Fast OVS Datapath with XDP

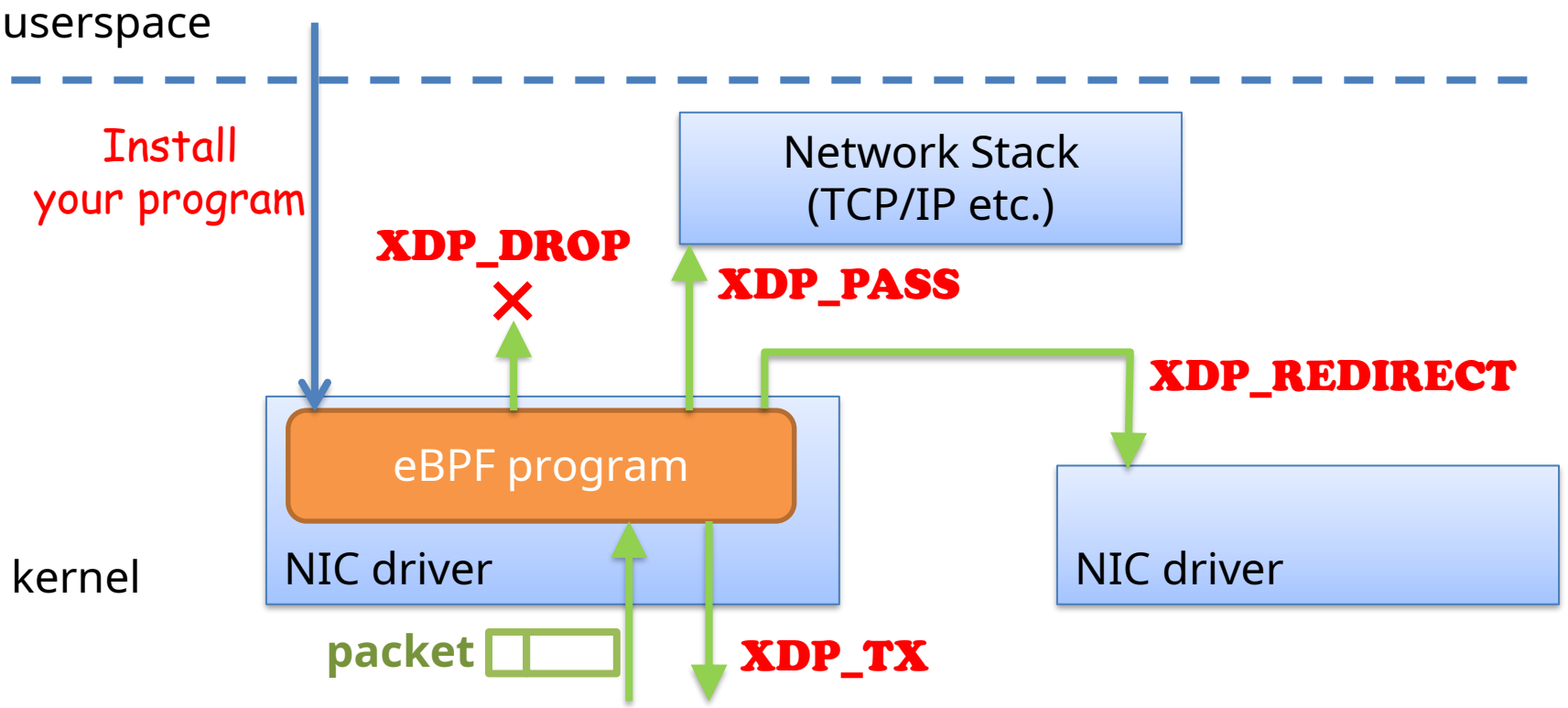
Toshiaki Makita (NTT Open Source Software Center)
William Tu (VMware)

Outline

- XDP and AF_XDP
- OVS AF_XDP netdev
- Fast OVS datapath
  - Approach-A) xdp_flow
  - Approach-B) OVS xdp flow api provider
- Performance
- Challenges
- Summary

XDP (eXpress Data Path)

- In-kernel fast (express) data path
- Execute your eBPF program at driver
  - Immediately after it receives packets
  - Low overhead: No metadata (skb) allocation



XDP

- Fast
  - Minimum overhead
- Flexible
  - High programmability by eBPF
- Good integration with existing kernel network stack
  - XDP_PASS

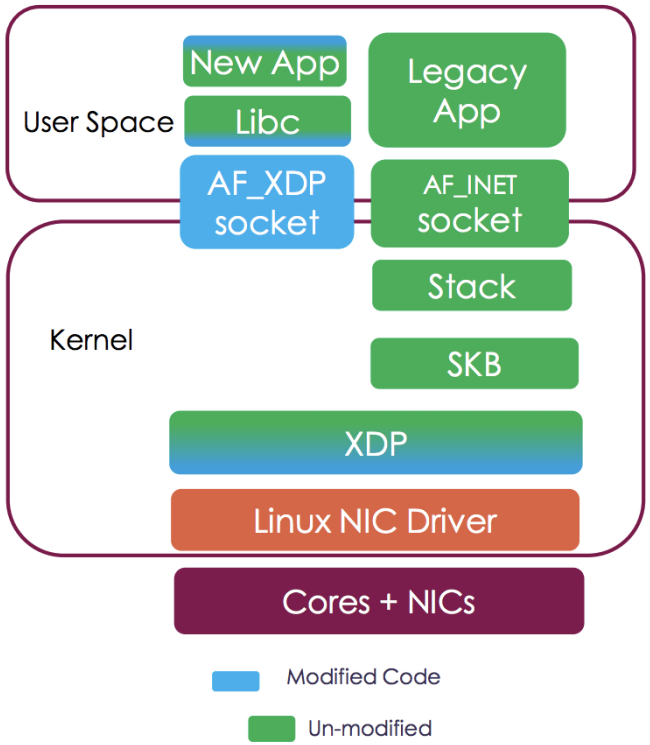
- Hard to use
  - Fight with eBPF verifier error

XDP

- Pre-implemented typical XDP network functions help users
  - Accelerate existing features by XDP
    - Virtual switch, routing, firewall, traffic control…
  - Virtual switch: **Open vSwitch**

