




Bifrost: Analysis and Optimization of Network I/O Tax in Confidential Virtual Machines

Dingji Li^{1,2,3}, Zeyu Mi^{1,2}, Chenhui Ji^{1,2}, Yifan Tan^{1,2},
Binyu Zang^{1,2}, Haibing Guan⁴, and Haibo Chen^{1,2}

¹*Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University*

²*Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China*

³*MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University*

⁴*Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University*

Abstract

Existing confidential VMs (CVMs) experience notable network performance overhead compared to traditional VMs. We present the first thorough performance analysis of various network-intensive applications in CVMs and find that the *CVM-IO tax*, which mainly comprises the bounce buffer mechanism and the packet processing in CVMs, has a significant impact on network I/O performance. Specifically, the *CVM-IO tax* squeezes out virtual CPU (vCPU) resources of performance-critical application workloads and may occupy more than 50% of CPU cycles. To minimize the *CVM-IO tax*, this paper proposes Bifrost, a novel para-virtualized I/O design that 1) eliminates the I/O payload bouncing tax by removing redundant encryption and 2) reduces the packet processing tax via pre-receiver packet reassembly, while still ensuring the same level of security guarantees. We have implemented a Bifrost prototype with only minor modifications to the guest Linux kernel and the userspace network I/O backend. Evaluation results on both AMD and Intel servers demonstrate that Bifrost significantly improves the performance of I/O-intensive applications in CVMs, and even outperforms the traditional VM by up to 21.50%.

1 Introduction

As more and more data-processing applications [9, 13, 14, 57] embrace the cloud, widespread concerns are being raised about the security and privacy of data in-use on the cloud. To address these concerns, various confidential computing solutions have been proposed to safeguard data from unauthorized parties. Among them, confidential virtual machine (CVM) solutions, such as AMD SEV [2, 3], Intel TDX [32] and ARM CCA [8], run guest operating systems (OSes) in hardware-isolated environments. In these environments, the complex virtualization stack, such as hypervisor and host OS, is no longer trusted and cannot access data in guest OSes arbitrarily, while still providing resource management functions. This CVM abstraction transparently protects user workloads without requiring any modifications and integrates easily into

the existing cloud infrastructure. Therefore, it has gained popularity and is increasingly deployed in data centers.

Unfortunately, while the speed of modern network devices continues to grow (Terabit Ethernet [60] like NVIDIA 400Gbps NIC [51]), the security protections introduced by existing CVM solutions have a significant negative impact on network performance. This paper first conducts a series of experiments to thoroughly analyze the network I/O performance of CVMs. We evaluate widely-deployed network-intensive applications in an AMD SEV-ES/SNP server and a simulated Intel TDX server. The results demonstrate that CVM's security protections significantly increase the *CVM-IO tax*, which we define as the CPU resources used during CVM's I/O procedure, resulting in up to 29% overhead over a traditional VM that does not use any CVM protections. The *CVM-IO tax* is caused by both security protections and intrinsic network I/O procedures in CVMs, draining substantial CPU resources from diverse application workloads.

Concretely, there are three common components in the CVM-IO tax: ① **VM exits** consume up to 11.54% more CPU cycles than the traditional VM. The time consumption of VM exits is greatly increased due to the security checks and protections from the trusted modules (e.g., AMD-SP [3], Intel TDX module [31]) and making the guest aware of emulation events (e.g., AMD #VC [3], Intel #VE [29]). ② The **bounce buffer mechanism**, an I/O staging memory shared between the CVM and hypervisor, takes up to 19.45% CPU cycles for bouncing packets (including headers and payload). I/O operations that could previously be done directly by the hypervisor to the traditional VMs must now be assisted by the bounce buffer mechanism in guest OSes. For example, to emulate a virtual NIC, the hypervisor in traditional VM systems can forward packets between the guest OS and the host network stack by directly copying I/O data to/from the guest private memory. But hypervisors in CVM systems require the guest OS to bounce packets to/from a hypervisor-visible shared memory region due to the memory encryption, introducing I/O data copy overhead. ③ The **packet processing** also spends up to 36.14% CPU cycles preparing payloads from massive network packets for application workloads. The

 Corresponding author: Zeyu Mi (yzmizeyu@sjtu.edu.cn).

higher the number of packets transferred to the network stack, the more vCPU resources a CVM requires to process their headers. Fortunately, the cost of VM exits becomes negligible when the posted interrupt [59] feature is supported by the hardware, leaving the bounce buffer mechanism and packet processing as the main components of the CVM-IO tax.

This paper aims to **reduce as much CVM-IO tax as possible** for I/O-intensive applications in CVMs by bypassing the bounce buffer mechanism and offloading the packet processing. A straightforward design to bypass the bounce buffer mechanism is to keep the packet content in place by dynamically adjusting the accessibility of the same memory region to the hypervisor. However, this approach is limited by the memory encryption hardware support [34], which does not allow the plaintext contents of a memory region to be preserved when modifying the accessibility of the memory region [29]. To reduce network packet processing cost, the existing design is to pass fewer packets to the network stack by reassembling multiple small packets into a large one in the guest device driver. But the guest device driver still has to process a large number of packets, consuming substantial CPU resources.

Fortunately, there are three observations that can help us address the challenges mentioned above. We observe that either end-to-end encryption or a CVM’s private memory alone can protect data security, while applying both protections to the payload is redundant. Additionally, we notice that end-to-end encryption/decryption can also change the payload’s memory location, which is functionally equivalent to bouncing between two memory regions. As a result, bypassing payload bouncing can be achieved by directly encrypting/decrypting the payload into/from the guest-host shared memory. Another observation is that the network I/O backend typically has plenty of residual CPU resources. Given the bottleneck experienced by the saturated vCPUs of network-intensive CVMs, an opportunity arises to offload packet processing to the network I/O backend. This approach effectively utilizes the available CPU resources, alleviating the strain on vCPUs and resulting in improved performance.

Based on these observations, this paper proposes **Bifrost**¹ to improve the paravirtual network performance of the CVM with three techniques: ① The *zero-copy encryption deduplication* eliminates payload bouncing by leveraging dedicated guest-host shared memory to remove redundant encryptions on the payload in a zero-copy way. When receiving packets, the end-to-end encrypted payload is directly decrypted from the shared memory. When sending packets, the payload is directly encrypted into the shared memory. To minimize modifications, the shared memory is in the form of dedicated non-uniform memory access (NUMA) [37] nodes, allowing memory allocators in the guest kernel to be reused. ② The *one-time trusted read* mechanism protects guest OSes from time

of check to time of use (TOCTTOU) attacks while accessing packets in the dedicated shared memory. With these two techniques, the bouncing of the end-to-end encrypted payload, which takes up much CPU resources of CVMs, is securely bypassed. ③ The *pre-receiver packet reassembly* reduces vCPU resources utilized by the device driver by offloading the task of reassembling received packets to the network I/O backend. Thus, CVMs are able to process fewer packets with larger payload, reducing the packet processing cost on vCPUs.

We have implemented a Bifrost prototype by modifying the guest OS kernel and host user-level software. The prototype extends the Linux v6.0-rc1 kernel in the guest OS with 815 lines of code, and adds 175 lines to OpenvSwitch v2.17.3 and 541 lines to DPDK v21.1.2, both of which run in the host user mode. We have also evaluated Bifrost’s performance on both AMD and Intel platforms. The results show that, with advanced posted interrupt support, Bifrost enhances the performance of I/O-intensive applications in CVMs, surpassing traditional VMs by up to 21.50%.

In summary, this paper makes the following contributions:

- The first thorough performance analysis of I/O-intensive applications in CVMs on existing and next-generation hardware platforms, revealing their bottlenecks and overhead sources compared to traditional VMs.
- A secure paravirtual I/O design that greatly reduces the CVM-IO tax, significantly improving the performance of I/O-intensive applications in CVMs.
- A Bifrost prototype and a comprehensive evaluation on AMD and Intel platforms, demonstrating improvements on existing and future CVM hardware. The prototype is available at <https://github.com/IPADS-Bifrost>.

2 Background

2.1 Confidential VMs (CVMs)

There are different CVM solutions based on specialized hardware extensions. All of these solutions leverage hardware memory encryption and integrity checking [30, 34] to enforce confidentiality and integrity. They share the same CVM abstraction that excludes the entire virtualization stack from the trusted computing base (TCB). As shown in Figure 1, the trusted firmware, which is the unique software TCB, isolates the CVM from untrusted hypervisors and traditional VMs.

Existing CVM systems typically divide the physical memory of a VM into two major security types: private memory and shared memory. The private memory is encrypted by hardware and cannot be accessed or modified by any untrusted entities outside the VM, while the shared memory holding plaintext data can be accessed by the hypervisor. The CVM systems also allow the hypervisor to switch private memory and shared memory to each other at runtime. However, the data content of the memory page cannot be preserved before and after the security type switch [29]. Hence, the guest must

¹Bifrost, the rainbow bridge from Norse mythology, metaphorically represents the secure and rapid transfer of CVM’s I/O data (Asgard’s gods) to and from the untrusted hypervisor (Midgard).

