# Extending AF_XDP for fast co-located packet transfer

Debojeet Das, Kevin Prafull Baua, Aditya Kansara, Arghyadip Chakraborty, Dheeraj Kurukunda, Mythili Vutukuru, and Purushottam Kulkarni

debojeetdas@cse.iitb.ac.in

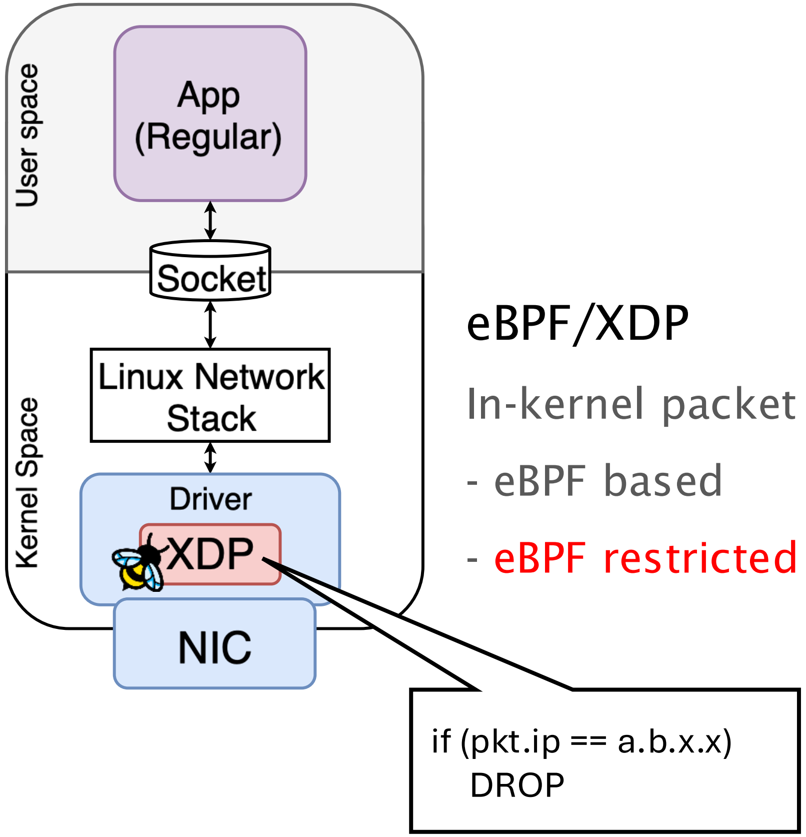
Systems and Networking Research Group (SynerG)
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

FOSDEM'26

JAN 31, 2026

# Fast network I/O with XDP and AF_XDP

XDP and AF_XDP allows us to deploy our-own high-performance load balancer, firewall, etc.



```
if (pkt.ip == a.b.x.x)
    DROP
```

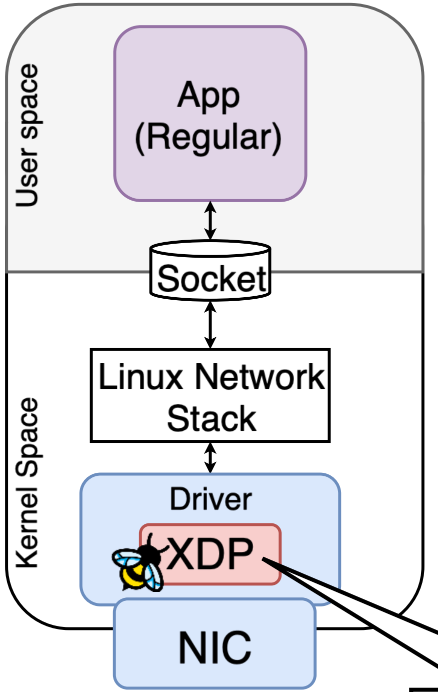
## eBPF/XDP

In-kernel packet processing

- eBPF based
- eBPF restricted constraints

# Fast network I/O with XDP and AF_XDP

XDP and AF_XDP allows us to deploy our-own high-performance load balancer, firewall, etc.

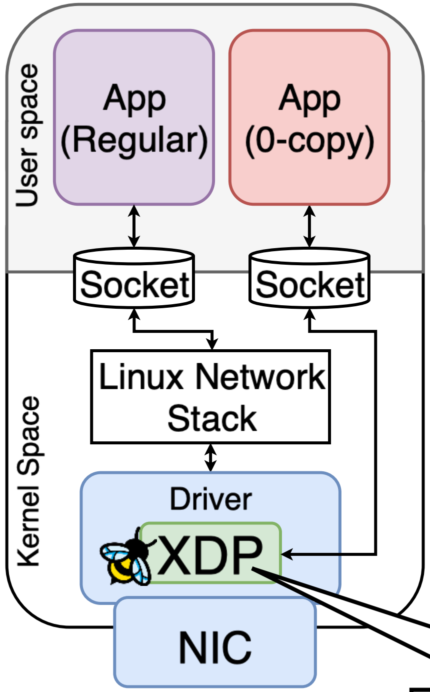


```
if (pkt.ip == a.b.x.x)
    DROP
```

## eBPF/XDP

In-kernel packet processing

- eBPF based
- eBPF restricted constraints



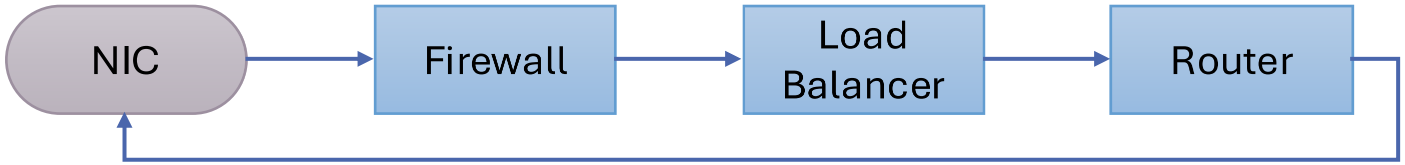
```
if (pkt.ip == a.b.x.x)
    Redirect to AF_XDP sock 1
```