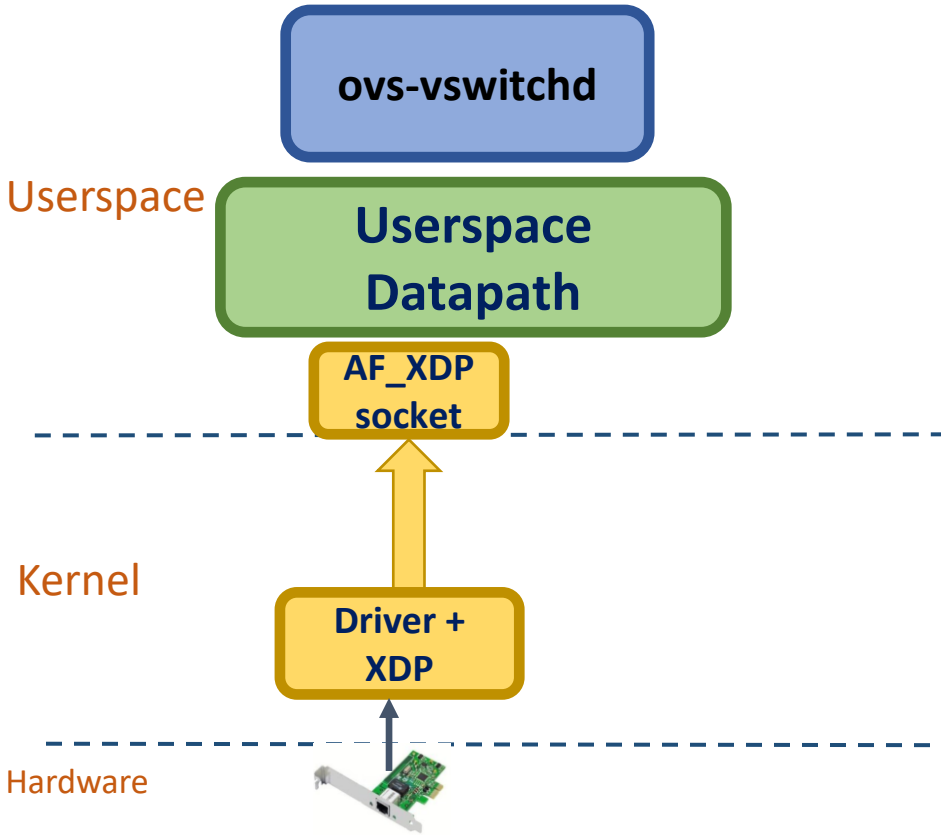
Linux AF_XDP

- A new socket type that receives/sends raw frames with high speed
- Use XDP program to trigger receive
- Userspace program manages Rx/Tx ring and Fill/Completion ring.
- Zero Copy from DMA buffer to user space memory.



From "DPDK PMD for AF_XDP"

OVS AF_XDP netdev



Idea

- Use AF_XDP socket as a fast channel to usersapce OVS datapath
- Flow processing happens in userspace
- Re-use existing OVS-DPDK datapath
- Attach a minimum XDP program, no processing in kernel ☹

OVS AF_XDP netdev

- Main objective is flexibility
  - Accomplished
- Basically faster than OVS kernel module, but…
  - Userspace solution has extra overhead for some cases, e.g. containers traffic
- In-kernel flow steering is better performance-wise
  - Steer packets in XDP context
  - No kernel-to/from-userspace copying

Fast OVS Datapath

- We proposed two approaches

Approach-A) xdp_flow

  – in-kernel solution

Approach-B) OVS xdp flow api provider

  – Using OVS flow offload API provider

Approach-A

xdp_flow: Flow offload to XDP

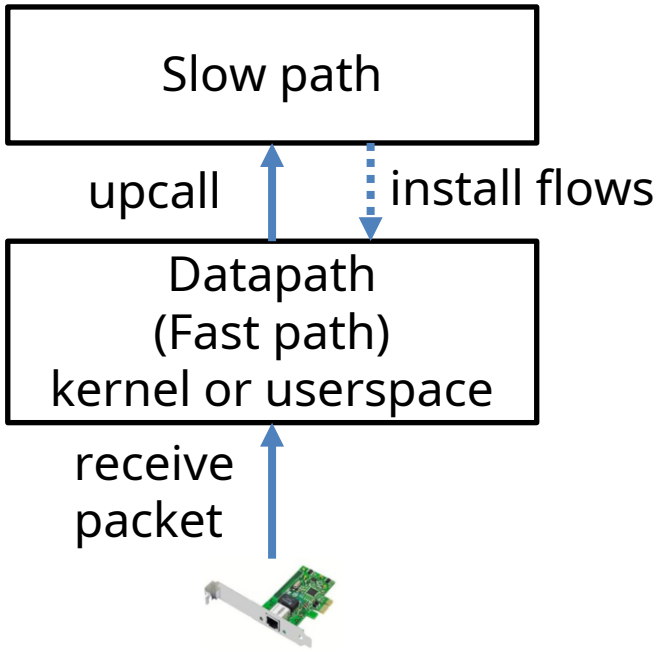
- Generic flow offload engine in kernel

- Why in-kernel?
  - Multiple network features are doing similar flow handling
    - OVS, TC flower, nftables
  - Create a generic XDP flow offload engine in-kernel
    - Can offload all of similar flow features with the same mechanism
    - Use existing generic flow HW-offload framework in-kernel
    - OVS can be offloaded through TC-flower HW-offload

Existing OVS HW-offload mechanism

- OVS uses TC flower to offload flows (existing feature)

