SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Performance Analysis and Optimization of Confidential Virtual Machines

Luca Mathias

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Performance Analysis and Optimization of Confidential Virtual Machines

Leistungsanalyse und Optimierung Vertraulicher Virtueller Maschinen

Author:	Luca Mathias
Supervisor:	Prof. Dr-Ing. Pramod Bhatotia
Advisor:	Dr. Masanori Misono
Submission Date:	31.10.2024

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 31.10.2024 Luca Mathias

Acknowledgments

My best thanks to my advisor, Dr. Masanori Misono, for his continued support throughout this thesis and for taking the time to meet weekly to guide me through it.

I also want to thank Prof. Dr. Pramod Bhatotia and the TUM Systems Research Group for providing me with the opportunity to work on such an interesting topic. I learned a lot along the way.

Abstract

Confidential Virtual Machines (CVMs) enable secure computation in the cloud, protecting data in use from an untrusted system using main memory encryption, CPU state protection, and more. Multiple hardware vendors have designed architectural extensions to make them possible, with AMD SEV-SNP and Intel TDX being the most prominent technologies for the x86-64 architecture. The additional steps necessary to provide robust security guarantees cause decreased performance for CVMs, especially in I/O-heavy environments.

In this thesis, we first present an overview of the inner workings of AMD SEV-SNP and Intel TDX. We design an automated benchmarking system with integrated data collection and conduct a performance analysis featuring multiple benchmarks. We examine the overheads CVMs introduce for network I/O and computational workloads, analyzing their VMEXIT behavior and certain hardware events, such as branch or TLB misses. Our examination includes local and remote networking.

We show that, depending on the scenario, CVMs can exhibit overheads upward of 80% that are caused mainly by their special VMEXIT procedures and disabled features rather than memory encryption or a higher rate of hardware miss events. However, while our analysis finds many indications of why performance might be decreased, we cannot conclusively identify a root cause in every scenario. We further present that guest-side CPU polling is an easy-to-implement and general optimization strategy that can significantly reduce overheads by eliminating unnecessary and costly VMEXITs due to CPU inactivity.

Contents

1 Introduction

Cloud computing is omnipresent in the modern world. At its core lies system virtualization, enabling the scalable and rapid deployment of computing services to governments, companies and individuals alike. With more and more computational needs outsourced to data centers owned by one of the countless cloud providers, the issue of trust is crucial. While drive-encryption techniques make it possible to store data securely and inaccessible to the provider, performing computations on sensitive data still requires trust that the provider is not compromised and does not act maliciously. This is because data in use by Virtual Machines, e.g., register contents or memory pages, can be easily accessed by the hypervisor running said VMs.

In order to solve this problem, the concept of confidential computing arose, providing isolated secure enclaves in which data in use can be protected from the rest of an untrusted system [5, 30]. While these solutions were a step in the right direction, they required modification to the applications running inside the secure environment to allow them to access the new security features.

As the concept of confidential computing evolved further, confidential virtualization techniques were introduced [23, 22, 4, 21]. They enable a secure and confidential virtualized environment or Confidential Virtual Machine (CVM), in which application software can run wholly unmodified and still be protected from unauthorized access and modification by the host system.

As is prominent with new technologies, CVM technology still experiences significant performance overheads compared to traditional VMs. These overheads arise from the additional steps they must take to enforce their security guarantees and are especially prominent in I/O-heavy workloads.

In recent years, there have already been works focusing on analyzing and removing overheads experienced by CVMs [28, 48]. Their analyses, however, are either not focused on the latest available x86-based technologies or not very in-depth. Their proposed solutions provide tangible improvements but require a specific software stack and are not readily applicable in every environment. There has also been an analysis of network I/O performance [33] suggesting CPU polling as an optimization strategy. However, it did not include “two-node”/remote networking.

Therefore, this thesis provides an overview of the performance gap that currently exists between non-confidential VMs and the latest and most prominent CVM technolo-

gies (AMD SEV-SNP and Intel TDX). It focuses on two contrasting settings: network I/O, including local and remote networking scenarios, and purely computational workloads. To this end, several performance metrics with strong real-world relevance are analyzed using various benchmarks.

These are:

- Network latency
- Maximum achievable network throughput using TCP and UDP
- In-memory database server performance
- Web server performance
- Highly parallel computing performance
- Machine learning performance

In this thesis, we show that CVM performance overheads can be higher than 80% in some scenarios and identify VMEXIT-related issues as the main reasons for these severe decreases. To reduce the performance gap, we examine guest-side CPU polling as an easy and low-cost optimization technique to prevent unnecessary VMEXITs. We show that polling effectively improves CVM performance and can reduce overheads by more than 70%.

The remainder of this thesis is structured as follows: Chapter 2 provides some background information on the AMD SEV-SNP and Intel TDX CVM technologies and explains the basics of their functionality. It also briefly introduces other core technologies encountered in the later chapters, such as KVM and QEMU. Chapter 3 details the experimental setup we use in this thesis. It provides information about the hardware and benchmarks used to acquire the data presented in later chapters and explains how they are configured. In addition to the experimental setup, chapter 4 outlines how we design a measurement system to automate benchmark runs and data collection. Chapter 5 presents two types of guest-side CPU polling as an effective optimization method for CVMs. In chapter 6, we conduct a detailed analysis of the performance overheads of CVMs. It is broken down into network I/O performance, including local and remote networking, computational performance, a discussion, and a summary highlighting the most important findings. The results we achieve by enabling polling are also discussed here. Chapter 7 rounds off the thesis with a discussion of related work before we draw a conclusion in chapter 8. Finally, chapter 9 gives an outlook on potential directions for future research work.

2 Background

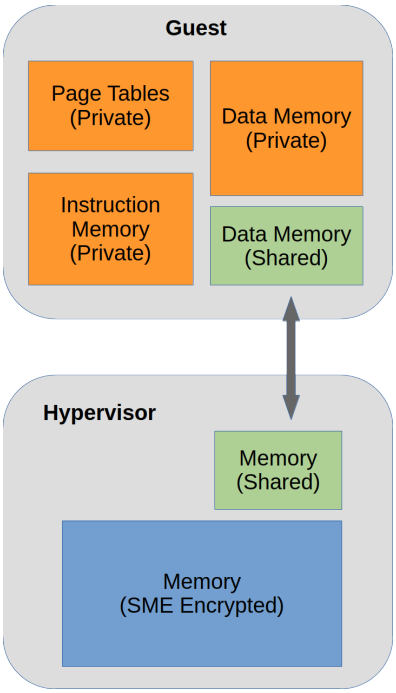
This section provides the relevant background information for the following sections of this work. First, it gives a general explanation of confidential computing and CVMs. Then, AMD's and Intel's CVM platforms are introduced, and their functionality is explained. Finally, we present a brief overview of the virtualization and networking technologies used to set up the analyzed virtual machines.

2.1 Confidential Virtual Machines

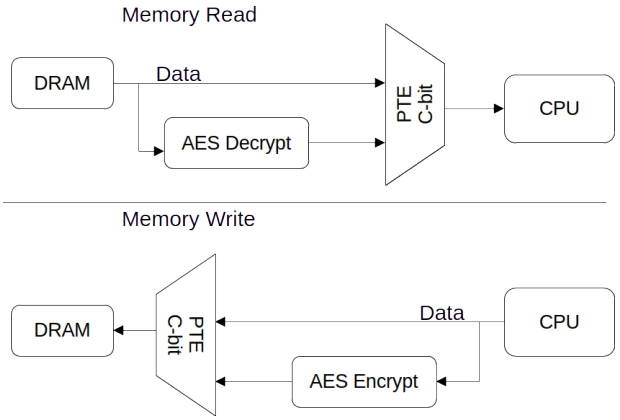
In response to increased security awareness amongst cloud customers, hardware vendors have started offering confidential computing products that provide novel security guarantees. The goal is to create a Trusted Execution Environment (TEE) that minimizes the Trusted Computing Base (TCB), the hardware and software components that need to be trusted [15].

A CVM applies this concept on a VM level. It removes the need to trust the host system and the hypervisor. It only requires trust in the soundness of the specific confidential virtualization technology and the software running inside the CVM (including the guest OS). This is achieved by leveraging hardware-assisted encryption and decryption of the VM's main memory and registers/CPU state, preventing the host from reading data in use by the VM. Data at rest and data in transit continue to be protected by existing methods like drive encryption and Transport Layer Security (TLS). Therefore, the CVM can securely handle sensitive data, even when deployed in an untrusted cloud environment where it could potentially be spied upon or manipulated by other VMs or even the host system/hypervisor.

As replacing local computing infrastructure with cloud services is very prominent, there is a significant demand for secure and confidential cloud computing services that allow safe computations using sensitive data. This led to both Intel and AMD integrating confidential virtualization into their server CPUs through AMD SEV-SNP and Intel TDX, and cloud providers like Google Cloud or Microsoft Azure offering their CVM services [18, 32]. While other CPU architectures, like ARM, also support CVMs [6], this analysis focuses on the technologies offered by Intel and AMD. An overview of their respective approaches to enabling CVM support is provided in the following sections.



(a) AMD SEV memory architecture with public and private pages of memory.



(b) AMD SME encryption process. The page table entry C-bit specifies if the page should be encrypted/decrypted.

Figure 2.1: The AMD SEV memory architecture and AMD SME encryption process, as described in the SEV whitepaper [23].

2.2 AMD

2.2.1 AMD SEV

Secure Encrypted Virtualization (SEV) is the name AMD gave to its confidential virtualization technology. It was first introduced and described by AMD in 2016 [23] and combines existing architectural features like Secure Memory Encryption (SME) and AMD-V (AMD's hardware virtualization extension) to enable encryption of VM data in use.

The encryption of main memory pages is carried out by the memory controller using the Advanced Encryption Standard (AES). This process is managed entirely on the die by a dedicated security subsystem called AMD Secure Processor (ASP) and is isolated from the CPU's cores. The ASP randomly generates and securely stores the unique encryption keys for each CVM on each system reset, protecting them from

unauthorized access.

SEV enables a new security model in which lower privileged code (inside the VM) is cryptographically isolated from higher privileged code (hypervisor). By tagging data as belonging to a particular owner (e.g., a specific CVM or the hypervisor), it is ensured that only the owner can access it. Figure 2.1b shows that by setting a page's C-bit (the highest order address bit) accordingly, each CVM can choose which memory pages it wants to be encrypted and which should remain readable to the hypervisor. Figure 2.1a depicts the SEV memory layout, distinguishing between shared and private memory pages. While pages are by default private and encrypted to ensure their confidentiality, shared pages are necessary to enable communication with the hypervisor.

While SEV can be used without modifying application software, the guest OS needs to be aware of the new hardware features to take advantage of them.

AMD continued to build on the foundations laid by AMD SEV and introduced additional functionality with AMD SEV-ES and AMD SEV-SNP.

2.2.2 AMD SEV-ES

As the first extension to its CVM technology, AMD presented AMD SEV with Encrypted State (ES). While SEV can protect guest memory pages from being read by the host, other locations that might hold sensitive data, like CPU registers, are not protected.

When a VM is suspended, e.g., due to an interrupt, the VM's state, including the register contents, is saved to the hypervisor. A malicious or compromised hypervisor could easily extract secrets like encryption keys from the unencrypted CPU state. Protecting the register contents from being freely accessible by the hypervisor poses a challenge, as in some scenarios, register access is required to perform certain services such as device emulation or Model Specific Register (MSR) handling. To solve this problem, AMD SEV-ES allows the guest VM to choose which data it wants to expose to the hypervisor on a per-case basis.

The process of running and suspending the VM is redesigned to function atomically instead of the multistep process present in traditional AMD-V VMs. Traditionally, only a part of the VM state is saved/loaded using the VMRUN instruction. The hypervisor has to load other data, such as the guest's general purpose or floating point registers, using additional instructions. AMD SEV-ES redesigned this process so that the VMRUN instruction handles it entirely on its own, ensuring it is uninterruptible and no guest information can leak to the hypervisor.

Similarly, when a VMEXIT occurs, the VM state is atomically saved back to memory by hardware, and the hypervisor state is restored. AMD SEV-ES separates VMEXITs into two groups, Automatic Exits (AEs) and Non-Automatic Exits (NAEs). An NAE occurs when the guest requires services from the hypervisor, such as Memory Mapped

I/O (MMIO) or MSR access. In contrast, AEs do not require hypervisor emulation. They include asynchronous interrupts and shutdown events. Only AE events cause a transfer of control to the hypervisor, together with saving and encrypting the guest register state.

The Virtual Machine Control Block (VMCB) is divided into two parts: the "Control Area", which is managed by the hypervisor, and the "Save Area", which includes the guest registers and is saved in a page of memory encrypted with the guest's VM encryption, making it inaccessible to everything else. Loading and saving the state is also combined with an integrity check to prevent unwanted modification of the encrypted register contents. In addition to the VMCB, a Guest Hypervisor Communication Block (GHCB) stored in public memory, is used to facilitate communication between the guest and the hypervisor.

A new Virtual Machine Manager (VMM) Communication Exception (#VC) is defined, which informs the guest OS about the need for hypervisor intervention. Its handler is called when an NAE occurs and has to decide at runtime how to proceed and what state needs to be exposed to the hypervisor to enable it to perform the required services. After copying the relevant state into the GHCB, it hands control to the hypervisor via the new VMGEXIT instruction. This instruction causes an AE, which saves and encrypts the VM state and transfers control to the hypervisor. The hypervisor then reads the GHCB and, as it can not directly modify the VM state, places any new state inside it before resuming the VM. The VM is resumed after the VMGEXIT instruction, meaning the exception handler is still active. It inspects the data provided by the hypervisor and can then copy the new state into the relevant registers [3, 22].

2.2.3 AMD SEV-SNP

As the latest improvement, AMD introduced Secure Nested Paging (SNP), a technology that adds memory integrity protection to AMD SEV-ES. It is based on the principle that if a CVM can read a private page of memory, it should always read the data it last wrote. This means that either the correct value can be read or an exception should be raised to inform the VM that the read attempt failed. These guarantees still have to hold after events such as a swap to disk or a VM migration to a new host.

There are three critical rules this guarantee stands on. Firstly, only the owner of a private memory page is allowed to write to it. Secondly, each physical memory page can only be mapped to one guest memory page simultaneously. Thirdly, each guest memory page only maps to one physical page at a time. Together, the last two rules ensure an injective mapping of guest memory to physical memory, preventing a wide range of possible memory attacks.

The rules are enforced using a structure called Reverse Map Table (RMP). It is a data

Figure 2.2: Overview of the AMD SEV-SNP page table walk featuring the RMP [4].

structure shared over the whole system and tracks the owner of each memory page. The hypervisor, a specific VM, or the ASP may own the pages. This structure is used in conjunction with the regular page tables to ensure that only the page owner can write to it. The RMP is placed at the end of the regular x86 page table walk. If the walk is performed in native/non-VM mode, the RMP is indexed using the resulting System Physical Address (SPA). If it is performed in SEV-SNP context, nested paging [3] is used to first translate a Guest Virtual Address (GVA) into a Guest Physical Address (GPA) and then translate that into a SPA. This SPA is then used to index the RMP. A simplified version of this process is depicted in Figure 2.2.

In the non-VM case, if the RMP indicates that the requested page is a hypervisor/host-owned page, the check passes; if not, a page fault (#PF) is raised. Similarly, for the SNP case, it is checked if the page belongs to the specific guest requesting access. Additionally, the RMP yields the GPA to which the GVA should be mapped. If it does not match the GPA to which it is actually mapped, a nested page fault (#NPF) is raised.

For security reasons, the RMP is not writable by non-privileged software directly, and its entries can only be manipulated using a special set of new CPU instructions reserved for the hypervisor. Each entry contains a validation bit to ensure the hypervisor can not break the SNP guarantees by manipulating the RMP entries. This bit is automatically set to zero when a new guest RMP entry is created. Pages that have been assigned to a guest but have yet to be validated (validation bit equals zero) are not usable by a guest as private memory or by the hypervisor. The guest must first validate the page via

the newly introduced CPU instruction PVALIDATE, which it can only use on its own memory. Therefore, adding a new page to a VM requires a two-step procedure. In step one, the hypervisor assigns the page to the guest by creating the corresponding RMP entry using the RMPUPDATE instruction. In the second step, the guest validates the page assigned to it, making it fully usable. To conform to SNP's standards, the guest should never validate a page corresponding to the same GPA more than once. The implementation details are left to the guest, but should it correctly validate its memory, an injective mapping between the GPAs and SPAs is guaranteed.

SNP also adds some optional features, such as Virtual Machine Privilege Levels, which are not further relevant to this thesis [4].

2.3 Intel

2.3.1 Intel SGX

Intel developed Intel Software Guard Extensions (SGX) to provide protected software containers or "enclaves". SGX leverages already existing Intel technology [14] to provide software running inside one of these enclaves with confidentiality and integrity protection, making it a process-based confidential computing environment.

One of its most significant downsides is that applications need to implement the SGX programming model to take advantage of its security guarantees. The need to modify applications slowed down adoption and introduced significant development costs. Intel has already announced to stop SGX support on consumer platforms and shifted the focus to its more recent CVM technology Intel TDX [8, 30].

2.3.2 Intel TDX

Trust Domain Extensions (TDX) is Intel's newest confidential computing technology and allows for the creation of Trust Domain (TD) featuring confidentiality and integrity protection for both memory and CPU state, address translation integrity, and secure interrupt and exception delivery. Like AMD SEV, it is also based on several pre-existing CPU capabilities, such as Intel's virtualization extensions VT-x or Intel Total Memory Encryption Multi-Key (TME-MK) [10].

To achieve the security guarantees mentioned above, a new CPU mode called Secure-Arbitration Mode (SEAM) is introduced, hosting a security-services module (Intel TDX Module) located in a reserved memory space, the SEAM range. Memory access to this AES-encrypted memory region is only allowed to software running inside it, with all other access attempts, including DMA, being aborted by the CPU. In addition to the encryption, integrity protection is achieved by default using a SHA-3-based Message

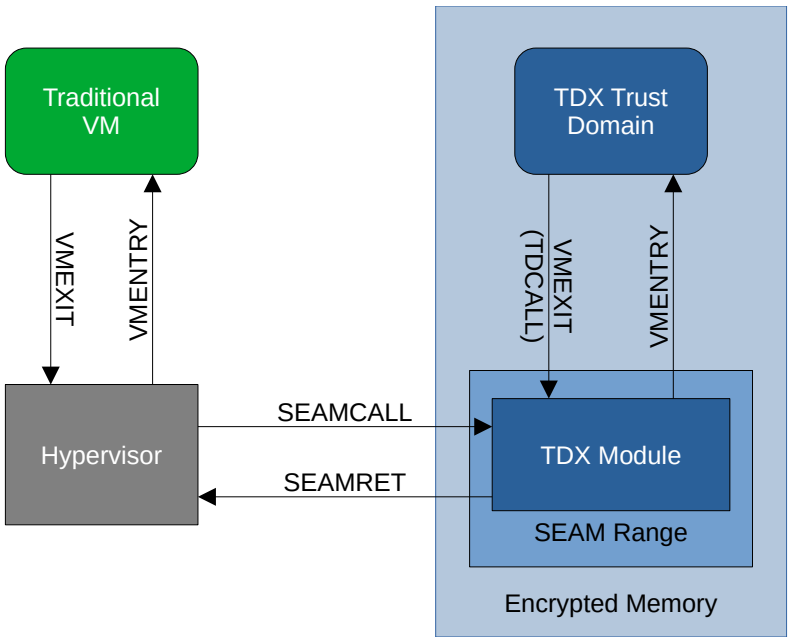


Figure 2.3: A simplified view of Intel TDX's basic architecture with the SEAM memory range, the TDX module, and a Trust Domain [21].