## AF_XDP

Partial kernel bypass via XDP

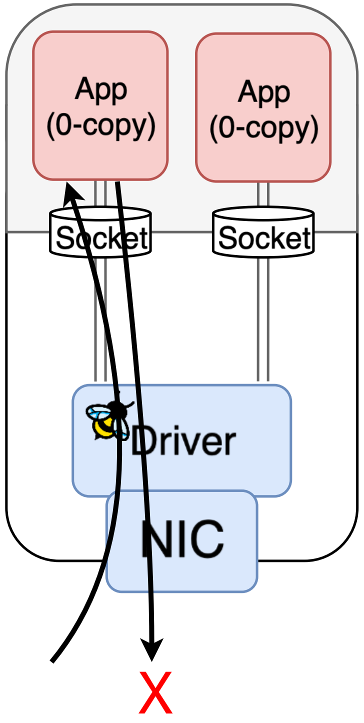
- zero-copy delivery to user apps
- Interrupt driven/Busy poll

# Co-locating multiple AF_XDP zero-copy apps



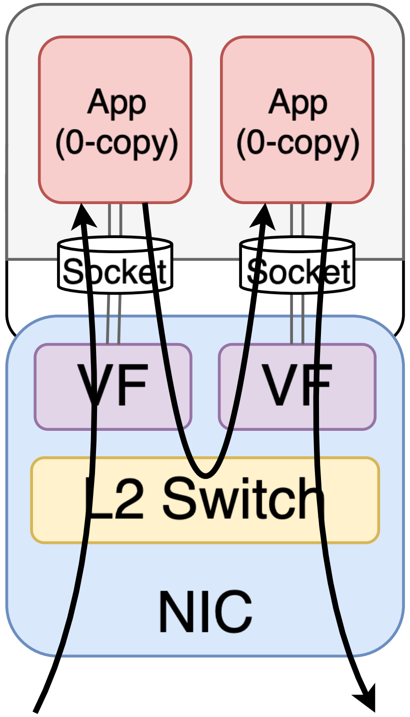
Goal: Efficient co-location of AF_XDP zero-copy apps to provide complex services

Problem: AF_XDP lacks socket-to-socket redirection much less zero-copy transfers



# Existing solutions for co-located packet transfers between AF_XDP sockets

SR-IOV



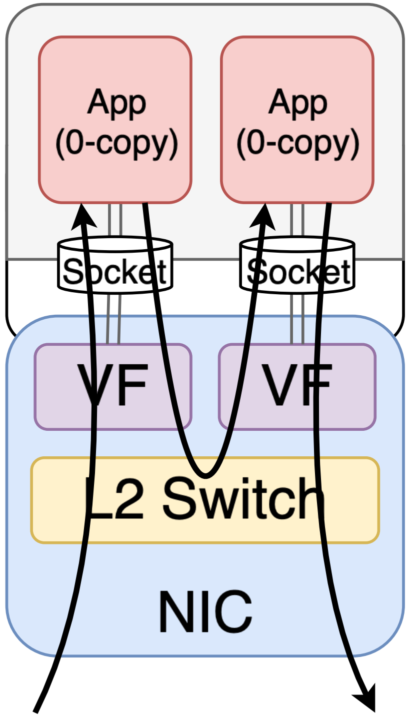
Pros:
No API modifications needed

Cons:
Limited AF_XDP support
- VF support only in mlx5
- SF support only in ice and mlx5

Performance hit from app-to-app copying

# Existing solutions for co-located packet transfers between AF_XDP sockets

## SR-IOV

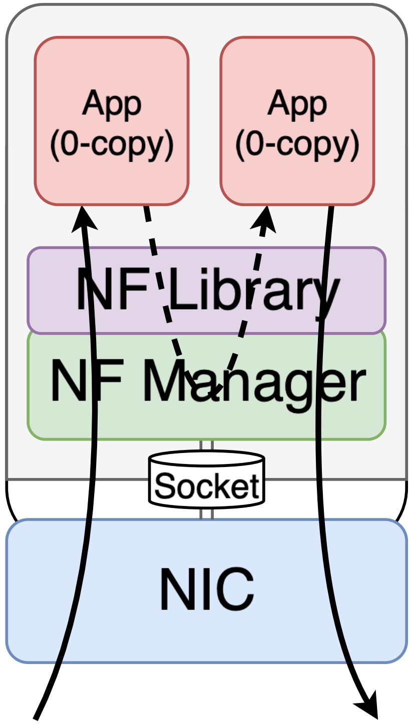


Pros:
No API modifications needed

Cons:
Limited AF_XDP support
- VF support only in mlx5
- SF support only in ice and mlx5

Performance hit from app-to-app copying

## Userspace Chaining*



Motivated from existing DPDK solutions like OpenNetVM, etc.

