

VFIO PCIe TPH Userspace API & Progressive Security Policy

Chengwen Feng (Huawei)



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Agenda

01 Background & Current Status
PCIe TPH and Linux kernel support overview

02 Motivation
Rationale for Userspace TPH API requirements

03 Isolation-Related Risk Model
Key isolation risks to consider

04 Community Core Concerns
Key review feedback from v1 to v20 iterations

05 Final API Design
Progressive security policy and implementation

06 Remaining Challenges & Roadmap
Open issues, future enhancements and plans



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Background & Current Status

PCIe TPH:

- TPH is a **requester-completer**-based mechanism. The Requester may carry **PH/ST** information towards the completer.
- Applicable for memory space transaction flows:
D2H(device-to-host), **D2D**(device-to-device), **H2D**(host-to-device) .

Linux kernel support overview:

Already:

- PCI TPH support landed in kernel 6.13.
- Broadcom and Mellanox Ethernet drivers implement TPH, targeting D2H flows.

Ongoing:

- Meta is working on upstream “vfio/dma-buf: add TPH support for peer-to-peer access”, targeting D2D flow.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Motivation: Why We Need This API

Performance Value:

- TPH technology **improves cache hit rates**, a critical optimization for high-throughput DPDK/SPDK user datapath workloads.

Current Limitation:

- VFIO exposes TPH capability enumeration but provides no standardized, safe userspace interface to program ST entries.

Core Goal: Empower userspace with secure, efficient access to TPH via VFIO, bridging hardware capabilities and software control for optimal datapath performance.



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Isolation-Related Risk Model

Risk Model:

- Assumption: PCIe device(s) is non-malicious.
- Treat the entire guest VM as an untrusted source.

Fact:

- The target (host or device) may perform cache optimization based on the ST value in the TLP.

Core Risk - Cross-tenant Cache Interference:

- A malicious guest could craft ST values to induce cache pollution across tenants, resulting in performance side-channel interference, not memory access violation.

Case1 – MSI-X Table ST

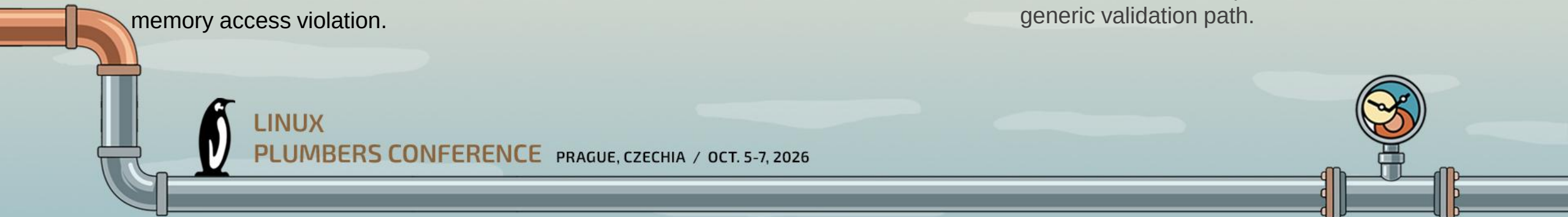
- Guests directly modify entries via un-trapped BAR MMIO writes.
- Hypervisor cannot intercept or validate these writes.

Case2 – Config-Space ST

- PCI config access is trapped. Hypervisor/VFIO can validate ST values map to CPU cores or DMA-BUF fd permitted for this guest.

Case3 – Device-specific ST

- Modified via device-specific driver interfaces; no generic validation path.



Community Core Concerns

Evolved from April RFC up to v20 in July. The final architecture converges through **four core community discussion pillars**:

01. Progressive Security Gate

- Evolve from no isolation control to **dual-layer opt-in security model**
- Global module parameter (tph_policy) for unsafe mode gating
- Per-device opt-in authorization for fine-grained isolation control

02. API Placement Convergence

- Abandon standalone sysfs/ioctl solution
- Iterate: sysfs -> VFIO -> hybrid model -> **fully VFIO-device-feature based UAPI**

