MQSimNet: An Open-Source Simulator for Network-Based SSDs

***Abstract*—Solid-state drives (SSDs) have revolutionized non-volatile storage technology for computer systems and have become widely used in today's digital systems. This is due to SSDs offering high performance with a lower power consumption compared to traditional hard disk drives. While data sizes and the demand for higher-capacity storage systems grow, SSDs haven't seen considerable growth in capacity, mainly because it is extremely difficult to scale up the current architecture of SSD devices with reasonable performance. A revolutionary idea is to use a network to interconnect flash chips to solve the scalability issue. Partly due to the absence of an integrated, accurate, and extensible simulator, research on such architectures as the most viable structure for next-generation SSDs has been limited. In this paper, we present MQSimNet, an open-source integrated simulator for SSDs with networked flash chips, offering accurate results, ease of operation, and the possibility of low-effort extensibility.**

***Index Terms*—Network-based storage systems, Solid-State Drives (SSDs), Next-generation SSDs, Open source simulator.**

I. Introduction

Solid-state drives (SSDs) have become the preferred storage solution over traditional hard disk drives (HDDs) because of their superior speed, energy efficiency, and overall cost [12]. Despite the rapid advancements of processor and memory technologies, hard disk drive performance has been kept back mostly due to the limitation of their mechanical components; making them the bottleneck in many operations, particularly in data-intensive applications. The mechanical components of hard disk drives also take up considerable space, and SSDs have now far surpassed HDDs in terms of data density per unit volume. The moving of the physical components in HDDs impacts reliability and makes it harder to operate in close adjacency, driving up the operating costs. As a result, SSDs have gained widespread popularity as a more reliable and efficient alternative to HDDs.

Initially, however, SSDs were constrained by storage interfaces originally designed for HDDs, such as SATA [18] and SAS. These protocols, made primarily for HDDs, use a single queue for operations maintained by the operating system. The storage requests go through the operating system's storage stack, are queued, and then issued to the device one at a time. While these interfaces work well for hard disks, they limit the performance of SSDs. Unlike mechanical drives, the nature of SSDs enables them to perform multiple operations in parallel, which is not possible using SATA or SAS. Additionally, SSDs contain processing components that can offload certain storage operations from the operating system to the drive itself, thus, reducing the overhead of disk accesses.

To address these issues with traditional storage interfaces, a non-volatile memory express (NVMe) [17] protocol was introduced. NVMe devices connect to the rest of the system using PCI-Express (PCI-e) buses. As NVMe devices do the bulk of required processing in the device itself, the software stack required to interact with them is much simpler and thus, introduces lower overheads and can achieve higher throughputs [14]. Unlike traditional single-queue protocols, NVMe employs multiple device-level queues, allowing SSDs to overlap and process multiple operations simultaneously. They often take advantage of DRAM caching for writes and frequent reads to reduce operation latency observed by the rest of the system. Utilizing PCI-e busses enables these devices to take full advantage of their capabilities by giving them a high throughput and low latency connection to processors. Furthermore, NVMe specifications have received many revisions and currently the latest revision is NVMe 2.1. Each NVMe revision has brought incremental improvements to the interface, with NVMe 2.0 being the largest revision, adding support for modern SSD designs.

Despite these advantages, SSDs face certain limitations. Notably, the storage capacity of single devices is usually lower compared to HDDs, and higher-capacity SSDs come at a premium cost. This limitation is mainly due to how SSDs operate. Generally, increasing the capacity of SSDs leads to degradation of performance, both in terms of latency and throughput, and can potentially reduce the device lifespan. In SSDs, data is stored in flash chips that are grouped together in rows, each row connected to a single flash controller via a channel. Increasing SSD capacity can be achieved through two primary approaches: enlarging individual flash chips or incorporating more flash chips. Expanding the size of each flash chip can be done by increasing the dimensions of the die, plane, block, or page, or by employing higher-density cells such as multi-level cells (MLC) and quad-level cells (QLC). Increasing the physical size of components (die, plane, block, or page) introduces significant cost overheads and often cannot be implemented efficiently due to various design and manufacturing constraints. Alternatively, using higher-density flash cells (MLC, QLC) raises reliability concerns, as increased cell density elevates the error rate, leading to more frequent error-correction processes. This in turn degrades performance. Additionally, with lower margin for error and buffer between states voltages, lifespan of the cells and, consequently, the entire SSD device is reduced. In contrast, increasing the number of flash chips can potentially enhance device longevity but presents limitations in terms of performance and cost. Adding more chips to each channel, extends the channel length, increasing overhead and reducing throughput, due to the physical limitations of channels. Additionally, since each flash controller can operate only on one chip at a time, operations are blocking, and parallelism remains constrained despite the additional capacity.

Conversely, increasing the number of channels has the potential of increasing the parallelism in addition to increasing capacity, but, this approach is considerably more expensive. Increasing the number of flash controllers introduces more computation to the Flash Translation Layer (FTL) and requires stronger processors and larger caches for the devices, further driving up the cost.

