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

Paper #860

Abstract

Existing confidential VMs (CVMs) incur non-trivial 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-I/O tax*, which mainly comprises the bounce buffer mechanism and the packet processing in CVMs, has a significant impact on I/O performance. Specifically, the *CVM-I/O tax* squeezes out vCPU resources of performance-critical application workloads and may occupy more than 50% of CPU cycles. To this end, this paper proposes Bifrost, a novel para-virtualized I/O design that eliminates the I/O payload bouncing tax via removing redundant encryption and reduces the packet processing tax via pre-receiver packet reassembling, while still preserving the same level of security guarantees. We have implemented a prototype of Bifrost with only small modifications to the guest Linux kernel and the vhost-user network I/O backend. The evaluation results on both AMD and Intel servers show that Bifrost significantly improves the performance of I/O-intensive applications in CVMs, which even outperforms the traditional VM by up to 21.50%.

1 Introduction

As more and more data-processing applications (e.g., machine learning and financial services) are embracing the cloud, widespread concerns about the security and privacy of in-use data on the cloud have been raised. As a result, various confidential computing solutions have been proposed to safeguard data from unauthorized parties. Among them, the confidential virtual machine (CVM) solution, such as AMD SEV [2, 3], Intel TDX [23], ARM CCA [7], runs guest operating systems (OSes) in hardware-isolated environments where the complex virtualization stack (e.g., hypervisor and host OS) is no longer trusted and cannot arbitrarily access data in guest OSes, but still provides resource management functionalities. This CVM abstraction transparently protects user workloads without any modification and is easily integrated 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 (like NVIDIA 400Gbps NIC [39]), 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*, resulting in up to 29% overhead over the traditional VM that does not use any CVM protections. The *CVM-IO tax* is brought by common security protections and the 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 because of the CVM-introduced guest CPU state protection. 2) The **bounce buffer mechanism** takes up to 19.45% CPU cycles for bouncing packets (including headers and payload) due to the guest private memory protection in CVMs. 3) The **packet processing** also spends up to 36.14% CPU cycles preparing payloads from massive network packets for application workloads. Fortunately, the VM exits cost becomes negligible when the posted interrupt is available, leaving the bounce buffer mechanism and packet processing in 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, the memory encryption hardware support [24] does not allow the plaintext contents of a memory region to be preserved when modifying the accessibility of the memory region [20]. To reduce network packet processing cost, the current design is to carry the same amount of payload in fewer packets, which can be done by reassembling multiple small packets into larger ones prior to the network stack. But the guest device driver still has to process a large number of network packets, consuming numerous CPU resources.

Fortunately, there are three observations that can help us address the above challenges. 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. We also notice that the payload's memory location can be changed during the end-to-end encryption and decryption. Hence, the elimination of payload bouncing can be achieved by directly encrypting/decrypting payload into/from the guest-host shared memory. Another observation is that the network I/O backend typically has plenty of residual CPU resources. Therefore, it is a good chance to utilize these free computing resources on the network I/O backend to pre-process network packets before being forwarded to the VM to help release the burden of vCPUs.

Based on these observations, this paper proposes **Bifrost** to improve the para-virtualized (PV) network performance of

the CVM with three techniques: ① The *zero-copy I/O NUMA (ZCIO NUMA)* eliminates payload bouncing by always keeping the end-to-end encrypted payload in the guest-host shared memory of ZCIO NUMA. The end-to-end encrypted payload is directly encrypted into the ZCIO NUMA (receiving) or decrypted from the ZCIO NUMA (sending) without bouncing. ② The guest OS's accessing packet content in the ZCIO NUMA may suffer from time of check to time of use (TOCTTOU) attacks. The *one-time trusted read (OTTR)* protects the guest OS from such attacks. With these two techniques, the bouncing of end-to-end encrypted payload, which takes up much CPU resources of CVMs, is securely bypassed. ③ The *pre-receiver packet reassembling (PRPR)* reduces vCPU resources consumed by the device driver via offloading its received packets reassembling to the network I/O backend. Thus, CVMs are able to process fewer packets with larger payload, reducing packet processing cost on vCPUs.

We have implemented a prototype of Bifrost based on modifying guest OS kernel and host user-level software. The ZCIO NUMA is implemented by extending the guest OS kernel (Linux v6.0-rc1) with 815 LoC, while the PRPR is realized by adding 175 LoC and 541 LoC to OpenvSwitch (v2.17.3) and DPDK (v21.1.2), both of which run in the host user mode. We also evaluated Bifrost performance on both AMD and Intel platforms. The results show that, with advanced posted interrupt support, Bifrost is able to enhance the performance of I/O-intensive applications in CVMs to be at most 21.50% better that the traditional VMs.

In summary, the contributions of this paper are:

- The first thorough performance analysis of I/O-intensive applications in CVMs on existing and next-gen hardware platforms, which reveals their performance bottlenecks and overhead sources compared with traditional VMs.
- The secure PV I/O design that releases large quantities of CPU resources for applications by reducing the CVM-IO tax via the ZCIO NUMA and the BE-GRO techniques.
- A prototype system of Bifrost (which will be publicly available) and a comprehensive evaluation on both AMD and Intel platforms to show the improvements on existing and future hardware platforms.

2 Background

2.1 Confidential VM (CVM)

There are different CVM solutions based on specialized hardware extensions. All these solutions leverage hardware memory encryption and integrity checking [21, 24] to enforce confidentiality and integrity. They share the same CVM abstraction that excludes the whole virtualization stack from the trusted computing base (TCB). As shown in Figure 1, the trusted firmware (the unique software TCB) isolates the CVM from untrusted hypervisors and traditional VMs.

