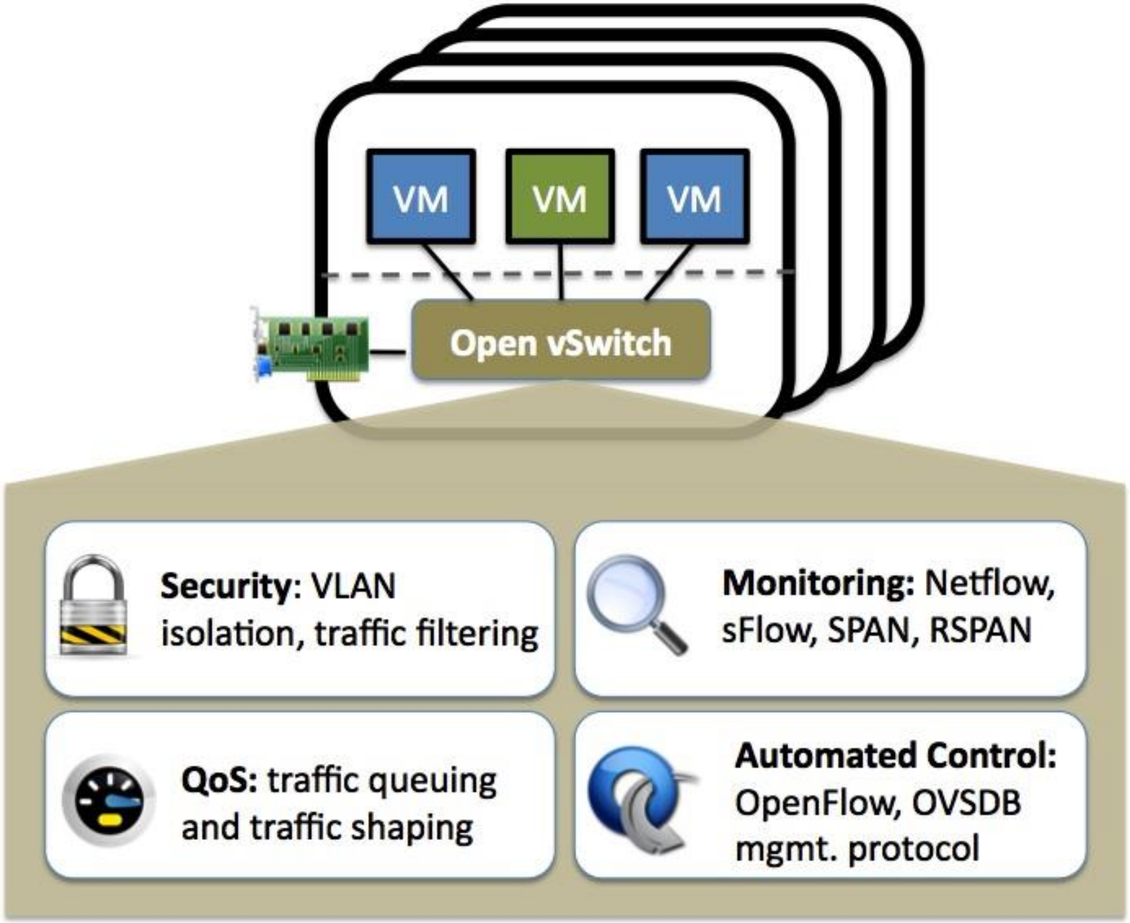
An Introduction to Open vSwitch

17TH FEB, 2020

SAHIL SEMICONDUCTOR

Open vSwitch

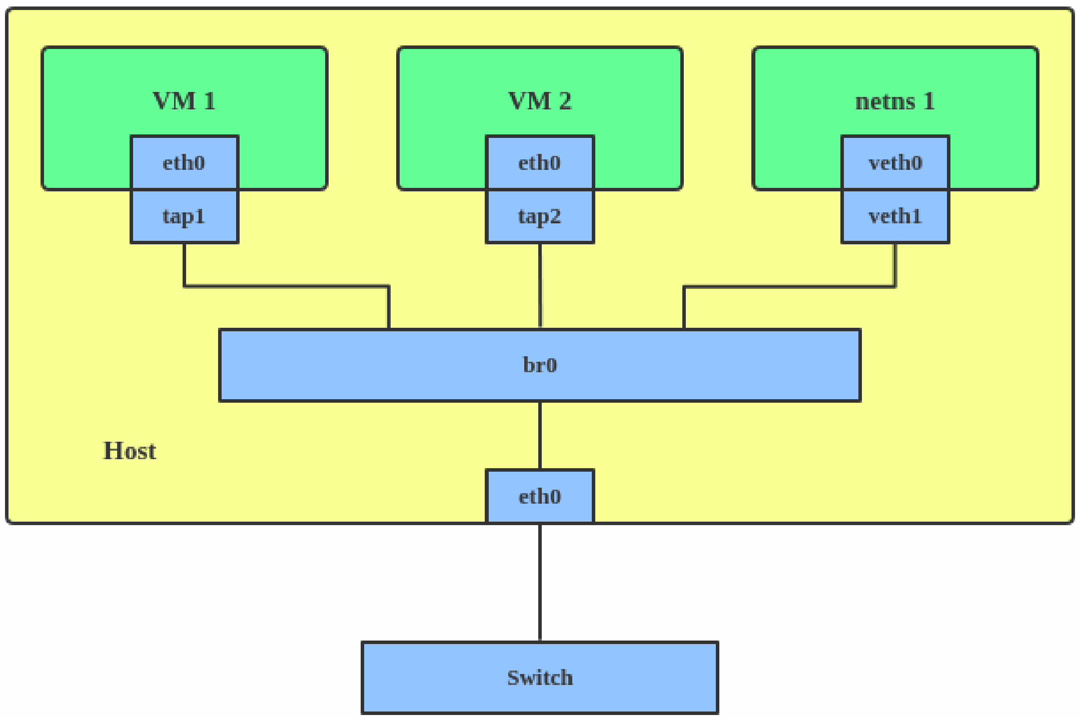
- Software based Open Source, OpenFlow capable virtual switch
- Used with hypervisors to interconnect VMs within a host and between different hosts across networks.
- Provides network isolation for VM traffic
- Flexible controller in user space
- Fast data path in kernel space
- Introduction to Open vSwitch (OVS)



Why Open vSwitch?

- Hypervisors need the ability to bridge traffic between VMs and with the outside world.
- On Linux-based hypervisors, this is done using built-in L2 switch (the Linux bridge), which is fast and reliable.

- So, it is reasonable to ask why Open vSwitch is used?