Authentication Code (MAC). The MAC protects against host/system memory accesses, software state tampering, and some physical attacks. A logic-integrity protection scheme can also be chosen, only protecting against host memory accesses.

Figure 2.3 shows the general architecture of TDX, with the hypervisor using the new SEAMCALL and SEAMRET instructions to communicate with the TDX module, which acts as a trusted intermediary between the hypervisor and a TD/CVM. It is responsible for implementing the security policies and does so as follows:

Memory Confidentiality and Integrity

On creation, each TD is assigned a unique encryption key by the TDX module. The CPU prevents anyone except the TDX module or the owner TD from accessing the keys. Like AMD SEV, Intel TDX distinguishes between shared and private memory. While private memory holds all the confidential TD data, shared memory is used to communicate with the host system, allowing hypervisor services and I/O. Whether a page is private or public is determined by the "Shared" bit (the highest order GPA bit). Using TME-MK and the key assigned to the TD, private memory is AES encrypted on the cache line level and enjoys the same SHA-3 MAC integrity protection as the

TDX module's memory. Shared memory may also be encrypted and integrity-protected using a shared key managed by the hypervisor.

Address-Translation Integrity

TDX uses Extended Page Tables (EPTs) to provide GPA to SPA translation. Shared memory address translation is handled by a shared EPT managed by the hypervisor, and a secure EPT managed by the TDX module handles private memory address translation. To ensure the correct and safe mapping of GPAs to SPAs, an additional data structure, the Physical Address Metadata Table (PAMT), is introduced. It ensures, that a page mapped inside a secure EPT of a TD can not be mapped inside the EPT of any other TD, and each page is only mapped to a single GPA.

CPU State Confidentiality and Integrity

Upon creation of a TD, the TDX module requests a set of memory pages from the hypervisor. These pages are reserved for VM control and confidential data structures like the secure EPT. There is also an area where the TD's CPU state is saved. These memory pages are encrypted using the TD's key, and the TDX module prevents any CPU state from leaking to the hypervisor during a VMEXIT. Instead of directly exiting to the hypervisor, the TD calls the TDX module using the new TDCALL instruction. The module then acts accordingly and interacts with the hypervisor on the TD's behalf.

Secure Interrupt and Exception Delivery

Intel TDX is designed to rely on the VMX-APIC virtualization and virtual-interrupts architecture. The Advanced Programmable Interrupt Controller (APIC) is a hardware chip responsible for handling interrupts in a system, while VMX-APIC is responsible for efficiently managing VM access to the APIC. The TDX module manages a virtual APIC through a dedicated memory page similar to the VM control structures.

As an additional security measure, TDX does, by default, not allow the host to collect performance metrics of the guest TD/CVM. While this can be manually enabled in some cases, it is entirely unsupported by some TDX-capable CPUs [21, 9].

In the following parts of this thesis, we use the term CVM to describe both Intel TDX TDs and AMD SEV-SNP VMs.

2.4 Technology Stack

2.4.1 KVM

Kernel Virtual Machine (KVM) is an open-source virtualization technology that is part of Linux and was initially released in 2007.

A hypervisor needs several OS-level components, such as a scheduler, memory management, networking, and many more, to function. All these things, as well as access to privileged virtualization instructions, are already present in the Linux kernel. KVM leverages these capabilities and essentially turns the Linux kernel into a hypervisor. This integrates KVM tightly with Linux, allowing it to benefit from new kernel features or bug fixes without additional engineering effort.

Its primary focus lies with x86_64 systems, and processor-specific virtualization features (Intel VT-x and AMD-V) are made use of by the vendor-specific kernel modules *kvm-intel.ko* or *kvm-amd.ko* that complement the main *kvm.ko* module.

Besides its near-native performance, KVM offers a rich feature set, including, e.g., live migration [44, 27]. It provides a file-descriptor-centred API through */dev/kvm*, accessible through IOCTL system calls [26].

KVM can be combined with QEMU to support virtualized hardware devices and provide fully featured virtual machines. Therefore, the combination KVM/QEMU often appears in the context of KVM-based virtualization.

2.4.2 QEMU

QEMU is open-source emulation and virtualization software that supports several use cases, the most popular of which is system emulation in user space. It can fully emulate the CPU by itself or be combined with a hypervisor, such as KVM, to allow the guest vCPUs to be run directly on the host CPU [38].

It further provides an extensive CLI interface through which the virtual environments can be configured and also allows for direct boot from a Linux kernel image without the need to create a full bootable image. Additionally, a wide range of hardware components can be emulated using QEMU. This includes many VirtIO devices [40, 39].

2.4.3 VirtIO

VirtIO (Virtual I/0) is a standardized interface specification for virtualizing I/O devices such as block storage or Network Interface Cards (NICs). It is open-source and maintained by OASIS Open [31].

As this thesis only examines network I/O, virtio-net, a virtual Ethernet card, is particularly relevant. It can be separated into two layers: the control and the data

plane. The control plane enables communication between the guest and the host so that the respective capabilities can be compared, and the data plane can be established or terminated. It is required to be as flexible as possible to support a wide variety of vendors and devices. The data plane is responsible for transmitting the actual data through network packets. It must be as performant as possible to enable maximum packet processing speeds.

By default, both planes are implemented within the QEMU process, requiring a context switch for every packet passing between the host and the guest. To avoid this inefficiency, the vhost protocol allows the data plane to be implemented separately from the QEMU process.

2.4.4 Vhost Protocol

The vhost protocol specifies the establishment of the data plane for a virtio-net device. It can be implemented in the kernel (vhost-net) or in user space (vhost-user).

In this thesis, we use vhost-net, which connects the host kernel directly to the virtual virtio-net NIC inside the guest kernel, bypassing the QEMU process during packet delivery. This enables a form of para-virtualized I/O directly from the host's kernel to the guest VM's kernel, eliminating the need to perform a context switch for every network packet. Therefore, the vhost protocol, in theory, allows for a much higher packet processing speed during network I/O.

2.4.5 swiotlb

With the robust security guarantees CVMs provide, functionality like Direct Memory Access (DMA) can not easily be supported without opening up many new possible attack vectors. Therefore, SNP and TDX VMs need help to offer support for DMA features.

The Software I/O Translation Lookaside Buffer (swiotlb) library provides a mechanism in the Linux kernel that plays a crucial role in maintaining security and functionality despite the lack of DMA support. Created to handle DMA devices with addressing limitations, it is now also used to enable CVM I/O by allocating so-called "bounce buffers" in unencrypted guest memory regions that act as intermediary storage for I/O data. The CPU is then responsible for copying outgoing data from encrypted memory to the bounce buffer and incoming data from the bounce buffer to encrypted memory.

In a CVM setup, a kernel-wide option is set, forcing all DMA I/O operations to use bounce buffers while the kernel DMA layer handles the memory synchronization operations. This mechanism enables existing device drivers to work without modification at the cost of additional data copies [24].

Through setting the appropriate kernel parameters, non-confidential VMs can be forced to enable *swiotlb* for all I/O, just like CVMs.

2.5 VMEXIT

The previous sections mention that Intel TDX and AMD SEV-SNP exhibit special behavior during a VMEXIT compared to legacy VMs. Additional measures need to be taken when transferring control back to the hypervisor to protect the CPU state. This includes securing it upon exiting and safely reloading it on re-entry.

For AMD SEV-SNP, most VMEXITs are classed as NAE and, therefore, go through the special #VC handler routine before and after transferring control with the VMGEXIT instruction. For Intel TDX, each VMEXIT has to pass through the TDX module intermediary, which performs the steps necessary to secure the TD. Therefore, for both SNP and TDX, VMEXITs are costly operations, causing a latency increase of up to 240% and 472%, respectively [33].

Keeping this in mind is vital during the analysis of CVM performance, as in many scenarios, the VMEXIT cost increase can severely diminish performance. It is, therefore, advisable to try to minimize the number of VMEXITs needed to perform a task.

3 Experimental Setup

This chapter provides an overview of the hardware systems we use and the software we run on them to collect the data for our analysis.

3.1 Testing Machines

We conduct all measurements on servers with two 32-core Non-Uniform Memory Access (NUMA) nodes featuring 64 cores in total. In the case of the AMD SEV-SNP server, the nodes are slotted with two AMD EPYC 9334 32-core processors running at a base frequency of 2.7 GHz. The Intel server is equipped with two Intel Xeon Platinum 8562Y+ processors with a base frequency of 2.8 GHz. Both commercially available processors support the latest version of their vendor's CVM technology. In both cases, we disable frequency scaling and simultaneous multithreading to make the results less prone to fluctuations. We also pin the vCPUs to specific cores to reduce the effect of scheduling on the results.

As the OS, we use NixOS 24.05 (Uakari) [35]. We use Linux kernel version 6.9.0-rc7 on the AMD server and version 6.8.0-rc1 on the Intel server.

For the remote networking measurements, the client machine is a single-socket Intel server featuring a 24-core Xeon Gold 5317 CPU running at 3.0 GHz. This server also runs NixOS 24.04 (Uakari) but with Linux kernel version 5.15.84. The remote server is connected to the main server via an Intel E810-C 100 Gbit/s NIC device.

3.2 Virtual Machine Setup

We execute the benchmarks using VMs based on KVM/QEMU virtualization with VirtIO networking devices, using four sizes of virtual machine configurations. Table 3.1 shows the concrete sizes with the amount of vCPUs and memory.

We boot all VMs using QEMU's direct boot option and set up their networking using IPv4. The VMs are connected to the host via a tap device and a virtual bridge. For the remote measurements, the physical NIC connecting the test servers is enslaved to the virtual bridge, which itself is connected to the tap device, making the VM reachable from the remote server.

Size	vCPUs	Memory
small	1	8 GB
medium	8	64 GB
large	32	256 GB
xlarge	64	512 GB

Table 3.1: Virtual Machine sizes with their respective available resources.

3.3 Benchmarks

We use six benchmarks, four to examine network I/O performance and two to test computational workloads. We choose them to represent a mix of synthetic and real application workloads. The following sections briefly present the benchmarks and their configuration options.

3.3.1 Ping

We use the well-known *ping* program form iputils [52], which is by default installed on most Linux distributions, to measure the network latency differences between CVMs and non-confidential VMs. We ping a medium-sized VM 100 times, with an interval of 0.1 seconds between pings, using a 64-byte packet.

The program sends ICMP Echo Requests to a network host and measures the time until an ICMP Echo Response arrives. While it reports average latency, maximal latency, and some other metrics, we are most interested in the average latency.

3.3.2 Iperf

We use the *iperf* [12] benchmark to measure the maximum achievable network bandwidth between two network nodes. The server runs inside the VM, and the client runs on either the host or the remote system. The benchmark supports both the TCP and UDP protocols, and the most important reported metric is the achieved bitrate. We use it to simulate two scenarios:

Scenario 1: Maximum throughput under high CPU load: In this scenario, we configure *iperf* to use the TCP protocol and a packet size of 128 KB. The goal is to measure the maximum achievable throughput under a high CPU load caused by the large packet size. Therefore, we use a multi-queue tap device (8 queues) and configure the client to run 32 parallel streams to further stress the server CPU.

Scenario 2: Maximum throughput under low/medium CPU load: In this scenario, we configure *iperf* to use the UDP protocol with varying packet sizes of 64, 128, 256, 512, 1024, and 1460 bytes. The goal is to measure the throughput under a lower to medium CPU load. Therefore, we do not use a multi-queue tap device. As *iperf*, by default, limits the target bitrate for UDP, we manually remove the limit and configure the client to use eight parallel streams.

In both scenarios, we run the benchmark for 30 seconds using a VM of size medium.

3.3.3 Memtier Benchmark

The *memtier_benchmark* is developed by RedisLabs and can be used to benchmark NoSQL key-value-based databases [45]. We use it to benchmark a *redis* [46] and a *memcached* [7] server running inside the VM. We conduct the tests both with and without TLS enabled.

While the *redis* server is single-threaded by design, *memcached* supports multiple threads. We again choose a VM of size medium and set up *memcached* to utilize all available eight cores. Communication between the client and the server happens using the standard "redis" protocol if we benchmark the *redis* server or with the "memcache-binary" protocol if we benchmark the *memcached* server.

3.3.4 Wrk

As a tool to test HTTP and HTTPS performance, we use the *wrk* [16] benchmark in conjunction with an *nginx* [49] server instance that is running inside a VM of size medium. It hosts two different files, of the sizes: 100 KB, 1 MB. For both files, we use *wrk* to generate HTTP/HTTPS requests for 30 seconds. We configure *wrk* to use eight threads with a total of 300 connections at one time.

The HTTPS connections use certificates from a locally generated Public Key Infrastructure (PKI). Out of the many metrics reported by this benchmark, we are most interested in the achieved HTTP/HTTPS transfer rate.

3.3.5 NAS Parallel Benchmark UA

The NAS parallel benchmark [34] is developed by NASA and contains several benchmarking programs. We use the "ua" benchmark to examine the performance for highly parallelized computational workloads.

The benchmark solves heat equations, is parallelized using OpenMP [36] and features dynamic and irregular memory access patterns. It comes in different classes that

determine the size of the test problems to compute. For this examination, we choose the largest of the standard classes, “C”, and run the benchmark on a VM of each available size, once with an “ACTIVE” and once with a “PASSIVE” OpenMP wait policy [36].

3.3.6 TensorFlow (BERT)

TensorFlow is an open-source machine learning framework maintained by Google [1]. It can be used to build and train models for various tasks.

Bidirectional Encoder Representations from Transformers (BERT) is a deep learning model designed for natural language processing [11], which can be implemented in TensorFlow. In this configuration, we use it as a further computational test with the squad dataset [43].

We run the model directly on the CPU without any GPU acceleration and configure the application to use all CPUs available to the VM. Then, we measure the throughput of the inference as the result of the benchmark.

4 Data Collection

After describing the experimental setup, including the hardware and software used, this chapter describes the design of the measurement system we use to automate data collection for this analysis. The goal is to facilitate scripted and repeatable benchmark execution and automatically collect essential performance metrics. Additionally, the results we gather should be saved centrally in a format that we can easily examine and analyze.

4.1 Task System

To allow for automatic benchmark execution with configurable parameters, we formalize the execution processes in Python and organize them into CLI-invokable tasks using the PyInvoke framework [13]. With the help of this framework, we design a task to start a VM, allowing for the configuration of the VM's parameters through command-line flags. Among the configurable options are VM size, VM type (Intel, TDX, AMD, SNP), and whether the VM should use VirtIO networking and vhost. Additional QEMU or Linux kernel parameters can be passed in the same way.

By default, the task system starts a VM with the specified parameters and attaches an SSH shell. However, it can also execute commands inside the VM, making it possible to programmatically run the benchmarks introduced in chapter 3. In this case, the task to start the VM is executed, and the metrics capture system is called. The benchmark to be run can be specified via the CLI and determines the specific function called. All relevant configuration options, together with information about the started VM, are handed off to the metrics capture system as arguments. Figure 4.1 shows some example commands illustrating how we can configure different options from the CLI.

```
$ inv vm.start --size medium --type snp --virito-nic --action run-iperf
$ inv vm.start --virito-nic --virtio-nic-vhost --action run-ping --virtio-
    iommu --extra-cmdline swiotlb=524288,force
```

Figure 4.1: Example commands to start a VM with specific parameters and run a benchmark.

Making all of these configuration options available from the command line facilitates the creation of simple scripts we use to achieve repeatable measurements with a consistent and less error-prone setup. The task system also provides tasks to plot the data captured during previous benchmark runs, thereby making the creation of plots for data interpretation scriptable and highly configurable.

4.2 Metrics Capture System

The metrics capture system is composed of two parts. The first part is the collection of benchmark-specific Python functions callable from the task system. Each function performs all the necessary steps to set up and run its respective benchmark. These steps include starting a server inside the VM before connecting to it from a client running on the host or a remote server or just making the VM run a benchmark program.

Besides running the benchmarks, automatically and reliably capturing the reported results is also very important. While we save all benchmark outputs to files for later manual inspection, we also parse the *stdout* and *stderr* streams of the processes running the benchmarks with regular expressions to extract the relevant information. We integrate this parsing procedure into the same functions first called from the task system. The system then saves the extracted data to a local *sqlite* [19] database.

The second part of the metrics capture system deals with capturing additional system metrics during the benchmark runs. To this end, we start metrics capture processes inside the VM and on the host independently of the benchmark processes that are already running. They can be categorized into three categories:

CPU Utilization

We use the *mpstat* [17] program to measure the CPU utilization of both the host and the guest system. While we measure the guest CPU utilization over all cores available to the VM, we restrict the measurement of the host utilization to the cores tasked with executing the host/client workload.

Performance Counters

The program *perf* [37], which is part of the Linux project, allows for the measurement of many performance metrics, such as cache or branch misses. We used it to measure such metrics for the guest VMs. As we run the VMs using KVM, both *perf* instances, the one for the guest and the one for the host, can be run on the host side, minimizing the performance impact on the guest. Unfortunately, Intel TDX CVMs deny the host

access to guest-side performance counters, preventing these measurements from being conducted on TDX guests.

We measure the following *perf* events:

- `kvm:kvm_exits`, to see if traditional VMs and CVMs differ in their required amount of VMEXITs to complete a task.
- `instructions`, e.g., to get an insight into how many of them are necessary to perform a certain action on traditional VMs vs. CVMs and to use them as a norm for other measured events.
- `branch-misses`, as the SNP whitepaper [4] mentions that SNP VMs may perform additional Branch Target Buffer (BTB) flushes to prevent side channel attacks.
- `dTLB-load-misses`, to see if the modified page table mechanism introduced by SNP affects the effectiveness of the Translation Lookaside Buffer (TLB).
- `L1-dcache-load-misses`, to see if SNP security guarantees can cause some cache flushes when switching to the hypervisor.

There are many more hardware counters that could be of interest, but to limit the overhead while measuring them and keep the scope of this analysis focused, we opt to measure only this set of metrics.

VMEXIT information

In addition to the number of VMEXITs that occur, we also measure the reasons for these VMEXITs using *bpftrace* [47]. Unfortunately, Intel TDX reports most VMEXITs only as TDCALLs, and our measurement system does not support a further breakdown, making gaining insight into the system's behavior more complicated. The VMEXIT reasons we pay special attention to are:

- `HLT` This signals that the VMEXIT occurred due to an idle vCPU.
- `MSR` This signals access to a MSR and can reveal some information about what happens in the system at a low level.
- `MMIO` This signal a MMIO memory operation, which, in addition to being used for actual I/O, also facilitates certain low-level configurations such as setting timers in the local APIC.

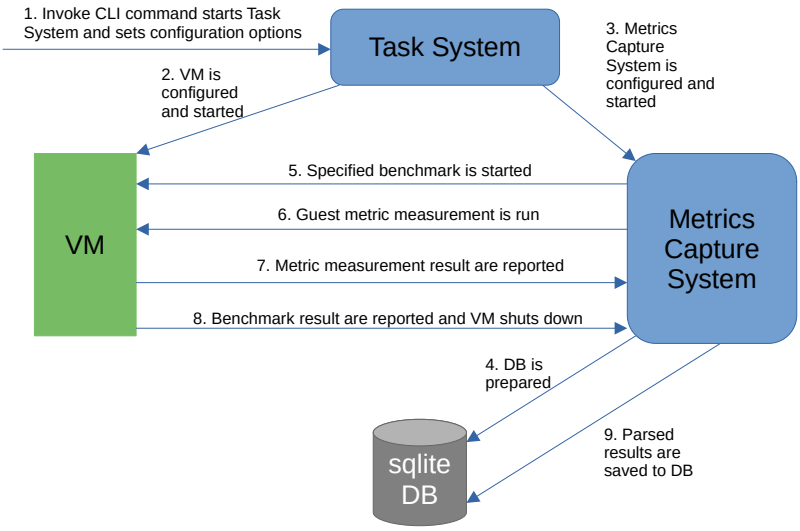


Figure 4.2: A simplified architecture of the measurement system we use to collect the results for this thesis. The events are numbered in chronological order.

