

Extending NVMeVirt for Edge Storage Research

Hongseung Yu
Seoul National University
hongseung@snu.ac.kr

Hyunah Kim
Seoul National University
khyunah03@snu.ac.kr

Minsung Kim
Seoul National University
minsungkim.cs@snu.ac.kr

Abstract

NVMe SSDs are increasingly deployed on edge and embedded platforms. NVMeVirt is a software-defined emulator that realizes a virtual NVMe SSD entirely in software, removing the dependence on physical hardware that bottlenecks the development and evaluation of new storage designs. NVMeVirt, however, cannot faithfully emulate the SSDs used at the edge, because it was built around x86 hosts and server-class SSDs. SSDs deployed in edge environments are typically DRAM-less—for form-factor, power, and cost efficiency—a design that differs fundamentally from the DRAM-based SSDs used in server environments. In this paper, we present *Edge-NVMeVirt*, an extension of NVMeVirt that brings it to the edge. To emulate edge DRAM-less SSDs, we first port NVMeVirt to ARM, and then model the DRAM-less translation path—specifically the hierarchical L2P cache it depends on, including the Host Memory Buffer. We evaluate Edge-NVMeVirt on a Jetson Orin Nano against a real DRAM-less SSD, and it reproduces the device’s characteristic random-read behavior. Beyond precise emulation, it provides the configurable parameters that are usually fixed in hardware, giving a more flexible platform for edge storage research.

1 Introduction

Non-Volatile Memory Express (NVMe) [10] is a PCIe-based host controller interface for non-volatile storage. Owing to its low software overhead and deeply parallel queue model, NVMe has become the de facto standard for solid-state storage across modern computing platforms. Although originally developed for datacenter servers, it is now widely deployed on edge and embedded platforms as well.

NVMeVirt [7] is a software-defined NVMe device, implemented as a Linux kernel module, that exposes a virtual PCIe NVMe controller whose internal parameters—FTL logic, NAND timing, and channel configuration—are directly controllable at the source level. This makes it a valuable platform for prototyping and reproducibly evaluating emerging SSD designs. However, NVMeVirt in its current form is tailored to server-class environments and does not reflect the realities of edge deployment, which depart from the server setting on two fronts—one on the *device side* and the other on the *host side*.

Device side: DRAM-less SSDs. Every NVMe I/O must consult the logical-to-physical (L2P) mapping table, and conventional SSDs keep this table in on-board DRAM so that translation cost stays negligible compared to NAND access. At the edge, however, form-factor, power, and cost constraints push

controller vendors toward *DRAM-less* designs, which have become widespread in such deployments [11, 12]. Without on-board DRAM, a controller can cache only a small fraction of the mapping table in its internal SRAM, and on a miss must fetch the missing entries from NAND. The NVMe *Host Memory Buffer (HMB)* feature mitigates this by lending the device a region of host DRAM as an extended cache. HMB narrows the gap but does not close it [5]: performance still varies strongly with the mapping-table size (which scales with capacity), the amount of host DRAM allocated as HMB, and the workload’s access pattern. NVMeVirt captures none of this, as it assumes the L2P table resides entirely in on-device DRAM and can be consulted in constant time, omitting translation cost from its latency model.

Host side: ARM-based SoCs. Edge and embedded platforms operate under strict power and thermal budgets and are typically built around ARM-based systems-on-chip (SoCs), so any emulation infrastructure meant to support storage research in this setting must be able to run on ARM hosts. NVMeVirt, however, reserves the backing store of its virtual NVMe device in a region of physical memory set aside at boot through the x86-specific `memmap=` kernel parameter. ARM platforms provide no such mechanism, and running NVMeVirt on them requires substantial redesign of how the emulator reserves memory and interfaces with the underlying platform.