As mentioned, the scalability of SSDs is not satisfactory for gaining maximum utilization of flash chips to reach high throughput. This is mostly due to the existence of a single path from each flash chip to FTL and blocking operations, which means the most number of operations that can be performed in parallel is equal to the number of unique flash controllers

(rows) with queued transactions.

Network-on-SSDs (NoSSD) proposed by Tavakol et al. [19] offers a scalable architecture for SSDs. The proposed method was the first that used an interconnection network for connecting flash chips to the FTL, thus, eliminating physical channel limitations and offering multiple paths between each chip and the FTL, paving the path to higher capacities SSDs as well as increasing possible parallelism and improving performance and throughput of the devices. Although this approach incurs higher costs for lower-capacity SSDs, it can be offset in higher-capacity SSDs by enabling the use of more cost-effective hardware components for other parts of the device. Other studies have also explored interconnection networks to link flash chips [1], [20], supporting the idea that using interconnection networks in the SSD back-end is an effective and innovative strategy for scaling SSD capacity. Consequently, having a reliable simulator to evaluate these architectures is crucial for advancing research in this area.

Despite its promise, research on this innovative architecture has been challenging and limited, primarily due to the absence of a reliable and accurate simulator. The need for such a simulator is especially critical because there is currently no physical hardware for this architecture to validate simulations against. In this work, we introduce MQSimNet, a flexible simulator that supports various interconnection network architectures in the SSD back-end. To ensure reliable simulation results, MQSimNet integrates two well-established simulators, MQSim [9] for SSD functionality and SuperSim [10] for networking, providing a comprehensive and robust platform. MQSimNet is both flexible and highly extensible, accommodating multiple network topologies, routing algorithms, and switching methods. Additionally, its modular design allows network and SSD components to be independently extended for customized configurations. The simulator provides detailed insights into both network and SSD operations, enabling comprehensive performance analysis. Unlike previous simulators for networked SSDs, MQSimNet's source code is openly accessible, promoting further research and development in this emerging field.

II. Background

A. The Architecture of SSD

Figure 1 demonstrates the structure of a modern SSD, composed of two main parts, front-end, and back-end. The front-end includes the Flash Translation Layer (FTL), the Host Interface Layer (HIL), and some Flash Controllers, while the back-end comprises arrays of flash memory chips.

Flash Translation Layer: FTL is the manager of SSD's operation and executes on an embedded microprocessor within SSDs, which is responsible for four main tasks: Address Mapping, Garbage Collection, Wear Leveling, and Cache Management. In the following, we discuss these tasks in detail.

Address Mapping: One of the primary responsibilities of the FTL is mapping logical addresses to physical addresses in flash chips, where each page of data is stored. When data is updated, a new page within the currently open block is allocated for the updated data, and the FTL updates the corresponding physical address. The previous data is then marked as invalid. Flash memory operates at page granularity for both reading and writing, with new data requiring a free page for storage [1].

Garbage Collection: In flash memory, a cell must be erased before new data can be written to it, and the erase operation occurs at block granularity. Over time, as data is updated, the number of invalidated pages increases, rendering them unusable for new or modified data. To address this, the FTL initiates a process called garbage collection (GC). During GC, valid pages in a selected (or 'victim') block are read and written to a free or new block. Once all valid pages are relocated, the victim block is erased, making it available for future write operations [15] [1].

Wear Leveling: The number of program-erase cycles, which is known as endurance in SSDs, is limited. As a result, a mechanism is needed to uniformly distribute the writing operations across all flash blocks to prolong the life time of the memory cells, and hence the SSD. This mechanism, called wear-leveling, tries to have all blocks wear out together [16].

Cache Management: Caching frequently accessed data in the SSD's internal DRAM significantly reduces the need to access flash memory, improving both speed and efficiency. By storing commonly used data in DRAM, SSDs can quickly retrieve this information without the latency associated with flash memory access. This approach helps lower wear on flash cells, conserves energy, and enhances overall performance, as DRAM access is much faster than flash memory access [1].

Host Interface Layer (HIL): The HIL serves as a bridge between the host system and the SSD controller [3]. HILs employ several communication protocols over the system I/O bus to communicate with the host system. Serial Advanced Technology Attachment (SATA) and Advanced Host Controller Interface (AHCI) interfaces have a single I/O queue to submit I/O requests which makes them insufficient to support high throughput demands of modern SSDs. To handle this inefficiency, NVMe was introduced, exploiting the PCI Express (PCIe) system bus which supports multiple I/O queues to enhance communication with the host system [1].

Flash Controllers (FC): The FC is a tiny embedded processor that is connected to multiple flash chips through a shared channel. It utilizes arbitration along with control/data pins for communicating with flash chips. For a write operation, the FC first randomizes the data to hinder high bit error rates. Next, it applies Error-Correcting Code (ECC) encoding to improve reliability [11]. In the third step, the FC sends a write request with the physical page address to the target flash chip followed by transferring the randomized, ECC-encoded data. For a read operation, the FC first transmits a read request to a specified flash chip. Subsequently, it receives the data from the flash chip and performs ECC decoding. Finally, it derandomizes the data to restore the original information.