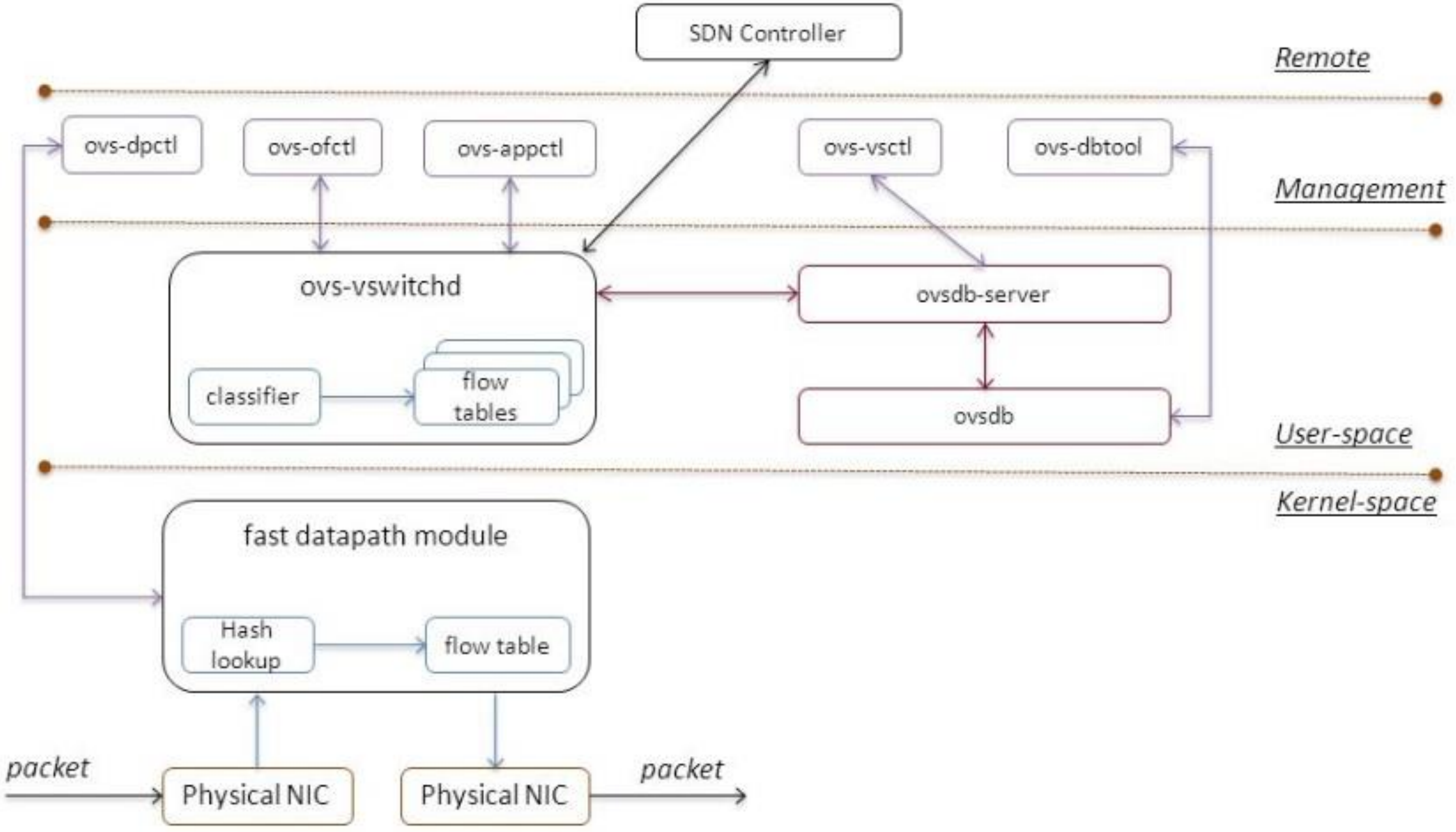


Why Open vSwitch?

- Open vSwitch is targeted at multi-server virtualization deployments, a landscape for which the previous stack is not well suited.
- These environments are often characterized by highly dynamic end-points, the maintenance of logical abstractions, and (sometimes) integration with or offloading to special purpose switching hardware.

- **Mobility of State:** All networks states of Open vSwitch belong to a VM (L2 table, L3 forwarding state, policy routing, ACL, QoS, monitoring configuration etc) can be easily migrated to another host.
- **Responding to Network Dynamics:** Virtual environments change at a high rate (VMs coming and going, changes to network environments).
- **Easy to configure and maintain:** Provides tools to remotely configure and integrate with OpenFlow and OpenStack.
- **Hardware Integration:** Forwarding path can be offloaded to hardware platforms.

Open vSwitch - Components



ovsdb-server

- Database that holds switch-level configuration
  - Creation, modification and deletion of bridges, data path ports, tunnel interfaces, queues
  - Stores OpenFlow controller addresses
  - Configuration of QoS (Quality of Service) policies and how to associate them to queues and ports
  - Stats collection
- Configuration is stored on disk and survives a reboot
- Speaks OVSDB protocol to OpenFlow controller and ovs-vswitchd
- The OVSDB protocol is defined in RFC-7047