We extend NVMeVirt into *Edge-NVMeVirt* along these two axes: porting the emulator to ARM hosts (§3.1), and modeling HMB-based L2P translation so that DRAM-less translation cost becomes part of the modeled I/O latency (§3.3). We evaluate it on a Jetson Orin Nano against the FORESEE XP1000, a 128 GB DRAM-less SSD. This paper makes the following contributions:

- We identify two characteristics a server-oriented emulator must address to model edge storage: DRAM-less SSDs on the device side and ARM-based SoCs on the host side.
- We extend NVMeVirt to address both—porting it to ARM hosts and interposing a multi-tier SRAM/HMB translation cache that charges per-tier latency—while leaving its core emulation logic intact, and validate its fidelity against a real DRAM-less SSD.
- We demonstrate Edge-NVMeVirt’s utility by exploring parameters the physical device fixes in hardware, showing its value as a platform for future edge storage research.

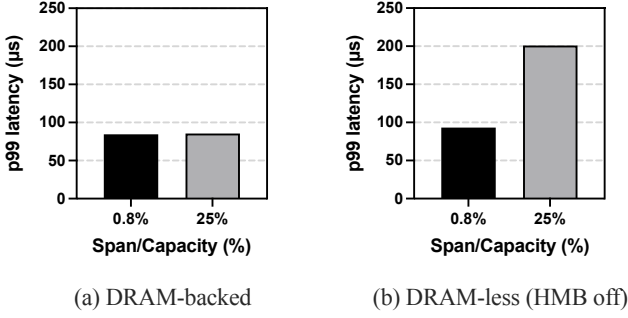


Figure 1. p99 random-read latency for (a) a DRAM-backed SSD (Samsung 950 PRO, 512 GB) and (b) a DRAM-less SSD with HMB turned off (FORESEE XP1000, 128 GB), each measured at two spans set to equal fractions of device capacity (0.8% and 25%).

2 Background and Related Work

2.1 NVMeVirt

NVMeVirt [7] is a software-defined SSD emulator implemented as a Linux kernel module. Unlike trace-driven simulators or VM-based emulators, it presents itself to the unmodified host NVMe driver as a real PCIe NVMe controller: the driver discovers it, issues admin and I/O commands, and creates namespaces exactly as it would for physical hardware. NVMeVirt reserves a region of host DRAM as the backing store of the virtual device, services the NVMe command set in kernel worker threads, and applies a configurable latency model that accounts for FTL logic, NAND timing, and channel/way parallelism. Because these parameters are controllable at the source level, a single code base can reproduce a range of SSD designs, which is what makes NVMeVirt useful for storage-stack research.

2.2 Host Memory Buffer for DRAM-less SSDs

Before discussing HMB, we first describe how a conventional DRAM-backed SSD and a DRAM-less SSD differ in handling the L2P mapping table.

Conventional DRAM-backed SSD. Every NVMe I/O must consult the L2P mapping table to translate a logical block address into a physical NAND location. Conventional SSDs keep the full table in on-board DRAM. With a 4 KB page and a 4 B entry, the table is roughly 1/1024 of the device capacity (e.g., 128 MB for a 128 GB device), and holding it in DRAM keeps translation cost negligible compared to NAND access. **DRAM-less SSD.** That DRAM, however, adds cost and power and consumes die and board area—the budgets tightest at the edge. DRAM-less SSDs therefore omit it entirely, so the controller can cache only a small fraction of the L2P table in its internal SRAM; on a miss, the missing entries must be fetched from NAND, inflating the latency of the I/O that triggered the miss. Figure 1 isolates this effect. The DRAM-backed

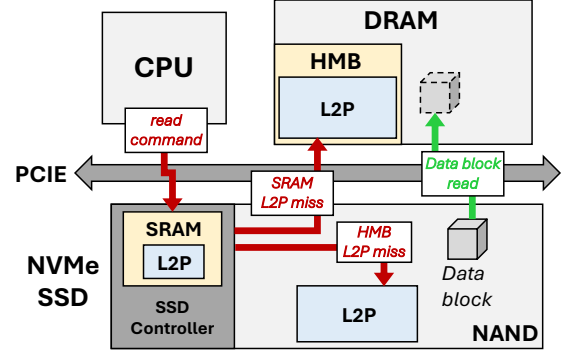


Figure 2. An illustration of data read in DRAM-less SSD using HMB. For each read, the controller’s FTL looks up the logical-to-physical (L2P) mapping across three tiers in order—the on-chip SRAM cache, the HMB-backed cache, and the L2P table in NAND. A hit at any tier resolves the physical address and proceeds to read the data block from NAND.

device stays flat across both spans, whereas the DRAM-less device is comparable at the small span but rises sharply as the span grows and the working set exceeds the SRAM cache coverage.

