Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 11.1111/ACCESS.2024.DOI

Extended Description of Controlled Shared Memory (COSM) Isolation: Principles, Validation, and Use Cases

VIGNESH SUNDARAVARATHAN[1], (Student Member, IEEE), MARTIN REISSLEIN[1] (Fellow, IEEE), AKHILESH THYAGATURU[2] (Senior Member, IEEE), GURPREET SINGH KALSI[3] (Senior Member, IEEE), JASON HOWARD[3] (Member, IEEE), JAN KAISRLIK[3], BARTOSZ MATWIEJCZYK[3], MAREK M. LANDOWSKI [3] (Member, IEEE), PIOTR DOROZYNSKI[3], HARVEY VRSALOVIC[3], and SANJAYA TAYAL[3]

[1]School of Electrical, Computer, and Energy Eng. Arizona State Univ., Tempe, AZ 85281 USA (e-mail: {vsunda21, reisslein}@asu.edu)
[2]Edge Computing Group, Intel Corporation, Chandler, AZ 85226, USA (e-mail: {akhilesh.s.thyagaturu}@intel.com)
[3]Extreme Scale Computing Group, Intel Corporation, Hillsboro, OR 97124, USA (e-mail: {gurpreet.s.kalsi, jason.m.howard, bartosz.matwiejczyk, jan.kaisrlik, marek.m.landowski, piotr.dorozynski, harvey.vrsalovic, sanjaya.tayal}@intel.com)

This work was supported in part by funding from the Defense Advanced Research Projects Agency (DARPA) to Intel Corp. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the US government.

ABSTRACT Recent memory sharing approaches, e.g., based on the Compute Express Link (CXL) standard, allow the flexible high-speed sharing of data (i.e., data communication) among multiple hosts. In information systems for sensitive data, the data sharing between hosts, must be closely controlled. Security policies may require strict isolation, so-called air-gapping. However, strict isolation mechanisms are currently lacking in data communications based on shared memory. We propose the novel *COntrolled Shared Memory (COSM)* framework for strictly and dynamically controlling the data communication via shared memory approaches. We introduce the novel concept of *COSM isolation*, which restricts data communication via shared memory regions with first-level isolation based on a write-and-read permission matrix and second-level isolation based on data inspection. These isolation levels are enforced by the memory controller on an externally-attached shared memory device (ESMD). COSM isolation is thus generally more secure than the existing software based isolation (e.g., virtual machine isolation of a hypervisor) and existing hardware-assisted isolation (e.g., single-root input/output virtualization). COSM isolation can be broadly employed to isolate applications, virtual machines, and containers within a given host or to isolate distinct distributed hosts. We implement COSM host-to-host isolation in a testbed with an ESMD built on a Field Programmable Gate Array (FPGA). We evaluate the host data write and read rates [bit/s] and latencies under various ESMD loads as well as write-and-read permission configurations. The introduced COSM isolation can serve as the foundation for a new sub-field of research within the general information technology (IT) security research field.

INDEX TERMS Compute Express Link (CXL), Data transmission, Data inspection, Data isolation, Shared memory, Write/read permission.

I. INTRODUCTION

A. MOTIVATION

An Externally-attached shared Memory Device (ESMD) can be used for low-latency external Inter-Process Communication (IPC) [3]–[8]. With memory sharing via an ESMD it is possible to achieve external IPC at high bitrates because memory sharing involves read and write operations in the same memory region, thereby avoiding the overhead of copying data between different address spaces [9]–[15]. However, current implementations of memory sharing lack hardware isolation mechanisms. Hardware isolation mechanisms provide physical isolation, thereby increasing data security.

The recently developed Compute Express Link (CXL) technology [16]–[21] enables the sharing of a common exter-

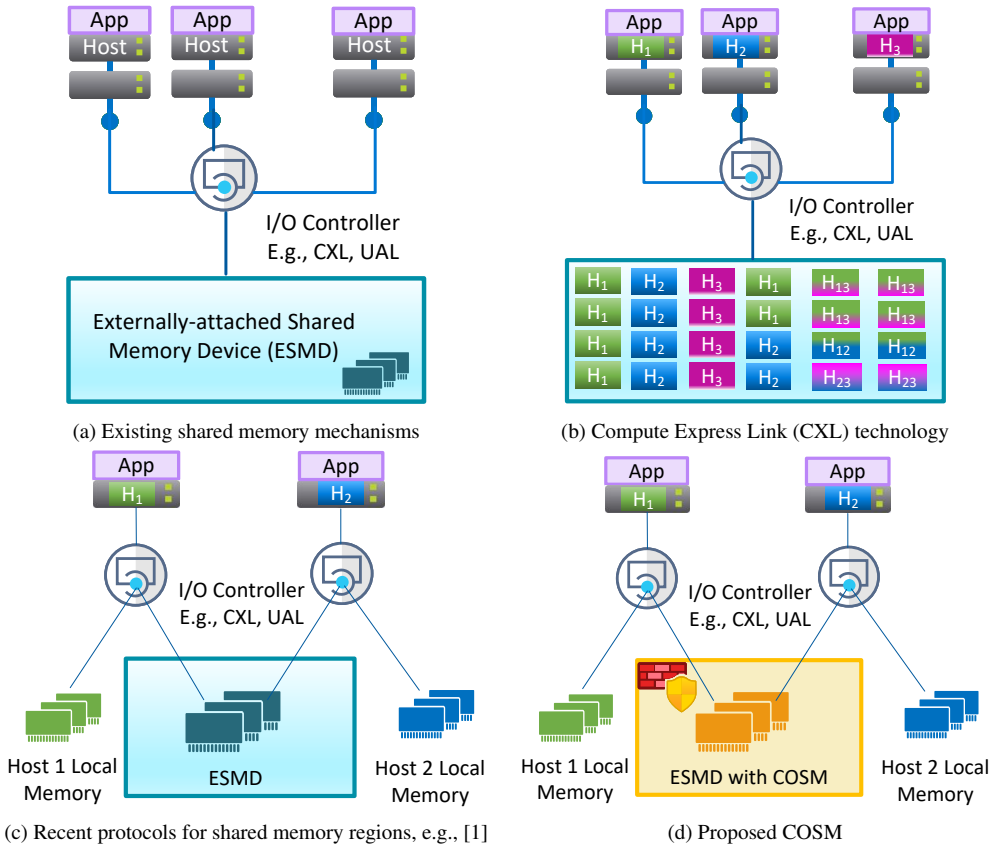


FIGURE 1: Illustration of original contribution of COntrolled Shared Memory (COSM) with respect to existing state-of-the-art: (a) Existing shared memory mechanisms allow multiple hosts to share an Externally-attached Shared Memory Device (ESMD) via direct I/O or network I/O [2]; (b) For instance, the Compute Express Link (CXL) technology allows to designate distinct memory regions as private memory for a specific host or as shared memory; (c) Recent research, e.g., [1], has focused on protocols and logic for (time) synchronizing the access of distinct hosts to shared memory regions, however, data security aspects have *not* been addressed; (d) Proposed COSM fills the research gap on controlling (restricting) the access of distinct hosts to the shared memory regions via specific read and write permissions.

nal memory device (appliance) via direct input/output (I/O), e.g., Peripheral Component Interconnect Express (PCIe) and Double Data Rate (DDR), or network I/O, e.g., Transmission Control Protocol/Internet Protocol (TCP/IP) networking and Remote Direct Memory Access (RDMA), see Fig. 1(a). From an architectural perspective, a CXL (hardware) switch allows the hosts to access the shared memory. Thereby, distinct memory regions can be assigned to individual hosts or to be designated as shared memory, see Fig. 1(b).

From a protocol perspective, illustrated in Fig. 1(c), a host accesses the shared memory via a CXL I/O controller. The CXL I/O controller functions essentially as a root port for the shared memory and implements the CXL access control logic. The main functionality of the CXL access control logic is to coordinate the accesses to the shared memory, e.g., semaphores that (time) synchronize the memory accesses.

However, to the best of our knowledge, the security aspects of the data accesses for achieving data communication between distinct hosts via shared memory, e.g., based on CXL protocol mechanisms, have *not* been addressed to date. As elaborated in Section I-B, the security and isolation mechanisms of the existing CXL protocol mechanisms are limited to the memory regions that are assigned to a specific host, e.g., a memory region assigned to host H_1 can only be accessed by host H_1 (and not by any other host). In contrast, the existing CXL protocol mechanisms do *not* restrict the read and write accesses to the shared memory in any way; that is, currently any host that is configured to access a given shared memory region can freely write to and read from the shared memory region.

To address this security issue we design the COntrolled Shared Memory (COSM) framework, see Fig. 1 (d). COSM restricts the read and write access of the different hosts to a given shared memory region according to host-specific read and write permissions, as elaborated in Section III.

B. RELATED WORK

Memory disaggregation [22]–[25] refers to the architectural paradigm that decouples CPU and memory resources, increasing flexibility. For example, the Externally-attached

Shared Memory Device (ESMD) in Figure 1(b) is decoupled from the CPU resources on the hosts [26]–[32]. However, achieving memory sharing via ESMDs between multiple hosts that are managed by different operating systems is difficult. This is because different operating systems behave differently and a common protocol is required to achieve communication between the different operating systems.

Recent advances in interconnects, such as Compute Express Link (CXL) [2], [33], have allowed interoperability between operating systems through memory sharing and memory pooling. Jain et al. [1] have recently provided hardware and software designs consisting of CXL to achieve memory sharing between multiple hosts that are managed by different operating systems. However, the Jain et al. [1] design lacks strict isolation mechanisms, which are critical for secure communications and transactions. In particular, in the Jain et al. [1] design, the conventional hypervisor [also referred to as Virtual Machine Manager (VMM)] or the fabric manager in the CXL switch are tasked with for providing isolation; this isolation is limited to memory address-space management that identifies and allocates the different memory regions and to synchronization so as to avoid race conditions when multiple hosts access a given memory region.

C. OVERVIEW OF NOVEL COSM ISOLATION

COSM provides a framework for dynamically configurable strict (near-air-gap) isolation between the multiple hosts that share data via an ESMD. Specifically, COSM regulates the write-and-read permissions of the various hosts to/from the ESMD via the memory controller of the ESMD.

1) COSM vs. Packet-Level Security (E.g., Firewalls)

Existing solutions for isolating hosts (and domains consisting of multiple hosts), i.e., so-called Cross-Domain Solutions (CDS) rely on other technologies; to the best of our knowledge, there is no prior CDS based on shared memory. The existing CDS are mainly limited to low-level packet checking (i.e., checking within the context of individual packets or observed packet flows).

The COSM framework's data-based inspection provide higher degrees of freedom compared to packet-level security mechanisms, such as Deep Packet Inspection (DPI) firewalls [34]–[40]. COSM operates at the shared memory level, enabling a broader and deeper analysis of entire data contexts rather than merely network packet traffic payloads. In particular, DPI firewalls are restricted to inspecting packets in transit, often limited by packet fragmentation, encryption, and protocol-layer constraints; whereas, COSM inspects data in-transit within the ESMD, with data residing in the ESMD, allowing full access to entire files, application-layer data, and structured in-memory datasets. This enables COSM to make context-aware decisions, detecting threats based on complete file content, metadata, or structured application data, rather than relying on fragmented network streams. Moreover, COSM operates in-memory, meaning it does not introduce network latency, making it more efficient for high-

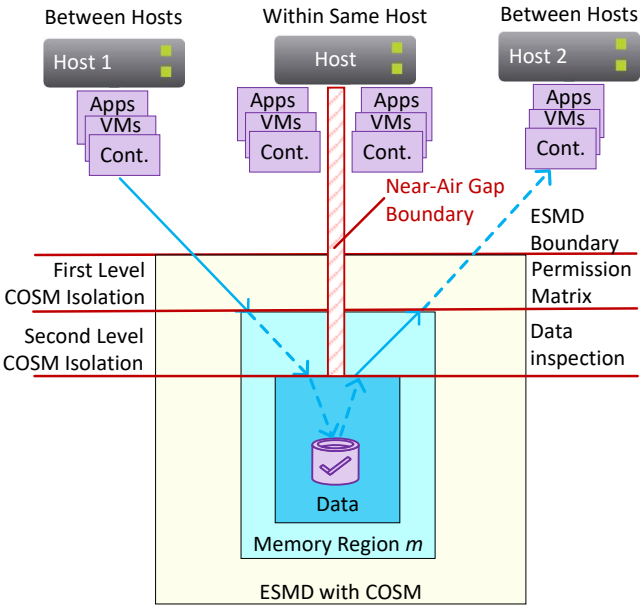


FIGURE 2: Overview of COSM Isolation: Software applications, VMs, and containers are generally isolated by OS and hypervisor principles; however, there are no strict isolation guarantees between two software entities [in contrast to the stricter air-gap (physical) isolation]. COSM aims to provide near air-gap level isolation by enabling two levels of isolation based on: *i*) a permission matrix, which enforces reads and writes blocks, and *ii*) data inspection, which enforces data integrity, performs data manipulations (e.g., deletions), and performs flow management by blocking data from being read or (re)written in a specific memory region, specific to the data type and the data itself.

throughput applications, real-time AI/ML workloads, and industrial control systems. In contrast, DPI firewalls require constant packet reassembly, deep protocol parsing, and can be bypassed through encryption or tunneling techniques; whereas, COSM enforces access control and filtering at the memory layer itself, preventing unauthorized data exposure before the data is even processed by the host applications. This makes COSM a suitable choice for zero-trust architectures, in-memory computing security, and secure multi-host environments where traditional network-based filtering is insufficient.

2) COSM vs. Software Isolation (Apps, VMs, Containers)

An ESMD with COSM isolates multiple hosts by enforcing memory segregation at the hardware level rather than relying on software-based isolation mechanisms, such as Virtual Machines (VMs), containers, and OS-level sandboxing. Traditional software isolation methods depend on hypervisors, kernel enforcement, or process-level boundaries, which are susceptible to software vulnerabilities, side-channel attacks, and misconfigurations [41], [42]. Containers, for example, share the same OS kernel, making them vulnerable to escape

attacks that allow a compromised container to gain unauthorized access to other resources. VMs have better isolation compared to containers; VMs are enabled by hypervisors, however, VMs are still vulnerable to hypervisor-level exploits and shared resource contention issues. In contrast, an ESMD with COSM enforces strict memory region isolation in hardware, ensuring that even a fully compromised OS or hypervisor cannot violate memory access restrictions. This makes ESMD-based isolation particularly valuable for high-security environments where absolute data and compute separation into independent domains is critical, such as defense, finance, and critical infrastructure systems.

Additionally, COSM's controlled access mechanism allows fine-grained permission enforcement at the memory level, ensuring that even within a shared memory environment, only authorized hosts can write to or read from designated regions. This eliminates the risk of unauthorized data leaks or inter-process interference that commonly affects software-based isolation techniques. The lack of reliance on an OS or software-based access control further reduces attack surfaces, making an ESMD with COSM inherently more secure against zero-day software exploits [43], such as privilege escalation, kernel vulnerabilities, or memory-based attacks (e.g., buffer overflows). This level of control is particularly advantageous for applications requiring deterministic performance and security, where software-based isolation might introduce unpredictability due to OS-level scheduling and resource management overhead.

3) COSM vs. Hardware-Assisted Isolation

Compared to hardware-assisted isolation techniques, such as Single Root I/O Virtualization (SR-IOV), Software Guard Extensions (SGX), and Trusted Execution Environment (TEE), an ESMD with COSM achieves a higher degree of isolation by physically segmenting memory access at the hardware controller (of the ESMD) supported by permission matrix, rather than relying on trust zones within a shared memory hierarchy achieved by software based memory management. Hardware-assisted isolation technologies, such as SR-IOV, provide isolation for virtualized I/O devices but do not prevent shared memory leaks across instances [44]–[46]. Whereas, SGX and TEE primarily focus on secure enclaves within an application's address space rather than providing full system-wide memory segregation. On the other hand, an ESMD with COSM enforces isolation at the level of the memory access controller of the ESMD, preventing unauthorized memory access even from privileged system software or compromised firmware of the host. This achieves near air-gap levels of security, where physical separation between memory regions mimics the effect of disconnected systems while maintaining shared memory efficiency.

This near air-gap isolation capability makes ESMD with COSM highly applicable in modern computing paradigms, such as network slicing and disaggregated service functions (e.g., microservices). In network slicing, multiple tenants operate in logically separate network partitions while sharing the same physical infrastructure. COSM's ability to enforce strict memory and access controls ensures that network slices remain completely isolated at the memory level, preventing cross-tenant data leakage or interference. Similarly, in microservices-based architectures, where different services execute independently but may require shared data exchange, an ESMD with COSM allows secure, high-performance memory sharing without exposing sensitive data to unauthorized services. This makes an ESMD with COSM well-suited for multi-tenant cloud environments, high-security computing clusters, and distributed edge computing architectures where software or hardware-assisted isolation alone is insufficient to guarantee true data and compute separation.

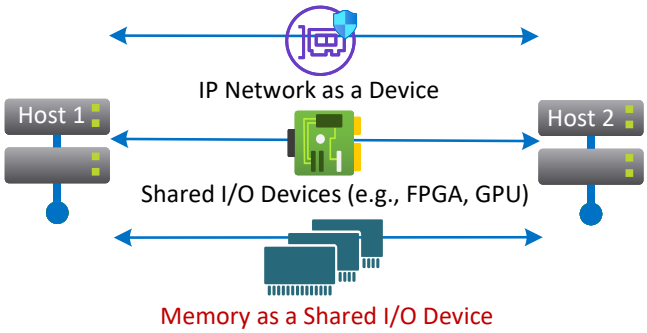


FIGURE 3: Illustration of existing host interconnection options: (i) traditional IP network with routers, switches, and gateways, (ii) an I/O component, such as FPGA or GPU, is connected to multiple hosts at the same time, and (iii) a shared memory component, which is the focus of this article.

II. BACKGROUND ON HOST INTERCONNECTIONS, ISOLATION AND SHARED MEMORY

A. INTERCONNECTING OPTIONS FOR HOSTS

To achieve data communication (exchange of information) or share resources between multiple hosts, a connection between multiple hosts must be established. The connection could be physical or virtual. Figure 3 illustrates the common interconnection options that include: Traditional Internet Protocol (IP) network, and a shared Input/Output (I/O) device.

1) Traditional IP Networks

The traditional IP network facilitates host-to-host communication based on the Transmission Control Protocol/Internet Protocol (TCP/IP) or Open Systems Interconnection (OSI) model. Each host consists of one or more virtual or physical Network Interface Cards (NICs) that can use a network socket to establish a connection between hosts. One such example is Intel's Infrastructure Processing Unit (IPU). The connection between hosts in a traditional IP network could potentially involve one or more intermediary devices, such as routers, switches, or gateways, when IP subnets are involved. An IP network has been the most commonly used interconnecting option for hosts for several decades. However,

communication via an IP network has several disadvantages, including but not limited to lower levels of security, low reliability, and high latency [47]–[49].

2) Shared I/O Devices [50]

Interconnection via an I/O device utilizes an I/O component, e.g., based on an FPGA or Graphics Processing Unit (GPU), that is connected to multiple hosts at the same time, whereby the I/O component anchors the data transfer. Multi Root I/O Virtualization (MR-IOV) [51] is a technology that allows multiple hosts to connect to an I/O device, such as a Solid State Drive (SSD), by leveraging Non-Volatile Memory express (NVMe) and a Peripheral Component Interconnect express (PCIe) switch. Markussen et al. [52] have designed their own distributed NVMe driver to achieve sharing of an SSD (in general, NVMe storage devices) between multiple hosts. However, using secondary memory, such as SSD, primarily for communication is not as fast as using primary memory, such as Random Access Memory (RAM).

a: Primary Memory Device as a Shared I/O device

Some primary memory devices can be used as a shared I/O device and allow multiple hosts to share the same memory region, thereby reducing latency. Recently, Jain et al. [1] described how CXL 3.0 can be used to allow communication between multiple hosts via a shared memory device. Jain et al. have shown how to achieve memory sharing using software modules, such as OpenSHMEM, and their custom framework. Moreover, they have implemented hardware-enabled memory sharing that includes address remap support and access control logic. However, the approach of Jain et al. [1] does not provide strict isolation, which is critical for achieving secure communications.