For the MSR and MMIO-related VMEXITs, we also observe the addresses/registers that they access.

We do not run the dedicated measurement processes during the whole benchmark but only during a benchmark-specific time window, providing a snapshot of data while not heavily affecting performance. Similarly to the benchmark output, we also parse their produced output and save it to the database, where it is available for analysis. Figure 4.2 shows the basic architecture of the system illustrating the chronological order of events during automatic benchmark execution.

5 Optimizations

As we know from chapter 2, VMEXITs are costly for CVMs and suffer from a significant latency increase in many cases. Therefore, it is advisable to employ all available strategies to limit unnecessary VMEXITs when possible. While exits caused by the need for hypervisor emulation can not be avoided without considerable effort, other types of VMEXITs, most notably those due to CPU inactivity, are not relevant for guest computations and can be significantly reduced via some easy optimizations.

CPUs are seen as idle when they have no tasks to run, with a task being the scheduler's representation of work. If there are no tasks to schedule for a CPU, a special "idle" task is assigned to it, allowing it to enter a low-power state until it is woken up again. This is achieved via the HLT instruction. For the virtual CPUs of a VM, encountering a HLT instruction results in a VMEXIT, transferring control back to the hypervisor.

Setting certain kernel parameters can alter what exactly happens when a CPU encounters a HLT instruction. Specifically, the transition to a low-power CPU state (in the case of a physical CPU) or the VMEXIT (in the case of a vCPU) can be omitted in favor of executing unproductive but lightweight instructions in a loop until new work is available. This mechanism is called "polling".

Intuitively, polling results in higher CPU utilization and higher CPU power draw and, therefore, might not be advisable in some scenarios [42]. CVMs, however, are most likely deployed in a cloud environment, where the customer is typically billed for the VM running time, regardless of how much of the VM's resources are used. Therefore, there is no financial downside to enabling polling on cloud-hosted VMs.

Polling can be implemented from the guest and the host side, but as the goal is to reduce the amount of VMEXITs, and host-side polling does not achieve this, it is of no further relevance. This thesis examines two types of guest-side polling and their effects on CVM performance. They are briefly explained below:

5.1 Idle Polling

Idle polling is the simplest and most extreme form of polling. Should the CPU enter an idle state, it starts busy waiting by executing an infinite loop (similar to a spin-lock) until the scheduler provides new work. This has the downside of essentially locking the

guest vCPU at 100% usage. A VMEXIT only happens if the VM requires services from the hypervisor to continue working or it is stopped by the hypervisor either because of an interrupt or because the host system requires CPU time.

Idle polling can be easily enabled by booting Linux with the kernel parameter *idle=poll* and is often used in latency-sensitive environments where every nanosecond counts.

5.2 Halt Polling

Halt polling is a more complex and adaptable form of polling, where the guest's vCPUs are actively polling for work only for a specified amount of time before the HLT instruction is executed. This should, in theory, be a good compromise between achieving the best performance and preventing excessive resource usage.

Halt polling can be enabled by starting the Linux kernel with the parameter *cpuidle_haltpoll.force=Y* and can further be configured via several parameters. The most important of these are listed here:

- *guest_halt_poll_ns*: The maximum polling time (nanoseconds) before the HLT instruction. The value exits globally and on a per vCPU basis. The default value is 200,000.
- *guest_halt_poll_shrink*: A division factor, by which the vCPU *guest_halt_poll_ns* is divided if a wake-up event occurs after the global *guest_halt_poll_ns*. The default value is 2.
- *guest_halt_poll_grow*: A multiplication factor used to grow the per vCPU value if a wake-up event occurs after it but before the global value. The default value is 2.

These and more parameters can be freely adjusted via a configuration file [25]. However, as this thesis is a general analysis without any specific use case in mind, each value is left on default.

6 Analysis

The last chapters provide the necessary background information on confidential virtualization, present both the experimental setup and measurement system design, and explain guest-side CPU polling as an optimization technique. This chapter analyzes the network I/O and computational performance of the current CVM designs by AMD and Intel, comparing them to non-confidential VMs.

First, we present network I/O performance for local and remote networking; then, we examine computational workloads, and finally, we discuss and summarize the results, highlighting the most important findings. We examine all data captured by the metrics capture system for all measurements but present only the most interesting findings and results. If we do not present one of the metrics we mention in chapter 4, this means that it does not reveal any clear trends or indications relevant to the analysis.

In the following the Virtual Machines examined are categorized as follows:

- "intel/amd" is a non-confidential VM of type Intel/AMD without any special options enabled. It serves as a baseline against which to compare.
- "snp" is a confidential VM of type AMD SEV-SNP.
- "tdx" is a confidential VM of type Intel TDX
- "swiotlb" is a non-confidential VM that we set up to perform I/O using the bounce buffer mechanism using *swiotlb*. It allows us to examine the bounce buffer-related overhead separately from other CVM overheads.
- "poll" is combined with the above categories to signify that idle polling is enabled.
- "hpoll" is combined with the above categories to signify that halt polling is enabled.
- "vhost" is combined with the above categories to signify that QEMU is set up to use the vhost protocol.

6.1 Network I/O Performance

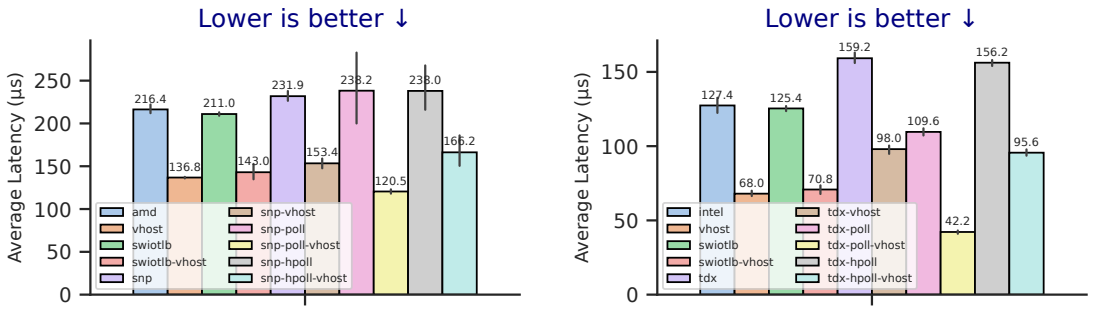
We examine the network I/O performance in two scenarios: firstly, if server and client run on the same physical machine (local networking) and secondly, if they are located on different physical machines connected via a physical NIC (remote networking).

CVM network I/O performance using local networking has already been analyzed [33, 28] but on different physical machines than the ones we use for this analysis. Therefore, we include local measurements alongside the more real-world-scenario-oriented remote measurements. As we start with local networking results for each benchmark, we discuss all general trends visible in both types of networking there. When discussing the results of remote networking, we only focus on changes compared to local networking to unnecessary repetition.

6.1.1 Network Latency

Our procedure to test network latency barely generates any computational load. This leads to a problematic situation when examining hardware performance metrics, as it is hard to identify which events are caused by the handling of a ping packet and which are just caused by the otherwise idle VM. Our measurement system does not support resolving hardware metrics by the process that caused them. Therefore, we focus on the average latency as the only meaningful metric.

Local Networking



(a) AMD average local network latency.

(b) Intel average local network latency.

Figure 6.1: Average local network latency for both AMD and Intel VMs with a packet size of 64 bytes.

Results: Figure 6.1 shows that both SNP and TDX CVMs exhibit a significant latency overhead. When comparing the Figures, the average latency of AMD VMs is higher than that of Intel VMs, but TDX CVMs incur a much larger overhead over non-confidential Intel VMs than SNP CVMs over their non-confidential counterparts. The exact overheads are listed in Table 6.1.

Analysis: As the guest has nothing relevant to compute, the CPU utilization is below 1% for all VMs, suggesting that the VM is only woken up when an ICMP packet arrives. As we know from chapter 2, performing a VMEXIT, and re-entering the VM takes much longer for CVMs, most likely contributing to the increased latency. The fact that TDX VMs have even higher VMEXIT latencies than SNP VMs also explains the bigger network latency increase compared to non-confidential Intel VMs.

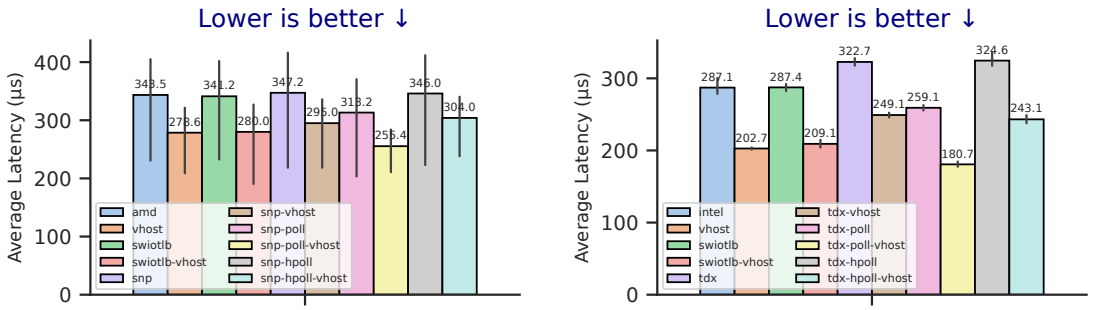
For both AMD and Intel cases, the bounce buffer mechanism shows small latency increases of not more than 5% if vhost is enabled. Without vhost, latencies for both Intel and AMD actually decrease slightly.

Idle polling decreases the latency of TDX VMs significantly and does so as well for SNP if vhost is enabled. We can not observe this positive effect for halt polling, as it has no effect on Intel TDX and actually increases the latency for SNP. This is expected, as the packets arrive in an interval of 1 ms, meaning the default halt polling timer has long run out between arrivals. However, setting the halt poll timer to more than 1 ms would de facto result in an idle polling strategy, as 1 ms is a very long timespan from the CPU's point of view.

Key Takeaways: Both SNP and TDX show significant local network latency overheads. While all AMD VMs show higher absolute latencies, Intel TDX VMs incur higher overhead than SNP VMs. Idle polling reduces CVM latencies significantly, while halt polling has no positive effect.

Type	w/o vhost	w/ vhost	poll w/o vhost	poll w/ vhost
SNP, local	7.16% (+15µs)	12.13% (+17µs)	7.76% (+17µs)	-11.92% (-16µs)
TDX, local	24.96% (+32µs)	44.12% (+30µs)	-13.97% (-18µs)	-37.94% (-26µs)
SNP, remote	0.84% (+3µs)	5.99% (+18µs)	-9.65% (-36µs)	-8.64% (-25µs)
TDX, remote	12.40% (+36µs)	22.89% (+46µs)	-9.75% (-28µs)	-10.85% (-22µs)

Table 6.1: Network latency overhead (average latency) incurred by CVMs over non-confidential VMs with and without vhost and idle polling for local and remote networking. A negative value signifies a performance increase.



(a) AMD average remote network latency.

(b) Intel average remote network latency.

Figure 6.2: Average remote network latency for both AMD and Intel VMs using a packet size of 64 bytes.

Remote Networking

Results: We can see in Figure 6.2 and in Table 6.1 that the latency overheads are generally reduced for remote networking. SNP CVMs are very close to traditional AMD VMs, but they both still show significantly increased latencies compared to Intel VMs, just as with local networking. AMD VMs also show a higher variance in the results that is not present for Intel.

Analysis: The reduced latency overheads compared to local networking can be explained by the higher absolute latency of remote networking. The data has to travel through the physical NIC and therefore has a much longer travel time. At the same time, the overheads introduced by the CVMs stay constant, leading to a smaller relative impact. The absolute added latencies change, but not by much, considering the sizeable general latency increase.

As for local networking, the bounce buffer mechanism, which also causes a constant overhead, has a relatively small effect. It is reduced to a maximum of about 3% with vhost, and the positive effect it had in local networking without it is no longer visible.

Polling effectiveness is also reduced relatively in most cases, except for SNP in conjunction with vhost. Idle polling still yields positive results however, while halt polling again fails to reduce latencies.

Key Takeaways: For remote networking, relative CVM overheads are reduced but still present. The bounce buffer's impact is minimal. Idle polling is reduced in effectiveness but still yields positive results. Halt polling is ineffective.

6.1.2 Network Throughput (TCP)

As this is the most computationally intensive of the networking benchmarks, we use it as an opportunity to present an example for CVM hardware events. Due to the higher computational load, we expect their impact to be the highest in this scenario. Because the performance in this scenario is highly dependent on vhost, we split the analysis into two parts, one without and one with vhost.

Local Networking without Vhost

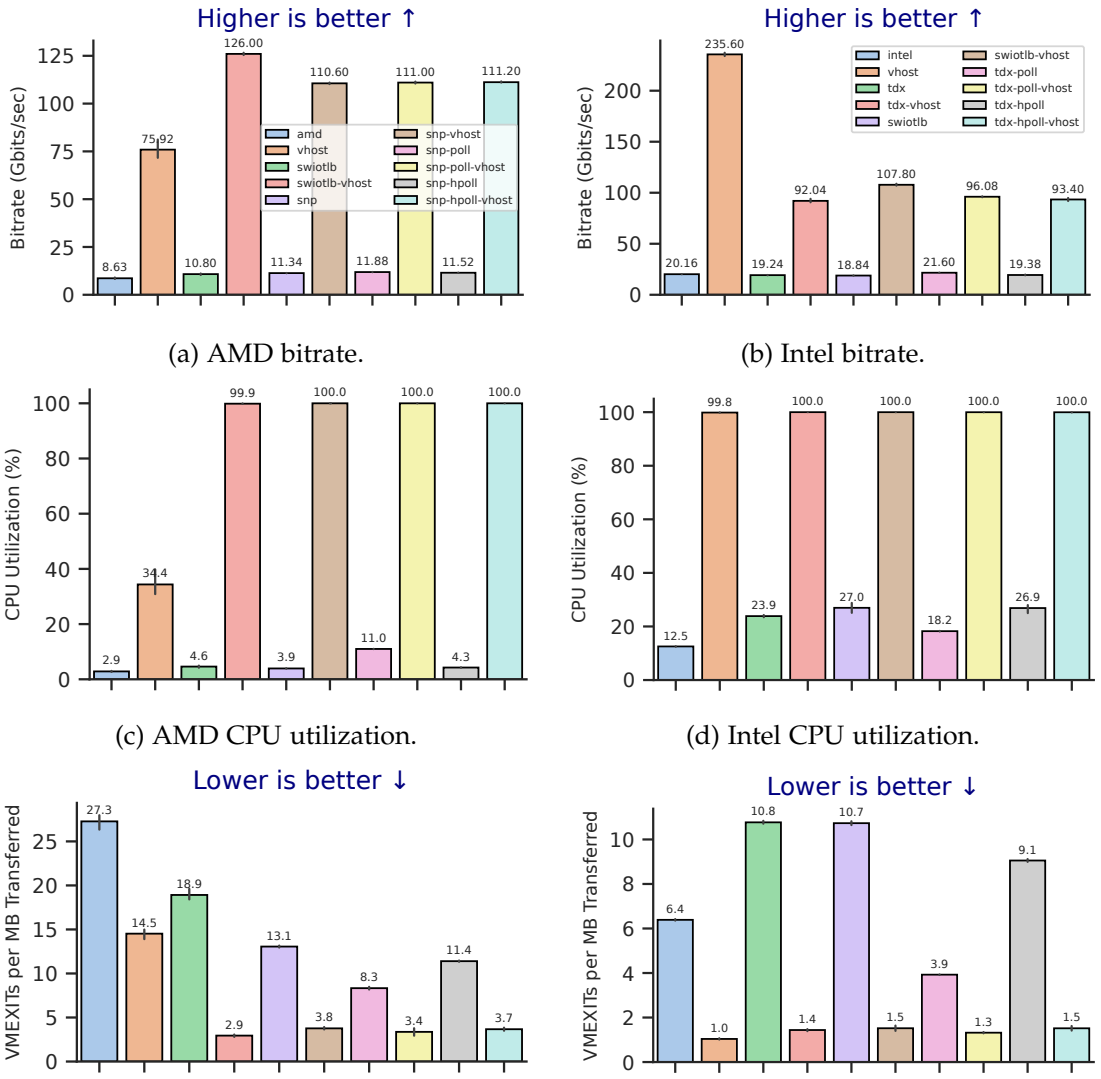
Results: We see in Figures 6.3c and 6.3d that without vhost, both Intel and AMD VMs are not capable of fully utilizing the guest CPU. While Intel achieves utilization levels roughly between 10% and 30%, the CPU of AMD VMs is not even 5% utilized.

Together with the higher CPU utilization, Intel VMs also show much higher bitrates than AMD. The TDX CVM suffers a small overhead compared to the traditional VM but outperforms the bounce buffer VM. SNP VMs also outperform the bounce buffer VMs, and interestingly, they both perform significantly better than the traditional AMD VM. The overhead numbers are listed in Table 6.2.

Analysis: The increased bounce buffer and CVM performance are most likely caused by the additional CPU load generated by the extra data copies and CVM protections, which reduce idle time between packet processing and, therefore, HLT-related VMEXITs. The metric measurement data supports this, showing that while the traditional VM exits about 270,000 times due to HLT instructions, the bounce buffer VM and SNP CVM only exit 195,000 and 123,000 times, respectively. Halt polling further reduces the number of exits recorded to about 70,000, and idle polling, as expected, completely removes HLT-related VMEXITs at the cost of much higher CPU utilization. This becomes visible in an increased performance for both halt and idle polling.

Figures 6.3e and 6.3f show a generally decreased rate of VMEXITs for AMD CVMs, while TDX CVMs exhibit a higher rate than non-confidential Intel VMs. This corresponds nicely to the performance differences observed. In both cases, halt and idle polling reduce the VMEXIT rate significantly, but idle polling has a stronger impact.

The performance counters in 6.4 show that SNP VMs must execute 2.41x more instructions to transfer a kilobyte of data through the network. Interestingly, the bounce buffer mechanism only slightly increases the instructions needed. This means that SNP's security measures cause many additional instructions. However, despite the considerable difference in instructions to be executed between the bounce buffer and the CVM, the performance difference is less extreme, suggesting that most of these instructions are lightweight.



(a) AMD bitrate.

(b) Intel bitrate.

(c) AMD CPU utilization.