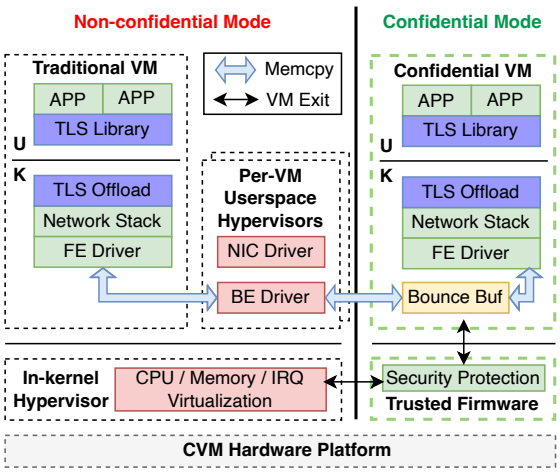


Figure 1: The PV I/O networking architecture of traditional VMs and CVMs atop the CVM hardware platform. The black arrows represent the path of VM exits and VM enters. The light blue arrows indicate memory copies that consume CPU resources. The FE Driver in the figure means Frontend Driver, while the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

Existing CVM systems usually 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. The shared memory holds plaintext data that 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. But the data content of the memory page cannot be preserved before and after the security type switch [20]. Hence, the guest should move data outside the 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 microarchitectures, requiring multiple VM exits and inter-processor communications [4, 20]. As a result, the security type switch is unsuitable to be frequent in CVMs.

2.2 PV I/O Networking in CVM

PV I/O has become a primary I/O virtualization choice for modern cloud providers due to its high performance and great compatibility. There are two cooperative drivers in PV 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 and directly controls the device in a busy-polling mode [9].

An example of PV 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 consists of network functions that handle conversions between payload and packets. For example, in the

transmission (TX) direction, the payload from applications and headers from the network stack are encapsulated into network packets via packet processing, after which the backend driver is notified to send them out. Because the entire memory space of a traditional VM is accessible to the hypervisor, the backend driver can freely copy the packets from the guest memory to its own memory and forwards them to the NIC driver.

As we have introduced in § 2.1, the host OS is untrusted and cannot access the private memory of CVMs. Therefore, PV I/O utilizes the bounce buffer in CVM shared memory to transfer packets from/to the host OS. As shown in the right part of Figure 1, the guest OS reserves a memory region shared with the host OS as the bounce buffer and copies the packets to it. Afterwards, the backend driver can copy the packets to the hypervisor as normal. As a result, the packet 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 protect their I/O payload in transit [6, 10, 37, 41, 47]. And CVM solutions have made it a mandatory requirement for their applications [14, 40, 45]. Moreover, today's OSes such as Linux have provided the in-kernel TLS support, enabling userspace applications to offload TLS to the kernel for enhanced performance and expanded features [11]. As shown in Figure 1, the in-kernel TLS allows the payload from the page cache to be encrypted without going to the userspace.

The industry currently implements the TLS protocol based on encryption algorithms that simultaneously assure the confidentiality and integrity of data. The output of such encryption algorithms consists of encrypted ciphertext (for confidentiality of data) and an authentication tag (for integrity of the ciphertext) generated from the ciphertext. The correctness of both ciphertext and its authentication tag must be guaranteed during the encryption to provide the complete data security protection, and vice versa.

3 Analysis of CVM-IO Tax

In this section, we quantify the performance impact of the CVM-IO tax by examining the I/O performance of existing CVMs and comparing them with traditional VMs. A CVM running I/O-intensive applications can be divided into halves. 1) *Application workloads*: diverse application workloads. 2) The *CVM-IO tax*: common CVM-introduced security protections and intrinsic network I/O procedures. Application workloads include business logic and payload processing that varies among different applications, while the CVM-IO tax consists of VM exits, the bounce buffer mechanism and the packet processing during the payload preparation for application workloads.

All experiments are based on an 128-core AMD SEV-ES/SNP server and an 24-core Intel server with 200Gbps NICs [1]. The AMD server is used to evaluate the I/O performance of real CVMs, which is named as ***CVM***. However, the AMD server does not support posted interrupts that inject virtual interrupts into a running CVM directly, which can greatly reduce the number of VM exits and thus improve virtualization performance. Therefore, we simulate next-gen CVMs using the Intel server that supports posted interrupts, which is called ***CVM+PI***. We increase the VM exit latency in the Intel VM to be the same as the AMD VM, and enable the fastest implementation of the bounce buffer mechanism (i.e., Linux SWIOTLB [33]) for both VM types. For fair performance comparison, *CVM*'s baseline is the vanilla AMD traditional VM, while *CVM+PI*'s baseline is the vanilla Intel traditional VM.

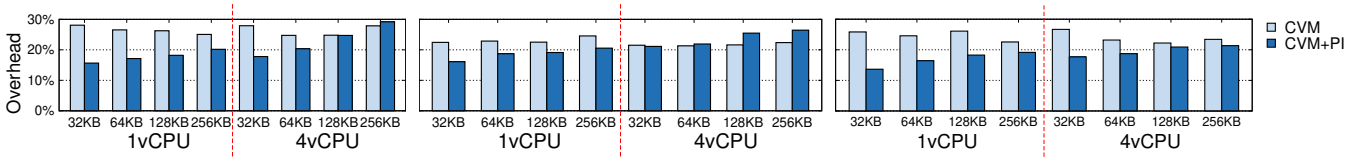
To achieve the best network performance, we choose vhost-user as the network backend in follow-up experiments. Despite that our AMD server supports the latest SEV-SNP extension, we still use the SEV-ES VM, since the SEV-SNP VM does not support vhost-user due to its not supporting huge pages. There should be no discernible I/O performance difference between the SEV-ES VM and the SEV-SNP VM using vhost-user backend, because the SEV-SNP VM's security protections do not increase the CVM-IO tax.

