



Networking Track

Modernizing NFS Direct I/O for PCI Peer-to-Peer DMA

Pranjal Shrivastava (Google) Shivaji Kant (Google)

South Hall 1 A, Prague Congress Centre



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Direct I/O Beyond Host DRAM

Non-Host Memory in Userspace

Drivers increasingly expose different types of non-host (ZONE_DEVICE) memory to userspace for Direct I/O:

- **NVMe:** Exports Controller Memory Buffers (via `/sys/bus/pci/<BDF>/p2pmem/allocate`).
- **Accelerators (habanalabs):** P2PMEM publisher (`pci_p2pdma_add_resource()`) via IOCTLs
- **DAX Memory (fsdax / devdax — PMEM, CXL):** Byte-addressable persistent & CXL memory mapped as ZONE_DEVICE pages (`MEMORY_DEVICE_FS_DAX / DEV_DAX`).
- **VFIO ZONE_DEVICE Exporter (ongoing RFC series [\[1\]](#))**
 - Exports PCIe EP BARs to userspace as `MEMORY_DEVICE_PCI_P2PDMA` pages for P2PDMA



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



ZONE_DEVICE on NFS Today: DAX Works, PCIe BARs Don't

What Works Today

Pinning ZONE_DEVICE pages already works for these.

- **DAX's ZONE_DEVICE Memory (CXL / PMEM)**
Pinning DAX's pages (`MEMORY_DEVICE_FS_DAX` / `DEV_DAX`) is allowed without P2P flags
- **PCIe BARs on Local NVMe**
The block layer checks P2P DMA capability [via `blk_queue_pci_p2pdma()`] and passes `ITER_ALLOW_P2PDMA` when extracting pages (block/bio.c).
- **NFS + DIRECT I/O works out of the box.**

What doesn't (PCIe BARs on NFS)

Pinning fails for PCIe EP memory (NVMe CMB or HBM BARs):

- **Rejected by GUP (-EREMOTEIO)**
GUP refuses to pin `MEMORY_DEVICE_PCI_P2PDMA` pages unless the caller explicitly passes `FOLL_PCI_P2PDMA` (mm/gup.c).
- **NFS is Pin-Unaware**
Uses `get_page()` references instead of retaining GUP pins across async RPCs.
- **Missing Capability Flags**
NFS relies on the legacy `iov_iter_get_pages_alloc2()` API which is flag-blind to begin with and can't even support handling P2P flags.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



PCIe Endpoints on NFS Today: The Bounce-Buffer Tax



Why the Host-DRAM Detour Hurts at Scale?

Because NFS O_DIRECT rejects PCIe BAR pages (-EREMOTEIO), every I/O must stage through host DRAM:

- **Bulky I/Os Force Fixed Bounce Pools (The Bottleneck)**
 - Reads can be **>10s of GBs** (e.g., AI workloads),
 - Apps cannot allocate "on-the-fly", must keep pre-allocated bounce regions to avoid host OOMs
 - Bottlenecks the entire data flow.
- **Added Latency:** Serializes distributed I/O behind an extra host bounce step before the endpoint can use the data.
- **2× Host Memory Bus Traffic:** Data crosses host DRAM twice (RNIC DMA write into DRAM, then to the PCIe BAR).



Motivating Example: LLM KV Cache Memory Bottlenecks

UNIVERSAL KV CACHE SIZE FORMULA *2 for Key & Value tensors | *2 bytes for FP16 / BF16

Size = $2 \times 2 \times \text{num_layers} \times \text{num_kv_heads} \times \text{head_dim} \times \text{seq_len} \times \text{batch_size}$

SINGLE-USER CACHE

~0.66 MB / Token

In Llama-3 70B (80 layers, 8 KV heads, dim 128), a 4,096-token prompt requires **2.68 GB** of KV cache for just ONE user.

At 16k context, a single request occupies **10.7 GB** of VRAM.

BATCH CONCURRENCY

85.8 GB / Step

With a standard batch size of 32 requests (4k context), the accelerator must load **85.8 GB of KV tensors** from HBM for every single output token!

Exceeds the total on-chip HBM of an 80GB GPU.

BANDWIDTH STARVATION

3.35 TB/s Saturated

Decoding 40 tokens/sec across 32 requests demands reading **3,432 GB/s** of KV data, immediately hitting the absolute physical ceiling of HBM3 for some configurations.

Arithmetic intensity drops (FLOP/byte).