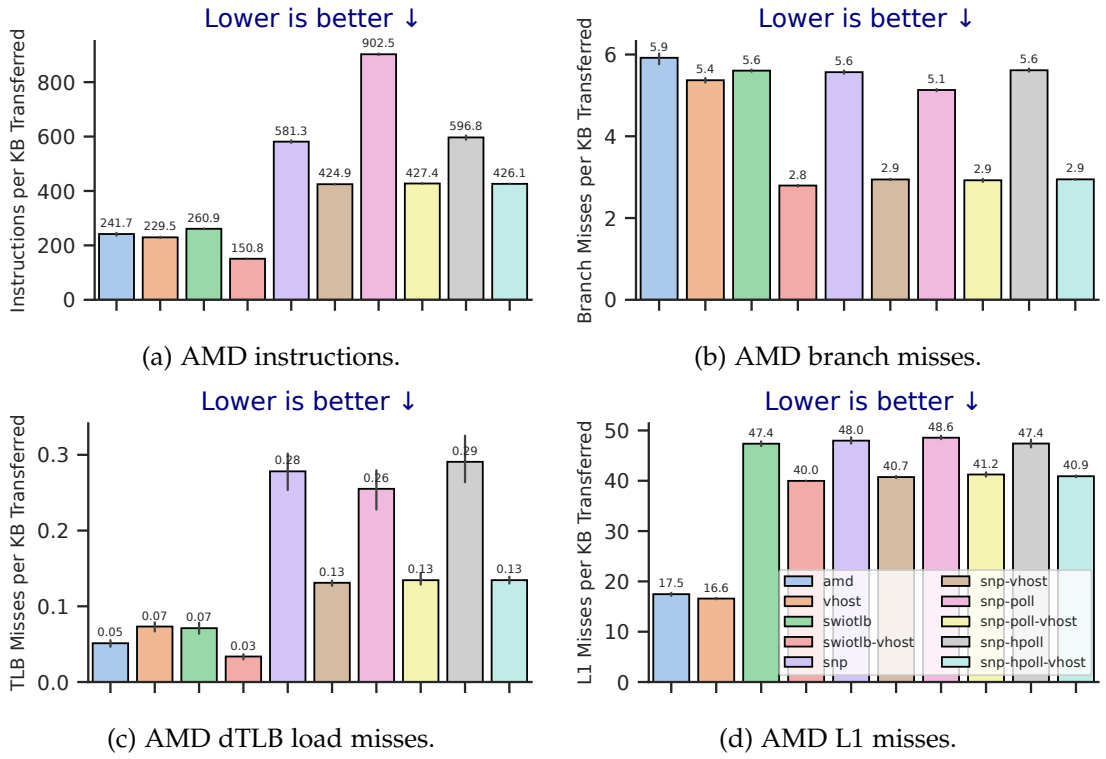
(d) Intel CPU utilization.

(e) AMD VMEXITs per transferred megabyte.

(f) Intel VMEXITs per transferred megabyte.

Figure 6.3: Achieved local TCP network throughput (top), guest CPU utilization (middle), and VMEXIT rate (bottom) for both AMD and Intel VMs.

The same can be said for polling, as both halt and idle polling increase the instruction count and performance. Especially idle polling causes many more instructions to be executed, but as mentioned in chapter 5, these arise from the active waiting loop executed while waiting for work.



(a) AMD instructions.

(b) AMD branch misses.

(c) AMD dTLB load misses.

(d) AMD L1 misses.

Figure 6.4: Guest hardware performance counters for AMD VMs during the local benchmark.

Local Networking with Vhost

Results: Figure 6.3a and 6.3b show a remarkable increase in performance if vhost is enabled. Performance increases by a factor of more than 10x in some cases. For Intel, both the bounce buffer VM and the CVM show large performance overhead of about 54% and 61%. Additionally, all Intel VMs show a fully utilized CPU in Figure 6.3d. In contrast, AMD VMs again show increased performance for the bounce buffer VMs and CVMs. Traditional AMD VM performance is much lower, at only around 35% CPU utilization. Table 6.2 holds the exact overheads.

Analysis: The results show that under high CPU load, the bounce buffer is responsible for most of the overhead encountered by TDX, as the bounce buffer VM's performance is nearly decreased as much as the performance of the CVM.

As the CPU is already fully utilized, the effect polling has on the results is reduced when compared with the results for Intel without vhost. However, both idle and halt polling still lead to a noticeable performance increase and significantly reduce the VMEXIT rate.

The non-confidential AMD VM shows a massively increased VMEXIT rate in Figure 6.3e. The traditional VM exits over 700,000 times due to a HLT instruction, while all other VMs do not do so at all, leading to a large performance difference. While polling lowers the VMEXIT rate and increases the bitrate for SNP VMs, the effects are minimal under full CPU load.

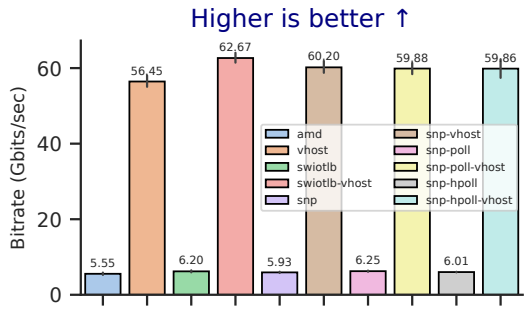
The hardware performance metrics in Figure 6.4 generally show similar trends as without vhost but with decreased rates. The instructions needed by CVMs to transfer one kilobyte are still increased, although by a smaller factor of about 1.85x. However, in this case, the bounce buffer VM has to execute fewer instructions than the traditional VM. This indicates that HLT-related VMEXITs may introduce many instructions that are useless to the workload.

Branch, L1, and TLB miss rates are decreased for CVMs and the bounce buffer VM compared to the non-vhost case, most probably as a direct result of fewer instructions needed per kilobyte of data transfer. The traditional VM does only see a minimal reduction of instructions and therefore L1 and branch misses. TLB misses are actually slightly increased.

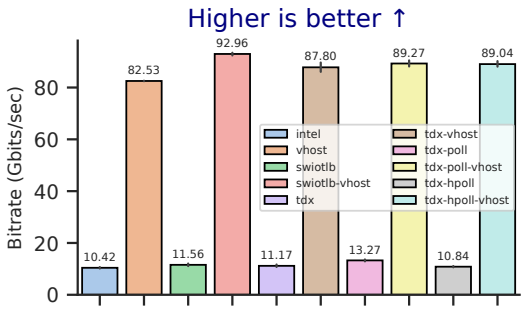
Key Takeaways: AMD and Intel VMs require vhost to utilize the CPU efficiently. TDX shows significant overhead when compared to traditional Intel VMs, with the bounce buffer being responsible for most of it under full CPU load. SNP CVMs perform better than traditional VMs. We do not identify hardware metrics such as branch misses as a reason for CVM overhead. While polling can improve bitrates slightly, its effectiveness is limited in this scenario.

Remote Networking without Vhost

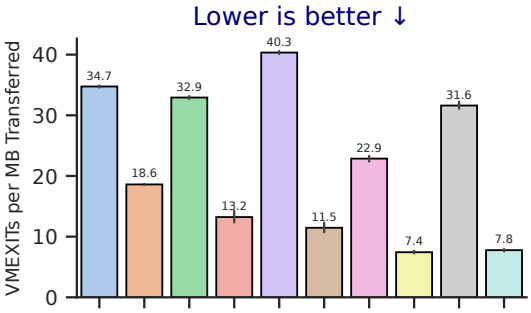
Results: The bitrates shown in Figures 6.5a and 6.5b are significantly decreased compared to the ones reported for local networking. This is expected due to the bottleneck of the physical NIC being introduced. Together with the bitrates, the overhead percentages listed in Table 6.2 are also decreased for both types of CVM.



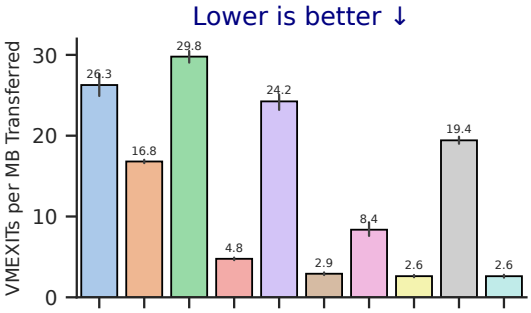
(a) AMD bitrate.



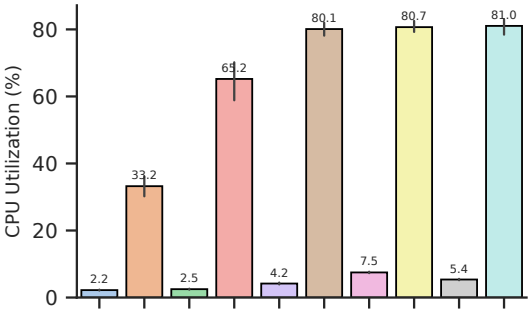
(b) Intel bitrate.



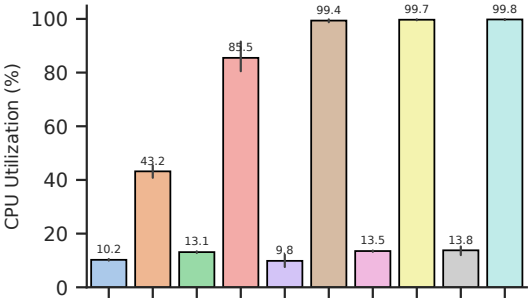
(c) AMD VMEXITs per transferred megabyte.



(d) Intel VMEXITs per transferred megabyte.



(e) AMD CPU utilization.



(f) Intel CPU utilization.

Figure 6.5: Achieved remote TCP network throughput (top), VMEXIT rate (middle), and guest CPU utilization (bottom) for both AMD and Intel VMs.

Type	w/o vhost	w/ vhost	poll w/o vhost	poll w/ vhost
SNP, local	-31.40%	-45.68%	-37.66% (-6.26%)	-46.20% (-0.52%)
TDX, local	4.56%	60.93%	-7.14% (-11.70%)	59.22% (-1.71%)
SNP, remote	-6.85%	-6.64%	-12.61% (-5.76%)	-6.08% (+0.54%)
TDX, remote	-7.20%	-6.39%	-27.35% (-20.15%)	-8.17% (-1.78%)

Table 6.2: The overheads encountered for maximum TCP throughput by the CVMs with and without vhost and idle polling for local and remote networking. For idle polling, the table includes the overhead change compared to a CVM without polling. A negative overhead percentage signifies a performance increase.

Analysis: The trend of the bounce buffer VMs and CVMs outperforming traditional VMs continues for AMD but is now also visible for Intel VMs. Interestingly, the VMEXIT rate shown in Figures 6.5c and 6.5d, is increased for SNP, while TDX still sees a small decrease. A look into the reported VMEXIT reasons for AMD VMs reveals that while the bounce buffer and CVM mechanisms indeed increase the total number of VMEXITs, the HLT-related VMEXITs are still reduced from about 280,000 to 125,000. Instead, the SNP features a much higher amount of interrupt-related VMEXITs, which are classified as AE and do not experience the overhead introduced by the #VC handler.

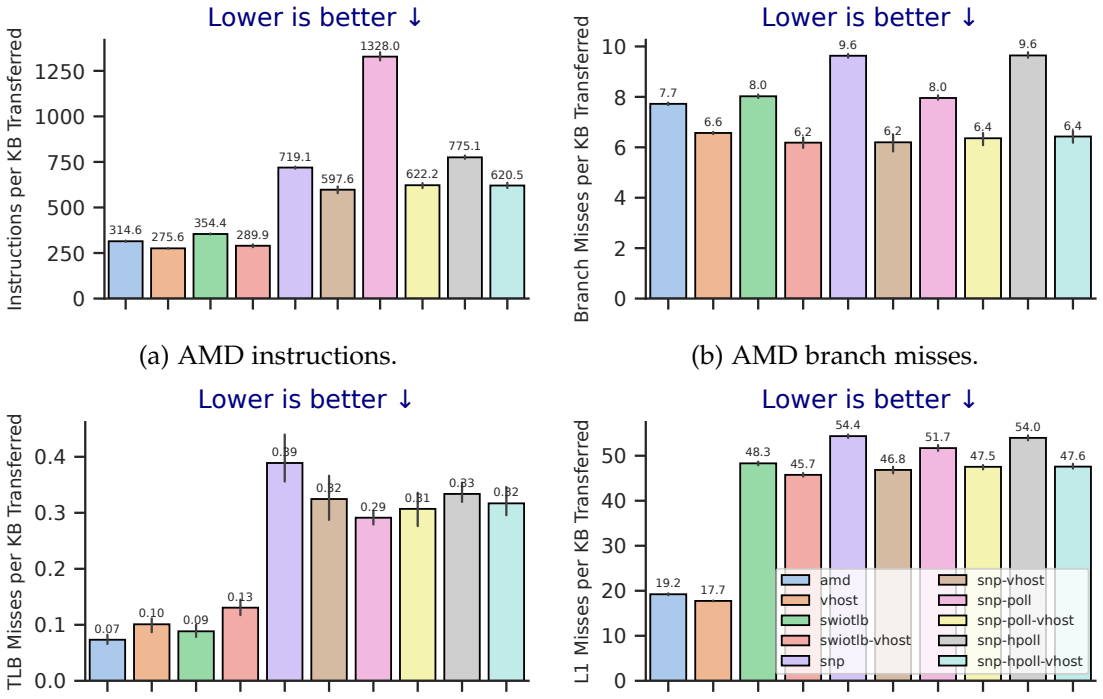
Idle and halt polling both achieve a slight performance increase. Idle polling removes all HLT-related exits again, while halt polling significantly reduces them. Figures 6.5e and 6.5f show that CPU utilization is increased for polling, as is expected. However, the increase is much smaller in the Intel case.

As expected, the hardware performance metrics in Figure 6.6 show similar trends as they did for local networking. Due to the fundamental workload remaining the same, the rates for the respective misses also stay in the same range. However, an increase in instructions per transferred kilobyte is visible and leads to slightly increased miss rates compared to local networking.

Remote Networking with Vhost

Results: Just as without vhost, the bitrates are much lower than for local networking, and the Figures 6.5a and 6.5b show that both Intel and AMD now display increased instead of decreased performance for bounce buffer VMs and CVMs.

Analysis: In Figures 6.5e and 6.5f, Intel TDX achieves both higher bitrates and higher CPU utilization, with its CPUs running near full utilization, while SNP CPUs are around 80% utilized. For both Intel and AMD, the bounce buffer VMs and the traditional VM show much decreased CPU utilization. Despite the lower load on the CPU, the vhost



(a) AMD instructions.

(b) AMD branch misses.

(c) AMD dTLB load misses.

(d) AMD L1 misses.

Figure 6.6: Guest hardware performance counters for AMD VMs during the remote benchmark.

bounce buffer VM achieves the highest bitrate for both AMD and Intel, and while the traditional VMs use only about half of the CPU resources of the bounce buffer VMs, they still perform only about 10% worse. Again, this indicates that under high CPU load, the bounce buffer mechanism severely limits data throughput.

As for local networking, the traditional VMs show a very high VMEXIT rate in Figures 6.5c and 6.5d if vhost is enabled. For AMD, this is again caused by a high rate of HLT instructions. While the traditional VM exits about 670,000 times due to being idle, the SNP and bounce buffer VM only do so about 160,000 and 340,000 times, respectively.

Both polling forms reduce the VMEXITs further and slightly increase Intel TDX performance. AMD SEV-SNP performance, however, actually decreases, proving that polling can, in some scenarios, be detrimental to performance.

As without vhost, we observe that the fundamental trends for the hardware metrics in Figure 6.6 remain the same. One noticeable difference, however, is the decrease

of vhost's effectiveness. The factor by which it lowers the miss rates is decreased, especially for branch and TLB misses. Other than that, the data shows nothing that would indicate a noticeable CVM disadvantage.

Key Takeaways: Bitrates are significantly decreased compared to local networking. Intel continues to achieve higher performance. Intel and AMD CVMs have higher performance and CPU utilization than non-confidential VMs. We do not identify hardware metrics, such as branch misses, as a reason for CVM overhead. Polling effects are minimal and can decrease performance in some cases.

6.1.3 Network Throughput (UDP)

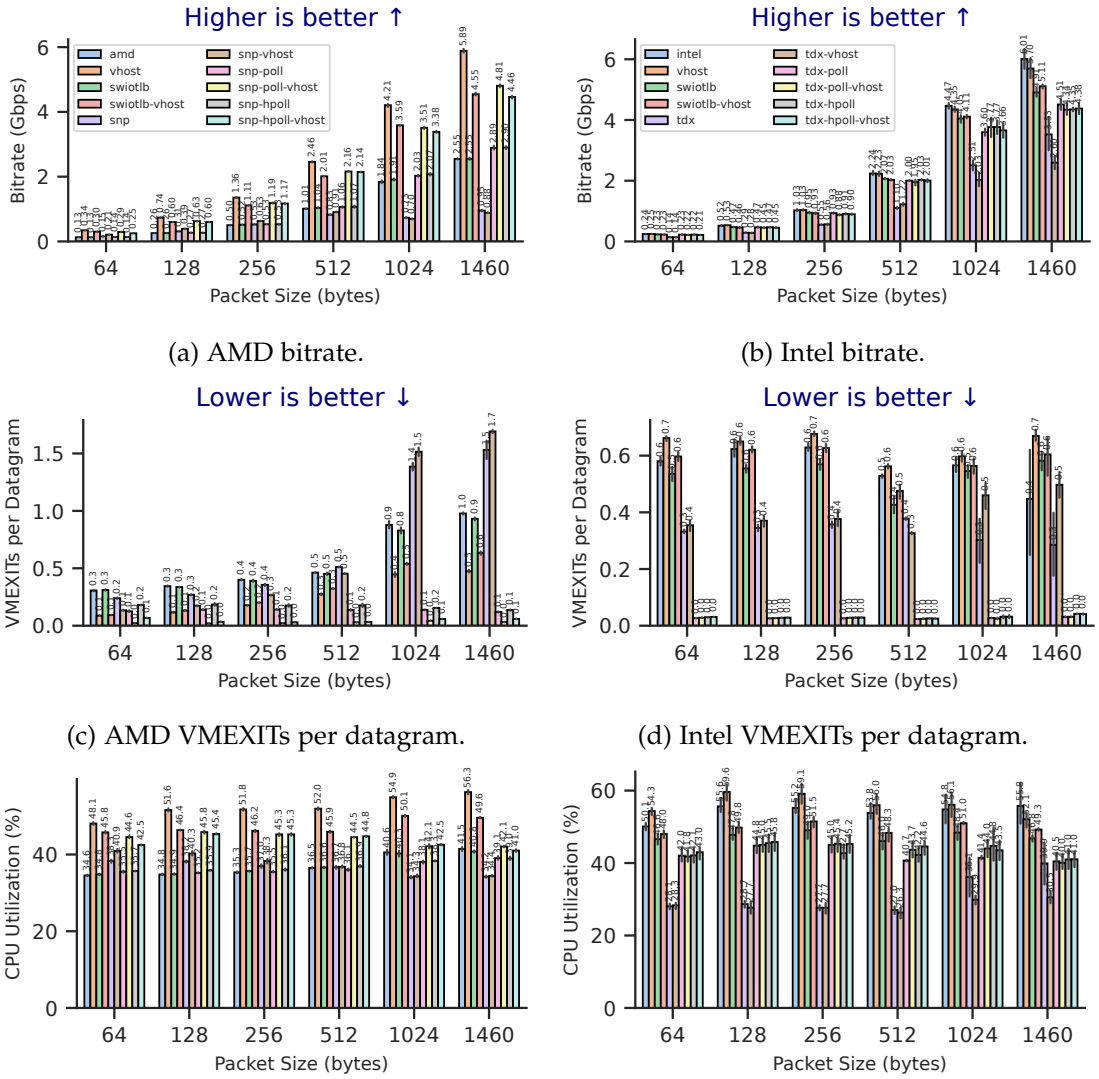
While this scenario is less computationally intensive than the TCP one, it still significantly uses the CPU. In addition to the achieved bitrate, we look again at the guest CPU utilization and the VMEXIT rate, which we norm by the number of handled UDP datagrams.