Flash Chips: Flash chips are connected to a shared channel which is controlled by a flash controller. Each flash chip contains one or more dies which can operate individually, thereby enhancing system throughput by enabling parallel processing within the chip. However, since multiple dies share a single interface on the channel, contention can arise when multiple dies attempt to access the channel simultaneously, potentially limiting throughput.

Each die contains one or more planes, which are subdivisions designed to further increase parallelism. Within each plane, there are a large number of blocks (hundreds to thousands). These blocks are fundamental storage units, and each block is further divided into multiple pages, which are the smallest writable or readable units in flash memory [1]. This hierarchical structure of chip, die, plane, block, and page enables high-density storage and contributes to the

performance capabilities of SSDs, though it also introduces complexity in managing access and reducing contention across shared resources.

B. Interconnection Networks

Interconnection networks are fundamental to the performance of any networked system. Three main concepts that can affect network efficiency are topology, switching method, and routing algorithm. Below, we discuss these main factors.

Topology: Network topology determines how nodes are arranged in the network, with nodes connected through physical channels. The transmission of a packet from one node to another via a single physical link contributes one hop. The maximum number of hops along the shortest distance between any two node in the network defines the diameter, influencing overall communication latency. Many topologies have been proposed, with mesh being the most commonly used due to its straightforward implementation [4].

Switching Methods: The switching method determines how a message meets intermediate nodes in its path [5]. There are several switching methodologies, with two main categories: Circuit Switching and Packet Switching. On the other hand, Circuit Switching establishes a dedicated communication path between the source and destination nodes before any data transfer occurs. This path remains reserved for the duration of the communication, ensuring that data is transmitted without interference from other traffic. In Packet Switching, data is divided into discrete packets that are routed independently through the network. Packet Switching can be further subdivided into three main types based on how packets are buffered and transmitted: 1) store-and-forward [6], 2) virtual cut-through (or VCT for short) [7], and its buffer-limited variation 3) wormhole switching [8].

Routing Algorithm: Routing algorithms determine the path that a packet takes to reach its destination node within the network. Routing algorithms are divided into two main groups: 1) Deterministic routing, and 2) Adaptive routing. In deterministic routing, the packet has to traverse a fixed, pre-determined path to reach its destination. One of the most famous deterministic routing algorithms is dimension-ordered routing (DOR) in a multi-dimensional mesh, where a packet traverses the network dimensions in a specified order. On the other hand, adaptive routing provides the flexibility of multiple possible routes, allowing the network to dynamically adjust based on network conditions, such as congestion or link failures. Although adaptive algorithms are more sophisticated than deterministic ones, they improve the interconnection network's performance and reliability. The main requirement for any routing algorithm is to be deadlock-free [13]. Deadlock happens when there is a cyclic dependency in the network and packets cannot move because each packet is waiting for another to move, but none can advance due to congestion in shared buffers. deadlocks are usually prevented by limiting the use of (virtual) channels by the packets.

III. Network-based SSDs and MQSimNet

As SSDs grow in popularity, their architecture faces significant scalability challenges. Flash chips in conventional SSDs are connected to shared channels, which inherently limits their ability to achieve sufficient parallelism. Multiple I/O requests must traverse the shared channel serially, allowing only one flash chip to utilize the channel at a time. This restriction results in performance bottlenecks, underutilized resources, and reduced throughput. Furthermore, increasing the number of flash chips by extending channel length exacerbates latency issues, making it increasingly difficult to scale SSD capacity.

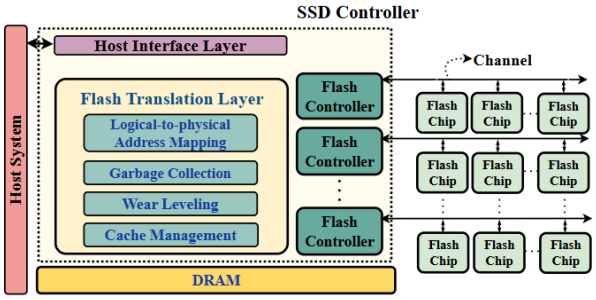

Fig. 1: High-level overview of a solid-state drive.

Several approaches have attempted to address these limitations, such as employing multichannel shared buses or increasing bus width. However, these solutions are constrained by physical limitations, such as the capacitance of wires, which prevent further bandwidth increases. To overcome these scalability barriers, researchers have proposed a revolutionary design: integrating interconnection networks into the SSD back-end.