Host Memory Buffer (HMB). To mitigate this penalty, the NVMe HMB feature lets the device borrow a region of host DRAM—typically tens of megabytes—and use it as an extended L2P cache. Figure 2 illustrates the resulting translation path for a read. The region is accessed and managed by the device itself over PCIe DMA, so the host CPU only issues the command and reaps the completion while the controller’s FTL performs every lookup. On each read, the FTL resolves the mapping through three tiers consulted in order: it first checks the controller’s internal SRAM; on a miss it consults the HMB-backed cache in host DRAM over PCIe; and only on a miss in both does it fetch the L2P entry from NAND. Once the physical address is resolved at any tier, the device reads the data block from NAND and returns it to the host. The three tiers differ in latency—an SRAM hit is fastest, an HMB hit requires a PCIe round trip to host DRAM, and a miss in both must first fetch the mapping from NAND—so a DRAM-less SSD’s performance depends largely on how often translations are served from SRAM or HMB rather than falling through to NAND.

2.3 Related Work

Modeling DRAM-less SSDs and HMB. Kim et al. [5] empirically characterize how commercial DRAM-less NVMe SSDs use HMB, showing through reverse-engineering experiments that HMB serves predominantly as a cache for the L2P mapping table and that the resulting performance depends strongly on the allocated HMB size, the mapping-table size, and the workload’s access pattern. Their study

establishes the observations that motivate this work, but it remains a measurement-based analysis of existing devices and does not expose HMB behavior as a parameter that can be varied in controlled studies. By contrast, HMB-SSD [6] extends an open-source SSD emulator to support the HMB feature and demonstrates its benefit, but it targets an older simulation framework and focuses on the write-buffer use of HMB rather than on the L2P translation behavior that dominates DRAM-less SSD performance in practice.

Host Performance Booster (HPB). A related line of work pushes mapping-cache responsibility onto the host itself. HPB [4] caches L2P entries in host memory and piggybacks physical page numbers onto read commands, achieving significant random-read improvements; it has since been standardized as part of UFS 3.1. The key distinction from HMB is the management entity: HPB is host-managed (the host explicitly pushes L2P entries), whereas HMB is device-managed. HP-UFS [9] extends the FEMU emulator [8] to model UFS and implements HPB-style host-device cooperation on top of it. These approaches, however, rely on a dedicated command path and matching host-side driver support, and are defined within the UFS ecosystem rather than as NVMe features; importing such a scheme into a device-side NVMe emulator like NVMeVirt would require intrusive changes to both the emulator and the host NVMe driver, constituting a separate line of research. In contrast, our work focuses on faithfully reproducing the behavior of standard HMB-backed DRAM-less NVMe SSDs within NVMeVirt, without altering the host-device interface.

3 Edge-NVMeVirt

3.1 Porting NVMeVirt for ARM.

NVMeVirt’s core emulation logic—the virtual PCIe NVMe controller, the FTLs, and the latency model—is architecture-independent, but its initialization path is strictly tied to x86. We organize the port around the NVMeVirt bring-up sequence the host NVMe driver triggers: (1) reserving and validating the device’s backing memory, (2) constructing the synthetic PCIe function and its interrupt path, and (3) exposing the controller registers with the correct memory attributes. Three changes are required along this path; the core logic is left unchanged, so the ported build behaves identically to its x86 counterpart from the perspective of the guest NVMe driver and any FTL on top.

Reserved backing memory. NVMeVirt places both the virtual device’s block data and its register/doorbell/MSI-X¹ backing in a region of host DRAM that must be withheld from the kernel’s general-purpose allocator. On x86, it reserves this region through the `memmap=` kernel parameter and validates it against the E820 memory map. However, ARM provides neither mechanism; the boot-time memory

layout is described by the Device Tree (DT) instead. We therefore add a reserved-memory node with the `no-map` attribute to the device tree—patched into the device tree blob (DTB), which is passed by the bootloader—so the kernel excludes the range from System RAM and guarantees its integrity at runtime.

