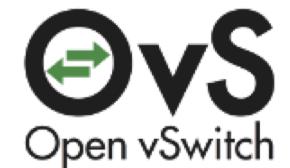


Bringing the Power of eBPF to Open vSwitch

Linux Plumber 2018

William Tu, Joe Stringer, Yifeng Sun, Yi-Hung Wei

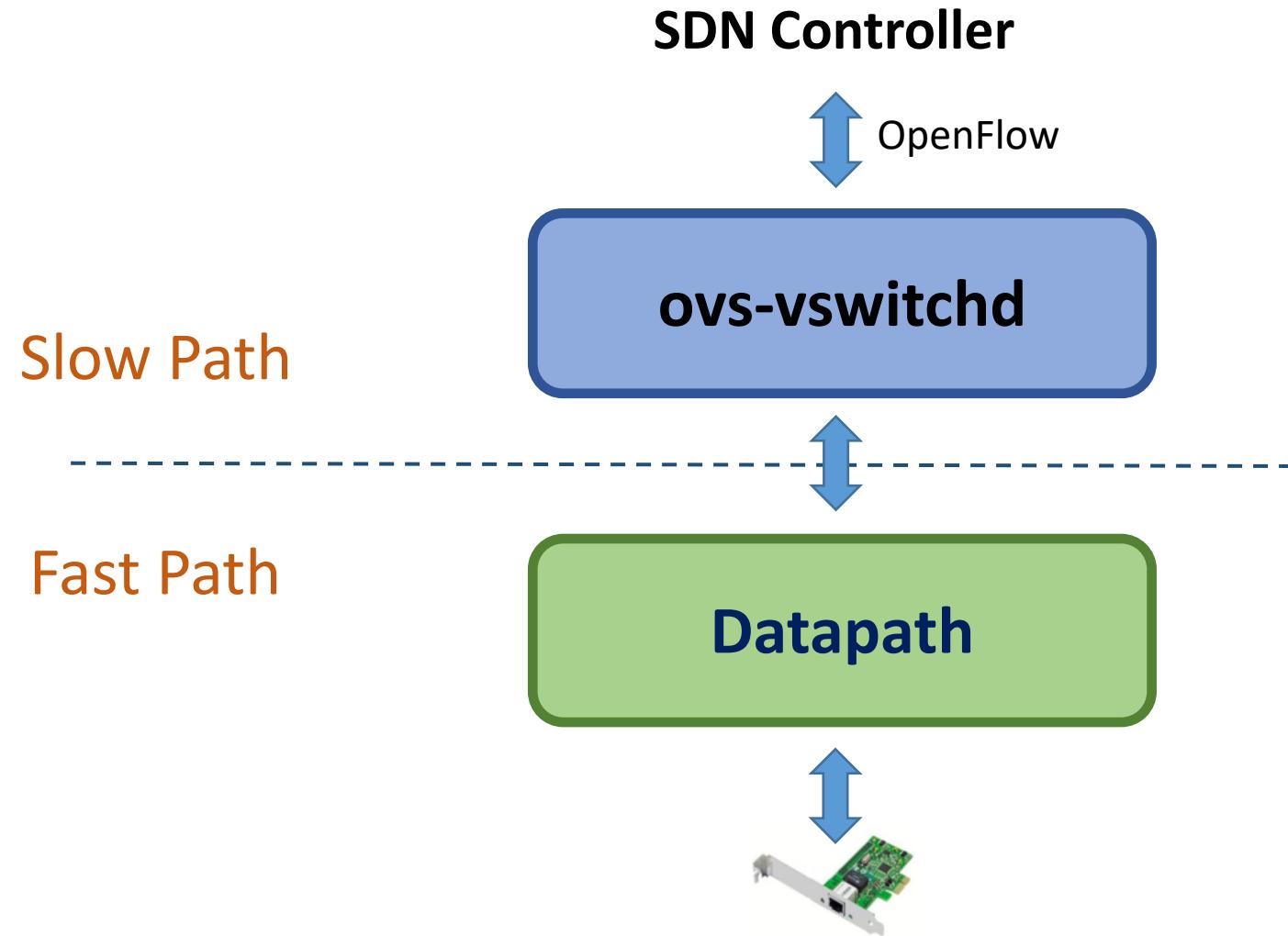
VMware Inc. and Cilium.io



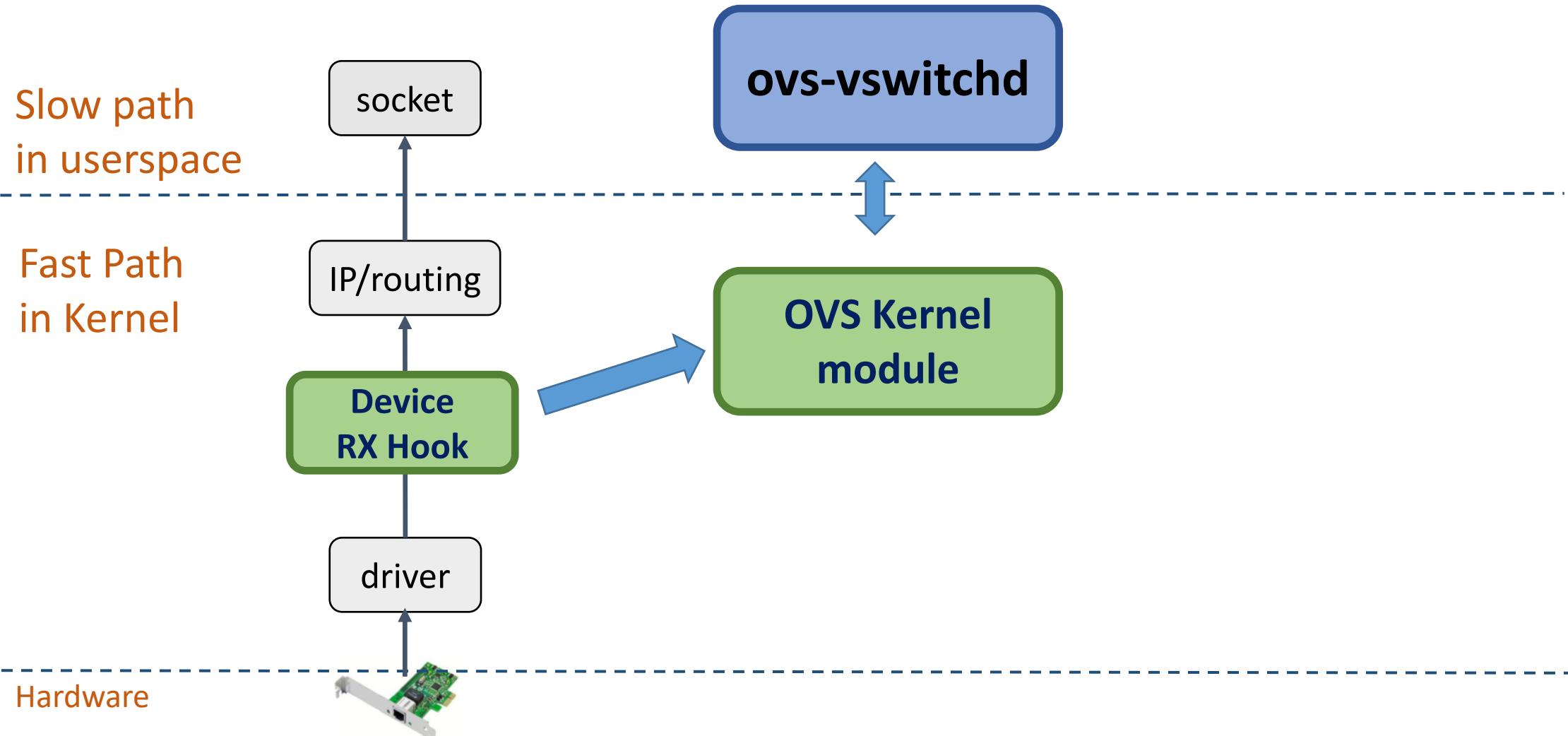
Outline

- Introduction and Motivation
- OVS-eBPF Project
- OVS-AF_XDP Project
- Conclusion

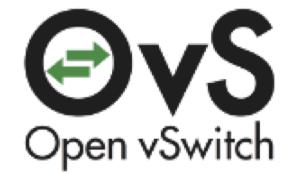
What is OVS?