The Network-on-SSD (NoSSD) architecture [19], introduced by Tavakkol et al., offers a promising solution by replacing conventional shared channels with an interconnection network (Figure 1. This design enables flash chips to communicate through dedicated routers organized in a network topology, such as a mesh. Each router is equipped with essential components, including input/output queues, a routing computation unit, a switch crossbar, and allocators for virtual channels and switches. This architecture introduces multiple paths between flash chips and the flash translation layer (FTL), significantly enhancing parallelism and throughput. Other studies [1], [20] have similarly explored interconnection networks for SSD back-ends, further demonstrating their potential to address scalability challenges and improve overall performance.

In a networked SSD architecture like NoSSD, I/O requests are processed by flash controllers and forwarded into the network, where they traverse based on a routing algorithm until they reach the router corresponding to the target flash chip. This method not only resolves throughput bottlenecks but also optimizes system performance, paving the way for SSDs with higher capacity and improved efficiency.

Despite the promise of networked SSD architectures, their advancement has been hindered by the absence of a reliable and openly accessible simulator. Current research in this domain often involves modifying existing simulators or developing bespoke tools for specific experiments. This lack of a unified simulation platform leads to fragmented efforts and hinders systematic comparisons across studies. Additionally, most existing simulators are hard-coded for specific architectures, making them less adaptable to evolving research needs.

To address these challenges, we introduce MQSimNet, an open-source, integrated simulator that bridges the gap between SSD and interconnection network simulation. By integrating MQSim for SSD functionalities and SuperSim for network modeling, MQSimNet provides a unified, modular platform

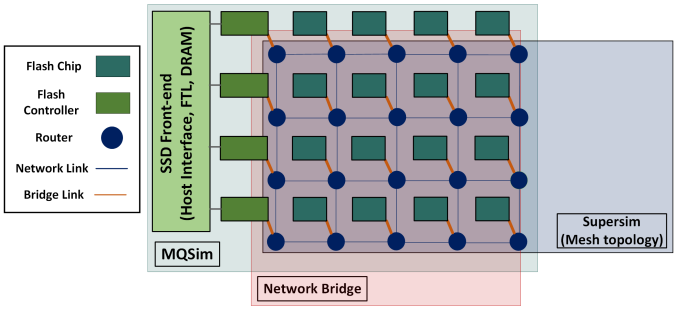

Fig. 2: High-level view of MQSimNet components.

for evaluating networked SSD architectures. It supports a wide range of network topologies, routing algorithms, and switching methods, enabling researchers to explore diverse configurations with minimal modifications. MQSimNet also offers detailed insights into SSD and network operations, fostering a deeper understanding of their interplay. By making the source code openly available, MQSimNet promotes collaboration and accelerates innovation in scalable SSD design.

A. *MQSimNet Components*

MQSimNet simulates a novel SSD architecture. As such, there are no real-world counterparts to the simulated devices that are yet built. This lack of real-world devices makes it extremely difficult to verify the proper functionality of this simulator. To counter this issue, MQSimNet was built from two well-known, verified, and time-tested simulators, each responsible for one part of this revolutionary design and a bridge to connect them. Figure 2 illustrates an overview of the main components of MQSimNet at a high level. MQSim [9] is designed to simulate almost every part of an SSD device except for communication between the SSD front-end and the back-end, and SuperSim [10] is used to create the remaining communication section of the MQSimNet simulator. As both MQSim and SuperSim have been verified to properly simulate the SSD and interconnection networking, respectively, we are confident that the final simulator proposed in this work would present accurate results.

MQSim: MQSim [9] is a discrete-event, open-source simulator written in C++ and introduced in 2018. It was specifically developed to address critical gaps in SSD modeling by supporting multi-queue (MQ) I/O request processing, accurate steady-state behavior modeling, and comprehensive end-to-end latency estimation. These features make MQSim particularly well-suited for studying modern MQ-enabled SSDs, while also retaining the ability to model traditional SATA-based SSDs. Released under the permissive MIT License, MQSim encourages adoption and extensibility in academic and industrial research.

One of MQSim's strengths is its detailed modeling of flash memory chips. It accurately considers three major latency components: the transfer of addresses and commands to the memory chip, the execution of read and write operations across various flash technologies (such as Single-Level Cell, Multi-Level Cell, and Triple-Level Cell), and the data transfer between memory chips and the SSD controller. The simulator also incorporates advanced command execution features and accounts for die- and plane-level parallelism constraints. This level of granularity ensures precise modeling of SSD back-end operations.

The front-end modeling in MQSim also stands out for its comprehensiveness. It includes accurate representations of NVMe and SATA host interfaces, with the NVMe interface supporting key features like multi-queue processing. Moreover, MQSim supports priority classes for NVMe queues, aligning with industry standards and enabling the study of quality-of-service (QoS) mechanisms. The data cache management component of MQSim is another area of innovation. It implements a DRAM-based cache with a Least Recently Used replacement policy, configurable to handle recently written data, recently read data, or both. Unlike previous SSD modeling tools, MQSim incorporates a DRAM contention model that captures the effects of concurrent accesses and latency. This feature enables a more realistic evaluation of cache performance and its influence on overall SSD behavior.