Synthetic PCIe/MSI delivery. NVMeVirt is not a real PCIe device; it constructs a synthetic PCI function in software so the host NVMe driver can bind to it. On x86, the completion and interrupt delivery relies on the APIC-based interrupt path, which ARM lacks, and a synthetic root bus does not inherit a platform MSI controller—so advertising MSI-X alone does not produce the Linux IRQs that the host driver needs. For ARM, we build a software parent IRQ domain and a PCI-MSI child domain, attach them to a synthetic root bus and deliver each completion by resolving its MSI-X table index to a Linux virtual IRQ and invoking the host handler².

Register memory attributes. The host NVMe driver maps the controller’s Base Address Register (BAR) region as memory-mapped I/O (Device memory) and accesses registers and doorbells through it. If NVMeVirt internally touched the same physical addresses as normal cacheable memory, ARM would see one physical address under two conflicting memory attributes—a hazard that does not clearly manifest on x86 but can reorder or corrupt register state under the ARM memory model. We therefore treat the BAR, the doorbells, and the MSI-X table strictly as MMIO-mapped, while keeping the SSD data-backing region as normal write-back memory³. The two regions are thus separated by memory attribute even though they share the same reserved physical range.

3.2 Performance modeling

Before extending NVMeVirt with an HMB model, we must provide its address-translation path with a timing model: a latency that the emulator charges for every L2P lookup. We anchor this model to a real DRAM-less SSD that serves as our reference device⁴. Two of the internals the model needs—the effective SRAM cache size and the per-tier lookup latencies—are not given in the device specification, so we estimate them empirically from the device’s measured random-read behavior.

Estimating SRAM capacity. The SRAM is the fastest translation tier, holding only a small fraction of the mapping table; its effective size is therefore a first-order parameter of the model. To infer it, we measure 4 KB random-read latency with HMB disabled while gradually growing the read span, so that any latency rise must come from translation alone. As shown in Figure 3, p99 latency stays nearly flat up to a span

¹MSI-X is an abbreviation of Message Signaled Interrupts - eXtended, which is a standard function to handle interrupts in PCIe.

²We implemented such functionalities through Linux kernel standard APIs, such as `pci_create_root_bus()`, `pci_scan_child_bus()`, `pci_irq_vector()`, `generic_handle_irq()`.

³MMIO-mapped regions are created and handled through `ioremap()` and `memset_io()`, both are Linux kernel standard APIs.

⁴FORESEE XP1000F128G, a 128 GB DRAM-less PCIe Gen3×4 NVMe SSD.

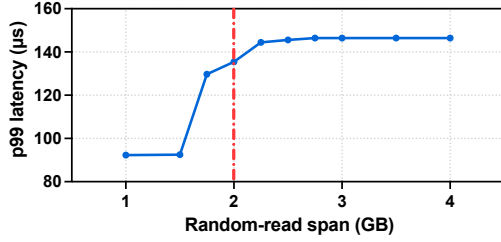


Figure 3. p99 4 KB random-read latency of the FORESEE XP1000 (128 GB) with HMB disabled as a function of working-set span.

of roughly 2 GB and then rises sharply. We read this knee as the point where the mapping entries for the accessed span no longer fit in SRAM and mapping pages must be fetched from NAND. Under a page-level mapping with 4-byte entries, covering a 2 GB span requires a mapping table about 1/1024 of that size, implying an effective translation SRAM of roughly 2 MB.

Lookup latency calibration. The lookup latency parameters are calibrated using a combination of hardware characteristics and measurements from the reference SSD. Since the latency of an SRAM hit is negligible compared to the overall read latency, which is on the order of tens of microseconds, we model SRAM accesses with a small but non-zero delay of 100 ns. For HMB hits, the controller must access host DRAM through PCIe. As the reference SSD is an M.2 2280 device directly attached to the motherboard, propagation delay is negligible and the access latency is dominated by PCIe transaction overhead. A PCIe Gen3 transaction involves TLP generation, root complex arbitration, DDR4 access on the host side, and the return path, each contributing on the order of hundreds of nanoseconds. Consequently, the round-trip latency falls in the range of 1–2 μ s, and we use 2 μ s as the HMB access latency. For NAND-resident mapping pages, we derive the lookup latency directly from measurements on the reference SSD. The sharp latency increase observed in Figure 3 can be interpreted as the additional time required to fetch a missing translation entry from NAND. The measured increment is approximately 54 μ s, which is adopted as the NAND lookup latency in our model.