B. SHARED MEMORY INFRASTRUCTURES

1) Isolation Technologies

Providing isolation is a fundamental approach to protecting data from information leaks and thefts [53]–[57]. Information leaks and thefts can result in cyber-attacks, such as ransomware attacks, which can cost millions of dollars to recover the data [58], [59]. Implementing high levels of isolation results in high levels of security [60]–[62]. Isolation can be classified into software and hardware isolation.

Generally, isolation technologies for shared memory can be categorized into: (i) software isolation, which provides logical separation within a given host (e.g., for the different virtual machines within a host); (ii) hardware-assisted isolation, which provides physical isolation (also within a given host); and (iii) hardware isolation, which also provides physical isolation, but between distinct hosts.

a: Software Isolation

Software technologies, such as operating systems (OS), Virtual Machines (VMs), containers, and network slicing, provide software isolation using code-based techniques. For instance, Memstrata [32] employs software-based isolation (although the memory tiers in Memstrate are managed with hardware). The increase in flexibility and scalability as well as lower cost are some of the advantages of implementing software isolation. However, software isolation only provides logical separation (isolation), and thereby is prone to software-based attacks [63]–[66]. Therefore, implementing solutions that provide software isolation in places that require high levels of security results in poor security features.

b: Hardware-Assisted Isolation

Hardware-assisted isolation provides physical separation (isolation). Intel's Software Guard Extensions (SGX) and Trust Domain Extensions (TDX) [67] as well as AMD's Secure Encrypted Virtualization (SEV) are some examples of hardware-assisted solutions that can provide physical isolation by creating a Trusted Execution Environment (TEE). However, hardware-assisted isolation can only be applied within a host and not between hosts.

c: Hardware Isolation: Controlled Domain Transfers

Presently, hardware isolation exists only in the context of so-called controlled domain transfers [53], which we briefly review in the following. A domain is commonly defined as a set of system resources, e.g., hosts, to which certain users have prescribed access rights as governed by security policies. A Cross-Domain Solution (CDS) helps to share information securely between domains or to view information present in another domain without any intrusions, information leaks, or information thefts [53]. One such CDS is a data diode, which only allows unidirectional flow of data, thereby providing physical isolation. Moreover, implementing data diodes via policy enforcement (control logic) could be considered a controlled domain transfer technology because of their ability to control data flows.

To the best of our knowledge, this study is the first *to design and evaluate a shared memory based hardware isolation of distributed hosts*.

2) I/O Attached Memory Device

For many decades, memory has been attached directly (combined) to the Central Processing Unit (CPU), reducing utilization, flexibility, and scalability in modern applications, such as datacenters. With the introduction of memory disaggregation, it is now possible to separate memory components from the CPU. This separation allows multiple CPUs to access a single memory module (flexibility) whenever required, increasing utilization. Also, this separation allows a CPU to increase its memory capacity (scalability) and reduces the cost.

Memory disaggregation has become prevalent with the introduction of CXL, an interconnect that mainly focuses on memory expansion, heterogeneous compute, and disaggregation of system resources [68]. CXL acts as an I/O, a bridge between the host and the memory device. Although there are many available interconnects and networking technologies,

such as InfiniBand and Ethernet, CXL stands out due to its ability to achieve low latency and high bandwidth with cache-coherent load-store semantics [68]. In addition, the CXL fabric manager handles resource allocation. The I/O attached memory device is considered as the host's own memory due to the illusion created by CXL.

3) I/O Attached Memory Pooling

Memory pooling refers to the aggregation of memory modules. For example, multiple ESMD devices can be aggregated together. Each ESMD device will have its own memory controllers (typically, each memory (hardware) module inside an ESDM has its own memory controller). These memory controllers are attached to the I/O controller. The difference between memory sharing and memory pooling is that memory sharing occurs between hosts via an ESMD whereas memory pooling helps to aggregate the individual memory modules (ESMD devices) to increase the memory capacity. For example, four 8 GB DRAMs can be combined via memory pooling to form a 32 GB DRAM.

4) Externally-attached Shared Memory Device (ESMD)

An ESMD is a memory device that is not tightly coupled to the CPU of the host. With the CXL standard, it has become possible to connect multiple hosts to multiple end-devices, such as accelerators, memory expanders, and smart I/O communication. An ESMD is a type of memory expander, which can be realized using a reconfigurable Field-Programmable Gate Array (FPGA). It is also possible to realize an ESMD using other Integrated Circuits (ICs), such as Application-Specific Integrated Circuit (ASIC). Importantly, depending on the reconfigurability characteristics of the underlying ESMD technology, the COSM framework can achieve different levels of air-gapping that span, for instance, from hardware (physical) air-gapping with physical, e.g., circuit-level, reconfiguration, to logical air-gapping with reconfiguration of the memory controller permission settings.

Figure 4 illustrates the complete architecture of an ESMD consisting of memory (DRAM), memory controllers, ASIC accelerators (allowing the performance of ASIC in FPGA), CPU cores, as well as L2 and L3 caches. The ESMD also includes a packet processing engine which comprises packet processing elements, such as security engines, security protocols, RDMA, and InfiniBand to support direct remote memory access between the ESMD and the host if needed.

Figure 5 illustrates the memory map of an ESMD. The hosts connected to the ESMD, and the applications that run on the hosts. Application A runs on host 1, while applications B and C run on host 2. Suppose application A of host 1 needs to communicate or exchange data with applications B and C of host 2; the most commonly used method to achieve the communication would be a traditional IP network. However, there would be security and latency issues. An alternative to traditional IP networking is to use an externally-attached shared memory device (ESMD) to conduct read and write operations at the same memory region, thereby reducing latency.

a: Dedicated Memory on ESMD

An ESMD allows hosts to use the ESMD memory regions as the hosts' own memory. This provides increased flexibility and scalability to the hosts. For example, if host 2 is out of its local primary memory, i.e., primary memory that is located directly on the host device, then host 2 can now utilize ESMD memory as its own memory as shown in Figure 5. Suppose application C running on host 2 uses memory region M of the ESMD. Memory region M is dedicated to application C of host 2, meaning that memory region M is basically isolated from other hosts, even isolated from other applications running on the same host.

b: Shared Memory on an ESMD

Figure 5 also illustrates an example of memory sharing between application A running on host 1 and application B running on host 2. Memory regions 1, 2, 3, and 4 are shared between hosts 1 and 2 whereby memory regions k and l are utilized by application A running on host 1 and application B running on host 2. However, with the existing shared memory techniques, e.g., [1], there is no strict isolation during memory sharing between different hosts, i.e., between hosts 1 and 2 in this case. As a result, information leaks or thefts may occur, especially in an environment where host 1 and host 2 belong to different domains.

III. PROPOSED COSM DESIGN

A. NEED FOR CONTROL OF SHARED MEMORY

An Externally-attached Shared Memory Device (ESMD), e.g., based on an FPGA, can achieve external IPC (data transfer between hosts) at low latency due to read and write operations occurring in the same memory region. In this beta era, data privacy is significant. An individual's data is considered as an asset. Protecting asset is of prime importance. ESMD has software isolation mechanisms provided by the operating system to protect the data. However, physical isolation is considered as a better form of isolation than software isolation, as physical isolation provides physical separation. In summary, ESMD lacks physical isolation or separation. Therefore, we propose COSM, a hardware-based memory device that provides physical (air-gap) separation. An ESMD with COSM expresses data diode functionality in the logic of the ESMD memory controller to provide physical separation between hosts (or domains consisting of multiple hosts). COSM creates memory-channels for the transactions to flow, thereby making COSM a memory-based solution.

B. BENEFITS OF USING MEMORY DEVICE

The main benefits of the using memory devices compared to traditional IP networking methods are:

1) Memory-based transactions offer more reliable data transfer and simpler protocol operations for retransmissions and data tracking compared to IP network

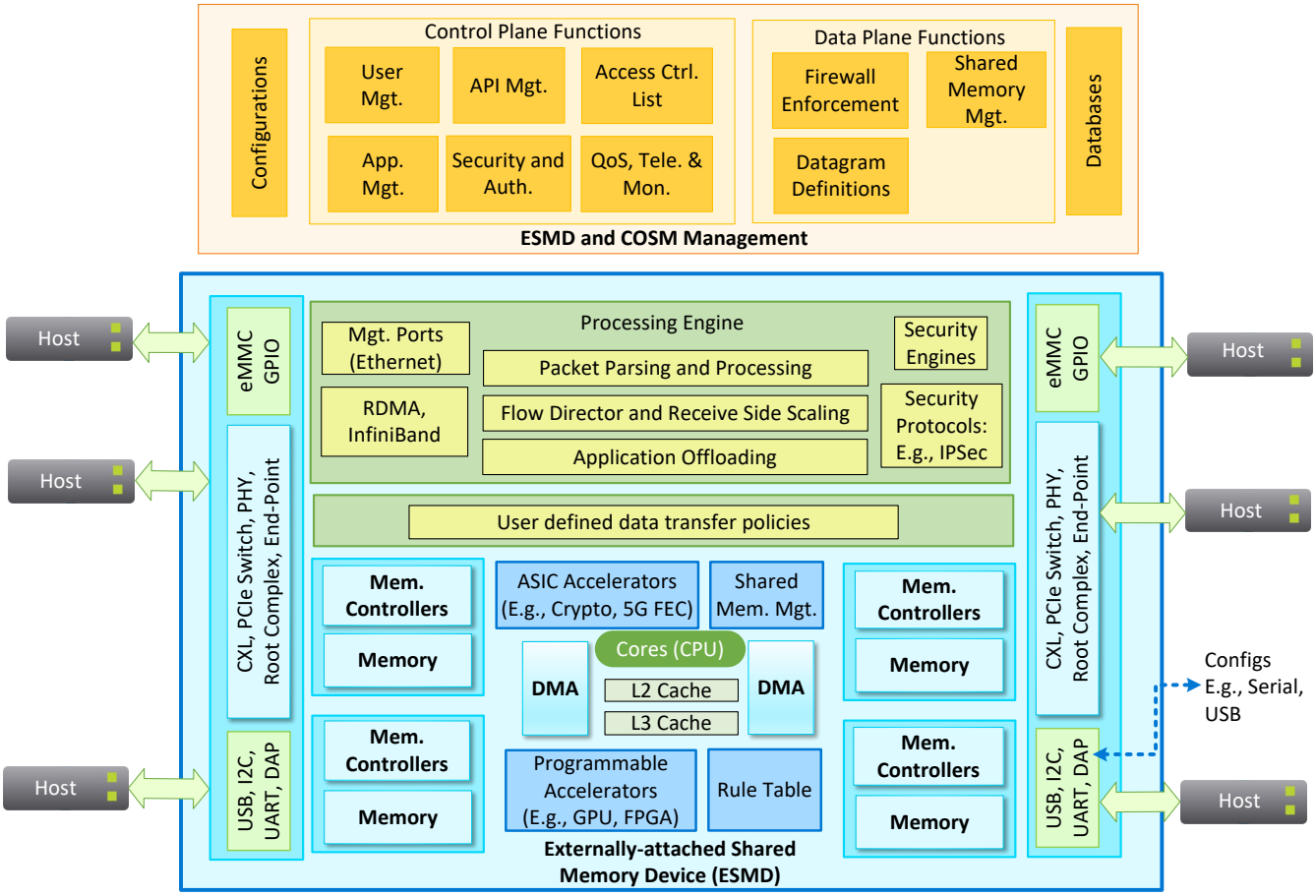


FIGURE 4: Overview of management plane (top with yellow shading) and reference hardware implementation (bottom with blue shading) of an Externally-Attached Shared Memory Device (ESMD). The management plane consists of a control plane which manages the setting up of ESMD memory to hosts, and a data plane which manages the data access and transfer properties of the ESMD memory by the hosts. The depicted reference hardware implementation supporting software-defined applications for Controlled Shared Memory (COSM) management involves multiple components working together to provide secure, efficient, and scalable access to shared memory regions. The hardware must enable access control, high-speed data transfer, and protocol compatibility while maintaining data integrity and security.

packets that have to navigate through switches and gateway which can drop the packets. As a result of simpler operations, memory-based solutions can offer high-performance communication techniques between interconnecting hosts through shared memory

2) The memory interfaces in the memory devices are enforced with access controls and policies for secure operations, such as an enabling data-diode [69] which restricts communications to only one direction with access controls, such as write-only, read-only, or read-/write permitted.
3) These memory interfaces can be a standard implementation and could be open-sourced for ease-of-use, community adoption, and public participation in technology contributions without compromising the security and isolation properties of the data transactions. The open implementation provides transparency of communication procedures over open interfaces to identify any security vulnerabilities.
4) One of the most important aspects of memory devices is the support for application-defined communication protocols over open interface definitions (and open implementation). Application-defined communication protocols enable applications to create their own datagram format, segmentation, encryption, and flow control mechanisms that are decoupled from the interfaces and memory buffers. In a simple analogy, the memory-based solution only provides the systems with physical memory space to communicate, but how to communicate is defined by the applications (independent of the memory-based solution). Figure 4 illustrates the different planes along with the management techniques present in the memory-based solution architecture and explains how the external memory device reports itself

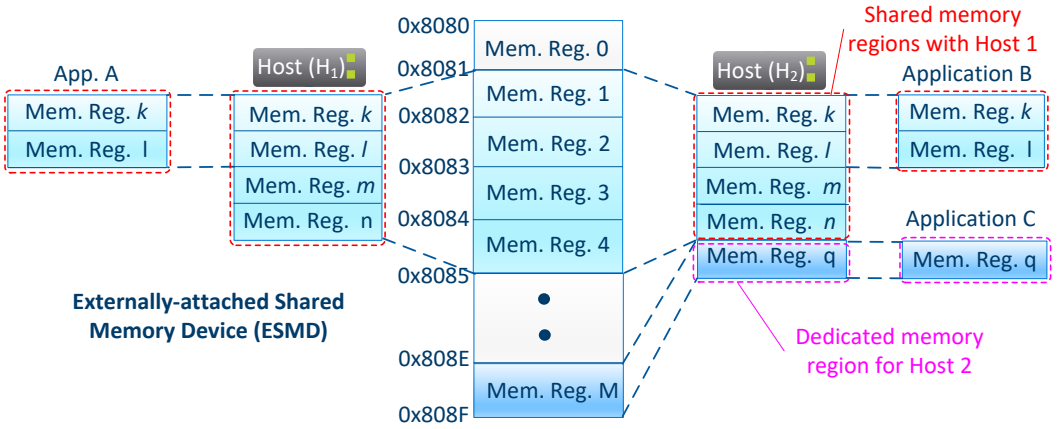


FIGURE 5: Illustration of COSM sharing of memory regions between hosts 1 and 2. The control plane sends the instructions to the ESMD to allocate memory regions 1 through 4 to the host 1 OS, and the host 1 OS further allocates these shared memory regions to application A. To facilitate the sharing of data between host 1 and host 2, the same memory regions, i.e., memory regions 1 through 4, are also shared with host 2 OS, which then eventually assigns these memory regions to application B.

as a memory device.

1) Permission Matrix-Based Filtering

The COSM framework in the ESMD provides a robust permission matrix that allows fine-grained filtering and control at the granularity of individual hosts and individual memory regions. This mechanism ensures that each host accesses the shared memory with tailored read/write permissions based on the specific memory region being accessed. For example, a host performing real-time updates to a critical shared memory region may have write access, while other hosts are restricted to read-only permissions to prevent accidental overwrites or corruption. This fine-granular control enables precise enforcement of access policies, minimizing the risks of unauthorized data manipulation or accidental interference in multi-host environments.

Also, permission matrix filtering facilitates dynamic and context-aware memory management. For instance, the ESMD can update permissions on the fly based on the operational state of the system or application requirements, such as granting temporary write access to a host for a specific task and then revoking the permission once the task is complete. This flexibility is particularly important in scenarios involving hierarchical or distributed memory allocation, where different hosts or processes may have varying levels of privilege. By enabling this fine-granular level of control, the ESMD ensures both security and performance, allowing shared memory to be utilized efficiently without compromising the integrity of data or system operations.

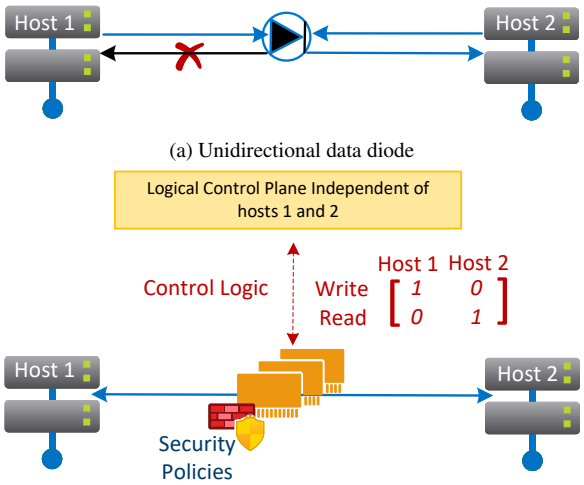
2) User-Defined Data Based Inspection and Filtering

The ESMD with COSM can enable user-defined data handling and inspection-based filtering to effectively manage and validate data passing through shared memory regions. For instance, applications can implement logic within the ESMD to inspect data in real-time as it is read or written by hosts. This feature can be used to verify parameters in data, such as validating that a specific field falls within a predefined range or ensuring that a packet header conforms to a particular protocol format. By incorporating inspection-based filtering at the ESMD level, developers can offload such validation tasks from individual hosts, improving overall system efficiency and consistency. This is particularly useful in high-assurance environments, such as healthcare or financial systems, where strict validation of data integrity is crucial to prevent errors or misuse.

Moreover, inspection-based filtering enables advanced use cases, such as enforcing application-level policies or implementing security features. For example, the ESMD can block or flag data that does not meet predefined criteria, such as unauthorized parameter values or invalid checksum calculations. This capability can be extended to support application-specific protocols by parsing user-defined data structures and enforcing format compliance. Embedding this functionality into the ESMD ensures that data integrity and compliance are maintained across the system while allowing hosts to focus on their core application logic without additional overhead for data inspection.

C. COSM PRINCIPLES

Figure 6 illustrates the basic idea of COSM. COSM is a general framework that provides policy enforcement capability for memory that is shared by multiple hosts. In particular, a COSM control logic enforces the access control to the memory regions within an Externally-attached Shared Memory Device (ESMD). The access control model is formalized in Section III-D. Here, we only briefly note that a value of 1 for host 1 for write capability along with the value of 1 for host 2 for read capability indicates the permitted function, while 0 for the remaining write-and-read permissions indicates the blocked function, effectively enforcing directional (one-way) traffic over the ESMD device, emulating the behavior of a



(a) Unidirectional data diode

(b) COSM configuration as unidir. data diode

FIGURE 6: Proposed COntrolled Shared Memory Device (COSM): (a) Conventional uni-directional data diode function, e.g., for CDS in IP network; (b) Memory as a shared I/O device with write and read permissions governed by COSM.