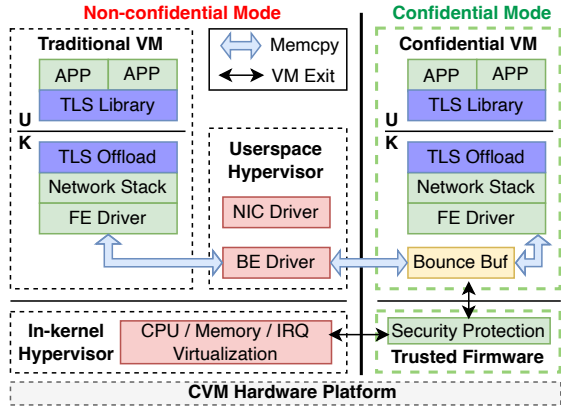


Figure 1: The paravirtual I/O networking architecture of traditional VMs and CVMs atop the CVM hardware platform. The black arrows represent the path of VM exits and VM enters. The light blue arrows indicate memory copies that consume CPU resources. The FE Driver and the BE Driver in the figure represent drivers in frontend and backend, respectively.

move data outside of private memory before the security type switch, and then copy it back to the new shared memory. Besides, it takes much effort to finish the security type switch. The guest OS has to cooperate with the host hypervisor to alter address translation data structures and maintain CPU micro-architectures, requiring multiple VM exits and inter-processor communications [4, 29]. As a result, the security type switch is unsuitable to occur frequently in CVMs.

2.2 Paravirtual I/O Networking in CVM

Paravirtual I/O has become a primary I/O virtualization choice for modern cloud providers owing to its high performance and excellent compatibility. There are two cooperative drivers in paravirtual I/O, a frontend driver in the guest VM and a backend driver in the hypervisor, which communicate with each other through shared memory. To provide maximum network performance, the backend driver can be deployed in the host userspace, for instance, using vhost-user [46, 53], to directly control the device in a busy-polling mode [16].

An example of paravirtual I/O networking of the traditional VM is shown in the left part of Figure 1. The applications in the userspace deal with payload, while the network stack and the frontend driver in the kernel handle packet processing. The packet processing includes network functions that handle conversions between payload and packets. For example, in the transmission (TX) direction, the payload from applications and the headers from the network stack are encapsulated into network packets, after which the backend driver is notified to send them out. Because the hypervisor can access the entire memory space of a traditional VM, the backend driver can copy the packets freely from the guest memory to its own memory and forwards them to the NIC driver.

Memory pages in CVMs, including those containing packets, are set to private by default. However, the host OS is untrusted and cannot access the private memory of CVMs

(see § 2.1). To allow the host OS to transfer packets, CVMs utilize a bounce buffer mechanism that sets up a guest-host shared memory as an intermediary. As shown in the right part of Figure 1, the guest OS reserves a shared memory region with the host OS as the bounce buffer and copies the outgoing packets to it. Afterwards, the backend driver can copy the packets to the hypervisor as normal. As a result, the bounce buffer leads to excessive memory copies for I/O virtualization.

2.3 Transport Layer Security (TLS)

TLS is an end-to-end security protocol designed to protect data in transit by leveraging cryptography. It has been commonly used by modern applications to secure their I/O payload in transit [6, 17, 49, 55, 61]. CVM solutions have made it a mandatory requirement for their applications [22, 26, 54]. Moreover, today’s OSes, such as Linux, provide in-kernel TLS support, enabling userspace applications to offload TLS to the kernel for enhanced performance and expanded features [19]. As shown in Figure 1, in-kernel TLS allows the payload from the page cache to be encrypted without going through the userspace.

The industry currently implements the TLS protocol based on encryption algorithms such as AES-GCM [18] to assure the confidentiality and integrity of data simultaneously. The output of these encryption algorithms consists of encrypted ciphertext for confidentiality, and an authentication tag generated from the ciphertext for integrity. To provide complete data security protection, the correctness of both the ciphertext and its authentication tag must be guaranteed during encryption, and vice versa.

2.4 Exitless Interrupt Virtualization

In the paravirtual I/O networking scenario, when a virtual NIC (i.e., network backend) receives some network packets, it notifies the guest VM with a virtual interrupt. The guest VM then needs to interact with the virtual interrupt controller to perform Acknowledgment (ACK) and End of Interrupt (EOI). Traditional techniques rely on the hypervisor to emulate interrupt delivery and interrupt controller access of guest VMs using trap-and-emulate approaches. However, virtualizing interrupts in this way can be a significant source of overhead, as each virtual interrupt’s completion necessitates multiple VM exits and entries. To address this issue, modern hardware platforms have introduced the posted interrupt technique to enable exitless virtual interrupt delivery. They have also extended their interrupt controllers with specialized virtualization support to eliminate VM exits caused by ACK and EOI. Interrupt controllers with virtualization extensions are currently in production by all mainstream hardware vendors such as Intel, AMD, and ARM. Full-featured posted interrupt is available on the Intel platform, and it will soon be supported on other platforms (e.g., next-generation products with AMD AVIC [4] and ARM GICv4 [7]).

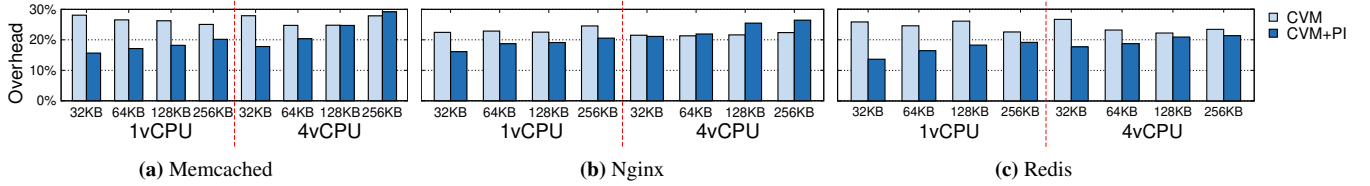


Figure 2: Normalized overhead compared with traditional VM in I/O-intensive applications. The Y-axis is the normalized overhead compared to the baseline of each group. CVM+PI represents CVM + Posted Interrupt.

3 Analysis of CVM-IO Tax

In this section, we quantify the performance impact of the CVM-IO tax by comparing the I/O performance of existing CVMs with that of traditional VMs. The CPU execution time of I/O-intensive applications running in a CVM can be divided into two parts: 1) *Application workloads*: the time spent on executing application workloads, including business logic and payload processing. 2) The *CVM-IO tax*: the time spent on CVM-specific security protections and intrinsic network I/O procedures. It consists of VM exits, the bounce buffer mechanism, and the packet processing during the payload preparation for application workloads.

All experiments are conducted on a 128-core AMD SEV-ES/SNP server and a 24-core Intel server with 200Gbps NICs. The AMD server is used to evaluate the I/O performance of real CVMs, referred to as *CVM*. However, the AMD server does not support posted interrupt, resulting in degraded performance due to numerous VM exits during virtual interrupt deliveries. Therefore, we simulate next-generation CVMs using the Intel server that supports posted interrupt, named *CVM+PI*. More detailed testbed and simulation configurations are described in § 7.1. For fair performance comparison, *CVM*'s baseline is the vanilla AMD traditional VM, while *CVM+PI*'s baseline is the vanilla Intel traditional VM.

To achieve optimal network performance, we choose vhost-user as the network backend in follow-up experiments. We still use the SEV-ES VM because the SEV-SNP VM does not support vhost-user due to its lack of huge page support. Theoretically, the security protections introduced by SEV-SNP do not further increase the CVM-IO tax.

3.1 CVM-IO Tax Breakdown

We first evaluate the overall performance using three representative network-intensive applications: Memcached and Redis for key/value stores, and Nginx for web servers. All applications and benchmarks enable the in-kernel TLS support for end-to-end protection. Figure 2 depicts the normalized performance overhead of CVMs compared to their respective baselines. In all three benchmarks, *CVM* incurs 21%-28% overhead, while *CVM+PI* exhibits 13%-29% performance degradation. As a result, the performance impact of CVM-IO tax results in significant overhead over baselines.

We further take the Memcached benchmark with the 4vCPU-256KB test cases as an example to break down the

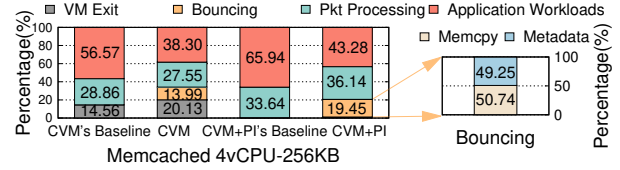


Figure 3: CPU time breakdown of the Memcached 4vCPU-256KB case in CVMs. The left subfigure shows the CPU time breakdown of *CVM*, *CVM+PI* and their baselines in the Memcached 4vCPU-256KB case. The right subfigure shows the CPU time breakdown of the bounce buffer part in *CVM+PI*.

time cost of the CVM-IO tax of CVMs, as shown in Figure 3. In the left subfigure, the *VM Exit*, *Bouncing* and *Pkt Processing* denote the three corresponding parts of the *CVM-IO tax* mentioned earlier. Because all vCPUs are fully utilized during the benchmark, the impact on overall performance becomes more significant as the percentage of CPU time consumed by the CVM-IO tax grows. The CVM-IO tax consumes over 50% of the total CPU time in all CVMs during the benchmark. For *CVM*, VM exits taking up more than 20% CPU time have more impact than the bounce buffer. For *CVM+PI*, the bounce buffer cost occupies more than 19% CPU time, while VM exits take up a tiny percentage of the total time in both *CVM+PI* (less than 1.2%) and its baseline (less than 0.5%). Packet processing in all the above cases accounts for about 30% of CPU time and thus has a considerable impact.