OVS Linux Kernel Datapath

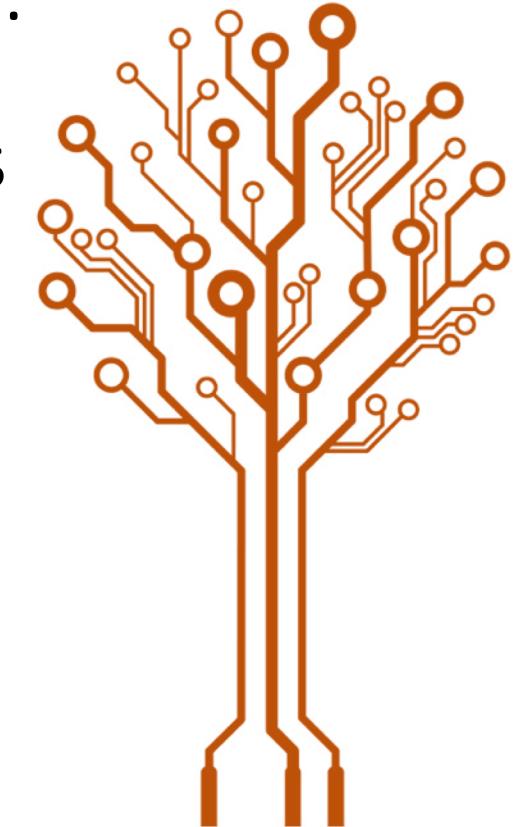


OVS-eBPF



OVS-eBPF Motivation

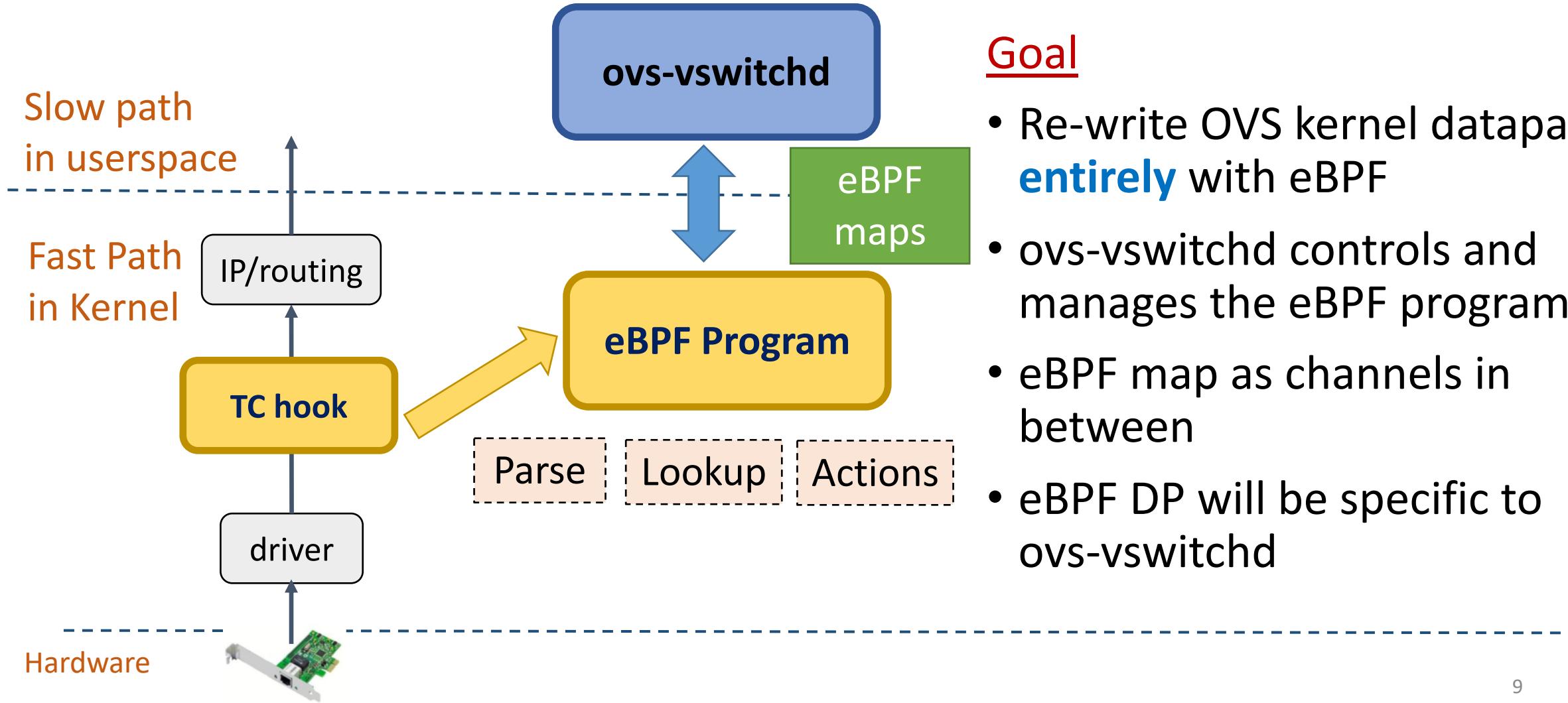
- Maintenance cost when adding a new datapath feature:
 - Time to upstream and time to backport
 - Maintain ABI compatibility between different kernel and OVS versions.
 - Different backported kernel, ex: RHEL, grsecurity patch
 - Bugs in compat code are often non-obvious to fix
- Implement datapath functionalities in eBPF
 - More stable ABI and guarantee to run in newer kernel
 - More opportunities for experiments / innovations



What is eBPF?

- An in-kernel virtual machine
 - Users can load its program and attach to a specific hook point in kernel
 - Safety guaranteed by BPF verifier
 - Attach points: network, trace point, driver, ... etc
- Maps
 - Efficient key/value store resides in kernel space
 - Can be shared between eBPF prorgam and user space applications
- Helper Functions
 - A core kernel defined set of functions for eBPF program to retrieve/push data from/to the kernel

OVS-eBPF Project



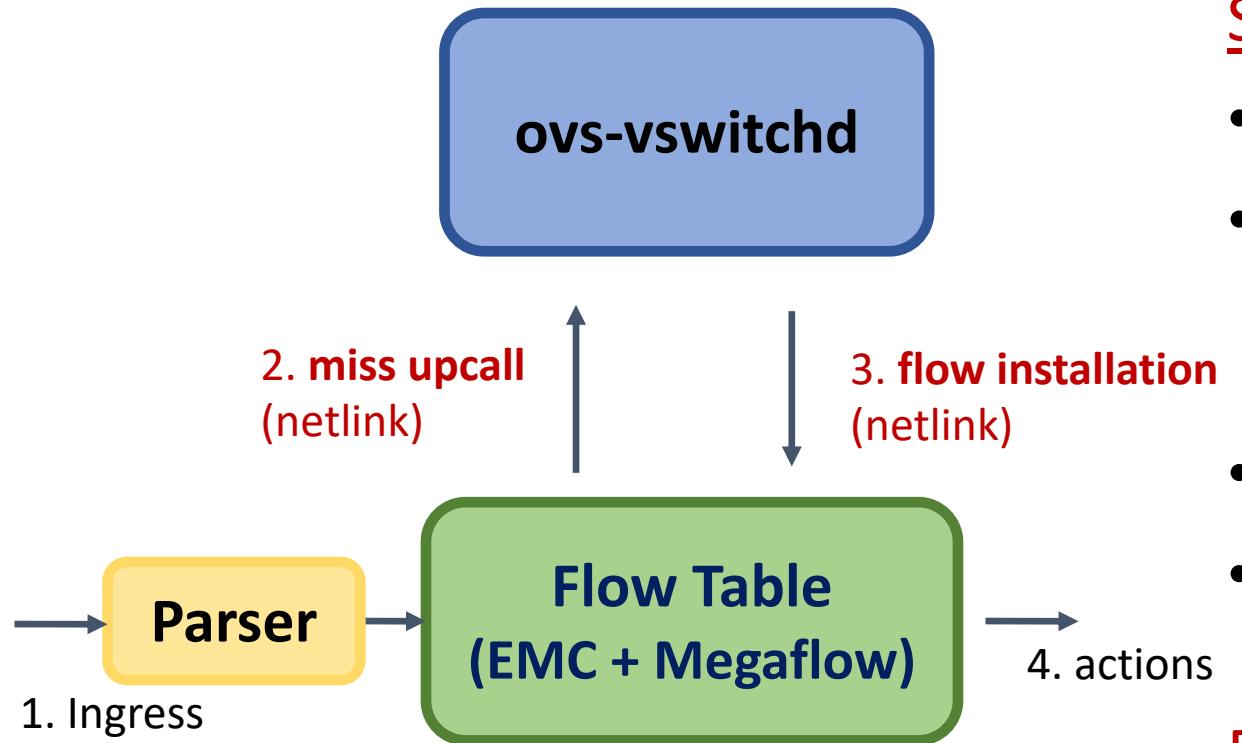
Headers/Metadata Parsing

- Define a flow key similar to struct sw_flow_key in kernel
- Parse protocols on packet data
- Parse metadata on struct __sk_buff
- Save flow key in per-cpu eBPF map