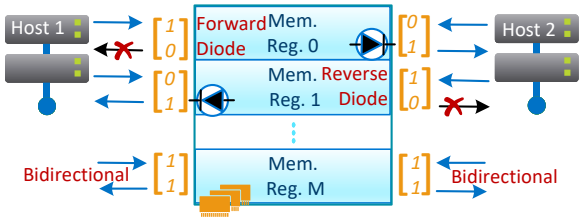
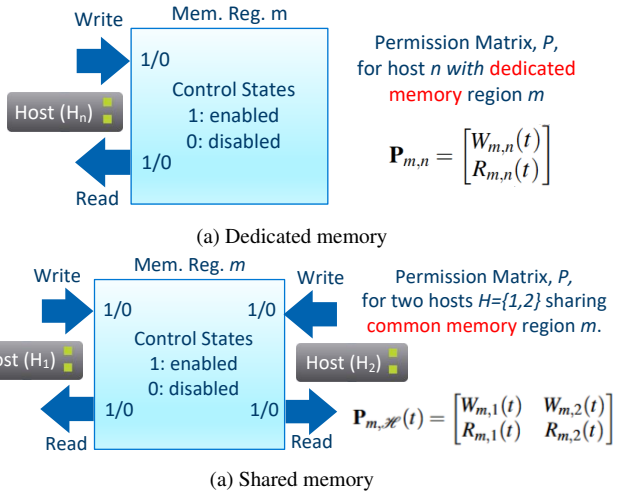


FIGURE 7: Flexible dynamic hardware isolation provided by configurable COntrolled Shared Memory (COSM): The vector $\begin{bmatrix} W(t) \\ R(t) \end{bmatrix}$ represents the write and read permissions (1 = permitted; 0 = prohibited) of a host at time t. The top part of the figure illustrates the COSM configuration for forward diode functionality [host 1 sends (writes), host 2 receives (reads)] and reverse diode functionality [host 1 reads, host 2 writes], which are forms of a so-called controlled air-gap. The COSM configuration in the bottom part of the figure allows for unrestricted bidirectional communication.

data diode for the direction of data flow from host 1 to host 2.

The basic isolation principle of COSM is further illustrated in Figure 7. The top part of Figure 7 illustrates the configuration of COSM to function as a data diode, which provides a so-called controlled air-gap. Specifically, the illustration at the top of Figure 7 shows that memory region 0 is currently configured as a forward diode that permits only one-way data communication that flows from host 1 to host 2; whereas, host 2 can currently not send any data to host 1 via memory region 0. In particular, host 1 is currently permitted to write data into memory region 0 (but not to read from memory region 0); whereas, host 2 is currently permitted to read data from memory region 0 (but not to write to memory region 0). This current COSM configuration of the write-and-read permissions gives rise to the forward (one-way) data diode functionality of memory region 0.



(a) Dedicated memory

(a) Shared memory

FIGURE 8: Illustration of write and read permission matrices for dedicated and shared memory.

Complementarily, memory region 1 in the top part of Figure 7 is currently configured as a reverse data diode (from the perspective of host 1), i.e., memory region 1 can currently only be used for data communication from host 2 to host 1, but not for data communication from host 1 to host 2. This one-way host 2-to-host 1 data diode functionality is enforced by the COSM write-and-read permissions for memory region 1: host 2 is permitted to write (not read) while host 1 is permitted to read (not write).

The bottom part of Figure 7 shows that memory region M is currently configured for bidirectional communication by permitting both host 1 and host 2 to freely write to and to read from memory region M. We emphasize that the write-and-read permissions for the different memory regions can be dynamically configured with COSM, e.g., the write-and-read permissions can be set to $\begin{bmatrix} W(t) \\ R(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ anytime to create an air-gapped environment.

We note that the notation of air-gap in COSM corresponding to time periods when the write-and-read permissions are all set to zero (0) for a host means that the host's connection to the shared memory region is disconnected (separated) according to the level of air-gap that is supported by the underlying memory technology (controller and circuit characteristics). That is, the all-zero permission setting enforces that no data can pass between the host and the shared memory region.

D. COSM WRITE AND READ PERMISSIONS MODEL

1) Notation

We denote $\mathscr{H} = \{H_1, H_2, \ldots, H_n, \ldots, H_N\}$ for the set of distinct hosts that participate in the COSM-controlled memory sharing. We denote $\mathscr{M} = \{M_1, M_2 \ldots, M_m, \ldots, M_M\}$ for the set of distinct (non-overlapping) memory regions. A given

memory region M_m is characterized by the memory address of the start of memory region M_m and the memory address of the end of memory region M_m.

Consider a given memory region M_m, $m = 1,2,\ldots,M$, and a given host $H_n, n = 1,2,\ldots,N$; whereby the host H_n is configured for sharing the memory region M_m by an underlying memory sharing technology, e.g., CXL Host H_n has two degrees of freedom for sharing the memory region M_m: write permission and read permission, which may change with time t, i.e., we consider a time-dependent control of the shared memory region. We denote $W_{m,n}(t) = 1$ if host H_n is permitted to write into memory region M_m at time t; complementarily, $W_{m,n}(t) = 0$ indicates that H_n is prohibited from writing into memory region M_m at time t. Analogously, we denote $R_{m,n}(t) = 1$ if host H_n is permitted to read from memory region M_m at time t; complementarily, $R_{m,n}(t) = 0$ indicates that H_n is prohibited from reading from memory region M_m at time t. We summarize these permissions for memory region M_m and host H_n in the permission vector: $\mathbf{P}_{m,n}(t) = \begin{bmatrix} W_{m,n}(t) \\ R_{m,n}(t) \end{bmatrix}$.

For a particular (given) memory region M_m, a particular sharing permission (or restriction) policy among a set $\mathscr{H}$ of N (sharing-configured) hosts is created by combining the permission vectors to form a sharing permission matrix of dimension $2 \times N$:

$$\mathbf{P}_{m,\mathscr{H}}(t) = \begin{bmatrix} W_{m,1}(t) & W_{m,2}(t) & \cdots & W_{m,n}(t) & \cdots & W_{m,N}(t) \\ R_{m,1}(t) & R_{m,2}(t) & \cdots & R_{m,n}(t) & \cdots & R_{m,N})(t) \end{bmatrix}. \tag{1}$$

2) Transfer Channel Characterization

According to the characteristics of the sharing permission matrix $\mathbf{P}(t)$, a particular (given) memory region M_m forms essentially a "channel" for the transfer of data between a set $\mathscr{H}$ of distinct hosts according to a host-specific time-dependent sharing permission (restriction) policy. For illustration, consider a pair of $N = 2$ distinct hosts, i.e., $\mathscr{H} = \{H_1, H_2\}$. We illustrate the practically relevant 2×2 permission matrices as follows. The permission matrix

$$\mathbf{P}_{m,\mathscr{H}}(t) = \begin{bmatrix} W_{m,1}(t) & W_{m,2}(t) \\ R_{m,1}(t) & R_{m,2}(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{2}$$

completely blocks hosts H_1 and H_2 from communicating with each other. In contrast, the permission matrix $\mathbf{P}_{m,\mathscr{H}}(t) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ effectively creates a "wire" channel for host H_1 and H_2 to freely communicate with each other.

The permission matrix

$$\mathbf{P}_{m,\mathscr{H}}(t) = \begin{bmatrix} W_{m,1}(t) & W_{m,2}(t) \\ R_{m,1}(t) & R_{m,2}(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3}$$

effectively creates a data diode for unidirectional communication from host H_1 (which has write permission into M_m) to host H_2 (which has read permission from M_m); whereas, host H_2 is blocked from communicating data to host H_1. Conversely, the permission matrix $\mathbf{P}_{m,\mathscr{H}}(t) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ effectively creates a data diode for unidirectional communication from host H_2 (which has write permission into M_m) to host H_1 (which has read permission from M_m); whereas, host H_1 is blocked from communicating data to host H_2.

While we considered controlling the access to the shared memory on the basis of hosts in this section, we note that multiple hosts can be combined to form a so-called "domain". Commonly, a domain is defined as a set of system resources, e.g., hosts, to which certain users have prescribed access rights as governed by security policies

E. MANAGEMENT PLANE FOR COSM

The COSM management plane for an ESMD, which is illustrated in the top part of Figure 4 manages control plane functions, such as configuration management, policy enforcements, databases, and management (app, user, API), as well as data plane functions, such as shared memory management, datagram definitions, and firewall enforcement.

1) Control Plane Unit

The control plane unit of the management plane can be realized as a central logical entity that manages control plane functions, e.g., configurations, databases, applications, as well as user and QoS policy enforcement.

a: Configuration Management

The role of the configuration management is:

1) Setting up the memory region, which includes configuring the shared memory's boundaries, size, and segmentation for specific use cases.
2) Setting up the access policies, which includes determining which hosts or devices have access to the shared memory regions and under what conditions. Then, the write-and-read permissions matrix $\mathbf{P}_{m,\mathscr{H}}(t)$ has to be configured according to the access policies for each memory region and connected host or device.

b: Policy Enforcement

Policies such as

1) Access control rules help COSM to function in a systematic way by dynamically updating permissions in response to changing conditions or policies.
2) User authentication and authorization protects the information by ensuring only authenticated and authorized users or devices can access or modify the shared memory regions.
3) Quality of Service (QoS) helps to enhance the performance by defining priorities and bandwidth allocations for different applications or users. For example, higher-priority users may get faster or more consistent access to shared memory regions.

c: Databases

COSM must know what rules are deployed on each memory region at every instance to allow or deny memory sharing. Policy and configuration databases store the necessary

information, such as access permissions, user roles, and application-specific requirements for COSM to take necessary actions related to allowing or denying access to memory at time t. Audit logs maintain a record of access attempts, configuration changes, and policy violations ensuring accountability.

d: User, Application, and API Management

Management applications provide user interfaces or Application Programming Interfaces (APIs) to configure and monitor the ESMD and COSM features. Monitoring tools help track performance, usage, and security events related to the shared memory access. API management helps to manage the APIs, to ensure secure, reliable, flexible, and scalable handling of user requests for COSM services to be provided by the ESMD. User management, on the other hand, is specifically related to individual user access, e.g., for assigning permissions. Application management takes care of updates and deployments that are needed for an application to run properly.

2) Data Plane Unit

The data plane unit of the management plane manages data plane functions, e.g., firewall, datagram definitions, and shared memory management for end-to-end data sharing and transfers between hosts. The data plane handles the actual data transfer, processing, and enforcement of the policies set by the control plane focusing on high-performance, real-time data transactions and operations, as elaborated in the following.

a: Shared Memory Management

The shared memory management comprises:

1) Memory Access that facilitates data reads and writes between hosts and the shared memory regions, adhering to the permission matrix $\mathbf{P}_{m,\mathscr{H}}(t)$ defined in the control plane.
2) Memory protection that implements hardware or software mechanisms to block unauthorized memory access in real-time.
3) Concurrency management that ensures consistency and integrity of data when multiple hosts or applications access shared memory simultaneously.

b: Data-Firewall

COSM provides a framework to perform data filtering, similar to firewalls performing packet processing. A network firewall can act on a on a limited set of packet-level information; in contrast, COSM on ESMD can implement a higher-dimensional data-firewall, where user-defined filtering of data can be performed within the context of COSM. The data-firewall inspects and filters the User-Defined Protocol Data Units (UDPDU) that enter or leave the ESMD to ensure that the data definitions as well as data transfers adhere to COSM-defined policies. In addition to data filtering, the data-firewall can block the data residing on the COSM based on a prescribed policy, so as to prevent unauthorized communication paths, or unintended data sharing with other hosts.

c: Datagram Definitions

The data format defines the structure and encoding of the datagrams that are exchanged between hosts to ensure compatibility and efficient processing. Protocol handling manages protocols for shared memory access and communications for stateful end-to-end data transfer.

d: Data Transfers, Error Handling, and Recovery

The data plane is responsible for efficient (low latency, high throughput) and reliable end-to-end data transfer between hosts using the shared memory region. The QoS enforcement applies the bandwidth and latency guarantees as defined in the control plane, ensuring fair resource allocation. Detection and response to errors, such as access violations or memory corruption, are also conducted in the data plane. In addition, the mechanisms for rollback or data recovery in case of a failure is implemented in the data plane. We note that the COSM framework does not enforce QoS at the physical layer; rather, COSM, can have some QoS mechanisms, e.g., some form of token bucket based buffer management assistance within the framework of COSM memory buffers. This buffer management assistance can aid in in-memory queue management for data-transfers so as to provide QoS and load-balancing.

3) Reference Hardware Implementation of COSM

The reference hardware implementation, which is illustrated in the bottom part of Figure 4, consists of:

1) General-purpose CPU cores to implement the software-defined applications that realize the COSM policies for the controlled access to the shared memory.
2) Generalized I/O, realized through common interconnect technologies, such as PCIe, CXL, and UCIe.
3) DMA engines that can help in the I/O data transfer, reducing the utilization of host CPU resources.
4) Accelerators that can be used to implement data-specific functions, such as encryption, compression, and error corrections.
5) Hardware Processing Engine (PE) that allows COSM IP datagram processing using shared memory regions to abstract the data transfer to emulate network interfaces for OS and applications. A PE can be employed to seamlessly use the ESMD for data transfers that are based on a wide range of data units, e.g., on Ethernet frames, IP datagrams, or User-Defined Protocol Data Units (UPDUs), within the memory-as-a-shared-I/O-device paradigm. This means that the data passing through the memory interface can be in the IP format. This IP compatibility facilitates adoption by a wide range of applications, as many cloud-native applications involve HTTPS based connections. These HTTPS

based cloud-native applications typically require network sockets and some packet processing assistance on the COSM device, e.g., in-memory compute.

The role of the notable hardware components can be listed as:

a: Input/Output Controllers

The role of I/O controllers is (*i*) to manage connectivity between external devices (hosts) and the ESMD, and (*ii*) to translate device-specific protocols into formats compatible with the shared memory system.

An ESMD device can connect with multiple hosts with different I/O controllers, such as UAL, CXL, and PCIe, simultaneously. Integrated protocol implementations on the ESMD can support user-defined protocols to facilitate the diverse data streams into shared memory-compatible formats, such as audio, static files, video, and web-traffic. For the external configuration of the ESMD device to have the independent (of connecting hosts) access capability, e.g., side-band access, a serial I/O technology (such as USB) can be used to enable the ESMD configuration as an end point such that an external management device can be connected with the ESMD management plane for programming and monitoring.

b: General-Purpose Central Processing Unit (CPU)

The role of CPUs is to implement the control plane functions in a software defined manner, which includes the management of the COSM configurations, access permissions, and high-level data flow coordination. CPU cores also handle exceptions and supervisory tasks, including logging, auditing, and policy enforcement. A multi-core CPU can run control software, including memory configuration tools, permission matrix updates, and real-time monitoring. CPU cores also assist in QoS enforcement by dynamically updating policies or throttling resource allocation based on usage. Software defined high-speed data processing pipelines on ESMD can be implemented on the general-purpose CPUs using accelerated software stacks, such as Data Plane Development Kit (DPDK) and P4-programmable engines, for efficient, low-latency operations within an ESMD device.

c: Accelerators

The role of accelerators is to offload computationally intensive tasks, such as encryption, compression, or real-time data transformations, from the CPU. This primarily reduces latency for data transfers and preprocessing tasks, especially when CPUs are busy with the software-defined tasks on the ESMD. Integrated hardware accelerators can, e.g., be FPGA, ASIC, or GPU, implementing specific workloads, such as cryptographic operations (e.g., AES, SHA) or data parsing for COSM policy enforcement.

d: User-Defined Protocol Data Unit (UPDU)

UPDU definitions allows the applications to define how data is structured and interpreted during transfer through shared memory regions, ensuring privacy among shared hosts, while also providing compatibility and standardization across different applications over multiple hosts that shared the ESMD with COSM.

The UPDU definitions include headers, and payload data formats, as well as control information between hosts for end-to-end flow control. The ESMD hardware can be configured to parse UPDUs, validate their integrity, and interpret metadata to reference the address to actual physical memory, enforce policies, and transfer the data to other hosts. For example, a UPDU can include metadata that indicates priority and access permissions, thus enabling the system to apply QoS and security policies during data processing.

e: Hardware Processing Engines (PEs)

The role of Processing Engines (PEs) can be split into two categories: (*i*) traditional packet processing, to process incoming and outgoing data packets, acting as an Internet Protocol (IP) network device implemented over a shared memory along with COSM features, and (*ii*) processing of User Defined Protocol Data Units (UDPUs), whereby applications define the data format to share between applications separated by COSM over multiple hosts.

PEs also ensure compliance with User-Defined Protocol Data Unit (UPDU) definitions based on the ESMD capabilities as well as the COSM policies to enforce the filtering actions that may dynamically change over time. In addition to permission matrix-based data-firewall enforcement, the ESMD can be enabled to inspect data headers and protocol data units to provide deep-data inspection before allowing the data transfer to other hosts over a shared memory.

f: Direct Memory Access (DMA) Engines

The role of DMAs on an ESMD device is to enable high-speed data transfers between hosts and ESMD shared memory, as well as between ESMD-accelerators and shared memory, without using the host-CPU resources. Whereby, the throughput and latency can be improved by bypassing intermediate software processing layers at the hosts and ESMD while transferring the data between hosts and ESMD. Implementing multiple DMA channels to support concurrent data streams, e.g., DMA channels per host or per memory region, can increase the overall performance of the ESMD device. These DMA engines can transfer large data blocks from host to a specific shared memory region of ESMD, or several memory regions at the same time based on permission and priority configurations, thus enabling QoS programmed into the DMA engines by the data plane management functions of the ESMD.

g: Memory Controllers

Memory controllers in the ESMD: (*i*) manage physical memory resources, ensuring efficient allocation, protection, and access control, and (*ii*) handle low-level operations such as address translation, concurrency management, and error detection.

Advanced memory controllers provide support for state-of-the-art memory technologies, including High Bandwidth Memory (HBM) and Double Data Rate (DDR) for high density, low-cost, power-efficient, high-bandwidth, and low-latency operations. The COSM functions can be implemented as integrated memory protection units or hardware-based memory management units (MMUs) to enforce permissions as per the COSM-defined rules for the shared memory regions. Hence, memory controllers are a key component in the ESMD reference hardware architecture that ensures isolation between hosts creating access-control based memory regions as defined by the control plane functions of the ESMD.

F. OPERATIONAL CHARACTERISTICS OF COSM

The operational characteristics of a shared memory device, such as the Externally-attached Shared Memory Device (ESMD) with Controlled Shared Memory (COSM) involve both ESMD-specific and host-specific functions, such as, enumeration, memory management, address management, memory access, and synchronization across both host and ESMD devices. These functions contribute to the successful end-to-end transfer of the data between hosts connected by an ESMD device via a common memory region. Figure 9 illustrates the general approach for the end-to-end data transfer between hosts connected by an ESMD device with COSM capabilities.

1) ESMD Device and I/O Controller at the Host

The COSM implementation judiciously exploits existing shared management mechanisms. During the device enumeration process of the Basic Input/Output System (BIOS) at a given host, e.g., host H_1, an externally-attached shared memory device (ESMD) is enumerated as a memory component that the Operating System (OS) on this host can use as its own memory. More specifically, the ESMD—and equivalently the memory region provided by the ESMD—that is allocated to the particular host is available to the host via its OS as if the shared memory region were the host's own memory region. The granularity of the enumeration of shared memory regions, i.e., the sizes of the shared memory regions, can be configured via memory controllers, such that a given shared memory region (of a particular size in bytes) corresponds to one BIOS-enumerated ESMD. On the other hand, from the point-of-view of the ESMD, the host representation is determined by the existence of an I/O controller within the host. The ESMD answers to the I/O controller, i.e., the ESMD is talking to a host's I/O controller, or just the "host".

Generally, a particular shared memory technology has its associated specific memory drivers. For instance, CXL-attached memory and the Linux Kernel have specific memory drivers [1] with special libraries [e.g., a special memory allocation (`malloc(...)`) library that is referred to as `shmem_alloc(...)`] in order to support multi-host applications. The specific memory drivers coordinate the involvement of multiple hosts (and correspondingly of multi-host applications) in the use of the shared memory.

2) ESMD Device and BIOS Enumeration at the Host

Enumeration is the process by which hosts discover and identify the ESMD as a shared memory resource. The ESMD exposes itself to the host system using standard communication protocols, such as CXL, UALink (UAL), PCIe, Non-Volatile Memory express (NVMe), or Ethernet. A host BIOS performs I/O enumeration, e.g., via PCIe to detect the ESMD and maps the ESMD memory regions into the host's memory space for further configuration. The host's OS (or the driver) queries the ESMD during the boot or initialization process to identify the ESMD capabilities, including supported memory regions, access permissions, memory block sizes, and configurations. Device descriptors are exchanged, providing the host with details, e.g., device ID, shared memory capacity, and QoS capabilities. Based on the BIOS feedback, the host OS typically marks the ESMD memory regions as a dedicated NUMA node, other than those memory modules which are already enumerated as part of the host system. Hence, the ESMD can use standardized enumeration protocols [e.g., PCIe Base Address Registers (BARs)] to interact with the host, maintaining backward compatible with legacy I/O technologies. From an application viewpoint, the OS and device driver software on the host abstract the ESMD enumeration, so as to ensure technology compatibility with the COSM and the memory access capabilities.