In the Flash Translation Layer (FTL), MQSim provides detailed modeling of key components such as address mapping, garbage collection, wear leveling, and transaction scheduling. It includes multiple state-of-the-art algorithms for each component, ensuring flexibility and accuracy in simulating modern SSD operations. For instance, MQSim supports advanced garbage collection mechanisms, including intra-plane data movement and program/erase suspension, which help reduce the interference of maintenance tasks with regular I/O operations. Additionally, the FTL components are inherently multi-queue-aware, enabling the exploration of performance isolation and QoS schemes for multi-queue SSDs.

Another feature of MQSim is that it supports various workloads and can simulate workloads accurately, achieving performance results that align with real-world, state-of-the-art SSDs within a margin of only 6%-18% error.

Despite its strengths, MQSim has some limitations. One notable challenge is its memory management, as prolonged simulations can lead to significant memory leakage, sometimes exceeding the size of the simulated data. Additionally, the event queue algorithm, while functional, lacks efficiency for handling long-running simulations. In MQSimNet, these issues have been addressed by enhancing memory management and optimizing the event queue algorithm. These improvements ensure greater stability and efficiency, particularly for large-scale and extended experiments, making MQSimNet an even more robust and reliable platform for SSD research.

SuperSim: SuperSim [10] is an open-source, event-driven interconnection network simulator designed to model the architectural details of large-scale networks with precision written in C++ and introduced in 2018. SuperSim is dedicated to flit-level simulation, capturing the finest details of router microarchitecture and network behavior. This level of granularity is essential for accurately studying the performance and design trade-offs of networks.

SuperSim's modular architecture supports a wide range of network topologies, routing algorithms, and router microarchitecture. Additionally, it includes both oblivious and adaptive routing algorithms, allowing researchers to evaluate a variety of routing strategies under different traffic patterns and workloads. The simulator's ability to isolate network and workload modeling ensures that it can be used to study the network behavior of diverse applications without predefined assumptions, enabling a wide range of experiments.

In addition to its core simulation capabilities, SuperSim

is supported by a suite of tools that enhance usability and productivity. These tools allow users to automate simulation runs, parse output data, analyze results, and generate plots for visualization.

Network Bridge: To minimize modifications to the base simulators, most of the functionality in MQSimNet was added in separate files and components. The key additions include a network bridge for MQSim, the SuperSim application to communicate with the mentioned network bridge, and a unified event loop.

Instead of considering a constant delay for bus communication in the original implementation of MQSim, in our work, a corresponding message is created and passed to the network bridge. This message is then forwarded to SuperSim, where it traverses the network to its destination. Once the message reaches its destination, it is sent back to the network bridge for processing. The SSD simulator considers the arrival time of this response as the time for completion of the communication.

The use of both simulators with minimal changes is enabled by the unified event loop. Instead of having a single event loop, our work maintains a separate event queue for MQSim and SuperSim. Instead of using the event loop of either simulator, a new event loop was developed, which considers the first event of both queues and executes the one that happens sooner. This way both base simulators can maintain their queue structure.

IV. Case Studies

To demonstrate the capabilities and flexibility of our integrated simulator, MQSimNet, we present two case studies that evaluate key aspects of SSD and interconnection network performance. The first case study investigates the impact of three distinct address mapping strategies, one static and two dynamic algorithms, on the throughput (measured in Input/Output operations Per Second (IOPS)) and overall speedup in networked SSD architectures. This analysis highlights how address mapping approaches influence performance in highly parallel storage systems. The second case study examines the effect of different routing algorithms, specifically deterministic (dimension-ordered) routing and fully adaptive routing, on throughput and average SSD response time. By exploring these scenarios, we aim to illustrate the effectiveness of MQSimNet in providing detailed insights into SSD and network behavior, enabling researchers to optimize designs for scalable and high-performance storage systems. Table **??** outlines the essential parameters utilized in the simulation experiments.

A. Impact of Address Mapping

Efficient utilization of flash chips is a key challenge in SSD design, and traditional static address mapping strategies often fall short in this regard. In conventional SSDs with a bus-based back-end, the address mapping unit in the FTL assigns transactions to specific locations based on a static mapping policy. This approach typically involves mapping transactions sequentially to channels, ensuring that each channel is used in turn. However, when the designated location is busy, the transaction must wait, leading to suboptimal utilization and increased latency. Moreover, this static approach is inherently constrained by the bus-based architecture, which limits the number of parallel operations that can be performed.

In contrast, SSDs with a networked back-end provide multiple paths to each flash chip, enabling the use of dynamic address mapping strategies that can better leverage the available resources. Dynamic mapping algorithms take into account the busyness and idleness of destinations, dynamically directing transactions to free resources. These algorithms can operate at varying levels of granularity. For instance, one dynamic approach assigns a specific die and plane address and checks each flash chip sequentially until it finds an idle plane and die matching the assigned address. A more flexible dynamic strategy may fix only the plane address and check dies sequentially until it finds an idle die at the specified plane. These dynamic strategies exploit the inherent parallelism of the networked back-end architecture, reducing contention and improving throughput.