Local Networking

Results: In this scenario, where the average CPU utilization never exceeds 60% for both AMD and Intel VMs, we observe severe performance degradation, especially for SNP CVMs. Figure 6.7a shows that the SNP overhead increases with packet size, culminating in a vast 85.1% overhead for a packet size of 1460 bytes and vhost enabled. On the contrary, Intel CVM overheads do not directly increase with packet size and lie roughly between 40% and 55%, making them not quite as extreme as in the AMD case. In addition to showing lower CVM overhead, Intel VMs achieve higher bitrates in most cases, especially if vhost is disabled. More overhead numbers are listed in Table 6.3.

Analysis: The reason for this massive performance degradation seems to be an excessive number of VMEXITs encountered per UDP datagram shown by Figure 6.7c. For the largest two packet sizes, where the overhead is the most extreme, the amount of VMEXITs needed to handle one datagram is also massively increased.

The most frequent reasons for these VMEXITs are reported to be MMIO accesses to the address range from *0xfee00300* to *0xfee00380*, which corresponds to local APIC registers. More specifically, the *Interrupt Command Register* and the *Initial Count Register* at *0xfee00300* and *0xfee00380* [20] are accessed with very high frequency. While the SNP VMs access these memory locations millions of times during the benchmark, the non-confidential AMD VMs do not do so more than a handful of times. At the same time, the traditional VMs access the *IA32_TSC_DEADLINE* MSR register, located at



(a) AMD bitrate.

(b) Intel bitrate.

(c) AMD VMEXITs per datagram.

(d) Intel VMEXITs per datagram.

(e) AMD CPU utilization.

(f) Intel CPU utilization.

Figure 6.7: The achieved local UDP bitrates (top), VMEXITs per handled datagram (middle) and CPU utilization during the benchmarks (bottom) for both AMD and Intel.

address *0x6e0* millions of times, while there is not a single access to this register from SNP VMs.

All these registers are used to set timers inside a CPU core's local APIC. The timers schedule timed interrupts and can operate in two modes, "one-shot" and "periodic". In one-shot mode, the initial counter is set, and the value is copied to the *Current Count Register*. It is then decremented until it reaches 0, and an interrupt is generated. It remains at 0 until it is manually set again. In periodic mode, the current count is reloaded from the initial count once it reaches 0. Additionally, newer hardware supports a third mode called TSC-Deadline mode that uses the CPU's Time Stamp Counter (TSC). It is a more precise version of one-shot mode that does not count down from an initial value but instead uses a timestamp, settable in the *IA32_TSC_DEADLINE* MSR register, to schedule an interrupt for a fixed absolute time. It requires less overhead to access and provides better time resolution, based on the global CPU time and not the local APIC time [20].

SNP guests do not support this third mode of operation currently, leading to the legacy modes being used, resulting in a high number of MMIO operations instead of MSR accesses alongside less precise interrupt timers. As UDP is a connectionless protocol, setting interrupts more precisely and with less overhead is vital to minimize packet loss and maximize bitrate. Therefore, the absence of TSC-Deadline mode plays an important role in the high overheads observed.

Intel TDX supports it, as TDX CVMs and traditional Intel VMs feature many accesses to the *IA32_TSC_DEADLINE* register. In addition, Intel TDX shows a large amount of VMEXITs due to accesses to the *IA32_X2APIC_ICR* register, located at address *0x830*. This MSR register corresponds to the *Interrupt Command Register* that the SNP VM accesses using the legacy MMIO method.

However, Figure 6.7d shows that contrary to AMD CVMs, the Intel CVMs do not show an increased rate of VMEXITs per datagram handled, but rather a decreased one. While the amount of VMEXITs needed to handle larger datagram sizes is increased for AMD VMs, Intel VMs require a similar amount of VMEXITs regardless of packet size.

Interestingly, vhost increases the number of VMEXITs necessary per datagram for both SNP and TDX for large packet sizes, leading to worse CVM performance if it is enabled. We observe this for both Intel and AMD, but while AMD's traditional VMs benefit significantly from vhost, Intel's do not.

We can see in Figure 6.7f that the CPU utilization for TDX VMs is much lower across packet sizes. It is decreased by over 50% in some cases, showing a strong relation to the reduced bitrates. Due to the restrictions introduced by TDX regarding the metrics that the guest can trace, it is hard to pinpoint what exactly causes this issue.

The bounce buffer VM behaves differently for Intel and AMD. In the AMD case, it does not show any overhead if vhost is disabled but causes performance to decrease by

Type	w/o vhost	w/ vhost	poll w/o vhost	poll w/ vhost
SNP, local	62.75%	85.06%	-13.33% (-76.08%)	18.34% (-66.72%)
TDX, local	41.26%	54.39%	24.98% (-16.28%)	23.86% (-30.43%)
SNP, remote	33.01%	71.40%	-13.11% (-46.12%)	-0.67% (-72.07%)
TDX, remote	48.60%	47.11%	7.44% (-41.16%)	7.78% (-39.33%)

Table 6.3: The overheads encountered by the CVMs for maximum UDP throughput (packet size 1460) with and without vhost and idle polling for local and remote networking. The table includes the overhead change for idle polling compared to a CVM without polling. A negative overhead percentage signifies a performance increase.

up to 23% otherwise. For Intel, the overhead of the bounce buffer mechanism is slightly smaller at up to about 18% but is present regardless of whether vhost is used.

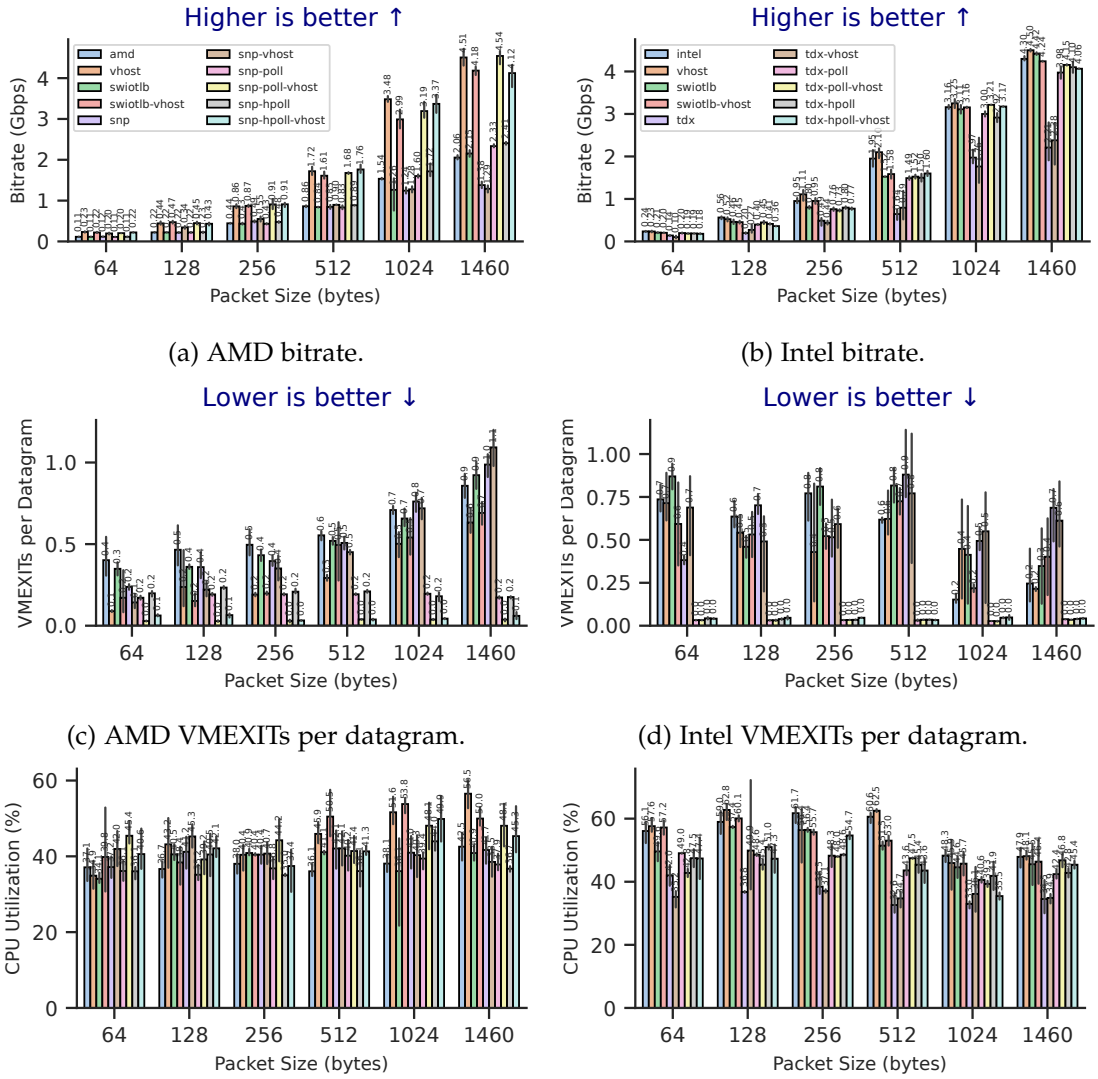
Polling has remarkable effects in this scenario, especially for AMD. Here, both idle and halt polling only slightly increase the CPU utilization but massively increase the performance for both vhost and non-vhost cases. If vhost is disabled, the performance is brought on par with the traditional VM; if not, the performance level of the bounce buffer VM is reached.

For Intel TDX, both polling methods significantly improve performance, but they do not achieve the same level as the non-confidential VM or the bounce buffer VM, still leaving an overhead of about 30% for the largest packet size. Other than for AMD, it does also increase CPU utilization, especially for lower packet sizes.

Figures 6.7c and 6.7d show that the VMEXIT rate per datagram is reduced to nearly 0 if any form of polling is enabled. Despite removing all HLT VMEXITs, the MMIO-related VMEXITs for the SNP CVMs are also decreased massively from several million to under 200,000.

As we observe for TCP throughput, the measured hardware performance metrics do not reveal anything that would indicate a performance decrease for SNP CVMs. Therefore, no additional graphs are shown here.

Key Takeaways: Both SNP and TDX show extreme overhead in this scenario. SNP does not support scheduling interrupts in TSC-Deadline mode and needs to resort to less precise legacy MMIO APIC programming. TDX does support TSC-Deadline mode. Polling radically increases performance back to the levels that would be expected.



(a) AMD bitrate.

(b) Intel bitrate.

(c) AMD VMEXITs per datagram.

(d) Intel VMEXITs per datagram.

(e) AMD CPU utilization.

(f) Intel CPU utilization.

Figure 6.8: The achieved UDP bitrates (top), VMEXITs per handled datagram (middle) and CPU utilization (bottom) for AMD and Intel VMs using remote networking.

Remote Networking

Results: For remote networking, both CVM types still show a substantial overhead. However, we can see in Figures 6.8a and 6.8b, as well as in Table 6.3, that while AMD VMs continue the trend of shrinking overheads in remote networking scenarios with lower bitrates, the TDX overhead actually increases when compared to local networking if vhost is disabled. In general, the Figures 6.8a, 6.8c and 6.8e show very high variance in the results for AMD VMs, although the trends remain similar to local networking.

Analysis: SNP overheads still increase with packet size, and the number of VMEXITs per datagram is still increased for larger packet sizes, although much less so than for local networking. The lower rate of increase explains the noticeably lower overhead percentages. As expected, the increase is again caused by frequent MMIO accesses to the same local APIC addresses seen in local networking, and MSR accesses are still absent from SNP machines.

Contrary to what we observe in local networking, Figure 6.8d also shows increased VMEXITs per handled datagram for TDX. While the rate reported for local measurements is between 0.3 and 0.5 VMEXITs per datagram, now it reaches as high as 1. This is most probably the reason why the TDX overheads increased compared to local networking, as Figure 6.8f shows a small increase in CPU utilization compared to Figure 6.7f. Due to not being able to resolve TDCALLs further, it is hard to pinpoint what might be the cause for this higher rate of VMEXITs.

Vhost is again significantly improving the performance of traditional AMD VMs, but in most other cases, there is no clear improvement, with the minimal performance effects being both negative and positive, depending on the specific situation.

The lower bitrates, as expected, decrease the bounce buffer overhead. As with the TCP throughput, the bounce buffer increases performance without vhost in some scenarios. However, if vhost is enabled, it still shows a tangible overhead for both Intel and AMD in most cases.

For both Intel and AMD, polling again has a very positive effect where large overheads are encountered without it. However, Table 6.3 shows that when using remote networking, idle polling is less effective for SNP and more effective for TDX than it is with local networking. This corresponds nicely to the decreased VMEXIT rate seen with SNP and the increased VMEXIT rate visible for TDX in the remote case. Regardless of this, both forms of polling still reduce VMEXIT rates to very low levels for both Intel and AMD. Polling is especially effective in combination with vhost and, if enabled, can elevate CVM performance to traditional VM levels. Depending on the packet size, either idle or halt polling is more effective.

Key Takeaways: Remote network UDP throughput is still severely decreased for both TDX and SNP. VMEXIT behavior is again the primary reason and the bounce buffer has limited effects. Polling remains very effective for remote networking, especially for TDX.

6.1.4 Database Server Performance

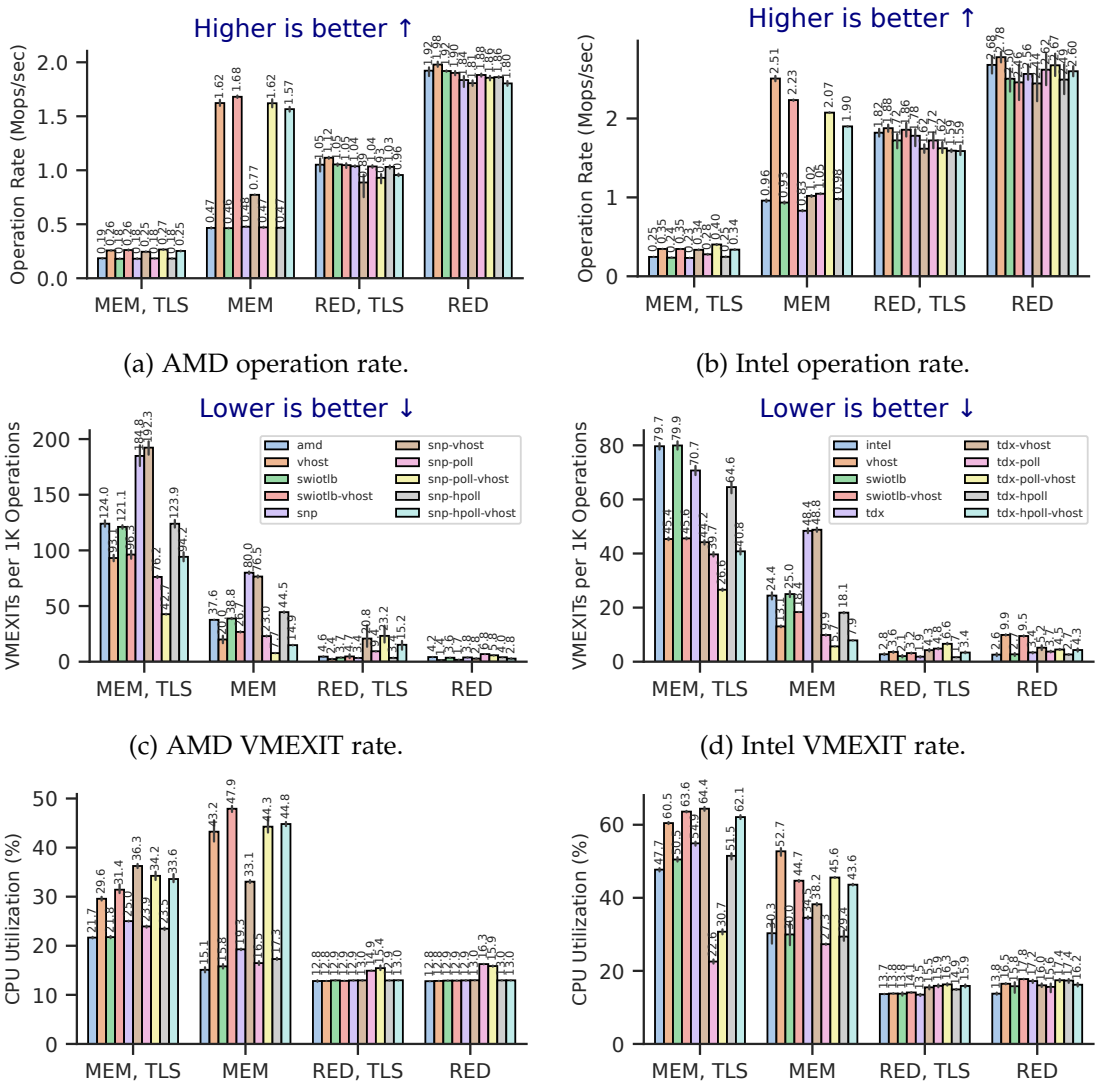
Here, we examine two different servers with the same benchmark. We analyze each server's performance in terms of operation rate, VMEXIT rate, and CPU utilization. It is interesting to use two database server implementations, as they have different characteristics and allow us to observe a similar workload in two different settings. The most striking difference is that the *redis* server only uses a single CPU core, while the *memcached* server is multithreaded.

Local Networking with Redis

Results: Both Intel and AMD CVMs show a decreased rate of operations per second in Figures 6.9a and 6.9b. If vhost is not enabled, it is relatively small at not even 5% both with and without TLS but increases to about 21% and 14% for SNP and TDX, respectively, if vhost is combined with TLS, as is listed in Table 6.4. Generally, overheads stay in an expected range, and there is no extreme performance degradation as seen in previous benchmarks.

Analysis: We can observe in Figure 6.9e and 6.9f that the single-threaded *redis* server fully utilizes one CPU of the eight cores with above 12.5% CPU utilization for both Intel and AMD. It does this regardless of whether TLS is active.

Due to this high load, the VMEXIT rate shown in Figures 6.9c and 6.9d is relatively low. Polling increases it in some cases because it is enabled on all cores and the server only utilizes one. This leads to a higher rate of interrupt-related VMEXITs due to the host requiring CPU resources. In general, VMEXITs have a limited impact on the performance difference in this case. However, there is an outlier in the combination of TLS and vhost for SNP that shows a drastically increased rate of VMEXITs and the highest overhead. This is caused by an increased rate of HLT-related VMEXITs, from about 20,000 for the traditional VM to nearly 220,000 for the CVM. The increased HLT rate is accompanied by a high frequency of MMIO accesses to the local APIC registers. The SNP VM without vhost also shows an increased rate of HLT exits at about 120,000 measured; it does not show a high rate of MMIO exits.



(a) AMD operation rate.

(b) Intel operation rate.

(c) AMD VMEXIT rate.

(d) Intel VMEXIT rate.

(e) AMD CPU utilization.

(f) Intel CPU utilization.

Figure 6.9: In-memory database server performance (top), VMEXIT rate (middle) and CPU utilization (bottom) for both Intel and AMD. The client is running on the same host machine as the server VM. The *redis* server is labelled as "RED", while the *memcached* server is labelled as "MEM".

This is interesting, as we see the opposite of the earlier effect, where the CVM mechanisms decreased the HLT VMEXIT rate. As this only happens when vhost is enabled together with TLS, it may be an outlier caused by some mechanisms not playing well together.

While TDX also shows low performance for a combination of TLS and vhost, the VMEXIT rate only slightly increases in this case. In general, vhost introduces a higher variance for both SNP and TDX.

AMD's bounce buffer overhead is minimal, especially if TLS is enabled. However, for Intel, the bounce buffer VM shows an overhead of around 12% if TLS is disabled. This again illustrates that Intel's performance is generally more affected by the bounce buffer.