The overhead over baselines can be attributed to the reduction in CPU time of application workloads due to the CVM-IO tax. For example, in the 4vCPU-256KB case of *CVM+PI*, the CVM-IO tax leaves 34.35% fewer cycles for application workloads than the baseline, explaining the 27.44% overhead. Since packet processing in both CVMs and their baselines consumes a similar portion (about 30%) of CPU time, VM exits and the bounce buffer in CVMs contribute the most to the overhead.

Take-away I

The CVM-IO tax that occupies more than half of total CPU time incurs a substantial performance impact on CVMs. VM exits and the bounce buffer are the primary sources of overhead over baselines.

Lengthy VM Exits The AMD SEV-ES hardware introduces protection for CPU states (e.g., registers) of each CVM against the untrusted hypervisor during VM exits. In contrast

to traditional VMs, this protection adds thousands of cycles to each VM exit. We first break down the VM exit handling cost during every virtual interrupt delivery, finding that, on average, *CVM* spends 7,476 cycles on guest-host world switches, whereas a traditional VM only spends 1,643 cycles. We then collect the number of VM exits per second during the Memcached 4vCPU-256KB benchmark for different CVMs. *CVM* averagely triggers 41,615 VM exits per second on each vCPU, while *CVM+PI* only triggers 2,803 VM exits per second on each vCPU, an order of magnitude less than *CVM*.

The results indicate that frequent VM exits taking up more than 20% of total CPU time have a significant impact on *CVM*. However, with posted interrupt support (*CVM+PI*), the impact of VM exits is almost negligible. Fortunately, all next-generation CVM platforms, including AMD SEV, Intel TDX and ARM CCA, support posted interrupt, so that the performance impact of VM exits can become minimal.

Take-away II

VM exits may take up a large portion of the CPU time of CVM due to their high frequency and latency, but their performance impacts can become minimal with the posted interrupt support on next-generation hardware.

Bounce Buffer To analyze the overhead of bounce buffers, we break down the CPU time spent on bounce buffers in the 4vCPU-256KB case of *CVM+PI* into two parts: copying I/O data (packets in this case) and maintaining metadata for buffer allocation and freeing. The breakdown result shown in the right subfigure of Figure 3 indicates that I/O data copy (corresponding to the *Memcpy*) consumes 50.74% of the bounce buffer time, while the metadata maintenance (corresponding to the *Metadata*) spends 49.25% of the bounce buffer time. Besides, the experimental results of small and large data sizes reflect that the performance impact of the bounce buffer rises as the data size increases.

Take-away III

The bounce buffer consumes a large percentage of CPU resources due to I/O data copying and metadata maintenance. It is necessary to avoid bouncing large-size I/O data to minimize the bounce buffer's performance impact.

Packet Processing Packet processing in both the frontend driver and the network stack consumes up to 36.14% of CPU time in Memcached 4vCPU-256KB cases. Since the packet processing time cost is proportional to the number of packets processed, the large number of packets in I/O-intensive scenarios can demand a significant amount of CPU resources.

Take-away IV

Packet processing occupies a large fraction of CPU time due to the large number of packets to be processed. Reducing the number of packets to be processed can mitigate its performance impact.

3.2 Summary

To sum up, our experiments have demonstrated that CVMs incur up to 29% overhead in I/O-intensive applications compared with traditional VMs. On the next-generation hardware with posted interrupt support, the tax of VM exits becomes negligible while the bounce buffer tax has a more significant impact on CVMs. Additionally, the packet processing tax consumes a great portion of CVMs' vCPU resources due to the large number of packets, which is also the case for traditional VMs. Therefore, it is essential to reduce the cost of the bounce buffer as well as the packet processing in CVMs to minimize the CVM-IO tax and achieve high network performance.

4 Overview

4.1 Design Goals

The primary goal of Bifrost is to reduce the paravirtual I/O network tax of existing CVM solutions while maintaining the same level of security guarantees. Besides, the design of Bifrost should be general enough to be easily applied to CVM solutions on various platforms, such as x86, ARM and RISC-V, and to support different host and guest OSes, including Linux, FreeBSD and Windows. Further, it is demanding that Bifrost should avoid intrusive modifications to existing software stacks and keep transparent to userspace applications in CVMs to make it practical for real-world scenarios.

4.2 Challenges

To reduce the CVM-IO tax, Bifrost should optimize the bounce buffer mechanism and the packet processing procedure. However, it is not easy to implement these optimizations due to the following two technical challenges:

C1: Out-of-place hardware encryption and decryption.

The ideal way to eliminate the bounce buffer mechanism for a network packet is to enable zero copy by maintaining the packet within the same memory region throughout its entire lifecycle. In addition, either the guest or the host should have exclusive access to the packet's memory region while processing it to ensure data security, necessitating memory security type switches at runtime. However, as mentioned in § 2.1, when a private page containing a packet is converted to a shared page (and vice versa), the packet in this page is lost and unable to be correctly passed to the hypervisor. Moreover, changing the security type of guest memory pages is too expensive to be a frequent operation on the I/O critical path.

C2: Costly packet pre-processing in the device driver.

Packet processing primarily operates on packet headers rather than payloads. To minimize the cost of packet processing, a commonly employed technique is to pre-process multiple small packets within the same flow into larger packets before submitting them to the network stack. Nonetheless, the virtual NIC driver still has to occupy large quantities of vCPU resources to handle massive small packets coming from the high-speed NIC.

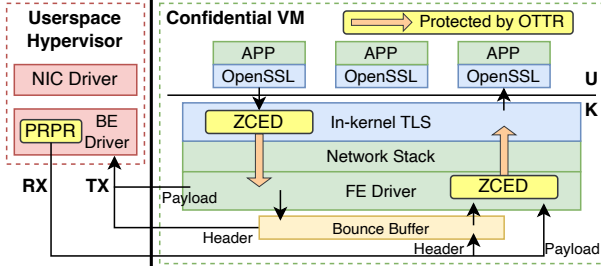


Figure 4: The overall architecture of Bifrost.

4.3 Observations and Insights

We observe three characteristics of existing CVM systems, allowing us to propose new designs that are appropriate for CVM scenarios to address the above challenges.

O1: Either private memory or end-to-end encryption alone is sufficient to assure data security. Data security can be ensured by either private memory protection or end-to-end encryption. Besides, it is not always better to apply multiple security protections to a piece of data at the same time, especially for performance-critical I/O data. Specifically, the guest OS in current CVM solutions initially encrypts the payload into private memory. But private memory protection is a redundant security mechanism for data that has already been encrypted. As a result, the payload bouncing tax can be eliminated by retaining existing end-to-end encryption while removing private memory protection at the same time.

O2: End-to-end encryption has the side effect of moving the memory location of the payload. The procedure of adding end-to-end encryption allows CVM to relocate payloads to a different memory location. Hence, end-to-end encryption at the in-kernel TLS layer provides an opportune moment to also remove private memory protection on the payload for userspace applications. In particular, the generated ciphertext during encryption can be directly written to the target shared memory with the host, ensuring data security while eliminating unnecessary copies and bouncing overhead.

O3: I/O backends usually have plenty of residual CPU resources available. Modern virtualization systems usually leverage dedicated CPUs to run I/O backends of VMs for high and predictable I/O performance [16, 45]. Unlike CPUs running CVMs' vCPUs, which are likely to be fully loaded due to complex in-guest logic, those running I/O backends have less work to do and thus have plenty of free CPU resources. As a result, I/O backends with adequate residual CPU resources can be utilized to release the burden of vCPUs by pre-processing packets before passing them to CVMs.

4.4 Architecture and High-Level Design

Based on the above observations, Bifrost leverages the side effect of end-to-end encryption and the residual CPU resources of network I/O backends to eliminate payload bouncing and reduce packet processing cost in CVMs in an application-transparent way. Figure 4 shows the architecture of Bifrost.

To address challenge C1, Bifrost proposes two designs to

enable zero-copy transparently and securely for the end-to-end encrypted payload in the CVM. **D1: zero-copy encryption deduplication** (§ 5.1) eliminates payload bouncing by keeping the end-to-end protected payload in the same shared memory during its lifetime. Specifically, Bifrost directly stores output from the in-kernel TLS layer to the guest-host shared memory without copying any payload, and vice versa. To minimize modifications, Bifrost creates dedicated NUMA nodes to serve as shared memory for this design, so that memory allocators in the guest kernel can be reused. However, concurrent memory accesses to plaintext data in shared memory may lead to TOCTTOU attacks. A malicious host can tamper with data that has passed the security checks of the guest OS, such as altering a packet header after it has passed the checksum check. To defend against this attack, Bifrost introduces **D2: one-time trusted read** (§ 5.2), which ensures that the guest OS can only read and trust the target data content from shared memory once, as additional reads from the same memory may lead to host-tainted data content. The guest must process data after it has been read into registers or private memory to defend against host tampering during guest processing, thus eliminating TOCTTOU issues.