- **It does not store per-flow information (OVS rules)**

ovs-vsctl

- Configures ovs-vswitchd, but really a high-level interface for database (ovsdb-server)
- Used for configuration and viewing OVS switch operations including port configuration, bridge additions/deletions, bonding, and VLAN tagging etc.

  – ovs-vsctl add-br <bridge>
  – ovs-vsctl list-br
  – ovs-vsctl add-port <bridge> <port>
  – ovs-vsctl list-ports <bridge>
  – ovs-vsctl get-manager <bridge>
  – ovs-vsctl get-controller <bridge>
  – ovs-vsctl list <table>

ovs-vswitchd

- Core component in the system:
  - Communicates with outside world using OpenFlow
  - Communicates with ovsdb-server using OVSDB protocol
  - Communicates with kernel module over netlink
  - Communicates with the system through netdev abstract interface
- Supports multiple independent datapaths (bridges)
- Packet classifier supports efficient flow lookup with wildcards and “explodes” these (possibly) wildcard rules for fast processing by the datapath
- Implements mirroring, bonding, and VLANs through modifications of the same flow table exposed through OpenFlow
- Checks datapath flow counters to handle flow expiration and stats requests
- CLI Tools: ovs-ofctl, ovs-appctl

ovs-ofctl

- ovs-ofctl is a command line tool for monitoring and administering OpenFlow switches.
- It can also show the current state of an OpenFlow switch, including features, configuration, and table entries.
- It works with any OpenFlow switches including Open vSwitch.