03. API Design & Capability Coverage Refinement

- Replace custom independent ioctls with **standard VFIO_DEVICE_FEATURE paradigm**
- Shift from discrete query/enable calls to **virtualized register-based control**
- Extend capability coverage: **from D2H-only to D2H / D2D scenarios**
- Drop shadow ST table design per community consensus

04. Remaining Upstream Blocker: ST Target Lifecycle tracking

- Integrate with dma-buf TPH metadata framework
- Unresolved gap: **no invalidate_mappings lifecycle hook for TPH metadata.**



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Final API Design

Introduce three VFIO device-features:

Device Feature Name	Description
VFIO_DEVICE_FEATURE_TPH	[SET] Opt in TPH feature for the device [GET] Return supported TPH capability bits
VFIO_DEVICE_FEATURE_TPH_RESOLVE	[GET only] Resolve TPH attributes (PH/ST) from given source
VFIO_DEVICE_FEATURE_TPH_ST	[SET only] Batch-program architected steering tag table entries

Progressive security policy (module parameter – tph_policy) :

Value	Mode	Description	Risk analysis
0	No-ST mode (default)	RESOLVE only returns PH for DMABUF source	Eliminates ST-related risk because ST programming unavailable
		ST programming unavailable	
1	Add IV mode	RESOLVE only returns PH for DMABUF source	Risk remains in Case 1 – MSI-X Table ST
		ST programming supports NONE/DMABUF/CPU sources	
2	Add DS mode	RESOLVE returns PH for DMABUF source, ST for DMABUF/CPU sources	Risk remains across all three cases: Case 1 - MSI-X Table ST, Case 2 - Config-Space ST, Case 3 - Device-specific ST
		ST programming supports NONE/DMABUF/CPU sources	



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Remaining Challenge - ST Target Lifecycle tracking

ST Target Lifecycle change:

- case1: DMA-BUF: valid->invalid
- case2: DMA-BUF: ST change



Handling Strategies:

- **No handling:** TLPs may still carry stale ST values, potentially causing cache pollution and performance interference.
- **Handling:** two available handling options if we choose to manage the lifecycle:
 - Option 1: Userspace tracking
 - Option 2: Kernel tracking (please see below)

Required Actions

1. Detect target state change
2. Find target ST entries (devices and indexes)
3. Invalidate / update hardware ST entries



Open Challenge

Reverse Mapping: ST index -> Target

For Architected ST table:

1. Kernel refcount + reverse mapping
2. Attach dma-buf on first use, detach when refcount=0
3. Update if ST changed

For Device-Specific ST table:

1. **Notify userspace**



Remaining Challenge - ST Target Lifecycle tracking (cont)

Common baseline:

Device-specific ST tables provide no generic kernel interface for enumeration or invalidation. **Both Option 1 and Option 2 rely on userspace for device-specific ST management.**

	Option 1: Userspace tracking	Option 2: Kernel tracking
Kernel behavior	No kernel tracking of dma-buf resources	Kernel maintains dma-buf refcounts; clear ST on resources freed, update ST on target changes
Responsibility	Userspace manages ST sync	- For architected ST table: kernel manages ST sync - For device-specific ST table: userspace manages ST sync
Pros	Minimal kernel changes	Automatically sync ST entries for architected ST table, reduces userspace workload for that subset
Cons	Userspace must monitor target/ST state changes	- Complex cross-subsystem logic - Risk of refcount cycles & resource leaks - Higher upstream review cost

Recommendation: **option 1**, with clarifications added in kernel UAPI documentation. Rationale:

- A split hybrid model would introduce inconsistent semantics: architected-ST handled by kernel, device-specific-ST handled by userspace. Unified userspace-managed lifecycle delivers consistent contract across all ST types.
- Kernel-side refcount tracking brings significant cross-subsystem complexity and refcount cycle risks with dma-buf.



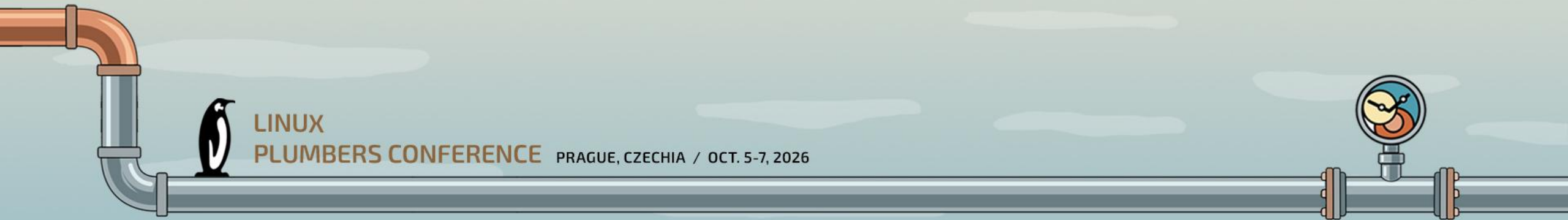
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

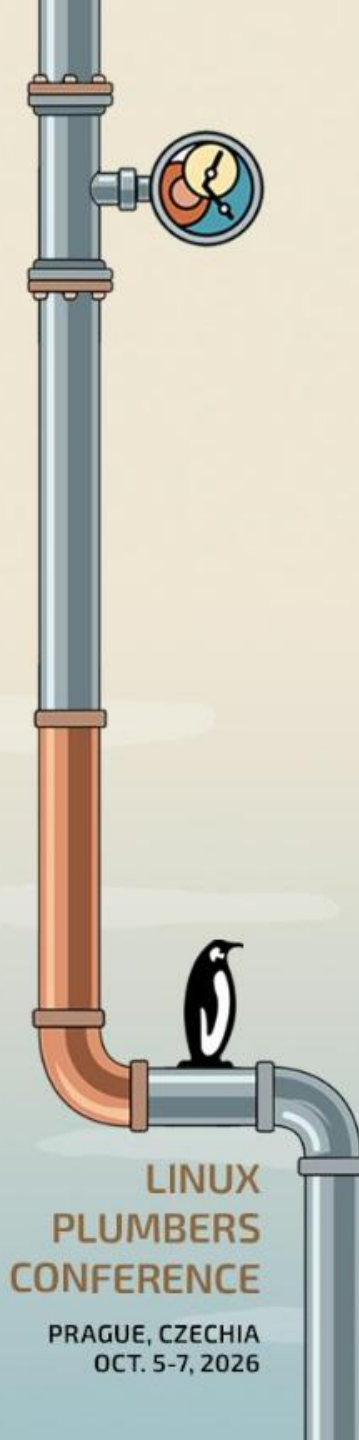


Roadmap

- In Progress
 - Community review & upstream iteration
 - Add `pci_p2pdma_distance()` before ST resolve for dma-buf – (in coming v21)
- Open Discussion
 - ST target lifecycle tracking
- Future
 - Automated selftest coverage without physical TPH hardware
 - TPH virtualization support
 - Userspace SDK (e.g. DPDK/SPDK) integration



Q&A



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

API Definition (1):

```
/* PCIe TPH device feature definitions for VFIO_DEVICE_FEATURE ioctl */
#define VFIO_DEVICE_FEATURE_TPH      14
#define VFIO_DEVICE_FEATURE_TPH_RESOLVE 15
#define VFIO_DEVICE_FEATURE_TPH_ST   16

/*
 * VFIO_DEVICE_FEATURE_TPH - Control and query PCI TPH capabilities
 *
 * SET: Opt-in to TPH feature for the device; flags must be zero.
 * GET: Return supported TPH capability bits (see VFIO_DEVICE_TPH_CAP_*)
 *      in flags. Returns error if SET has not been invoked first.
 *
 * Userspace must first invoke SET on this feature to enable TPH support,
 * receive valid capability bits via GET, gain permission for using
 * VFIO_DEVICE_FEATURE_TPH_RESOLVE / VFIO_DEVICE_FEATURE_TPH_ST, and access
 * the virtualized TPH capability registers via VFIO config space accesses.
 */
struct vfio_device_feature_tph {
    __u32 flags;
};

/* DMABUF source resolve processing hint supported */
#define VFIO_DEVICE_TPH_CAP_RESOLVE_DMABUF_PH (1u << 0)
/* DMABUF source TPH_ST batch programming supported */
#define VFIO_DEVICE_TPH_CAP_ST_DMABUF         (1u << 1)
/* CPU source TPH_ST batch programming supported */
#define VFIO_DEVICE_TPH_CAP_ST_CPU            (1u << 2)
/* NONE source (means steering tag is zero) TPH_ST batch programming supported */
#define VFIO_DEVICE_TPH_CAP_ST_NONE           (1u << 3)
/* DMABUF source resolve steering tag supported */
#define VFIO_DEVICE_TPH_CAP_RESOLVE_DMABUF_ST (1u << 4)
/* CPU source resolve steering tag supported */
#define VFIO_DEVICE_TPH_CAP_RESOLVE_CPU_ST    (1u << 5)
```



API Definition (2):

```
/* Source holds dma-buf fd */
#define VFIO_DEVICE_TPH_SRC_DMABUF    (1u << 0)
/* Use extended 16-bit steering tag namespace */
#define VFIO_DEVICE_TPH_EXTENDED      (1u << 1)
/* Source holds volatile memory CPU ID */
#define VFIO_DEVICE_TPH_SRC_CPU_VOLATILE    (1u << 2)
/* Source holds persistent memory CPU ID */
#define VFIO_DEVICE_TPH_SRC_CPU_PERSISTENT  (1u << 3)
/* TPH_ST only: set steering tag to zero */
#define VFIO_DEVICE_TPH_SRC_NONE    (1u << 4)
/* TPH_ST only: fail if resolved steering tag is zero */
#define VFIO_DEVICE_TPH_REQUIRE_ST  (1u << 5)
```

```
/*
 * VFIO_DEVICE_FEATURE_TPH_RESOLVE - Resolve TPH attributes from given source
 *
 * GET only
 *
 * Requires prior successful SET on VFIO_DEVICE_FEATURE_TPH, otherwise
 * this feature will return error.
 *
 * @flags: IN - source type selector (see VFIO_DEVICE_TPH_SRC_*)
 *          and steering tag namespace selector (see VFIO_DEVICE_TPH_EXTENDED)
 * @src:   IN - source identifier, e.g. dmabuf fd when SRC_DMABUF is set in
 *          flags
 * @valid: OUT - bitmap indicating valid output fields, see
 *          VFIO_DEVICE_TPH_VALID_*
 * @ph:    OUT - TPH processing hint (valid when VALID_PH is set in valid)
 * @st:    OUT - TPH steering tag (valid when VALID_ST is set in valid)
 */
struct vfio_device_feature_tph_resolve {
    __u32 flags;
    __u32 src;
    __u8 valid;
#define VFIO_DEVICE_TPH_VALID_PH (1u << 0) /* ph holds valid processing hint */
#define VFIO_DEVICE_TPH_VALID_ST (1u << 1) /* st holds valid steering tag */
    __u8 ph;
    __u16 st;
};
```



API Definition (3):

```
/*
 * VFIO_DEVICE_FEATURE_TPH_ST - Batch program architected steering tag
 * table entries
 * Return value: >=0 = number of successfully written entries;
 *               <0 = negative errno
 *
 * SET only
 *
 * Additional constraint: Userspace must complete two prerequisite steps
 * before invoking this feature:
 * 1. Call VFIO_DEVICE_FEATURE_TPH with SET to enable vfiio TPH permission;
 * 2. Write the TPH control register to activate device hardware TPH logic.
 *
 * Note: Steering Tag table entries are NOT preserved across PCI device
 * resets. For devices with ST table located in MSI-X vector space, all
 * ST entries are cleared along with the MSI-X table on reset and will
 * not be restored by PCI core state save/restore. Userspace must
 * re-program all required ST entries via this feature after any reset.
 *
 * @flags: IN - source type selector (see VFIO_DEVICE_TPH_SRC_*) and
 *          steering tag namespace selector (see VFIO_DEVICE_TPH_EXTENDED) and
 *          failure policy (see VFIO_DEVICE_TPH_REQUIRE_ST)
 * @start: IN - first steering tag table index / MSI-X vector
 * @count: IN - number of contiguous entries to program
 * @dests: IN - user pointer to array of source u32 values
 */
struct vfio_device_feature_tph_st {
    __u32 flags;
    __u16 start;
    __u16 count;
    __aligned_u64 dests;
};
```



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 3-7, 2020