To address challenge C2, Bifrost proposes another design to complete pre-processing network packets before they reach the CVM. **D3: pre-receiver packet reassembly** (§ 5.3) makes use of the network backend's free CPU resources to pre-process multiple small incoming packets into a large one before transmitting them to the guest OS.

As mentioned in § 3.2, while **D1** and **D2** are designed to optimize payload bouncing issues that are specific to CVMs, **D3** can also be leveraged to reduce packet processing cost in traditional VMs.

We explain the Bifrost architecture and its design points by describing the high-level workflows of packet receiving and sending. **Packet receiving workflow:** When a network packet carrying an end-to-end encrypted payload arrives at the network I/O backend, Bifrost attempts to merge it with other same-flow packets, if possible, by pre-processing the packet with PRPR (**D3**). Then Bifrost flushes those pre-processed network packets to the frontend driver through virtual network queues of the CVM. The zero-copy aware TOCTTOU defense (**D2**) in the frontend driver only copies small metadata such as packet headers to private memory for security, while keeping the end-to-end encrypted payload in the shared memory allocated from dedicated NUMA nodes (**D1**). Next, the frontend driver constructs basic data structures (e.g., *skbuff* in Linux) for these pre-processed incoming packets before passing them to the network stack. Afterwards, Bifrost utilizes the in-kernel TLS support to decrypt the end-to-end encrypted payload directly from shared memory into the application's private memory. As a result, the packet receiving workflow experiences no end-to-end encrypted payload bouncing and less packet processing cost in the CVM.

Packet sending workflow: When an application begins to

send out a payload from the guest OS, Bifrost first leverages the in-kernel TLS support to encrypt plaintext from either application memory or kernel page cache in private memory and places the encrypted result directly into the guest-host shared memory allocated from dedicated NUMA nodes (D1). The memory copy of the end-to-end encrypted payload from private memory to the shared bounce buffers is removed at this step. For small metadata that is not protected by end-to-end encryption, the zero-copy aware TOCTTOU defense (D2) enforces Bifrost to fall back to the bounce buffer mechanism. Consequently, there is no end-to-end encrypted payload bouncing in its sending workflow.

4.5 Threat Model and Assumptions

The threat model of Bifrost is the same as that of existing CVM solutions. The TCB only comprises the CPU hardware and minimized trusted monitor firmware or software, if any. Attackers can control any untrusted software entities or hardware devices to launch attacks on CVMs. Therefore, for a specific CVM, all software outside it, including the hypervisor and other CVMs, and hardware devices, are untrusted. We assume that a CVM does not voluntarily reveal its sensitive data and protects its I/O data with end-to-end encryption. Denial-of-Service (DoS) attacks [11] are out of scope. Although CVM implementations may have bugs [5, 41] and are subject to side-channel attacks [12, 39, 40, 47], we do not consider them because they are orthogonal to this paper.

5 Design and Implementation Details

5.1 Zero-Copy Encryption Deduplication (ZCED)

Bifrost reserves a contiguous shared memory region for paravirtual I/O networking in the guest physical address (GPA) space. This shared memory appears as NUMA nodes dedicated for ZCED (hereinafter called ZCED NUMA), allowing Bifrost to utilize mature memory management mechanisms in existing guest OSes. Moreover, the location and size of ZCED NUMA memory are fixed at the boot time of a CVM for optimal performance.

Boot-time initialization: The memory range of a ZCED NUMA node can be configured by setting the base GPA and total length via the kernel command line. As shown in § 7.4, ZCED NUMA nodes of 200MB can satisfy the demands of all network-intensive benchmarks in our experiments. Bifrost parses the number of ZCED NUMA nodes and adds the specified guest memory range to each node. All ZCED NUMA nodes are created with no associated vCPU. Before a ZCED NUMA node is available for memory allocations, Bifrost sets its memory security type to shared. To achieve optimal performance, proper distances should be specified between NUMA nodes to assist the guest kernel in allocating memory [36]. The distances between ZCED NUMA nodes are the same as those between normal NUMA nodes to which their memory

ranges originally belonged, while each ZCED NUMA node is zero distance from its original NUMA node.

Runtime allocation: To prevent data leakage caused by unintentional data store into the ZCED NUMA memory, Bifrost adjusts the memory allocation policies of the guest OS to only allow explicit allocation to acquire ZCED NUMA memory. Hence, the original memory allocations in the system do not allocate from ZCED NUMA nodes, avoiding the security issue of inadvertently exposing sensitive data. Guest kernel components are merely able to allocate memory from ZCED NUMA nodes by assigning a special allocation flag (e.g., a *GFP* flag in Linux) provided by Bifrost to parameters of memory allocation invocations. The allocator will first try to acquire memory from the closest ZCED NUMA node to the vCPU running this component. In the frontend driver, Bifrost checks the memory location in which the payload resides, and if it belongs to a ZCED NUMA node, Bifrost will bypass the bounce buffer mechanism.

In the TX direction, Bifrost modifies the in-kernel TLS layer to transparently intercept communications between upper applications and the lower network stack. The payload in the TX direction must go through *sendmsg* and *sendpage* functions of the existing in-kernel TLS layer before entering the network stack. *sendmsg* is the most often used function for sending payload from userspace, whereas *sendpage* is specialized for transferring payload from the storage (e.g., page cache). Bifrost just adds the special allocation flag to the parameters of memory allocation invocations in these two functions to allocate memory from ZCED NUMA nodes for storing encrypted payload.

In the RX direction, the guest memory regions used to accept incoming packets are allocated and assigned by the frontend driver (i.e., virtio-net in our case). Bifrost modifies the memory allocation invocations for these regions by adding the allocation flag as well. When an application attempts to receive payload, Bifrost decrypts the ciphertext directly from the ZCED NUMA memory to private memory.

5.2 One-Time Trusted Read (OTTR)

To defend against TOCTTOU attacks, Bifrost only trusts the data obtained from the first read of the ZCED NUMA memory during the guest OS's handling of packet headers and end-to-end encrypted payload.

Packet header handling: The content of each packet header should only be used after it has been validated by the guest OS's packet processing functions. However, if a malicious host modifies the header after the guest OS's validation, the guest OS may encounter problems due to the invalid header. For instance, buffer-overflow problems can happen if the guest OS uses a modified length to extract payload from packets.

To prevent this, Bifrost must read a packet header from the ZCED NUMA memory into a private memory region before further processing it. This read only happens once for each packet header, and Bifrost will never read the header from

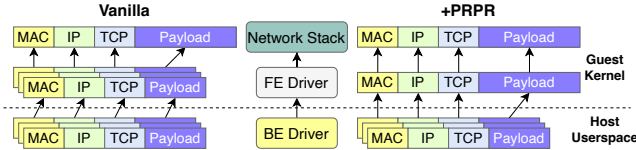


Figure 5: Comparison of the packet reassembly workflow of the vanilla CVM and the CVM with PRPR.

the ZCED NUMA memory again to prevent the host from subsequently tampering with the content of packet headers. In the TX direction, unlike the end-to-end encrypted payload that is stored in the ZCED NUMA memory, network packet headers are placed in private memory. The frontend driver still leverages the bounce buffer mechanism to copy packet headers to the guest-host shared memory before transmitting them to the backend driver. It has a very small impact on performance due to the small size of packet headers.

End-to-end encrypted payload handling: For end-to-end encrypted payload, as mentioned in § 2.3, data security protection requires that the encryption must generate both correct ciphertext (i.e., encrypted payload) and authenticated tag, and the decryption must correctly verify the ciphertext integrity with the authenticated tag. Both the authenticated tag generation and the integrity verification take the ciphertext as input, which exists in ZCED NUMA memory and may be tampered with by the host, resulting in compromised payload integrity. For instance, in the context of zero-copy I/O, the current Linux AES-GCM implementation on the x86-64 platform double reads the ciphertext from the same ZCED NUMA memory in the last phase of parallel decryption, suffering from TOCTOU attacks on the ciphertext.

To prevent payload TOCTOU attacks, in the decryption procedure, Bifrost reads only once from the ZCED NUMA memory to load the ciphertext value into CPU registers and always uses the correct ciphertext in the registers afterwards, avoiding reading a potentially compromised ciphertext. Similarly, during the encryption procedure, the ciphertext for each payload is guaranteed to remain valid from the moment it is generated in the register until it exits the register. Thus, Bifrost calculates the authentication tag using the correct ciphertext that is still in the CPU registers.

5.3 Pre-receiver Packet Reassembly (PRPR)

Large packets are split into smaller ones before sending out due to transmission size limit. To save CPU resources consumed by packet handling, prior work [33, 52] has decreased the number of packets passed to the network stack by reassembling small packets into large ones in advance. Modern OSes support small packets coalescing at the device driver layer using GRO [15]. However, packet reassembly in the guest device driver can still consume significant vCPU resources, severely affecting the application performance when handling large numbers of packets. While modern NICs enable hardware coalescing without engaging CPU using LRO [23], it is hard for hardware to dynamically adjust reassembly rules and

support new packet formats. Inappropriate coalescing even causes metadata loss and network connection disruptions [15].

In comparison to prior approaches, Bifrost offloads the packet reassembly to the hypervisor backend driver which has sufficient CPU time, freeing up precious vCPU resources for CVMs without sacrificing flexibility. The packet reassembly logic in Bifrost is similar to that of previous work [15] since network packets share the same format.