Difficulties

- Stack is heavily used (max: 512-byte, sw_flow_key: 464-byte)
- Program is very branchy

Review: Flow Lookup in Kernel Datapath



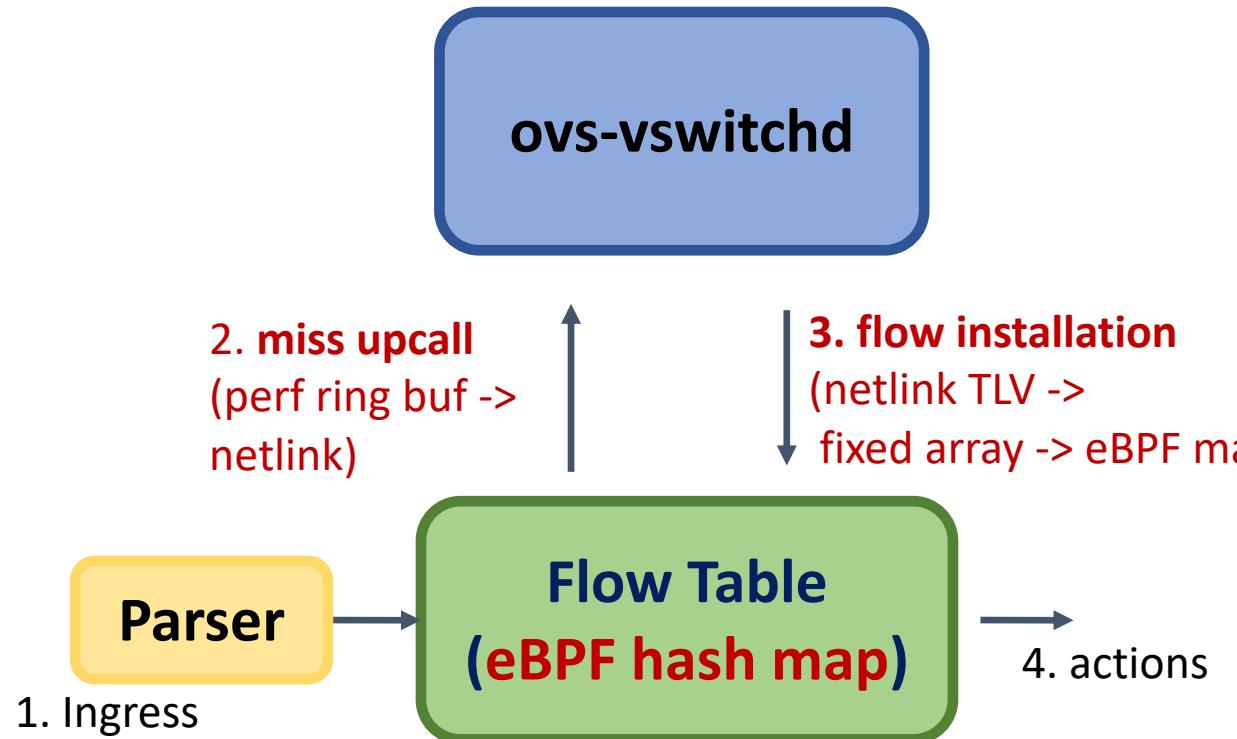
Slow Path

- Ingress: lookup miss and upcall
- ovs-vswitchd receives, does flow translation, and programs flow entry into flow table in OVS kernel module
- OVS kernel DP installs the flow entry
- OVS kernel DP receives and executes actions on the packet

Fast Path

- Subsequent packets hit the flow cache

Flow Lookup in eBPF Datapath



Limitation on flow installation:

TLV format currently not supported in BPF verifier

Solution: Convert TLV into fixed length array

Slow Path

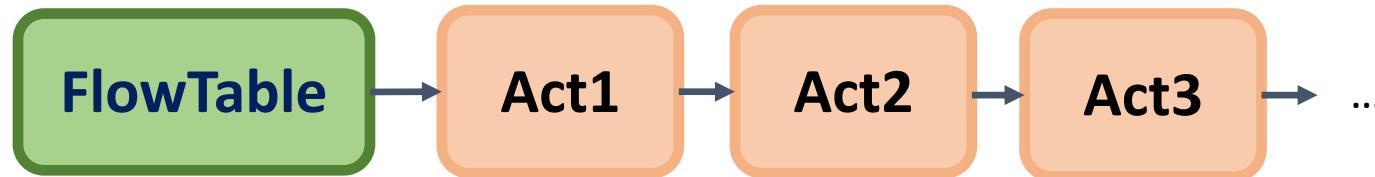
- Ingress: lookup miss and upcall
- Perf ring buffer carries packet and its metadata to ovs-vswitchd
- ovs-vswitchd receives, does flow translation, and programs flow entry into **eBPF map**
- ovs-vswitchd sends the packet down to trigger lookup again

Fast Path

- Subsequent packets hit flow in **eBPF map**

Review: OVS Kernel Datapath Actions

A list of actions to execute on the packet



Example cases of DP actions

- Flooding:
 - Datapath actions= output:9,output:5,output:10,...
- Mirror and push vlan:
 - Datapath actions= output:3,push_vlan(vid=17,pcp=0),output:2
- Tunnel:
 - Datapath actions:
`set(tunnel(tun_id=0x5,src=2.2.2.2,dst=1.1.1.1,ttl=64,flags(df|key))),output:1`

eBPF Datapath Actions

A list of actions to execute on the packet



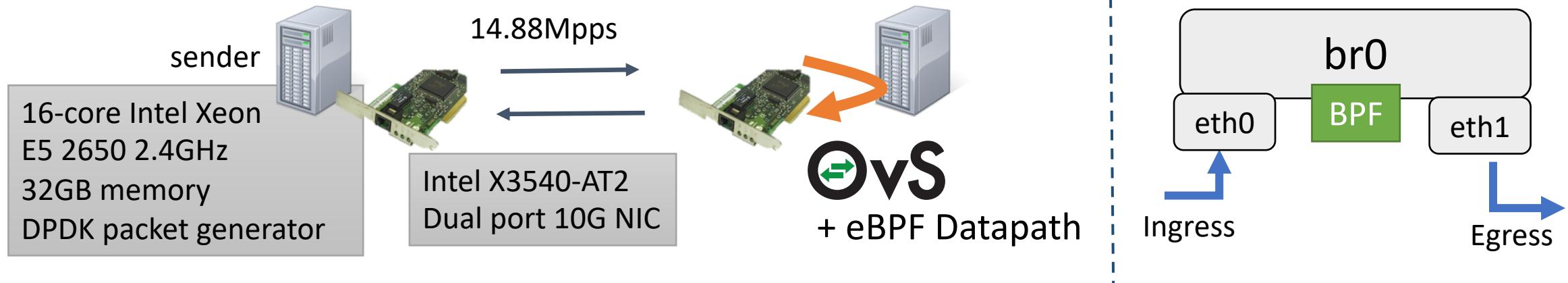
Challenges

- Limited eBPF program size (maximum 4K instructions)
- Variable number of actions: BPF disallows loops to ensure program termination

Solution:

- Make each action type an eBPF program, and tail call the next action
- Side effects: tail call has **limited context** and **does not return**
- Solution: keep **action metadata** and **action list** in a map

Performance Evaluation



- Sender sends 64Byte, 14.88Mpps to one port, measure the receiving packet rate at the other port
- OVS receives packets from one port, forwards to the other port
- Compare OVS kernel datapath and eBPF datapath
- Measure single flow, single core performance with Linux kernel 4.9-rc3 on OVS server

OVS Kernel and eBPF Datapath Performance

OVS Kernel DP Actions	Mpps	eBPF DP Actions	Mpps
Output	1.34	Redirect (no parser, lookup, actions)	1.90
Set dst_mac + Output	1.23	Output	1.12
Set GRE tunnel + Output	0.57	Set dst_mac + Output	1.14
		Set GRE tunnel + Output	0.48

All measurements are based on single flow, single core.