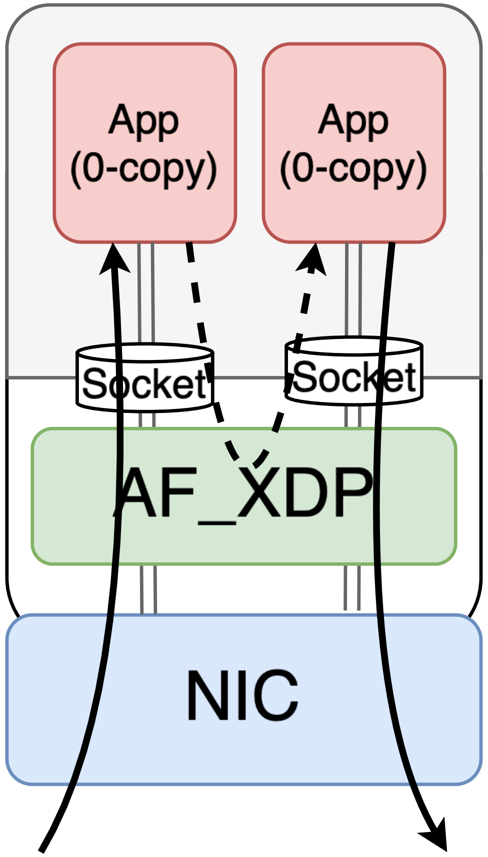
Pros:
Zero-copy packet transfers

Cons:
Requires new APIs

Creates tight coupling between Apps and libraries

*Possible Solution

# Our work: In-kernel packet transfers for AF_XDP



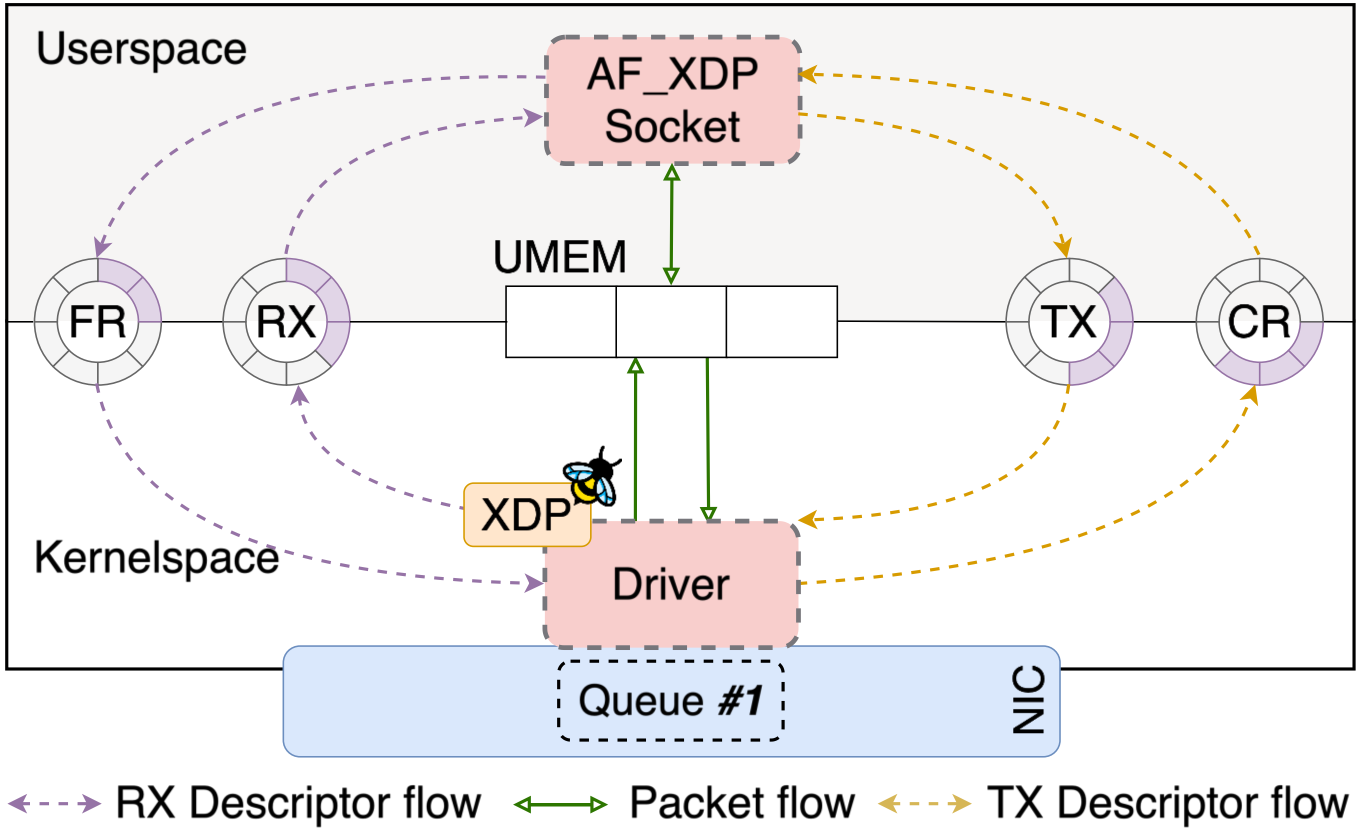
Flexible co-location via AF_XDP kernel extensions

Zero-copy for shared memory sockets; single-copy for others

Backward compatible solution: Transparent redirection with zero changes to userspace APIs