Overall procedure: When a network packet arrives at the network backend, some packets may be cached in the backend and waiting for reassembly. Bifrost first parses the current packet header to determine if any cached packets from the same flow exist. If present, Bifrost tries to merge the current packet with the cached same-flow ones. Eventually, based on the status information in the currently cached packets in the network I/O backend, Bifrost decides whether it is time to flush them to the frontend driver in the guest OS.

Same-flow packet detection: Same-flow packets are network packets that share the same source, destination and sequence number. As our current implementation focuses on TCP/IP packets, Bifrost first recognizes headers that have the same MAC address, IP address and TCP port in both source and destination directions as same-flow candidates. Then Bifrost regards these candidates with an identical TCP acknowledgment (ACK) number as same-flow packets.

Flexible per-VM flush rules: It is essential to flush packets to the guest OS at an appropriate time since the network performance is highly sensitive to packet latency. When a newly received packet has a cached same-flow packet, Bifrost first checks whether these two packets have consistent status information, such as the time to live (TTL) field. If not, Bifrost flushes the old cached packet to the frontend driver. Otherwise, Bifrost reassembles these two packets into a new one. Finally, Bifrost flushes the new packet if it contains an immediate-flush flag (e.g., the TCP PSH flag). For a received packet that has no same-flow packet, Bifrost directly determines whether to flush it by checking its immediate-flush flag.

In addition to the above basic rules, Bifrost also allows each guest OS to customize flush rules. Bifrost provides paravirtual interfaces for receiver CVMs to install their own rules to disable reassembly, adjust the maximum number and timeout of cached packets.

Packet reassembly: Among the cached same-flow packets, the currently received packet can only be reassembled with the packet whose payload is contiguous with it. Bifrost finds neighbors of the received packet for reassembly by comparing their TCP sequence (SEQ) numbers. As depicted in Figure 5, duplicate packet headers are merged during reassembly.

6 Implementation Complexity

We implement a Bifrost prototype using Linux as the guest kernel and OpenvSwitch-DPDK as the network I/O backend.

In the Linux v6.0-rc1, we introduce 815 lines of code to support ZCED and OTTR. These changes include initializing

ZCED NUMA nodes during memory subsystem bootstrapping, replacing memory allocation invocations, and defending against TOCTTOU risks in AES-GCM assembly.

In the DPDK v21.11.2, we add 541 lines of code to implement PRPR, which primarily consists of two parts: 1) Reorganization of network packets, including header trimming during packet reassembly, flag resetting, etc. 2) Flush rules, which mainly focus on deciding whether to cache or immediately flush an incoming packet. Our implementation also adds 175 lines of code to OpenvSwitch v2.17.3, which pre-processes the network packets by parsing headers in advance, and invokes the interfaces provided by the DPDK.

7 Evaluation

7.1 Experimental Setup

Testbed: Our testbed remains the same as in § 3, consisting of an AMD server and an Intel server running Ubuntu 20.04.4 LTS. The AMD server has two 64-core AMD EPYC 7T83 CPUs at 2.45GHz (128 cores in total) and 500GB DDR4 DRAM. The Intel server has two 12-core Intel Xeon Gold 5317 CPU at 3.00GHz (24 cores in total) and 188GB DDR4 DRAM. Both machines are equipped with one single-port Mellanox Connect-X6 200Gbps NIC and are back-to-back connected with a fabric cable. We disable CPU frequency boost features to lessen performance data fluctuation. The AMD server’s host kernel is Linux v5.19.0-rc6 with SEV-ES and SEV-SNP support, while the Intel server’s host kernel is Linux v5.4.0. The guest kernel version of all CVMs and their baseline VMs is Linux v6.0-rc1. Each guest OS is assigned with either 1 vCPU or 4 vCPUs, 16GB memory and a 2-virtqueue virtio-net device backed by the vhost-user backend based on OpenvSwitch v2.17.3 and DPDK v21.11.2. For each benchmark, the server side runs in the guest OS while the client side runs in the host OS on the other server.

To avoid the interference of unintended scheduling or interrupts, we isolate 6 cores on each server. The CPU isolation is achieved by the *isolcpus* function in the Linux kernel, and the binding is done by the *qemu-affinity* command for vCPUs and *pmd-cpu-mask* parameter for the OpenvSwitch-DPDK-based vhost-user backend. Each thread of vCPUs and the vhost-user backend is pinned to a different isolated CPU. IOMMUs of both machines are set to passthrough mode.

Naming Convention and Configurations: As mentioned in § 3, *CVM* represents real SEV-ES/SNP CVMs on the AMD server, while *CVM+PI* represents simulated TDX CVMs with posted interrupt enabled on the Intel server. The simulation is based on a vanilla Intel traditional VM, which further enables bounce buffer (i.e., Linux SWIOTLB [62]) for virtio devices and adds an additional 10,000 cycles to each VM exit to simulate the cost of guest-host world switches. The simulated world switches consume 2,524 more cycles than that of the SEV-ES/SNP VM. To the best of our knowledge, SEV-ES/SNP’s lengthy VM exits primarily result from uncore co-processor (i.e., AMD-SP) intervention, whereas TDX’s

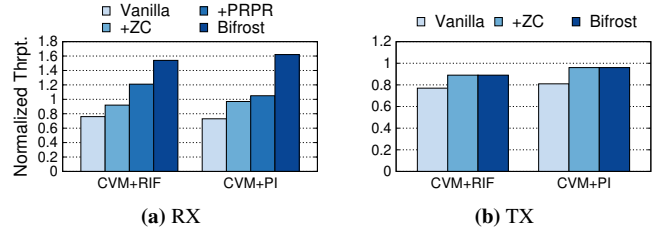


Figure 6: TLS normalized throughput on both AMD and Intel server. The Y-axis is the normalized throughput compared with the baseline. +PRPR is not shown in (b) because it does not provide benefits for TX performance.

VM exits solely involve in-core firmware, leading to lower latency. As a result, though the increased VM exit latency does not guarantee identical performance to real Intel TDX hardware, the simulated VM exit overhead should be no less than that of real TDX VMs.

To get the performance of *CVM* close to the SEV-ES VM with posted interrupt, we optimize AMD *CVM* with reduced VM exit frequency by modifying the network backend to lower its notification frequency to the guest OS. We call the optimized version *CVM+RIF*, signifying *CVM + Reduced Interrupt Frequency*. *CVM+RIF*’s baseline is the vanilla AMD traditional VM with reduced VM exit frequency, while *CVM+PI*’s baseline is the vanilla Intel traditional VM. To show the individual contribution of Bifrost’s each technique, +ZC denotes only applying the ZCED as well as the OTTR, while +PRPR indicates adding the PRPR alone. The PRPR is configured to cache up to 1024 TCP flows for 2 virtqueues. Each flow can hold up to 1024 packets, and at most 32 packets can be submitted to the I/O frontend simultaneously.

7.2 Performance Improvement

In this section, we focus on the performance improvement of *CVM+RIF* and *CVM+PI*. We first build a microbenchmark to investigate the upper bound on the performance improvement that Bifrost can bring to I/O-intensive applications and then study the performance gains of real-world applications from Bifrost’s design. Since the posted interrupt hardware has been able to minimize the performance impact of VM exits, we concentrate on Bifrost’s effect on CVMs atop such hardware.

7.2.1 Microbenchmark

We develop a TCP-based TLS client/server pair to evaluate the network throughput. They simply contain simple code for single-threaded I/O data sending and receiving. This minimizes the time cost of business logic, demonstrating the maximum possible application performance improvement. To fully saturate the vCPU like an I/O-intensive application, we run 4 TLS server instances in a 1-vCPU VM.

RX Throughput: Figure 6a shows the network throughput comparisons in the RX direction. *CVM+RIF* attains 5.26 Gb/s, which is 24.10% slower than its baseline’s 6.93 Gb/s. With the ZCED, +ZC alone (6.38 Gb/s) can reduce the slowdown to

7.81%. With the PRPR, +PRPR itself (8.39 Gb/s) can outperform the baseline by 21.10%. By combining both techniques, Bifrost reaches 10.64 Gb/s, which is 53.55% higher than the baseline. For CVM+PI (7.48 Gb/s), it incurs 27.03% overhead than its baseline (10.26 Gb/s). The throughput is increased to 9.99 Gb/s in +ZC (2.59% overhead) and grows to 10.76 Gb/s in +PRPR (4.95% better). Integrating both techniques makes Bifrost reach 16.64 Gb/s, which is 62.20% higher than the baseline. Therefore, Bifrost can boost performance by up to 89.23% (from 27.03% overhead to 62.20% better than the baseline) for applications experiencing high RX traffic in existing CVMs.

TX Throughput: Figure 6b shows the throughput comparisons in the TX direction. CVM+RIF attains 9.59 Gb/s, which is 23.03% slower than its baseline’s 12.45 Gb/s. +ZC (11.04 Gb/s) reduces the slowdown to 11.37%. Bifrost has 10.79% overhead (11.11 Gb/s), slightly better than +ZC. Experiments of CVM+PI yield similar results. Therefore, Bifrost can have up to 12.24% (CVM+RIF) and 15.00% (CVM+PI) performance improvement for applications with high TX traffic.