3.1 CVM-IO Tax Breakdown

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

We further take the Memcached benchmark with the 4vCPU-256KB test cases as an example to breakdown the time cost of the CVM-IO tax of CVMs, as shown in Figure 3. In the left subfigure, the keys *VM Exit*, *SWIOTLB* and *Pkt Processing* correspond to the three parts of *CVM-IO tax*: VM exits, the bounce buffer mechanism and the packet processing, respectively, while the key *App Workload* corresponds to *Application workloads*. Because all vCPUs are saturated during the benchmark, the higher the percentage of CPU time consumed by the CVM-IO tax, the greater the overall performance will be impacted. The CVM-IO tax uses more than 50% of total CPU time in all CVMs during the benchmark. For *CVM*, VM exits consuming more than 20% CPU time have more impact than SWIOTLB. For *CVM+PI*, the SWIOTLB cost occupies more than 19% CPU time, while VM exits take a very low percentage of the total time in both

[1] The detailed testbed configurations are described in § 7.1.



(a) Memcached **(b)** Nginx **(c)** Redis

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.

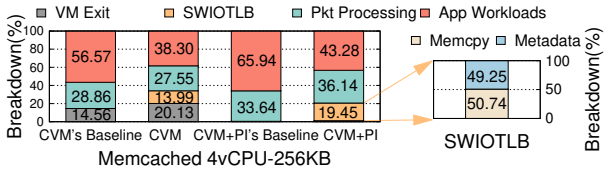


Figure 3: CPU time breakdown of the Memcached 4vCPU-256KB case in CVMs. The left subfigure shows the CPU time breakdown of *CVM*, *CVM+PI* and their baselines in the Memcached 4vCPU-256KB case. The right subfigure shows the CPU time breakdown of the SWIOTLB part in *CVM+PI*.

CVM+PI (less than 1.2%) and its baseline (less than 0.5%). Packet processing in all above cases takes up about 30% of CPU time and thus has a considerable impact.

The overhead over baselines can be explained by the reduction percentage of application workloads' CPU time 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 SWIOTLB in CVMs contribute most of the overhead.

Take-away I

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

Lengthy VM Exits The AMD SEV-ES hardware starts to protect CPU states such as registers and cache of each CVM from the untrusted hypervisor on each VM exit, which adds thousands of cycles per VM exit than a traditional VM. We first breakdown the the VM exit handling cost, finding that *CVM* has to spend 5,833 more cycles on guest-host world switches than its baseline during every virtual interrupt delivery, which happens very frequently in I/O-intensive applications. We then collect the number of VM exits per second during Memcached 4vCPU-256KB benchmark for different CVMs. *CVM* averagely triggers 41,615 VM exits per second on each vCPU, while *CVM+PI* triggers only 2,803 VM exits per second on each vCPU, a magnitude fewer 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, the next-gen CVM platforms including the AMD SEV, Intel TDX and ARM CCA all support posted interrupt, so that the performance impact caused by lengthy VM exits can become minimal.

Take-away II

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

Bounce Buffer To analyze the SWIOTLB overhead, we breakdown the CPU time of SWIOTLB in the 4vCPU-256KB case of *CVM+PI* into the I/O data (i.e., packets in this case) copy and the metadata maintenance (e.g., buffer allocation and free). The breakdown result shown in the right subfigure of Figure 3 indicates that I/O data copy (corresponding to the key *Memcpy*) consumes 50.74% of the SWIOTLB time, while the metadata maintenance (corresponding to the key *Metadata*) spends 49.25% of the SWIOTLB time. Besides, the experimental results of small and big data sizes reflects that the performance impact of SWIOTLB rises as the data size increases.

Take-away III

> *SWIOTLB consumes a large percentage of CPU resources due to the I/O data copy and the metadata maintenance. It is necessary to avoid large-size I/O data bouncing to minimize SWIOTLB'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 all Memcached 256KB cases. Since the packet processing time cost is proportional to the number of packets processed, the huge number of network packets in I/O-intensive scenarios causes it to demand considerable CPU resources.

Take-away IV

> *Packet processing occupies a large ratio of CPU time due to the large number of packets to be processed. Reducing the number of packets to be processed is able to mitigate its performance impact.*

4 Overview

4.1 Design Goals

The primary goal of Bifrost is to reduce the PV 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 all platforms (e.g., x86, ARM, RISC-V), and support different host and guest OSes (e.g., Linux, FreeBSD, 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. But it is not easy to implement the two optimizations due to the following two technical challenges:

C1: out-of-place hardware encryption and decryption. An ideal way to remove the bounce buffer mechanism while ensuring data security is to give either the guest or the host exclusive access to the same memory region when using the packet in it. However, as mentioned in § 2.1, when a private page containing a packet is converted to a shared page, the packet in this page is lost and unable to correctly be passed to the hypervisor, and vice versa. As a result, changing the security type of guest memory pages is too costly to be a frequent operation on the I/O critical path.

C2: costly packet pre-processing in the device driver. Because packet processing mainly focuses on packet headers rather than payloads, one common current approach to reducing packet processing cost while maintaining high network performance is to pre-process multiple same-flow small packets into a large one 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 fast-speed NIC.

4.3 Observations and Insights

We observe unique characteristics of existing CVM systems, allowing us to suggest 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 the private memory protection or the 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 first encrypts the payload into private memory. The private memory protection is a redundant mechanism for data that has already been encrypted. As a result, the payload bouncing tax can be eliminated by the existing end-to-end encryption while removing the private memory protection at the same time.