3) Memory Management at the Host

The shared memory address management function ensures that the host is aware of the shared memory's physical or virtual address spaces and can map them into its own memory address space. The ESMD communicates the physical or virtual addresses of its shared memory regions to the host during initialization. The host maps these memory regions into its own address space using Direct Memory Access (DMA) mapping for efficient data transfers, and Address Translation Services (ATS) for virtual-to-physical address resolution. On the ESMD devices, the access permissions are configured in accordance with the COSM permission matrix before mapping of the physical addresses to the shared memory region of the host. Memory address management is handled by host drivers, which interact with the ESMD through standardized kernel-space drivers or user-space library APIs.

Memory regions are dynamically allocated and managed using memory management units (which maintain page tables). For example, a host can map a 2 GB shared memory region of the ESMD into its (the host's) virtual address space using a memory mapping call provided by the device driver, where this 2 GB shared memory region is part of a shared memory region at the ESMD device that is managed by its own address and physical memory units. Once the ESMD memory region is mapped by the OS, applications can use traditional write and read instructions

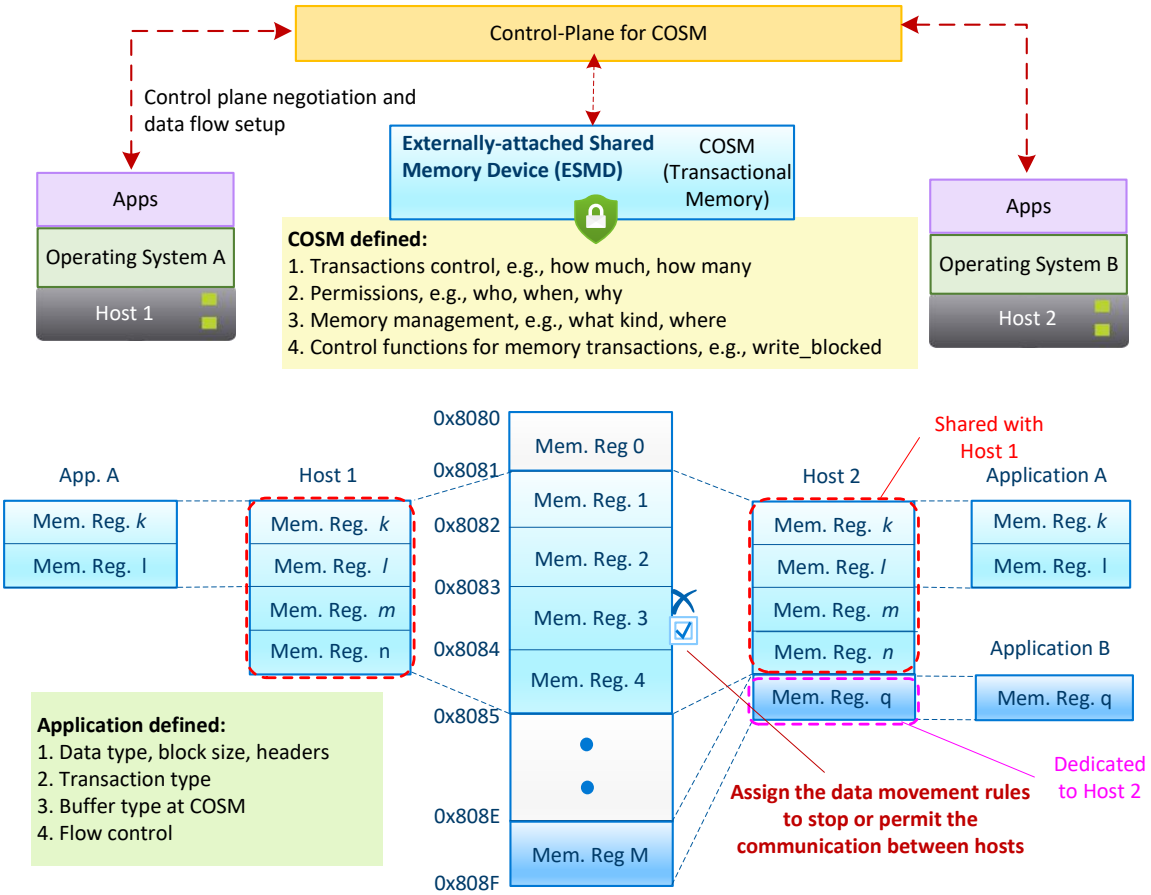


FIGURE 9: Data transfers over ESMD can be conducted with User Defined Protocol Data Units (UPDUs), whereby applications determine the data structure to be used for sharing over common (shared) memory regions. This allows the applications to create User-Defined Protocol Data Units (UPDUs) over the shared memory, whereby applications define the data block specifications, such as data type, block size, and headers.

with addresses pointing to specific regions of the shared memory, while the COSM memory management libraries provide APIs for applications to request memory access, such as `cosm_malloc(...)` (custom allocators for shared regions), which are similar to `malloc(...)` for memory allocation and `free(...)` for memory de-allocation.

4) Memory Management at ESMD

The ESMD centrally manages the shared memory regions and enforces COSM-defined policies. The ESMD allocates, deallocates, and tracks the usage of the shared memory regions for all participating hosts in $\mathscr{H}$. The ESMD enforces the access permissions using the permission matrix $\mathbf{P}_{m,\mathscr{H}}(t)$, thus blocking unauthorized requests in real-time. The ESMD also handles the fragmentation and compaction to ensure efficient use of memory resources. The ESMD employs a memory controller to manage the physical memory and an access control unit to enforce the COSM policies. An internal policy engine dynamically updates the permissions and QoS rules based on host requests. ESMD enforce the access policies locally, ensuring that applications or processes within the host comply with the COSM rules. For example, the ESMD allocates a new 1 GB region for host A, restricts it to read-only access for host B, and blocks host C entirely.

5) Setting up of Shared Memory Access at the Hosts

The ESMD supports simultaneous access by multiple hosts, ensuring data consistency and security. Each host is assigned a memory region or offset within the shared memory, isolating its access from other hosts. Hosts use locking mechanisms or atomic operations to ensure consistency when accessing shared regions. The ESMD enforces access permissions and resolves conflicts dynamically. QoS enforcement prioritization can allow certain hosts or applications to yield the memory access requests to higher priority hosts during contention. Multiple memory regions can be used for directional transfer of data when the control policies are in place to establish synchronizations in a restrictive environment of memory access. For example, when host A writes to a shared memory region while host B reads the same data. The host synchronization techniques ensures that host B sees a consistent state of the data using locks or atomic operations. Use

of host-specific hardware synchronization mechanisms like semaphores, spinlocks, or atomic operations to coordinate concurrent access can achieve better results when multiple hosts share same memory regions for data access. The information about synchronization, e.g., semophore values can be shared over a dedicated shared memory which is negotiated by all the participating hosts within a common memory region which can be coordinated by the control and data plane functions of the ESMD management.

6) Synchronization principles in COSM

Synchronization is critical to ensure data consistency and prevent race conditions during bidirectional traffic. The ESMD and hosts use hardware or software synchronization mechanisms, such as mutexes or semaphores to coordinate access to shared regions, ring buffers to manage data flow in producer-consumer models, and interrupts or polling to notify hosts when data is ready for consumption. Flow control mechanisms throttle traffic based on memory bandwidth or QoS requirements, however in a shared memory context where multiple hosts and OS are involved, a dedicated communication channel for flow-control information sharing is necessary for the hosts that are accessing a common memory region at the ESMD. The ESMD provides a hardware-backed synchronization interface (e.g., hardware semaphores), which can block the write until a flag or semaphore is cleared by the writing host. The hosts can expose synchronization primitives to applications through drivers and library functions specific to ESMD and COSM. For example, host A writes data into a shared memory region and raises an interrupt to notify host B. Host B acknowledges receipt and processes the data, ensuring bidirectional traffic flow.

7) Control Permissions

TABLE 1: Illustration of Binary Control Permissions for different Hosts

Host	ID	Read (R)	Write (W)
Host	A	1	1
Host	B	1	0
Host	C	0	0
Host	D	1	1

In a Controlled Shared Memory (COSM) ESMD system, a simple mechanism to manage concurrent access to shared memory regions, particularly for critical sections of data accessed by multiple hosts, can be defined with binary permissions by regulating read (R) and write (W) operations. When represented in a matrix form, a binary permission matrix can be enumerated, where 1 indicates enable and 0 indicates disable for each host on read (R) and write (W) provides a straightforward mechanism to enforce these access controls. The binary permission matrix with values as described in Table 1 defines the access rights of each host to a specific memory region, with 1 (Enable): The operation (read or write) is allowed, and 0 (Disable): The operation is denied. Interpreting the Table 1, host A and host D can both read and write to the critical section, while host B can only read the data, and host C is entirely restricted from accessing the memory region.

Permissions to individual hosts can have more granular control, where permissions can be defined at the level of hosts, memory regions, or even sub-regions, i.e., specific offsets within a memory block, and can have dynamic adjustments, where permissions can be updated in over time based on runtime policies, ensuring flexibility in managing access. Figure 10 shows the role of permission matrix in protecting the critical section of data. For protecting the critical section, the memory regions that contains shared data which is being shared with multiple hosts can be enforced with strict permissions (e.g., read-only for most hosts) to prevent accidental or unauthorized modifications. The write access to critical data can be limited to specific hosts or synchronized with locking mechanisms to avoid data corruption.

TABLE 2: An example scenario where host A can update the configurations present in the shared memory region, while other hosts B, C, and D can only perform read operations to maintain the latest updated configuration by host A

Host	ID	Read (R)	Write (W)
Host	A	1	1
Host	B	1	0
Host	C	1	0
Host	D	1	0

When multiple hosts need access to a shared memory region containing critical data, control permissions help prevent conflicts. An example scenario can be where a critical section contains configuration data for a distributed application, host A is responsible for updating this configuration, while hosts B, C, and D need read-only access to ensure they operate with the latest configuration. To safeguard the critical section, the permission matrix can be set to configuration as shown in the Table 2 which indicates to COSM that, host A is the only entity allowed to write updates to the configuration, while hosts B, C, and D can only read the configuration and cannot modify it.

A time varying permissions can be useful in dynamic environment such as industrial facilities, where there needs to be maintenance mode such that only the supervisory control needs access, and all other hosts except supervisory host may be denied access, i.e., with control values $R = 0$, $W = 0$. In another scenario where there is a host failure (e.g., host A), the data needs to be immediately enabled for a secondary host (e.g., host B), this can be achieved by enabling the permissions to read the critical section in the shared memory to other host. The access enabling can be done in real-time allowing other host to gain access to the critical section immediately without losing the data itself and for the fail over mechanisms to take over. Synchronization and locks over critical section can also be achieved through permission

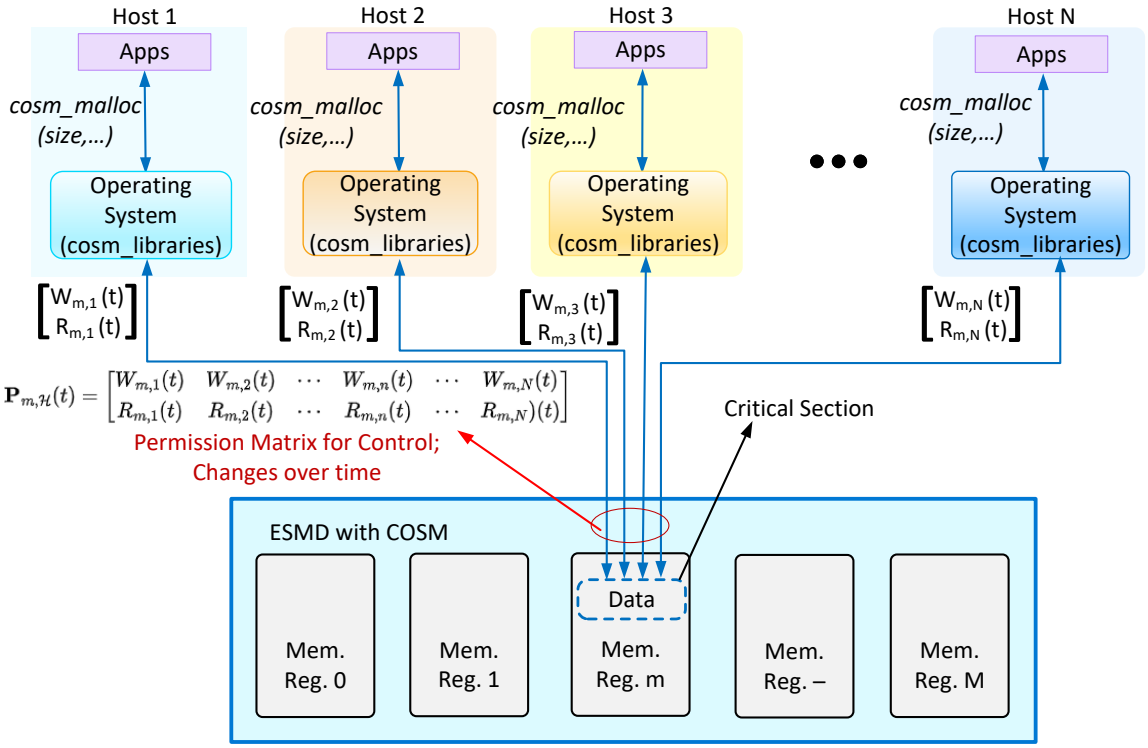


FIGURE 10: A shared memory region can be the critical section, which can be accessed by multiple hosts. The control plane moderates the access control by setting the permissions through a permission matrix $\mathbf{P}_{m,\mathcal{H}}(t)$. This permission matrix $\mathbf{P}_{m,\mathcal{H}}(t)$ represents the overall access control for a given memory region m across a shared set of hosts. Once the control plane sets up the shared memory regions to hosts, then applications can use the library functions from the OS memory management services to allocate and access the memory addresses.

matrix configuration. For example, if host A acquires a lock to update the data in the shared memory region, all other hosts could be temporarily blocked from both reading and writing into the critical section of the shared memory region. The COSM policy engine in the management plane can dynamically adjust the permissions (e.g., Write=0) to prevent unintended modifications during the critical update.

8) Role of ESMD and COSM management

The ESMD and COSM system handles the memory region allocation in coordination with the host OS. The process involves the ESMD-COSM Management to coordinate the memory region allocation and access policies by the applications and hosts. The ESMD enforces access control through its permission matrix and assigns memory regions to specific hosts based on requested memory size by the applications, and policies, such as, QoS parameters, priority levels, or security requirements. The ESMD management plane binds specific physical or virtual memory addresses of the ESMD device to each allocated region and communicates these bindings to the host. The host OS integrates with the ESMD through dedicated drivers. During memory allocation, the OS queries the ESMD for available regions and applies the allocation to its memory map. As illustrated in Fig. 11, applications interact with COSM memory via kernel APIs and library functions, such as `cosm_malloc(size, policy)`, where, size specifies the required memory size, and policy specifies attributes such as read/write permissions, priority, or data consistency preferences. The API abstracts the underlying complexities, making COSM memory allocation similar to standard memory allocation (e.g., malloc).

G. COSM APPLICATIONS AND USE CASES

1) COSM in a Datacenter Infrastructure

Traditional IP networking is currently being implemented as the main communication standard in cloud and data center infrastructures [70]. Memory devices could be considered as a better alternative for traditional IP networking, as memory devices can provide isolated environments for data processing. Compared to interconnecting solutions based on I/O devices and network-based interfaces, memory-based solutions use a common shared memory to facilitate data exchanges and communication between independent domains of computing environments. The shared memory component is independent of the computing environment domains, providing security and isolation as a service to the data being exchanged over a common shared memory.

A Top-of-the-Rack-Memory (TORM) solution [71], [72] acts as a broker that physically connects the racked systems present in both trusted and untrusted domains via optical,

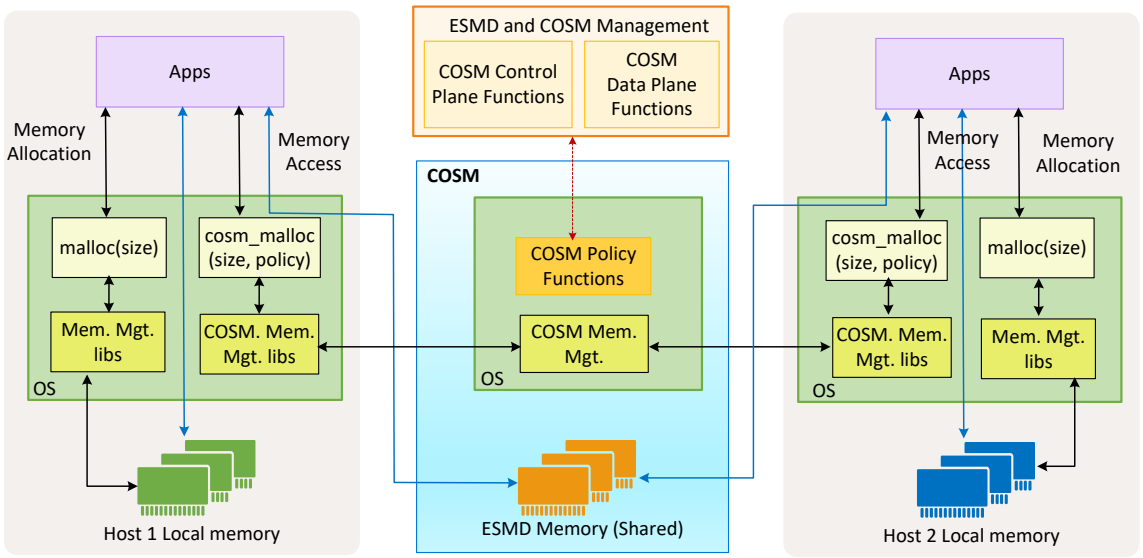


FIGURE 11: ESMD and COSM management coordinates the memory region allocation to the host operating system. Applications use the memory management services from the operating system through kernel APIs and library functions to allocate the memory regions, e.g., `cosm_malloc(size, policy)`. The allocated COSM memory is access through normal write-and-read instructions by the host system which are then directed to respective I/O controllers based on the addresses bound to the COSM memory.

PCIe (CXL), cable, UPI, or Ethernet [73]. Figure 14 illustrates a TORM solution that can replace a TOR switch [74]. TORM can provide connectivity services to datacenter hosts similar to a TOR switch, however with better levels of isolation and reliability. Figure 15 illustrates the presence of a COSM network consisting of interconnected ESMD devices with COSM; the COSM network can provide networking services to hosts in large datacenters. Moreover, the COSM network could be used to run applications in critical infrastructures, such as smart grid and healthcare, within secure isolated environments.

2) COSM for 5G Applications

Memory channels can help to achieve low latency and high bandwidth data transfers, making COSM suitable for next-generation networks, such as 5G and 6G networks, whose applications require low latency and high bandwidth along with better levels of isolation (physical isolation) [75]–[78]. In addition, COSM does not involve traditional network connectivity devices, such as switches, routers, and gateways, which could drop packets. Moreover, COSM offers reliable data transfers with high-performance communication techniques using simple protocols for retransmissions and data tracking.

One such example is network slicing in 5G, a concept of creating multiple virtual slices [79]–[81]. Each slice needs to be isolated from the other. Network slicing only provides software isolation (logical isolation) or hardware-assisted isolation but not the isolation level of COSM isolation. COSM has the ability to provide physical isolation via air-gap between slices. Therefore, COSM could be used in network slicing use cases that require low latency, high bandwidth, and better levels of isolation to improve performance and security.