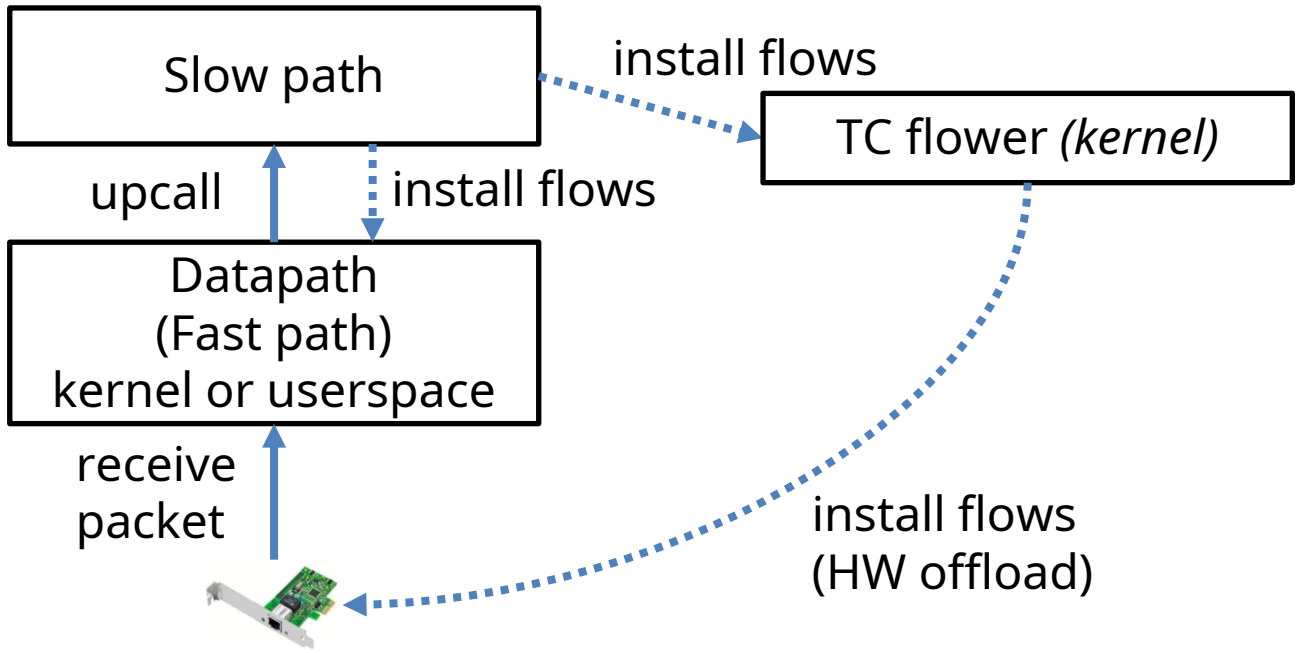
Figure: Packet handling without TC flower offload



Existing OVS HW-offload mechanism

- OVS uses TC flower to offload flows (existing feature)

