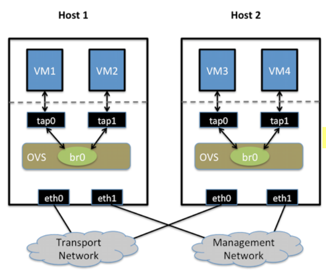
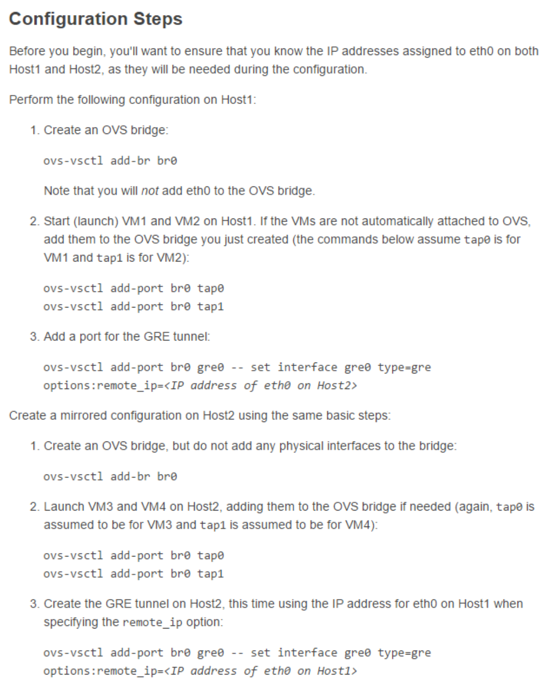
All about OVS

Friday, August 26, 2016 11:12 PM

http://openvswitch.org/support/config-cookbooks/port-tunneling/



Screen clipping taken: 8/26/2016 11:13 PM



Configuration Steps

Before you begin, you'll want to ensure that you know the IP addresses assigned to eth0 on both Host1 and Host2, as they will be needed during the configuration.

Perform the following configuration on Host1:

1. Create an OVS bridge:

```
ovs-vsctl add-br br0
```

   Note that you will *not* add eth0 to the OVS bridge.

2. Start (launch) VM1 and VM2 on Host1. If the VMs are not automatically attached to OVS, add them to the OVS bridge you just created (the commands below assume tap0 is for VM1 and tap1 is for VM2):

```
ovs-vsctl add-port br0 tap0
ovs-vsctl add-port br0 tap1
```

3. Add a port for the GRE tunnel:

```
ovs-vsctl add-port br0 gre0 -- set interface gre0 type=gre
options:remote_ip=<IP address of eth0 on Host2>
```

Create a mirrored configuration on Host2 using the same basic steps:

1. Create an OVS bridge, but do not add any physical interfaces to the bridge:

```
ovs-vsctl add-br br0
```

2. Launch VM3 and VM4 on Host2, adding them to the OVS bridge if needed (again, tap0 is assumed to be for VM3 and tap1 is assumed to be for VM4):

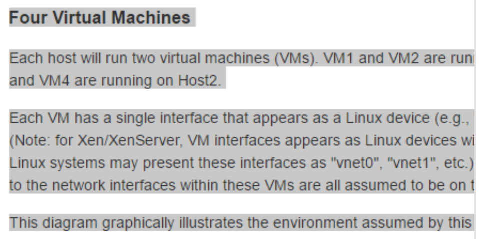
```
ovs-vsctl add-port br0 tap0
ovs-vsctl add-port br0 tap1
```

3. Create the GRE tunnel on Host2, this time using the IP address for eth0 on Host1 when specifying the remote_ip option:

```
ovs-vsctl add-port br0 gre0 -- set interface gre0 type=gre
options:remote_ip=<IP address of eth0 on Host1>
```

Screen clipping taken: 8/26/2016 11:13 PM

https://github.com/openvswitch/ovs/blob/master
FAQs by OVS developers.



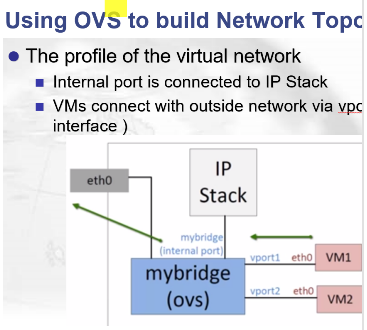
Four Virtual Machines

Each host will run two virtual machines (VMs). VM1 and VM2 are runn
and VM4 are running on Host2.