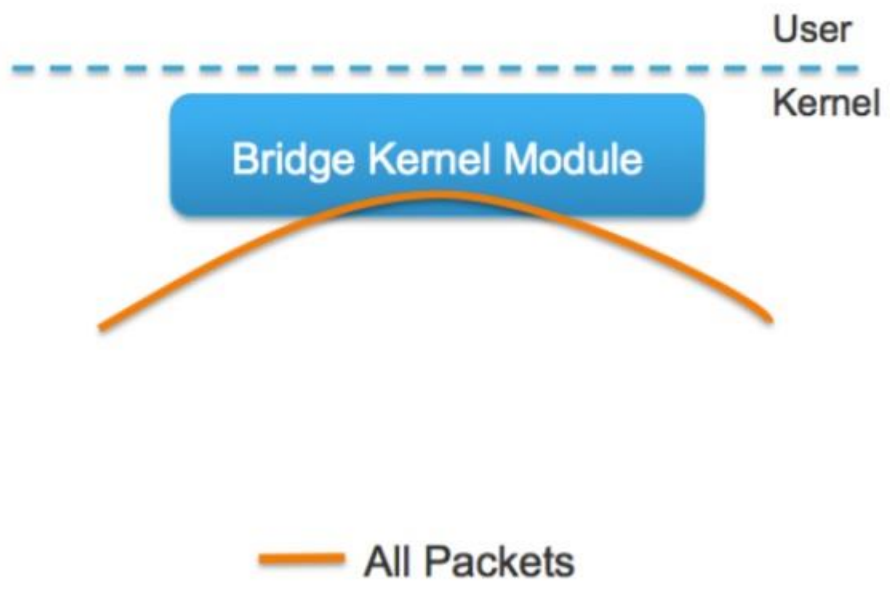
- ovs-ofctl speaks to OpenFlow module
- – ovs-ofctl show <bridge>
- – ovs-ofctl dump-flows <bridge>
- – ovs-ofctl add-flow <bridge> <flow>
- – ovs-ofctl del-flows <bridge> [flow]
- – ovs-ofctl snoop <bridge>

OVS Kernel Module (Fast DataPath Module)

- Kernel module that handles switching and tunneling
- Fast cache of non-overlapping flows
- Designed to be fast and simple
  - Packet comes in, if match found, associated actions are executed and counters updated. Otherwise, sent to user space
  - Does no flow expiration
  - Knows nothing of OpenFlow
- Implements tunnels

Traditional Linux Bridge Design

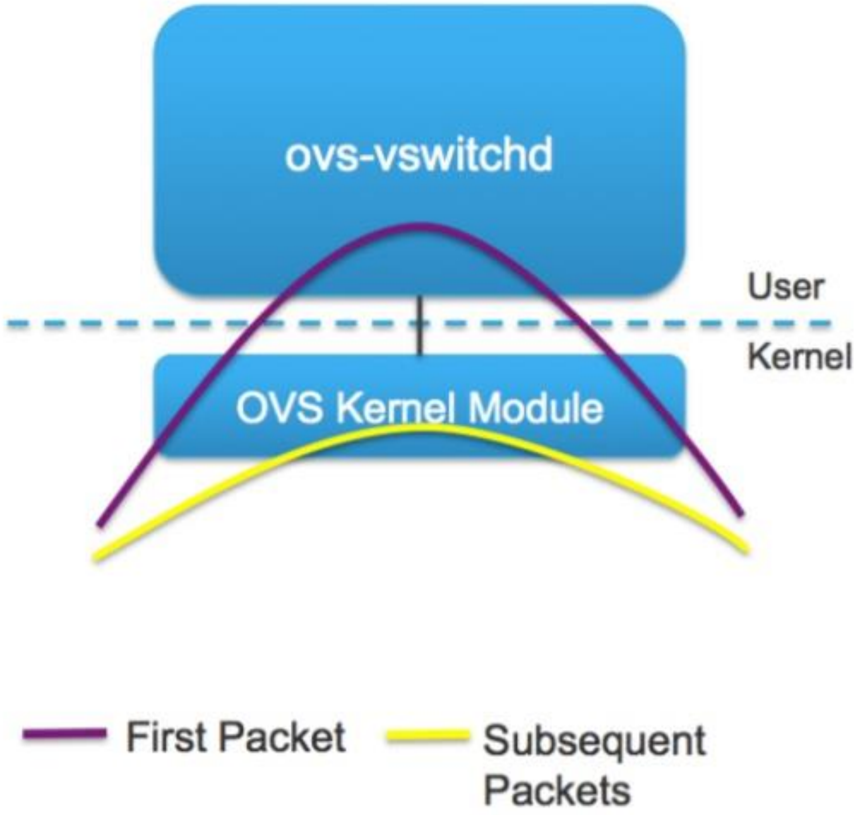
- Simple forwarding
- Matches destination MAC address and forwards
- Packet never leaves kernel