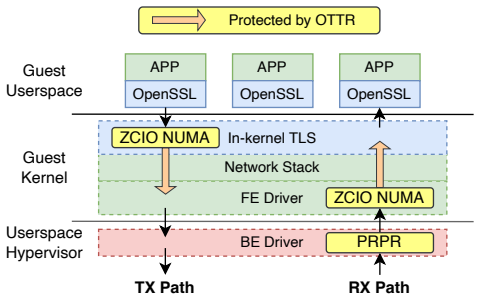


Figure 4: The overall architecture of Bifrost.

O2: End-to-end encryption has the side effect of moving the memory location of payload. The procedure of adding the end-to-end encryption allows CVM to move payloads to a different memory location. Hence, the end-to-end encryption in the in-kernel TLS layer becomes an ideal timing for adding the end-to-end encryption while removing the private memory protection on payload transparently for userspace applications and libraries. In particular, the ciphertext generated during the encryption can be directly written to the target shared memory with the host, which ensures 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 [9,36]. 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, the I/O backend with adequate residual CPU resources can be utilized to release the burden of vCPUs by pre-processing network packets before passing them to CVMs.

4.4 Architecture

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

To address challenge **C1**, Bifrost proposes two techniques to enable zero-copy transparently and securely for the end-to-end encrypted payload in the CVM. **D1:** zero-copy I/O NUMA (ZCIO NUMA) 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 any payload copy, and vice versa. However, concurrent memory accesses to the plaintext data in the shared memory allocated from the ZCIO NUMA may lead to a TOCTTOU attack. A malicious host can tamper with data that has passed the security checks of the guest OS, such as tainting a packet header after it has passed the checksum

check. To defend against this attack, Bifrost introduces **D2:** one-time trusted read (OTTR), which reads the target data content into the private memory or registers before allowing the guest OS to handle it. To address challenge **C2**, Bifrost proposes another technique to finish pre-processing network packets before they reach the CVM. **D3:** pre-receiver packet reassembling (PRPR) makes use of the network backend's free CPU resources to pre-process small incoming packets into large ones before transmitting them to the guest OS.

We explain the Bifrost architecture and its techniques by describing the high-level workflows of packet receiving and sending. **Packet receiving workflow:** When a network packet carrying end-to-end encrypted payload arrives at the network I/O backend, Bifrost tries 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 the private memory for security, while keeping the end-to-end encrypted payload in the shared memory allocated from ZCIO NUMA (**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 in the ZCIO NUMA memory directly into 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 starts to send out payload from the guest OS, Bifrost first leverages the in-kernel TLS support to encrypt plaintext from application memory or kernel page cache in the private memory and places the encrypted result directly into the guest-host shared memory allocated from ZCIO NUMA (**D1**). The memory copy of the end-to-end encrypted payload from the private memory to the shared bounce buffers is removed at this step. For small metadata that is not encrypted by the end-to-end protection, 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 existing CVM solutions. The TCB only contains the CPU hardware and minimized trusted monitor firmware or software if any. Attackers can control any untrusted software entities or hardware devices to conduct 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 are out of scope. Even though CVM implementations may have bugs [5, 30] and are subject to side-channel attacks [27–29, 46], we do not consider them because they are orthogonal to this paper.

5 Detailed Design

5.1 Zero-Copy I/O NUMA (ZCIO NUMA)

Bifrost reserves a contiguous shared memory region for PV I/O networking in the guest physical address (GPA) space. This shared memory is presented as separate ZCIO NUMA node(s) for ease of management, allowing Bifrost to readily utilize the mature memory management mechanisms in existing guest OSes. Moreover, the location and size of ZCIO NUMA are fixed at the boot time of a CVM for high performance.

Boot-time initialization: The memory range of a ZCIO NUMA node can be configured by setting the base GPA and the total length via the kernel command line. Bifrost parses the number of ZCIO NUMA nodes and adds the specified guest memory range to each of them. All ZCIO NUMA nodes are created with no associated vCPU. Before a ZCIO NUMA node is available for memory allocations, Bifrost sets the security type of its memory to shared. The distances between ZCIO NUMA nodes are the same as the distances between normal NUMA nodes to which their memory ranges originally belonged, while each ZCIO NUMA node is zero distance from its original NUMA node.

Runtime allocation: To avoid data leakage due to the unintentional storing data into the ZCIO NUMA memory, Bifrost adjusts the memory allocation policies of the guest OS to allow only explicit allocation to acquire ZCIO NUMA memory. In other words, the original memory allocations in the system do not allocate memory from ZCIO NUMA, so there is no security issue of incorrectly exposing sensitive data. Guest kernel components are merely able to allocate memory from ZCIO 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 nearest ZCIO NUMA node to the vCPU that is running this component. In the frontend driver, Bifrost checks the memory location where the payload is located, and if it belongs to a ZCIO NUMA node, Bifrost will skip 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 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 ZCIO NUMA node for

storing encrypted payload.

In the RX direction, the guest memory regions used to accept incoming network packets are allocated and assigned by the frontend driver (i.e., virtio-net driver 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 directly decrypts the ciphertext from the ZCIO NUMA memory to the private memory.

5.2 One-Time Trusted Read (OTTR)

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

Packet header handling: For packet headers, guest OS's packet processing functions should only use the header's content after validating the header. However, if a malicious host modifies the header after 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 this end, Bifrost needs to read a packet header from the ZCIO NUMA into the private memory before further processing it. This read only happens one time for each packet header and Bifrost will never read the header from the ZCIO NUMA again to prevent the host from tampering with the content of packet headers later. In the TX direction, unlike the end-to-end encrypted payload that is stored in the ZCIO NUMA, network packet headers are placed in the 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 an 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 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 ZCIO NUMA and may be tampered with by 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 ZCIO NUMA memory in the last phase of the parallel decryption, suffering from TOCTTOU attacks on the ciphertext.