3.3 Modeling HMB

NVMeVirt treats L2P translation as instantaneous, directly accessing the in-memory mapping table—reasonable for DRAM-backed SSDs but not for DRAM-less ones, whose mapping tables reside hierarchically so that translation latency depends on where the entry is found and joins the critical path of every I/O. To emulate this, we extend NVMeVirt with an explicit multi-tier translation cache, leaving the underlying FTL unchanged and using the timing model from §3.2.

Translation-cache architecture. The translation cache consists of two levels corresponding to controller SRAM and HMB. Each level maintains metadata for lookup and replacement management, and its capacity and access latency are configurable through module parameters. The cache operates at translation page granularity rather than at the level of individual LPNs: entries are indexed by a page key derived by grouping 1,024 consecutive LPNs, so that each cache entry covers 4 MB of logical address space. This matches the granularity at which commercial controllers fetch mapping data from NAND [3]. Consistent with commercial SSD controllers, the cache hierarchy is shared across all FTL partitions rather than being replicated per channel or die.

Translation lookup procedure. Lookups proceed in order from the fastest to the slowest tier. The SRAM level is checked first; on a hit, the SRAM access latency is charged and the lookup completes. On an SRAM miss, the HMB level is consulted; on a hit, the HMB access latency is charged and the entry is promoted to SRAM so that subsequent accesses to the same translation page are served at the lower-latency tier. If neither level contains the entry, the mapping page must be fetched from NAND, incurring the full NAND access latency, after which the entry is inserted into the HMB tier. When the HMB capacity is set to zero, lookups bypass the HMB level and proceed directly from SRAM to NAND, degenerating to the conventional SRAM–NAND organization and preserving the original NVMeVirt behavior in HMB-disabled configurations.

Timing model integration. Throughout this process, the original mapping table remains the authoritative source of physical page addresses. The cache hierarchy determines only the latency charged to each lookup, not the address returned, and mapping updates are propagated to the cache so that subsequent accesses observe the correct translation level. Simply adding translation latency to a request’s completion time would misplace when NAND operations begin and distort NVMeVirt’s parallel execution model. Instead, we incorporate it before NAND commands are issued: a local timestamp is advanced by the translation delay, and NAND operations are scheduled from the updated timestamp. NAND accesses thus begin only after translation completes, making translation latency part of the critical path while preserving the original parallel execution model.

4 Evaluation

We evaluate Edge-NVMeVirt along two axes: whether it faithfully reproduces the qualitative behavior of a real DRAM-less SSD (fidelity, §4.2), and whether it enables exploration of the edge-storage design space that the physical device cannot support (utility, §4.3).