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Bridging the Gap: Direct PCIe P2PDMA for NFS Direct I/O



The Kernel Building Blocks Already Exist — Except in NFS

Multiple Linux subsystems already bypass host DRAM via PCIe Peer-to-Peer DMA:

- **Block Layer & NVMe:** Queues advertise `BLK_FEAT_PCI_P2PDMA` (`blk_queue_pci_p2pdma()`), passing `ITER_ALLOW_P2PDMA` so GUP pins BAR pages into `struct bio` and `pci_p2pdma_state()` validates DMA mapping.
- **RDMA & DMA Core:** Checks NIC capability via `ib_dma_pci_p2p_dma_supported()` and maps peer BAR scatter-lists directly in `ib_dma_map_sg()` / `dma_direct_map_sg()` (used by NVMe-oF).
- **Our Proposal for NFS:** Plumb PCIe P2PDMA awareness through `fs/nfs/direct.c` and `xprtrdma`
 - NICs DMA directly to and from PCIe endpoint BARs, bypassing host DRAM completely.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Modernizing NFS Direct I/O: A Two-Phase Approach

Phase I: Pin-Aware Direct I/O (Queued for 7.4)

- **Pin-Aware struct `nfs_page`**
 - Add `PG_PINNED` & `wb_nr_pinned`
 - Replace `get_page()` / `put_page()` with `(un)pin_user_pages()`
- **Flag-Aware Iterator API**
 - Drop `iov_iter_get_pages_alloc2()` (flag-blind)
 - Use `iov_iter_extract_pages()` (GUP flags)
- **Unified Direct I/O Extraction**
 - Centralize read/write in `nfs_direct_extract_pages()`

Phase II: P2PDMA Enablement (Discussion)

- **Lookahead Capability Detection**
 - `ITER_ALLOW_P2PDMA` must be set upfront at extraction time
 - NFS rides on RPC: Transport visibility is required before extracting pages
- **Decoupled from the Underlying DMA Device**
 - Underlying transport can reconnect, trunk, or change
- **Gatekeeping During RPC Transmit**
 - Must guard against invalid P2P routes



Phase II Proposed Design: Per-RPC Gatekeeping

No Mount-Time Capability Bit: Decide Per RPC, Where the Data Moves

Pass the Flag: `ITER_ALLOW_P2PDMA` on RDMA mounts without pNFS.
Other mounts still get `-EREMOTEIO` from GUP.

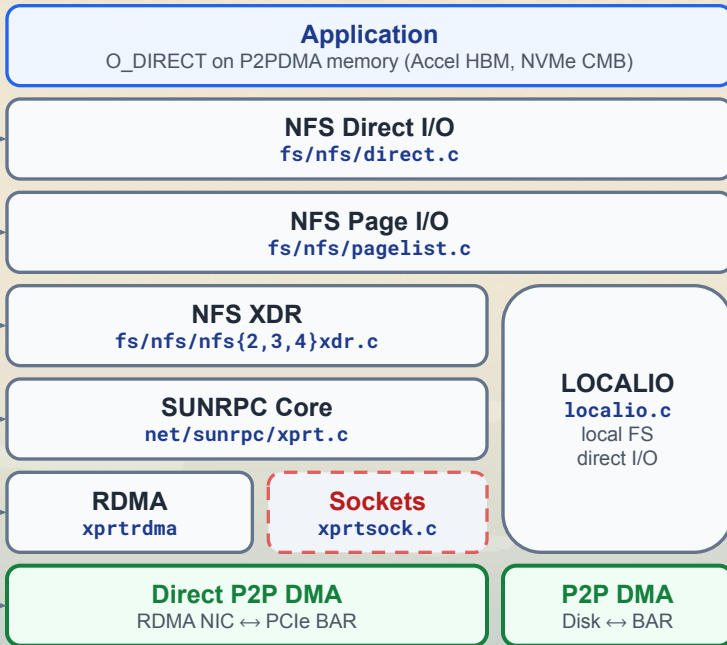
Mark the Request: if it holds P2PDMA pages. Refuse `krb5i` and `krb5p`.
Use LOCALIO only if the local disk does P2PDMA.

Tag the Payload: READ and WRITE encoders set `XDRBUF_P2PDMA`.
`READ_PLUS` is never used.

Fail Only That RPC: when a transport refuses it, with `-EREMOTEIO`. The
rest of the queue keeps moving.

Decide on the Device: `xprt_rdma` uses Read/Write chunks only, and only
if the NIC supports P2PDMA. Sockets always refuse.

Move Only by DMA: the NIC, or the local disk with LOCALIO, reaches the
BAR directly. No host DRAM bounce.



RFC: <https://lore.kernel.org/all/20261006233248.705086-1-praan@google.com/>



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Phase II Plumber's Discussion: Caveats & Open Questions

ZONE_DEVICE Memory May Not Be CPU-Safe, So P2PDMA Payloads Move Only by DMA

Trade-offs: Refuse Access

Refused Operations:

- `krb5i` / `krb5p` fail with **-EREMOTEIO**
- `READ_PLUS` is never used
- Small I/O always uses chunks, never inline

LOCALIO Handling:

- Aligned direct I/O requires `BLK_FEAT_PCI_P2PDMA`
- Misaligned I/O fails with **-EINVAL**

Open Questions

- **Layering:** "RDMA transport, IpNFS" is just a hint for `ITER_ALLOW_P2PDMA`; Each RPC re-checks the `xprt` / NIC. **Should the `xprt` advertise P2P capability up front?**
- **nconnect:** Fail the RPC on a transport without P2PDMA, or pick a capable one?
- **Migration / reconnect:** The path can move to a NIC without P2PDMA, and I/O that worked starts failing with **-EREMOTEIO**. Acceptable?
- **Hard mounts:** Refused I/O fails at once instead of retrying. Acceptable?

Future Topic / Work

pNFS Integration:

Opt in per layout driver, starting with flexfiles & SCSI layouts over NVMe-oF/RDMA.

Server-Side P2PDMA:

- Bridge `nfsd` / `svcrdma` directly to NVMe P2P targets (bypassing DRAM).
- **ZONE_DEVICE Memory:** How will local disks expose memory? What CMB sizes are required?



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Latency Performance with RFCv1

BENCHMARK HIGHLIGHT

~56 μ s

Lowest Average Latency

19% Faster

vs Host DRAM

33% Faster

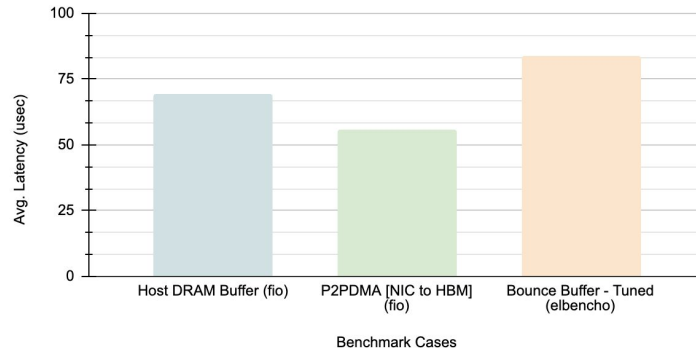
vs Bounce Buffer

P2PDMA delivers the lowest latency at **~56 μ s** (19% faster vs Host DRAM, 33% vs Bounce Buffer).

Average Latency (usec)

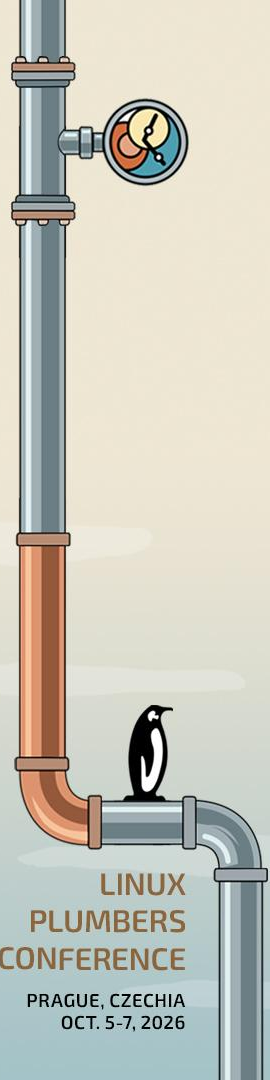
NFS P2PDMA Performance - Latency

Configs: bs=4K and iodepth=1



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

A vertical pipe with a pressure gauge and a penguin on a horizontal section. The pipe is grey with orange-brown sections. A pressure gauge is attached to the upper part, and a penguin is perched on a horizontal section near the bottom.

Thank You!

Questions & Discussion

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Backup: Why dmabuf + io_uring Doesn't Fly for NFS clients

- **Ongoing Upstream Work:** [\[PATCH v6 00/13\] Add dmabuf read/write via io_uring](#)
 - Registers a **dma-buf** to an **io_uring** instance for a specific file and uses it with **IORING_OP_{READ,WRITE}_FIXED**.
 - Creates a pre-mapped DMA context via a new file op (**init_dma_buf_io_ctx**) and passes it through the I/O stack via **ITER_DMABUF_MAP** (currently NVMe block devices only).
- **Why NFS is NOT compatible today**
 - **1 File = 1 Static DMA Device Assumption:** **init_dma_buf_io_ctx** calls **dma_buf_attach(dmabuf, dma_dev) + dma_buf_map_attachment()** once upfront for a single block device. NFS has no fixed **dma_dev** at registration time (**ib_device** lives behind **rpc_xprt** reconnects, **nconnect**, pNFS).
 - **Page-Based:** **struct nfs_page**, **xdr_buf->pages**, and **frwr_map()** (**ib_map_mr_sg()**) all operate on **struct page** arrays; page-less **dma_buf** scatterlists require new type of payload plumbing across **fs/nfs**, **sunrpc**, and **xprtrdma**.
 - **All Phase II Protocol Caveats Still Apply:** Still must guard against CPU-touching paths (**krb5i / krb5p** **RPCAUTH_AUTH_DATATOUCH**, **READ_PLUS** sparse-hole **memset**, and **rpcrdma_inline_pullup()**).