To prevent the payload TOCTTOU attack, in the decryption procedure, Bifrost reads only once from the ZCIO NUMA to load the ciphertext value into CPU registers and always uses the proper ciphertext in the registers afterwards, avoiding reading potentially compromised ciphertext. Similarly, during the encryption procedure, the ciphertext of each payload is bound to be valid at the moment it is generated and still kept in CPU registers. Thus, Bifrost calculates the authentication tag using the proper ciphertext that is still in the CPU registers.

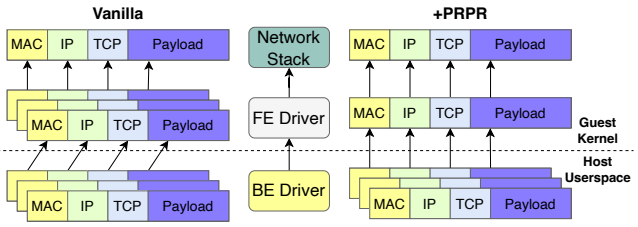


Figure 5: Comparison of the packet reassembling workflow of the vanilla CVM and the CVM with PRPR.

5.3 Pre-receiver Packet Reassembling (PRPR)

Large packets are split into smaller ones before sending out due to the transmission size limit. Modern OSes also reassembles small packets at the device driver layer to reduce the number of packets handled by the network stack. But the packet reassembling in the guest device driver still takes up a lot of vCPU resources when the number of packets is large, severely affecting the application performance. Bifrost frees up precious vCPU resources for CVMs by moving the packet reassembling to the hypervisor backend driver that has sufficient CPU time, so that the number of packets processed by the guest greatly decreases.

Overall procedure: When a network packet arrives at the network backend, some packets may be cached in the backend and waiting for reassembling. Bifrost first parses the current packet header to determine if there are any cached packets with the same flow. If so, Bifrost tries to merge the current packet with the cached same-flow ones. Eventually, based on the status information in the current 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: The same-flow packets are network packets that share the same source and destination and sequence, which can be parsed from their headers in link layer (L2), network layer (L3) and transport layer (L4). As our current implementation focuses on TCP/IP packets, Bifrost first recognizes packets that have the same MAC address, IP address and TCP port in both source and destination directions as candidates. Then Bifrost considers these candidates with the same TCP acknowledgement (ACK) number as actual same-flow packets,

Flexible per-VM flush rules: It is essential to flush packets to the guest OS at the appropriate time, because the network performance is very sensitive to the latency of network packets. If a packet has been cached in the backend driver for too long, the receiver CVM may incorrectly assume that the network is blocked, thus lowering its throughput for congestion control. To flush packets to the receiver in time, Bifrost sets up a set of basic flush rules based on the status information of packets.

For a newly received packet that 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 then 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.

Apart from the above basic rules, Bifrost also allows each guest OS to customize flush rules. For example, a latency-sensitive VM that merely forwards packets after simple processing may want to receive each network packet with no delay. Bifrost provides PV interfaces for receiver CVMs to install their own rules, so that they can disable reassembling, adjust the maximum number and the timeout of cached packets, etc.

Packet reassembling: Among the cached same-flow packets, the currently received packet can only be reassembled with the packet whose payload is continuous with it. Bifrost attempts to find neighbors of the currently received packet for reassembling by comparing their TCP sequence (SEQ) numbers. For example, Bifrost will append the currently received packet to the cached packet whose sequence number is immediately before it by carefully merging their headers. As shown in Figure 5, duplicate packet headers are merged during reassembling. For example, if one of the packets to be reassembled contains an immediate-flush flag, Bifrost will also set this flag in the reassembled header.

6 Implementation

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

In the Linux kernel, we have added 815 lines of code to support ZCIO NUMA. It mainly contains the creation of ZCIO NUMA nodes during memory subsystem initialization as well as memory allocation invocation replacement.

In the DPDK, we added 541 lines of code, which consists of two parts: 1) Reorganization of the network package, including header trimming during packet merging, flag resetting, etc. 2) Flush rules as mentioned above, this part is mainly focused on deciding whether to cache or immediately flush one incoming packet. Our implementation also adds about 175 lines of code to OpenvSwitch, which invokes the interfaces provided by the DPDK and pre-processes the network packets (e.g., parsing the header in advance).

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 6.0-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 the passthrough mode.

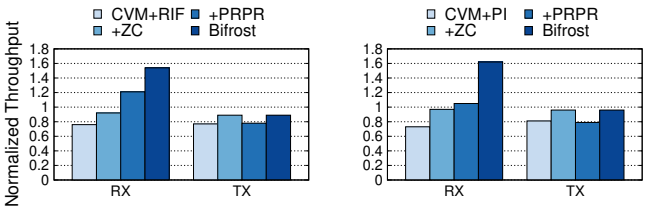
7.2 Performance Improvement

We first build a microbenchmark to investigate the upper bound of 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 should concentrate on Bifrost's effect CVMs atop such hardware. 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*, which represents *CVM + Reduced Interrupt Frequency*. In this section, we focus on the performance improvement of *CVM+RIF* and *CVM+PI*. *+ZC* means only applying the ZCIO NUMA as well as the OTTR techniques, while *+PRPR* means merely adding the PRPR technique.

7.2.1 Microbenchmark

We write a TCP-based TLS client/server pair to evaluate the network throughput. They just contain simple code for single-threaded I/O data sending and receiving, minimizing the time cost of business logic and thus 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.

Throughput in the RX direction Figure 6a shows the comparisons of network throughput in the RX direction.