Each VM has a single interface that appears as a Linux device (e.g.,
(Note: for Xen/XenServer, VM interfaces appears as Linux devices wi
Linux systems may present these interfaces as "vnet0", "vnet1", etc.)
to the network interfaces within these VMs are all assumed to be on t

This diagram graphically illustrates the environment assumed by this

Screen clipping taken: 8/26/2016 11:14 PM

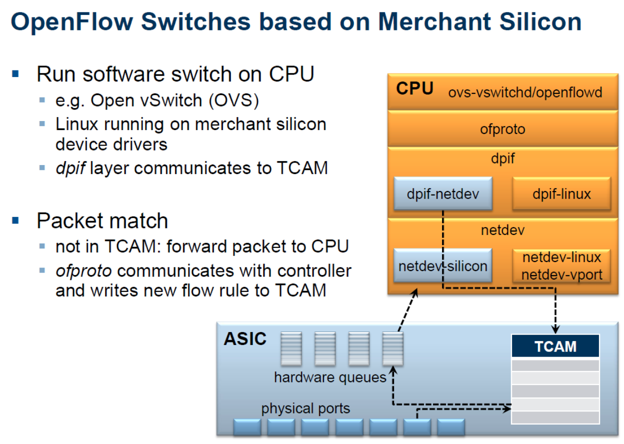


Using OVS to build Network Topo

- The profile of the virtual network
  - Internal port is connected to IP Stack
  - VMs connect with outside network via vpo interface)

Screen clipping taken: 8/26/2016 11:43 PM

To achieve this configuration, refer to the video on youtub
(OVS)



OpenFlow Switches based on Merchant Silicon

- Run software switch on CPU
  - e.g. Open vSwitch (OVS)
  - Linux running on merchant silicon device drivers
  - *dpif* layer communicates to TCAM
- Packet match
  - not in TCAM: forward packet to CPU
  - *ofproto* communicates with controller and writes new flow rule to TCAM

Ref: 06_SDN_Switches.pdf in NEC folder on my pen drive

What is a datapath?

In Open vSwitch terminology, a "datapath" is a flow-based software switch.

```
Using the Userspace Datapath with ovs-vswitchd
----------------------------------------------

To use ovs-vswitchd in userspace mode, create a bridge
"netdev" in the configuration database.  For example:

    ovs-vsctl add-br br0
    ovs-vsctl set bridge br0 datapath_type=netdev
    ovs-vsctl add-port br0 eth0
```

A datapath has no intelligence of its own.

Typically, the client of a datapath is the software switch module in "ovs-vswitchd", but other clients can be written.Ex: "ovs-dpctl" utility

What happens when a packet is received?

When a datapath "port" receives a packet, it extracts the headers (the "flow"). If the datapath's "flow table" contains a "flow entry" matching the packet, then it executes the "actions" in the flow entry and increments the flow's statistics. If there is no matching flow entry, the datapath instead appends the packet to an "upcall" queue.

```
* Ports
* =====
*
* A datapath has a set of ports that are analogous to the ports on an Ethernet
* switch.  At the datapath level, each port has the following information
* associated with it:
*
*   - A name, a short string that must be unique within the host.  This is
*     typically a name that would be familiar to the system administrator,
*     e.g. "eth0" or "vif1.1", but it is otherwise arbitrary.
*
*   - A 32-bit port number that must be unique within the datapath but is
*     otherwise arbitrary.  The port number is the most important identifier
*     for a port in the datapath interface.
*
*   - A type, a short string that identifies the kind of port.  On a Linux
*     host, typical types are "system" (for a network device such as eth0),
*     "internal" (for a simulated port used to connect to the TCP/IP stack),
*     and "gre" (for a GRE tunnel).
*
*   - A Netlink PID for each upcall reading thread (see "Upcall Queuing and
*     Ordering" below).
```

What is a dpif interface?

The dpif interface has functions for adding and deleting ports. When a datapath implements these (e.g. as the Linux and netdev datapaths do), then Open vSwitch's ovs-vswitchd daemon can directly control what ports are used for switching.

Each datapath must have a port, sometimes called the "local port", whose name is the same as the datapath itself, with port number 0. The local port cannot be deleted.

How are ports available for programming?

Ports are available as "struct netdev"s.

What is a flow table?

The flow table is a collection of "flow entries". Each flow entry contains:

- A "flow", that is, a summary of the headers in an Ethernet packet. In Open vSwitch userspace, "struct flow" is the typical way to describe a flow,
- A "mask" that, for each bit in the flow, specifies whether the datapath should consider the corresponding flow bit when deciding whether a given packet matches the flow entry.
- A list of "actions" that tell the datapath what to do with packets within a flow. Some examples of actions are OVS_ACTION_ATTR_OUTPUT, which transmits the packet out a port, and OVS_ACTION_ATTR_SET, which modifies packet headers.
- Statistics: the number of packets and bytes that the flow has processed, the last time that the flow processed a packet, and the union of all the TCP flags in packets processed by the flow. (The latter is 0 if the flow is not a TCP flow.)

