

EEMU: A FEMU-based Accurate, Parametric Ethernet-SSD Emulator

Xikun Jiang*, Chao Li[†], Tianyu Wang[‡], Xiaowei Chen[†], Zhaoyan Shen[§], Zili Shao*

*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China

[†]Inspur Cloud Information Technology Co, Ltd.

[‡]College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China

[§]School of Computer Science and Technology, Shandong University, Qingdao, China

Abstract—Ethernet-SSDs incorporate built-in Ethernet connectivity, allowing them to function as standalone network-attached storage devices. This architecture enables efficient and scalable disaggregated storage by providing direct network access without host dependencies. Understanding their internal architecture and fine-grained performance interactions is critical for advancing this emerging design, yet is difficult to study on proprietary hardware. In this paper, we present EEMU, a configurable and accurate Ethernet-SSD emulator built on the FEMU framework. EEMU reproduces the end-to-end NVMeoF I/O translation path and models latency across three key components: the Network Interface, the NVMeoF Target Module, and the AXI Bus. It further employs a fine-grained parametric latency model that decomposes end-to-end latency into component-level contributions, enabling systematic exploration of architectural trade-offs via controlled parameterization. Experiments show that EEMU closely matches both component-level behaviors and end-to-end performance across diverse Ethernet-SSD configurations.

Index Terms—Ethernet-SSD, Emulator, NVMeoF

I. INTRODUCTION

Ethernet-SSDs [1] integrate network interfaces into drives and co-locate the NVMeoF target and SSD controller within an on-disk SoC, enabling offloading heavy storage stack to the on-disk lightweight ARM cores. However, existing SSD emulators do not model Ethernet-SSD semantics: trace-driven simulators (e.g., DiskSim+SSD [3], FlashSim [4], SSDSim [5]) lack real OS interaction and dynamic workload support, while execution-driven emulators such as FEMU [6] provide detailed flash and firmware modeling but omit Ethernet/NVMeoF-specific components. Therefore, it is essential to expose the internal architecture of Ethernet-SSDs and characterize the fine-grained performance attributes and interactions among their constituent components.

To address this gap, we present EEMU, a FEMU-based emulator that captures the core architecture of Ethernet-SSDs. EEMU extends FEMU with three key components: an RDMA-capable network interface, a full NVMeoF target stack, and an AXI system bus. It runs in a QEMU-emulated ARM environment to provide timing-approximate behavior, enabling systematic architectural exploration and fine-grained performance analysis for Ethernet-SSD-based disaggregated storage systems.

The work described in this paper is partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (GRF 14200224).

II. IMPLEMENTATION

EEMU is a modular Ethernet-SSD emulator built on FEMU, combining a function model for I/O execution and a calibrated latency model for timing, as detailed in Sections II-A and II-B.

A. Function Model of EEMU

EEMU’s function model operates within a QEMU-emulated ARM environment, executing unmodified Linux software stacks to accurately replicate an Ethernet-SSD’s execution context. As shown in Figure 1, the I/O processing pipeline comprises three key modules: the Network Interface, the NVMeoF Target Module, and the SSD Emulator.

The Network Interface emulates a physical NIC with full RDMA capabilities, relying on Linux SoftRoCE to provide standards-compliant RDMA functionality. The NVMeoF Target Module implements the standard NVMeoF I/O stack, including RDMA-based transport, NVMeoF capsule parsing and translation via the Linux nvmet subsystem, block-layer scheduling and dispatch, and NVMe command management with submission/completion queues. The SSD Emulator, built on FEMU, reproduces SSD internal behaviors such as latency characteristics, parallelism, and flash-specific constraints.

B. Latency Model of EEMU

As shown in Figure 1, EEMU adopts a fine-grained latency model that decomposes end-to-end I/O latency into three components: AXI bus, NIC, and NVMeoF target latencies.

AXI Bus Latency. We model NIC-DRAM on-chip transfers over memory-mapped AXI4 as the sum of bandwidth-limited transfer time, burst setup overhead, and fixed arbitration delay.

NIC Latency. We model RoCEv2 RDMA latency by explicitly capturing the TX/RX protocol pipeline, combining fixed per-packet processing costs with bandwidth-dependent transmission time, including MTU-aware segmentation and control-plane overheads.

NVMeoF Target Latency. We segment the software target stack into RDMA transport, capsule handling, block layer, and NVMe layer, model each stage independently, and temporally compose them to preserve inter-stage dependencies and enable stage-wise optimization.

Model Calibration and Generality. We calibrate all stage models via microbenchmarks on a real ARM-based NVMeoF target and fit size-dependent behaviors using cubic spline