To evaluate the impact of these address mapping strategies, we conducted simulations using the MQSim simulator for the baseline bus-based SSD and MQSimNet for SSDs with a networked back-end. We used 11 different traces from the MSR Cambridge workload and compared the performance of three mapping algorithms: a static mapping algorithm and two dynamic algorithms (chip-level and die-level). Our analysis focuses on SSD throughput, measured in terms of IOPS, and speedup, defined as the reduction in execution time relative to the baseline.

The results in Figure 3, normalized against the baseline bus-based SSD, demonstrate that dynamic address mapping algorithms significantly enhance throughput. By dynamically adapting to resource availability, these algorithms effectively minimize idle time and maximize parallelism, underscoring the advantages of networked back-end architectures for scalable SSD.

B. Impact of Routing Algorithm

The choice of routing algorithm in the SSD back-end plays a critical role in determining system performance, particularly in terms of network latency. When a transaction arrives at the SSD back-end to be routed to its destination flash chip, the routing algorithm determines the path it takes within the networked architecture. This decision can significantly impact the efficiency and responsiveness of the system, as the routing approach directly influences how well the network adapts to varying traffic conditions.

Routing algorithms can be broadly classified as deterministic or adaptive. Deterministic routing algorithms, such as dimension-ordered routing, assign a predefined path for each transaction. Regardless of network conditions, transactions follow this fixed route to reach their destination. While deterministic algorithms are straightforward and predictable, they lack the flexibility to account for real-time network congestion, which can lead to suboptimal performance in high-traffic scenarios.

In contrast, adaptive routing algorithms, such as Duato's routing algorithm [2], dynamically select paths based on the current state of the network. These fully adaptive algorithms evaluate network conditions, such as congestion or idle resources, to choose the most efficient route for each transaction. This dynamic decision-making capability allows adaptive routing algorithms to balance traffic more effectively and reduce contention, potentially leading to lower latency and improved throughput.

To analyze the impact of deterministic and adaptive routing on SSDs with a networked back-end, we conducted simulations using the MQSimNet simulator. Specifically, we

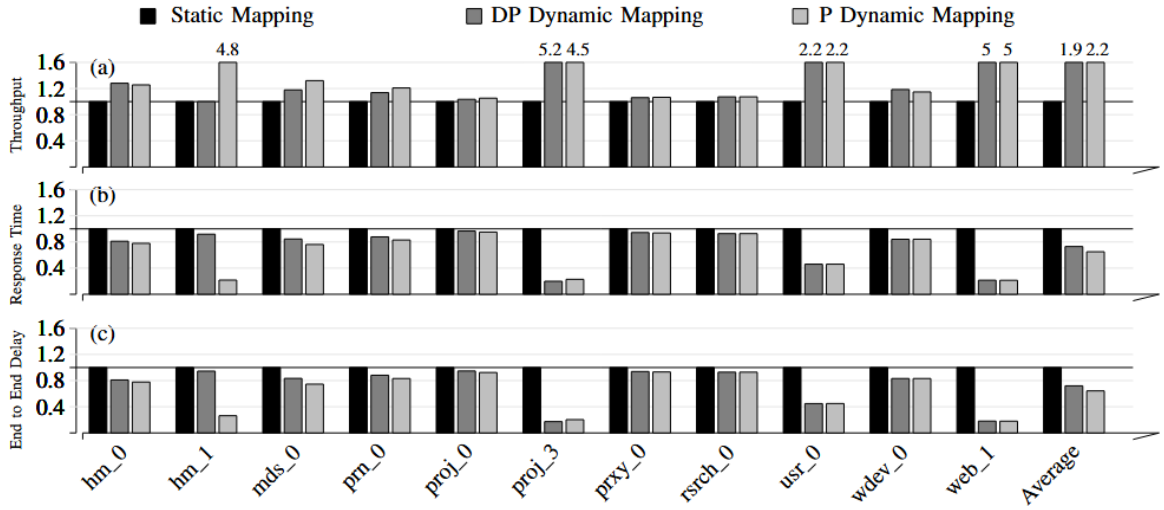

Fig. 3: Obtained results normalized to baseline bus-based SSD; (a) Normalized Throughput, (b) Normalized Device Response Time, and (c) Normalized End to End Request Time.

evaluated the dimension-ordered routing algorithm as the deterministic approach and Duato's routing algorithm as the fully adaptive alternative. The evaluation used 13 distinct traces from the MSR Cambridge workload, focusing on the latency of the network component in the SSD back-end.

The results, summarized in the Table **??** of average network latency, indicate that while Duato's fully adaptive routing algorithm shows some efficiency gains compared to dimension-ordered routing, the differences are not substantial. This is primarily because the traces used in the study were not large enough to generate high traffic levels, where adaptive routing algorithms typically demonstrate their full potential. Moreover, adaptive algorithms like Duato tend to perform better in topologies with greater diversity, such as 3D mesh networks, where multiple routing options are available to effectively avoid congestion. By supporting detailed evaluations of routing strategies under various conditions, MQSimNet enables researchers to systematically explore and understand the performance implications of different algorithms in SSDs with networked back-end architectures.