FLASH: Fast Linked AF_XDP Sockets for High-Performance Chaining

# Split data path of AF_XDP



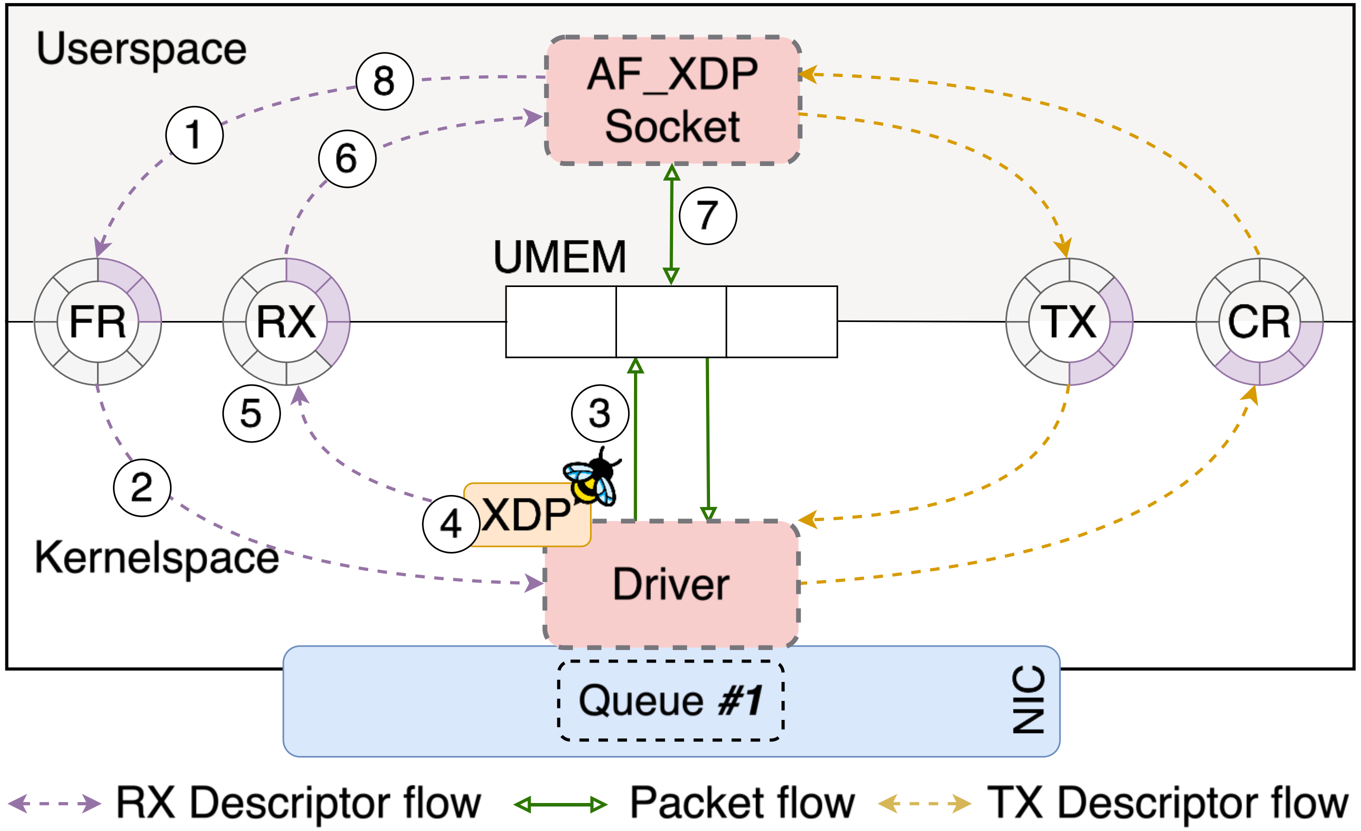
Driver handles RX/TX

UMEM (buffer pool) shared between userspace application and kernel

SPSC rings to maintain packet accesses

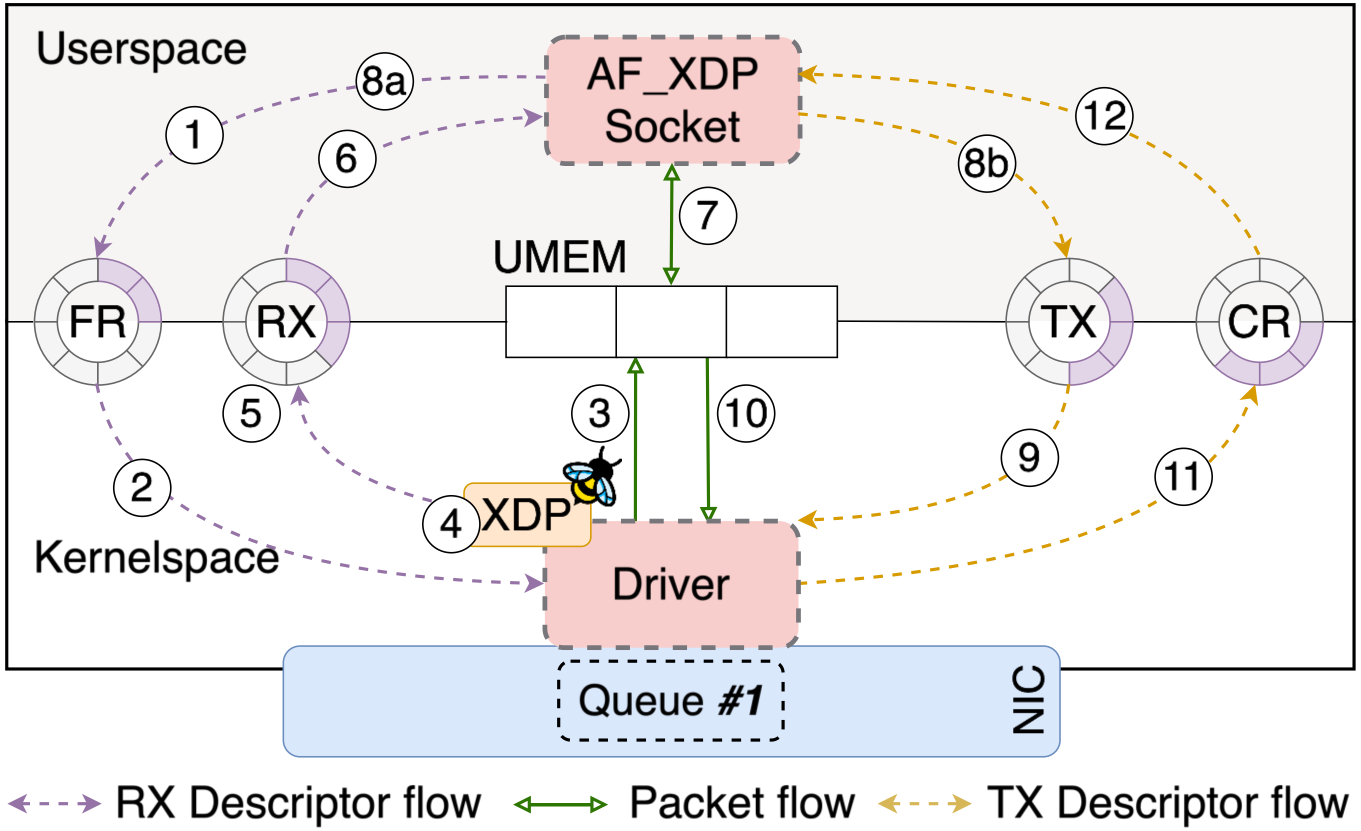
- Fill Ring and Rx Ring used in packet reception
- Tx Ring and Completion Ring used in packet transmission