Figure: How existing TC flower offload control plane works

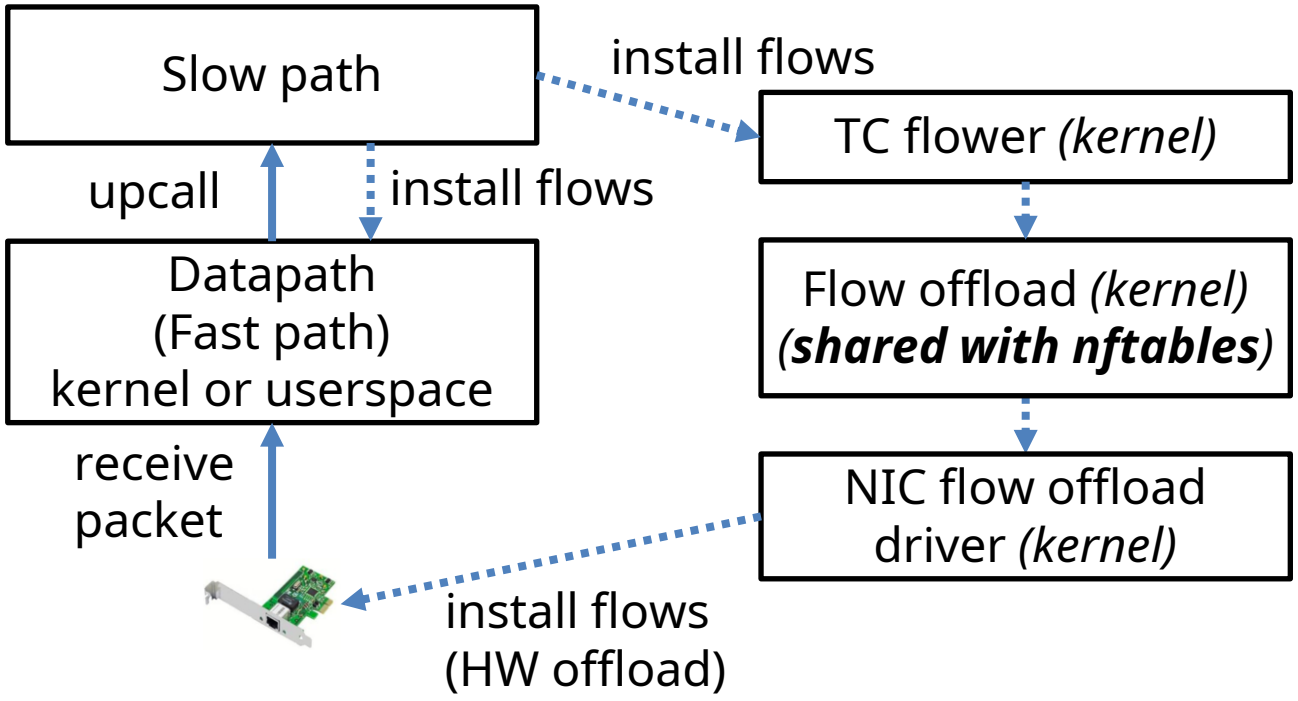


* For OVS-DPDK, rte_flow can be used instead

Existing OVS HW-offload mechanism

- TC flower uses generic flow offload infrastructure shared with nftables

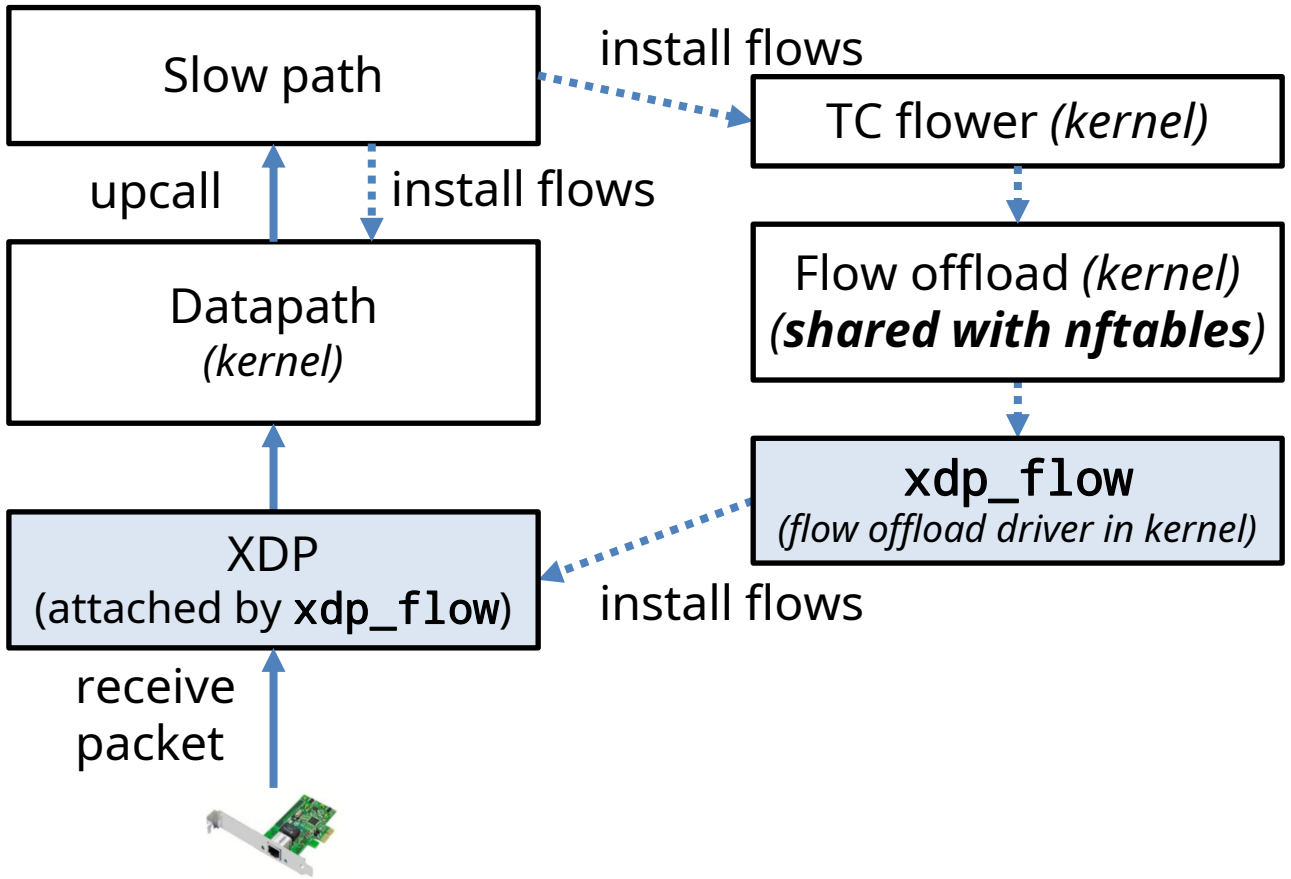
Figure: TC flower offload through generic flow offload infrastructure



Approach-A

xdp_flow: for OVS

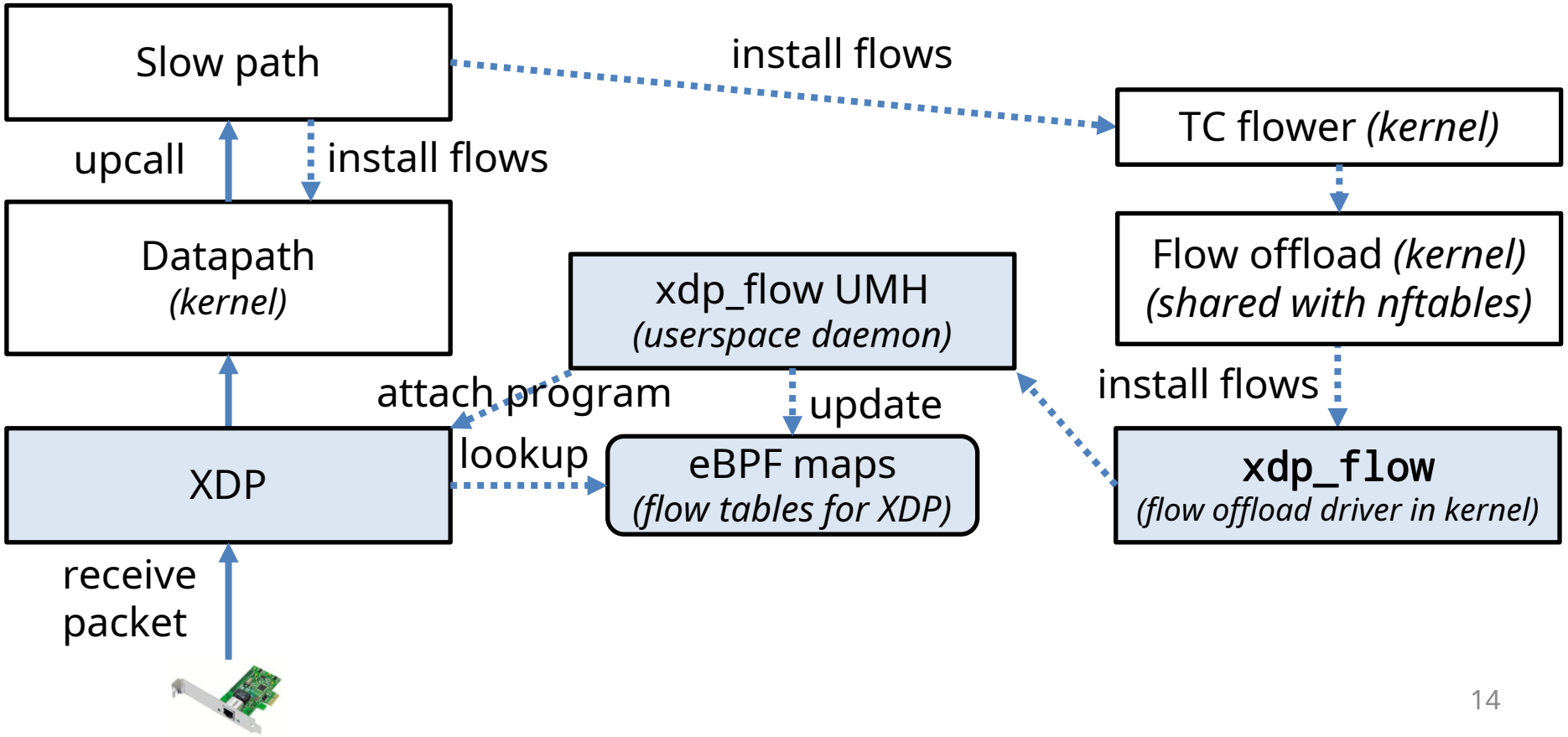
- Offload flows (e.g. TC flower) to XDP
  - OVS can be offloaded to XDP through TC