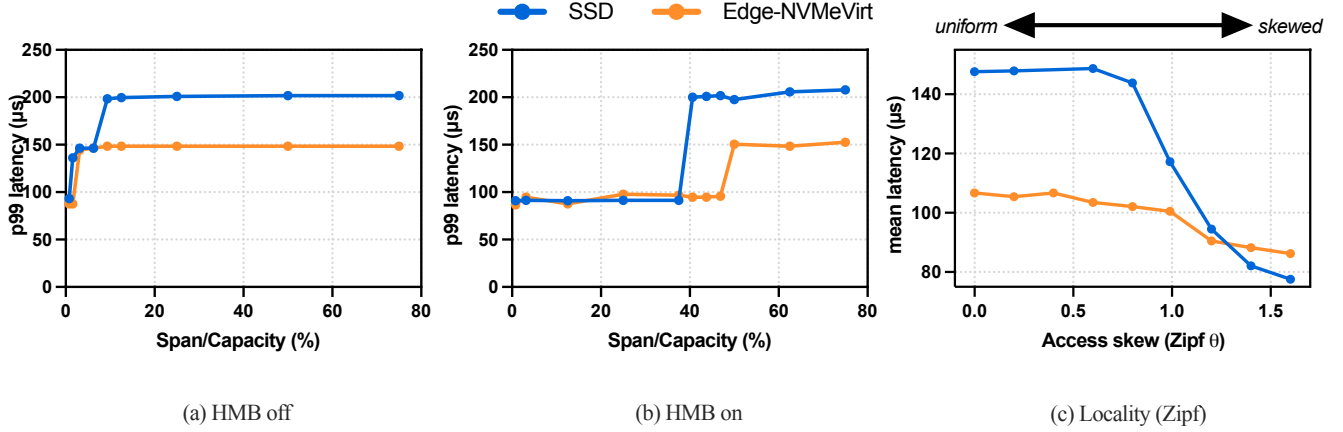


Figure 4. Fidelity of Edge-NVMeVirt against the target device. (a, b) p99 4 KB random-read latency versus working-set span, with HMB disabled (a) and enabled (b). (c) Mean latency under a Zipf-skewed access pattern at a fixed 75% span with HMB enabled; $\theta = 0$ is uniform and larger θ concentrates accesses onto a smaller working set.

4.1 Experimental Setup

Physical reference device. Our emulation target device is the FORESEE XP1000, the 128 GB DRAM-less SSD characterized in §3.2, whose controller exposes a 64 MB HMB that can be toggled on or off but not resized. This yields two well-defined operating points—HMB on and HMB off—against which we anchor our emulator.

Scaled emulator configuration. We run Edge-NVMeVirt on a Jetson Orin Nano, whose 16 GB of DRAM cannot model the 128 GB target device directly. We therefore scale all capacities down by $1/64$, yielding a 2 GB virtual SSD with a 1 MB HMB and a 32 KB SRAM L2P cache. The scaling preserves the 4 KB-page / 4 B-entry layout (the $1/1024$ rule), and with it the device’s $\approx 50\%$ HMB-to-L2P coverage ratio—the quantity that governs the behavior we set out to reproduce.

Normalized comparison axis. Because the two platforms differ in absolute capacity, we plot every result against the random-read span expressed as a fraction of device capacity, so a given point denotes proportional working sets on both. For instance, 25% is a 32 GB span on the reference device and a 512 MB span on our emulator.

Benchmark methodology. All workloads are generated with fio [1] using 4 KB requests and `direct=1` to bypass the page cache, at a fixed queue depth of one. Each span is preconditioned with a sequential write; every run is a 30 s time-based measurement with a 5 s ramp to discard the warm-up transient, and we report the median of five runs. We use p99 latency for the span sweeps, where misses surface only in the tail, and mean latency for the locality sweep, where the hit rate varies over a wide range.

4.2 Fidelity: Comparison with a Physical SSD

We evaluate the emulation quality of Edge-NVMeVirt against the target device with two experiments: reproducing the HMB coverage cliff and the response to access locality.

HMB off. We sweep the random-read span with HMB disabled on both the device and our emulator (Figure 4a). Once the span exceeds the SRAM-resident portion of the L2P table, translations begin to miss to NAND and p99 latency steps up; both curves show the transition at the same normalized span. Since the SRAM cliff location was calibrated from this same HMB-off sweep (§3.2), the match confirms that Edge-NVMeVirt reproduces the calibrated regime.

HMB on. With HMB enabled (Figure 4b), SRAM misses are absorbed by the HMB-backed cache without touching NAND, so p99 latency stays flat well past the SRAM cliff and rises only once the span outgrows the HMB coverage and translations again fall through to NAND. Both platforms reproduce this delayed cliff. Since the HMB tier operates here on its independently calibrated latency, this match is a prediction rather than a fit.

Quantitative differences. Edge-NVMeVirt captures the qualitative behavior of the target device—the location of each cliff and the flat regions on either side. However, two quantitative differences remain. First, the post-cliff latency is higher on the real device than on our emulator, under both HMB settings. Second, the HMB-on cliff appears earlier on the device than on our emulator. We analyze both in §5 and discuss what they reveal about the gap between our model and a commercial controller.