# Packet Reception in AF_XDP



1. Applications provide free descriptors for DMA
2. Kernel allocates them to NIC
3. NIC DMA’s the packet to UMEM
4. napi_poll executes the XDP program which returns the socket where the packet should be sent
5. The Driver calls AF_XDP subsystem’s function to place the descriptor in Rx ring
6. Application reads 7. processes and 8. recycles descriptors.

# Packet transmission and modes of AF_XDP



8b. to 12. shows the transmission path which uses the Tx and completion rings

The kernel space operations are initiated by the driver with the help of AF_XDP subsystem functions*

Driver can either invoke them based on interrupts or via hints using *`sendto()`* and *`recvfrom()`* syscalls (Busypoll)

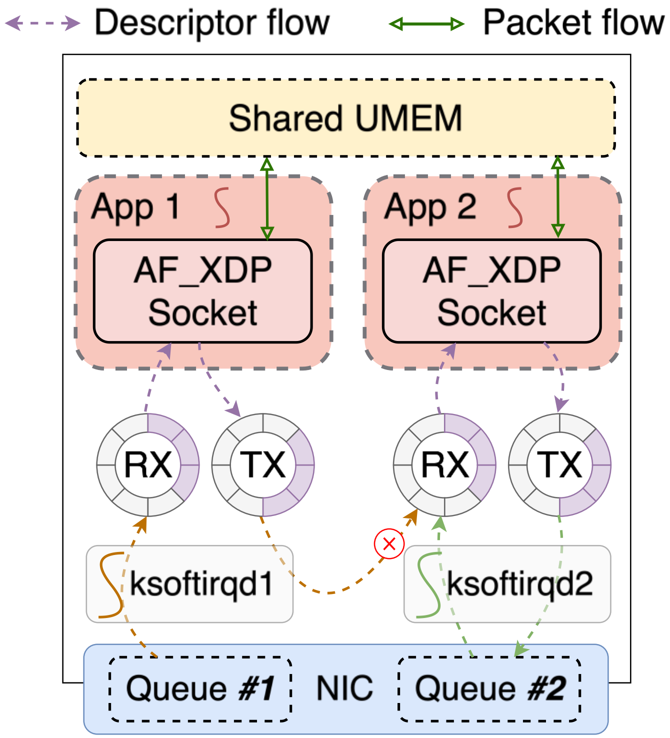
*The functions can be found at `include/net/xdp_sock_drv.h`

# Challenge #1 Ring semantics

Redirection violates SPSC (Single-Producer/Single-Consumer) semantics

## Key Idea #1: Backward-compatible MP/MC lockless rings

- CAS operations serialize MP/MC while maintaining SP/SC for userspace



```
struct xdp_ring {
    u32 producer;      // Old field; represents producer_tail
    u32 consumer;      // Old field; represents consumer_tail
    u32 flags;
    u32 producer_head; // Used only for MPSC
    u32 consumer_head; // Used only for SPMC
};
```

```
int xskq_enqueue_desc(struct xsk_queue *q, u64 addr, u32 len, u32 flags);
bool xskq_dequeue_desc(struct xsk_queue *q, struct xdp_desc* desc, struct xsk_buff_pool *pool);
```