(a) AMD Server **(b)** Intel Server

Figure 6: TLS normalized throughput on both AMD and Intel server. The Y-axis is the normalized throughput compared with the baseline.

CVM+RIF attains 5.26 Gb/s, which is 24.10% slower than its baseline's 6.93 Gb/s. With the ZCIO NUMA, *+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) shown in Figure 6b, 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 make Bifrost reach 16.64 Gb/s, which is 62.20% higher than the baseline.

Therefore, Bifrost can enhance the performance by up to 89.23% (from 27.03% overhead to 62.20% better than the baseline) for applications that have high RX traffic in existing CVMs. It is also shown that the performance improvement of combining two techniques is greater than the sum of each technique's individual improvement.

Throughput in the TX direction Figure 6a also shows the comparisons of network throughput 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%. While *+PRPR* (9.71 Gb/s) has little improvement. Bifrost has 10.79% overhead (11.11 Gb/s), slightly better than *+ZC*. Experiments of *CVM+PI* in Figure 6b show 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. The performance improvement in the TX direction is much smaller than in the RX direction, mainly because the PRPR technique only focuses on optimizing the RX traffic.

7.2.2 Applications

We utilize the same network-intensive applications as in § 3 to evaluate and breakdown the performance improvement of Bifrost. TLS/SSL is enabled in all of those applications. We run each benchmark for 30 seconds and report the average value of 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.

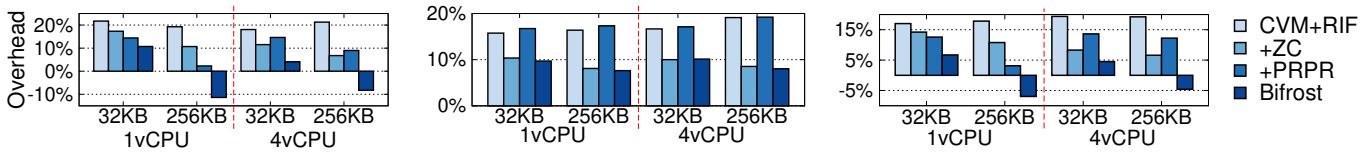
Memcached [12] is a popular multi-threaded in-memory key-value store application. We use the *memtier_benchmark* [42] tool to measure the throughput. The *Memcached* server is configured with either 1 or 4 threads for 1vCPU or 4vCPU VM respectively, and 4096MB memory. The *memtier_benchmark* uses 4 clients, 16 concurrencies for 1-thread server and 8 clients, 32 concurrencies for 4-thread server.

Figure 7a shows the Memcached benchmark results of *CVM+RIF*. 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 improvement increases as the data size grows. This is because the performance impact of CVM-IO tax is amplified with the growth of data size, resulting in more room for performance improvement. First, the increasing data amount leads to more data copies, which enlarges the SWIOTLB tax. Moreover, the packet processing tax also rises since the growing data size increases the amount of data transmitted and the number of packets to be processed.

Figure 9 shows the breakdown of 4vCPU-256KB cases. The ZCIO NUMA technique reduces the total timeshare of the SWIOTLB tax from 15.67% to less than 2.50%. It is not able to completely eliminate the SWIOTLB cost because some small I/O data (e.g., TCP handshake packets) still falls back to the SWIOTLB. The PRPR technique reduces the timeshare of the packet processing tax from 28.73% to 21.88%. Bifrost spends 2.41% more time on application workloads than the baseline, with Bifrost's more than 10% speed gain on the TLS processing in application workloads, which explains the 8.26% improvement over the baseline.

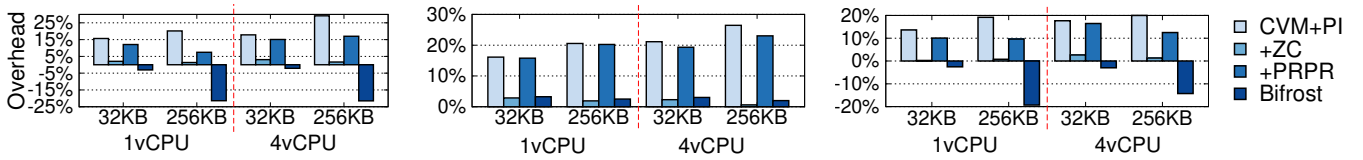
Figure 8a shows the Memcached benchmark results of *CVM+PI*. Bifrost outperforms the baseline in all cases, where the acceleration over the baseline can reach 3.06% in 32KB cases and 21.50% in 256KB cases. *+ZC* obtains more individual performance gain than *+PRPR*, and their combined improvement is larger than the sum of their individual one. That's because more vCPUs increase throughput, which raises the overall quantity of data during the evaluation. Similar to the *CVM+RIF* cases, a larger amount of data brought by larger data sizes and more vCPUs leads to a higher CVM-IO tax impact that highlights Bifrost's strengths even more.

Figure 10 shows the breakdown of 4vCPU-256KB cases. The ZCIO NUMA technique reduces the total timeshare of the SWIOTLB tax from 19.45% to less than 2.77%. The PRPR technique reduces the timeshare of the packet processing tax from 36.14% to 26.83%. Bifrost spends 11.53% more time on application workload than baseline. Considering Bifrost's



(a) Memcached **(b)** Nginx **(c)** Redis

Figure 7: Application performance overhead on the AMD server. The Y-axis indicates relative overhead, negvative overhead represents performance improvement. CVM+RIF represents CVM + Reduced Interrupt Frequency.



(a) Memcached **(b)** Nginx **(c)** Redis

Figure 8: Application performance overhead on the Intel server. The Y-axis indicates relative overhead, negvative overhead represents performance improvement.

more than 10% speed gain on the TLS processing in application workloads, the 21.50% improvement over the baseline can be explained.