Conclusion and Future Work

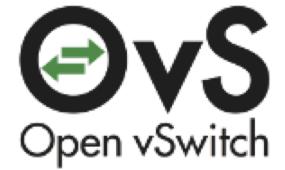
Features

- Megaflow support and basic conntrack in progress
- Packet (de)fragmentation and ALG under discussion

Lesson Learned

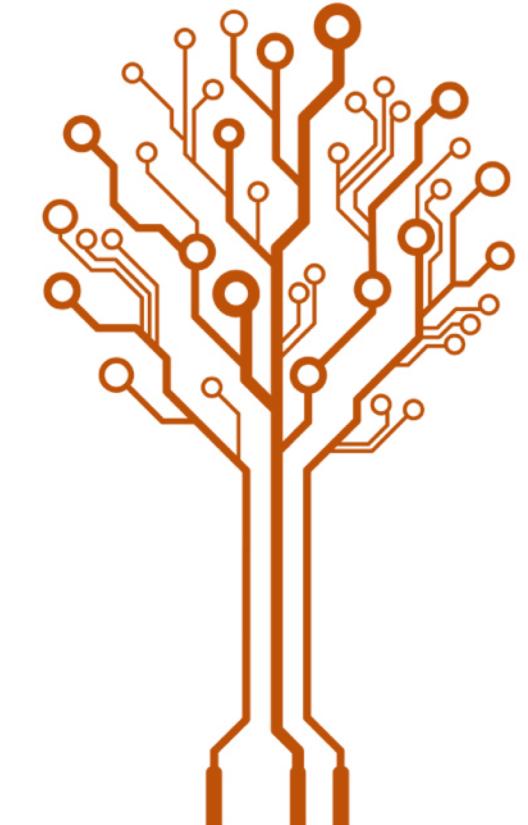
- Taking existing features and converting to eBPF is hard
- OVS datapath logic is difficult

OVS-AF_XDP

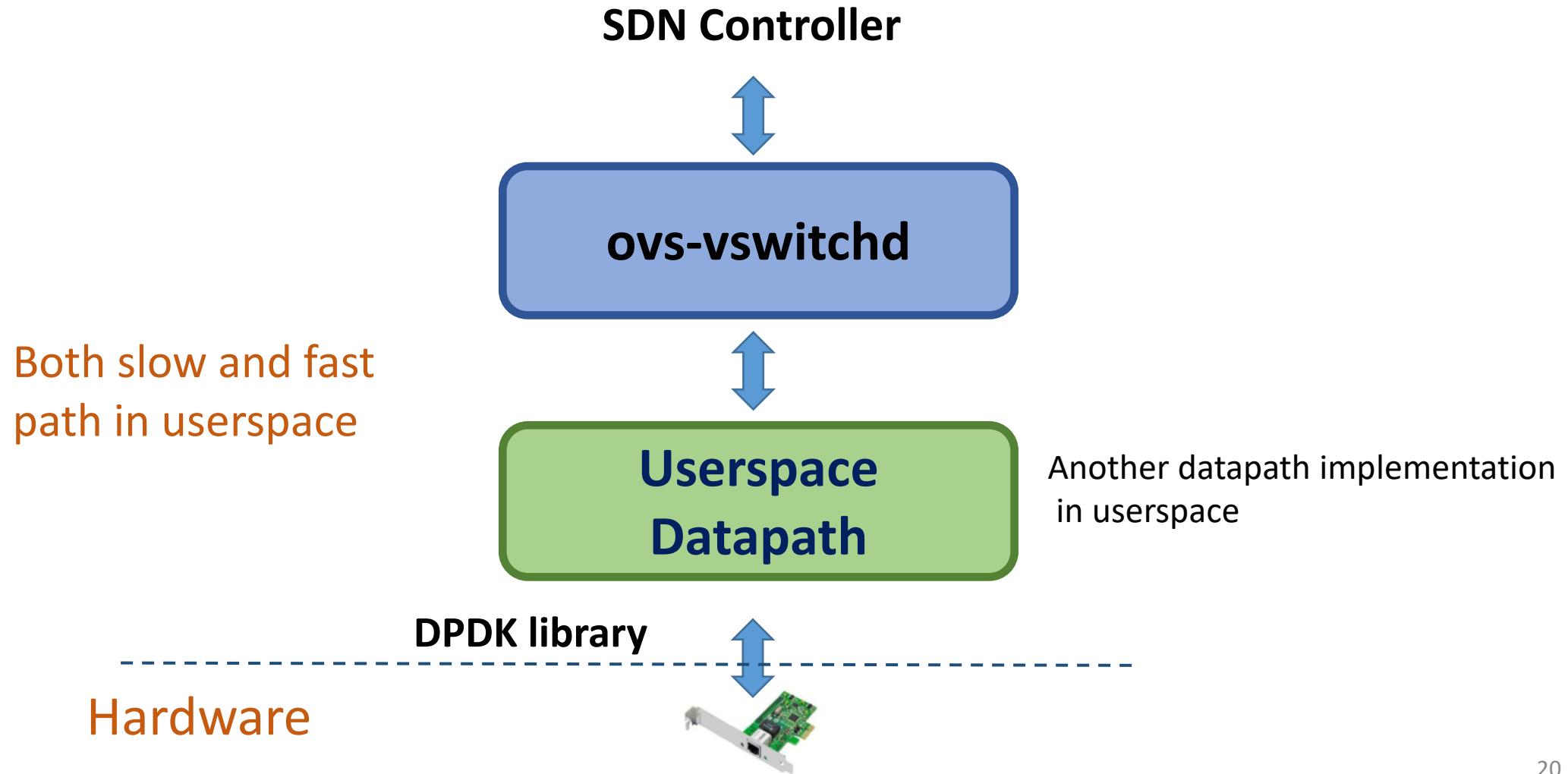


OVS-AF_XDP Motivation

- Pushing all OVS datapath features into eBPF is not easy
 - A large flow key on stack
 - Variety of protocols and actions
 - Dynamic number of actions applied for each flow
- **Ideas**
 - Retrieve packets from kernel as fast as possible
 - Do the rest of the processing in userspace
- **Difficulties**
 1. Reimplement all features in userspace
 2. Performance

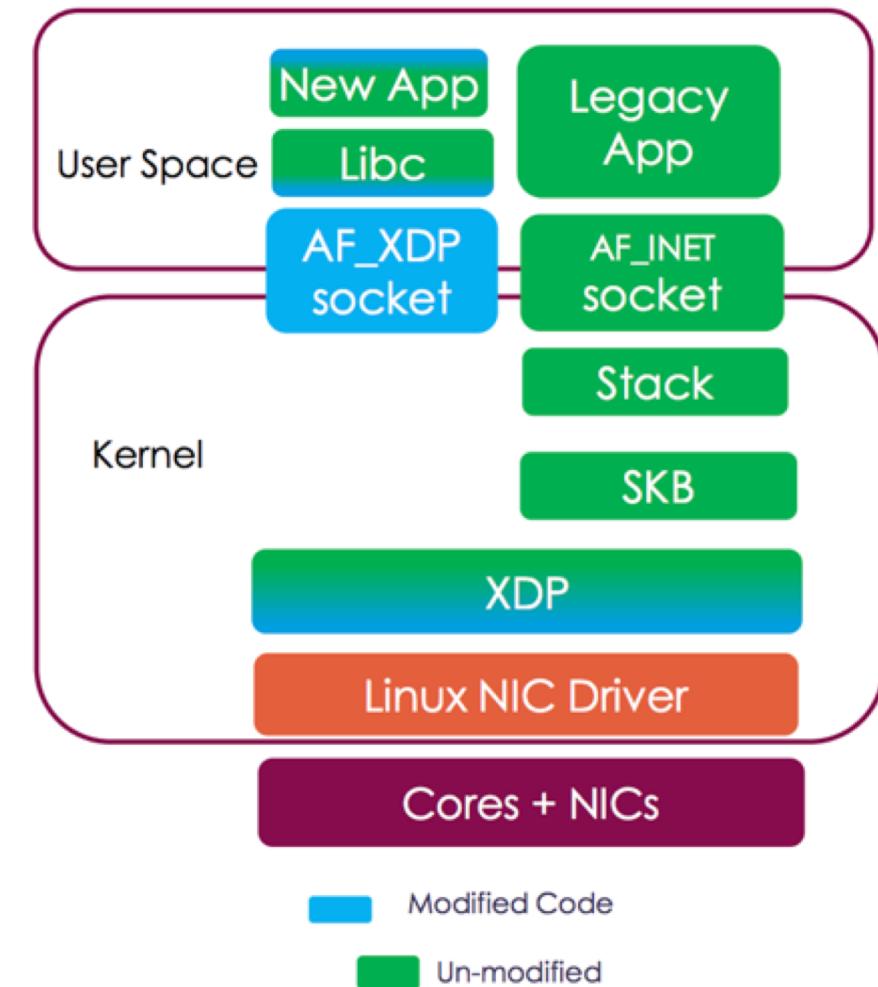


OVS Userspace Datapath (dpif-netdev)