As is expected, when the CPU operates at high utilization and the rate of VMEXITs is relatively low, polling effectiveness is not very high. The only case in which it significantly improves performance is for Intel TDX without TLS, where it nets an overhead decrease of about 8%. As idle polling is enabled across all cores, it does not decrease the VMEXIT rate despite eliminating the HLT-related VMEXITs. Instead, it introduces many interrupt exits in their place. As expected, halt polling does not do this to the same extent but also brings even less performance improvement.

Local Networking with Memcached

Results: Despite being multithreaded and exhibiting higher CPU utilization, Figures 6.9a and 6.9b show that the *memcached* server performs worse than the *redis* server in all cases. If TLS is enabled, neither SNP nor TDX show any significant overhead. If it is disabled, however, the picture changes. Without vhost, the overhead stays inside an expected range for TDX at about 14%, and SNP shows no overhead. If vhost is enabled, Table 6.4 shows that the overhead drastically increases for both Intel and AMD to 52% and 59%.

Analysis: In this case, we can observe an increased rate of VMEXITs for both AMD and Intel CVMs. Interestingly, for AMD, the performance of the CVM without vhost is not reduced, and polling has no impact on performance despite drastically lowering the VMEXIT rate. This suggests that performance without vhost is bottlenecked by the context switching between QEMU and the kernel. If this bottleneck is reduced by vhost, the CPU can be used much more effectively, as Figures 6.9e and 6.9f show, and the performance is bottlenecked by a high VMEXIT rate.

Polling significantly improves CVM performance if vhost is enabled for both Intel and AMD. For AMD, both halt and idle polling elevate the performance to traditional

VM levels. The results for TDX are also excellent, but there remains an overhead of 18% and 25% for idle and halt polling, respectively.

The bounce buffer only affects performance in a relevant way if vhost is enabled. For TDX, it creates an overhead of about 11% and slightly increases AMD performance with a negative overhead of -4%. Similar to previous results, this is again caused by a lower rate of HLT-related VMEXITs experienced by the bounce buffer VM.

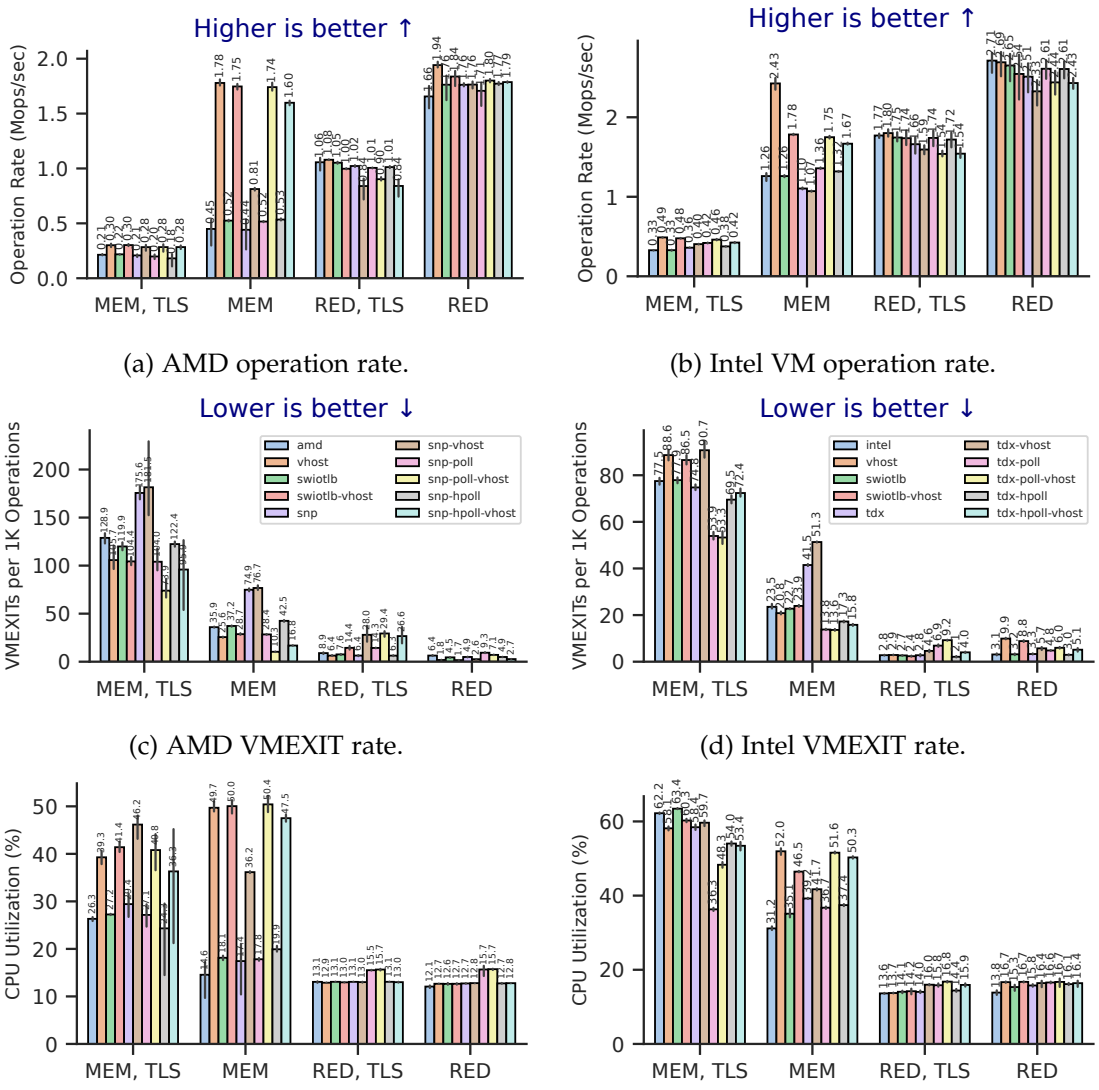
Key Takeaways: For the *redis* server, the overheads stay in the expected range except for one outlier. Its performance is bottlenecked by the CPU, and polling has little effect. Both TDX and SNP show very little overhead for the *memcached* server if TLS is enabled. If vhost is used without TLS, there are large overheads that polling can significantly reduce.

Type	RED	RED w/ poll	MEM	MEM w/ poll
SNP, local	8.59%	6.06% (-2.53%)	52.47%	0.00% (-52.47%)
SNP, local, TLS	20.54%	16.96% (-3.58%)	3.85%	-3.85% (-7.70%)
TDX, local	12.23%	3.96% (-8.27%)	59.36%	17.53% (-41.83%)
TDX, local, TLS	13.83%	13.83% (-0.00%)	2.85%	-14.29% (-17.14%)
SNP, remote	9.28%	7.22% (-2.06%)	54.49%	2.25% (-52.24%)
SNP, remote, TLS	22.22%	16.67% (-5.55%)	6.67%	6.67% (-0.00%)
TDX, remote	13.38%	9.29% (-4.09%)	55.97%	27.98% (-27.99%)
TDX, remote, TLS	11.67%	14.44% (+2.77%)	18.37%	6.12% (-12.25%)

Table 6.4: The overheads encountered by CVMs on the operation rate with vhost enabled for both *memcached* and *redis* servers running inside AMD and Intel VMs. Includes numbers for local and remote networking with and without TLS.

Remote Networking with Redis

The remote networking results shown in Figure 6.10 are very similar to the local networking results, just with very slightly reduced bitrates. This proves that the performance of the *redis* server is not bottlenecked by network performance but rather by the CPU. Table 6.4 shows that the remote overheads are mostly very close to the ones encountered with local networking. As all major trends stay the same, we do not discuss them here again.



(a) AMD operation rate.

(b) Intel VM operation rate.

(c) AMD VMEXIT rate.

(d) Intel VMEXIT rate.

(e) AMD CPU utilization.

(f) Intel CPU utilization.

Figure 6.10: In-memory database server performance (top), VMEXIT rate (middle) and CPU utilization (bottom) for both Intel and AMD. The client is running on a different host machine as the server VM. The *redis* server is labelled as "RED", while the *memcached* server is labelled as "MEM".

Remote Networking with Memcached

Just as with the *redis* server, the results are very similar to local networking, but we observe a small increase in operation rates in many cases. This is an indication, that the *memcached* server also is not bottlenecked by network throughput in those cases. However, the highest performing VMs in local networking, Intel with vhost and without TLS, show a significant performance decrease, suggesting that the *memcached* server eventually runs into a network bottleneck.

Key Takeaways: The fundamental trends remain the same as in local networking. The achieved operation rates are not much affected by remote networking in most cases.

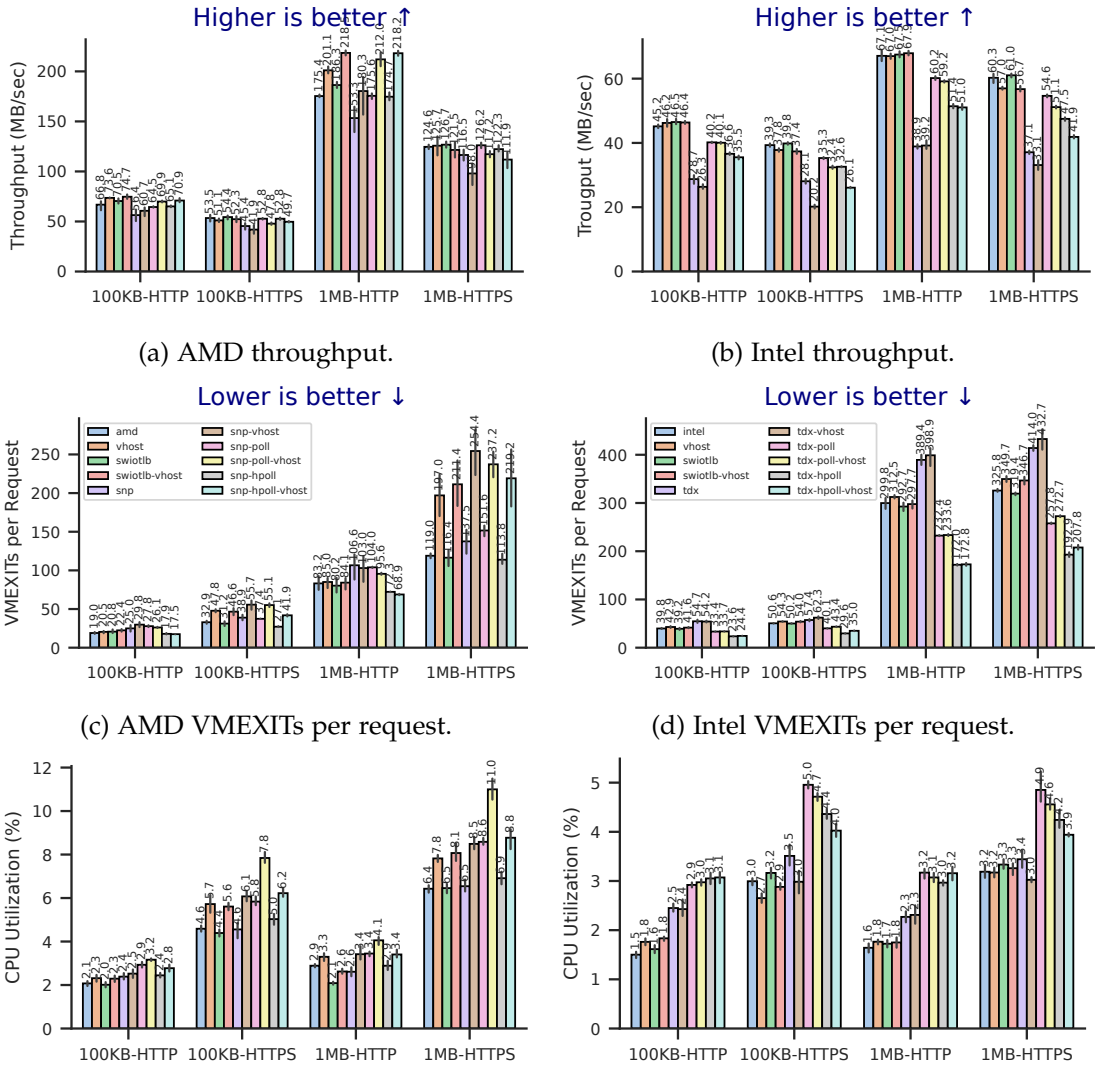
6.1.5 Web Server Performance

As the *nginx* server hosting the files for this scenario is very efficient and can handle many connections simultaneously, it mostly only uses one CPU core. Due to its design, our measurement system does not support metrics capture on only specific CPU cores, which is why the metrics presented in the following are, as always, measured across all vCPUs. This results in some distortions, e.g., for the VMEXIT rate, especially when polling is enabled. Polling is active for each vCPU, leading to an increased rate of external interrupts from the hypervisor. This might make it appear that polling increases the VMEXIT rate in some cases while actually reducing HLT-related exits on the server's performance-critical core.

Local Networking with HTTP

Results: For the HTTP protocol, AMD SNP shows up to about 14% overhead for the 100 KB file if vhost is enabled and 12% if it is not. Figure 6.11a shows this overhead decreasing to about 12% and 10% for the 1 MB file size. For the first time, Intel shows significantly lower bitrates than AMD in Figure 6.11b. Along with the decreased performance Intel also shows much larger CVM overheads. As in most other benchmarks, Intel VMs can not take advantage of vhost in this scenario and perform worse when it is enabled. The overheads in Table 6.5 also show a severe TDX performance degradation of about 41%.

Analysis: We observe in Figures 6.11e and 6.11f that for both AMD and Intel, the CPU utilization stays below 5% if the HTTP protocol is used. However, while the SNP



(a) AMD throughput.

(b) Intel throughput.

(c) AMD VMEXITs per request.

(d) Intel VMEXITs per request.

(e) AMD CPU utilization.

(f) Intel CPU utilization.

Figure 6.11: The HTTP/HTTPS throughput (top), the VMEXIT rate per request (middle), and the guest CPU utilization (bottom) for both AMD and Intel using local networking.

CVMs and traditional AMD VMs show a similar utilization percentage, TDX stresses the CPU slightly more than non-confidential Intel VMs.

Both AMD and Intel CVMs show an increased rate of VMEXITs per HTTP/HTTPS request in Figures 6.11c and 6.11d, but the increase is much larger for TDX, especially for the larger of the two file sizes. Intel VMs also show a generally much higher VMEXIT rate per request in accordance with the decreased throughput. For SNP VMs, the increase is again caused by a high amount of MMIO accesses to the previously mentioned local APIC registers. As we can not see the reasons for the TDX VMEXITs, it is hard to pinpoint what causes their increased rate. Regardless, this result again shows that the performance of TDX VMs suffers heavily if they have a high rate of VMEXITs.

As previously seen, the bounce buffer mechanism has minimal impact on performance but increases it in some cases. This effect is especially prominent for AMD bounce buffer VMs and the 1 MB file size but is also present for Intel.

Polling has a positive effect on performance for both Intel and AMD. For the small file size, idle polling reduces the SNP overhead to about 5% with and 7% without vhost, while halt polling interestingly achieves even better results, especially for the larger file size. Here, it shows a negative overhead of about -5% if vhost is enabled and, just like idle polling, brings performance on par with the traditional VM if it is not. The larger TDX overheads are reduced significantly by both idle and halt polling, but idle polling is more effective. The overheads for the small file size are reduced to about 11% without, and 13% with vhost. For the larger file, we observe a similar reduction to about 10% and 12%. Polling does not change the fact that Intel VMs perform worse if vhost is enabled in most cases. For both Intel and AMD, polling causes CPU utilization to rise visibly, but it still stays below 5% in the HTTP case. We see again that in cases of extreme CVM overhead, polling can significantly improve results without excessively increasing resource usage.

Local Networking with HTTPS

Results: When we examine the HTTPS protocol, some trends visibly change. We observe that the throughput is lowered across Figures 6.11a and 6.11b, which is expected due to the additional overhead introduced by TLS. At the same time, Table 6.5 shows that the CVM overhead more than doubles for SNP to about 22% for the larger file size. While the TDX overhead shrinks if vhost is disabled, it stays nearly the same if it is turned on. As for database server performance, vhost does not play well with TLS encryption.

Analysis: We generally see higher VMEXIT rates when HTTPS is used for both Intel and AMD. The reasons for these VMEXITs, however, remain the same. As is expected,

Type	HTTP	HTTP w/ poll	HTTPS	HTTPS w/ poll
SNP, local	10.34%	-5.42% (-15.76%)	22.04%	6.76% (-15.28%)
TDX, local	41.49%	11.64% (-29.85%)	41.93%	10.33% (-31.60%)
SNP, remote	5.92%	-3.12% (-9.04%)	10.65%	4.64% (-6.01%)
TDX, remote	38.94%	11.60% (-27.33%)	46.61%	19.93% (-26.69%)

Table 6.5: The overheads encountered by the CVMs for HTTP and HTTPS throughput (1 MB, vhost enabled) with and without idle polling for local and remote networking. The table includes the overhead change for idle polling compared to a CVM without polling. A negative overhead percentage signifies a performance increase.

due to additional steps introduced by the TLS encryption process, the amount of hypervisor calls is increased for AMD VMs. This increased dependence on hypervisor services is most likely the reason for the increased rates of Intel VMs as well.

CPU utilization significantly increases when HTTPS is used, but it still remains very low. This time, AMD VMs show a higher utilization but still stay much below 10% in most cases.

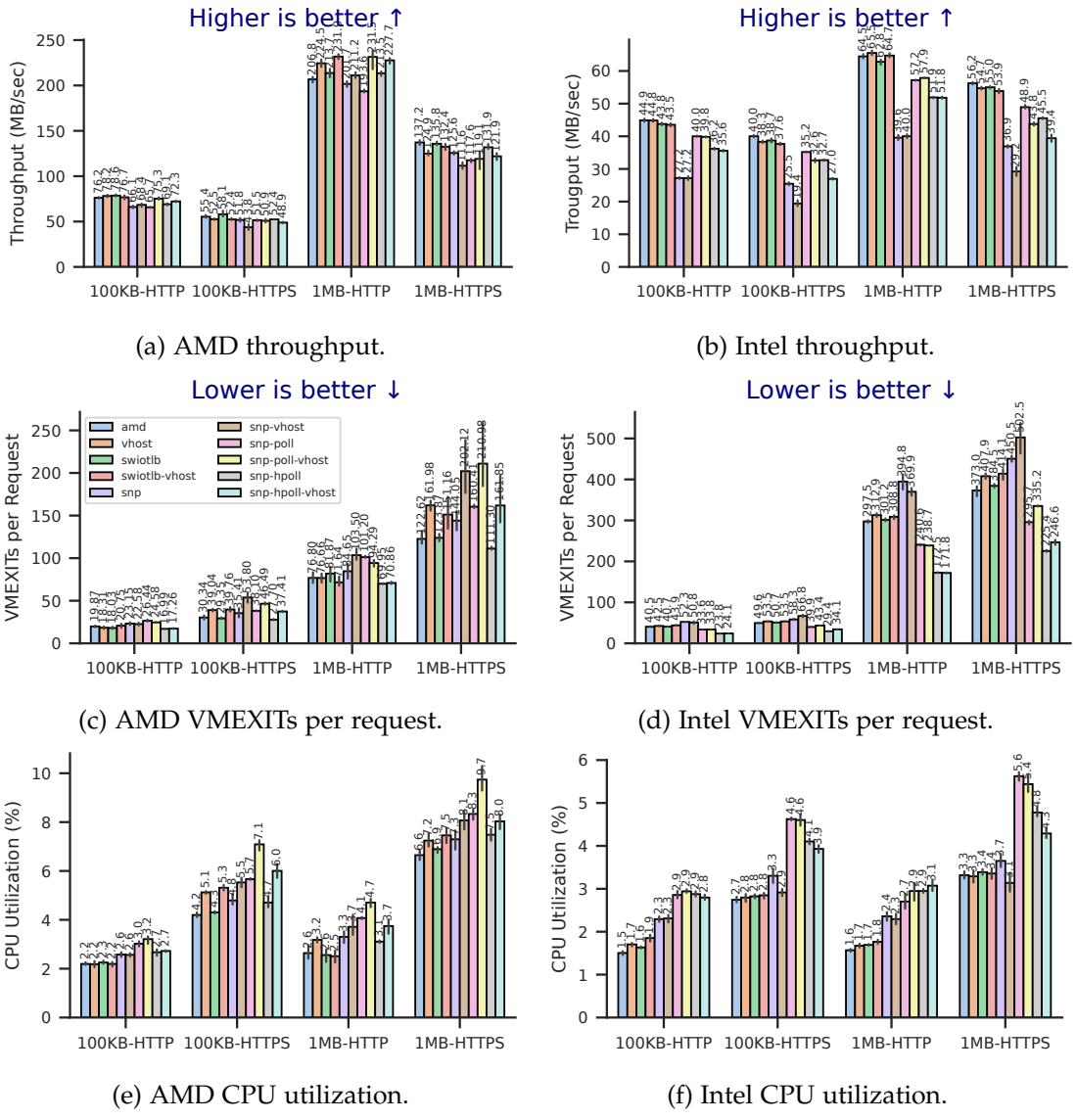
The bounce buffer behaves very similarly as it did for HTTP, increasing the performance in some scenarios but mainly affecting it very little.