Nginx [38] is a well-known high-performance HTTP(S) web server. We run the *wrk* [13] benchmark tool measure the throughput represented by requests per second (RPS). The client configurations are similar to the other two applications.

Figure 7b shows the Nginx benchmark results 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 ZCIO NUMA technique, as analyzed in § 7.2.1. *+ZC* reduces the overhead by less than half because lengthy VM exits still significantly impact performance in these benchmarks. The PRPR even increases the overhead for a little bit in the 1vCPU-256KB case, because there are more VM exits during the benchmark after the PRPR is applied. Figure 8b shows the Nginx benchmark results of *CVM+PI*. Bifrost brings the overhead to less than 2.8% in all cases, thanks to the ZCIO NUMA. The PRPR no longer has a negative effect on the performance because VM exits cost is trivial when posted interrupt is enabled.

Redis [19] 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 8c present the Redis benchmark results, which have similar patterns to those of Memcached.

7.3 TOCTTOU Protection Overhead

Bifrost defends the guest OS against TOCTTOU attacks with OTTR by copying packet headers into the private memory and keeping end-to-end encrypted payload in registers during their processing. To evaluate the performance impact of these operations, we implement a prototype of Bifrost without applying OTTR, called Bifrost-noprot. We repeat the Memcached, Redis and Nginx benchmarks on Bifrost-noprot and compare the performance of Bifrost with it. The overhead of Bifrost over Bifrost-noprot in different benchmarks is shown in Figure 11 and Figure 12, indicating no more than 2.0% overhead caused by TOCTTOU protections in all cases.

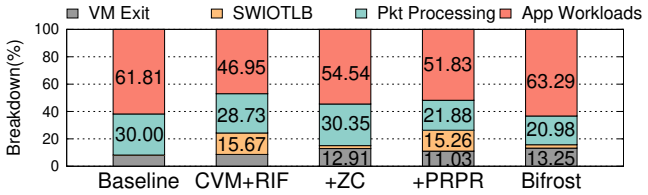


Figure 9: Breakdown of Memcached 4vCPU-256KB experiments on AMD server. The Y-axis represents the percentage of cycles in different parts of the system.

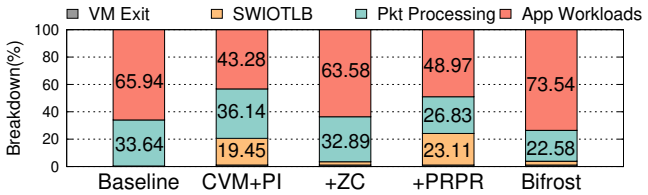
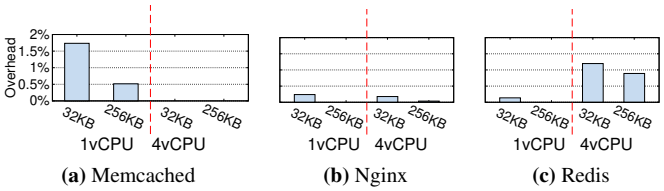


Figure 10: Breakdown of Memcached 4vCPU-256KB experiments on Intel server. The Y-axis represents the percentage of cycles in different parts of the system.

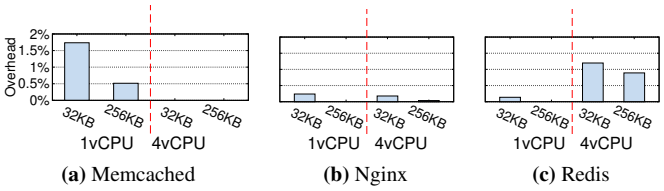
7.4 Memory Footprint

It is important for Bifrost to utilize the memory resource efficiently in case of unavailability due to depletion of the ZCIO NUMA memory. We first measure the ZCIO NUMA's memory consumption via the *numastat* tool. Among all of our benchmarks, Bifrost consumes no more than 200MB memory of the ZCIO NUMA, which fits in the 512MB size of the ZCIO NUMA node and does not exceed the 1GB default size



(a) Memcached **(b)** Nginx **(c)** Redis

Figure 11: TOCTTOU protection overhead of Memcached experiments on the AMD server. Performance normalized to Bifrost without TOCTTOU protection.



(a) Memcached **(b)** Nginx **(c)** Redis

Figure 12: TOCTTOU protection overhead of Memcached experiments on the Intel server. Performance normalized to Bifrost without TOCTTOU protection.

of the SWIOTLB area in the 16GB *CVM+RIF* VM. PRPR maintains a packet cache list for each virtual network receive queue in the backend memory. 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 virtual network receive queues, which bring only 2.125MB memory cost.

8 Security Analysis

Bifrost introduces three major techniques to existing CVMs, in which only the ZCIO NUMA and the OTTR retrofit the guest kernel and 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 the private memory. Thus, we only need to analyze the security risks caused by TOCTTOU attacks on network packets during the network I/O.

Headers: In the RX direction, a header is received in the guest-host shared memory. Bifrost copies the header into the 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 the private memory and sent out through the bounce buffer mechanism, which is same as 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 the private memory. Bifrost ensures that the decryption code has a consistent view of the encrypted payload by reading only once from the shared memory and keeping it in CPU registers. In the TX direction, the in-kernel TLS layer encrypts the plaintext payload directly into the shared memory. Bifrost ensures that the encryption code always refers to the correct ciphertext in CPU registers that is isolated by CVM platforms, 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 the shared memory to the private memory. In the TX direction, the plaintext payload is not accessed after the guest kernel copies it from the private memory to the shared memory. No shared memory accessing results in no TOCTTOU risks.