Combining +ZC and +PRPR in the RX direction shows a greater performance improvement than the sum of each technique’s individual improvement (explained in § 7.2.2). The TX improvement is less significant because PRPR only optimizes RX traffic. Limited CPU resources on a single vCPU for both packet processing and TLS operations result in a large gap from reaching the NIC’s maximum bandwidth.

7.2.2 Applications

We utilize the same network-intensive applications as in § 3 to evaluate and break down the performance improvement of Bifrost. TLS/SSL is enabled in all the applications. We run each benchmark for 30 seconds and report the average value of the results from 10 rounds. To save space, we only present and analyze the results of the 32KB and the 256KB data sizes in detail, and provide an overview of the results for other data sizes. The detailed benchmark configurations and results are shown below.

Memcached [20] is a popular multi-threaded in-memory key-value store application. We use the *memtier_benchmark* [56] tool to measure throughput and average latency. The *Memcached* server is configured with either 1 or 4 threads for VM with 1 or 4 vCPUs respectively, and 4096MB memory. We set up 4 clients, 16 concurrent requests for 1-thread server and 8 clients, 32 concurrent requests for 4-thread server.

Figure 7a and Figure 8a show the throughput and latency overhead, respectively, of Memcached in CVM+RIF. Both throughput and latency improve as a result of alleviating the vCPU bottleneck. In 32KB cases, Bifrost cuts down more than half of CVM+RIF’s overhead over its baseline. Either +ZC or +PRPR alone slightly mitigates the overhead. In 256KB cases, Bifrost performs about 10% better than its baseline. Either +ZC or +PRPR alone reduces the overhead by more than half. With the same number of vCPUs, Bifrost’s performance im-

provement increases as the data size grows. This is primarily because the performance impact of the CVM-IO tax, especially the bounce buffer tax, becomes more pronounced with the growth of data size, providing more room for improvement.

Figure 9a displays the time breakdown and backend utilization of CPUs in 4vCPU-256KB cases. The ZCED reduces the total timeshare of the bounce buffer tax from 15.67% to less than 2.50%. It cannot completely eliminate the bounce buffer cost because some small I/O data (e.g., TCP handshake packets) still falls back to the bounce buffer. The PRPR reduces the timeshare of the packet processing tax from 28.73% to 21.88%. Bifrost spends 2.39% more time on application workloads than the baseline and has more than 10% speed gain on the TLS processing in application workloads, which explains the 8.26% improvement over the baseline. Due to the higher throughput and PRPR cost, Bifrost’s backend CPU utilization increases by 8.75% compared to the baseline.

Figure 7d and Figure 8b show the throughput and latency overhead, respectively, of Memcached in CVM+PI. Bifrost outperforms the baseline in all cases. The throughput acceleration over the baseline can reach 3.06% in 32KB cases and 21.50% in 256KB cases. The latency overhead is almost eliminated in 32KB cases and can outperform the baseline by 17.45% in 256KB cases. +ZC obtains more individual performance gain than +PRPR, and their combined improvement is larger than the sum of their individual gains. This is because applying both techniques can provide more available CPU cycles to application workloads than applying only one of them. As shown in Figure 9a, when only applying PRPR, part of the released CPU cycles will be occupied by bouncing packets.

Figure 9b depicts the breakdown and backend utilization of CPUs in 4vCPU-256KB cases. The ZCED reduces the total timeshare of the bounce buffer tax from 19.45% to less than 2.77%. The PRPR reduces the timeshare of the packet processing tax from 36.14% to 26.83%. Compared to the baseline, Bifrost provides application workloads with 11.53% more CPU time and more than 10% speedup in TLS processing. This can explain the 21.50% improvement over the baseline. Due to the higher throughput and PRPR cost, Bifrost’s backend CPU utilization is 19.05% more than the baseline.

Nginx [50] is a well-known high-performance HTTP(S) web server. We run the *wrk* [21] benchmark tool to measure the throughput represented by requests per second (RPS). The client configurations are similar to the other two applications.

Figure 7b illustrates the Nginx throughput overhead of CVM+RIF. Since the traffic type of the Nginx benchmark is mainly in the TX direction, the majority of Bifrost’s performance improvement comes from the ZCED, as analyzed in § 7.2.1. +ZC reduces the overhead by less than half because lengthy VM exits still significantly impact performance. The PRPR even increases the overhead for a little bit in the 1vCPU-256KB case, because there are more VM exits after

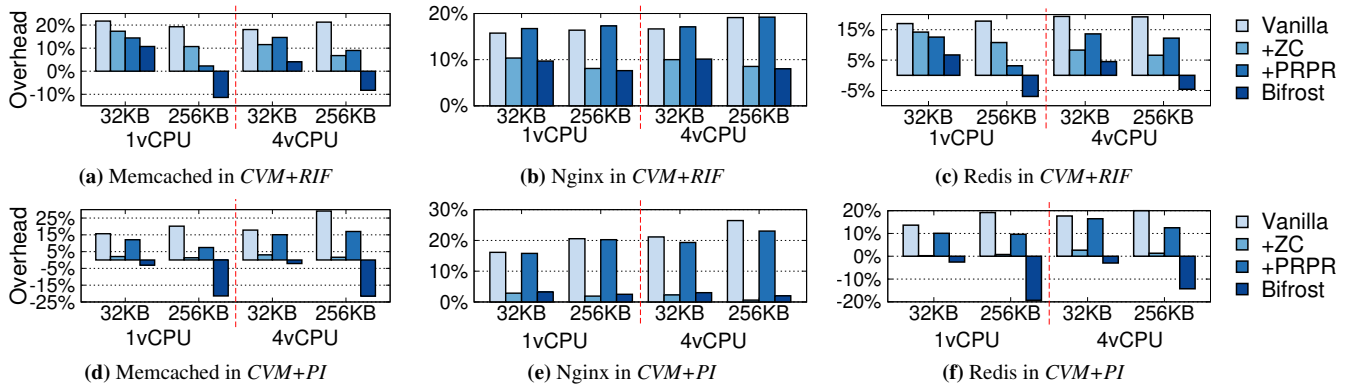


Figure 7: Application performance comparisons when applying some or all of Bifrost’s techniques. The Y-axis indicates relative overhead compared with baseline VMs, negative overhead represents performance improvement. (a), (b) and (c) compare throughput of *CVM+RIF* on the AMD server. (d), (e) and (f) compare throughput of *CVM+PI* on the Intel server.

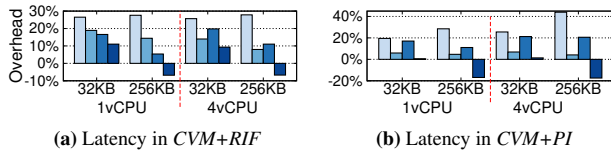


Figure 8: Memcached average latency comparisons when applying some or all of Bifrost’s techniques. The Y-axis indicates relative overhead compared with baseline VMs, negative overhead represents performance improvement.

the PRPR is applied. Figure 7e shows the Nginx throughput overhead of *CVM+PI*. Bifrost brings the overhead to less than 2.8% in all cases, thanks to the ZCED. The PRPR no longer impacts the performance negatively because VM exits cost is trivial when posted interrupt is enabled.

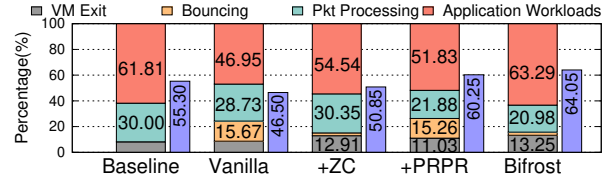
Redis [28] is a single-threaded in-memory key-value store application widely deployed in production environments. We also use the *memtier_benchmark* tool to measure the throughput. The *Redis* server is configured with 4096MB memory. The *memtier_benchmark* uses the same configurations as *Memcached*. To fully utilize vCPU resources in 4vCPU cases, we use *redis-cli* to build a *Redis* cluster with 4 instances.

Figure 7c and Figure 7f present the *Redis* throughput overhead, which have similar patterns to those of *Memcached*.

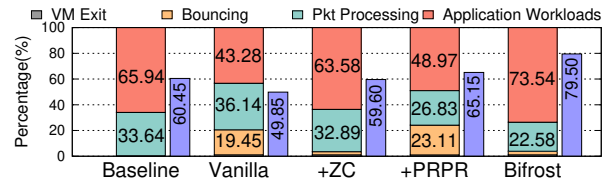
VM Scalability: To show Bifrost is scalable as the number of VM grows, we evaluate applications in 1, 2 and 4 Bifrost-enabled 4-vCPU 2-virtqueue *CVM+RIF*s. We conduct experiments on the AMD server because it has sufficient CPU cores on a single NUMA node. Figure 10 demonstrates that in multi-VM cases, Bifrost can always achieve comparable performance improvements to that of the single VM scenario. Bifrost’s good VM scalability comes naturally because ZCED uses Linux’s scalable memory allocator and PRPR is applied to each virtqueue without contention.

7.3 TOCTTOU Protection Overhead

Bifrost defends the guest OS against TOCTTOU attacks with OTTR by copying packet headers into private memory and keeping end-to-end encrypted payload in registers during



(a) Breakdown of *CVM+RIF* and its baseline & Backend CPU utilization



(b) Breakdown of *CVM+PI* and its baseline & Backend CPU utilization

Figure 9: VM CPU time breakdown and backend CPU utilization in Memcached 4vCPU-256KB experiments on AMD and Intel servers. The Y-axis represents the percentage of CPU cycles. In each case, the left side shows the breakdown of CPUs that run VMs, while the right side presents the utilization of backend CPUs.