Approach-A

xdp_flow control plane details

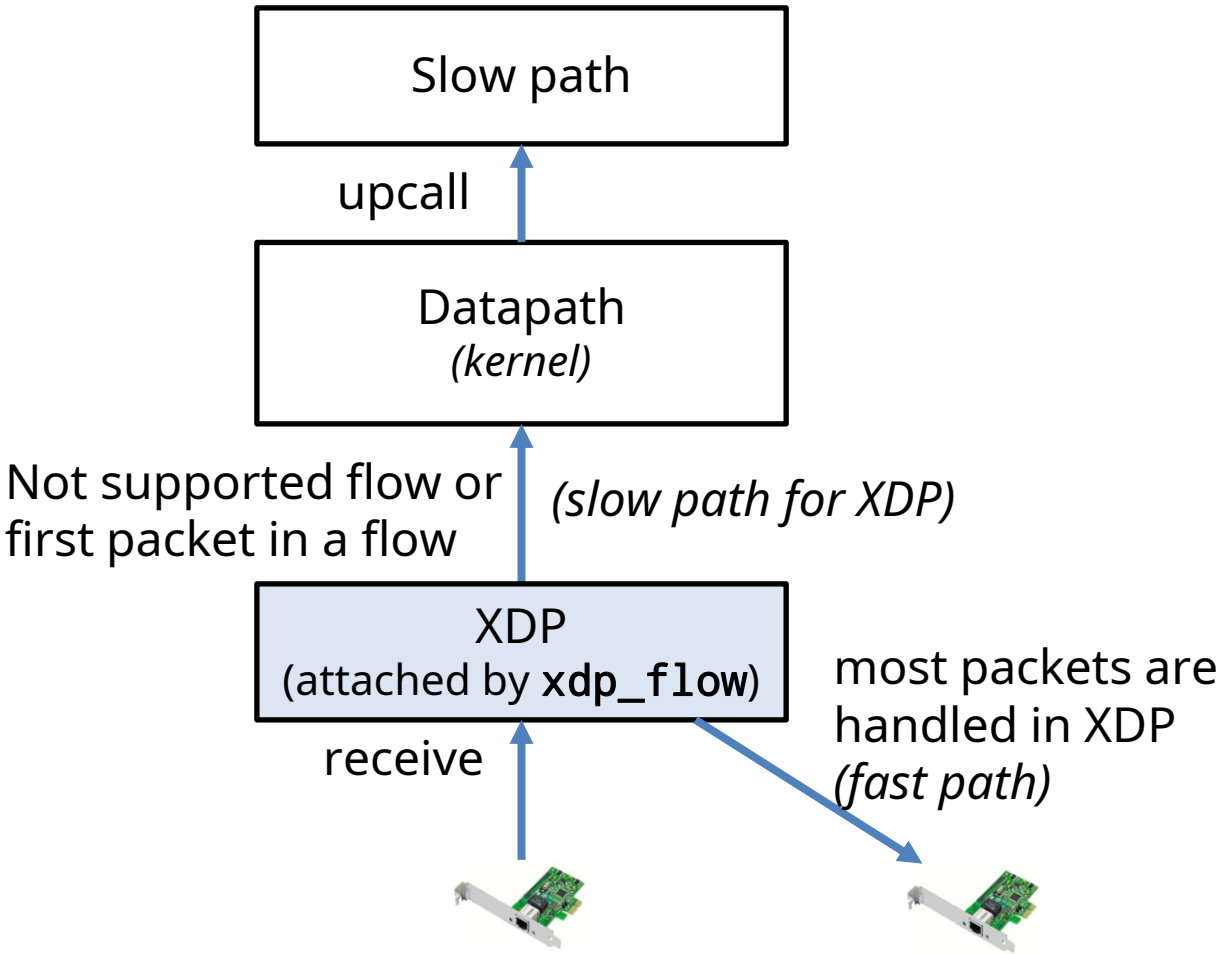
- Use UMH (user-mode helper) embedded in kernel module (user-mode blobs)
- eBPF program is also embedded in UMH



Approach-A

xdp_flow data plane

- Handle most packets in XDP



Approach-A

xdp_flow pros/cons

- Pros
  - Share codes with multiple functions
    - OVS, TC, nftables
  - Transparent UI
    - Existing commands (ovs-ofctl, etc) can be used
  - Simple UI to enable the feature
    - ethtool –K eth0 flow-offload-xdp on
- Cons
  - Complexity due to indirection layers like UMH
  - Hard-coded (precompiled) embedded eBPF program
    - Cannot customize codes for individual users
    - No integration with existing AF_XDP program of OVS

Approach-A

xdp_flow RFC review

- Feedback from netdev community (Other ideas for offloading to XDP transparently)
  - Userspace daemon (for generic flow handling)
    - Load its original XDP program for flow handling
    - Snoop netlink TC event and modify BPF maps to inform BPF prog of that
    - Hard to emulate TC behavior as it's changing frequently…
  - BPF helper function (for OVS only)
    - An API to access ovs kernel module flow table
    - Hard to refactor ovs module to expose the API
    - Does not work with AF_XDP datapath
  - OVS userspace daemon (for OVS only) (**Approach-B**)
    - Load its original XDP program for OVS flow table handling
    - Natural extension for OVS slow path as a software flow offload mechanism

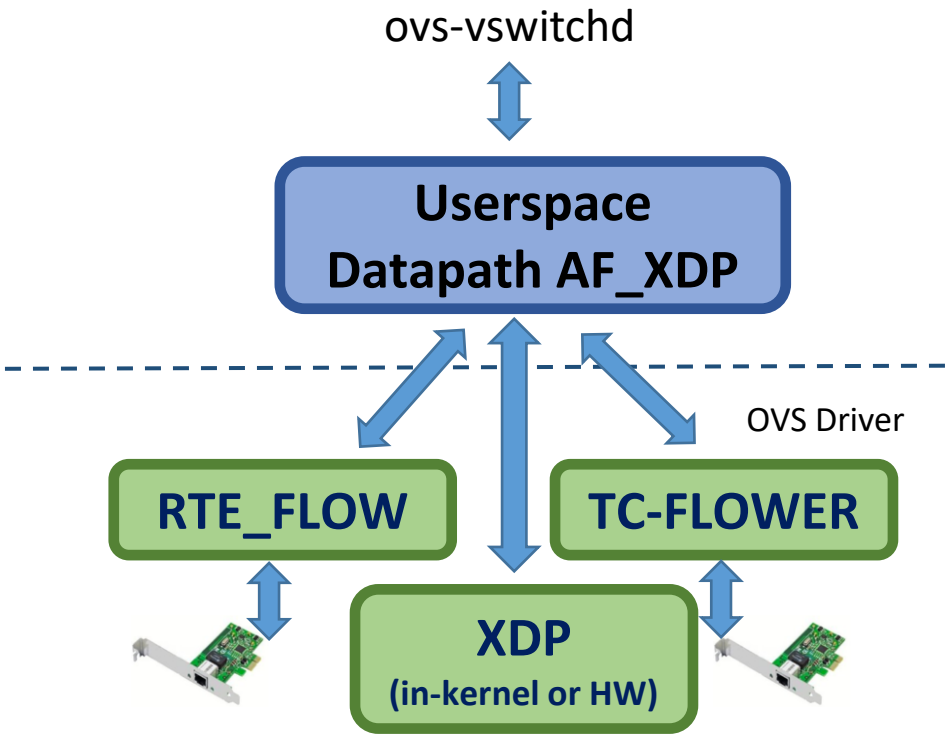
Approach-B

OVS xdp flow api provider

- Userspace implementation of xdp_flow
- Uses offload mechanism in ovs-vswitchd
- Work with AF_XDP netdev
- Admins can attach any XDP program
  - As long as it uses AF_XDP
  - Can be minimal program for each use-case (lower overhead)
  - OVS source tree provides reference program for XDP flow offload