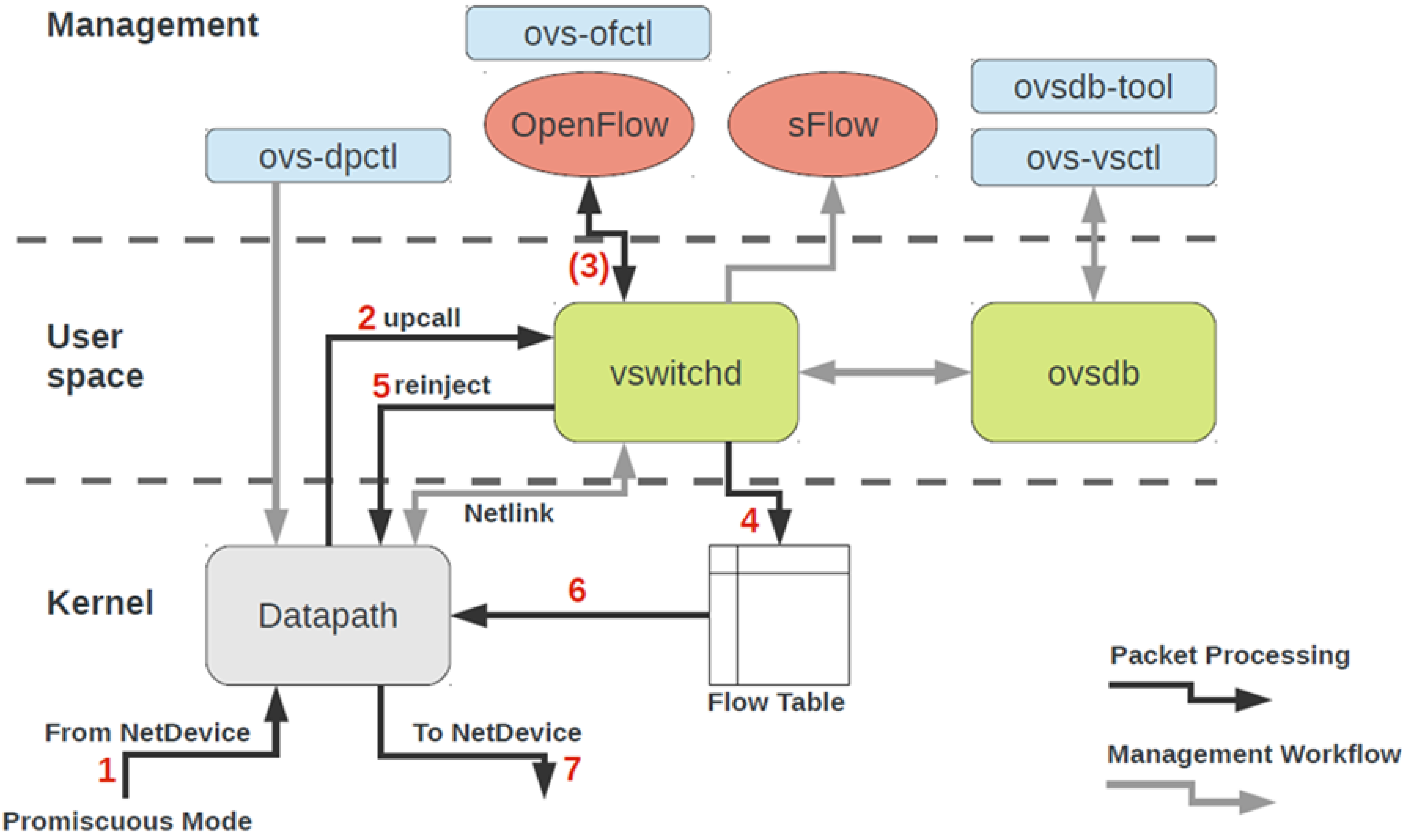
Open vSwitch Design

- Decision about how to process a packet made in user space (ovs-vswichd)