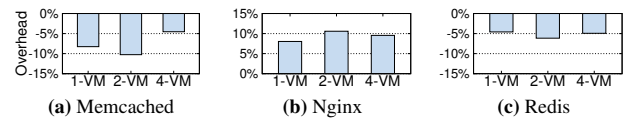


Figure 10: Performance comparisons of Bifrost in 1, 2 and 4 *CVM+RIF* VMs using Memcached 4vCPU-256KB experiments. The Y-axis indicates relative overhead compared with baseline VMs, negative overhead represents performance improvement.

their processing. To evaluate the performance impact of these operations, we implement a prototype of Bifrost without applying OTTR, called Bifrost-noprot. We repeat application benchmarks to compare Bifrost’s performance with that of Bifrost-noprot. The overhead of Bifrost in different benchmarks is shown in Figure 11, indicating no more than 2.0% overhead caused by TOCTTOU protections in all cases.

7.4 Memory Footprint

Bifrost must utilize memory efficiently to avoid unavailability due to the depletion of the ZCED NUMA memory. We first

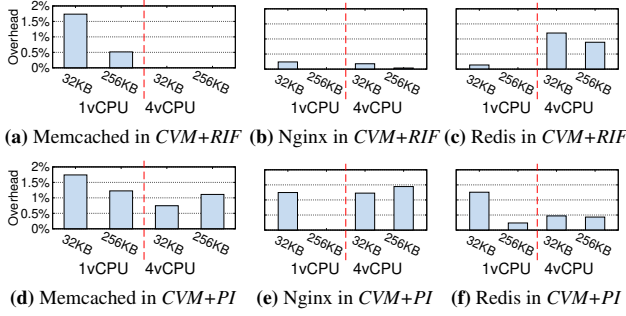


Figure 11: Application performance overhead of TOCTTOU protection. (a), (b) and (c) compare Bifrost with Bifrost-noprof (i.e., w/o OTTR) on the AMD server. (d), (e) and (f) compare Bifrost with Bifrost-noprof on the Intel server.

measure the ZCED NUMA’s memory consumption via the *numastat* tool. Among all of our benchmarks, Bifrost consumes no more than 200MB out of the 512MB capacity of the ZCED NUMA memory. This amount does not exceed the default 1GB size of the bounce buffer area in the 16GB *CVM+RIF* VM. As for the footprint in the network backend, PRPR maintains a packet cache list in the backend memory for each virtqueue. Each cache list contains at most 8 network flows, each caching at most 1024 network packets. The maximum memory cost of one cache list is 1,088KB. In our benchmarks, we enable 2 virtqueues, consuming only 2.125MB memory.

8 Security Analysis

Bifrost introduces three major techniques to existing CVMs, in which only the ZCED and the OTTR retrofit the guest kernel and may have an impact on the network I/O data security. The only difference between the network I/O of Bifrost and a vanilla CVM is that Bifrost needs to process packets in the guest-host shared memory, while a vanilla CVM handles packets in private memory. Thus, we only need to analyze the security risks caused by TOCTTOU attacks on network packets during network I/O.

Headers: In the RX direction, a header is received in the guest-host shared memory. Bifrost copies the header into private memory, and subsequent header processing only uses the private copy, which does not suffer from TOCTTOU attacks. In the TX direction, each header is born in private memory and sent out through the bounce buffer mechanism, which is the same as in the existing CVMs.

Encrypted payload: In the RX direction, the in-kernel TLS layer decrypts the encrypted payload from the guest-host shared memory into private memory. Bifrost ensures that the decryption code has a consistent view of the encrypted payload by reading from shared memory only once and keeping it in CPU registers. In the TX direction, the in-kernel TLS layer encrypts the plaintext payload directly into shared memory. Bifrost ensures that the encryption code always refers to the correct ciphertext in CPU registers, which is isolated by CVM platforms and immune to TOCTTOU attacks.

Plaintext payload: There are also packets carrying plaintext

payload due to procedures such as handshaking. In the RX direction, the plaintext payload is not accessed until the guest kernel copies it from shared memory to private memory. In the TX direction, the plaintext payload is no longer accessed once the guest kernel copies it from private memory to shared memory. Avoiding shared memory access eliminates the risks of TOCTTOU vulnerability.

Therefore, Bifrost does not expose guest OS’s network processing to TOCTTOU attacks, achieving the same level of security guarantees as vanilla CVMs.

9 Related Work

Secure Virtualized Systems. A long line of research works and commercial products have been proposed to build secure virtualized systems [3, 8, 10, 11, 26, 32, 38, 48]. AMD SEV [2, 3], Intel TDX [29, 32] and ARM CCA [8] enable the CVM abstraction with hardware extensions, especially memory encryption and integrity protection [30, 34]. While AMD SEV relies on a secure processor [1], Intel TDX and ARM CCA employ trusted firmware [31] to manage CVMs. TwinVisor [38] provides a TrustZone-based alternative to ARM CCA by retrofitting the virtualization extension on existing ARM platforms. The design of Bifrost is not restricted to AMD or Intel and can be applied to other CVM systems.

Zero Copy I/O. Prior research works have proposed various techniques to eliminate data copies for better I/O performance [24, 27, 35, 42–44]. For user-level applications, zIO [58] can transparently remove redundant I/O copies. For the I/O stack in the kernel, DAMN [44] and Demikernel [63] eliminate I/O memory copies by directly allocating buffers from the I/O memory pool. PASTE [25] performs DMA directly into non-volatile memory to avoid copies. Unlike these systems that target traditional scenarios and/or require intrusive software modifications, Bifrost focuses on eliminating unnecessary I/O data copies in CVMs with minor modifications.

10 Conclusion

This paper presents the first systematic analysis of the CVM-I/O tax for network-intensive workloads in CVMs. To optimize the I/O performance of CVMs, we propose a new paravirtual I/O design called Bifrost. Bifrost eliminates redundant packet bounces and greatly reduces packet processing cost, while maintaining the same level of security guarantees as existing CVMs. Evaluation results show that Bifrost significantly improves the I/O performance of CVMs, and even outperforms traditional VMs by up to 21.50%.

11 Acknowledgments

We express our sincere gratitude to our shepherd Kartik Gopalan, whose valuable suggestions have greatly improved our paper. We thank the anonymous reviewers for their insightful suggestions. This work was partially supported by NSFC (No. 62002218 and 61925206), the Fundamental Research Funds for the Central Universities, NSFC (No. 62132014 and 62141218) and Huawei Innovation Research Plan.

References

- [1] BlackHat 2020. All you ever wanted to know about the AMD Platform Security Processor and were afraid to emulate. <https://i.blackhat.com/USA-20/Wednesday/us-20-Buhren-All-You-Ever-Wanted-To-Know-About-The-AMD-Platform-Security-Processor-And-Were-Afraid-To-Emulate.pdf>, 2020.
- [2] AMD. Protecting VM Register State With SEV-ES. <https://www.amd.com/system/files/TechDocs/Protecting%20VM%20Register%20State%20with%20SEV-ES.pdf>, 2017.
- [3] AMD. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. <https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf>, 2020.
- [4] AMD. AMD64 Architecture Programmer's Manual Volume 2: System Programming. <https://www.amd.com/system/files/TechDocs/24593.pdf>, 2022.
- [5] AMD. AMD64 Architecture Programmer's Manual Volume 2. <https://www.amd.com/en/corporate/product-security/bulletin/amd-sb-1021>, 2023.
- [6] BEN ARENT. Securing MySQL Databases with SSL/TLS. <https://goteleport.com/blog/secure-database-with-tls/>, 2022.
- [7] ARM. What are key features in GICv3.x and GICv4.x? <https://developer.arm.com/documentation/ka004701/latest>, 2022.
- [8] ARM. ARM Confidential Compute Architecture. <https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture>, 2023.
- [9] Microsoft Azure. Azure AI. <https://azure.microsoft.com/en-us/solutions/ai/#overview>, 2023.
- [10] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 267–283, Broomfield, CO, October 2014. USENIX Association.
- [11] Jiahao Chen, Dingji Li, Zeyu Mi, Yuxuan Liu, Binyu Zang, Haibing Guan, and Haibo Chen. Security and Performance in the Delegated User-level Virtualization. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, Boston, MA, July 2023. USENIX Association.
- [12] Sanchuan Chen, Fangfei Liu, Zeyu Mi, Yinqian Zhang, Ruby B. Lee, Haibo Chen, and XiaoFeng Wang. Leveraging Hardware Transactional Memory for Cache Side-Channel Defenses. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, ASIACCS '18*, page 601–608, New York, NY, USA, 2018. Association for Computing Machinery.
- [13] Google Cloud. Memorystore for Memcached overview. <https://cloud.google.com/memorystore/docs/memcached/memcached-overview>, 2023.
- [14] Google Cloud. Memorystore for Redis overview. <https://cloud.google.com/memorystore/docs/redis/redis-overview>, 2023.
- [15] Jonathan Corbet. JLS2009: Generic receive offload. *Linux Weekly News (LWN)*, 2009.
- [16] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arfin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauch Zermeno, Erik Rubow, James Alexander Docauer, Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc De Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat. Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation, NSDI'18*, page 373–387, USA, 2018. USENIX Association.
- [17] Docker. Protect the Docker daemon socket. <https://docs.docker.com/engine/security/protect-access/>, 2022.
- [18] Morris J Dworkin. *NIST Special Publication 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. National Institute of Standards & Technology, 2007.
- [19] Jake Edge. TLS in the kernel. <https://lwn.net/Articles/666509/>, 2015.
- [20] Brad Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.
- [21] Will Glozer. wrk - a HTTP benchmarking tool. <https://github.com/wg/wrk>, 2022.
- [22] Google. Confidential Space security overview. <https://cloud.google.com/docs/security/confidential-space>, 2022.
- [23] Leonid Grossman. Large receive offload implementation in neterion 10GbE Ethernet driver. In *Linux Symposium*, page 195, 2005.
- [24] P. Halvorsen, E. Jorde, K.-A. Skevik, V. Goebel, and T. Plagemann. Performance tradeoffs for static allocation of zero-copy buffers. In *Proceedings of the 28th Euromicro Conference*, pages 138–143, 2002.
- [25] Michio Honda, Giuseppe Lettieri, Lars Eggert, and Douglas Santry. PASTE: A Network Programming Interface for Non-Volatile Main Memory. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation, NSDI'18*, page 17–33, USA, 2018. USENIX Association.
- [26] Guerny D. H. Hunt, Ramachandra Pai, Michael V. Le, Hani Jamjoom, Sukadev Bhattiprolu, Rick Boivie, Laurent Dufour, Brad Frey, Mohit Kapur, Kenneth A. Goldman, Ryan Grimm, Janani Janakirman, John M. Ludden, Paul Mackerras, Cathy May, Elaine R. Palmer, Bharata Bhasker Rao, Lawrence Roy, William A. Starke, Jeff Stuecheli, Enriquillo Valdez, and Wendel Voigt. Confidential Computing for OpenPOWER. In *Proceedings of the 16th European Conference on Computer Systems, EuroSys '21*, page 294–310, New York, NY, USA, 2021. Association for Computing Machinery.