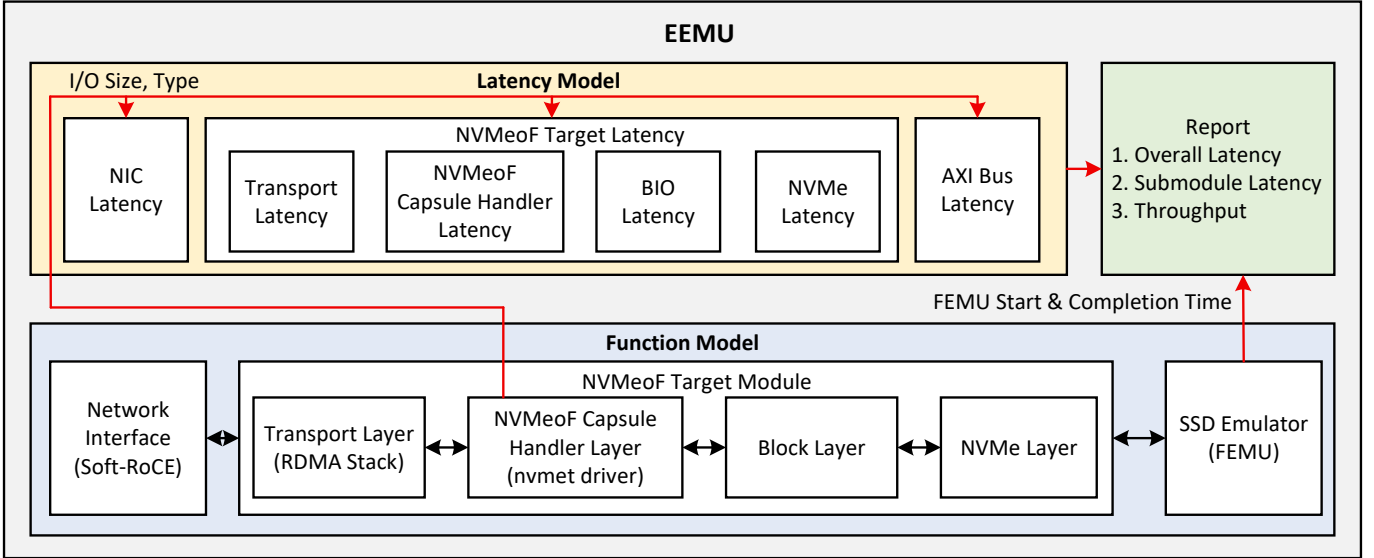


Fig. 1: EEMU architecture.

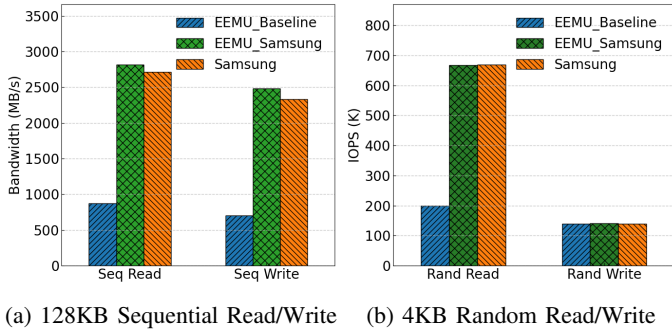


Fig. 2: Bandwidth and IOPS comparison between EEMU and Samsung Ethernet-SSD.

smoothing regression. To accommodate diverse Ethernet-SSD designs, the framework is parameterized to selectively include or bypass specific target submodules.

III. RESULTS

Our evaluation uses two QEMU-based VMs configured as an NVMeoF target and client. Each VM emulates an 8-core ARM Cortex-A53 with 8 GB DRAM and runs Linux 4.19 with the full NVMeoF stack. We calibrate the target-side latency model using fine-grained measurements from a Xilinx FPGA SoC platform (quad-core Cortex-A53) running the same kernel. For validation, we use Samsung’s Ethernet-SSD as the baseline and its official white paper [2] for bandwidth and IOPS. With default kernel settings and a 1020 KB NVMeoF payload limit, we issue sequential 0.5–1020 KB read/write I/Os via `nvme-cli`, capture target-side latency using `ftrace`, and apply cubic-spline smoothing regression to derive the latency model.

To demonstrate the effectiveness of EEMU, we evaluate two architectures: `EEMU_Baseline`, which processes NVMeoF commands through the full target stack, and `EEMU_Samsung`, which follows Samsung’s Ethernet-SSD design by extracting NVMe read/write commands directly at the RDMA transport

layer and bypassing intermediate layers. Both configurations match Samsung’s hardware parameters (25 Gbps NIC, 1024-byte MTU, 500 MHz AXI bus with 1024-bit width and 256-beat bursts), with estimated flash latencies of 12 μ s (read) and 45 μ s (program). As shown in Fig. 2, `EEMU_Samsung` closely aligns with Samsung’s reported results, achieving sequential read/write bandwidths of 2817/2482 MB/s (errors of 3.68%/6.39%) and 4 KB random read/write performance of 667/140 K IOPS (errors of 0.30%/0.72%).

IV. CONCLUSION

In this paper, we present EEMU, the first full-system emulator for Ethernet-SSDs, which extends FEMU with a faithful functional emulation, together with a parametric latency model that captures fine-grained, component-level timing. Our evaluation shows that EEMU accurately reproduces latency trends and enables systematic exploration of architectural trade-offs and workload impacts. We open-source EEMU to promote reproducibility and future research: <https://github.com/xkjiang-srfv/EEMU>.

REFERENCES

- [1] Samsung nvme-of ethernet-ssd. https://www.snia.org/sites/default/files/SDCIIndia/2019/PDF/2%20-%20Samsung%20NVMe-oF_Ethernet-SSD_final.pdf.
- [2] Samsung nvme-of ethernet-ssd test white paper. <https://www.odcc.org.cn/download/p-1437655581737832449.html>.
- [3] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for {SSD} performance. In *2008 USENIX Annual Technical Conference (USENIX ATC 08)*, 2008.
- [4] Aayush Gupta, Youngjae Kim, and Bhuvan Urganekar. Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. *Acm Sigplan Notices*, 44(3):229–240, 2009.
- [5] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the international conference on Supercomputing*, pages 96–107, 2011.
- [6] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Björling, and Haryadi S Gunawi. The {CASE} of {FEMU}: Cheap, accurate, scalable and extensible flash emulator. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, pages 83–90, 2018.