- First packet of new flow goes to ovsvswitchd, following packets hit cached entry in kernel



Packet flow through OVS



Packet flow through OVS

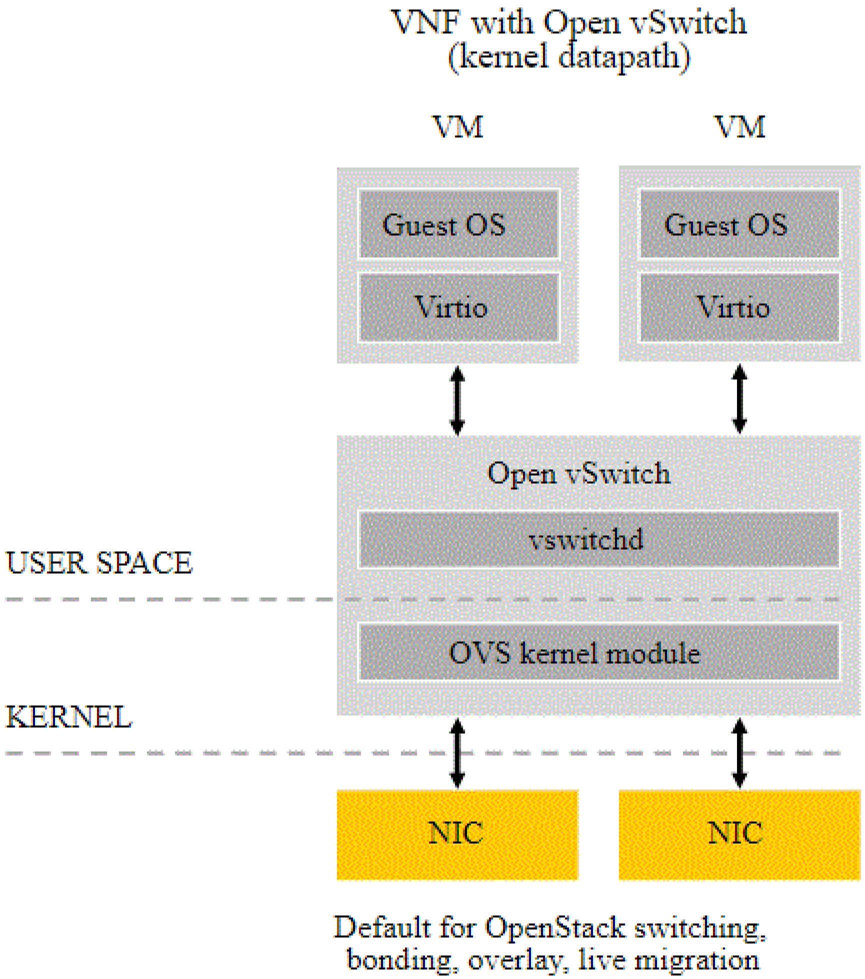
- Kernel Processing:
  - Packet arrives and header fields extracted
  - Header fields are hashed and used as an index into a set of large hash tables
  - If entry found, actions applied to packet and counters are updated
  - If entry is not found, packet sent to user space and miss counter incremented

- User space processing:
  - Packet received from kernel
  - Given to the classifier to look for matching flows and actions
  - If entry is found, kernel table is updated.
  - In case of a miss, packet it sent to an OpenFlow controller which sends a flow entry associated to the packet back to the ovs-vswitchd.