Approach-B

OVS Offload API (flow api)



Currently two APIs

- Translate the datapath flow into rte_flow or tc-flower
- ovs/lib/netdev-offload-*.c

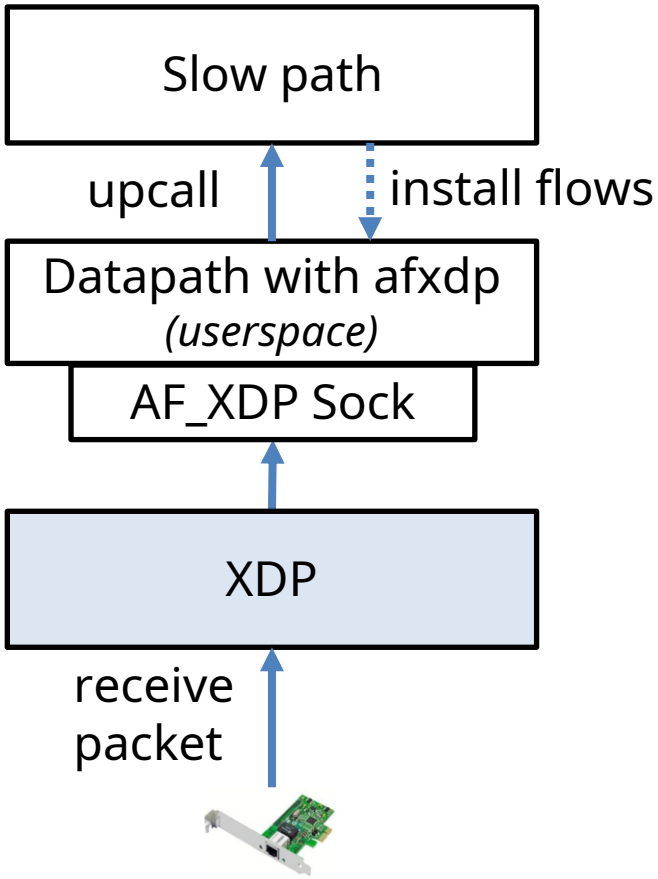
xdp flow api provider

- Use the same control plane
- Translate into XDP flow
- Add another offload API
- ovs/lib/netdev-offload-xdp.c

Approach-B

OVS xdp flow api provider control plane

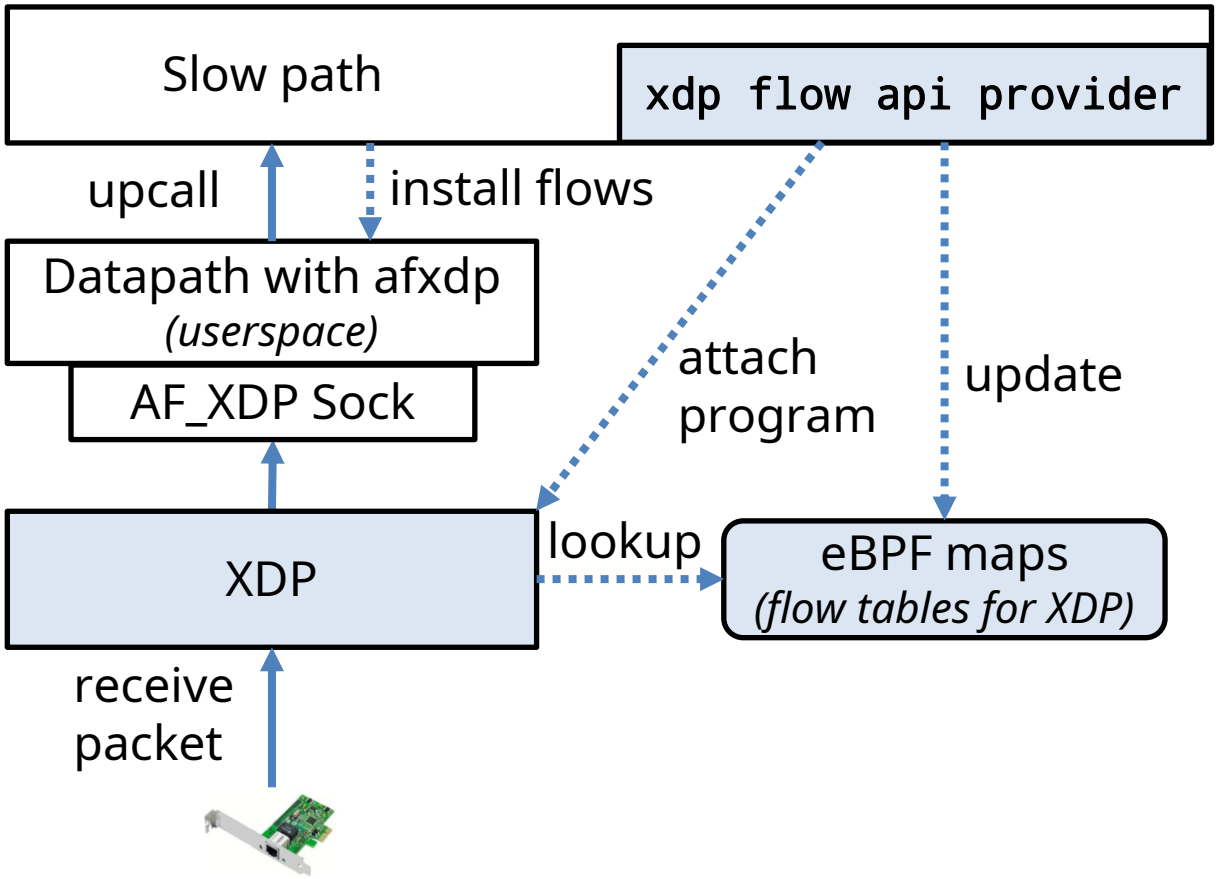
Figure: afxdp netdev packet handling without OVS xdp flow api provider



