**Why Open vSwitch?**

 **=================**

Hypervisors need the ability to bridge traffic between VMs and with the

outside world. On Linux-based hypervisors, this used to mean using the

built-in L2 switch (the Linux bridge), which is fast and reliable. So,

it is reasonable to ask why Open vSwitch is used.

The answer is that 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.

The following characteristics and design considerations help Open

vSwitch cope with the above requirements.

\* The mobility of state: All network state associated with a network

 entity (say a virtual machine) should be easily identifiable and

 migratable between different hosts. This may include traditional

 "soft state" (such as an entry in an L2 learning table), L3 forwarding

 state, policy routing state, ACLs, QoS policy, monitoring

 configuration (e.g. NetFlow, IPFIX, sFlow), etc.

 Open vSwitch has support for both configuring and migrating both slow

 (configuration) and fast network state between instances. For

 example, if a VM migrates between end-hosts, it is possible to not

 only migrate associated configuration (SPAN rules, ACLs, QoS) but any

 live network state (including, for example, existing state which

 may be difficult to reconstruct). Further, Open vSwitch state is

 typed and backed by a real data-model allowing for the development of

 structured automation systems.

\* Responding to network dynamics: Virtual environments are often

 characterized by high-rates of change. VMs coming and going, VMs

 moving backwards and forwards in time, changes to the logical network

 environments, and so forth.

 Open vSwitch supports a number of features that allow a network

 control system to respond and adapt as the environment changes.

 This includes simple accounting and visibility support such as

 NetFlow, IPFIX, and sFlow. But perhaps more useful, Open vSwitch

 supports a network state database (OVSDB) that supports remote

 triggers. Therefore, a piece of orchestration software can "watch"

 various aspects of the network and respond if/when they change.

 This is used heavily today, for example, to respond to and track VM

 migrations.

 Open vSwitch also supports OpenFlow as a method of exporting remote

 access to control traffic. There are a number of uses for this

 including global network discovery through inspection of discovery

 or link-state traffic (e.g. LLDP, CDP, OSPF, etc.).

\* Maintenance of logical tags: Distributed virtual switches (such as

 VMware vDS and Cisco's Nexus 1000V) often maintain logical context

 within the network through appending or manipulating tags in network

 packets. This can be used to uniquely identify a VM (in a manner

 resistant to hardware spoofing), or to hold some other context that

 is only relevant in the logical domain. Much of the problem of

 building a distributed virtual switch is to efficiently and correctly

 manage these tags.

 Open vSwitch includes multiple methods for specifying and maintaining

 tagging rules, all of which are accessible to a remote process for

 orchestration. Further, in many cases these tagging rules are stored

 in an optimized form so they don't have to be coupled with a

 heavyweight network device. This allows, for example, thousands of

 tagging or address remapping rules to be configured, changed, and

 migrated.

 In a similar vein, Open vSwitch supports a GRE implementation that can

 handle thousands of simultaneous GRE tunnels and supports remote

 configuration for tunnel creation, configuration, and tear-down.

 This, for example, can be used to connect private VM networks in

 different data centers.

\* Hardware integration: Open vSwitch's forwarding path (the in-kernel

 datapath) is designed to be amenable to "offloading" packet processing

 to hardware chipsets, whether housed in a classic hardware switch

 chassis or in an end-host NIC. This allows for the Open vSwitch

 control path to be able to both control a pure software

 implementation or a hardware switch.

 There are many ongoing efforts to port Open vSwitch to hardware

 chipsets. These include multiple merchant silicon chipsets (Broadcom

 and Marvell), as well as a number of vendor-specific platforms. (The

 PORTING file discusses how one would go about making such a port.)

 The advantage of hardware integration is not only performance within

 virtualized environments. If physical switches also expose the Open

 vSwitch control abstractions, both bare-metal and virtualized hosting

 environments can be managed using the same mechanism for automated

 network control.

In many ways, Open vSwitch targets a different point in the design space

than previous hypervisor networking stacks, focusing on the need for

automated and dynamic network control in large-scale Linux-based

virtualization environments.

The goal with Open vSwitch is to keep the in-kernel code as small as

possible (as is necessary for performance) and to re-use existing

subsystems when applicable (for example Open vSwitch uses the existing

QoS stack). As of Linux 3.3, Open vSwitch is included as a part of the

kernel and packaging for the userspace utilities are available on most

popular distributions.