OVS Models (OVS Datapath in Kernel)

- OVS data path in kernel
- Mature and most commonly used
- Uses virtIO driver in VM making the VMs hardware independent
- Broad array of supported guest Operating Systems
- Supports VM Live Migration

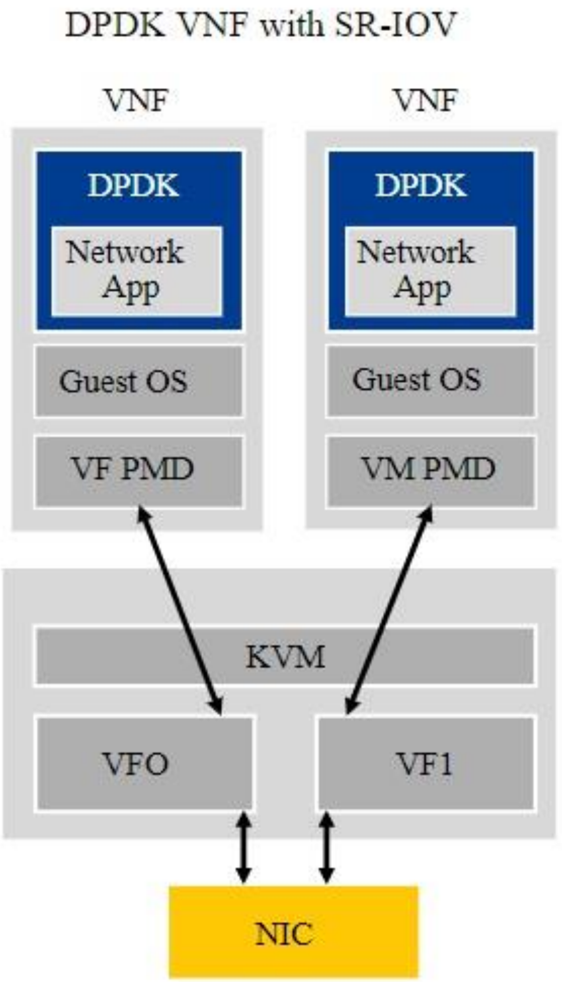
- Poor data path performance



OVS Models (SR-IOV with OVS bypass)

- Matured and proven model.
- Excellent performance. Packets are delivered directly to VMs.

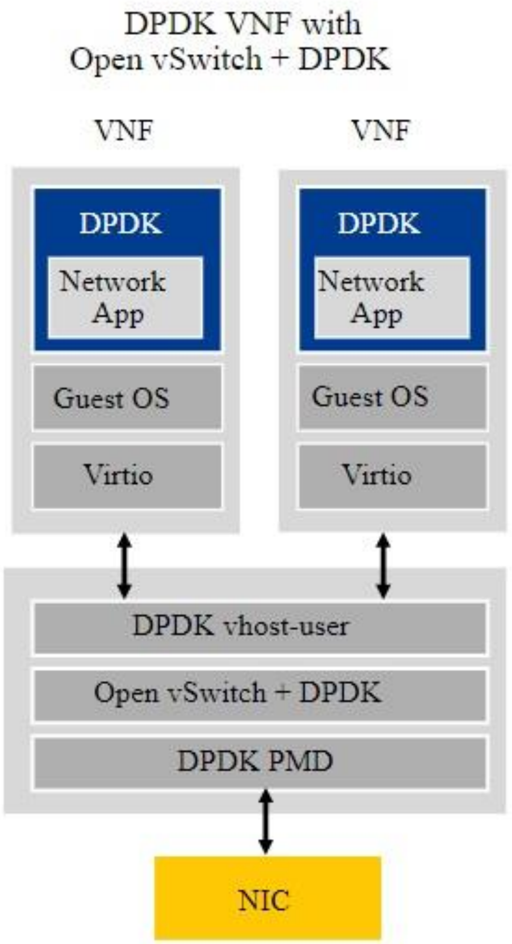
- No OVS switching. Traffic is routed using VFs.
- OVS data path in kernel but not used (bypassed using SR-IOV)
- Vendor provided drivers in VM making VMs hardware dependent.
- Live Migration is not supported.



OVS Models (OVS data path in user space)

- Matured DPDK support.
- Uses virtIO driver in VM to talk to DPDK cores. This enables this model to become hardware independent.
- Support large number of Guest Operating Systems.
- Supports VM Live Migration.
- Good data path performance