The effects of polling are the same as for the HTTP protocol. Both idle and halt polling significantly improve CVM performance and decrease VMEXIT rates. As Intel TDX is again the CVM technology that is more severely affected by VMEXITs, it also benefits more from polling.

Key Takeaways: AMD outperforms Intel on HTTP/HTTPS throughput. SNP overheads are in an expected range, but TDX shows extreme overheads. The bounce buffer plays a minor role due to low CPU utilization. TLS and the vhost protocol perform poorly in combination. Polling is very effective for both CVM types.

Remote Networking with HTTP

Results: For the HTTP protocol, we see two different behaviors for AMD and Intel in Figures 6.12a and 6.12b. AMD throughput increases with remote networking, and Table 6.5 shows a reduction of the overheads by about half. On the other hand, Intel does not feature increased throughput, and the overheads show no clear decreasing trend. AMD again achieves much better performance than Intel.



(a) AMD throughput.

(b) Intel throughput.

(c) AMD VMEXITs per request.

(d) Intel VMEXITs per request.

(e) AMD CPU utilization.

(f) Intel CPU utilization.

Figure 6.12: The HTTP/HTTPS throughput (top), VMEXIT rate (middle), and the guest CPU utilization for both AMD and Intel using remote networking.

Analysis: The CPU utilization does not change much from local networking for both Intel and AMD, still staying below 5% in all HTTP cases. Again, SNP does not cause additional CPU load, while TDX does.

We observe that combined with the increased performance of AMD VMs, their VMEXIT rate per request significantly decreases. The VMEXIT data for this benchmark shows that remote networking significantly reduces HLT-related exits for all AMD VMs. At the same time, the VMEXIT rates of Intel VMs do not change much and stay much higher than those of AMD VMs.

As we observe in all low CPU utilization cases, the bounce buffer has a very limited impact. Again, it shows slightly increased performance for AMD VMs by lowering the VMEXIT rate per request, but it does not do so for Intel.

As expected, polling has less effect on SNP VMs due to the decreased overheads, but it still reduces them by up to 10%. Halt and idle polling are not far apart, but idle polling has a slight edge at the cost of additional CPU utilization. For TDX, polling is just as effective as in local networking, but CVMs with polling can still not reach the performance level of traditional or bounce buffer VMs.

Remote Networking with HTTPS

Results: If HTTPS is used, we still observe increased performance for AMD VMs compared to local networking. Intel VMs, on the other hand, show slightly reduced throughput. Table 6.5 shows that both TDX's and SNP's overheads increase for HTTPS, but while SNP's overhead is reduced compared to local networking, TDX shows higher overhead.

Analysis: Figure 6.12d shows that the VMEXIT rate for Intel TDX is increased for remote networking, leading to a higher overhead. Conversely, AMD shows significantly lower VMEXIT rates in Figure 6.12c.

Interestingly, both Intel and AMD show a slightly reduced CPU utilization in Figures 6.12e and 6.12f. While this is not surprising for Intel due to the reduced performance, it is for AMD, as its performance increases. Due to the CPU utilization still being very low (below 10% in all cases), the bounce buffer continues to play an insignificant role.

As for HTTP, polling effectiveness for SNP is reduced due to the lower remote overheads and decreased VMEXIT rates. However, the effect of polling is also slightly reduced for TDX, which shows a higher overhead. Halt polling continues to perform worse than idle polling.

Key Takeaways: AMD outperforms Intel on remote HTTP/HTTPS throughput. SNP overheads are lower than for local networking, but TDX still shows extreme overheads. The bounce buffer again plays a minor role due to low CPU utilization. While polling remains effective for both CVM types, its impact is reduced.

6.2 Computational Performance

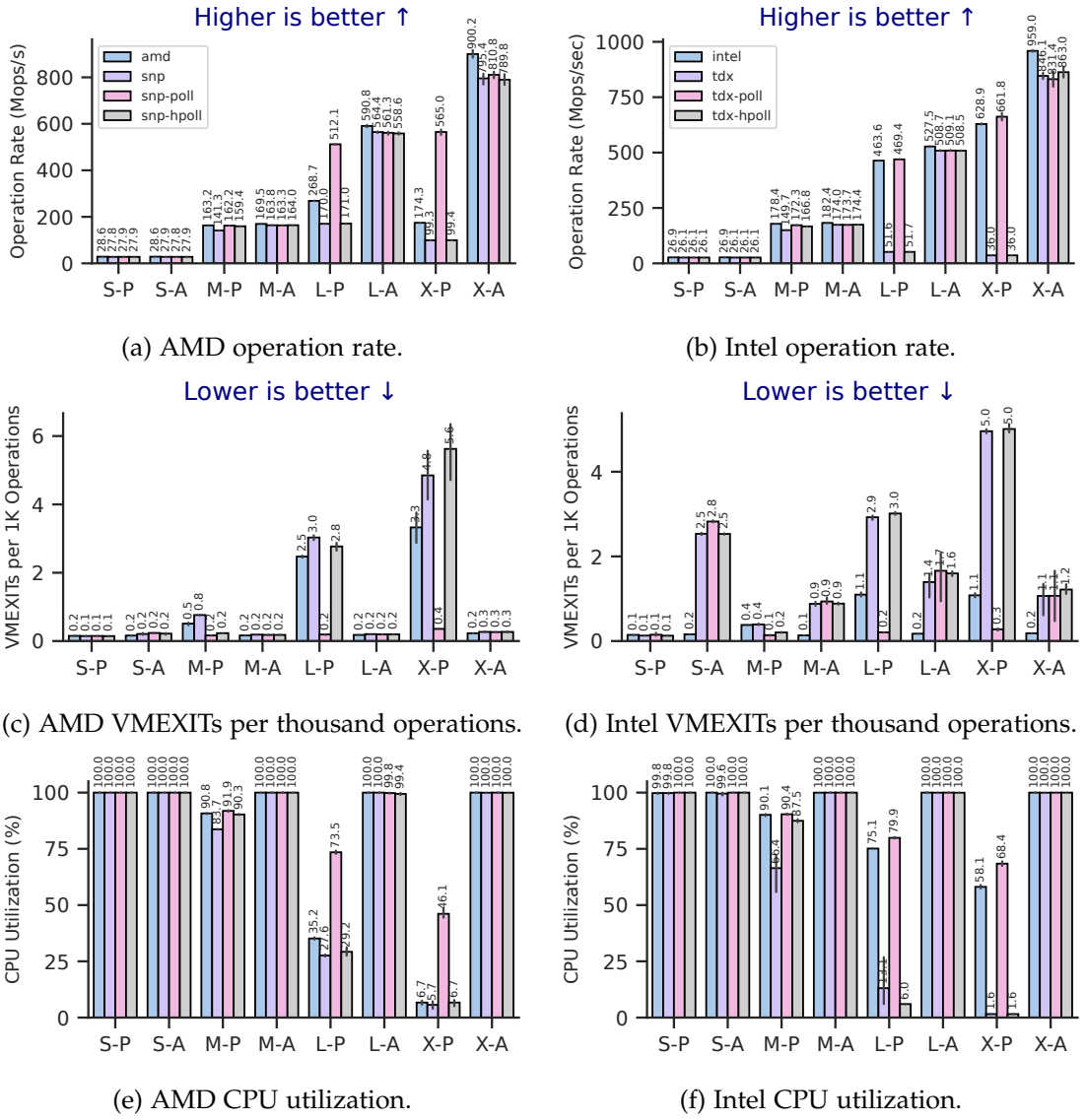
After analyzing the network I/O performance of CVMs, we examine the performance in computationally intensive workloads devoid of I/O in this section. As vhost and the bounce buffer are no longer relevant in these scenarios, they are omitted from the examined set of VM configurations, leaving only the non-confidential VMs "intel"/"amd" and the CVMs "tdx"/"snp" with the polling options.

6.2.1 Highly Parallel Computing

We observe the same metrics for this benchmark as we did for the networking ones. We distinguish between "ACTIVE" and "PASSIVE" OpenMP wait policy. While the passive policy allows threads to enter an idle state when they have no work to do, the active policy implements an active waiting mechanism. Therefore, the active wait policy and CPU polling do similar things but on different levels.

Results: Both Intel and AMD show vastly different results for the active and passive wait policy in Figures 6.13a and 6.13b. Intel outperforms AMD, but the performance is much closer than in many networking benchmarks. For the small VM size, Intel and AMD show a minimal overhead of under 5%, regardless of the wait policy used. For the active policy, the overhead increases with the VM size and reaches about 12% on the xlarge size for both Intel and AMD. In the case of the passive policy, the overheads are also more prominent for larger VM sizes, but the results show much more extreme differences. While traditional Intel VMs show similar performance to the active case, the performance of TDX VMs is extremely low for larger VM sizes with an overhead of up to 94%. AMD CVMs also perform poorly with the passive wait policy. However, the traditional AMD VMs also show radically decreased performance, leading to a lower but still very large maximum overhead of 43%.

Analysis: For the active wait policy case, we observe a fully utilized CPU in all cases and very low VMEXIT rates across Figure 6.13. As the overhead is similar for SNP and TDX in this case and stays below 15%, it is most likely caused by memory



(a) AMD operation rate.

(b) Intel operation rate.

(c) AMD VMEXITs per thousand operations.

(d) Intel VMEXITs per thousand operations.

(e) AMD CPU utilization.

(f) Intel CPU utilization.

Figure 6.13: NPB ua benchmark performance (top), VMEXIT rate (middle), and guest CPU utilization (bottom) for AMD and Intel VMs. The first letter of the x-axis markers corresponds to the VM size, and the second to the OpenMP wait policy used. Example: L-A means VM of size large with OpenMP policy "ACTIVE".

encryption and other CVM mechanisms. In the passive wait policy case, however, the extreme overheads correlate with significantly reduced CPU utilization and much higher VMEXIT rates. In accordance with the network benchmarks, we observe a high rate of HLT and MMIO-related VMEXITs for SNP VMs. Contrary to the Intel VM, the traditional AMD VM also shows a very high rate of HLT VMEXITs in the passive case, leading to severely decreased performance compared to the active case.

This highly parallel scenario requires many synchronization events between threads, which seemingly poses a problem for SNP and TDX if the passive policy is used. The high latency encountered when being woken up due to a synchronization event seems to introduce a huge performance bottleneck. This also explains why the performance drops significantly with more available CPUs, as more cores mean more synchronization events.

Just as for networking benchmarks, the measured hardware metrics show no clear trends or explanations for any of the overheads, leading to the conclusion that for both I/O-heavy and CPU-heavy workloads, a high VMEXIT rate and increased VMEXIT costs are the primary causes of overhead in extreme overhead scenarios.

For both AMD and Intel TDX, in the passive policy case, idle polling massively increases performance back to normal levels and reduces the VMEXIT rate immensely. Halt polling, on the other hand, has no real impact. As for other high CPU intensity cases, polling does not bring much of a performance increase if the active policy is used, as there are already no VMEXIT due to idle CPUs.

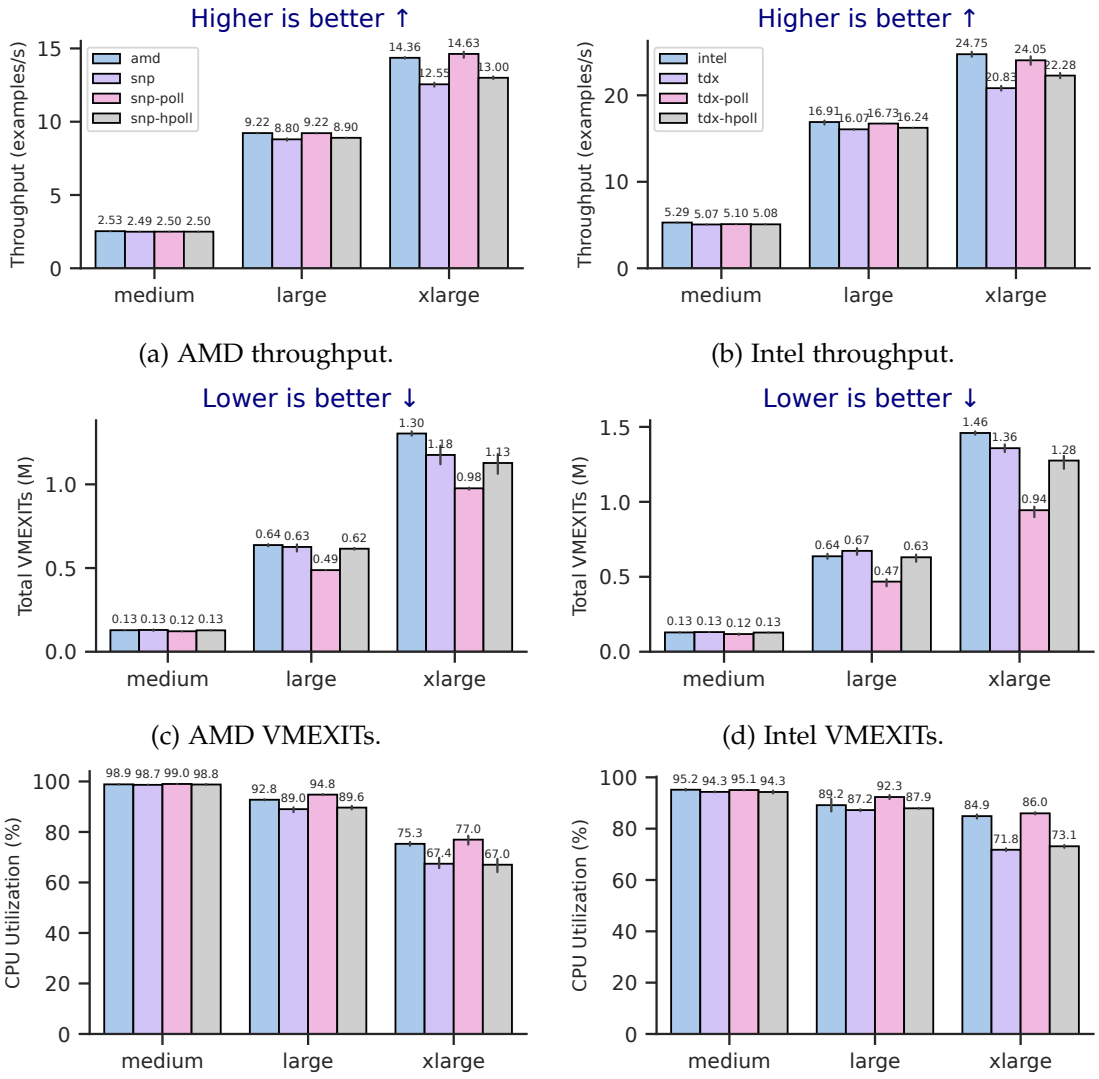
Key Takeaways: SNP and TDX show an expected overhead of about 12% if the active wait policy is used. If not, they show extreme performance degradation of up to 94%. Idle polling fixes the issue, returning performance to normal levels. Halt polling has no effect.

6.2.2 Machine Learning

While this benchmark also heavily uses multiple CPU cores, it does not feature the same high frequency of synchronizations as the last benchmark. As the benchmark would not successfully run on a small-sized VM, only the bigger sizes are presented.

The absolute number of VMEXITs presented in this subsection is the number measured during our metric measurement window and not the number for the whole benchmark.

Results: In this scenario, Figures 6.14a and 6.14b show that Intel performs much better than AMD again, but TDX also shows a slightly larger overhead than SNP. The



(a) AMD throughput.

(b) Intel throughput.

(c) AMD VMEXITs.

(d) Intel VMEXITs.

(e) AMD CPU utilization.

(f) Intel CPU utilization.

Figure 6.14: TensorFlow BERT benchmark performance (top), number of VMEXITs (middle), and CPU utilization (bottom) for AMD and Intel.

performance drop for CVMs increases again, with VM size reaching a maximum of about 13% and 16% for SNP and TDX, respectively. For the medium and large VM sizes, the overhead for both SNP and TDX stays below 5%.

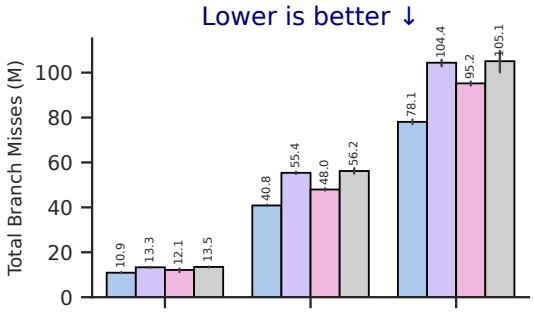


Figure 6.15: Total number of branch misses for AMD VMs.

Analysis: Figures 6.14e and 6.14f show that, just like in the previous benchmark, the CPU utilization decreases with VM size. At the same time, the CVMs do not show an increased number of total VMEXIT performed during the benchmark. Thus, the performance decrease mostly comes from CVM confidentiality mechanisms and the increased VMEXIT cost. As the total amount of VMEXITs shown in Figures 6.14c and 6.14d is very similar for both Intel and AMD, the higher overhead of TDX can be explained by its higher VMEXIT cost compared to SNP.

The rate of MMIO-related VMEXITs is very low for SNP, meaning that the problems with programming the local APIC do not play a huge role here. However, in Figure 6.15, for the first time in any benchmark, we observe a significantly higher rate of branch misses for the SNP CVM. SNP VMs show about 30% more branch misses during the benchmark, suggesting a possible impact on performance. L1 cache misses, TLB misses, and the total number of instructions executed do not show an increase for SNP.

Both halt and idle polling have a positive impact on performance for both SNP and TDX. Idle polling is much more effective, however. It reduces the maximum overhead to only 3% for TDX and removes it entirely for SNP. CPU utilization only slightly increases compared to traditional VMs. The effects can also be observed in the total amount of VMEXIT and branch misses, which are both reduced significantly. However, even with polling enabled, the total branch misses are still increased for SNP. This shows that the branch misses are not solely tied to the VMEXIT process and must result from some additional BTB flushes as mentioned in the SNP whitepaper [4].