- [27] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 445–458, Seattle, WA, April 2014. USENIX Association.
- [28] Redis Inc. Introduction to Redis. <https://redis.io/docs/about/>, 2022.
- [29] Intel. Intel TDX® Module v1.5 Base Architecture Specification. <https://www.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-module-1.5-base-spec-348549001.pdf>, 2022.
- [30] Intel. Intel® Architecture Memory Encryption Technologies. <https://www.intel.com/content/www/us/en/develop/download/intel-mktme-specification.html>, 2022.
- [31] Intel. Intel® Trust Domain Extension (Intel® TDX) Module. <https://www.intel.com/content/www/us/en/download/738875/intel-trust-domain-extension-intel-tdx-module.html>, 2022.
- [32] Intel. Intel® Trust Domain Extensions. <https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf>, 2022.
- [33] Li Jie, Chen Shuhui, and Su Jinshu. Implementation of TCP large receive offload on multi-core NPU platform. In *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 258–263, 2016.
- [34] David Kaplan. AMD x86 Memory Encryption Technologies. Austin, TX, August 2016. USENIX Association.
- [35] Yousef A. Khalidi and Moti N. Thadani. An Efficient Zero-Copy I/O Framework for UNIX. Technical report, Sun Microsystems, Inc., USA, 1995.
- [36] Christoph Lameter. Local and remote memory: Memory in a Linux/NUMA system. In *Linux symposium*, pages 1–25, 2006.
- [37] Christoph Lameter. NUMA (Non-Uniform Memory Access): An Overview: NUMA Becomes More Common Because Memory Controllers Get Close to Execution Units on Microprocessors. *Queue*, 11(7):40–51, jul 2013.
- [38] Dingji Li, Zeyu Mi, Yubin Xia, Binyu Zang, Haibo Chen, and Haibing Guan. TwinVisor: Hardware-Isolated Confidential Virtual Machines for ARM. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, SOSP '21*, page 638–654, New York, NY, USA, 2021. Association for Computing Machinery.
- [39] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 337–351, 2022.
- [40] Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. CrossLine: Breaking "Security-by-Crash" Based Memory Isolation in AMD SEV. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS '21*, page 2937–2950, New York, NY, USA, 2021. Association for Computing Machinery.
- [41] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. TLB Poisoning Attacks on AMD Secure Encrypted Virtualization. In *Annual Computer Security Applications Conference, ACSAC '21*, page 609–619, New York, NY, USA, 2021. Association for Computing Machinery.
- [42] Jiuxing Liu. Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12, 2010.
- [43] Alex Markuze, Adam Morrison, and Dan Tsafir. True IOMMU Protection from DMA Attacks: When Copy is Faster than Zero Copy. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '16*, page 249–262, New York, NY, USA, 2016. Association for Computing Machinery.
- [44] Alex Markuze, Igor Smolyar, Adam Morrison, and Dan Tsafir. DAMN: Overhead-Free IOMMU Protection for Networking. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '18*, page 301–315, New York, NY, USA, 2018. Association for Computing Machinery.
- [45] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. Snap: A Microkernel Approach to Host Networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP '19*, page 399–413, New York, NY, USA, 2019. Association for Computing Machinery.
- [46] Eugenio Pérez Martín and Adrian Moreno Zapata. A journey to the vhost-users realm. <https://www.redhat.com/en/blog/journey-vhost-users-realm>, 2019.
- [47] Zeyu Mi, Haibo Chen, Yinqian Zhang, Shuanghe Peng, Xiaofeng Wang, and Michael K. Reiter. CPU Elasticity to Mitigate Cross-VM Runtime Monitoring. *IEEE Transactions on Dependable and Secure Computing*, 17(5):1094–1108, 2020.
- [48] Zeyu Mi, Dingji Li, Haibo Chen, Binyu Zang, and Haibing Guan. (Mostly) Exitless VM Protection from Untrusted Hypervisor through Disaggregated Nested Virtualization. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*, pages 1695–1712. USENIX Association, August 2020.
- [49] MongoDB. Configure mongod and mongos for TLS/SSL. <https://www.mongodb.com/docs/manual/tutorial/configure-ssl/>, 2022.
- [50] Nginx. Nginx. <https://www.nginx.com/>, 2022.
- [51] NVIDIA. ConnectX SmartNICs. <https://www.nvidia.com/en-us/networking/ethernet-adapters/>, 2022.
- [52] Hitoshi Oi and Fumio Nakajima. Performance Analysis of Large Receive Offload in a Xen Virtualized System. In *2009 International Conference on Computer Engineering and Technology*, volume 1, pages 475–480, 2009.

- [53] Michele Paolino, Nikolay Nikolaev, Jeremy Fanguede, and Daniel Raho. SnabbSwitch user space virtual switch benchmark and performance optimization for NFV. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pages 86–92, 2015.
- [54] Joana Pecholt and Sascha Wessel. CoCoTPM: Trusted Platform Modules for Virtual Machines in Confidential Computing Environments. In *Proceedings of the 38th Annual Computer Security Applications Conference, ACSAC '22*, page 989–998, New York, NY, USA, 2022. Association for Computing Machinery.
- [55] PostgreSQL. Secure TCP/IP Connections with SSL. <https://www.postgresql.org/docs/current/ssl-tcp.html>, 2022.
- [56] Redis. memtier_benchmark: A High-Throughput Benchmarking Tool for Redis & Memcached. https://redis.com/blog/memtier_benchmark-a-high-throughput-benchmarking-tool-for-redis-memcached/, 2013.
- [57] Amazon Web Services. AWS for Financial Services. <https://aws.amazon.com/financial-services>, 2023.
- [58] Timothy Stamler, Deukyeon Hwang, Amanda Raybuck, Wei Zhang, and Simon Peter. zIO: Accelerating IO-Intensive Applications with Transparent Zero-Copy IO. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 431–445, Carlsbad, CA, July 2022. USENIX Association.
- [59] Cheng-Chun Tu, Michael Ferdman, Chao-tung Lee, and Tzicker Chiueh. A Comprehensive Implementation and Evaluation of Direct Interrupt Delivery. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '15*, page 1–15, New York, NY, USA, 2015. Association for Computing Machinery.
- [60] Wikipedia. Terabit Ethernet. https://en.wikipedia.org/wiki/Terabit_Ethernet, 2022.
- [61] Dan York. Google Is Now Always Using TLS/SSL for Gmail Connections. <https://www.internetsociety.org/blog/2014/03/google-is-now-always-using-tlsssl-for-gmail-connections/>, 2014.
- [62] Dongli Zhang. swiotlb: 64-bit DMA buffer. <https://lwn.net/Articles/845096/>, 2021.
- [63] Irene Zhang, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson, Omar S. Navarro Leija, Ashlie Martinez, Jing Liu, Anna Kornfeld Simpson, Sujay Jayakar, Pedro Henrique Penna, Max Demoulin, Piali Choudhury, and Anirudh Badam. The Demikernel Datapath OS Architecture for Microsecond-Scale Datacenter Systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, SOSP '21*, page 195–211, New York, NY, USA, 2021. Association for Computing Machinery.