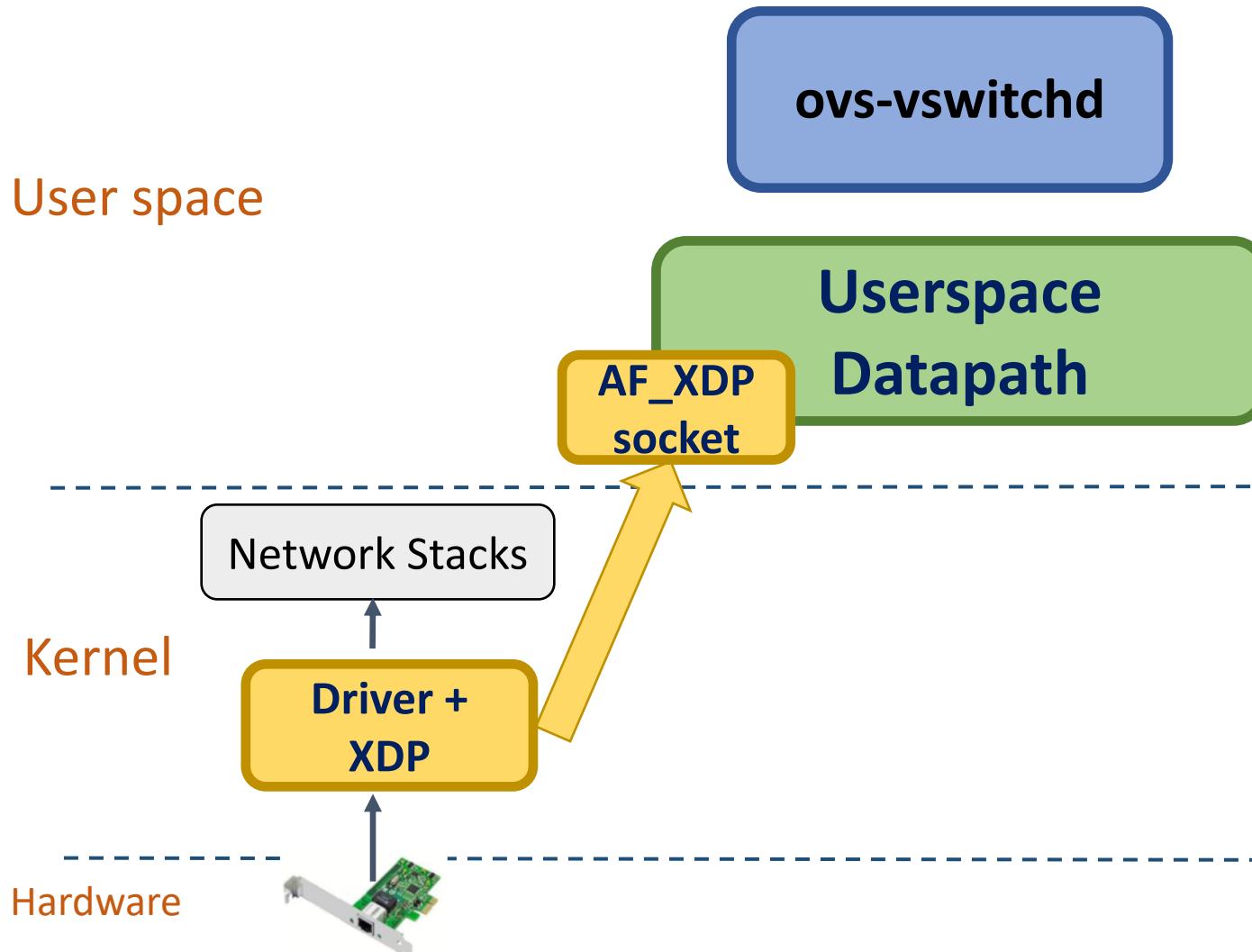
XDP and AF_XDP

- XDP: eXpress Data path
 - An eBPF hook point at the network device driver level
- 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, **achieving line rate (14Mpps)!**



From “DPDK PMD for AF_XDP”

OVS-AF_XDP Project

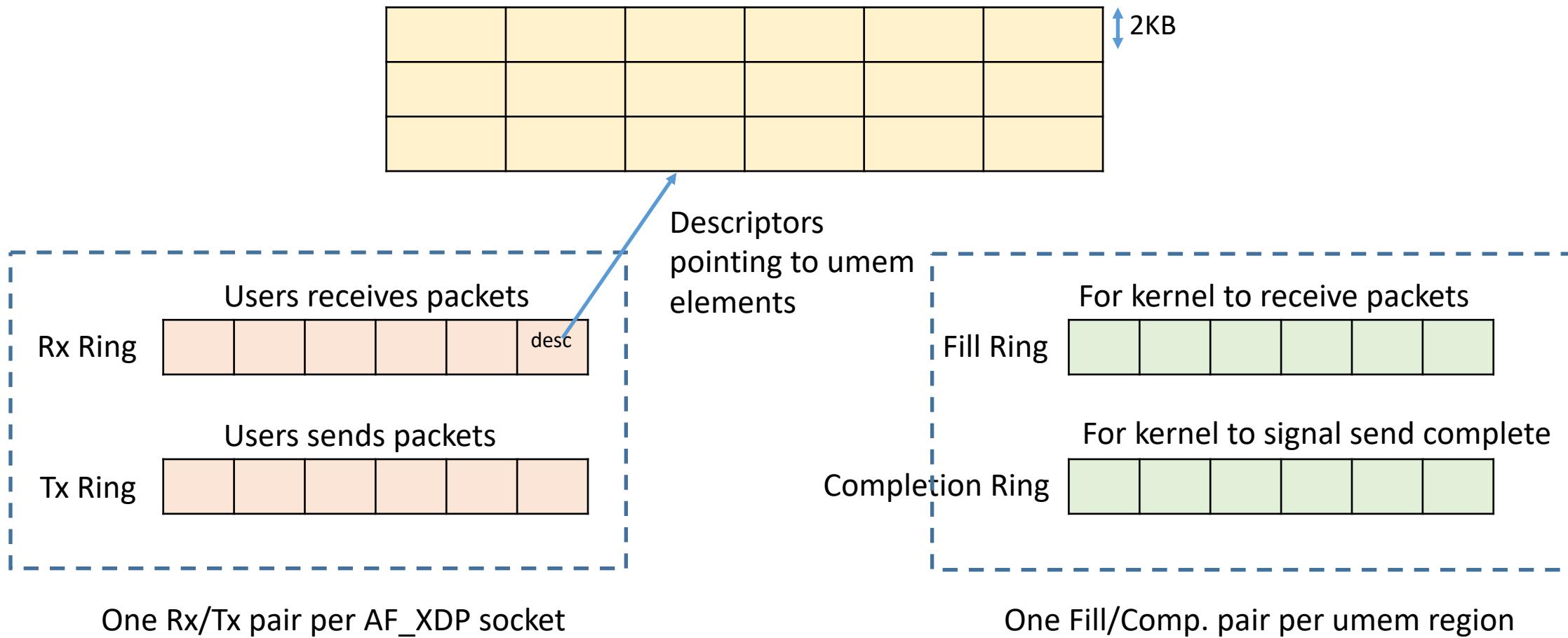


Goal

- Use AF_XDP socket as a fast channel to userspace OVS datapath
- Flow processing happens in userspace