V. Research Directions Enabled by MQSimNet

The integration of networked back-end architectures in SSDs opens a wealth of opportunities for future research and innovation. MQSimNet, with its modular and extensible framework, provides an invaluable tool for exploring these avenues. Below, we outline several promising research directions enabled by MQSimNet.

Exploration of Network Topologies: Traditionally, research on SSDs with networked back-ends has been limited to mesh network topologies. However, the scalability and performance of SSDs can vary significantly depending on the topology of the back-end interconnection network. MQSimNet supports a variety of common network topologies out of the box, such as torus, hypercube, and dragonfly, and enables researchers to create custom topologies. This flexibility allows for systematic evaluation of topologies to identify the most suitable ones for specific workloads, performance targets, or cost constraints. Such explorations can guide the development of more efficient and scalable SSD architectures.

Distributed Flash Translation Layer (FTL): The FTL is the central component of an SSD controller, managing tasks like address mapping, wear leveling, garbage collection, and error correction. In conventional SSDs, these responsibilities are centralized in the front-end controller, creating potential bottlenecks and limiting scalability. MQSimNet allows researchers to explore a distributed approach, where FTL responsibilities are shifted to the back-end, closer to the flash chips. This redistribution reduces the processing load on the front-end and leverages the networked architecture to distribute FTL tasks uniformly across the system. By aligning computational tasks with data locality, this approach enhances scalability, improves performance, and optimizes resource utilization, especially in large-scale storage systems.

Cost-Optimized SSD Designs: The ability to distribute FTL components across the device opens avenues for cost-efficient SSD designs. While the inclusion of network components may increase costs for smaller SSDs, larger devices can benefit from the reduced cost of FTL components and distributed processing. MQSimNet facilitates research into finding the optimal balance between performance and cost by allowing fine-grained customization and evaluation of hardware configurations. Researchers can identify design strategies that achieve high performance at minimal cost, making SSDs more accessible and scalable.

In-Storage Processing (ISP): The distributed nature of networked back-end architectures presents an exciting opportunity for integrating in-storage processing capabilities. In designs like NoSSD and Venice, tiny processors adjacent to each flash chip already handle local control and communication tasks. MQSimNet enables the exploration of extending these processors' functionality to perform computational tasks directly within the storage network. By reusing existing on-chip resources, this approach minimizes hardware costs, reduces data movement latency, and enables efficient parallel execution of computational tasks. Researchers can leverage MQSimNet to evaluate the feasibility, performance gains, and cost implications of ISP in SSDs with networked back-ends.

Data Tiering within a Single Device: In traditional SSDs, all flash chips exhibit the same access latency, typically deter-

TABLE I: Parameters for MQSimNet

SSD Parameters		Network Parameters	
Host Interface	PCIe 3.0 (NVMe 1.2)	Network Topology	Mesh
SSD Capacity	1.5 TB	Routing Algorithm	XY Routing
Write Cache	256 MB, CMT: 2 MB	Switching	Packet Switching
Queue Fetch Size	512	Routing Architecture	input-output-queued (IOQ)
Flash Communication	ONFI 3.1 (NV-DDR2)	Flit Size	64 Byte
NAND Config	8 channels, 8 chips/channel, 2 die/chip, 4 planes/die, 1024 blocks/die, 4KB page	Flits in Packet	512
Channel Width	64 Bytes	Number of VCs	2
Read Latency	3 μs	Input Buffer Size	6 KB
Program Latency	100 μs	Output Buffer Size	8 KB
Erase Latency	1 ms	Network Channel Latency	1 ns

TABLE II: Network Latency in SSD's back-end

Trace	Average Network Latency (ns)	
	Dimension-ordered Routing	Duato Routing
hm_0	17625.76	17350.71
hm_1	2712.50	2612.55
mds_0	31324.61	31068.56
prn_0	24571.76	24222.59
proj_0	51125.79	50725.80
proj_3	2232.99	2211.89
prxy_0	55084.71	54709.74
rsrch_0	44259.83	43776.13
urs_0	5161.83	4996.53
wdev_0	25923.49	25566.43
web_1	3243.74	3213.14
Average	23933.36	23677.64

mined by the farthest chip's distance from the FTL. Networked back-end architectures introduce varying latencies based on chip proximity to the FTL. MQSimNet enables researchers to study data tiering strategies that leverage these latency variations. For instance, hot data (frequently accessed) can be placed on chips closer to the FTL, while cold data (rarely accessed) can be stored on more distant chips. Such strategies can optimize performance, reduce network congestion, and extend chip lifetime. Additionally, MQSimNet allows the study of how these tiering strategies interact with garbage collection and wear leveling processes.