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

9 Related Work

Secure Enclaves. Hardware vendors have released different secure enclave systems [3, 7, 8, 17, 23, 43]. AMD SEV [2, 3], Intel TDX [20, 23] and ARM CCA [7] enable the CVM abstraction with hardware extensions, especially memory encryption and integrity protection [21, 24]. Unlike AMD SEV, which relies on a secure processor [1], Intel TDX and ARM CCA use trusted firmware [22] to manage CVMs. TwinVisor [26] provides an alternative to ARM CCA by retrofitting the virtualization extension in the existing ARM TrustZone. The design of Bifrost is not restricted to AMD or Intel and can be applied to other CVM systems.

Previous work [31] has reported preliminary I/O performance of ARM CCA solution, showing that the overhead incurred by CCA VM over unmodified KVM VM is at most 18% for I/O-intensive workloads. But the testbed is only equipped with 2 CPUs and 1Gbps NIC, which does not match the hardware settings available in modern data centers.

Zero Copy I/O. Prior researches have proposed various techniques to eliminate data copies for better I/O performance [15, 18, 25, 32, 34, 35]. For user-level applications, zIO [44] can transparently remove unnecessary I/O copies for applications. For the I/O stack in kernel, DAMN [35] and Demikernel [48] eliminate I/O memory copies by directly allocating buffers from the I/O memory pool. PASTE [16] performs DMA directly into non-volatile memory to avoid copies. Unlike these systems that optimize applications or require intrusive software modifications, Bifrost focuses on eliminating I/O copies in CVMs with minor modifications.

10 Conclusion

This paper is the first to systematically analyze the IO tax for network intensive workloads in CVMs. To optimize CVM I/O performance, we propose a PV I/O design called Bifrost to eliminate redundant data bouncing and reduce packet processing cost. Evaluation results show that Bifrost significantly improves CVM I/O performance and even outperforms the traditional VM by up to 21.50%.

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. AMD Server Vulnerabilities. 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. ARM Confidential Compute Architecture. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture, 2023.

[8] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding applications from an untrusted cloud with haven. *ACM Trans. Comput. Syst.*, 33(3), aug 2015.

[9] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich 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.

[10] Docker. Protect the Docker daemon socket. https://docs.docker.com/engine/security/protect-access/, 2022.

[11] Jake Edge. TLS in the kernel. https://lwn.net/Articles/666509/, 2015.

[12] Brad Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.

[13] Will Glozer. wrk - a HTTP benchmarking tool. https://github.com/wg/wrk, 2022.

[14] Google. Confidential Space security overview. https://cloud.google.com/docs/security/confidential-space, 2022.

[15] P. Halvorsen, E. Jorde, K.-A. Skevik, V. Goebel, and T. Plagemann. Performance tradeoffs for static allocation of zero-copy buffers. In *Proceedings. 28th Euromicro Conference*, pages 138–143, 2002.

[16] 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.

[17] Guerney 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 Sixteenth European Conference on Computer Systems*, EuroSys '21, page 294–310, New York, NY, USA, 2021. Association for Computing Machinery.

[18] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. NetVM: High performance and flexible networking using virtualization on commodity platforms. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 445–458, Seattle, WA, April 2014. USENIX Association.

[19] Redis Inc. Introduction to Redis. https://redis.io/docs/about/, 2022.

[20] 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.

[21] Intel. Intel® Architecture Memory Encryption Technologies. https://www.intel.com/content/www/us/en/develop/download/intel-mktme-specification.html, 2022.

[22] 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.

[23] Intel. Intel® Trust Domain Extensions. https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf, 2022.

[24] David Kaplan. AMD x86 Memory Encryption Technologies. 2016.

[25] Yousef A. Khalidi and Moti N. Thadani. An efficient zero-copy i/o framework for unix. Technical report, USA, 1995.

[26] 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.

[27] 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.

[28] 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.

[29] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 717–732, 2021.

[30] 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.

[31] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. Design and verification of the arm confidential compute architecture. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 465–484, Carlsbad, CA, July 2022. USENIX Association.

[32] 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.

[33] LWN.net. swiotlb: 64-bit DMA buffer. https://lwn.net/Articles/845096/, 2021.

[34] Alex Markuze, Adam Morrison, and Dan Tsafrir. True iommu protection from dma attacks: When copy is faster than zero copy. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, page 249–262, New York, NY, USA, 2016. Association for Computing Machinery.

[35] Alex Markuze, Igor Smolyar, Adam Morrison, and Dan Tsafrir. Damn: Overhead-free iommu protection for networking. In *Proceedings of the Twenty-Third 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.

[36] 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.

[37] MongoDB. Configure mongod and mongos for TLS/SSL. https://www.mongodb.com/docs/manual/tutorial/configure-ssl/, 2022.

[38] Nginx. Nginx. https://www.nginx.com/, 2022.

[39] NVIDIA. ConnectX SmartNICs. https://www.nvidia.com/en-us/networking/ethernet-adapters/, 2022.

[40] 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.

[41] PostgreSQL. Secure TCP/IP Connections with SSL. https://www.postgresql.org/docs/current/ssl-tcp.html, 2022.

[42] 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.

[43] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 955–970, New York, NY, USA, 2020. Association for Computing Machinery.

[44] Timothy Stamler, Deukyeon Hwang, Amanda Raybuck, Wei Zhang, and Simon Peter. zIO: Accelerating IO-Intensive applications with transparent Zero-Copy IO. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 431–445, Carlsbad, CA, July 2022. USENIX Association.

[45] Artemii Ustiukhin. Exploring approaches for secure workload deployment and attestation in virtualization-based confidential computing environment. 2022.

[46] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656, 2015.

[47] 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.

[48] 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.