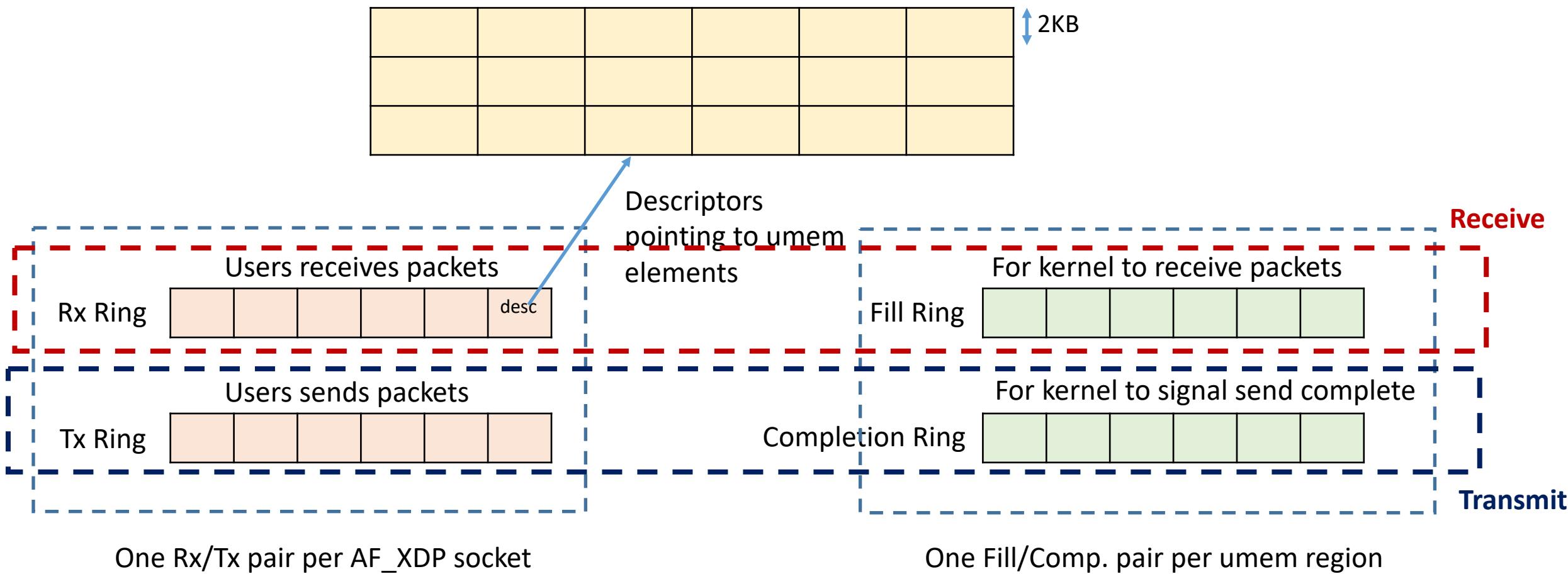
AF_XDP umem and rings Introduction

umem memory region: multiple 2KB chunk elements

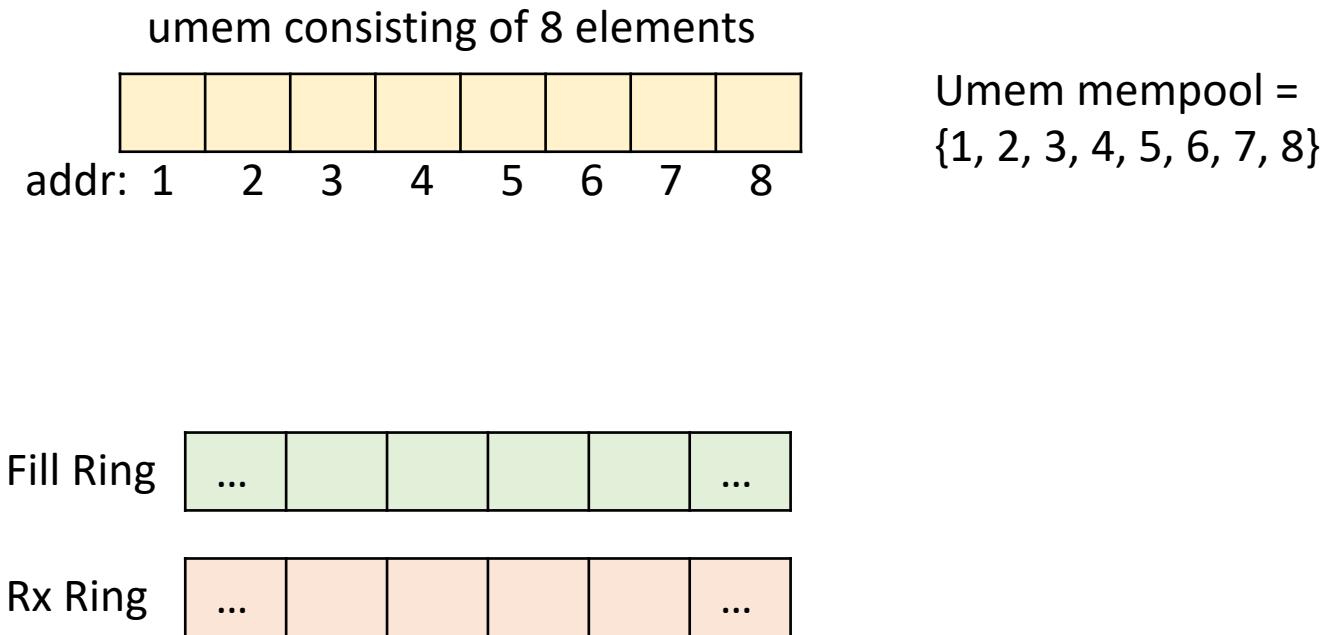


AF_XDP umem and rings Introduction

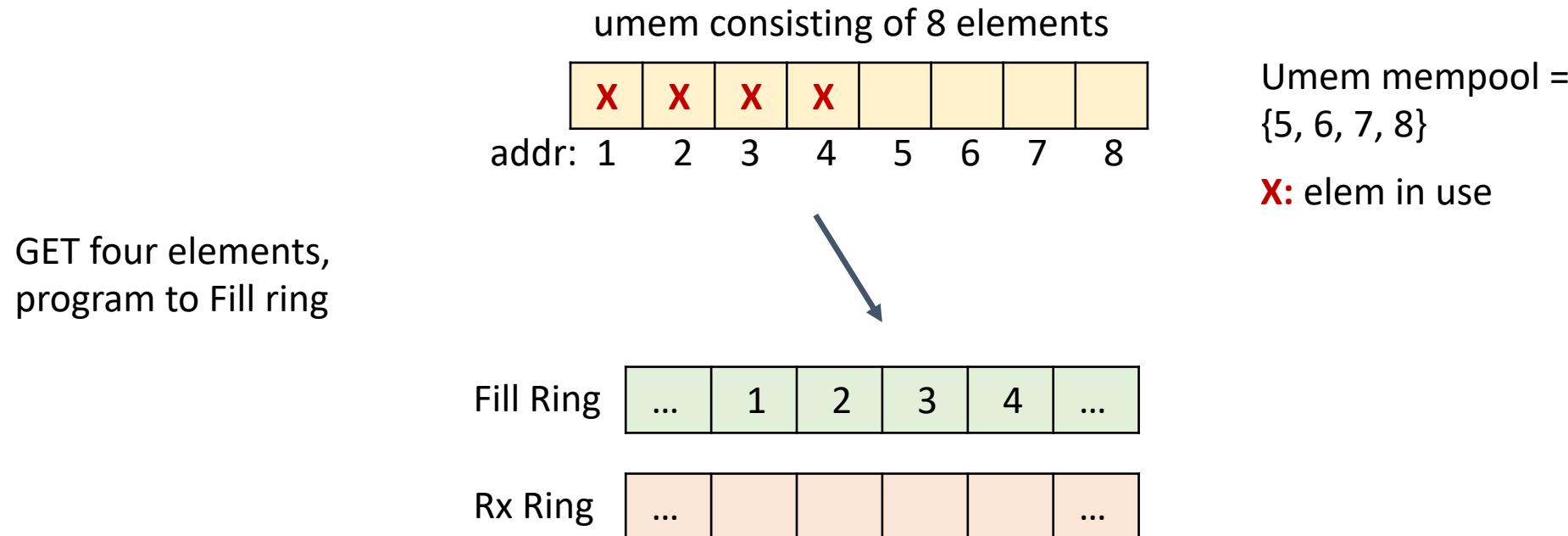
umem memory region: multiple 2KB chunk elements



