TEEs in P4
- Used an architecture which allows for easy exchange with other technologies (i.e. TDX, SEV-TIO)

Findings:

- AMD-SEV offers better scalability compared to Intel SGX
- However, using AMD-SEV within DPDK without adoptions comes with the performance penalty of two required copies

Future Work:

- Analyze multi-core performance, influence of packet size, packet rate, and latency
- Evaluate and integrate Intel TDX, ARM TrustZone, and SEV-TIO (when available)

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.

Bibliography

[1] 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, 44(3):87–95, 2014.

[2] 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.

[3] 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.

[4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek.
The Click Modular Router.
ACM Trans. Comput. Syst., 18(3):263–297, Aug. 2000.

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

[6] 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, 2016. USENIX.

[7] 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, 2018. USENIX.

Bibliography

[8] 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, 2021. ACM.

[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] 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.