The datapath's client manages the flow table, primarily in reaction to "upcalls" (see below).

What are upcalls?

```
* Upcalls
* =======
*
* A datapath sometimes needs to notify its client that a packet was received.
* The datapath mechanism to do this is called an "upcall".
*
* Upcalls are used in two situations:
*
*   - When a packet is received, but there is no matching flow entry in its
*     flow table (a flow table "miss"), this causes an upcall of type
```

```
    ovs-vsctl add-port br0 eth1
    ovs-vsctl add-port br0 eth2
```

```
ovs-vswitchd will create a TAP device as the bridge's l
named the same as the bridge, as well as for each confi
interface.
```

Firewall Rules

On Linux, when a physical interface is in use by the user-space datapath, packets received on the interface still also pass into the kernel TCP/IP stack. This can cause surprising and incorrect behavior. You can use "iptables" to avoid this behavior, by using it to drop received packets. For example, to drop packets received on eth0:

```
iptables -A INPUT -i eth0 -j DROP
iptables -A FORWARD -i eth0 -j DROP
```

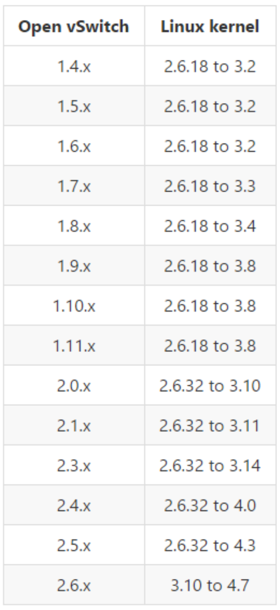
Reference:\ovs\INSTALL.userspace.md

```
           _
          |   +-------------------+
          |   |    ovs-vswitchd   |<-->ovsdb-server
          |   +-------------------+
          |   |      ofproto      |<-->OpenFlow controllers
          |   +--------+-+--------+  _
          |   | netdev | |ofproto-|   |
userspace |   +--------+ |  dpif  |   |
          |   | netdev | +--------+   |
          |   |provider| |  dpif  |   |
          |   +---||---+ +--------+   |
          |       ||     |  dpif  |   | implementation of
          |       ||     |provider|   | ofproto provider
          |_      ||     +---||---+   |
                  ||         ||       |
           _  +---||-----+---||---+   |
          |   |          |datapath|   |
   kernel |   |          +--------+  _|
          |   |                   |
          |_  +--------||---------+
                       ||
                    physical
                       NIC
```

```
* Typical Usage of Datapath by its clients
* =============
*
* Typically, the client of a datapath begins by configuring the da
* a set of ports.  Afterward, the client runs in a loop polling for
* arrive.
*
* For each upcall received, the client examines the enclosed pa
* figures out what should be done with it.  For example, if the c
* implements a MAC-learning switch, then it searches the forw
* for the packet's destination MAC and VLAN and determines th
to
* which it should be sent.  In any case, the client composes a se
* actions to properly dispatch the packet and then directs the d
* execute those actions on the packet (e.g. with dpif_execute())
*
* Most of the time, the actions that the client executed on the p
* to every packet with the same flow.  For example, the flow inc
* destination MAC and VLAN ID (and much more), so this is true
* MAC-learning switch example above.  In such a case, the clien
* direct the datapath to treat any further packets in the flow in
* way, using dpif_flow_put() to add a new flow entry.
*
*     Other tasks the client might need to perform, in addition to
*     UPCALLS, include:
*
*   - Periodically polling flow statistics, perhaps to supply to its
*     clients.
*
*   - Deleting flow entries from the datapath that haven't been
*     recently, to save memory.
```

*   DPIF_UC_MISS. These are called "miss" upcalls.
*
*   - A datapath action of type OVS_ACTION_ATTR_USERSPACE causes an upcall of
*   type DPIF_UC_ACTION. These are called "action" upcalls.
*
* An upcall contains an entire packet.
* After a client reads a given upcall, the datapath is finished with it, that
* is, the datapath doesn't maintain any lingering state past that point.
*
* To receive upcalls, a client has to enable them with dpif_recv_set().