OVS-AF_XDP: Packet Reception (0)

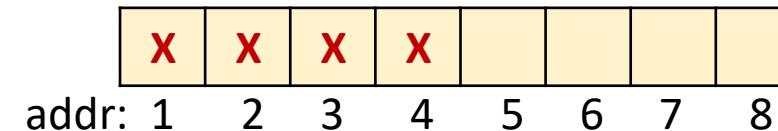


OVS-AF_XDP: Packet Reception (1)



OVS-AFXDP: Packet Reception (2)

umem consisting of 8 elements



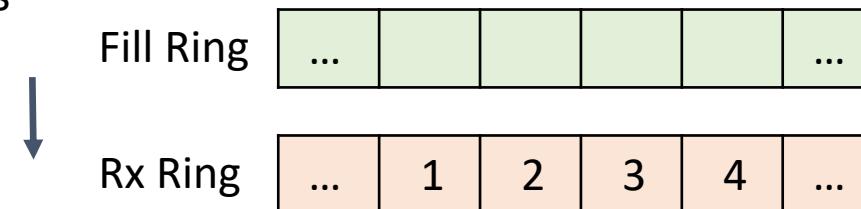
Umem mempool =
 $\{5, 6, 7, 8\}$

X: elem in use

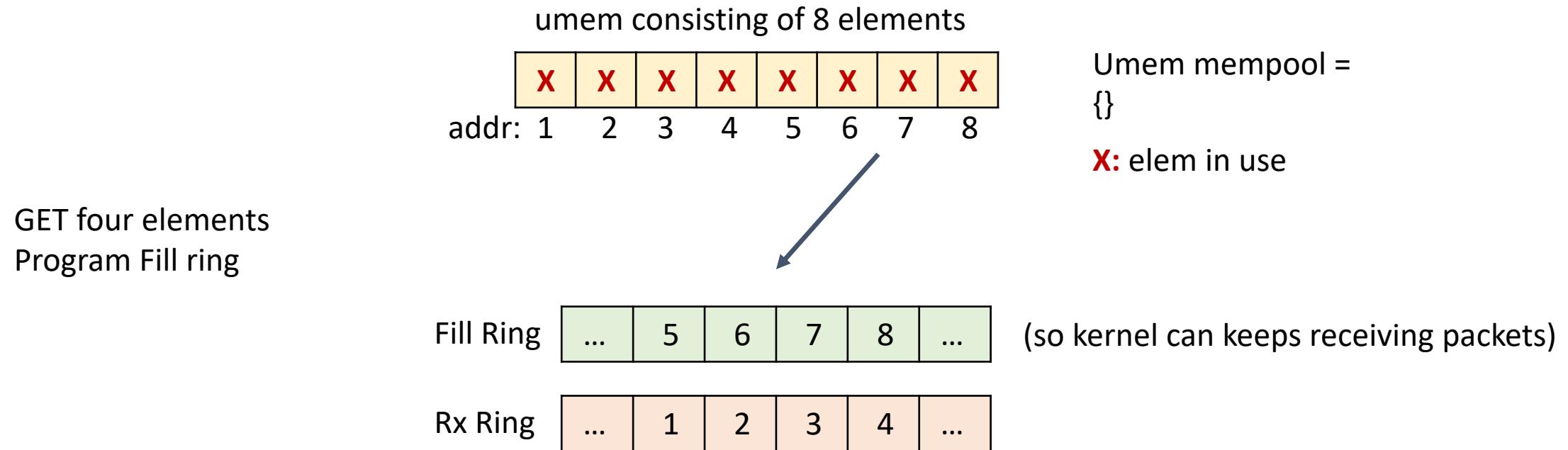
Kernel receives four packets

Put them into the four umem chunks

Transition to Rx ring for users

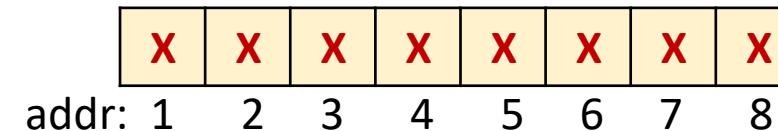


OVS-AFXDP: Packet Reception (3)



OVS-AFXDP: Packet Reception (4)

umem consisting of 8 elements



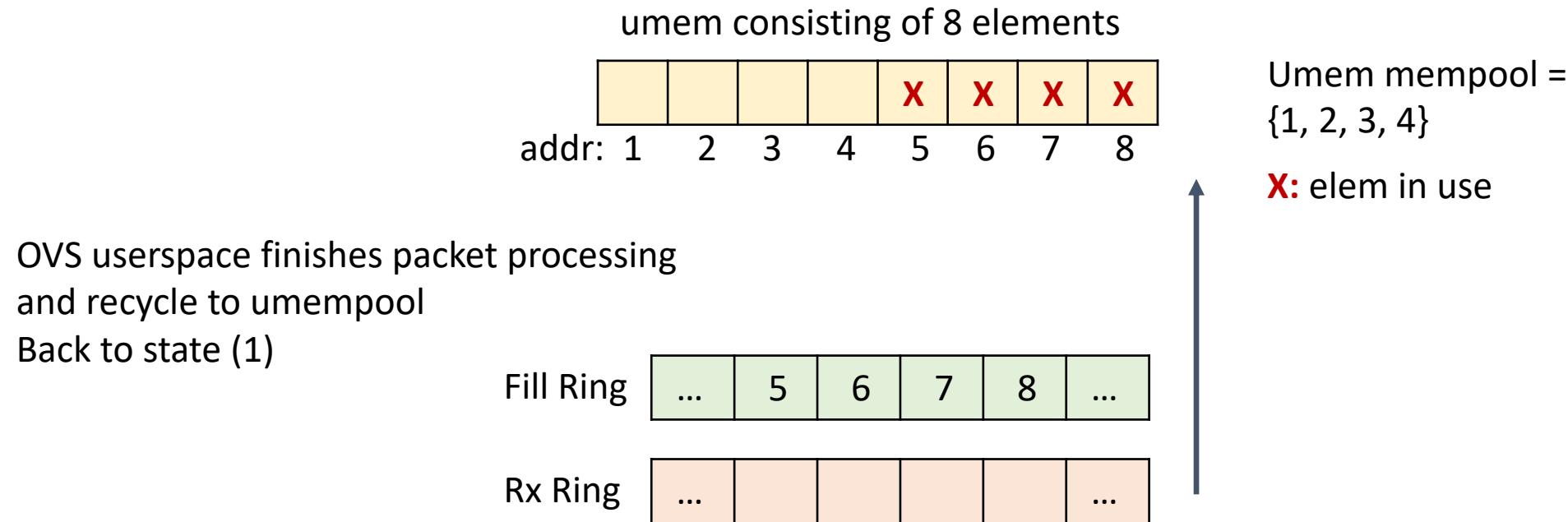
Umem mempool =
 {}

X: elem in use

OVS userspace processes packets
on Rx ring



OVS-AFXDP: Packet Reception (5)



Optimizations

- OVS pmd (Poll-Mode Driver) netdev for rx/tx
 - Before: call poll() syscall and wait for new I/O
 - After: dedicated thread to busy polling the Rx ring
- UMEM memory pool
 - Fast data structure to GET and PUT umem elements
- Packet metadata allocation
 - Before: allocate md when receives packets
 - After: pre-allocate md and initialize it
- Batching sendmsg system call