Sensitivity to access locality. To verify the cache replacement logic, we fix the span at 75% of capacity with HMB enabled and sweep the Zipf skew θ from 0 (uniform) to 1.6, recording mean latency (Figure 4c). At low skew the working set still exceeds the cache, so hot entries are evicted before

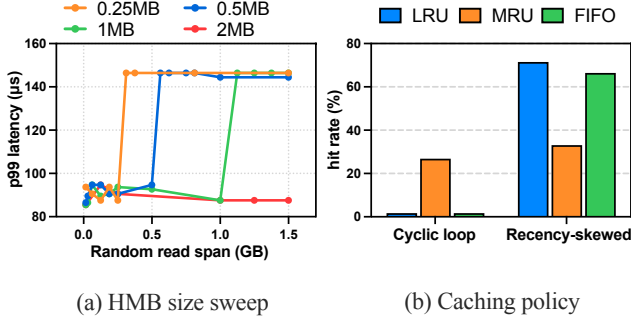


Figure 5. Design-space exploration on Edge-NVMeVirt. (a) p99 4 KB random-read latency versus span for emulator HMB sizes of 0.25–2 MB. (b) Hit rate of the LRU, MRU, and FIFO replacement policies under a cyclic sequential loop and a recency-skewed (Zipf $\theta = 0.99$) workload, at a 2 GB span with a 0.5 MB HMB.

reuse and mean latency stays flat on both platforms. As θ approaches 1.0 the accesses concentrate enough for LRU to retain the hot entries across reuses, and mean latency falls, with our emulator tracking the device closely. As in the span sweeps, only the absolute offset differs (§5).

4.3 Utility: Exploring the Edge-Storage Design Space

Beyond fidelity, we now use Edge-NVMeVirt to vary parameters that the physical device fixes in hardware, demonstrating its value as a platform for future edge-storage research.

HMB size sweep. On the real device the HMB is exposed only as a fixed-size binary switch—precisely the constraint that motivates emulator-level support. Holding the workload fixed, we sweep our emulator’s HMB size across 0.25, 0.5, 1 (the calibrated baseline), and 2 MB (Figure 5a). The coverage cliff moves in proportion to HMB size, while the 2 MB configuration shows no cliff within the swept range, as its coverage spans the whole device. This proportional shift is exactly what the 4B-entry / 4 KB-coverage model predicts.

Caching policy. Beyond LRU, we implement MRU and FIFO replacement and compare their hit rates under two workloads—a cyclic sequential loop and a recency-skewed (Zipf $\theta = 0.99$) pattern (Figure 5b). The span is 2 GB and the HMB is set to 0.5 MB so that misses are frequent and the policy effect is isolated. For the cyclic loop, MRU attains the highest hit rate, as expected since a looping scan larger than the cache is the canonical case in which MRU outperforms LRU [2]; LRU and FIFO trail it. Under the recency-skewed workload the ordering reverses—LRU is highest, then FIFO, then MRU. This shows that Edge-NVMeVirt can expose how the workload–policy interactions characteristic of edge and embedded systems bear on future storage and FTL design.

5 Discussion: Toward Higher-Fidelity Modeling

While Edge-NVMeVirt reproduces the qualitative behavior of the reference device, two quantitative gaps remain (§4.2). We discuss the cause of each and what additional information would be needed to close it.

Latency cost of an L2P miss. We calibrated the NAND lookup penalty from the single latency step in Figure 3. The full HMB-off sweep, however, reveals two steps rather than one (Figure 4a): p99 latency rises at 1.56% of capacity (a 2 GB span), from roughly 92 to 146 μ s, holds, and then rises again by a comparable amount at about 12.5% (a 16 GB span), to roughly 200 μ s. The two increments are nearly equal, and the upper level persists even with HMB enabled. A *two-level mapping table* offers the most plausible explanation. If a translation is resolved through a directory level and then a page-level table [3], a single miss—the page-level entry absent while its directory entry is still cached—costs one NAND access, whereas a span large enough to evict both levels costs two. Since each NAND access contributes the same fixed penalty, the second step matches the first in magnitude, reproducing the observed structure. Our model assumes a single-level lookup, as the controller’s internal translation scheme is unknown; vendor-level details on this mechanism would allow the model to capture the multi-level miss cost directly.