Compilation:



Open vSwitch	Linux kernel
1.4.x	2.6.18 to 3.2
1.5.x	2.6.18 to 3.2
1.6.x	2.6.18 to 3.2
1.7.x	2.6.18 to 3.3
1.8.x	2.6.18 to 3.4
1.9.x	2.6.18 to 3.8
1.10.x	2.6.18 to 3.8
1.11.x	2.6.18 to 3.8
2.0.x	2.6.32 to 3.10
2.1.x	2.6.32 to 3.11
2.3.x	2.6.32 to 3.14
2.4.x	2.6.32 to 4.0
2.5.x	2.6.32 to 4.3
2.6.x	3.10 to 4.7

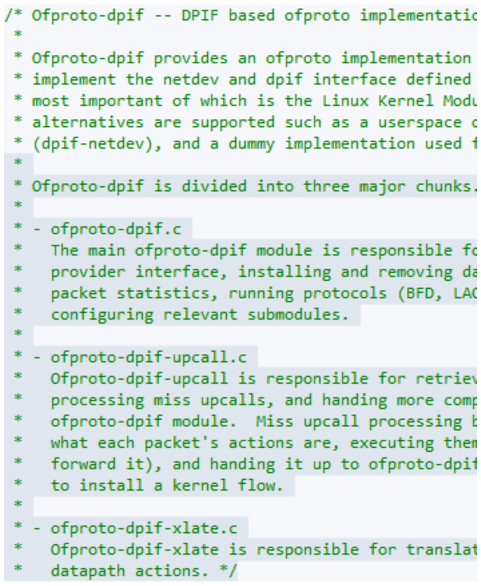
How to install OVS kernel module from source?

http://blog.ebiken.me/how-to-install-ovs-kernel-module-from-source-code/

*
*   - Updating flow entries whose actions should change. For e
*   MAC learning switch learns that a MAC has moved, then it
*   the actions of flow entries that sent packets to the MAC at
*   location.
*
*   - Adding and removing ports to achieve a new configuration

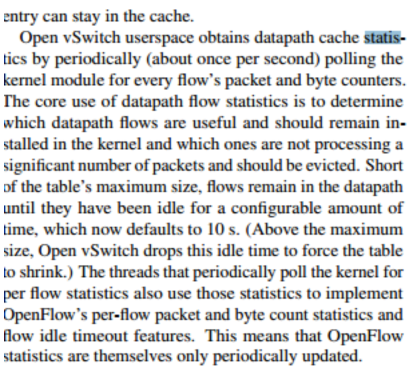
Design Decisions In Open vSwitch

https://github.com/openvswitch/ovs/blob/master/DESIGN.md#



```
/* Ofproto-dpif -- DPIF based ofproto implementatic
 *
 * Ofproto-dpif provides an ofproto implementation
 * implement the netdev and dpif interface defined
 * most important of which is the Linux Kernel Modu
 * alternatives are supported such as a userspace c
 * (dpif-netdev), and a dummy implementation used f
 *
 * Ofproto-dpif is divided into three major chunks.
 *
 * - ofproto-dpif.c
 *   The main ofproto-dpif module is responsible fc
 *   provider interface, installing and removing da
 *   packet statistics, running protocols (BFD, LAC
 *   configuring relevant submodules.
 *
 * - ofproto-dpif-upcall.c
 *   Ofproto-dpif-upcall is responsible for retriev
 *   processing miss upcalls, and handing more comp
 *   ofproto-dpif module.  Miss upcall processing b
 *   what each packet's actions are, executing them
 *   forward it), and handing it up to ofproto-dpif
 *   to install a kernel flow.
 *
 * - ofproto-dpif-xlate.c
 *   Ofproto-dpif-xlate is responsible for translat
 *   datapath actions. */
```

Screen clipping taken: 10/10/2016 2:54 PM



entry can stay in the cache.
Open vSwitch userspace obtains datapath cache statistics by periodically (about once per second) polling the kernel module for every flow's packet and byte counters. The core use of datapath flow statistics is to determine which datapath flows are useful and should remain installed in the kernel and which ones are not processing a significant number of packets and should be evicted. Short of the table's maximum size, flows remain in the datapath until they have been idle for a configurable amount of time, which now defaults to 10 s. (Above the maximum size, Open vSwitch drops this idle time to force the table to shrink.) The threads that periodically poll the kernel for per flow statistics also use those statistics to implement OpenFlow's per-flow packet and byte count statistics and flow idle timeout features. This means that OpenFlow statistics are themselves only periodically updated.

Screen clipping taken: 12/14/2016 6:50 PM

Above snapshot is taken from Open-Vswitch specification pa