3) COSM for Disaggregated and Distributed Functions or Resources

Disaggregating and distributing the available resources or functions increases flexibility. However, the probability of being attacked is expected to increase, as these resources or functions could now be present in different domains but still communicate or exchange information between each other. If the disaggregated and distributed resources or functions are shared between critical and non-critical infrastructures, then there could be a possibility of information leak or theft from the critical infrastructures. COSM can provide physical isolation via air-gap between the critical and non-critical infrastructure domains, thereby protecting data of the critical infrastructures.

4) Real-Time Industrial Control Systems

Industrial automation systems, such as those used in manufacturing plants, often involve a network of devices like sensors, Programmable Logic Controllers (PLCs), and actuators that exchange data in real-time and perform decisions real-time. An ESMD with COSM capabilities can act as the shared memory hub in such systems, providing a central point for data storage and exchange. For instance, sensors monitoring temperature, pressure, or an industrial machine real-time performance can write their data to designated memory regions, while controllers read this data to adjust actuator settings. COSM ensures that each device accesses

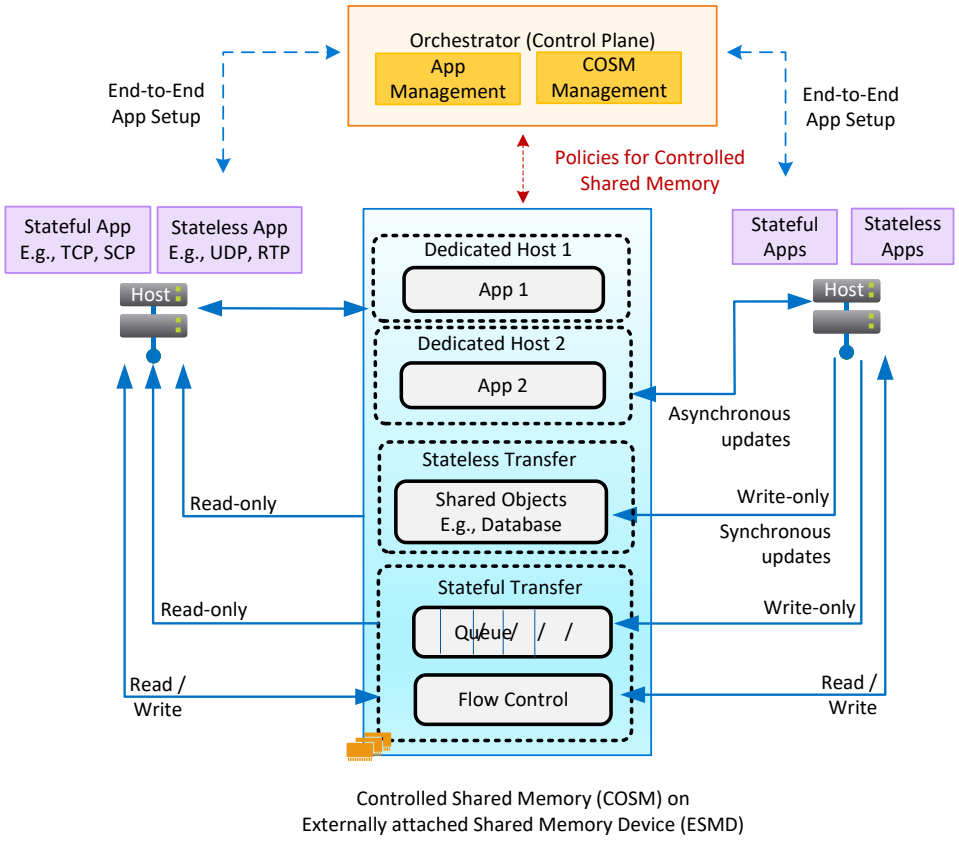


FIGURE 12: Applications may require both stateful and stateless data transfers, such as, TCP and UDP when using the memory regions on the ESMD device. Different types of memory access include dedicated memory to single hosts (i.e., no sharing), stateless transfers such as the shared data-structures, objects, and databases, as well as the stateful transfers such as data queues, circular buffers, associated with flow control mechanisms. Control plane would coordinate the setting up of memory regions required dedicated, stateless, and stateful transfers.

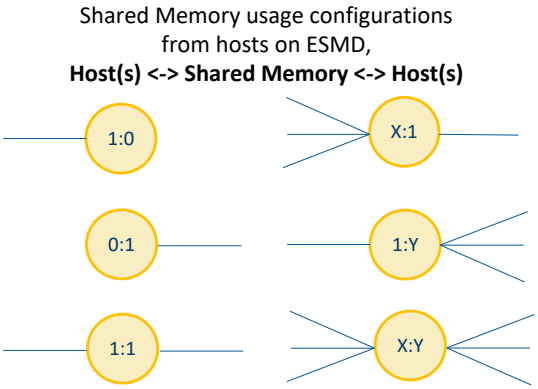


FIGURE 13: Different configurations in which a shared memory can be shared among the hosts. $1:0$ and $0:1$ represents that the COSM memory is just dedicated to just 1 host, $1:1$ represents that COSM memory is shared between two independent host and each host can share data among them, $X:1$ represents that multiple hosts can and $1:Y$

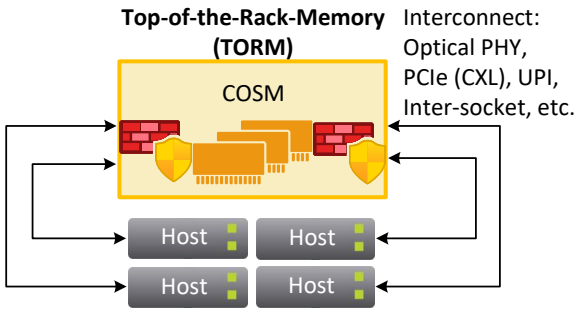


FIGURE 14: Top-of-the-Rack-Memory (TORM) is a deployment topology in which data is transferred among multiple hosts within a rack in a datacenter using ESMD devices. The TORM can replace the existing Top-of-the-Rack network switches; the TORM provides similar connectivity services to the hosts in a datacenter as ToR switches.

only its allocated memory regions with assigned permissions, preventing accidental interference or malicious tampering, which is crucial in safety-critical environments.

Synchronization provided by the ESMD plays a critical

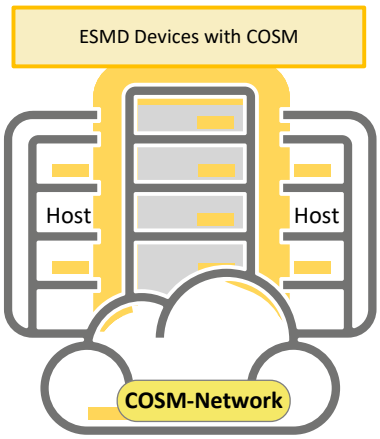


FIGURE 15: ESMD devices with COSM can be interconnected to form a COSM-network providing the networking services needed for the hosts in large datacenters. The shared memory based networking-services can enable isolation between compute domains, such as VMs and containers, at finer (more detailed) granularity levels than traditional IP networks; thus creating more secure environments to run applications in critical infrastructures, such as smart grid and healthcare

role in ensuring reliable operations across the industrial control system. For example, in a robotic assembly line, real-time coordination between conveyor belts, robotic arms, and inspection units is critical. The ESMD facilitates this coordination by allowing devices to share status updates and commands through synchronized memory access. By using hardware-based COSM, the system achieves the low-latency and deterministic performance needed for real-time control, while its flexibility allows dynamic reconfiguration of memory regions as production requirements evolve. This makes ESMD with COSM an essential component in future smart manufacturing deployments.

5) Autonomous Vehicle Sensor Fusion

Autonomous vehicles rely on multiple high-resolution sensors, such as LIDAR, radar, and cameras, to perceive their environment [82]–[84]. These sensors generate large amounts of data that must be processed in real-time to make vehicle navigation decisions. An ESMD with COSM capabilities can serve as the central shared memory hub, in the context of software-defined-vehicle operations [85] for storing and sharing sensor data among various onboard processors, including CPUs, GPUs, and AI accelerators. For example, LIDAR data can be written to a shared memory region by the LIDAR processing unit, while the path planning module simultaneously reads this data for decision-making. The COSM policy engine ensures that each module only accesses the memory regions relevant to its operation, preventing unauthorized access and maintaining operational safety.

In addition to access control, the ESMD's ability to synchronize data sharing is critical for real-time processing. As different sensors operate at varying frequencies, the COSM framework ensures that all modules receive a consistent and synchronized view of the shared data, preventing data races or inconsistencies. For instance, when fusing data from multiple sensors, the synchronization mechanism ensures that the timestamps of data from LIDAR, radar, and cameras align, enabling accurate environmental modeling. By leveraging the ESMD with COSM, autonomous vehicles can efficiently manage massive data flows, meet stringent real-time processing requirements, and enhance safety and reliability in dynamic driving conditions.

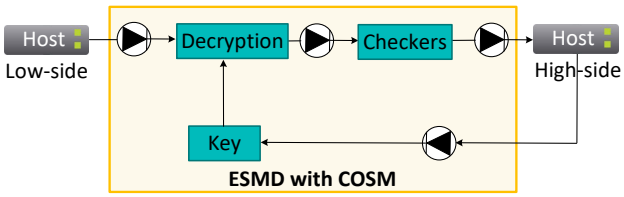


FIGURE 16: An example implementation of Cross-Domain Solution (CDS) for data transfers between high-side and low-side hosts, seperated by isolation mechanism provided by the ESMD. Where, the data-diodes are utilized to create directional data transfer, as well as to enable the ESMD device to perform in-memory compute for decryption and checker operations. The decryption key can be shared from the high-side host utilizing the directional data-diode into the ESMD which can be used to perform in-memory compute actions inside the ESMD.

6) ESMD with COSM as Cross-Domain Solutions (CDS)

A cross-domain solution (CDS) is an integrated information assurance system composed of specialized software or hardware dedicated for isolation framework. It provides a controlled interface to manually or automatically enable and/or restrict the access or transfer of information between two or more security domains based on a predetermined security policy. CDS is designed to enforce domain separation and typically include some form of content filtering, which is used to identify and block information that is unauthorized for transfer between security domains or levels of classification.

Checkers are a fundamental part of the CDS that restrict the transfer of files/data based on the implemented checker. The specific checker used can vary depending on the use case. Some checkers may verify the syntax of a file, the content of a file, or an ongoing stream. Currently, checks are only performed when data is transferred from the Low-side to the High-side. Such a CDS system shown in Fig. 16, where the system ensures the security of transactions by utilizing Address Translation Table, to enable diodes between various stages of the process when transferring data between the Low-side and High-side, as illustrated in Fig. 16. The two main stages of the process are Decryption and Checkers. The

decryption stage involves decrypting the data received by the Low-side. The checkers are a crucial part of the design, performing checks on the received and decrypted data.

7) Shared Memory in Real-Time Medical Systems

Modern healthcare relies heavily on advanced medical imaging systems like MRI, CT scanners, and ultrasound machines for diagnosis and treatment planning. These systems involve multiple compute systems, including image acquisition hardware, accelerated processing units (CPUs, GPUs, and FPGAs), and display systems that work together to deliver accurate and timely imaging results. An ESMD with COSM capabilities can serve as a central hub for data sharing among these compute components, enabling efficient, real-time image processing and analysis.

For instance, during an MRI scan, the image acquisition hardware generates large volumes of raw data that must be processed and reconstructed into interpretable images. The raw data can be written into specific regions of the shared memory managed by the ESMD. Image reconstruction algorithms running on GPUs or AI accelerators can then read this data from the shared memory, process it, and write the reconstructed images back to a separate region. The display system, in turn, retrieves the reconstructed images for real-time visualization by clinicians. COSM ensures strict access control, allowing only authorized components to read or write to specific memory regions. For example, the display system may have read-only access to the reconstructed image data to prevent accidental overwriting. The ESMD's ability to dynamically allocate memory regions is particularly valuable in healthcare, where imaging workflows can vary significantly between patients and procedures. By enabling secure, low-latency data sharing and efficient memory management, ESMD with COSM capabilities enhances the performance, reliability, and scalability of real-time medical imaging systems, directly improving patient care and diagnostic accuracy.

IV. PROTOTYPE EVALUATION OF ESMD WITH COSM

To verify COSM design, we have used a hardware prototype, where two Intel® Xeon® servers are bridged using Altera® Agilex® FPGA via an open standard high-bandwidth Compute Express Link (CXL) connections as shown in Figure 17. The goal is to enable an Externally-attached Shared Memory Device (ESMD) with Controlled Shared Memory (COSM) including address and data filtering mechanisms, organized similarly to virtual memory page tables, and permission checks that will provide data-diode like isolation. The ESMD provides CXL.mem interface to connect DDR memory and the access translation tables are implemented using CXL.io on the FPGA which straddles the memory traffic stream between Xeon sockets.

A. FPGA BASED HARDWARE ARCHITECTURE

The COSM prototype shown in Figure 17 utilizes an address-filtering mechanism (called Address Translation Table (ATT)) on outgoing AXI interface of CXL IP. This approach allows to control all transactions utilizes various links to facilitate lower latency and higher bandwidth between the system, peripherals, and CPU sockets. The prototype implements memory credential tables and enforces the correct protocol of memory traffic streams between hosts. The placement of the FPGA in such a topology, enables isolation of both socket-local DRAM and cores from other parts of the system. Hard Processing System (HPS) which is part of the FPGA could be used to offload computing, data conversion. The HPS has dedicated 1Gbit RJ-45 Ethernet port, and this connection could be used for additional air-gapped system or for management purposes, encryption key exchange and time protocol handling. An ATT is implemented in the FPGA on a per CXL type 3 device link basis. When programmed correctly, the table will firewall certain memory regions to

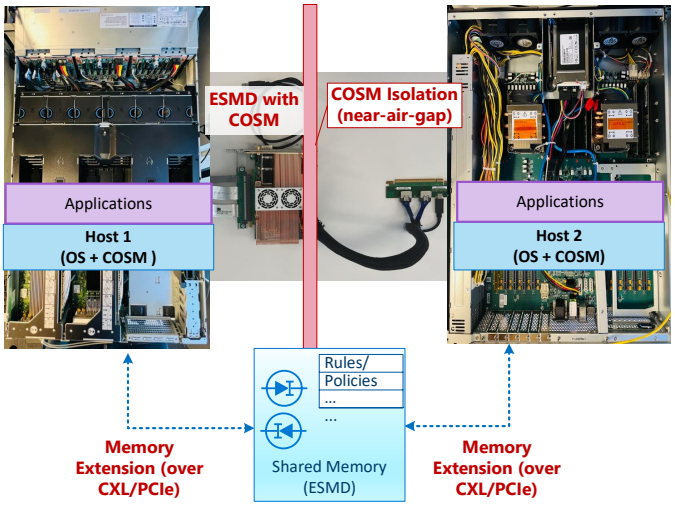


FIGURE 17: Photograph of instantiation of COSM testbed based on Compute Express Link (CXL).The testbed instantiation interconnect host 1 on the left-hand side via an FPGA-based ESMD with host 2 on the right-hand side.

TABLE 3: Configuration of FPGA used for the prototyping of ESMD with COSM

FPGA Configurations
Intel® Agilex™ 7 FPGA SoC I-Series `AGIB027R29A1E2V` device R29A/D package compatibility Quad Core ARM® Cortex® A53 (HPS)
Memory
x2 DDR4 SO-DIMM sockets: each capable up to x72 bit bus, 32 GBytes density, 2666 MT/s data rate, with ECC (FPGA only) On-board DDR4 bank: capable up to x64 bit bus 2666 MT/s data rate (mutual FPGA, HPS), 8 GBytes density
Board Interface
PCIe Gen5 x16 edge connector x2 HSI connectors: each capable up to x8 transceivers x16 total for PCIe Gen5 End Point compatible Gen5 End Point, Root Port, CXL End Point compatible

TABLE 4: Host Configurations for both H_1 and H_2

Configuration Type	Value
CPU	Intel(R) Xeon(R) Platform 8481C
Cores	112 per socket
Sockets	2
Frequency	800,000 MHz
Cache size	107520 KB
clflush size	64 B
cache_alignment	64 B
Address sizes	52 bits physical, 57 bits virtual
Memory	512 GB

only be accessible by the link. Permission rules can be programmed to allow access for some address ranges, while preventing access to other address ranges. Specifically, the following rules are allowed: reject-all, reject-read, reject-write, allow-all, allow-read, allow-write. The permission table allows for up to 32 rules. Each rule is checked sequentially, giving priority to rules checked earlier.

The hardware prototype allows sharing content of the memory with peripherals connected to AFU, while keeping security standards achieved for CPU sockets. AFU allows the implementation custom logic, various types memory checkers or compute accelerators. In addition AFU allows interrupt triggering to both CXL hosts based on ATT events or any other event triggered by customer logic. Memory requests from a socket, into the FPGA, are first filtered by that socket's associated ATT. The FPGA uses two memory channels to signal to DDR. The split memory channel requires a 10-port arbitration switch to enable each input to access either memory channel as shown in Figure 18. Four of the switch ports are used for socket to memory traffic, while two ports are used for accessing the address translation tables for programming or reprogramming or both. The final four ports are currently unused and reserved for future FPGA expansion.

Each x86 socket is connected to the COSM platform using Compute Express Link [33] (CXL) to facilitate fast communication between the CPU and platform. Memory requests from a socket, into the COSM , are first filtered by that socket's associated address translation table. CXL is open standard and the platform has two CXL interfaces to allow connecting the systems. Data are transferred over CXL.mem and control data are transferred via CXL.io interface. The Accelerator Functional Unit (AFU) is responsible for ensuring that memory requests, from either socket, arrive at the correct memory controller. The AFU consists of 10 input and 10 output ports. Each input port consists of multiple virtual channels. In addition to the input and output ports, the AFU uses a route compute block, a route reservation unit, a flow control unit, a virtual channel arbitration unit and an internal crossbar to perform the actual switching operation. The Address Translation Table (ATT) is primarily used as a filter for ingress memory accesses. When programmed correctly, the table will block certain memory regions to only be accessible by the socket. ATT rules can be programmed to allow access for some address ranges, while preventing

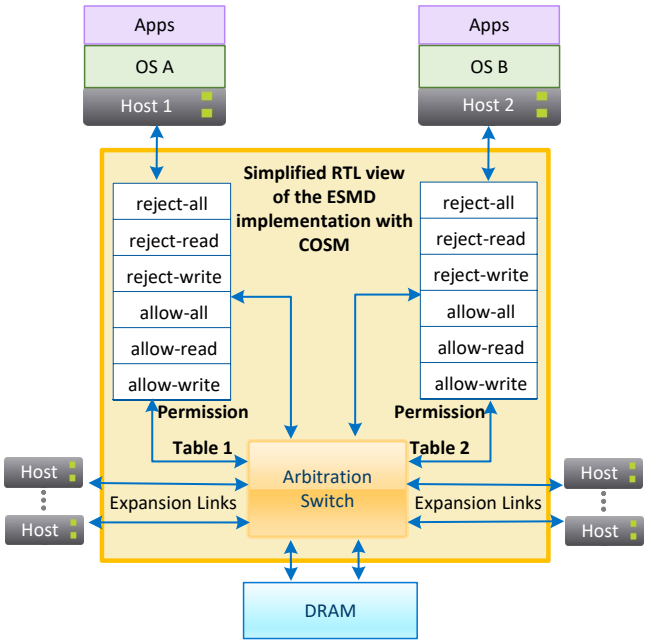


FIGURE 18: High-level RTL block diagram

access to other address ranges. The ATT allows for up to 32 rules. Each rule is checked sequentially, giving priority to rules checked earlier. A default rule at the bottom of the table ensures that at least one rule is hit. In addition to firewall, the ATT can also provide translation from one memory space to another space. In order to limit the hardware complexities in the initial implementation, several restrictions are imposed on the ATT rules:

1) If the ATT is used only as a firewall, i.e., for performing permission checks and no address translation, then the start and end addresses can be any arbitrary addresses.
2) If the ATT is used to perform address translation then the following rules must be followed:
   a) The size of each address slice being mapped must be ordered of 2.
   b) The range of the address being mapped can perhaps any arbitrary integer multiple of the size of each slice.
   c) The start address of the address range being translated must be aligned to the power of 2 greater than the size of the range.
   d) The start address at the destination location where the slice is being mapped in can be at any address, but software must ensure there is enough space for the entire slice at the destination location.