Premature HMB cliff. Our cache model assumes the entire HMB region holds L2P entries. At 4 B per 4 KB page, a 64 MB HMB then covers a 64 GB span—50% of the 128 GB capacity, matching the coverage ratio our scaled configuration is built around (§4.2). On the real device, however, the HMB-on cliff appears earlier, at roughly 37.5% of capacity (a 48 GB span). This shortfall suggests that the controller reserves part of the HMB for its own firmware state and data, leaving only about 48 MB—three-quarters of the 64 MB—for address translation, enough to cover the 48 GB span at which the cliff is observed. Accounting for the controller’s actual HMB usage would require knowledge of its internal memory layout, which, like its translation scheme, the device does not expose.

6 Conclusion

We presented Edge-NVMeVirt, an extension of NVMeVirt that ports the emulator to ARM hosts and models HMB-based L2P translation, making it the first software-defined NVMe emulator that captures DRAM-less SSD behavior on the edge platforms where such devices are deployed. On a Jetson Orin Nano, Edge-NVMeVirt reproduces the characteristic behavior of a commercial DRAM-less SSD such as the HMB coverage cliff and its response to access locality. By exposing as configurable parameters the device characteristics that real hardware fixes, Edge-NVMeVirt provides a foundation for storage research on the edge platforms that have so far lacked one.

References

- [1] Jens Axboe. 2022. Flexible I/O Tester (fio). <https://github.com/axboe/fio>.
- [2] Hong-Tai Chou and David J. DeWitt. 1985. An Evaluation of Buffer Management Strategies for Relational Database Systems. In *11th International Conference on Very Large Data Bases (VLDB)*.
- [3] Aayush Gupta, Youngjae Kim, and Bhuvan Ugaonkar. 2009. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.
- [4] Wookhan Jeong, Yongmyung Lee, Hyunsoo Cho, Jaegyu Lee, Songho Yoon, Jooyoung Hwang, and Donggi Lee. 2017. Improving flash storage performance by caching address mapping table in host memory. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*.
- [5] Kyusik Kim and Taeseok Kim. 2020. HMB in DRAM-less NVMe SSDs: Their usage and effects on performance. *PloS one* 15, 3 (2020), e0229645.
- [6] Kyusik Kim, Eunji Lee, and Taeseok Kim. 2019. HMB-SSD: Framework for efficient exploiting of the host memory buffer in the NVMe SSD. *IEEE Access* 7 (2019), 150403–150411.
- [7] Sanghoon Kim, Jaehoon Shim, Euidong Lee, Seongyeop Jeong, Ilkueon Kang, and Jinsoo Kim. 2023. NVMeVirt: A versatile software-defined virtual NVMe device. In *21st USENIX Conference on File and Storage Technologies (FAST)*.
- [8] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Björling, and Haryadi S. Gunawi. 2018. The CASE of FEMU: Cheap, Accurate, Scalable and Extensible Flash Emulator. In *16th USENIX Conference on File and Storage Technologies (FAST)*.
- [9] Jaesun No, Gyun Lee, Youngsok Kim, and Jinkyu Jeong. 2024. Fully harnessing the performance potential of DRAM-less mobile flash storage. In *38th International Conference on Massive Storage Systems and Technology (MSST)*.
- [10] NVM Express, Inc. 2025. NVM Express Base Specification. <https://nvmexpress.org/specifications/>.
- [11] Phison Electronics. 2025. DRAM or Not? The Difference Between DRAM and DRAM-less SSDs (and Why It Matters). <https://phisonblog.com/dram-or-not-the-difference-between-dram-and-dram-less-ssds-and-why-it-matters/>.
- [12] SSTC. 2025. SSD Form Factor Trends: M.2 SSD vs. EDSFF Hot-Swap. <https://www.sstc.com/knowledge-detail/ssd-m2-edsff/>.