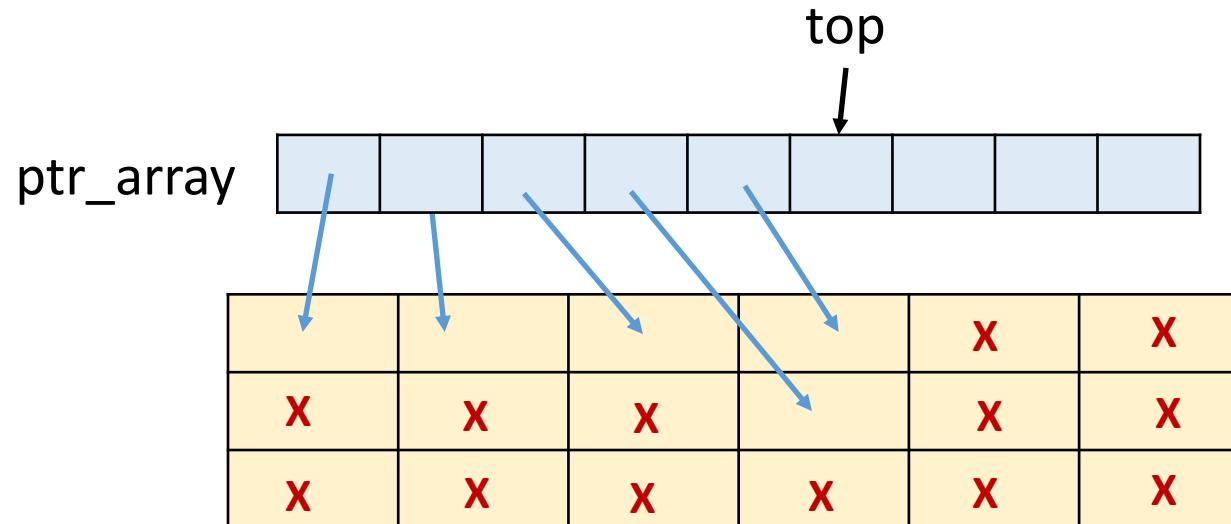
Umempool Design

- Umempool: A freelist keeps tracks of free buffers
 - GET: take out N umem elements
 - PUT: put back N umem elements
- Every ring access need to call umem element GET/PUT

Three designs:

- LILO-List_head: embed in umem buffer, linked by a list_head, push/pop style
- FIFO-ptr_ring: a pointer ring with head and tail pointer
- **LIFO-ptr_array**: a pointer array and push/pop style access (**BEST!**)

LIFO-ptr_array Design



Multiple 2K umem chunk memory region

Idea:

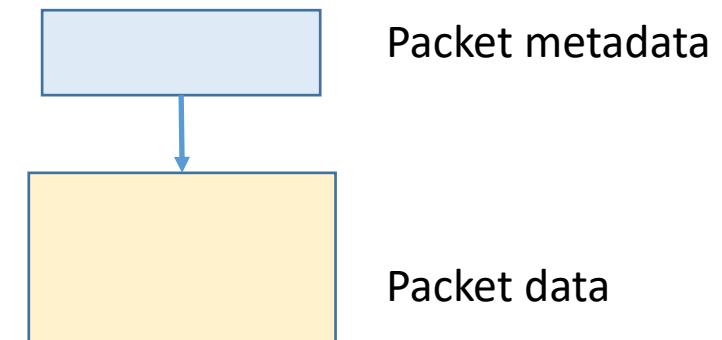
- Each ptr_array element contains a umem address
- Producer: PUT elements on top and top++
- Consumer: GET elements from top and top--

Packet Metadata Allocation

- Every packets in OVS needs metadata: struct dp_packet
- Initialize the packet data independent fields

Two designs:

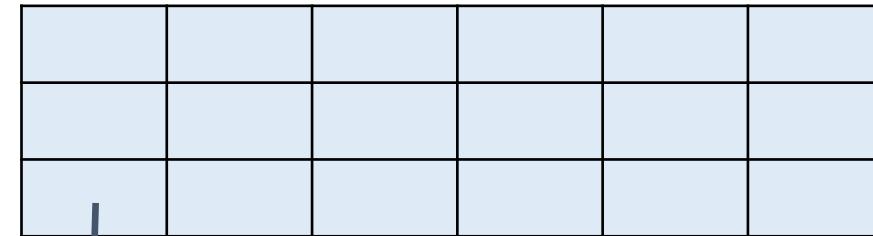
1. Embedding in umem packet buffer:
 - Reserve first 256-byte for struct dp_packet
 - Similar to DPDK mbuf design
2. Separate from umem packet buffer:
 - Allocate an array of struct dp_packet
 - Similar to skb_array design



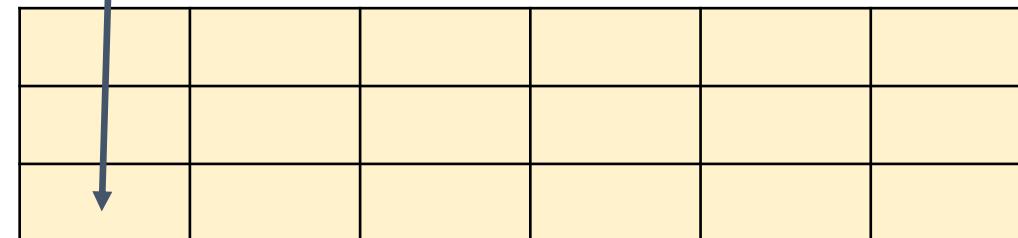
Packet Metadata Allocation

Separate from umem packet buffer

Packet metadata in another memory region

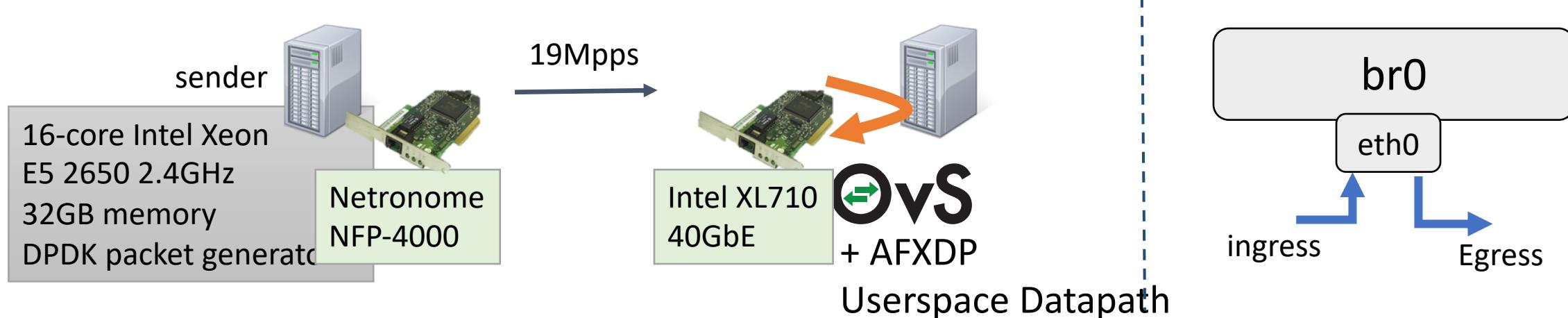


One-to-one maps to umem



Multiple 2K umem chunk memory region

Performance Evaluation



- Sender sends 64Byte, 19Mpps to one port, measure the receiving packet rate at the other port
- Measure single flow, single core performance with Linux kernel 4.19-rc3 and OVS 2.9
- Enable AF_XDP Zero Copy mode

Performance Evaluation

Experiments

- OVS-AFXDP
 - rxdrop: parse, lookup, and action = drop
 - L2fwd: parse, lookup, and action = set_mac, output to the received port
- XDP SOCK: AF_XDP benchmark tool
 - rxdrop/l2fwd: simply drop/fwd without touching packets
- LIFO-ptr_array + separate md allocation shows the best

Results

	XDP SOCK	OVS-AFXDP	Linux Kernel
rxdrop	19Mpps	19Mpps	< 2Mpps
l2fwd	17Mpps	14Mpps	< 2Mpps

Conclusion and Discussion

Future Work

- Try virtual devices vhost/virtio with VM-to-VM traffic
- Bring feature parity between userspace and kernel datapath

Discussion

- Balance CPU utilization of pmd/non-pmd
- Comparison with DPDK in terms of deployment difficulty

Comparison

	OVS-eBPF	OVS-AF_XDP	OVS Kernel Module
Maintenance Cost	Low	Low	High
Performance	Comparable with kernel	High with cost of CPU	Standard (< 2Mpps)
Development Efforts	High	Low	Medium
New feature deployment	Easy	Easy	Hard due to ABI change
Safety	High due to verifier	Depends on reviewers	Depends on reviewers

Question?

Thank You