Approach-B

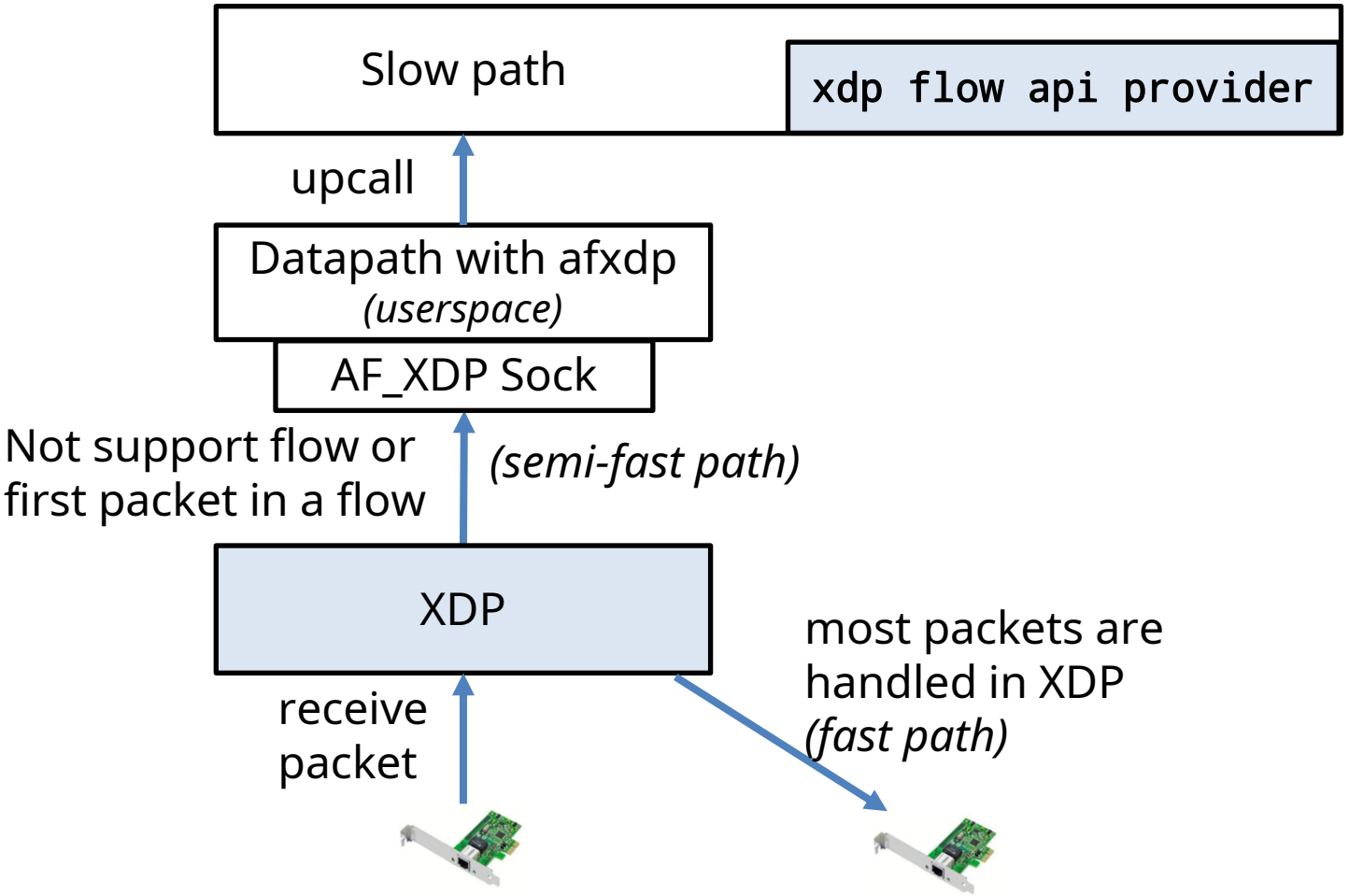
OVS xdp flow api provider control plane

Figure: How OVS xdp flow api provider control plane works



Approach-B

OVS xdp flow api provider data plane



OVS Offload Comparison

	rte_flow	tc-flower sw	tc-flower hw	This work (xdp flow api)
Performance	HW line rate	TC's performance	HW line rate	Driver level performance
Flexibility for new feature	Low	Medium	Low	High
System requirement	HW supported-NIC	Linux Kernel	HW supported NIC	Linux Kernel
Tunnel support	Yes	Yes	Yes	Not yet
Connection tracking support	Depends on vendor	Yes	Depends on vendor	Not yet

Approach-B

Performance

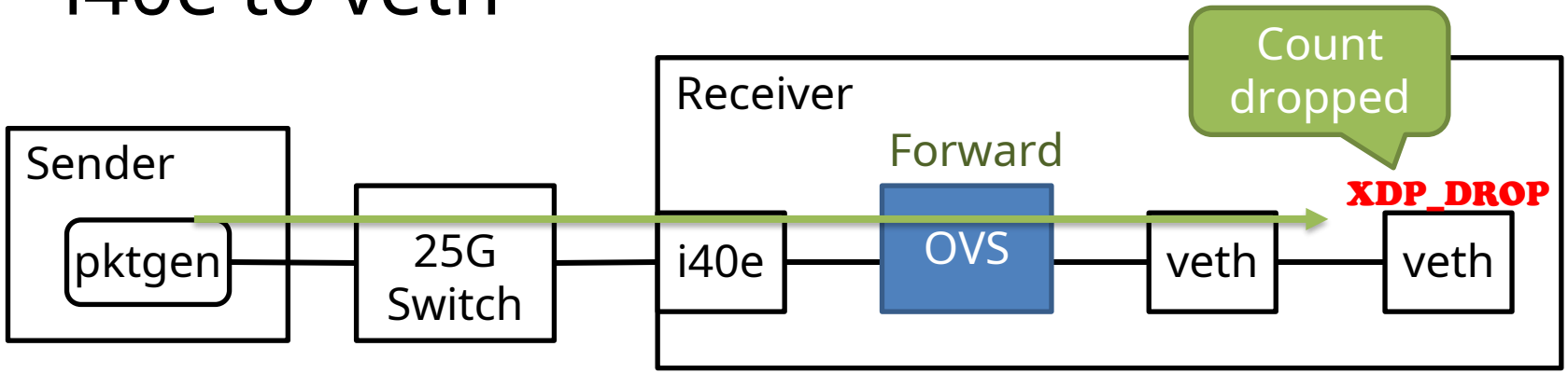
- Measure Approach-B (OVS xdp flow api provider) forwarding throughput