Key Takeaways: Both SNP and TDX show performance overheads of 13% to 16%, respectively. SNP shows an increased amount of branch misses. The total number of VMEXITs is not increased for SNP and TDX. Polling significantly improves performance and mostly removes CVM overheads.

6.3 Discussion

After examining and analyzing the performance differences on a per-benchmark scale, it is important to take a step back from the large amount of data analyzed and briefly discuss the results again across different benchmarks.

During this analysis, we identify multiple factors that can lead to reduced CVM performance. We find that in I/O-heavy settings with high CPU utilization, such as the TCP throughput benchmark with vhost enabled, the bounce buffer is responsible for a large portion of the overhead encountered, especially for Intel TDX. In these cases, the amount of VMEXITs takes on a minor role, leading to a significantly diminished effect of both idle and halt polling on CVM performance.

In scenarios with lower CPU utilization, such as the TCP throughput benchmark without vhost or the web server performance benchmark, we find that the bounce buffer either has minimal effects on performance or can even slightly improve it due to its additional CPU load, reducing the frequency of HLT instructions. Polling has a similar but more substantial effect in these cases.

The most common reason for large performance decreases by CVMs is an increased VMEXIT rate. Across most benchmarks, we see an increased number of VMEXITs for CVMs. For SNP, these high VMEXIT rates are mainly caused by the lack of support for TSC-Deadline mode. In addition to adding more VMEXITs, using legacy modes to control the local APIC also leads to less precise interrupt timing. Together with the increased cost each VMEXIT has for CVMs, these factors can result in severe performance degradation in some scenarios, such as UDP throughput or highly parallel computing. It seems that CVM performance is very heavily decreased in scenarios that do not feature a linear execution flow, as is the case with the unsynchronized and connectionless UDP protocol or certain multithreaded workloads such as the NPB benchmark and the *memcached* server. In these scenarios, we see the most extreme CVM overheads and the highest effect of polling, which massively improves performance in these cases. The strong effect CPU polling has supports that it is truly VMEXIT behavior that causes these extreme overheads.

Idle polling performs significantly better than halt polling in most cases while only incurring a slightly increased tax on CPU performance. Sometimes, however,

halt polling shows higher effectiveness, suggesting it may be possible to tweak the halt polling parameters according to the specific workload and gain a slightly better cost-performance ratio.

The vhost protocol significantly improves traditional VM performance in most scenarios, and it does so for CVMs in some cases, most notably for TCP throughput. However, it often has a negative effect on CVM performance, especially for Intel TDX.

For both computational and network I/O workloads, we generally do not see a relevant impact of SEV-SNP technology on system metrics such as branch and TLB misses, even if the CPU is fully utilized. The only outlier here is the machine learning benchmark, where we observe increased branch misses.

6.4 Summary

We find CVMs to suffer from significant overheads in both network I/O intensive and computational workloads. While the overheads stay below 20% in most scenarios, there are extreme cases with overheads upwards of 80%. Intel VMs mostly outperform AMD VMs, but Intel TDX often has a higher overhead than AMD SNP.

Depending on the specific workload, these overheads are caused mainly by a combination of the bounce buffer mechanism, an increased VMEXIT rate for CVMs, and a high cost for VMEXITs. For AMD SNP, we show that the absence of the TSC-Deadline mode to set APIC timers causes many MMIO-related VMEXITs and less precise interrupts.

Our examination of hardware performance metrics such as branch misses, L1 cache misses, or TLB misses shows that in the vast majority of cases, SNP CVM performance is not affected negatively. However, in some specific scenarios, SNP CVMs can show an increased rate of TLB misses or branch misses.

We show that guest-side CPU idle polling is an excellent mechanism for massively decreasing VMEXIT count in many scenarios. It helps to remove the most extreme cases of CVM overheads and generally increases performance at the cost of only slightly higher CPU utilization. Halt polling also proved effective in many cases but mostly lacked behind idle polling.

7 Related Work

7.1 CVM Analysis

Misono et al. thoroughly analyze both AMD SEV-SNP and Intel TDX CVMs [33]. They analyze boot time, memory allocation speed, I/O, context switch overhead, and more. They show that CVMs not only cause overhead in application workloads but also significantly increase boot times and cause heavy overheads during memory allocation. Furthermore, the authors reveal that performing VMEXITs takes much longer for CVMs, leading to significant performance overheads in scenarios reliant on frequent vCPU wake-ups. The authors propose CPU polling as an optimization technique and show that it can significantly reduce overheads in certain cases.

However, in contrast to this thesis, they do not include remote networking results or performance metrics such as branch, TLB, or cache misses and generally discuss the results in little detail.

7.2 Network I/O Optimization

Li et al. (2023) introduce "Bifrost" [28], a para-virtualized I/O design for AMD SEV-ES capable of significantly reducing CVM network I/O overhead caused by the bounce buffer mechanism. They leverage the encryption provided by TLS to eliminate the bounce buffer from the I/O mechanism and remove redundant encryption. They also use pre-receiver packet reassembly to reduce the packet processing cost while ensuring that SEV-ES's security guarantees are not compromised. Furthermore, they evaluate their approach using different network I/O benchmarks and show that their design significantly improves the I/O performance of CVMs, enabling them to outperform even traditional VMs by up to 21.5%.

Li et al. (2024) present "FOLIO" [29], a DPDK [50] based software solution for AMD SEV-SNP that removes CVM overheads in different areas and reaches performance close to Trusted I/O (TIO) solutions such as AMD SEV-TIO [2]. However, FOLIO relies on the CPU and software-based encryption to achieve end-to-end security like a VirtIO-based solution and does not place the NIC inside the TCB. The authors argue that main memory encryption is the primary part of CVM overhead that can not be

solved in software and may only be responsible for about 2% of all overhead.

7.3 Accelerated Context Switch

Weisse et al. propose "HotCalls" [51], an interface designed to reduce the impact of context switches for Intel SGX. The authors identify context switches necessary to facilitate system calls as a major bottleneck and propose a spin-lock synchronization-based mechanism using shared/unencrypted memory to reduce overheads massively. The solution uses a requester and responder model where an additional thread polls a shared memory region for a requested call and executes it for the requester. The authors evaluate their approach and show that it can increase application throughput by a factor of up to 3.7x and reduce latencies by up to 74%.

Vilanova et al. present an architecture to use the duplicate thread execution resources in CPUs that support Simultaneous Multithreading (SMT) to alleviate overheads caused by VM traps in nested virtualization. They propose to colocate different entities, such as VMs or the hypervisor, on different hardware threads of a single core, and replacing the costly context switch between them with thread stall and resume events. The authors show that their method significantly increases I/O throughput and latency, and applications can achieve speed-ups of up to 2.2x.

8 Conclusion

In this thesis, we analyze the overheads encountered by Intel TDX and AMD SEV-SNP Confidential Virtual Machines in both network I/O and computational workloads. We do this by designing and employing an automated system to run various benchmarks to examine performance in several key areas. In addition, we collect various metrics during the benchmark runs to facilitate the analysis of the root causes for the performance differences. Our system supports several configuration options, such as the vhost protocol or the bounce buffer mechanism, to give a broad overview of how different mechanisms affect CVM performance. Using these options, we analyze various VM configurations, allowing us to better understand the origin of CVM performance decreases.

Our analysis shows that CVMs can exhibit significant overheads of more than 80% in network I/O and purely computational workloads.

We use the knowledge gained from our experiments to identify VMEXIT behavior as the main reason for unusually high CVM overheads and present idle and halt guest-side CPU polling as an easy-to-implement and low-cost optimization technique to reduce the number of unnecessary VMEXITs.

We show that SNP CVMs are not able to access specific MSR registers to handle local APIC timer configuration and need to use legacy methods to achieve the same goal. These MMIO-based methods cause a high VMEXIT rate and can severely reduce performance in some cases where frequent timed interrupts are required. While we cannot present a conclusive reason for an increased VMEXIT rate of Intel TDX machines, we still show that TDX CVM performance is severely decreased in some cases where there is a high rate of VMEXITs.

In the performance metrics we analyze, in addition to VMEXIT-related data, we do not find any clear trends suggesting that SNP CVM performance is negatively affected by a high rate of branch, L1 cache, or TLB misses in most scenarios. There are, however, a few cases where SNP VMs show an increased amount of branch or TLB misses.

Our results show that, in most cases, polling is an effective way to reduce unnecessary VMEXITs for both Intel and AMD CVMs, reducing the overheads encountered drastically in some scenarios. While polling does not always significantly enhance performance, we show that it has no severe detrimental effects in these cases.

We believe that while our analysis can not wholly explain every phenomenon en-

countered, it still provides many insights into the performance challenges faced by AMD SEV-SNP and Intel TDX, forming a good foundation for further research.

9 Future Work

As we analyze CVM performance across an extensive range of scenarios in this thesis, its scope does not allow us to conduct a deep inspection of every phenomenon encountered. This opens up many interesting possible directions for future research work.

TDX hardware events. Our analysis of system hardware events is limited to SNP CVMs, as our TDX hardware does not support the capture of these metrics. Other hardware, however, supports it, making an analysis possible. It would be a valuable insight to compare TDX's effect on hardware events, such as branch or cache misses, with SNP's impact.

Deeper analysis of TDCALLs. In our thesis, we do not further resolve TDCALLs for Intel TDX, limiting the insights we obtain on the reasons for increased VMEXIT rates on TDX VMs. Using tools like *bpftrace*, it is possible to extract the reason leading to a TDCALL, enabling a deeper analysis of TDX's VMEXIT behavior.

Deeper analysis of high overhead scenarios. Due to the broad scope of our analysis, we cannot do a thorough deep-dive analysis of the scenarios we examine. Especially for scenarios where we see extreme overhead, e.g., UDP throughput, a more in-depth examination could identify the root cause and maybe enable a proposition on how to resolve the issue rather than only counteract it through CPU polling.

Vhost-user protocol. We conduct our network I/O performance analysis using the virtio-net/vhost-net network stack. As is briefly mentioned in chapter 2, there is a user-space implementation of the vhost protocol (vhost-user) [41] that would be an interesting direction to study the effects vhost has on CVMs further.

Halt polling parameters. In our analysis of polling as an optimization technique, we examine halt polling, but only with the often suboptimal default parameters. Adjusting these parameters to better match the specific workload could provide interesting results, especially compared to idle polling.

Abbreviations

AES Advanced Encryption Standard

AE Automatic Exit

APIC Advanced Programmable Interrupt Controller

ASP AMD Secure Processor

BERT Bidirectional Encoder Representations from Transformers

BTB Branch Target Buffer

CVM Confidential Virtual Machine

DMA Direct Memory Access

EPT Extended Page Table

ES Encrypted State

GHCB Guest Hypervisor Communication Block

GPA Guest Physical Address

GVA Guest Virtual Address

KVM Kernel Virtual Machine

MAC Message Authentication Code

MMIO Memory Mapped I/O

MSR Model Specific Register

NAE Non-Automatic Exit

NIC Network Interface Card

NUMA Non-Uniform Memory Access

PAMT Physical Address Metadata Table

PKI Public Key Infrastructure

RMP Reverse Map Table

SEAM Secure-Arbitration Mode

SEV Secure Encrypted Virtualization

SGX Software Guard Extensions

SME Secure Memory Encryption

SMT Simultaneous Multithreading

SNP Secure Nested Paging

SPA System Physical Address

TCB Trusted Computing Base

TDX Trust Domain Extensions

TD Trust Domain

TEE Trusted Execution Environment

TIO Trusted I/O

TLB Translation Lookaside Buffer

TLS Transport Layer Security

TME-MK Total Memory Encryption Multi-Key

TSC Time Stamp Counter

VMCB Virtual Machine Control Block

VMM Virtual Machine Manager

swiotlb Software I/O Translation Lookaside Buffer

List of Figures

List of Tables

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. `https://www.tensorflow.org/`. Accessed: 2024-10-08.

[2] Advanced Micro Devices. *AMD SEV-TIO: Trusted I/O for Secure Encrypted Virtualization*. Whitepaper. 2023.

[3] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual Volume 2: System Programming*. Revision 3.42. Mar. 2024.

[4] Advanced Micro Devices. *Strengthening VM Isolation with Integrity Protection and More*. Whitepaper. 2020.

[5] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. "Innovative technology for CPU based attestation and sealing." In: *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*. Vol. 13. 7. ACM New York, NY, USA. 2013.

[6] Arm Limited. *Introducing Arm Confidential Compute Architecture Version 3.0*. `https://developer.arm.com/documentation/den0125/0300/`. Accessed: 2024-09-19.

[7] Brad Fitzpatrick and contributors. *memcached*. `https://memcached.org/`. Accessed: 2024-09-08.

[8] CanaryBit. *Intel SGX vs TDX: what is the difference?* `https://www.canarybit.eu/intel-sgx-vs-tdx-what-is-the-difference/`. Accessed: 2024-09-24.

[9] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley. "Intel TDX demystified: A top-down approach." In: *ACM Computing Surveys* (2024), pp. 1–33.

[10] I. Corporation. *Runtime Encryption of Memory with Intel® Total Memory Encryption–Multi-Key (Intel® TME-MK)*. `https://www.intel.com/content/www/us/en/developer/articles/news/runtime-encryption-of-memory-with-intel-tme-mk.html`. Accessed: 2024-09-23.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics. 2019, pp. 4171–4186.

[12] Energy Sciences Network (ESnet). *iperf*. `https://software.es.net/iperf/`. Version 3.17.1. Accessed: 2024-09-07.

[13] J. Forcier. *PyInvoke: Pythonic task execution*. `https://www.pyinvoke.org`. Accessed: 2024-09-30.

[14] W. Futral and J. Greene. *Intel Trusted Execution Technology for Server Platforms: A Guide to More Secure Datacenters*. Springer Nature, 2013.

[15] A. Galanou, K. Bindlish, L. Preibsch, Y.-A. Pignolet, C. Fetzer, and R. Kapitza. "Trustworthy confidential virtual machines for the masses." In: *Proceedings of the 24th International Middleware Conference*. 2023, pp. 316–328.

[16] W. Glozer. *wrk*. `https://github.com/wg/wrk`. Accessed: 2024-09-08.

[17] S. Godard and contributors. *sysstat - System performance tools for the Linux operating system*. `https://github.com/sysstat/sysstat`. Version 12.7.4, Accessed: 2024-10-01. 2024.

[18] Google. *Confidential VM overview*. `https://cloud.google.com/confidential-computing/confidential-vm/docs/confidential-vm-overview`. Accessed: 2024-09-19.

[19] D. R. Hipp. *SQLite*. `https://www.sqlite.org`. Accessed: 2024-10-02.

[20] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual*. `https://cdrdv2.intel.com/v1/dl/getContent/671200`. Accessed: 2024-10-12.

[21] Intel Corporation. *Intel® Trust Domain Extensions*. Whitepaper. 2023.

[22] D. Kaplan. *Protecting VM Register State with SEV-ES*. Whitepaper. Advanced Micro Devices, 2017.

[23] D. Kaplan, J. Powell, and T. Woller. *AMD Memory Encryption*. Whitepaper. Advanced Micro Devices, 2016.

[24] Kernel development community. *DMA and swiotlb*. `https://docs.kernel.org/next/core-api/swiotlb.html`. Accessed: 2024-09-21.

[25] Kernel development community. *Guest halt polling*. `https://docs.kernel.org/virt/guest-halt-polling.html`. Accessed: 2024-09-26.

[26] Kernel development community. *The Definitive KVM (Kernel-based Virtual Machine) API Documentation*. https://docs.kernel.org/virt/kvm/api.html. Accessed: 2024-09-09.

[27] KVM. *Kernel Virtual Machine*. https://linux-kvm.org/index.php?title=Main_Page. Accessed: 2024-09-09.

[28] D. Li, Z. Mi, C. Ji, Y. Tan, B. Zang, H. Guan, and H. Chen. "Bifrost: Analysis and Optimization of Network {I/O} Tax in Confidential Virtual Machines." In: *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 2023, pp. 1–15.

[29] M. Li, S. Srivastava, and M. Yan. "Bridge the Future: High-Performance Networks in Confidential VMs without Trusted I/O devices." In: *arXiv preprint arXiv:2403.03360* (2024).

[30] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. "Innovative instructions and software model for isolated execution." In: *Hasp@isca* 10.1 (2013).

[31] Michael S. Tsirkin, Cornelia Huck. *Virtual I/O Device (VIRTIO) Version 1.2*. https://docs.oasis-open.org/virtio/virtio/v1.2/virtio-v1.2.html. Accessed: 2024-09-09.

[32] Microsoft. *About Azure confidential VMs*. https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-vm-overview. Accessed: 2024-09-19.

[33] M. Misono, D. Stavrakakis, N. Santos, and P. Bhatotia. "Confidential VMs Explained: An Empirical Analysis of AMD SEV-SNP and Intel TDX." In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)* 8.3 (2024).

[34] NASA. *NAS Parallel Benchmark*. https://www.nas.nasa.gov/software/npb.html. Accessed: 2024-09-08.

[35] NixOS contributors. *NixOS*. https://nixos.org/. Accessed: 2024-09-16.

[36] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 5.2*. https://www.openmp.org/specifications. Accessed: 2024-10-08. 2023.

[37] L. K. Organization. *perf: Linux profiling with performance counters*. https://perf.wiki.kernel.org. Accessed: 2024-10-01. 2024.

[38] QEMU Project Developers. *About QEMU*. https://www.qemu.org/docs/master/about/index.html. Accessed: 2024-09-09.

[39] QEMU Project Developers. *Direct Linux Boot*. https://qemu-project.gitlab.io/qemu/system/linuxboot.html. Accessed: 2024-09-09.

[40] QEMU Project Developers. *Introduction*. https://www.qemu.org/docs/master/system/introduction.html. Accessed: 2024-09-09.

[41] QEMU Project Developers. *Vhost-user Protocol*. https://qemu-project.gitlab.io/qemu/interop/vhost-user.html. Accessed: 2024-10-21.

[42] Rafael J. Wysocki. *CPU Idle Time Management*. https://www.kernel.org/doc/html/v5.0/admin-guide/pm/cpuidle.html. Accessed: 2024-09-26.

[43] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. "SQuAD: 100,000+ Questions for Machine Comprehension of Text." In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2016, pp. 2383–2392.

[44] RedHat Inc. *What is KVM?* https://www.redhat.com/en/topics/virtualization/what-is-KVM. Accessed: 2024-09-09.

[45] RedisLabs. *memtier_benchmark*. https://github.com/RedisLabs/memtier_benchmark. Version 2.1.0. Accessed: 2024-09-07.

[46] RedisLabs. *redis*. https://github.com/redis/redis. Accessed: 2024-09-08.

[47] A. Robertson and contributors. *bpftrace*. https://github.com/iovisor/bpftrace. Version 0.20.4, Accessed: 2024-10-02. 2024.

[48] S. Srivastava. "Architecting Trust: Building Secure and High-Performance Confidential VMs." PhD thesis. Massachusetts Institute of Technology, 2023.

[49] I. Sysoev. *nginx*. https://nginx.org/. Accessed: 2024-09-08.

[50] The DPDK Project. *Data Plane Development Kit (DPDK)*. https://www.dpdk.org. Accessed: 2024-10-21. 2024.

[51] O. Weisse, V. Bertacco, and T. Austin. "Regaining lost cycles with HotCalls: A fast interface for SGX secure enclaves." In: *ACM SIGARCH Computer Architecture News* 45.2 (2017), pp. 81–93.

[52] H. Yoshifuji, A. Kuznetsov, and contributors. *iputils*. https://github.com/iputils/iputils. Version 20240117. Accessed: 2024-10-9.