# Zero-copy redirection with MP / MC rings

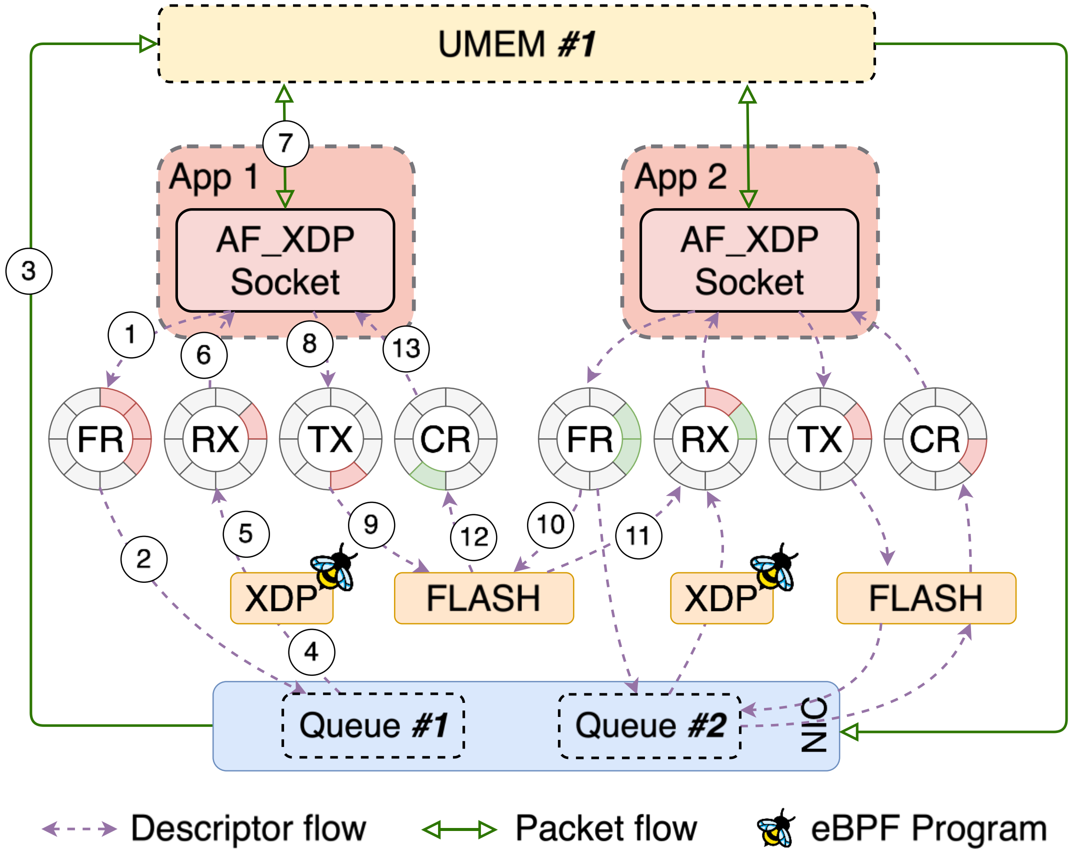
Packet data stays in shared UMEM

FLASH can move packet descriptor from TX ring of App 1 to RX ring of App 2

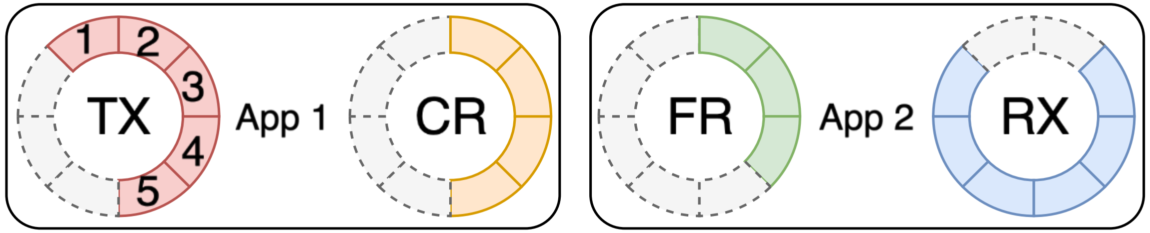
FLASH brings back empty descriptor from fill ring of App 2 to completion ring of App 1

Fill ring becomes MC in kernel and Rx ring becomes MC in kernel

*single-copy* redirection works similarily but using memcpy



# Challenge #2: Batched redirections



We, batch redirections to amortize CAS overhead

```
u32 xskq_bulk_enqueue_descs(struct xsk_queue *q, struct xdp_desc* descs, u32 n_descs);
```

- With new MPSC/SPMC rings, available capacity changes mid-redirection
- FR descriptors cannot be returned without violating ring semantics

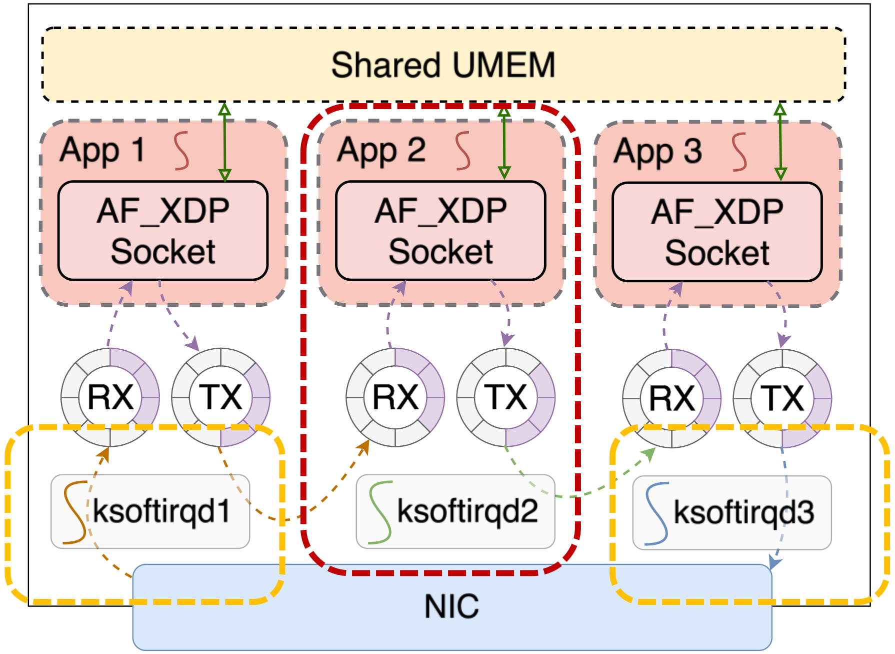
Key Idea #2: Partial TX + Descriptor Buffering

- Order ring updates so any packet placed in TX is guaranteed to redirect
- Buffer extra FR descriptors for use in the next iteration

# Challenge #3: Scheduling threads

To balance CPU utilization and performance a hybrid polling workflow is usually employed in applications:

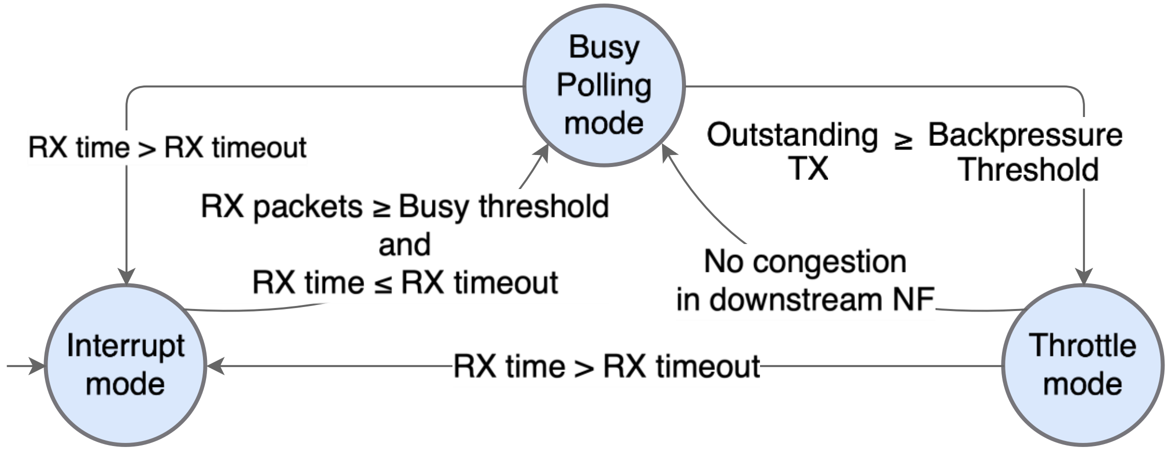
1. Start in interrupt mode: Use *poll()* with the `POLLIN` flag to block until packets arrive
2. Switch to busy-polling: At high rates, switch to a busy loop using *recvfrom()* to continuously process packets
3. Revert to interrupt mode: When load decreases, return to interrupt mode



By default, hybrid polling across packet transfers doesn’t work

Moreover, during congestion, the sender must retry transmissions until space becomes available this leads to wastage of CPU cycles

# Smart polling and early backpressure detection

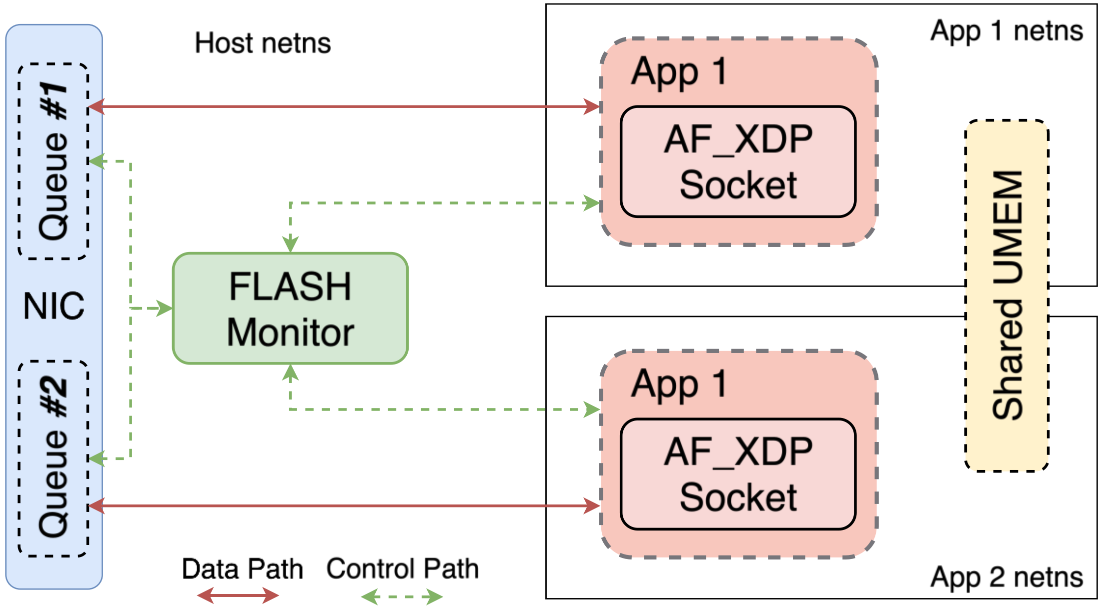

Busy Polling mode
RX time > RX timeout
RX packets ≥ Busy threshold and RX time ≤ RX timeout
Outstanding TX ≥ Backpressure Threshold
No congestion in downstream NF
Interrupt mode
RX time > RX timeout
Throttle mode

Key Idea #3: Smart Polling & Early Backpressure

- Extends hybrid polling across redirection paths
- Congestion Handling: Senders block via *poll()* (POLLOUT) until space
- Wake-up Signal: Receivers use *recvfrom()* (MSG_MORE) to signal senders

# Challenge #4: Performance vs. safety

How to isolate zero-copy Apps from each other in multi-tenant settings?



## Key Idea #4: FLASH monitor and safe runtime

- FLASH Monitor: Offloads privileged tasks and manages access control
- Rust Runtime: Ensures memory and packet isolation in multi-tenant setups

# But how do you specify the path of transfers?

We need a way to inspect active AF_XDP sockets and configure transfer rules between sockets

- FLASH should then use the rules to access the rings and perform the transfers
- But, the sockets (`struct xdp_sock`) and its rings are backed by process local fds

How do XDP get access of rings then?

- It stores the `struct xdp_sock` in a map and uses XDP program to select the socket on packet reception

Our Solution: Expose a sysfs interface under `/sys/kernel/flash` to store the sockets and configure redirection rules*

For example, if there are two sockets and we want all packets from socket 1 to go to socket 2

```
# echo 2 | sudo tee /sys/kernel/flash/1/next
```