An access check block, within the ATT. Performs a comparison against the Compare Low and Compare High range to check if the destination address of a memory request is the range of a rule (inclusive). When destination address is in the range of a rule, then the rules control bits are checked for permissions. The Control register used to determine if the destination address of the packet hits on an ATT rule are:

- **Enabled:** This 1b field controls whether the permission table rule is enabled. When set to 1 the rule is enabled, when not set the rule is considered to be a miss.
- **Reverse:** This 1b field controls if the result of the comparison of the destination address against the compare low and high range needs to be reversed. When set the result of the comparison should be inverted.

The remaining control bits are used to determine the address permissions. These fields are to be checked after the ATT rule hit is determined. On a hit, the associated access control bits are checked to determine whether the packet should be allowed to continue or rejected. The access control bits are:

- **Reject:** When set, this 1*b* field rejects all transactions, irrespective of the operation.
- **Read (Rd):** When set, this 1*b* field allows memory read transactions. When this bit not set (= 0), all memory read transactions are blocked.
- **Write (Wr):** When set, this 1b field allows memory write transactions. When this bit is not set (= 0), all write transactions are blocked.

B. SOFTWARE ARCHITECTURE

The COSM library provides a user-space API for interacting with the PCIe COSM driver. It simplifies the process of memory mapping and device configuration by abstracting the underlying system calls into easy-to-use functions. Additionally, the library offers simple memory structures for exchanging information between two systems.

The PCIe COSM Linux kernel module enables user-space libraries and applications to interact with the PCIe/CXL device using files exported in devfs, such as `/dev/pcp0`. The driver exposes memory and device configuration through standard system calls like `open()`, `mmap()`, `close()`, `read()`, `write()`, and `ioctl()`. This driver removes differences between various COSM versions and unifies the API for components built on top of it.

Example below shows an example usage of getting information about connected PCP platform:

```
int fd = open("/dev/pcp0", O_RDWR);
if (fd < 0) {
    perror("open");
    exit(1);
}

struct pcp_info tmp;
if (ioctl(fd, PCP_INFO, &tmp) == -1) {
    perror("ioctl");
    close(fd);
    exit(1);
}
close(fd);
```

C. TESTBED OVERVIEW

Testbed consisted of two server platforms, host 1 (H_1) and host 2 (H_2) with two sockets each. Every socket was equipped with 4th Gen Intel® Xeon® Scalable Processor. H_1 uses 56−ore processors, while host 2 used 48-core processors.

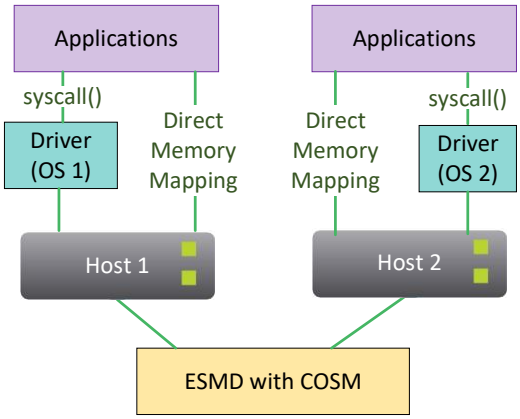


FIGURE 19: COSM in an ESMD can be accessed by application in to ways, *i*) OS managed, i.e., through a driver with `syscall(...)`, *ii*) direct mapping of address to application, for e.g., directly fetching the hardware address mapping by the ESMD from the BIOS for a specific memory and using the direct addresses of the COSM in an ESMD device.

This totals to 2 (sockets) * 56 (cores) * 2 (threads per core) = 224 total threads on host 1 and 2 (sockets) * 48 (cores) * 2 (threads per core) = 192 total threads on host 2.

D. VALIDATION AND EVALUATION

1) Metrics

a: Write and Read Rates

Generally, with the ESMD-based COSM approach, the effective transfer rate [bit/s] between applications on a pair of communicating hosts host 1 and host 2 is governed by the write and read rates of the ESMD hardware. Moreover, the memory management of the operating systems on hosts 1 and 2, as well as the synchronization of the applications (e.g., Application A and Application B) on the hosts affect the effective transfer rate, as illustrated in Fig. 20.

We denote W_1 for the write rate [bit/s] of host 1 into the ESMD; analogously, W_2 denotes the write rate [bit/s] of host 2 into the ESMD. Also, we denote R_1 for the read rate [bit/s] of host 1 from the ESMD; and W_2 for the read rate [bit/s] of host 2 from the ESMD. We denote T_{12} for the effective transfer rate from Application A on host 1 to Application B on host 2. The effective transfer rate T_{12} is governed by the minimum of the respective write and read rates of hosts 1 and 2 to (and from) the ESMD as well as potential reductions due to the internal data transfer in the ESMD, the memory management, and the application synchronization. We express these potential reductions through general functions f, i.e.,

$$T_{12} = \min(W_1, R_2) - f(\text{ESMD Int. data transfer char}_{12}) - f(\text{Mem. Mgt}_{12}) - f(\text{Synch}_{12}). \quad (4)$$

We measured the host write and read rates for different ratios of read-to-write operations so as to reflect the varied

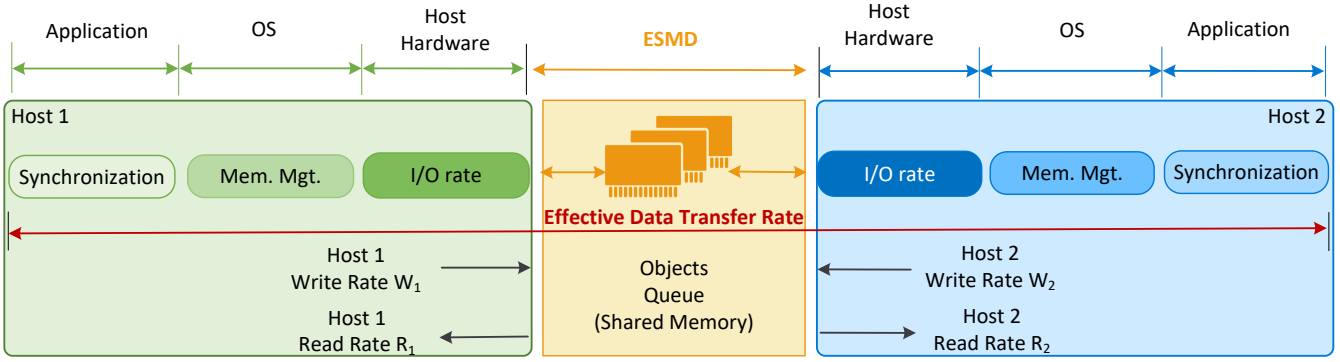


FIGURE 20: Effective data transfer rate between applications that belong to two different hosts interconnected through a shared memory region of an ESMD device is a function of four components: (a) I/O rate of an interconnect (physical link) and the interface (protocol) such as CXL, PCIe, UAL, (b) memory management by the OS for address management, (c) synchronization for stateful data transfers to access a critical section using flow control mechanisms, such as polling, interrupts, queue management, and buffer pointer updates, and (d) device characteristics of ESMD device such as the internal memory management and policy enforcement on the dataplane processing.

data interaction characteristics of a wide range of applications. Specifically, we consider scenarios with only reads, with three reads for one write, with two reads for one write, and with one read for one write. For these mixed scenarios of writing and reading, we report the overall rate [bit/s] at which the writing and reading occurred for the considered write-and-read ratio.

Our goal is to conduct a general evaluation of the COSM paradigm. Hence, we leave the effects of the internal ESMD data transfer, the memory management, and the application synchronization out of the scope of our evaluation. We measure and report the host write and read rates into (and from) the considered ESDM

b: Read Latency

Analogously to the measurement of the write and read rates, we evaluate the read latencies. Specifically, we measure the read latencies with the Intel® Memory Latency Checker tool (version v3.11b) [86].

Measuring write latencies on modern CPUs is complicated because modern CPU architectures employ multiple methods (e.g., caching, write combining, and write reordering) to hide write latencies. Due to these CPU mechanisms it is also complicated to ensure that writes have been completed and have actually reached the ESDM. For these reasons, we focus on read latency measurements.

2) Procedure

Our testing procedure utilizes Intel® Memory Latency Checker (mlc) tool (version v3.11b) [86] modified to support the ESMD memory allocations. We have added the parameter `-A<start address>:<size>` to inform mlc about the location of the ESMD in the memory address space of the host and about the available ESMD size (both parameters are expressed in hexadecimal). On our systems, the ESMD was located at address 8080000000_{16} and had 32 GB (800000000_{16}B) in size.

Bandwidth and latency measurements were conducted on H_1. In order to test bandwidth and latency under load, we started our tests by running mlc on H_2 to generate traffic on the ESMD and then started mlc on H_1 to measure bandwidth or latency. For bandwidth measurements, we used the same memory regions on both hosts to check concurrent access to the same memory locations and we used the entire 32 GB memory space, i.e., we set the parameter `-A8080000000:800000000` on both H_1 and H_2. We were able to do so because mlc does not verify the values read from or written to memory.

For latency measurements, we had to separate H_1 from H_2, because latency is measured by conducting dependent reads (see Section IV-D2b), which require that the chain of pointers prepared by H_1 is not be modified by H_2. Therefore, H_1 used the first 16 GB of memory (run with parameter `-A8080000000:400000000`) and H_2 used the second 16 GB of memory (run with parameter `-A8480000000:400000000`.

We used mlc's loaded latencies mode on H_2 to generate traffic with the following parameters `--loaded_latency -gno_delay.txt -b30M -Z -T -t900`:

- `-gno_delay.txt` – this option selects delays to test between memory accesses in load generation threads. Since we are interested in generating 100% load we have put single 0 inside `no_delay.txt` file,
- `-b30M` – this option selects buffer sizes used by workloads. Since workloads use up to three buffers (for Stream-Triad) and we are running on up to 224 threads we had to limit the buffer sizes to 30 MB per buffer as in the extreme case this uses $3 \times 224 \times 30 = 20160$ MB,
- `-Z` – this option selects to use AVX512 instructions for traffic generation. Since CXL transactions are all 64 B = 512 b in size, by using AVX512 operations one

instruction in code equals one transaction on CXL,
- `-T` – this option disables the latency measurement thread of the loaded latency mode. Since we want to only generate load by H_2 and we do not want to introduce any competing traffic, we disabled the latency measurement thread on H_2,
- `-t900` – this option specifies to run the load for 900 seconds. This ensures that H_2 will be generating load for the entire duration of the measurements on H_1.

We used four traffic types on H_2:
- 0% Reads and Writes on H_2 – no mlc running on H_2,
- 100% Reads on H_2 – default load in loaded latencies mode,
- 100% Writes on H_2 – additional mlc option `-W6` uses 100% non-temporal writes load. Non-temporal writes ensure that the CPU does not issue reads to the cache before writes,
- 50% Reads and 50% Writes on H_2 – additional mlc option `-W8` uses 1 read and 1 non-temporal write load.

Additionally, to evaluate the impact of access control rules on performance we conduct evaluations with different write and read permissions defined for the entire ESMD memory space in ATT rules for H_2. For loads on H_2 where access control was disabled for such traffic (e.g., 100% Read on H_2 with ATT set with read disabled) we still run mlc with the desired traffic type. The mlc tool does not verify values read or written in load generation threads, thus it simply ignores unsuccessful reads and writes and continues with load generation. Normally this results in mlc reporting significantly higher bandwidth (effectively incorrect bandwidth), but since we ignore measurements from H_2 this does not impact our measurements. ATT rules for H_1 accesses are set to enable both reads and writes into the entire ESMD memory space.

a: Write and Read Rates

For bandwidth measurements, we use mlc's peak injection bandwidth mode with parameters:

`--peak_injection_bandwidth -b30M -Z -t60`.

The meaning of these parameters are the same as for the parameters used on H_2 as described above. `-t60` specifies to run and measure each workload for 60 seconds. Bandwidth is calculated by measuring how much data was read/written to ESMD, divided by the amount of time each bandwidth generation thread was running. Total bandwidth is computed as a sum of bandwidth from all threads. Since the machines used for evaluation are clean-installed and do not run any additional services or loads, running each workload for 60 seconds and computing average bandwidth as specified above provided stable results, as we have verified with pilot tests involving multiple runs.

We used the following workloads for the bandwidth measurements: 100% Reads, 1 Reads : 1 Writes, 2 Reads : 1 Writes, 3 Reads : 1 Writes, Stream-Triad, 100% Non-Temporal-Writes and 1 Non-Temporal-Reads : 1 Writes. In all loads, reads and writes are conducted from/to separate buffers. All operations use AVX512 instructions (selected by `-Z` option), so all accesses always read 64 B, which equals the cache line size and a single CXL transaction. All accesses are sequential. The Stream-Triad scenario simulates memory access patterns (two reads from separate buffers and one write to a third buffer) from the Triad kernel from the synthetic STREAM benchmark for measuring sustainable memory bandwidth [87].

We also measured how rates scale for different numbers of concurrent threads accessing the memory, specifically, we evaluated 1, 2, 3, 4, 8, 12, 112 (all cores without HyperThreading threads – `-X` option in mlc), and 224 (all cores) cores. We used different per-thread buffer sizes (1 GB, 200 MB, or 30 MB) depending on the number of cores to fit into the ESMD memory.

Note on reproducibility: mlc does not measure 100% Non-Temporal-Writes and 1 Non-Temporal-Reads : 1 Writes workloads in peak injection bandwidth mode in the publicly available version. However, for those two workloads one can use loaded latency mode with `-W6` and `-W8`, respectively, to conduct the measurements.

b: Read Latency

For read latency measurement, we use mlc's idle latency mode with the following parameters `--idle_latency -b1G -t60`. In this mode, mlc first prepares a chain of pointers (last pointer points back to the beginning to create a cycle) and then goes through the chain by reading a value from memory and using this value as the next address to dereference. All pointers in the chain point to the address of the next cache line (to avoid hits into the cache) and CPU prefetchers are disabled before measurements (to avoid the CPU doing prefetch to hide latency). The mlc tool measures how many elements in the chain it was able to "visit" within a specified timeframe (60 seconds as specified by `-t60`). The mlc tool then divides the actual time for going through the chain (which may vary slightly as mlc does not check the time left after every step) by the number of visited elements to obtain the average read latency. Since this measurement is conducted on one thread we were able to use a large buffer for the chain (1 GB as specified by `-b1G`) to avoid data being only stored in the cache.

We use the idle latency mode on H_1 (i.e., H_1 is idling and only one thread conducts dependent reads) while H_2 generates load on the ESMD. This measurement method allows us to measure actual latency increments due to the load on the ESMD, while avoiding latency increments due to competition of multiple threads for resources on the same CPU (which would occur if H_1 generated traffic).

3) Results

We observe H_1 Memory access rate for different numbers of cores present in H_1, for different per-thread buffer size, and with and without hyper-threading. Hyper-threading allows handling of multiple tasks simultaneously by allowing multiple threads to run on a single core. All these different configurations are observed when H_2 is idle , refer Sec. IV-D3a.

In addition, we also observe H_1 memory access bandwidths (Sec. IV-D3b) and latencies (Sec. IV-D3c) when H_2 is connected to ESMD with COSM for 4 different scenarios, such as 0% Reads and Writes on H_2, 100% Reads on H_2, 100% Writes on H_2, 50% Reads and 50% Writes on H_2, along with implementing 4 different policies [1 1], [1 0] [0 1], [0 0] on H_2. Modern workloads are read-intensive, Intel's MLC is built in such a way to support read-intensive workloads, thereby explaining more number of reads than writes in read-write ratios. Note that reads performance is better than writes, therefore if there are more number of reads than writes (2 Reads : 1 Writes, 3 Reads : 1 Writes), then we observe a slight increase in the memory access bandwidth.

Moreover, we observe the bandwidth and latency for non-temporal reads and writes. The idea behind non-temporal is that we're giving the hint to CPU that this memory address is not going to be referenced in the near future, so caching it will probably give no benefit (or even hinder performance as there may be unnecessary read to cache, while only write to memory was needed). That's why CPU bypass cache in these cases. Stream [88] is a standard benchmark that helps to measure the sustainable memory bandwidth, considering different kernels. Triad is one such kernel, having notably the most complex scenario when compared to other kernels and at the same time related to High Performance Computing (HPC). The sustainable bandwidth will be less than the peak bandwidth, but note that there are random effects which could make the results vary.

a: H_1 Memory access rate for different numbers of cores present in H_1.

When the number of cores increases, the memory access rate also increases, but reaches the threshold after a few number of cores, this is because the system has reached the maximum available bandwidth. After a particular increase in number of cores, the bandwidth starts to decrease as the number of threads competing for the available limited resources increases. Moreover, the memory access rate also depends on the per-thread buffer size and the number of threads when hyper-threading is considered. Latency decreases with the decrease in buffer size, however, results in high CPU utilization, which could potentially lead to performance issues. Table 5 presents the H_1 Memory access rate for different numbers of cores present in H_1, considering different per-thread buffer size and hyper-threading. One notable observation is that the non-temporal writes reach the maximum bandwidth among all scenarios when the number of cores is 1. This behavior appears to be due to the asynchronism of the non-temporal write operations. For non-temporal writes, the CPU is informed that a particular address will not be referenced soon and the CPU is free to delay the write to memory to a convenient time in the future; thus, the CPU does not have to stall to wait for write completion.

For the subsequent evaluations in this study, we consider 224 cores with 30 MB per-thread buffer size and with hyperthreading, as this is a typical configuration for high-performance computing.

b: H_1 Mem. Acc. Bandwidth with different loads on H_2

When H_2 is idle (0% Reads and Writes), we get the maximum memory access bandwidth on H_1, refer table 6. Moreover, during this scenario (H_2 is idle), considering different policies (rules) on H_2 does not have any effect during the bandwidth measurement of H_1. This is because, H_2 is not sending or receiving data. However, implementing policies is required so as to provide isolation via air-gap.

When H_2 is not idle, either doing 100% reads or 100% writes, the configurations of policies on H_2 matters. During complete air-gap ([0 0]), the bandwidths are same when compared to the bandwidths when H_2 is idle. Also, when the configuration of policies on H_2 is not the same as the read and write operations on H_2, the bandwidth remains the same as the idle scenario. For e.g., consider a scenario when [01] configuration is enabled for H_2, H_2 can now write to ESMD with COSM, but cannot read from ESMD with COSM. Therefore, during [01] configuration, when H_2 only does 100% reads, there is no effect on bandwidth for H_1. Same applies when [10] configuration is enabled for H_2 and say H_2 is only doing writes. Table 7 presents the bandwidth of H_1, when H_2 only reads and table 8 presents the bandwidth of H_1, when H_2 only writes. Moreover, the bandwidths under [1 1] and [0 1] would be same, when H_2 only reads, as shown in table 7, as there are no writes done by H_2 and the bandwidths under [1 1] and [1 0] would be same, when H_2 only writes, as shown in table 8, as there are no reads done by H_2.

When H_2 does 50% Reads and 50% Writes, the bandwidths under [0 0], [0 1], [1 0] is a combination of tables 6, 7, 8, but the bandwidths under configuration [11] is low when compared to tables 6, 7, 8. This is because, now H_2 does both reads and writes affecting the bandwidth on H_1. Table 9 that presents H_1 memory access bandwidth when H_2 does 50% Reads and 50% Writes.

c: H_1 Memory Access Latency with different loads on H_2

As mentioned earlier, measuring write latencies on modern CPUs is complicated, refer Sec. IV-D1b. Therefore, we only observe the H_1 memory access latencies for only reads by H_1 with different loads on H_2. Table 10 presents the H_1 Memory Access Latency with different loads on H_2. The observation for H_1 Memory Access Latency is same as the observation for H_1 Memory Access bandwidth with respect to different policies (rules) enabled on H_2, but here the latency increases. For example, when H_2 is idle (0% Reads and Writes), the H_1 Memory Access Latency for all 4 policies of H_2 is same. During 100% reads on H_2 and when the policies ([1 1], [0 1]) allow read operations for H_2, the H_1 Memory Access Latency increases. Likewise, during 100% writes on H_2 and when the policies ([1 1], [1 0]) allow write operations for H_2, the H_1 Memory Access Latency increases. During the 50% reads and 50% writes on H_2 and when the policies ([1 1], [0 1], [1 0]) allow read and write operations for H_2, the H_1 Memory Access Latency increases. Notable application of

TABLE 5: Comparison Baseline: Memory (ESMD) access rate of host H_1 in Giga Byte per second [GB/s] (with full write and read permissions, i.e., $\mathbf{P}_{m,1} = [W_{m,1}, R_{m,1}]' = [1\ 1]'$) without any load from H_2, as a function of numbers of cores, without hyper-threading (HT) for 1 to 112 cores and with HT for 224 cores. The remaining evaluations in this study consider 224 cores with HT.

	Per-Thread Buffer Size							
	1 GB				**200 MB**			**30 MB**
	Number of Cores							
H_1 **Memory Accesses**	1	2	3	4	8	12	112	224 **(with HT)**
100% Reads	5.1	9.8	13.0	12.9	12.7	12.6	11.3	11.0
1 Reads : 1 Writes	5.7	7.3	7.3	7.4	9.2	8.1	8.5	8.5
2 Reads : 1 Writes	4.7	7.4	8.0	8.5	11.0	10.0	8.8	8.7
3 Reads : 1 Writes	5.7	9.1	9.8	10.1	11.3	10.3	9.1	9.0
Stream-Triad	6.7	9.5	9.8	9.9	13.5	10.0	8.9	11.0
100% Non-Temporal-Writes	14.6	14.1	14.0	13.5	12.1	11.2	9.8	9.7
1 Non-Temporal-Reads : 1 Writes	7.6	10.1	9.4	9.0	10.3	9.3	8.6	8.4

TABLE 6: Memory access rates [GB/s] of host H_1 (with full write and read permissions, i.e., $\mathbf{P}_{m,1} = [W_{m,1}, R_{m,1}]' = [1\ 1]'$) as a function of write-and-read permissions $\mathbf{P}_{m,2} = [W_{m,2},\ R_{m,2}]'$ of host H_2, for 0% Reads and Writes by host H_2.

	H_2 **Acc. Perm.** $[W_{m,2}, R_{m,2}]'$			
H_1 **Memory Accesses**	$[1\ 1]'$	$[1\ 0]'$	$[0\ 1]'$	$[0\ 0]'$
100% Reads	11.0	11.0	11.0	11.0
1 Reads : 1 Writes	8.4	8.4	8.5	8.5
2 Reads : 1 Writes	8.8	8.7	8.7	8.8
3 Reads : 1 Writes	9.0	9.0	9.0	9.0
Stream-Triad	10.9	11.2	11.0	11.3
100% Non-Temporal-Writes	9.7	9.7	9.7	9.7
1 Non-Temp.-Reads : 1 Writes	8.4	8.4	8.4	8.4

TABLE 7: Memory access rates [GB/s] of host H_1 (with full write and read permissions, i.e., $\mathbf{P}_{m,1} = [W_{m,1}, R_{m,1}]' = [1\ 1]'$) as a function of write-and-read permissions $\mathbf{P}_{m,2} = [W_{m,2},\ R_{m,2}]'$ of host H_2, for 100% Reads by host H_2.

H_1 **Mem. Acc. Bandwidth [GB/s]**

	H_2 **Acc. Perm.** $[W_{m,2}, R_{m,2}]'$			
H_1 **Mem. Acc. Configuration**	$[1\ 1]'$	$[1\ 0]'$	$[0\ 1]'$	$[0\ 0]'$
100% Reads	6.4	11.0	6.4	11.0
1 Reads : 1 Writes	5.9	8.4	5.9	8.4
2 Reads : 1 Writes	5.8	8.7	5.8	8.8
3 Reads : 1 Writes	5.8	9.0	5.8	9.0
Stream-Triad	7.0	10.9	6.9	10.9
100% Non-Temporal-Writes	4.7	9.7	4.7	9.7
1 Non-Temporal-Reads : 1 Writes	5.9	8.4	5.9	8.4

TABLE 8: Memory access rates [GB/s] of host H_1 (with full write and read permissions, i.e., $\mathbf{P}_{m,1} = [W_{m,1}, R_{m,1}]' = [1\ 1]'$) as a function of write-and-read permissions $\mathbf{P}_{m,2} = [W_{m,2},\ R_{m,2}]'$ of host H_2, for 100% Writes by host H_2.

H_1 **Mem. Acc. Bandwidth [GB/s]**

	H_2 **Acc. Perm.** $[W_{m,2}, R_{m,2}]'$			
H_1 **Mem. Acc. Configuration**	$[1\ 1]'$	$[1\ 0]'$	$[0\ 1]'$	$[0\ 0]'$
100% Reads	3.7	3.7	11.0	11.0
1 Reads : 1 Writes	5.6	5.6	8.4	8.5
2 Reads : 1 Writes	5.4	5.4	8.8	8.7
3 Reads : 1 Writes	4.9	4.9	9.0	9.0
Stream-Triad	6.0	6.0	10.9	10.8
100% Non-Temporal-Writes	4.8	4.8	9.7	9.7
1 Non-Temporal-Reads : 1 Writes	5.5	5.5	8.4	8.4

TABLE 9: Memory access rates [GB/s] of host H_1 (with full write and read permissions, i.e., $\mathbf{P}_{m,1} = [W_{m,1}, R_{m,1}]' = [1\ 1]'$) as a function of write-and-read permissions $\mathbf{P}_{m,2} = [W_{m,2},\ R_{m,2}]'$ of host H_2, for 50% Reads and 50% Writes by host H_2.

H_1 **Mem. Acc. Bandwidth [GB/s]**

	H_2 **Acc. Perm.** $[W_{m,2}, R_{m,2}]'$			
H_1 **Mem. Acc. Configuration**	$[1\ 1]'$	$[1\ 0]'$	$[0\ 1]'$	$[0\ 0]'$
100% Reads	2.9	3.7	6.4	11.0
1 Reads : 1 Writes	4.4	5.6	5.9	8.5
2 Reads : 1 Writes	3.7	5.4	5.8	8.7
3 Reads : 1 Writes	3.5	4.9	5.8	9.0
Stream-Triad	4.1	6.1	7.0	10.9
100% Non-Temporal-Writes	2.8	4.9	4.7	9.7
1 Non-Temporal-Reads : 1 Writes	4.2	5.5	5.9	8.4

using ESMD with COSM is that during air-gap configuration ($[0\ 0]$) on H_2, the H_1 Memory Access Latency is unaffected.

V. CONCLUSION

A. ARTICLE SUMMARY

B. LIMITATIONS AND FUTURE RESEARCH DIRECTIONS

1) Scalability

Scalability is a significant challenge for ESMDs with COSM as the number of connected hosts increases. For each additional host, the complexity of memory allocation, access permissions, and synchronization must be reorganized for all the participating hosts with the addition of a new host. The COSM permission matrix, which manages read/write permissions for shared memory regions, becomes larger and could become a limiting factor for the implementation of rule-tables in an ESMD with COSM. This scaling also amplifies the likelihood of contention for shared resources, especially in scenarios where multiple hosts require simultaneous access to the same memory regions. For exam-

TABLE 10: Memory (ESMD) read latency [ns] of host H_1 for different ESMD loads imposed by H_2

H_1 **Mem. Acc. Latency [ns]**				
	H_2 **Acc. Perm.** $[W_{m,2}, R_{m,2}]'$			
H_2 **Traffic**	$[1\ 1]'$	$[1\ 0]'$	$[0\ 1]'$	$[0\ 0]'$
0% Reads and Writes	460	460	460	460
100% Reads	548	460	547	461
100% Writes	921	921	462	460
50% Reads and 50% Writes	1020	860	536	459

ple, in multi-host applications, such as HPC or real-time analytics, the shared memory bandwidth and interconnect throughput can become bottlenecks as multiple hosts attempt to simultaneously access the same memory regions. These scalability issues are exacerbated at very high bandwidths, when the aggregate requested access bandwidth surpasses the memory access capabilities of the ESMD for the same memory regions. These scalability constraints necessitate robust architectural designs to dynamically balance resource allocation and prevent performance degradation.

Another scalability issue arises in the interconnects that are used to link hosts to the ESMD. Interconnect technologies, such as PCIe or CXL, have finite bandwidth and limited scalability for supporting numerous concurrent physical connections. Expanding the number of hosts across large infrastructures would require additional supporting hardware, such as CXL I/O switches [89] or higher-bandwidth interconnects, which increases costs and complexity. Additionally, as the number of hosts grows, managing synchronization and coordination between the hosts becomes exponentially more difficult. This scalability bottleneck often restricts the practical deployment of ESMDs in environments requiring dozens or hundreds of hosts, forcing developers to consider alternative data-sharing architectures or hierarchical designs to mitigate these limitations.

2) Synchronization

Synchronization is another critical challenge as ESMDs with COSM must support concurrent access to shared memory by multiple hosts while ensuring data integrity and consistency. Synchronization mechanisms, such as semaphores, locks, and barriers, are essential to prevent race conditions and to ensure orderly access to shared memory regions. However, these mechanisms introduce additional latency and overhead, particularly when multiple hosts are competing for access to the same memory. For instance, in producer-consumer models, where one host writes data while another reads the data, excessive locking or synchronization delays can significantly degrade throughput. Improperly designed synchronization logic may also lead to issues, such as deadlocks or resource starvation, disrupting critical workflows in real-time systems.

Additionally, the synchronization challenge becomes more pronounced in bidirectional communication scenarios where data flows between hosts in both directions. Efficient flow control and coordination are necessary to prevent traffic congestion and to ensure timely data transfer. For example, in industrial control systems, misaligned synchronization between a sensor and an actuator can lead to incorrect operations or delays, potentially causing system failures. While hardware-based synchronization primitives in COSM alleviate some of these issues, their implementation adds to the complexity of the system and can require significant tuning to meet the requirements of specific use cases.

3) Implementation

Implementing ESMDs with COSM for memory-intensive applications, such as AI model training, requires access to data in the ESMD by multiple hardware components, such as accelerators. Thereby, software components access the data over multiple iterations and copy the data multiple times over multiple regions moving between kernel space, user spaces, and across I/O, local and ESMD memory. The hardware architecture must incorporate memory request parsers, address mappers, access control logic, and synchronization primitives, all of which add to the FPGA or ASIC resource utilization. For example, the parsing and mapping of memory requests to the correct memory regions based on COSM policies demand intricate logic that scales with the number of hosts and memory regions. This increases the design time and may require iterative prototyping to ensure optimal performance. Additionally, hardware resources, such as Lookup Table (LUT), Block RAM (BRAM), and Digital Signal Processing (DSP) units on FPGAs, can become constrained due to the added complexities of COSM features, limiting the scalability of the implementation.

On the software side, the COSM framework must include robust APIs for host operating systems and applications to interact with the ESMD. These APIs, such as those for memory allocation (e.g., `cosm_malloc`) and synchronization, must be efficient and easy to use while ensuring compatibility across different platforms. Furthermore, integrating the ESMD with existing systems and workflows requires careful consideration of interconnect protocols, data formats, and application requirements. Testing and debugging such a complex system can be challenging, especially when dealing with real-time constraints or high-throughput workloads. Overall, the implementation of ESMD with COSM demands significant expertise in both hardware and software domains, making it a resource-intensive endeavor.

4) User-Defined Protocols in Shared-Memory Systems

The lack of user-defined protocols for data transfer over common memory regions creates significant rigidity in shared-memory systems, limiting their adaptability to specific application requirements. Standardized or pre-defined protocols may not adequately address the diverse needs of applications with unique data structures, synchronization patterns, or performance goals. For instance, an application that requires real-time data updates with minimal latency may face challenges when using a generic protocol that includes unnecessary synchronization mechanisms or metadata over-

head. Without the ability to tailor protocols, developers are forced to work within the constraints of existing frameworks, which may result in suboptimal performance, inefficient memory usage, or additional development overhead to create workarounds.

Additionally, the absence of user-defined protocols limits the ability to implement advanced features, such as application-specific quality-of-service (QoS) policies, custom data integrity checks, or domain-specific encryption methods. These features are often critical in critical applications, such as healthcare, finance, or defense, where data security, reliability, and prioritization are paramount. Without user-defined protocols, it becomes challenging to achieve fine-grained control over how memory regions are accessed and managed, leading to increased risk of data inconsistencies or unauthorized access. Furthermore, the inability to define protocols specific to an application's workload or traffic patterns makes it harder to optimize performance in multi-host or multi-threaded environments, leaving the system prone to bottlenecks, contention, and degraded scalability. This lack of flexibility restricts innovation and can prevent shared-memory systems from meeting the demands of emerging, highly specialized use cases, such as end-to-end real-time traffic (e.g., industrial IoT, voice, and video).

An application-defined protocol defines how applications interact with the shared memory, including the structure of the data, the methods for synchronization, and the rules for access control. Such a custom protocol can, for instance, be implemented in a financial trading applications where an application would define a memory region for order data, a separate region for market data, and synchronization primitives to ensure real-time updates and consistency among each of these, while other applications have simultaneous access to both order data and market data. Custom protocols could include additional in-memory compute features within the context of the ESMD, such as encryption for secure data sharing, compression for efficient use of memory, or quality-of-service (QoS) policies to prioritize critical data. While these protocols provide flexibility and can be optimized for specific use cases, they require significant development effort and careful design to ensure reliability and performance.

5) Striping

Striping is a mechanism in which the data is distributed across multiple memory regions to enhance performance or balance workloads. Striping presents significant challenges in the context of ESMD with COSM. One primary issue is the complexity of coordinating access to striped memory regions across multiple hosts. Each host must be aware of the striping pattern and the specific offsets required to access data correctly. Without precise coordination, hosts risk accessing incorrect data or causing corruption in critical sections of the memory. This becomes especially challenging in multi-host environments where simultaneous access to striped regions increases contention and the potential for race conditions. The overhead of maintaining metadata about the striping configuration (e.g., block size, start addresses) further complicates the implementation, particularly as the number of memory stripes or participating hosts scales.

Additionally, striping introduces latency and synchronization challenges when data dependencies exist between memory regions. For example, if one stripe contains metadata or headers needed to interpret the data in other stripes, hosts must access and process these sections sequentially, negating the performance benefits of parallelism. In scenarios involving COSM's permission matrix, striping further complicates access control, as each memory region within the stripe may require its own set of read/write permissions. Managing these permissions dynamically across a distributed system increases the overhead and complexity of the COSM framework. These challenges highlight the delicate balance between achieving high throughput through striping and maintaining robust, scalable memory management.

6) Memory Management in ESMD with COSM

Memory management in ESMD with COSM is inherently complex due to the need to efficiently allocate, deallocate, and monitor shared memory regions while maintaining strict access control. The COSM framework relies on dynamic memory management, where memory regions are assigned based on host requests, policies, and priorities. However, dynamically allocating shared memory to multiple hosts introduces the risk of fragmentation, which can lead to inefficient memory utilization and increased latency in fulfilling allocation requests. Furthermore, memory de-allocation must be carefully handled to prevent dangling pointers or premature release of regions still in use by other hosts, which could lead to data loss or corruption.

Another challenge lies in ensuring consistency and coherence across shared memory regions, particularly when hosts employ varying memory access patterns. The COSM framework must reconcile conflicting requests while preventing bottlenecks or deadlocks. For instance, in real-time systems, high-priority hosts may require immediate access to specific memory regions, necessitating preemption or reallocation strategies that do not disrupt ongoing operations for other hosts. The memory management policies must also account for fault tolerance and recovery, ensuring that memory regions can be reallocated or re-synchronized seamlessly in the event of host or system failures. These challenges underscore the need for sophisticated memory management algorithms and hardware support to achieve efficient, secure, and reliable operations in ESMD with COSM.

7) Address Space Management for ESMD with COSM

[90], [91] Managing address bits in ESMD with COSM for large-scale hardware cluster computing becomes increasingly complex to perform translations, look-up, and handling addresses within the ESMD. The management of address bit becomes especially complex as systems add support for 128-bit and 256-bit addressing mechanism to accommodate very large memory spaces. These extended address widths

are necessary to support large-scale clusters, such as the scale-out infrastructures for AI model developments. These massive shared memory regions provide simultaneous large-scale data access to processing agents, but they introduce significant challenges such as implementation, synchronization, and memory management. One of the primary challenges is the large size of the address translation tables required to map virtual to physical addresses, as each additional bit exponentially increases the potential addressable space. For 128-bit and 256-bit addressing, lookup tables and memory management units (MMUs) must process and store a very large number of entries, creating substantial memory overhead. Moreover, searching and updating such extensive tables in real-time can introduce latency, which would be which is a limiting for high-performance workloads in a large-scale cluster computing.

Another challenge arises from the hardware implications of supporting these wider address bit requirements. Extended address widths demand enhanced interconnects, buses, and data paths that are capable of handling the larger addresses, which increases hardware complexity for implementation and cost. Additionally, ensuring compatibility across heterogeneous clusters with different address widths adds another layer of difficulty. For example, some components may natively support 128-bit addresses, while others may be optimized for 256-bit addressing, requiring sophisticated bridge mechanisms for interoperability. This complexity further cascades into the COSM framework, as permission matrices tied to these extended addresses also grow, requiring more resources to enforce fine-grained access control. The balance between maintaining performance and ensuring scalability in the context of large address bit requirements is a key challenge for future ESMD designs.

8) No strict Isolation during a Bidirectional Transfer

During a bidirectional transfer, where read and write are enabled for multiple hosts in the same memory region, it is not possible for COSM to provide diode functionality (forward or reverse diode). Therefore, COSM does not have the ability to provide strict isolation for a bidirectional transfer in the same memory region. An alternative solution to increase the levels of security is to use a Multi-Level Security Cross Domain Solution that only allows authorized users to access authorized data.

The data is labeled into different levels either based on source or content (sensitivity level). Data that are being labeled based on source is simple to achieve. However, not all data coming from a host is going to be of same level, thereby using source-based labeling is simple but not efficient. On the other hand, content-based labeling is complex, but the data is labeled based on the sensitivity level of the data, thereby efficient. Once labeling is achieved, the CDS (MLS) will enforce Mandatory Access Control (MAC) to only allow authorized users to access authorized data. For e.g., Top-secret users can access all types of data (Top-secret and non-classified in this case), but non-classified users can only only access non-classified data and not Top-secret data.

Another alternative is to use 2 memory regions for TCP applications, one for data transfer or ACKs in the forward direction and the other for data transfer or ACKs in the reverse direction. As mentioned earlier, during unidirectional transfer, we can implement diode functionality, thereby providing strict isolation. In summary, if strict isolation is required, a minimum of 2 memory regions is required for TCP applications.

REFERENCES

[1] S. Jain et al., "Memory sharing with CXL: Hardware and software design approaches," in Proc. Workshop on Heterogeneous Composable and Disaggregated Systems (HCDS), 2024, pp. 1–4.

[2] K. Bowman, "Explaining CXL memory pooling and sharing," Aug. 2023, available from https://computeexpresslink.org/blog/explaining-cxl-memory-pooling-and-sharing-1049/.

[3] A. G. P. da Silva et al., "A model for the transmission of multimedia data flow based on the use of shared memory mechanism," in Proc. IEEE Int. Conf. on Adv. Inform. Networking and Applications Workshops, 2009, pp. 84–89.

[4] B. Fleisch and G. Popek, "Mirage: A coherent distributed shared memory design," ACM SIGOPS Operating Systems Review, vol. 23, no. 5, pp. 211–223, 1989.

[5] T. Huang et al., "TxCocket: An innovative solution for efficient cross-node data transmission enabled by CXL-based shared memory," CCF Transactions on High Performance Computing, pp. 1–14, 2025, DOI: 10.1007/s42514-024-00206-x.

[6] K. Lu et al., "Scythe: A low-latency RDMA-enabled distributed transaction system for disaggregated memory," ACM Trans. Archit. Code Optim., vol. 21, no. 3, pp. 57:1–57:26, Sep. 2024.

[7] T. Ma et al., "HydraRPC: RPC in the CXL era," in Proc. USENIX Ann. Techn. Conf. (USENIX ATC), 2024, pp. 387–395.

[8] A. Tanaka, "Multicast using distributed shared memory with commoditized transmission," in Proc. 20th IEEE Int. Conf. on Electronics Communications and Computers (CONIELECOMP), 2010, pp. 37–42.

[9] P. Alvaro et al., "NotNets: Accelerating microservices by bypassing the network," in Proc. 15th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys), 2024, p. 67–73.

[10] S. Mahar et al., "Telepathic datacenters: Efficient and high-performance RPCs using shared CXL memory," in Proc. 15th Ann. Non-Volatile Memories Workshop, 2024, pp. 1–2.

[11] Y. Ren et al., "Shared-memory optimizations for inter-virtual-machine communication," ACM Computing Surveys (CSUR), vol. 48, no. 4, pp. 49:1–49:42, 2016.

[12] W. Tang et al., "Yggdrasil: Reducing network I/O tax with (CXL-based) distributed shared memory," in Proc. ACM 53rd Int. Conf. on Parallel Processing, 2024, pp. 597–606.

[13] B. Verghese, A. Gupta, and M. Rosenblum, "Performance isolation: Sharing and isolation in shared-memory multiprocessors," ACM SIGPLAN Notices, vol. 33, no. 11, pp. 181–192, 1998.

[14] Y. Xu et al., "Position: CXL shared memory programming: Barely distributed and almost persistent," Preprint arXiv:2405.19626, 2024.

[15] X. Zhang et al., "DFabric: Scaling out data parallel applications with CXL-Ethernet hybrid interconnects," arXiv preprint arXiv:2409.05404, 2024.

[16] M. Ahn et al., "Enabling CXL memory expansion for in-memory database management systems," in Proc. 18th ACM Int. Workshop on Data Management on New Hardware (DaMoN), 2022, pp. 8:1–8:5.

[17] S.-J. Cha et al., "Trends in high speed fabric-interconnect-based memory centric computing architecture," Electronics and Telecommunications Trends, vol. 39, no. 5, pp. 98–107, 2024.

[18] C. Chen et al., "Next-gen interconnection systems with Compute Express Link: A comprehensive survey," Preprint arXiv:2412.20249, 2024.

[19] K. Choi et al., "ShieldCXL: A practical obliviousness support with sealed CXL memory," ACM Trans. on Architecture and Code Optimization, in print, 2025.

[20] C. Wang et al., "CXL over Ethernet: A novel FPGA-based memory disaggregation design in data centers," in Proc. IEEE 31st Ann. Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), 2023, pp. 75–82.

[21] X. Zhang et al., "Rethinking design paradigm of graph processing system with a CXL-like memory semantic fabric," in Proc. IEEE/ACM 23rd Int. Symp. on Cluster, Cloud and Internet Computing (CCGrid), 2023, pp. 25–35.

[22] H. Al Maruf and M. Chowdhury, "Memory disaggregation: Advances and open challenges," ACM SIGOPS Operating Systems Review, vol. 57, no. 1, pp. 29–37, 2023.

[23] M. Ewais and P. Chow, "Disaggregated memory in the datacenter: A survey," IEEE Access, vol. 11, pp. 20 688–20 712, 2023.

[24] M. K. Aguilera et al., "Memory disaggregation: Why now and what are the challenges," ACM SIGOPS Operating Systems Review, vol. 57, no. 1, pp. 38–46, 2023.

[25] N. Ding et al., "Methodology for evaluating the potential of disaggregated memory systems," in Proc. IEEE/ACM Int. Workshop on Resource Disaggregation in High-Performance Computing (REDIS), 2022, pp. 1–11.

[26] A. Geyer et al., "Working with disaggregated systems. What are the challenges and opportunities of RDMA and CXL?" pp. 751–755, 2023.

[27] ——, "Near to far: An evaluation of disaggregated memory for in-memory data processing," in Proc. 1st ACM Workshop on Disruptive Memory Systems, 2023, pp. 16–22.

[28] W. Hong et al., "A comprehensive simulation framework for CXL disaggregated memory," arXiv preprint arXiv:2411.02282, 2024.

[29] A. Puri et al., "Dracksim: Simulating CXL-enabled large-scale disaggregated memory systems," in Proc. 38th ACM SIGSIM Conf. on Principles of Adv. Discrete Simulation, 2024, pp. 3–14.

[30] Z. Wang et al., "Rcmp: Reconstructing RDMA-based memory disaggregation via CXL," ACM Trans. on Architecture and Code Optimization, vol. 21, no. 1, pp. 15:1–15:26, 2024.

[31] X. Wei et al., "Transactional indexes on (RDMA or CXL-based) disaggregated memory with repairable transaction," arXiv preprint arXiv:2308.02501, 2023.

[32] Y. Zhong et al., "Managing memory tiers with CXL in virtualized environments," in Proc. USENIX Symposium on Operating Systems Design and Implementation, 2024, pp. 37–56.

[33] Compute Express Link (CXL). [Online]. Available: https://computeexpresslink.org/

[34] S. Applebaum, T. Gaber, and A. Ahmed, "Signature-based and machine-learning-based web application firewalls: A short survey," Procedia Computer Science, vol. 189, pp. 359–367, 2021.

[35] H. Choi et al., "Cross domain solution with stateful correlation of outgoing and incoming application-layer packets," IEEE Access, vol. 12, pp. 26 830–26 838, 2024.

[36] S. Gupta et al., "DeeP4R: Deep packet inspection in P4 using packet recirculation," in Proc. IEEE INFOCOM, 2023, pp. 1–10.

[37] J. Liang and Y. Kim, "Evolution of firewalls: Toward securer network using next generation firewall," in Proc. IEEE 12th Ann. Computing and Communication Workshop and Conf. (CCWC), 2022, pp. 752–759.

[38] A. Siddiqui et al., "Survey on unified threat management (UTM) systems for home networks," IEEE Communications Surveys & Tutorials, vol. 26, no. 4, pp. 2459–2509, 2024.

[39] ——, "SUTMS: Designing a unified threat management system for home networks," IEEE Access, vol. 12, pp. 80 930–80 949, 2024.

[40] L. Singh and R. Singh, "Comparative analysis of traditional firewalls and next-generation firewalls: A review," in Latest Trends in Engineering and Technology. CRC Press, Boca Raton, FL, 2024, pp. 15–27.

[41] S. Abbas et al., "Securing data from side-channel attacks: A graph neural network-based approach for smartphone-based side channel attack detection," IEEE Access, vol. 12, pp. 138 904–138 920, 2024.

[42] A. Younis, Y. K. Malaiya, and I. Ray, "Assessing vulnerability exploitability risk using software properties," Software Quality Journal, vol. 24, pp. 159–202, 2016.

[43] L. Bilge and T. Dumitraş, "Before we knew it: An empirical study of zero-day attacks in the real world," in Proc. ACM Conf. on Computer and Commun. Security, 2012, pp. 833–844.

[44] A. S. Thyagaturu et al., "Operating systems and hypervisors for network functions: A survey of enabling technologies and research studies," IEEE Access, vol. 10, pp. 79 825–79 873, 2022.

[45] B. Zahran et al., "Security and privacy issues in network function virtualization: A review from architectural perspective," International Journal of Advanced Computer Science & Applications, vol. 15, no. 6, pp. 475–480, 2024.

[46] S. Zehra et al., "Securing the shared kernel: Exploring kernel isolation and emerging challenges in modern cloud computing," IEEE Access, vol. 12, pp. 179 281–179 317, 2024.

[47] D. Rico and P. Merino, "A survey of end-to-end solutions for reliable low-latency communications in 5G networks," IEEE Access, vol. 8, pp. 192 808–192 834, 2020.

[48] A. H. Abdi et al., "Security control and data planes of SDN: A comprehensive review of traditional, AI, and MTD approaches to security solutions," IEEE Access, vol. 12, pp. 69 941–69 980, 2024.

[49] W. Ding, Z. Yan, and R. H. Deng, "A survey on future internet security architectures," IEEE Access, vol. 4, pp. 4374–4393, 2016.

[50] J. Suzuki et al., "Disaggregation and sharing of I/O devices in cloud data centers," IEEE Transactions on Computers, vol. 65, no. 10, pp. 3013–3026, 2016.

[51] H3 Platform, "NVMe MR-IOV – High-performance storage solution for virtual environment deployments," Nov. 2021. [Online]. Available: https://www.h3platform.com/blog-detail/25?sort=latest&page=1

[52] J. Markussen et al., "Multi-host sharing of a single-function NVMe device in a PCIe cluster," in Proc. Workshops of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC24-W), 2024, pp. 1638–1645.

[53] V. Sundaravarathan et al., "Cross-Domain Solutions (CDS): A comprehensive survey," IEEE Access, vol. 12, pp. 163 551–163 620, 2024.

[54] L. Ni et al., "Construction of data center security system based on micro isolation under zero trust architecture," in Proc. 2nd Asia-Pacific Conf. on Communications Technology and Computer Science (ACCTCS), 2022, pp. 113–116.

[55] K. Z. Ye, "Application and parallel sandbox testing architecture for network security isolation based on cloud desktop," in Proc. Int. Conf. on Inventive Computation Technologies (ICICT), 2022, pp. 879–882.

[56] Y. Lan and Z. Wei, "A Linux security mechanism for multi-domain isolation environment," in Proc. 41st Chinese Control Conf. (CCC), 2022, pp. 7460–7465.

[57] S. Zehra et al., "Securing the shared kernel: Exploring kernel isolation and emerging challenges in modern cloud computing," IEEE Access, vol. 12, pp. 179 281–179 317, 2024.

[58] O. I. Falowo et al., "Evolving malware and DDoS attacks: Decadal longitudinal study," IEEE Access, vol. 12, pp. 39 221–39 237, 2024.

[59] Center for Strategic and International Studies (CSIS), "Significant Cyber Incidents," 2006, available from https://www.csis.org/programs/strategic-technologies-program/significant-cyber-incidents.

[60] S. Salim, N. Moustafa, and M. Reisslein, "Cybersecurity of satellite communications systems: A comprehensive survey of the space, ground, and links segments," IEEE Communications Surveys & Tutorials, in print, 2025.

[61] R. Shu et al., "A study of security isolation techniques," ACM Computing Surveys (CSUR), vol. 49, no. 3, pp. 50:1–50:37, 2016.

[62] S. Wong, B. Han, and H. D. Schotten, "5G network slice isolation," Network, vol. 2, no. 1, pp. 153–167, 2022.

[63] S. Narayan et al., "Hardware-assisted fault isolation: Going beyond the limits of software-based sandboxing," IEEE Micro, vol. 44, no. 4, pp. 70–79, 2024.

[64] D. Shen et al., "SecDINT: Preventing data-oriented attacks via Intel SGX escorted data integrity," in Proc. IEEE Conf. on Communications and Network Security (CNS), 2023, pp. 1–9.

[65] A. Saeed, A. Ahmadinia, and M. Just, "Hardware-assisted secure communication for FPGA-based embedded systems," in Proc. 11th Conf. on Ph.D. Research in Microelectronics and Electronics (PRIME), 2015, pp. 216–219.

[66] U. Lee and C. Park, "SofTEE: Software-based trusted execution environment for user applications," IEEE Access, vol. 8, pp. 121 874–121 888, 2020.

[67] Intel Corp., "Intel® Trust Authority." [Online]. Available: https://docs.trustauthority.intel.com/main/index.html#:~:text=Intel%C2%AE%20TDX%20allows%20an,of%20its%20validity%20and%20integrity.

[68] D. Moore, "CXL Fabric Management," Nov. 2022. [Online]. Available: https://computeexpresslink.org/blog/cxl-fabric-management-1089/

[69] I. A. Almaazmi et al., "Data diode for cyber-security: A review," in Proc. IEEE Int. Conf. on Artificial Intelligence of Things (ICAIoT), 2022, pp. 1–6.

[70] W. Li et al., "Survey on traffic management in data center network: From link layer to application layer," IEEE Access, vol. 9, pp. 38 427–38 456, 2021.

[71] S. Novakovic et al., "Mitigating load imbalance in distributed data serving with rack-scale memory pooling," ACM Transactions on Computer Systems (TOCS), vol. 36, no. 2, pp. 6:1–6:37, 2019.

[72] C.-C. Tu, C.-t. Lee, and T.-C. Chiueh, "Marlin: A memory-based rack area network," in Proc. Tenth ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS), 2014, p. 125–136.
[73] N. Nassif, et al., "Sapphire Rapids: The next-generation Intel Xeon scalable processor," in Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC), vol. 65, 2022, pp. 44–46.
[74] M. K. Khattak et al., "Effective routing technique: Augmenting data center switch fabric performance," IEEE Access, vol. 8, pp. 37 372–37 382, 2020.
[75] S. F. Drampalou et al., "A user-centric perspective of 6G networks: A survey," IEEE Access, vol. 12, pp. 190 255–190 294, 2024.
[76] T. Hoeschele et al., "Importance of internet exchange point (IXP) infrastructure for 5G: Estimating the impact of 5G use cases," Telecommunications Policy, vol. 45, no. 3, p. Art. no. 102091, 2021.
[77] S. Prasad Tera et al., "Toward 6G: An overview of the next generation of intelligent network connectivity," IEEE Access, vol. 13, pp. 925–961, 2025.
[78] M. Wei et al., "Envisioning the potential of 6G use cases—A dual perspective of innovation and business," IEEE Access, vol. 13, pp. 9831–9843, 2025.
[79] F. Bahramisirat, M. A. Gregory, and S. Li, "Multi-access edge computing resource slice allocation: A review," IEEE Access, vol. 12, pp. 188 572–188 589, 2024.
[80] G. Dobreff, A. Báder, and A. Pašić, "A QoE and availability-aware framework for network slice placement and resource allocation," IEEE Access, vol. 13, pp. 1481–1495, 2025.
[81] H. O. Otieno, B. Malila, and J. Mwangama, "Deployment and management of intelligent end-to-end network slicing in 5G and beyond 5G networks: A systematic review," IEEE Access, vol. 12, pp. 190 411–190 433, 2024.
[82] F. A. Butt et al., "On the integration of enabling wireless technologies and sensor fusion for next-generation connected and autonomous vehicles," IEEE Access, vol. 10, pp. 14 643–14 668, 2022.
[83] P. Y. Leong and N. S. Ahmad, "LiDAR-based obstacle avoidance with autonomous vehicles: A comprehensive review," IEEE Access, vol. 12, pp. 164 248–164 261, 2024.
[84] R. Yaswanth and M. R. Babu, "Revolutionizing automotive technology: Unveiling the state of vehicular sensors and biosensors," IEEE Access, vol. 12, pp. 192 786–192 812, 2024.
[85] Z. Liu, W. Zhang, and F. Zhao, "Impact, challenges and prospect of software-defined vehicles," Automotive Innovation, vol. 5, no. 2, pp. 180–194, 2022.
[86] V. Viswanathan et al., "Intel Memory Latency Checker v3.11b," May 2024, available from https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html.
[87] J. McCalpin, "Memory bandwidth and machine balance in current high performance computers," IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, pp. 19–25, Dec. 1995.
[88] K. R. Vamsi K Sripathi, "Optimizing Memory Bandwidth on Stream Triad," 2021, available from https://www.intel.com/content/www/us/en/developer/articles/technical/optimizin memory-bandwidth-on-stream-triad.html.
[89] D. Gouk et al., "Memory pooling with CXL," IEEE Micro, vol. 43, no. 2, pp. 48–57, 2023.
[90] C. S. Deshpande, A. Perais, and F. Pétrot, "Toward practical 128-bit general purpose microarchitectures," IEEE Computer Architecture Letters, vol. 22, no. 2, pp. 81–84, 2023.
[91] "RISC-V Instruction Set Manual," Dec. 2021, available from https://five-embeddev.com/riscv-user-isa-manual/Priv-v1.12/rv128.html.

VIGNESH SUNDARAVARATHAN is a current Ph.D. student and holds a Master's degree in Computer Engineering (Electrical Engineering) from the Ira A. Fulton Schools of Engineering, Arizona State University, Tempe, AZ, USA. He earned his bachelor's degree in Electronics and Communication Engineering from Pondicherry Engineering College, Puducherry, India. During his doctoral studies, he received an award through the Fulton Fellows program. During his bachelor's studies, he received the Above and Beyond Explorer Award for designing a prototype for people with physical disabilities.

MARTIN REISSLEIN (S'96-M'98-SM'03-F'14) received his Ph.D. in systems engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 1998. He is currently a Professor with the School of Electrical, Computer, and Energy Engineering, Arizona State University (ASU), Tempe, AZ, USA, and the Program Chair for Computer Engineering (CEN) at ASU.

AKHILESH S. THYAGATURU (M'17-SM'21) is a Sr. Software Engineer at Intel Edge Platforms Division (EPD), Intel Corporation, Chandler, AZ, USA, and an Adjunct Faculty in the School of Electrical, Computer, and Energy Engineering at Arizona State University (ASU), Tempe. He received the Ph.D. in electrical engineering from Arizona State University, Tempe, in 2017. He was with Qualcomm Technologies Inc., San Diego, CA, USA, as an Engineer from 2013 to 2015.

GURPREET SINGH KALSI is a Principal Engineer in Intel's Extreme Scale Computing team. His research focuses on hardware-software co-design for AI accelerators and system network architecture with co-packaged optics. He holds an MS in Electronics from Kurukshetra University, India, has over 35 patents, published 20+ papers, and contributed to 10+ revenue-generating product tapeouts.

JASON HOWARD received the M.S. degree in electrical engineering from Brigham Young University, Provo, UT. He is a Principal Engineer for the Extreme Scale Computing team within Intel's Office of the CTO. His research interests include parallel architectures, exascale processing, non-traditional computer mathematics.

JAN KAISRLIK received his M.S. degree in computer science from the Czech Technical University in Prague, Czech Republic. He is a Research Engineer on the Extreme Scale Computing team within Intel's Office of the CTO. His expertise covers topics such as computer architecture, the Linux kernel, and Embedded Linux.

BARTOSZ MATWIEJCZYK is a seasoned FPGA expert and Technical Leader at Intel, part of the Extreme Scale Computing team, holding both a Master's and Bachelor's degree in Computer Science. With 25 years of experience, Bartosz specializes in FPGA-based high-speed networking and encryption algorithm design, including advanced optimization, RTL design, and timing enhancements.

MAREK M. LANDOWSKI leads an applied research team at Intel, where he is part of the Extreme Scale Computing group within the Office of the CTO. He received his Ph.D. in electrical, electronic, and communications engineering from University College Dublin. Marek's research interests include software for novel architectures, quality-of-service networks, and neuromorphic computing.

PIOTR DOROZYNSKI received his master's degree in computer science from Gdansk University of Technology. He is a Software Research Engineer on the Extreme Scale Computing team within Intel's Office of the CTO. His expertise covers software/firmware development and product security assurance.

SANJAYA TAYAL leads an advanced R&D and prototyping team at Intel. His team is engaged in research topics on processor, software and system architecture and system simulation capabilities. Sanjaya has expertise in silicon design and development and has led several teams in various aspects of silicon development across Intel products. Sanjaya has a Masters' in Electrical Engineering from Southern Illinois University at Carbondale and a Bachelors in Electronics and Communication Engineering from National Institute of Technology- Karnataka (NITK).