- Test environment
  - Intel Xeon Silver 4114 2.20 GHz
  - Intel XXV710 (25G, i40e)
  - kernel 5.5.5
- Traffic
  - 1 flow
  - Send UDP from pktgen
    - Approx. 37 Mpps (25G wire rate)

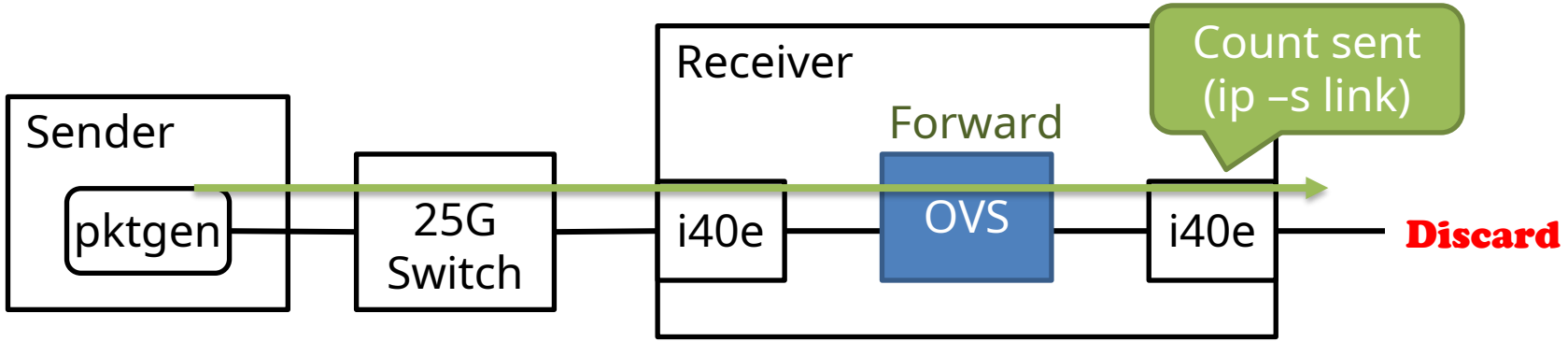
Approach-B

Performance

- i40e to veth



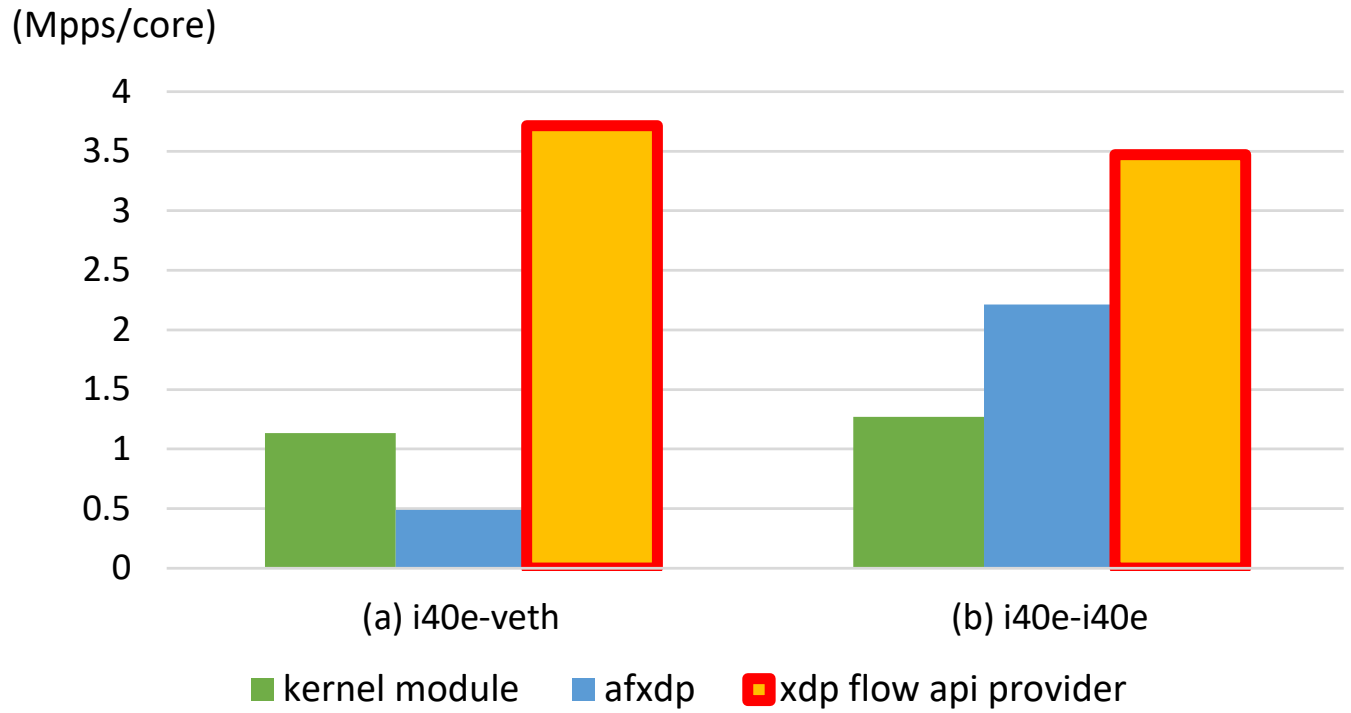
- i40e to i40e



Approach-B

Performance

- XDP flow api: 3.5 ~ 3.7 Mpps/core



Note1: afxdp result is devided by 2 since it uses two cores for one flow
Note2: veth AF_XDP performance is low (no AF_XDP zerocopy support on veth)

Challenges

- More keys/actions support
  - Currently support very basic keys/actions
  - Support more keys/actions like tunneling
- Further performance improvement
  - Experimental kernel patch for xdp_flow showed 5.2 Mpps/core
- Hardware offload
  - Offload XDP to NIC hardware
  - Currently impossible due to map-in-map

Summary

- OVS performance acceleration by XDP
- Approaches
  - A) xdp_flow (in-kernel)
  - B) OVS xdp flow api provider (in-userspace)
    - This works with OVS AF_XDP
- Performance (Approach B)
  - Forwarding: 3.5~3.7 Mpps/core
- Users can use XDP-accelerated virtual switch (OVS) with our approach
  - Fast and highly flexible

Backup

xdp_flow motivation

- A generic XDP flow processing for kernel
  - Can be used by different kernel components
  - Implemented as a kernel module