*We also support dynamic packet transfers

# How much change is needed to support all this?

~700 LoC in AF_XDP subsystem

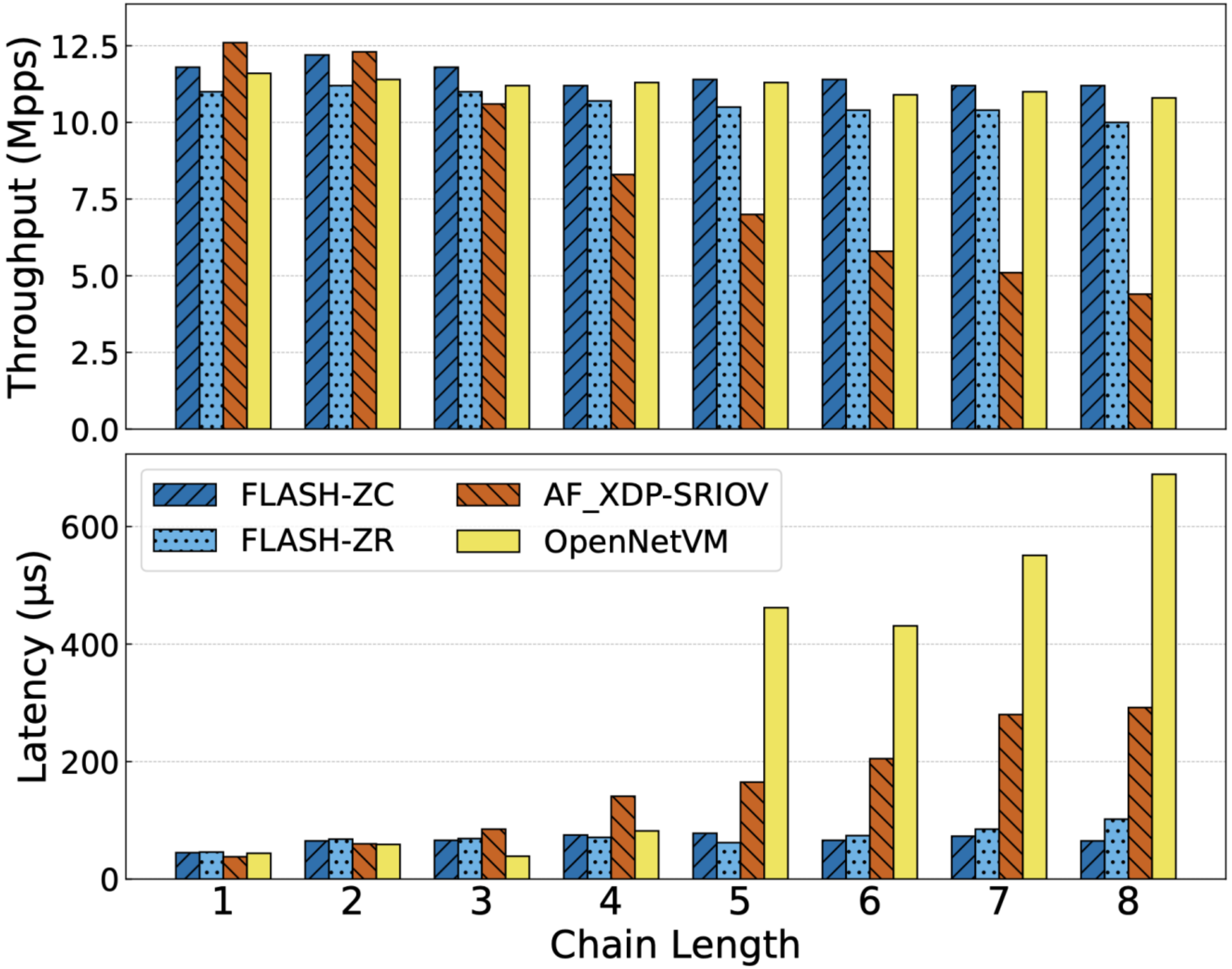
The changes mostly augments new features to the AF_XDP subsystem

Requires straightforward patching in driver to ignore packet transmissions

```c
bool xmit_batch(struct xsk_buff_pool *pool, unsigned int budget)
{
        struct xdp_desc *descs = pool->tx_descs;
        unsigned int nb_pkts = 0;

        // Fetch a batch of descriptors for transmission
        nb_pkts = xsk_tx_peek_release_desc_batch(pool, budget);
        if (!nb_pkts)
                return true;

        // If redirection is configured, skip NIC transmission
        if (pool->no_tx_out)                    // New lines
                return nb_pkts < budget;        // New lines

        // Proceed with normal transmission
        return nb_pkts < budget;
}
```

We have already added support for ixgbe, i40e, ice and mlx5

# FLASH performance gains



Baselines:

AF_XDP-SRIOV: AF_XDP over SR-IOV

OpenNetVM: DPDK userspace chaining

Our work:

FLASH-ZC: FLASH C library zero-copy

FLASH-ZR: FLASH Rust library zero-copy

FLASH provides lowest latency among all solutions.

FLASH-ZC provides 1.1 x - 2.5 x higher throughput than SR-IOV

24-core Intel Xeon Gold 5418Y, 128 GB RAM, HT disabled - Mellanox Connect X-4 MT27700 40Gbps (mlx5)
Ubuntu 22.04 LTS, Linux Kernel 6.10.6 – Chains of L2FWD switches – Used Pktgen for load generation

# Summary

- The Problem: AF_XDP lacks native support for zero-copy packet redirection between co-located sockets
- The Solution: FLASH extends the AF_XDP kernel subsystem to enable high-performance, transparent chaining
- Key Ideas:
    - MP/MC Lockless Rings
    - Batching & Buffering
    - Smart Polling
- A sysfs interface for setting up the redirections

# Future Directions

- From sysfs to XDP_EGRESS: Replace current sysfs-based rule configuration with a native XDP hook at the egress point
  - packet-steering logic can be entirely programmable via eBPF
  - Interesting use case of XDP_EGRESS? (LPC 2025)
- Generalizing the service chaining framework to support diverse set of applications
- mTCP Integration: Currently developing zero-copy support for TCP-based applications
- Using SPMC/MPSC rings of AF_XDP for other use-cases:
  - Packet redirection from one queue to another queue during ingress

We are also working on a Rust rewrite of FLASH monitor

# Thank You

debojeetdas@cse.iitb.ac.in

Project Link: