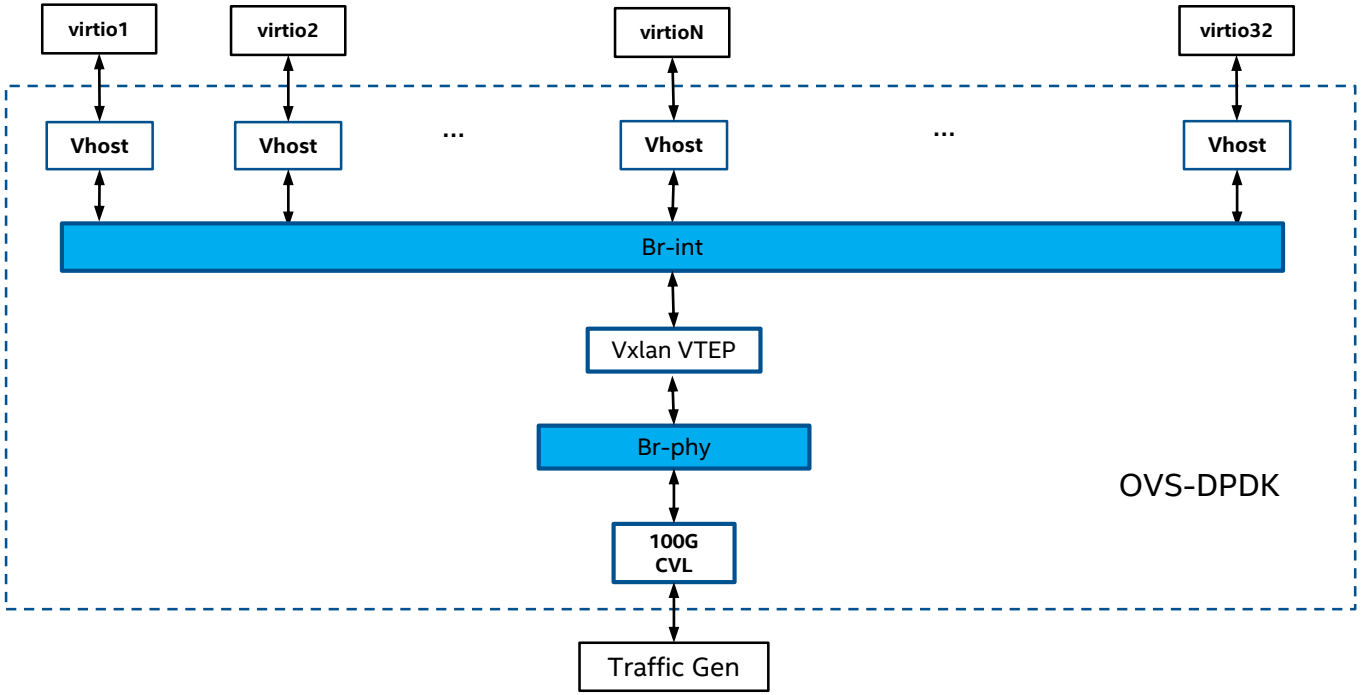
Vxlan Overlay Test Topology with 32VM




```
#Run virtio in a simple way by using virtio-user instead of booting heavy VMs
./testpmd --file-prefix=$fileprefix  --lcores=1,$pmdcore -n 4 --socket-mem 512,0 --single-file-segments --no-pci \
      --vdev net_virtio_user0,path=$sockpath,queues=1,server=1,queue_size=256,packed_vq=0 -- \
      --nb-cores=1 --rxq=1 --txq=1 --rxd=256 --txd=256 --forward-mode=macswap
```

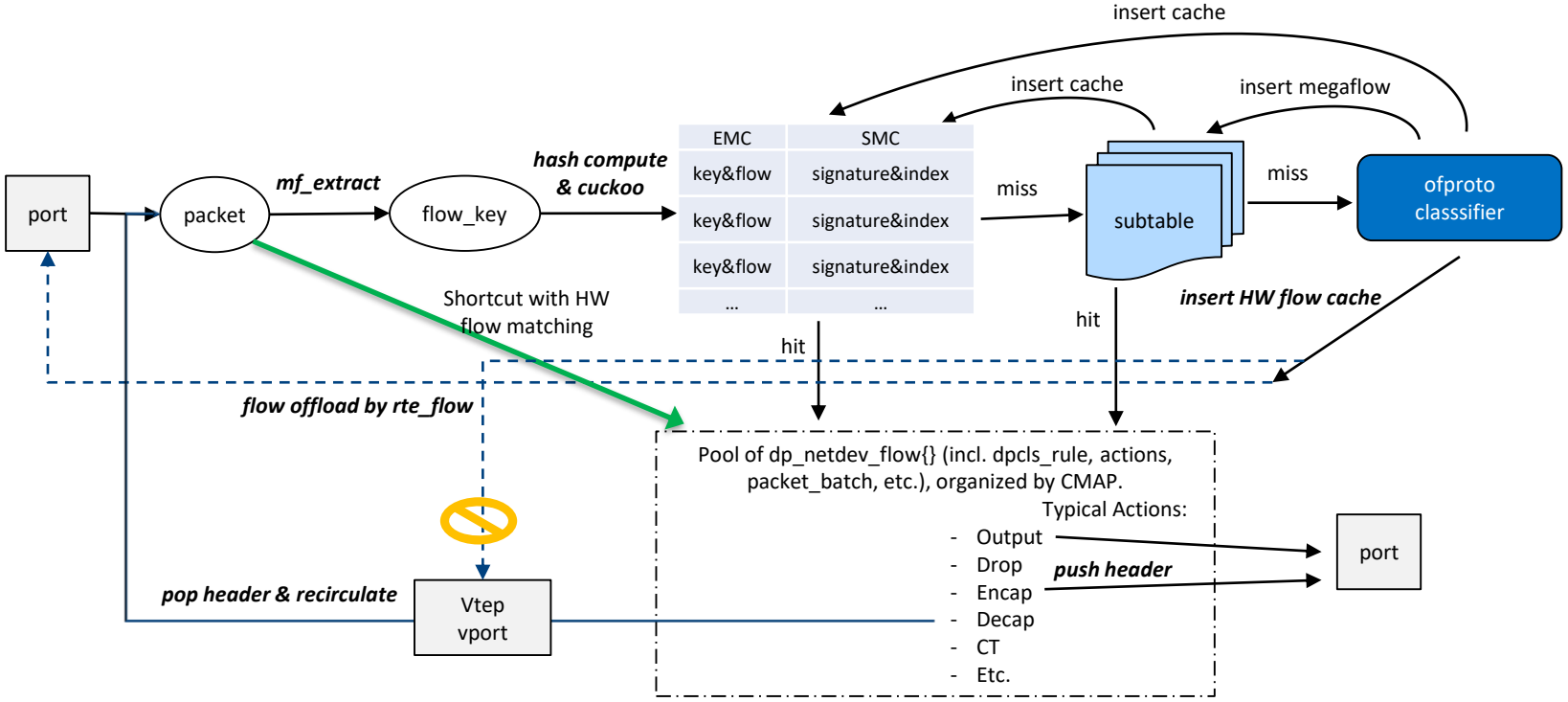
OVS-DPDK Config and Topology Setup

```
#!/bin/bash
$OVS_DIR/utilities/ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-socket-mem="1024,0"
$OVS_DIR/utilities/ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-lcore-mask="0x2“
#core 2-5 for OVS forwarding threads
$OVS_DIR/utilities/ovs-vsctl --no-wait set Open_vSwitch . other_config:pmd-cpu-mask="0x3c"
#$OVS_DIR/utilities/ovs-vsctl --no-wait set Open_vSwitch . other_config:hw-offload=true
#leave the other options as default, i.e. emc-insert-inv-prob as 1%, SMC off.
```

```
CPU BIOS config:
Turbo Boost off. /* Fixed frequency for test stability */
HT on. /* useful for running multi virtio frontend */
#core assignment (Skylake as example):
#NUMA node0 CPU(s):     0-27,56-83
#NUMA node1 CPU(s):     28-55,84-111
#core 2-5 for OVS forwarding threads.
#let each virtio pmd run on a dedicated core, the first 16 virtio pmd run
#on core 12-27, the other 16 run on core 68-83.
```

```
#!/bin/bash
#create br, port, vtep
#refer to Documentation/topics/dpdk/pmd.rst for rxq-core affinity config
#each ovs pmd thread takes 8 vhost rxqs and 1 cvl rxq
$OVS_DIR/utilities/ovs-vsctl --may-exist add-br br-int -- set Bridge br-int datapath_type=netdev -- br-set-external-id br-int bridge-id br-int -- set bridge br-int fail-mode=standalone

for i in {0..31}; do
pmdcore=$[$i/8 + 2];
$OVS_DIR/utilities/ovs-vsctl add-port br-int vhost-user$i -- set Interface vhost-user$i type=dpdkvhostuserclient options:vhost-server-path=/tmp/vhostsock$i \
options:n_rxq=1 other_config:pmd-rxq-affinity=0:$pmdcore
Done
$OVS_DIR/utilities/ovs-vsctl add-port br-int vxlan -- set interface vxlan type=vxlan options:remote_ip=flow options:local_ip=flow options:key=flow

$OVS_DIR/utilities/ovs-vsctl  --may-exist add-br br-phy -- set Bridge br-phy datapath_type=netdev -- br-set-external-id br-phy bridge-id br-phy -- set bridge br-phy fail-mode=standalone
other_config:hwaddr=10:00:00:00:00:01
$OVS_DIR/utilities/ovs-vsctl add-port br-phy dpdk-phy -- set Interface dpdk-phy type=dpdk options:dpdk-devargs=0000:18:00.0 options:n_rxq=4 other_config:pmd-rxq-affinity=0:2,1:3,2:4,3:5

#set up gateway, route and arp table for encapsulated packets forwarding
ip addr add 172.1.0.100/24 dev br-phy
ip link set br-phy up

$OVS_DIR/utilities/ovs-appctl ovs/route/add 172.1.0.0/24 br-phy
$OVS_DIR/utilities/ovs-appctl tnl/arp/set br-phy 172.1.0.0 10:00:00:00:00:02

for i in {0..31}; do
$OVS_DIR/utilities/ovs-ofctl add-flow br-int "in_port=vhost-user$i,actions=set_tunnel:1001,set_field:172.1.0.0->tun_dst,set_field:172.1.0.100->tun_src,output:vxlan"
$OVS_DIR/utilities/ovs-ofctl add-flow br-int in_port=vxlan,ip,nw_dst=192.1.0.$i,actions=output:vhost-user$i
done
$OVS_DIR/utilities/ovs-appctl vlog/set netdev_offload_dpdk::dbg
$OVS_DIR/utilities/ovs-appctl tnl/arp/show
```

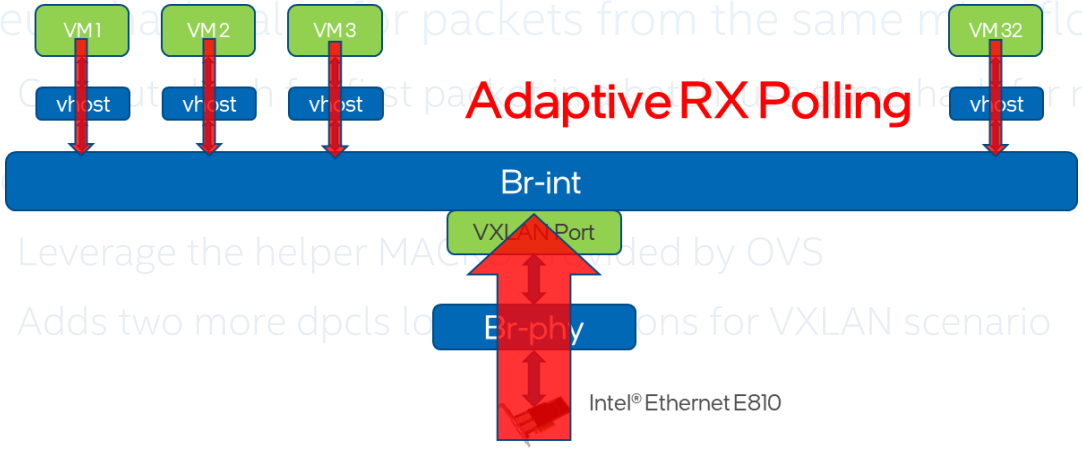
OVS Packet Pipeline



Accumulative Improvement by SW Opti

Adaptive polling instead of round robin

- Avoids overhead of polling vhost ports that have no packets
- Give more weight to active ports



Every polling round:

Credit ++;

If (credit > threshold)
  Do Polling;
Else
  Skip this round;

If (Polled packets)
  threshold --;
Else if (no packets polled)
  threshold ++;

Accumulative Improvement by SW Opti (Cont.)

Reuse hash value for packets from the same microflow

- Compute hash for first packet in a batch, use same hash for rest of the packets in a batch
- Flow key memcmp is much cheaper than hash calculation

Compile time DPCLS lookup optimization for VXLAN

- Dpcls lookup function abstracted as a per subtable pointer
- Give compiler more hint for generating fully optimized code
- Declare two more dpcls lookup functions for VXLAN scenario

Accumulative Improvement by SW Opti (Cont.)

A faster key extraction for common packet types (e.g. IP/UDP)

- Mini flow extract is optimized for commonly used packet types

Batching of header encap/decap

- Encap and decap performed in batch, rather than on packet basis

Queue size config for smaller memory footprint

- Reduce LLC-load-miss events significantly

```
perf  stat  -C 2 -e LLC-load,LLC-load-miss sleep 10

Performance counter stats for 'CPU(s) 2':
      108,602,622      LLC-load
        3,437,638      LLC-load-miss

#With smaller queue size
 Performance counter stats for 'CPU(s) 2':
       60,722,928      LLC-load
          228,277      LLC-load-miss
```

Rte_flow semantics of outer flow and inner flow

Attributes: ingress=1, egress=0, prio=0, group=0, transfer=0

rte flow eth pattern:

Spec: src=10:00:00:00:00:02, dst=10:00:00:00:00:01, type=0x0800

Mask: src=ff:ff:ff:ff:ff:ff, dst=ff:ff:ff:ff:ff:ff, type=0xffff

rte flow ipv4 pattern:

Spec: tos=0x0, ttl=40, proto=0x11, src=172.1.0.200, dst=172.1.0.100

Mask: tos=0x0, ttl=0, proto=0x0, src=0.0.0.0, dst=255.255.255.255

rte flow udp pattern:

Spec: src_port=1000, dst_port=4789

Mask: src_port=0x0, dst_port=0xffff

rte flow mark action:

Mark: id=0

rte flow RSS action:

RSS: queue_num=4

Attributes: ingress=1, egress=0, prio=0, group=0, transfer=0

rte flow eth pattern:

Spec = null

Mask = null

rte flow ipv4 pattern:

Spec: tos=0x0, ttl=40, proto=0x0, src=172.1.0.0, dst=172.1.0.100

Mask: tos=0xff, ttl=0, proto=0x0, src=255.255.255.255, dst=255.255.255.255

rte flow udp pattern:

Spec: src_port=34233, dst_port=4789

Mask: src_port=0x0, dst_port=0x0

rte flow vxlan pattern:

Spec: vni=1001

Mask: vni=0xffffff

rte flow eth pattern:

Spec: src=a0:00:00:00:00:02, dst=a0:00:00:00:00:01, type=0x0800

Mask: src=00:00:00:00:00:00, dst=ff:ff:ff:ff:ff:ff, type=0xffff

rte flow ipv4 pattern:

Spec: tos=0x0, ttl=40, proto=0x11, src=192.1.0.200, dst=192.1.0.1

Mask: tos=0x0, ttl=0, proto=0x0, src=0.0.0.0, dst=255.255.255.255

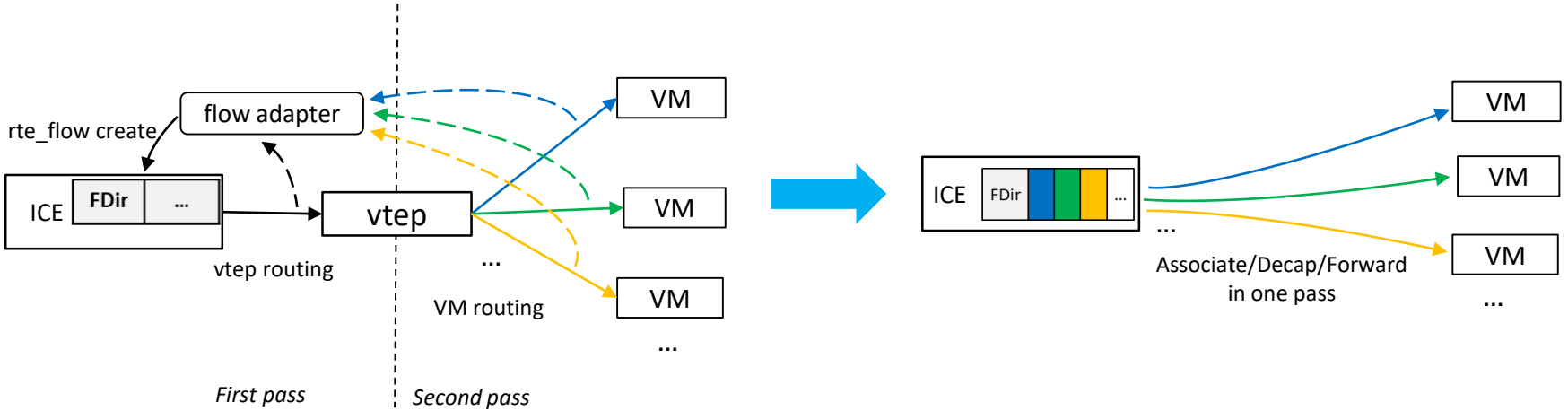
rte flow mark action:

Mark: id=0

rte flow RSS action:

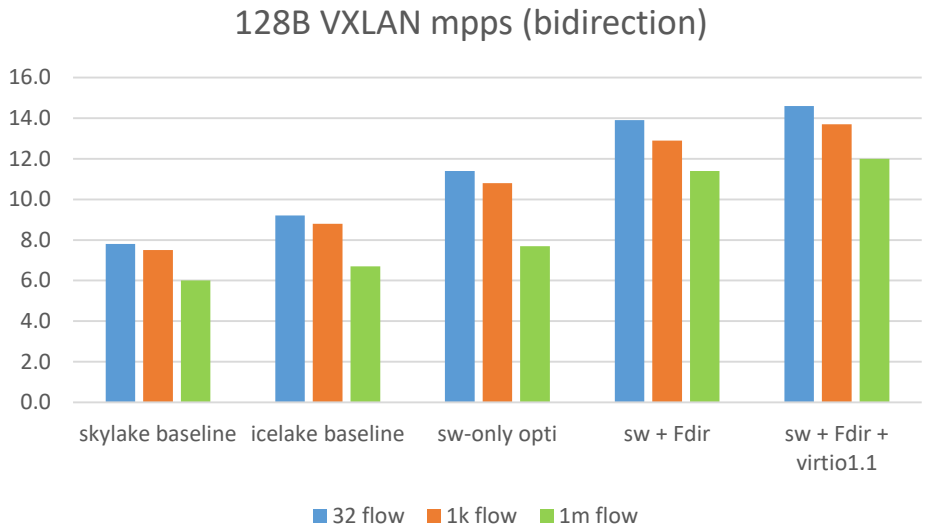
RSS: queue_num=4

VXLAN Flow Offload with E810 Flow Director



- First packet goes to slow path and triggers flow offload
- Subsequent packets are forwarded directly, reduce twice parsing and lookup

Benchmark (Oct 21st)



Skylake 8180 CPU @ 2.50GHz

Icelake HCC CPU @ 2.30GHz, with larger L1/L2 cache

OVS baseline: v2.13

DPDK baseline: v20.08

Burst size: 32 pkts/flow

# of forwarding cores: 4c/4t

VXLAN source port using hash on inner header

	skylake baseline	icelake baseline	sw-only opti	sw + Fdir	sw + Fdir + virtio1.1
32 microflow	7.8	9.2	11.4	13.9	14.6
1k microflow	7.5	8.8	10.8	12.9	13.7
1m microflow	6.0	6.7	7.7	11.4	12.0

Cont. (ww48)

Partial Offload (SW-centric OVS-DPDK Acc)

- Partial offload vs full offload, bare mental, etc.
- CPU does the heavy lifting with HW assisting part of packet processing
- Both slow path & fast path implemented in SW
- Inherit all legacy advantages e.g. live migration, hot-upgrade
- Still massive stock deployment

Existing techniques and challenges

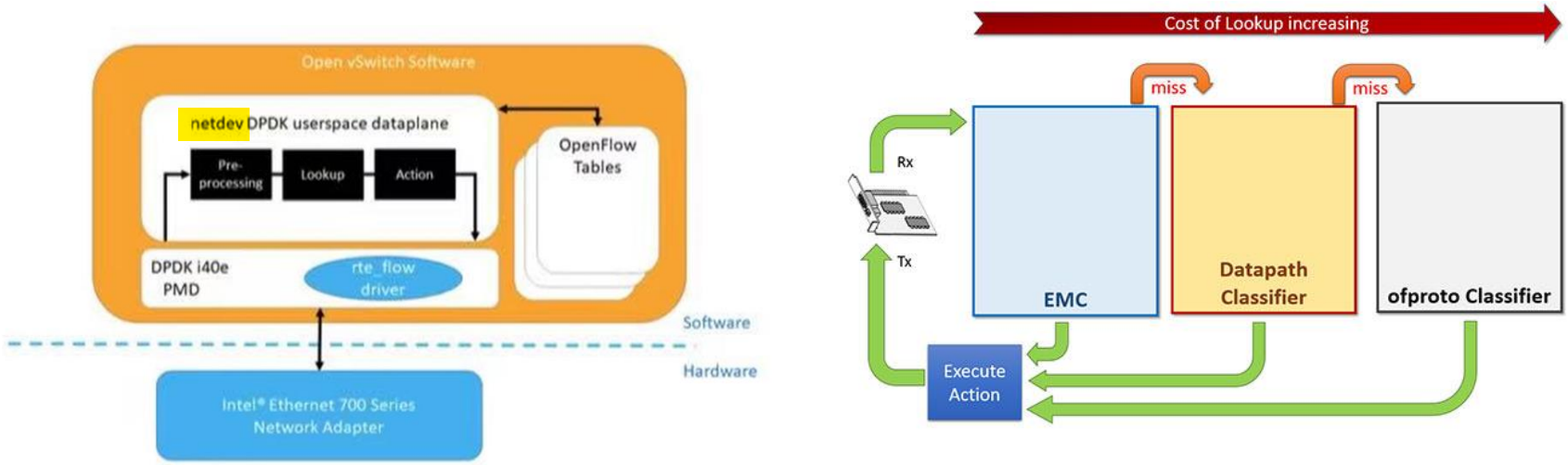
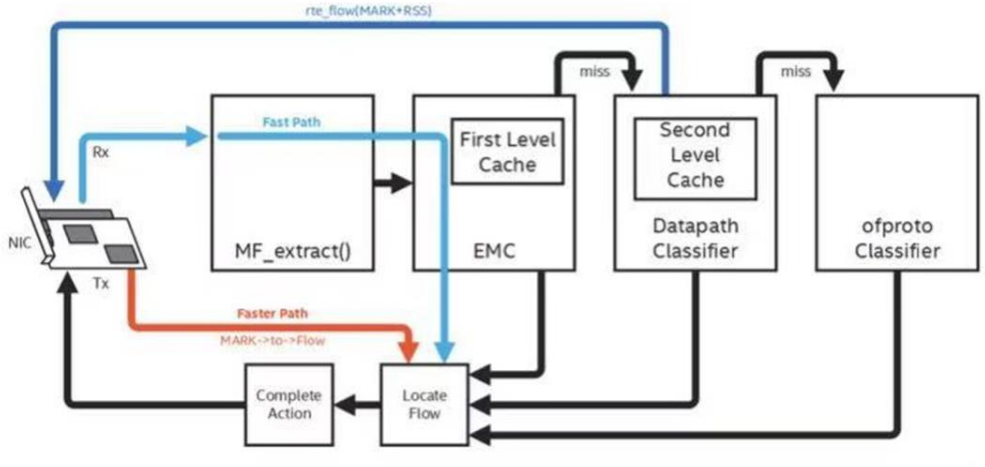


图1. OVS 数据包处理概述

https://software.intel.com/content/www/us/en/develop/articles/ovs-dpdk-datapath-classifier.html
Intel® Ethernet Controller 700系列: Open vSwitch硬件加速应用说明

Flow offload to NIC pipeline



- HW offload packet parse/table lookup
- Limited to only physical device
- flow mark support in vector path
- Insertion rate

- One packet type with multi masks?

图 2. OVS DPDK 流卸载提供了一种更快的查找方法

ufid:da18d4e2-b740-41dc-94fb-20d8042014d6,
skb_priority(0/0),skb_mark(0/0),ct_state(0/0),ct_zone(0/0),ct_mark(0/0),ct_label(0/0),recirc_id(0),dp_hash(0/0),in_port(veth_l0),packet_type(ns=0/0,id=0/0),eth(src=00:00:00:00:00:00/00:00:00:00:00:00,dst=00:00:00:00:00:00/00:00:00:00:00:00),eth_type(0x0800),ipv4(src=1.1.1.0,**dst=1.1.1.2/255.255.255.254**,proto=0/0,tos=0/0,ttl=0/0,frag=no), packets:0, bytes:0, used:3.630s, dp:tc, actions:veth_r0

ufid:e55ef147-434f-44ce-8a96-51e81e745daf,
skb_priority(0/0),skb_mark(0/0),ct_state(0/0),ct_zone(0/0),ct_mark(0/0),ct_label(0/0),recirc_id(0),dp_hash(0/0),in_port(veth_l0),packet_type(ns=0/0,id=0/0),eth(src=00:00:00:00:00:00/00:00:00:00:00:00,dst=00:00:00:00:00:00/00:00:00:00:00:00),eth_type(0x0800),ipv4(src=1.1.1.0,**dst=1.1.1.4/255.255.255.252**,proto=0/0,tos=0/0,ttl=0/0,frag=no), packets:0, bytes:0, used:1.470s, dp:tc, actions:veth_r0

SW Opti for multi subtables

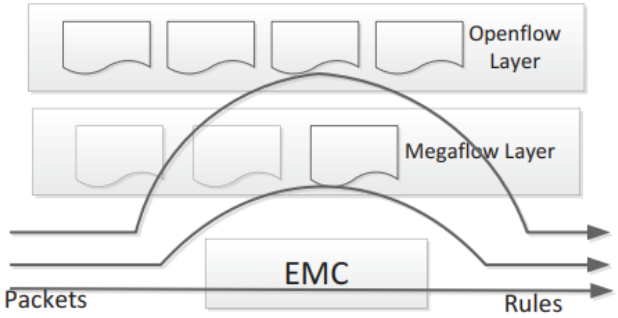


Fig. 1. Three layers of OvS packet classification process.

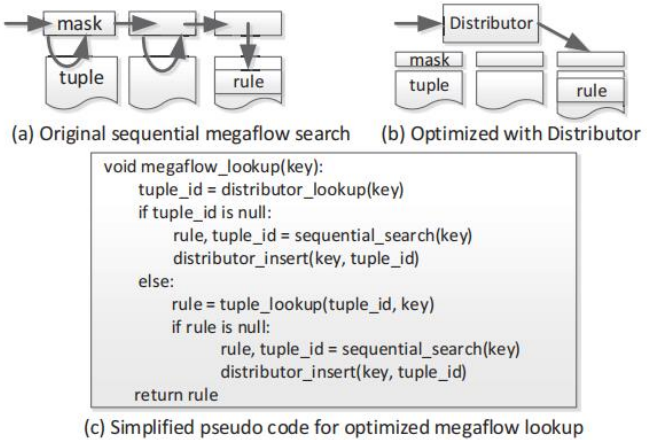
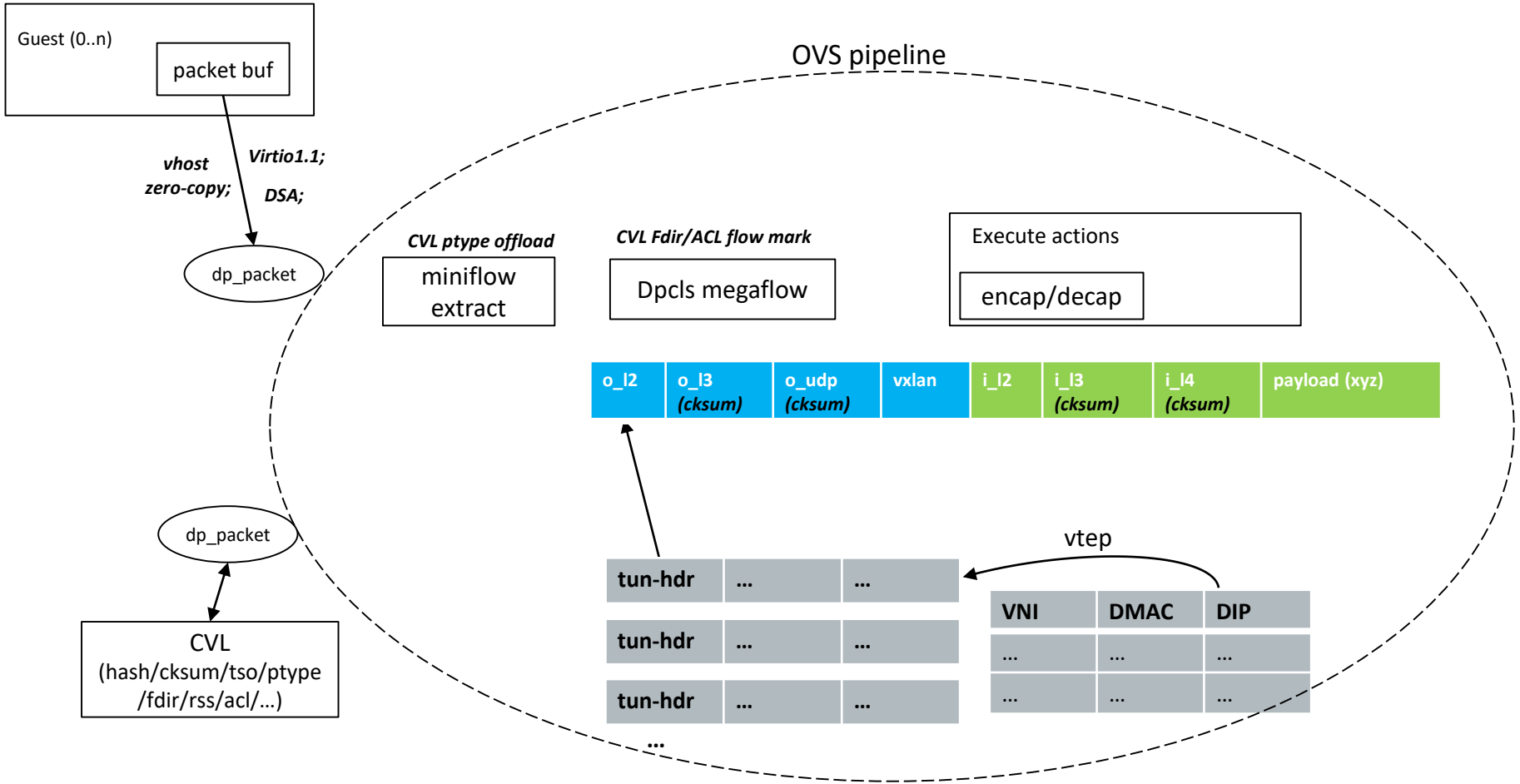


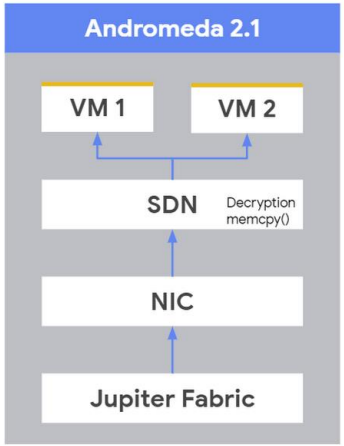
Fig. 2. Original megaflow search vs. distributor optimized search.

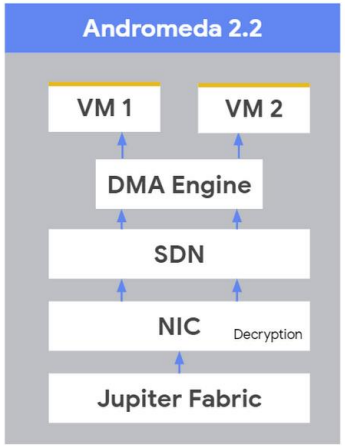
https://ieeexplore.ieee.org/Xplore/desktopReportingPrompt.jsp?tp=&arnumber=8254754&pdfRequest=true


Guest (0..n)
packet buf
vhost zero-copy;
Virtio1.1; DSA;
dp_packet
OVS pipeline
CVL ptype offload
miniflow extract
CVL Fdir/ACL flow mark
Dpcls megaflow
Execute actions
encap/decap
o_l2
o_l3 (cksum)
o_udp (cksum)
vxlan
i_l2
i_l3 (cksum)
i_l4 (cksum)
payload (xyz)
dp_packet
vtep
tun-hdr
...
VNI
DMAC
DIP
CVL
(hash/cksum/tso/ptype /fdir/rss/acl/...)

Features in hand and challenge

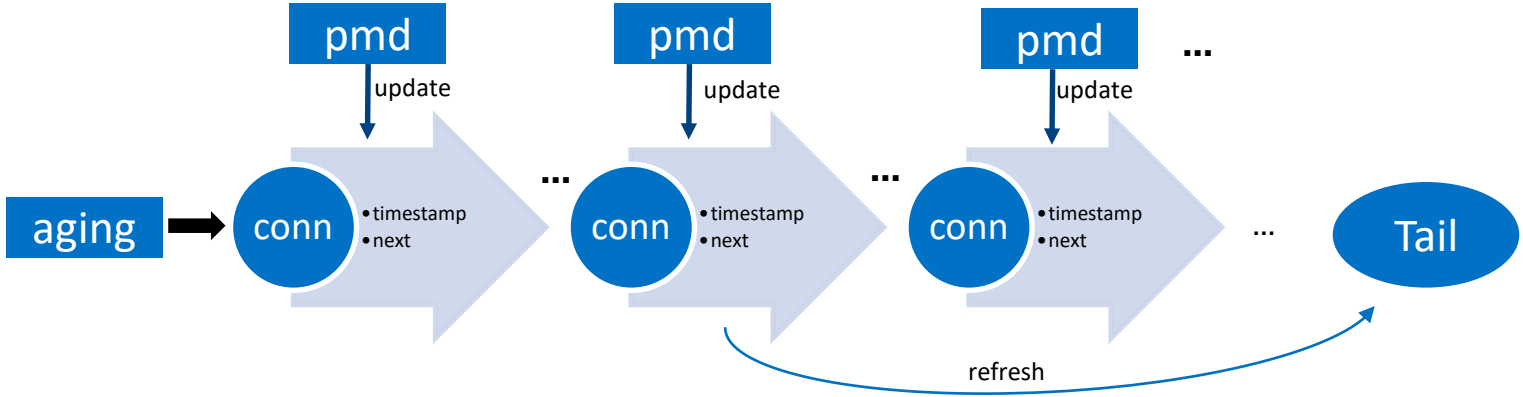
- Virtio 1.1: need guest kernel update
- Zero-copy: can not work with TUNNEL, headroom needed
- DSA for host-guest data movement: need OVS adoption
- HW hash & RSS
- HW+virtio CKSUM/TSO: compatible with tunnel encap?
- HW Ptype: ambiguity on VLAN





Other challenges

- Cross NUMA host-guest communication
- Perf pain with VXLAN + Conntrack + NAT
  - Recirculation overhead (5 times pipeline walkthrough, 7 times parsing/lookup)
  - Session management (locks, random/huge memory footprint, shared access)



THANKS