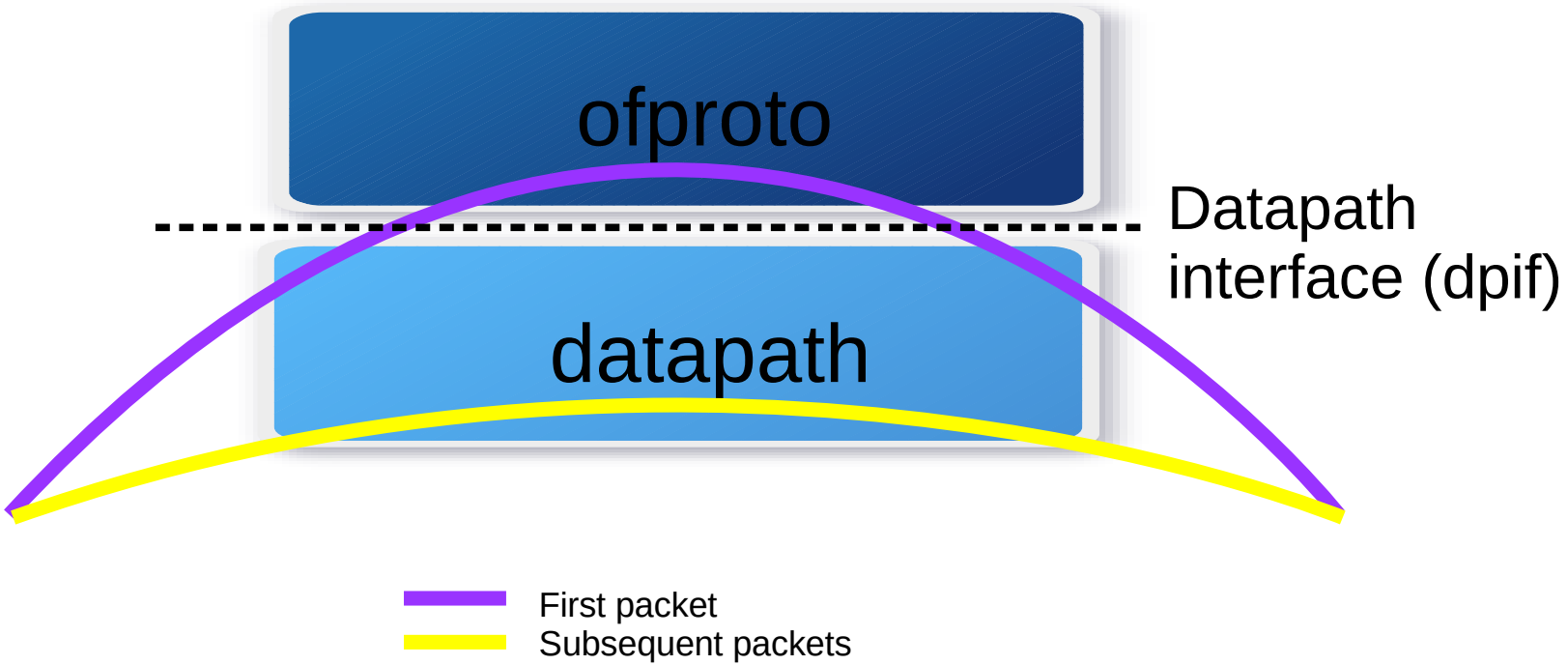
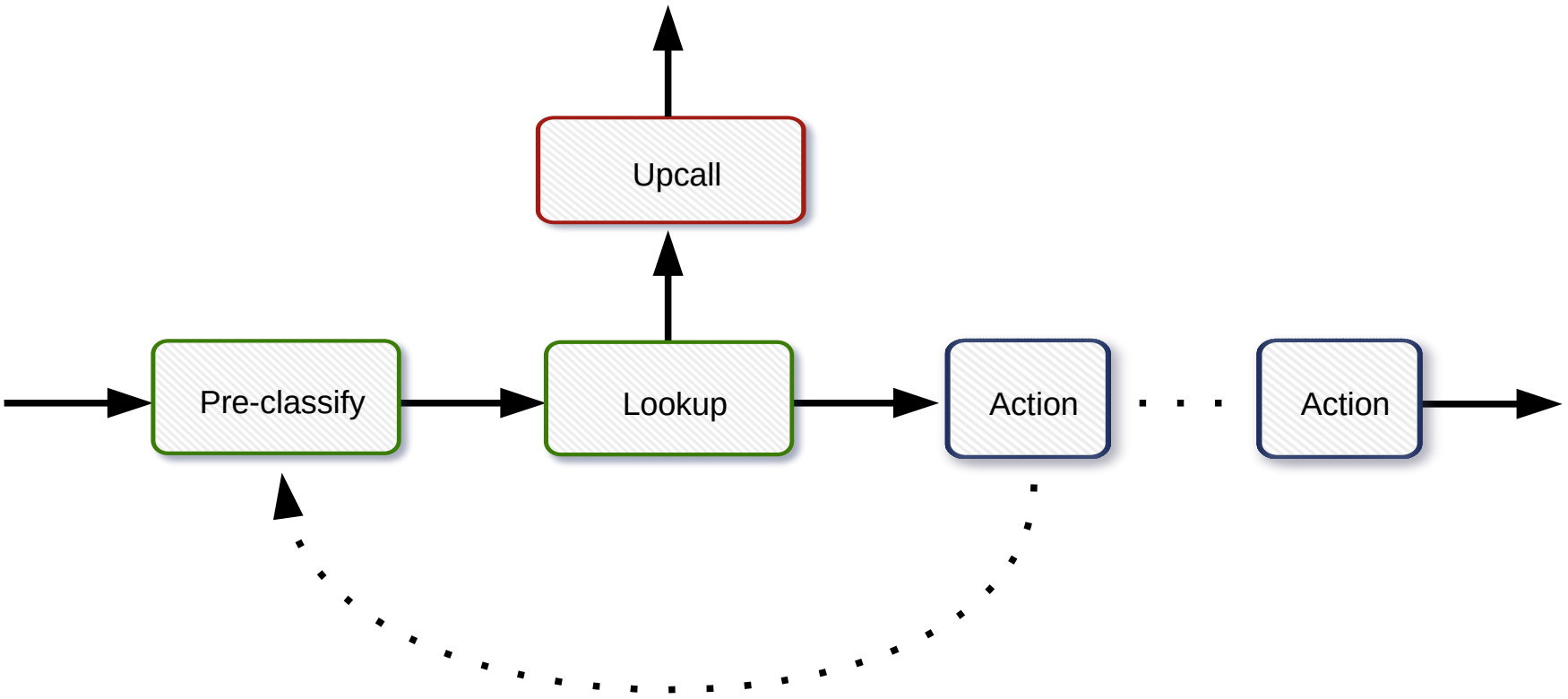
OVS Hardware offloads Discussion Panel

Joe Stringer
VMware

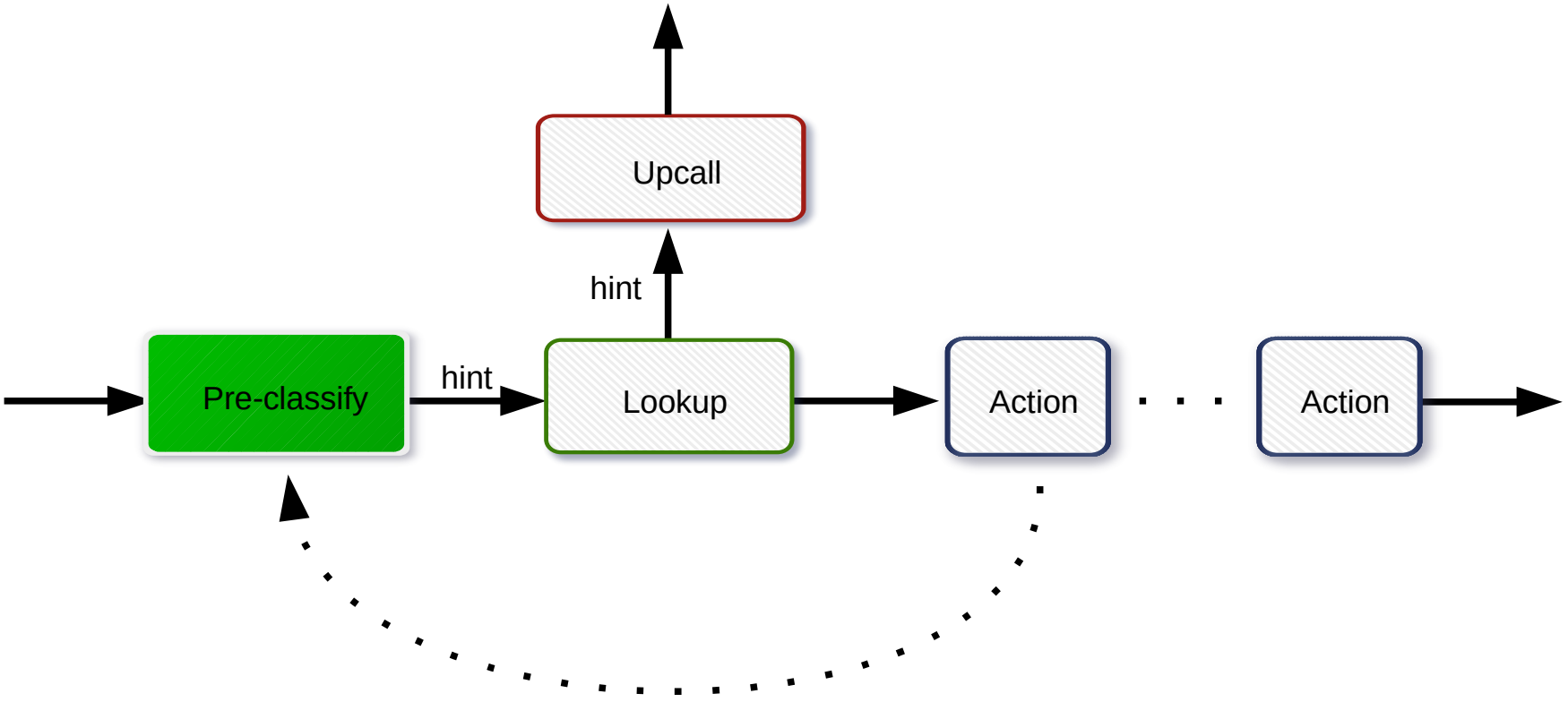
OVS caching model



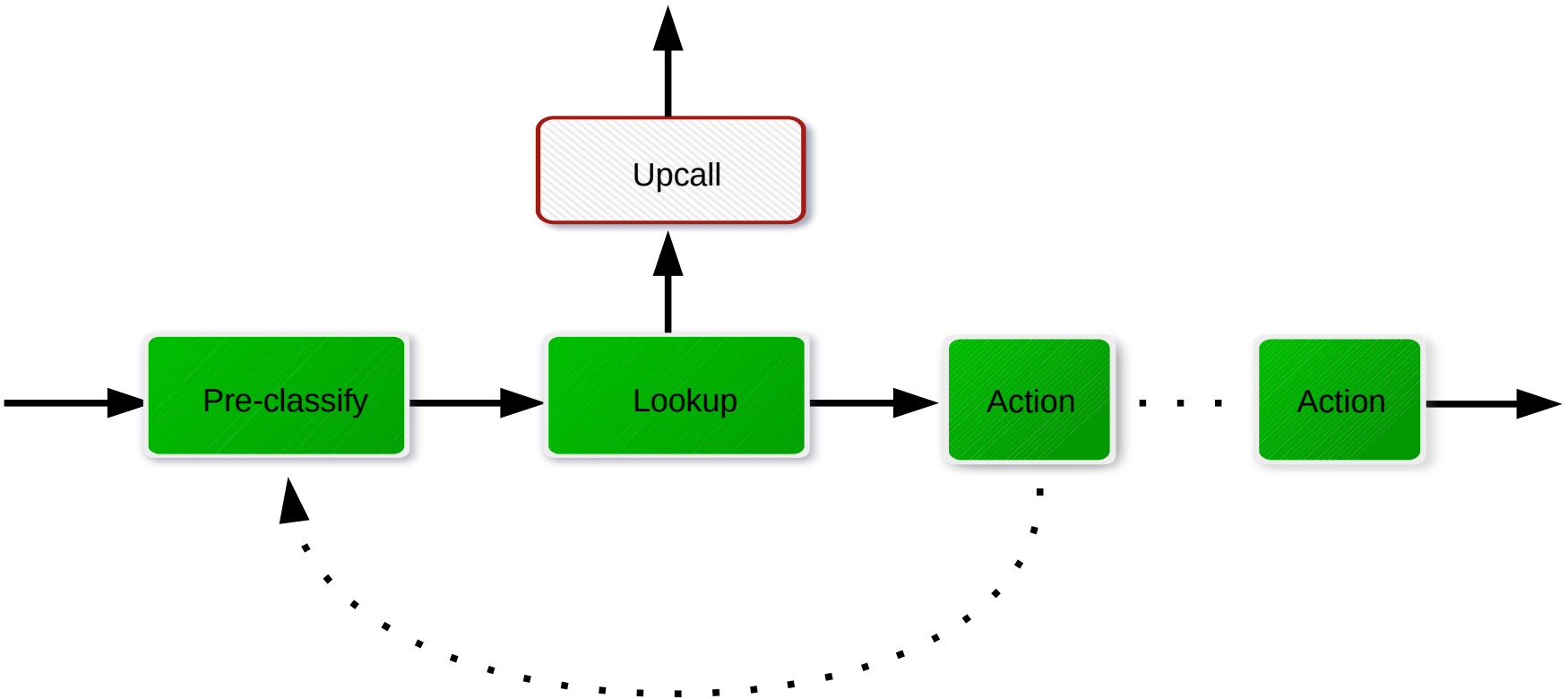
Datapath



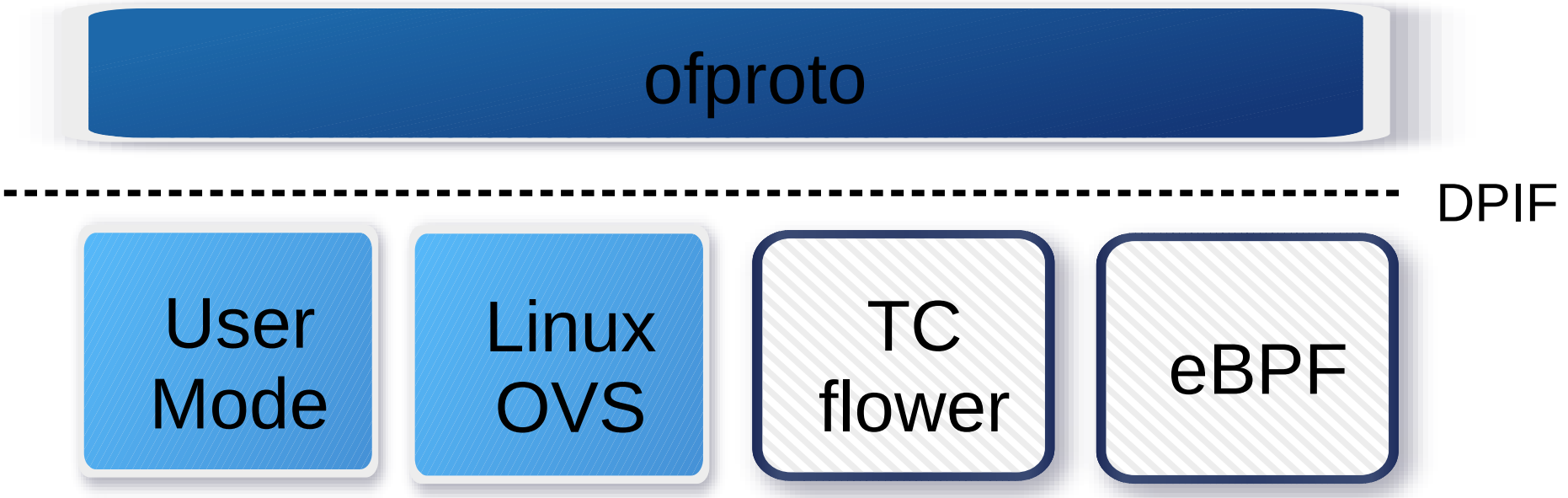
Partial Offload



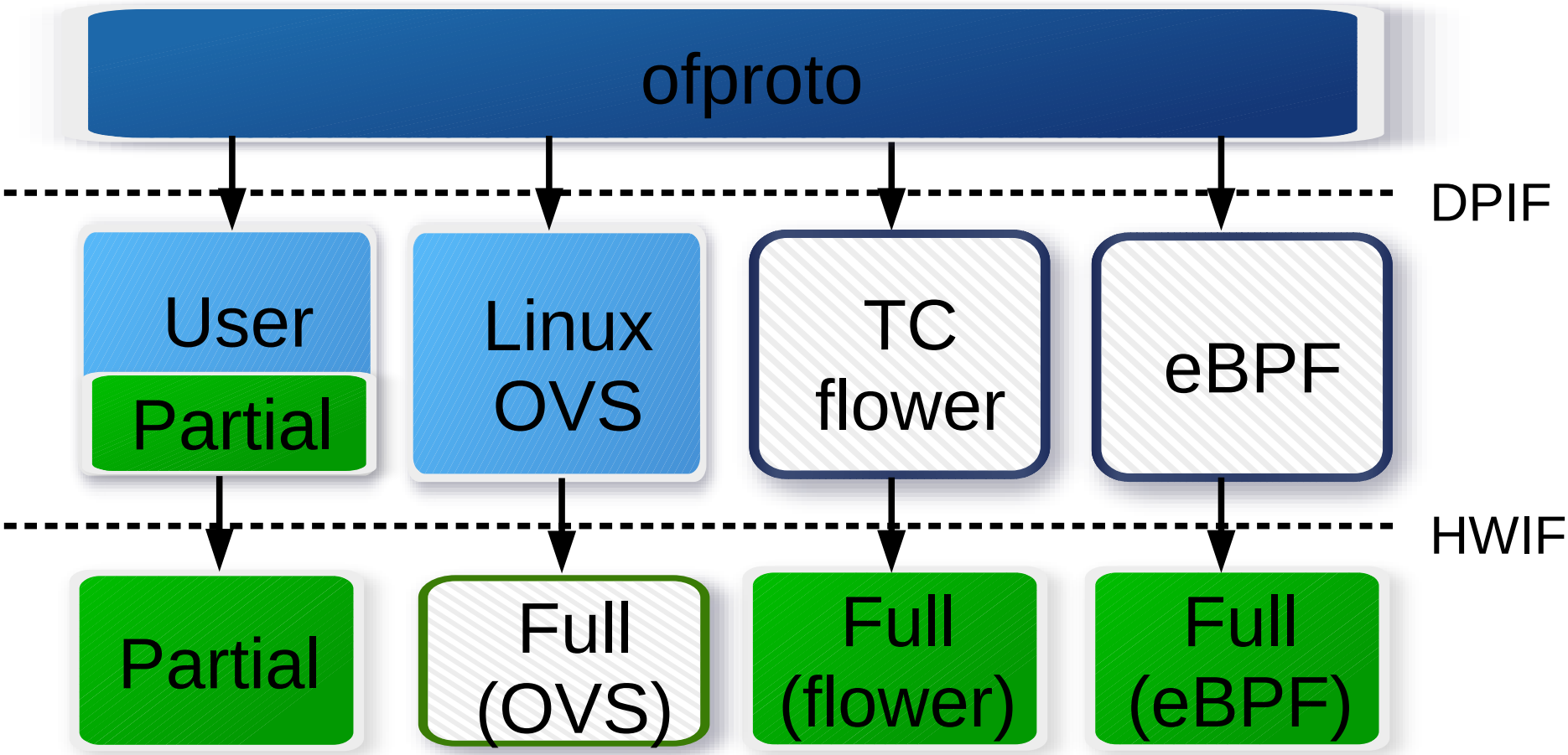
Full Offload



Datapath implementations



Flow offloads



SoftFlow TCAM Offload

Ethan J. Jackson

Tradeoff

- Pure Software
  - Highly Flexible
  - Slow
- Full Hardware Offload
  - Rigid
  - Fast

Key Insight

- Vast Majority of CPU time spend on packet classification
- So, let's just offload packet classification
  - Much faster
  - Still Highly Flexible

Implementation

- Treat NIC as an additional Flow Cache
- Assign megaflows an ID
- Offload frequently used megaflows to the NIC
- NIC writes the ID to packet metadata (if matched)
- Software checks the match was correct

Summary

- Significant performance improvement
  - 5% - 90%
- Without compromising software flexibility
  - Tolerant of limited hardware TCAM size
  - Evolve software features without requiring hardware changes

ASAP^2 - NIC HW Acceleration for Open-vSwitch

Rony Efraim

Open vSwitch 2016 Fall Conference

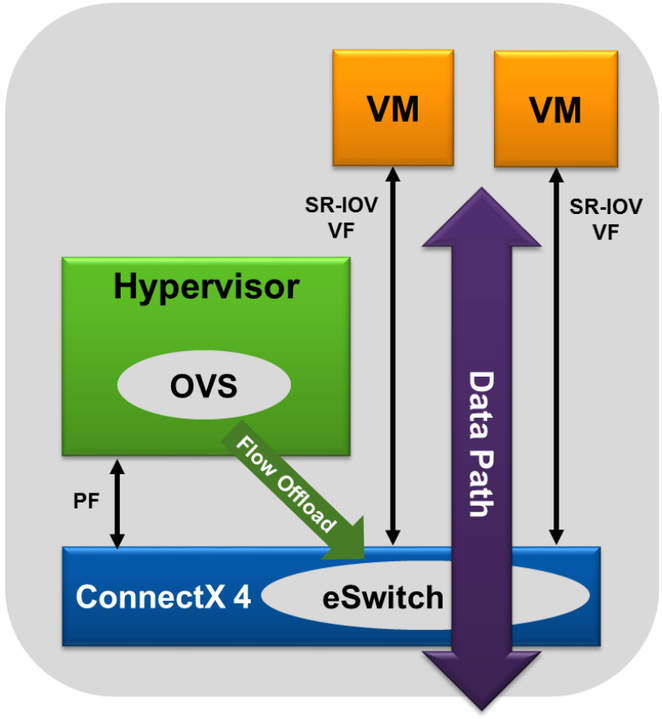
Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™

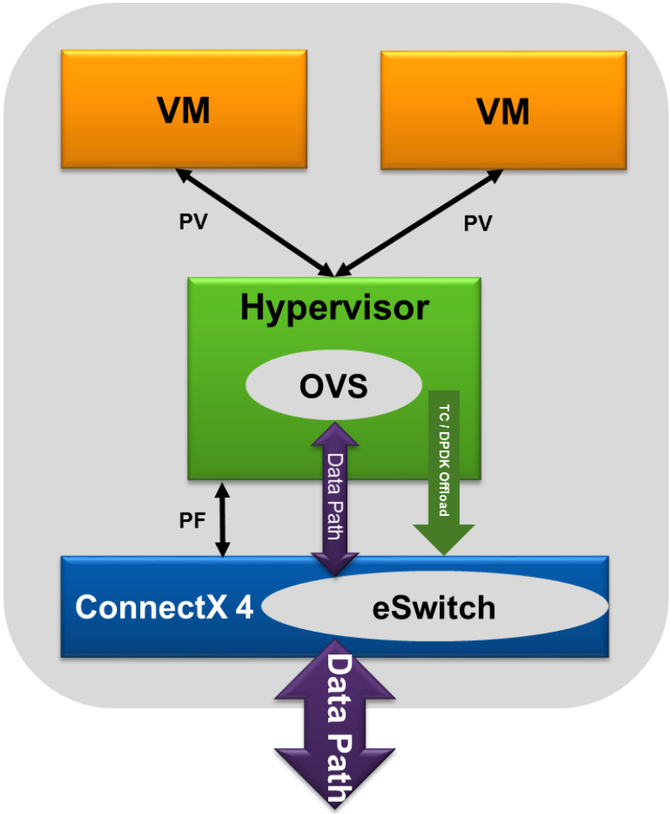
Accelerated Switch And Packet Processing (ASAP²)

- ASAP² takes advantage of ConnectX-4 capability to accelerate \ offload “in host” network stack
- Two main use cases:

ASAP² Direct
Full vSwitch offload (SR-IOV)



ASAP² Flex
vSwitch acceleration

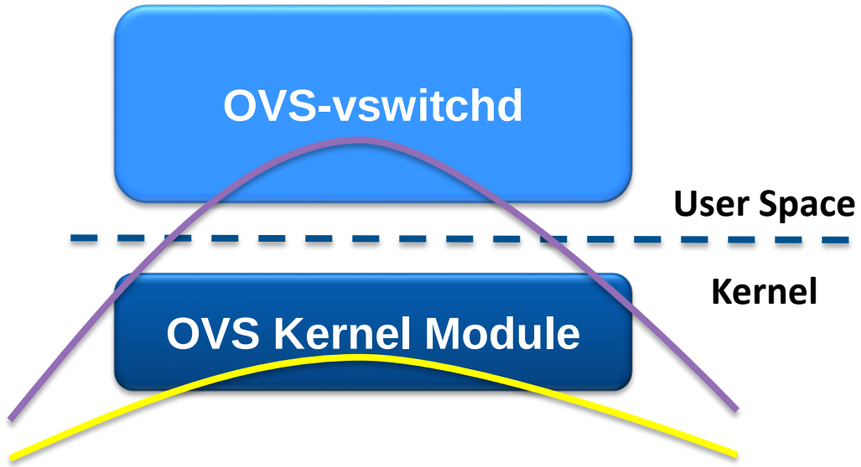


Full Virtual Switch Offload

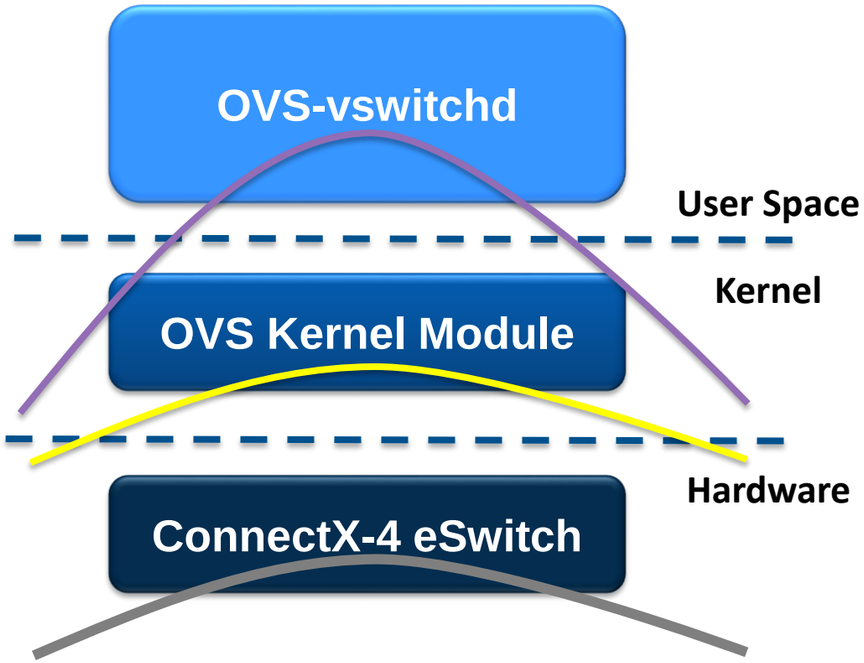
ASAP^2-Direct

Software based VS Hardware based

Traditional Model: All Software
High Latency, Low Bandwidth, CPU Intensive



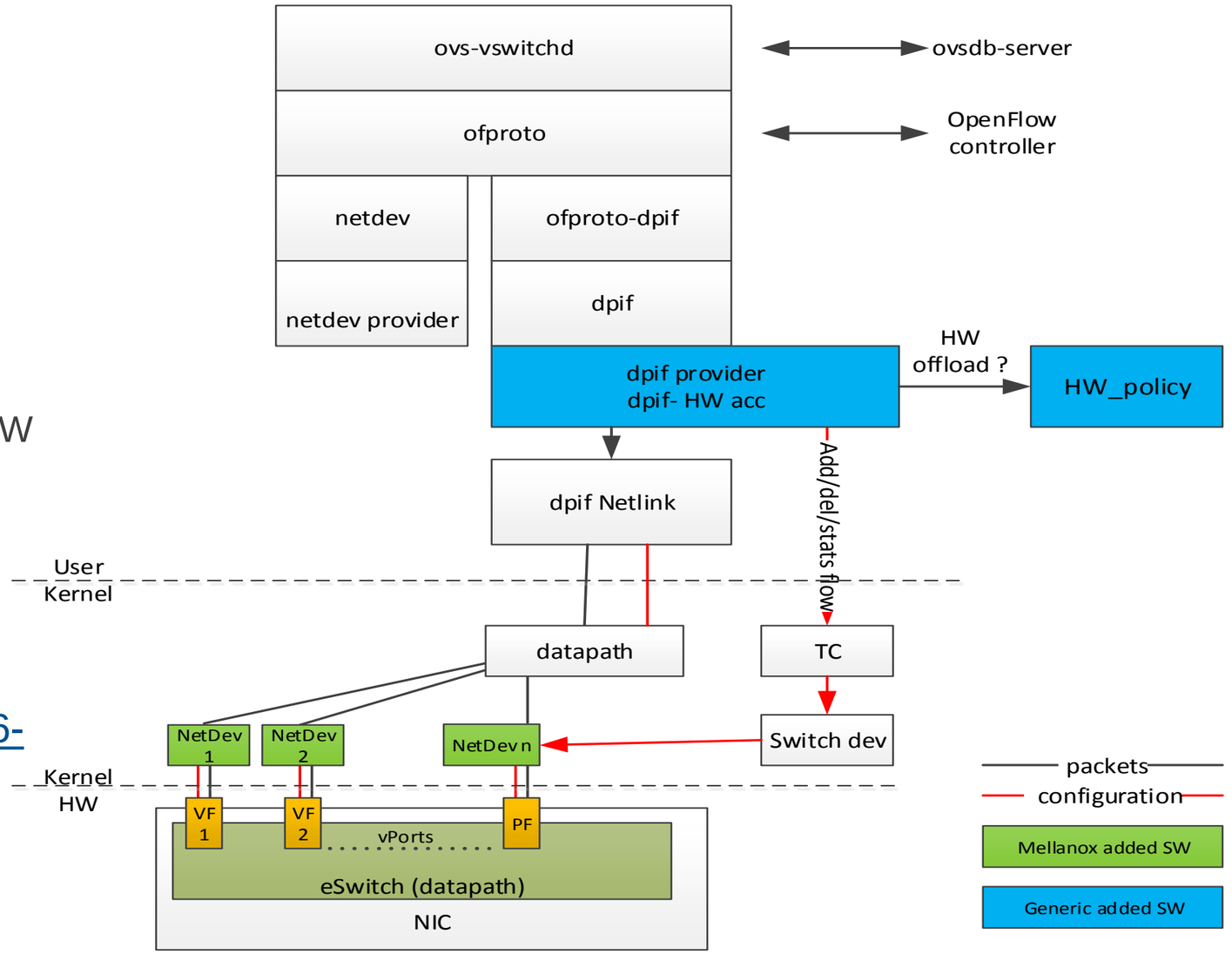
ConnectX-4: Hardware Offload
Low Latency, High Bandwidth, Efficient CPU



First flow packet Fallback FRWD path HW forwarded Packets

Policy Based Offloading

- Changes are made only in the OVS user space code (No OVS kernel datapath module changes)
- Plugin a new DPIF module
  - Same Northbound APIs
  - Same Southbound APIs in most cases
  - Add "*HW_offload_test_...*" APIs
    - Concentrate the Policy code in a specific SW module
    - Policy can decide on " HW ONLY/ NO OFFLOAD/ SPLIT"
  - Code is on the OVS ML http://openvswitch.org/pipermail/dev/2016-November/081141.html



Accelerated Virtual Switch

ASAP^2-Flex

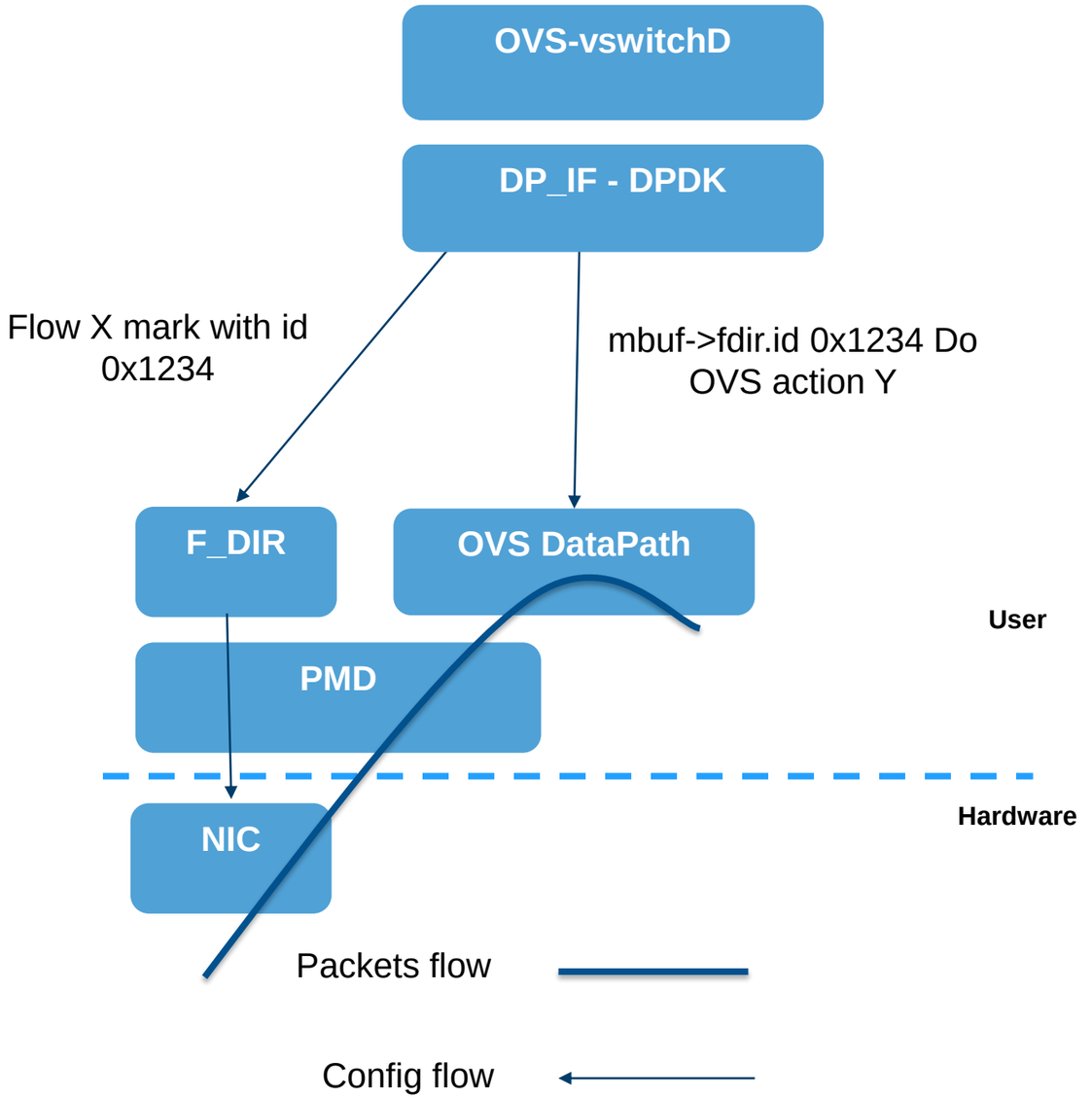
Concept

- Every switch (virtual or physical) has a notion of “packet processing pipeline”
  - (Push/pop vlan, Tunnel Encap/decap operations, QoS related functionality: (Metering, Shaping, Marking, Scheduling), Switching action)
- Typical ingress pipeline of a virtual switch can be:

- ASAP^2-Flex is a framework to offload part of the packet processing – one or more pipeline stages, onto the NIC HW engines
- the switching decision and Tx operation are left to the SW based dataplane
- Paravirt VMs will enjoy HW offload while actual switching decisions is done by OVS SW

openVswitch using HW classification

- For every OVS flow DP-if should use the DPDK filter (or TC) to classify with Action tag (report id) or drop.
- When receive use the tag id instead of classify the packet
- for Example :
  - OVS set action Y to flow X
    - Add a flow to tag with id 0x1234 for flow X
    - Config datapath to do action Y for mbuf->fdir.id = 0x1234
  - OVS action drop for flow Z
    - Use DPDK filter to drop and count flow Z
    - Use DPDK filter to get flow statistic



DPDK support

- DPDK uses Flow filters
- All current flow filters are either “fixed” or “RAW”
  - No filter support 12 tuple
  - No filter support different mask (mega flows)
  - No counter per flow , required for drop.

- Work in progress:
  - Define new filter type (RTE_ETH_FILTER_GENERIC)
  - Define Flow spec fields as a TLV
  - Define list of Actions for a matched Packet (as a TLV)
    - Flow_tag, Drop, count etc…
  - For more info: https://rawgit.com/6WIND/rte_flow/master/rte_flow.pdf

Thank You

Mellanox TECHNOLOGIES

Connect. Accelerate. Outperform.™

OVS Fall Conference
HW Offload Panel

Nic Viljoen - 7 November 2016

Considerations

NETRONOME

Hardware

- Low power fully programmable 72 core SmartNIC with 576 cooperatively multiplexed threads
- TCO (6x reduction) case for acceleration-more VM instances, higher throughput, lower latency

Current OVS offload approach: kernel datapath (megaflow / dpif) level, with/without conntrack

- Feature rich ‘Full Offload’ - offloads almost all OVS matching / actions, incl. tunnels and conntrack
- Fallback path (selective acceleration) improves compatibility and maintainability

Ready for eBPF based OVS in future

- Netronome have implemented transparent eBPF offload (for XDP and TC) upstream in Linux

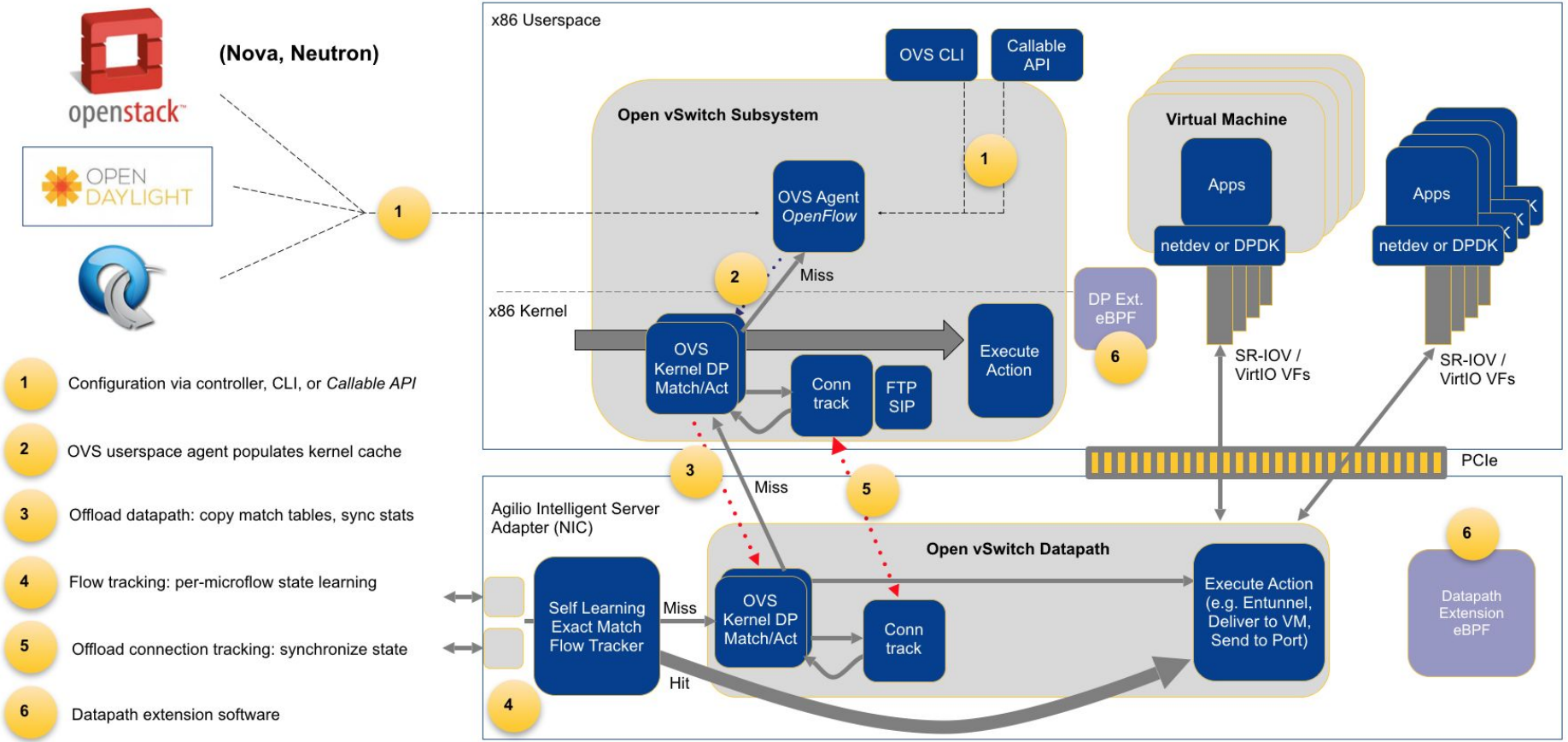
Requirements

- Acceleration should be transparent and feature rich without unduly limiting capacities and update rates
  - Full offload would include match/action, tunnels, load balancing, traffic mirroring, QoS
- Should support selective acceleration (fallback path)
- Needs to support stateful firewalling (conntrack)

Discussions of proposed approaches and architectures are ongoing in the community - please participate

Offload Model: OVS Kernel Datapath

NETRONOME

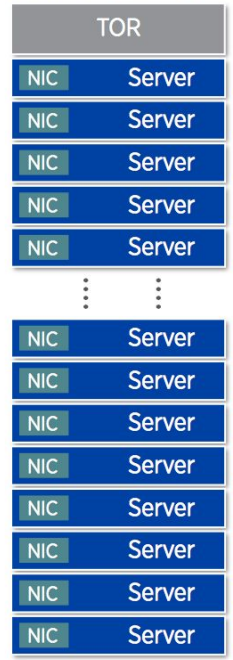


Offload Model: Performance

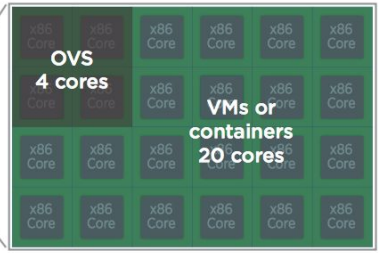
NETRONOME

Ericsson Cloud SDN with OVS in User Space and Traditional NIC

Rack Throughput: 72Mpps
Applications per Rack: 360



Server Core Allocation

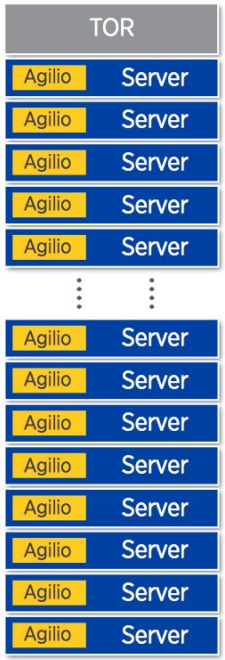


18 applications at 300Mbps/200Kpps each

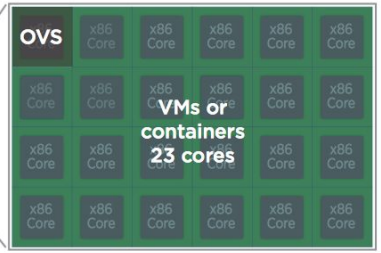
Racks needed to support 2200 VNFs

Ericsson Cloud SDN with OVS Running on Netronome Agilio Platform

Rack Throughput: 440Mpps
Applications per Rack: 2200



Server Core Allocation



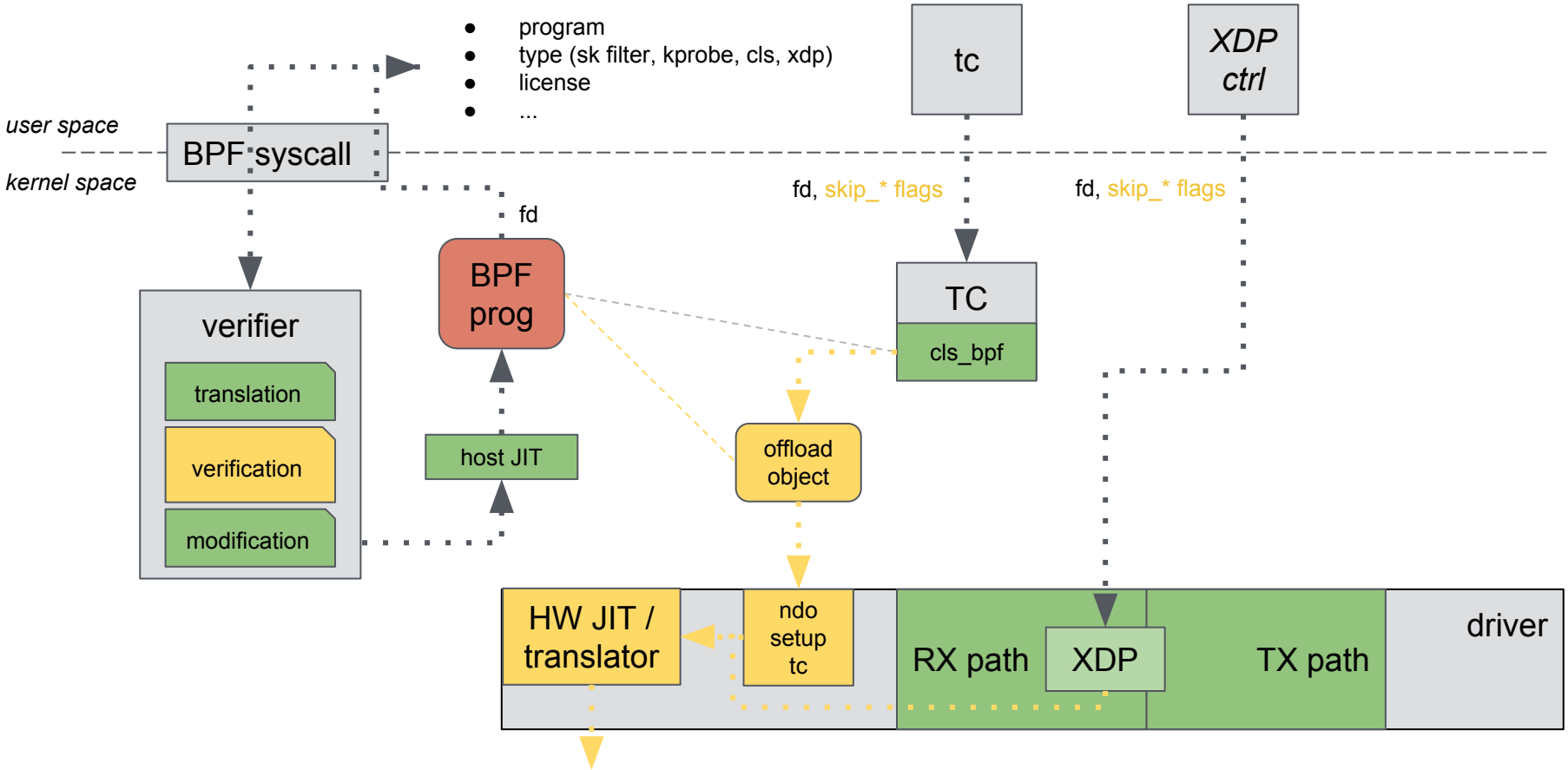
110 applications at 300Mbps/200Kpps each

Racks needed to support 2200 VNFs

6X LOWER TCO

Offload Model: eBPF Acceleration

NETRONOME



Summary

NETRONOME

Our priority is finding both a short term and long term solution to OVS working harmoniously with the kernel

- Short term - OVS kernel datapath acceleration
- Long term - accelerate eBPF, supporting eBPF based OVS?
- Our priority is ensuring that we are able to provide:
  - ‘Full Offload’, i.e the ability to offload almost all if not all OVS match/actions and tunneling
  - Per packet fallback path
  - Conntrack
  - High flow capacities and high update rates

Virtual Switch Datapath "Accelerators"

John Fastabend
Intel

Pantheon of Languages

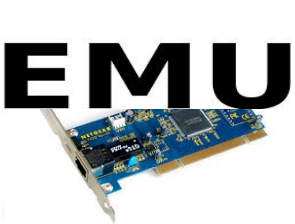
Machine Model : CPU

OPEN VSWITCH

An Open Virtual Switch

I/O Devices

QEMU



Pantheon of Languages

Machine Model : CPU

SSE

AVX

FMA

I/O Devices

TSO

Flow Control

CSUM_OFFLOAD

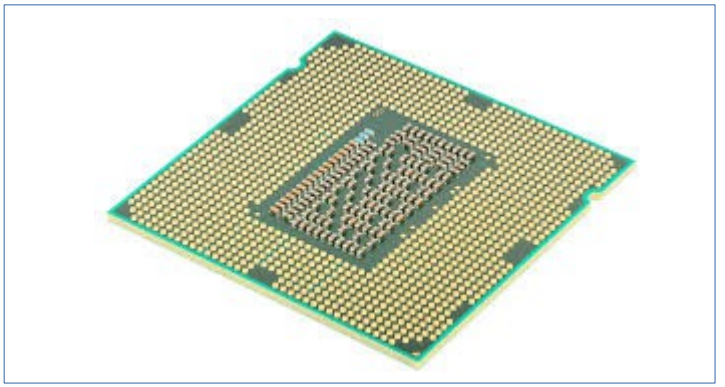
CTAG, STAG, ...

DDIO

commit ce8c839b74e3017996fad4e1b7ba2e2625ede82f
Author: Vijay Pandurangan <vijayp@vijayp.ca>
Date: Fri Dec 18 14:34:59 2015 -0500

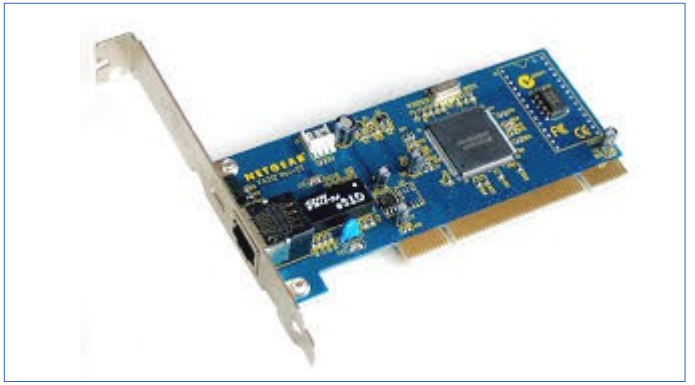
    veth: don<E2><80><99>t modify ip_summed; doing so treats packets with bad checksums as good.

    Packets that arrive from real hardware devices have ip_summed ==
    CHECKSUM_UNNECESSARY if the hardware verified the checksums, or
    CHECKSUM_NONE if the packet is bad or it was unable to verify it. The
    current version of veth will replace CHECKSUM_NONE with
    CHECKSUM_UNNECESSARY, which causes corrupt packets routed from hardware to
    a veth device to be delivered to the application. This caused applications
    at Twitter to receive corrupt data when network hardware was corrupting
    packets.



    We believe this was added as an optimization to skip computing and
    verifying checksums for communication between containers. However, locally
    generated packets have ip_summed == CHECKSUM_PARTIAL, so the code as
    written does nothing for them. As far as we can tell, after removing this
    code, these packets are transmitted from one stack to another unmodified
    (tcpdump shows invalid checksums on both sides, as expected), and they are
    delivered correctly to applications. We didn<E2><80><99>t test every possible network
    configuration, but we tried a few common ones such as bridging containers,
    using NAT between the host and a container, and routing from hardware
    devices to containers. We have effectively deployed this in production at
    Twitter (by disabling RX checksum offloading on veth devices).

    This code dates back to the first version of the driver, commit
    <e314dbdc1c0dc6a548ecf> ("[NET]: Virtual ethernet device driver"), so I
    suspect this bug occurred mostly because the driver API has evolved
    significantly since then. Commit <0b7967503dc97864f283a> ("net/veth: Fix
    packet checksumming") (in December 2010) fixed this for packets that get
    created locally and sent to hardware devices, by not changing
    CHECKSUM_PARTIAL. However, the same issue still occurs for packets coming
    in from hardware devices.



    Co-authored-by: Evan Jones <ej@evanjones.ca>
    Signed-off-by: Evan Jones <ej@evanjones.ca>
    Cc: Nicolas Dichtel <nicolas.dichtel@6wind.com>
    Cc: Phil Sutter <phil@nwl.cc>
    Cc: Toshiaki Makita <makita.toshiaki@lab.ntt.co.jp>
    Cc: netdev@vger.kernel.org
    Cc: linux-kernel@vger.kernel.org
    Signed-off-by: Vijay Pandurangan <vijayp@vijayp.ca>
    Acked-by: Cong Wang <cwang@twopensource.com>
    Signed-off-by: David S. Miller <davem@davemloft.net>

Pantheon of Languages

Machine Model : CPU

SSE

AVX

FMA

I/O Devices

TSO

Flow Control

CSUM_OFFLOAD

CTAG, STAG, ...

DDIO

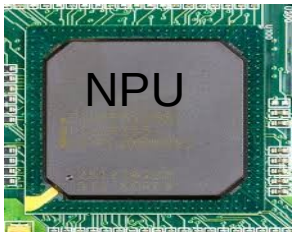
Pantheon of Languages

Machine Model : CPU

OPEN VSWITCH

An Open Virtual Switch

I/O Devices



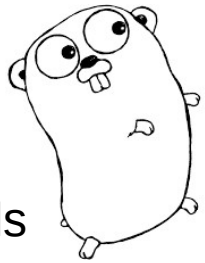
SR-IOV

Languages

Verifiers

Formal models

Runtime Environments



Encryption

Vtd/VTx

SIMD

EBPF / XDP

Data Structures

Flow Classification

P4

NPU/FPGA

tc

Discussion