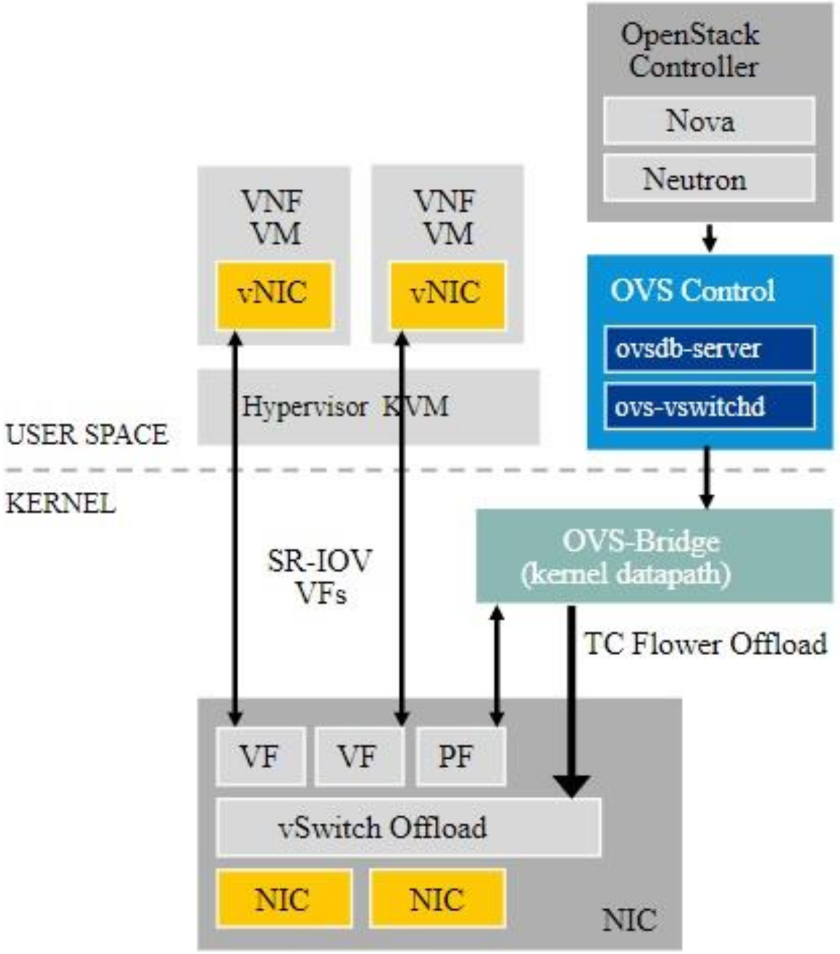
- Difficult to integrate with other kernel data path features (eBPF, IP tables, conntrack etc).



OVS Models (OVS data path offloaded to SmartNIC)

- OVS match-action is performed by the SmartNIC, fallback to kernel OVS for control traffic and new/first flow.
- Uses kernel-compliant TC/Flower based offload (offload already part of REHL 7.5).
- Excellent data path performance.
- Frees CPU cores from data path processing.

- Offload support is only available in latest kernels.
- Vendor provided drivers in VM making VMs hardware dependent.
- Live Migration is not supported.



Linux TC (Traffic Control)

- TC is used to configure Traffic Control in the Linux kernel.
- Traffic Control consists of following:
  1. Shaping – controlling rate of transmission
  2. Scheduling – transmission traffic prioritizing
  3. Policing – ingress traffic control
  4. Dropping – ingress/egress traffic drop under specific conditions

- Traffic processing is controlled by three kinds of objects:
  1. Qdiscs (Queueing Discipline) – kernel enqueues packets to qdisc interfaces when it needs to send them out.
  2. Classes – some qdiscs can contain classes, which contains further classes. A qdisc may prioritize certain kinds of traffic.
  3. Filters – decides in which class, a packet should be enqueued. Examples are BPF, flow, route etc.

- **More details: ‘man tc’**

Example of TC Flower

- Filter packets received on eth0
- Drop TCP packets with destination port 80

```
# tc qdisc add dev eth0 ingress
# tc filter add dev eth0 protocol ip parent ffff: \
    flower ip_proto tcp dst_port 80 \
        action drop
```

- Presentation on TC Flower Offload:

https://www.youtube.com/watch?v=lc20Yy-xFRs

What is TC Flower

- Flower is a flow-based traffic control filter for TC
- Packet classifier for Linux kernel traffic classification (TC) subsystem
- TC Flower classifier allows matching packets against pre-defined flow key fields:
  - Packet headers: For example, IPv4 source address
  - Tunnel meta data: For example, Tunnel key ID
  - Metadata: Input port
- TC actions allow packets to be modified, forwarded, dropped etc.
  - pedit: modify packet data
  - mirred: output packet
  - vlan: push, pop or modify vlan