Heterogeneous Flash Chips: In conventional SSDs, all flash chips must operate at the speed of the slowest chip due to shared communication channels. In SSDs with networked back-ends, independent communication paths enable the use of heterogeneous flash chips. MQSimNet provides the tools to investigate the impact of combining high-performance chips for hot data with lower-cost, lower-performance chips for cold data. This flexibility allows for cost-efficient designs that maintain high performance and durability. Researchers can explore configurations that balance capacity, cost, and performance, advancing the development of heterogeneous SSDs.

VI. Conclusion

In this work, we introduced MQSimNet, a simulator for network-equipped SSD devices that offers fairly accurate results for SSD and network-related operations of such devices. MQSimNet simulates both multi-queue NVMe and SATA SSDs. Relying on MQSim for SSD simulation, MQSimNet faithfully models all individual SSD components, leading to accurate steady-state behavior and end-to-end I/O request latency modeling. Similarly, network modeling is done by SuperSim which models every individual component and captures the specific behavior of those components, leading to dependable results. The source code of MQSimNet is available at https://shorturl.at/bseOb.

VII. Acknowledgment

This research was supported by the Brain Pool program funded by the Ministry of Science and ICT through the National Research Foundation of Korea (RS-2023-00284115).

References

[1] R. Nadig, M. Sadrosadati, H. Mao, N. M. Ghiasi, A. Tavakkol, J. Park, et al. "Venice: Improving Solid-State Drive Parallelism at Low Cost via Conflict-Free Accesses." In Proceedings of the 50th Annual International Symposium on Computer Architecture, pp. 1-16. 2023.

[2] A. E. Kiasari, H. Sarbazi-Azad, and M. S. Rezazad. "Performance comparison of adaptive routing algorithms in the star interconnection network." In Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05), pp. 8-pp. IEEE, 2005.

[3] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir. "HIOS: A host interface I/O scheduler for solid state disks." ACM SIGARCH Computer Architecture News 42, no. 3 (2014): 289-300.

[4] A. Gheibi-Fetrat, N. Akbarzadeh, S. Hessabi, and H. Sarbazi-Azad. "Tulip: Turn-Free Low-Power Network-on-Chip." IEEE Computer Architecture Letters (2023).

[5] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad. "Virtual point-to-point connections for NoCs." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 29, no. 6 (2010): 855-868.

[6] A. S. Tanenbaum. "Computer Networks, Prentice-Hall International." Upper Saddle River, NJ 2 (1996).

[7] E. Salminen, A. Kulmala, and T. D. Hamalainen. "Survey of network-on-chip proposals." white paper, OCP-IP 1 (2008): 13.

[8] Dally, and Seitz. "Deadlock-free message routing in multiprocessor interconnection networks." IEEE Transactions on computers 100, no. 5 (1987): 547-553.

[9] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu. "MQSim: A framework for enabling realistic studies of modern Multi-QueueSSD devices." In 16th USENIX Conference on File and Storage Technologies (FAST 18), pp. 49-66. 2018.

[10] N. McDonald, A. Flores, A. Davis, M. Isaev, J. Kim, and D. Gibson. "SuperSim: extensible flit-level simulation of large-scale interconnection networks." In 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 87-98. IEEE, 2018.

[11] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu. "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives." Proceedings of the IEEE 105, no. 9 (2017): 1666-1704.

[12] S. Cho, B. Kim, H. Cho, G. Seo, O. Mutlu, M. Kim, et al. "AERO: Adaptive Erase Operation for Improving Lifetime and Performance of Modern NAND Flash-Based SSDs." In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, pp. 101-118. 2024.

[13] J. Duato, S. Yalamanchili, and Lionel Ni. Interconnection networks. Elsevier, 2002.

[14] J. Hwang, M. Vuppalapati, S. Peter, and R. Agarwal. "Rearchitecting linux storage stack for µs latency and high throughput." In 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), pp. 113-128. 2021.

[15] W. Kang, D. Shin, and S. Yoo. "Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD." ACM Transactions on Embedded Computing Systems (TECS) 16, no. 5s (2017): 1-20.

[16] S. Lim, S. Lee, and B. Moon. "FASTer FTL for enterprise-class flash memory SSDs." In 2010 International Workshop on Storage Network Architecture and Parallel I/Os, pp. 3-12. IEEE, 2010.

[17] NVM EXPRESS. NVM Express Base Specification, 8 2024. Rev. 2.1.

[18] SERIAL ATA REVISION 3.1 – GOLD REVISION SERIAL ATA INTERNATIONAL ORGANIZATION. Serial ATA Revision 3.1, 7 2011. Rev. 3.1.

[19] A. Tavakkol, M. Arjomand, and H. Sarbazi-Azad. "Network-on-SSD: A scalable and high-performance communication design paradigm for SSDs." IEEE Computer Architecture Letters 12, no. 1 (2012): 5-8.

[20] A. Tavakkol, M. Arjomand, and H. Sarbazi-Azad. "Design for scalability in enterprise SSDs." In proceedings of the 23rd international conference on parallel architectures and compilation, pp. 417-430. 2014.