An Introduction to Open vSwitch
LinuxCon Japan, Yokohama

Simon Horman <simon@horms.net>
Horms Solutions Ltd., Tokyo

2nd June 2011
Contents

- Introduction
- Management and Configuration Basics
- Examples of Advanced Configuration
Open vSwitch

- Multi-Layer Virtual Switch
- Flexible Controller in User-Space
- Fast Datapath in Kernel
- An implementation of Open Flow
Open vSwitch Availability

- Available from openvswitch.org
- Development code is available in git
- Announce, discussion and development mailing lists
- User-space (controller and tools) is under the Apache license
- Kernel (datapath) is under the GPLv2
- Shared headers are dual-licensed
Open vSwitch Concepts

- A switch contains ports
- A port may have one or more interfaces
  - Bonding allows more than one interface per port
- Packets are forward by flow
A flow may be identified by any combination of:
- Tunnel ID
- IPv6 ND target
- IPv4 or IPv6 source address
- IPv4 or IPv6 destination address
- Input port
- Ethernet frame type
- VLAN ID (802.1Q)
- TCP/UDP source port
- TCP/UDP destination port
- Ethernet source address
- Ethernet destination address
- IP Protocol or lower 8 bits of ARP ppcode
- IP ToS (DSCP field)
- ARP/ND source hardware address
- ARP/ND destination hardware address
1. The first packet of a flow is sent to the controller.
2. The controller programs the datapath’s actions for a flow:
   - Usually one, but may be a list
   - Actions include:
     - Forward to a port or ports, mirror
     - Encapsulate and forward to controller
     - Drop
3. And returns the packet to the datapath.
4. Subsequent packets are handled directly by the datapath.
Open vSwitch Management

- Open vSwitch controller is configured via a JSON database
- Database and thus configuration is persistent across reboots
- Database actions won’t return until the controller is reconfigured
- JSON database may be controlled locally using a UNIX socket or remotely using TLS (SSL)
Basic Configuration

1. Ensure that Open vSwitch is running
   
   /etc/init.d/openvswitch-switch start

2. Create a bridge

   ovs-vsctl -- --may-exist add-br br0

3. Add port to a bridge

   ovs-vsctl -- --may-exist add-port br0 eth0
1. Ensure that Open vSwitch is running
   /etc/init.d/openvswitch-switch start

2. Remove a port from a bridge
   ovs-vsctl -- --if-exists del-port br0 eth0

3. Remove a bridge
   ovs-vsctl -- --if-exists del-br br0
Examples of Advanced Configuration

- Port Mirroring (SPAN)
- QoS
Port Mirroring (SPAN)

- Allows frames sent to or received on one or more ports to be duplicated on a different port
- Useful for debugging
1. Create a dummy interface that will receive mirrored packets
   modprobe dummy
   ip link set up dummy0
   modprobe dummy

2. Add the dummy interface to the bridge in use
   ovs-vsctl add-port br0 dummy0
Port Mirroring Configuration (Target)

1. Create a mirror
   
   ```bash
   ovs-vsctl \ 
     -- --id=@m create mirror name=mirror0 \ 
     -- add bridge br0 mirrors @m
   ```

2. Find the UUID of the target interface
   
   ```bash
   ovs-vsctl list port dummy0
   _uuid : 4d5ed382-a0c3-4453-ab3c-58e1e7f603b0
   ...
   ```

3. Configure the mirror to output mirrored packets to the target interface
   
   ```bash
   ovs-vsctl set mirror mirror0 \ 
     output_port=4d5ed382-a0c3-4453-ab3c-58e1e7f603b0
   ```
Port Mirroring Configuration (Selected Source)

- All packets sent to or received from tap0 will be mirrored on dummy0
- All flooded packets will go to dummy0

1. Find the UUID of the port or ports whose packets should be mirrored
   
   ```bash
   ovs-vsctl list port tap0
   _uuid : d624f5b1-f5e3-4f85-a907-bd209b5463aa
   ...
   ```

2. Mirror packets sent to and received from the interface of interest
   
   ```bash
   ovs-vsctl set mirror mirror0 \ select\_dst\_port=d624f5b1-f5e3-4f85-a907-bd209b5463aa
   ovs-vsctl set mirror mirror0 \ select\_src\_port=d624f5b1-f5e3-4f85-a907-bd209b5463aa
   ```
Port Mirroring Configuration (All Sources)

- All switch packets will go to dummy0

1. `ovs-vsctl set mirror mirror0 select_all=1`
Open vSwitch QoS capabilities

- Interface rate limiting
- Port QoS policy
QoS: Interface rate limiting

- A rate and burst can be assigned to an Interface
- Conceptually similar to Xen’s netback credit scheduler
- Utilises the Kernel tc framework’s ingress policing
- Simple

Configuration example. 100Mbit/s rate with 10Mbit/s burst:

```
# ovs-vsctl set Interface tap0 ingress_policing_rate=100000
# ovs-vsctl set Interface tap0 ingress_policing_burst=10000
```
QoS: Control: No interface rate limiting

# netperf -4 -t UDP_STREAM -H 172.17.50.253 -- -m 8972
UDP UNIDIRECTIONAL SEND TEST from 0.0.0.0 (0.0.0.0)...  

<table>
<thead>
<tr>
<th>Socket</th>
<th>Message</th>
<th>Elapsed</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Size</td>
<td>Time</td>
<td>Okay</td>
</tr>
<tr>
<td>bytes</td>
<td>bytes</td>
<td>secs</td>
<td>#</td>
</tr>
</tbody>
</table>

| 120832 | 8972    | 10.01   | 146797   | 0       | 1052.60     |
| 109568 | 10.01   | 146620  | 1051.33  |         |             |

- tap networking used
- jumbo frames required to reach line speed  
  (≈210Mbits/s with 1500 byte frames)
- virtio does much better
QoS: Interface rate limiting result

```
# netperf -4 -t UDP_STREAM -H 172.17.50.253
UDP UNIDIRECTIONAL SEND TEST from 0.0.0.0 (0.0.0.0)...
```

<table>
<thead>
<tr>
<th>Socket</th>
<th>Message</th>
<th>Elapsed</th>
<th>Messages</th>
<th>Size</th>
<th>Size</th>
<th>Time</th>
<th>Okay</th>
<th>Errors</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bytes</td>
<td>bytes</td>
<td>secs</td>
<td>#</td>
<td>#</td>
<td>10^6bits/sec</td>
</tr>
<tr>
<td>120832</td>
<td>8972</td>
<td>10.01</td>
<td>149735</td>
<td>0</td>
<td>0</td>
<td>1073.66</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>109568</td>
<td>10.01</td>
<td>14684</td>
<td>0</td>
<td>105.29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Difference in sent and received packets indicates that excess packets are dropped – no backpressure
- This is an inherent problem when using ingress policying
QoS: Port QoS policy

- A port may be assigned one or more QoS policies.
- Each QoS policy consists of a class and a qdisc.
- Classes and qdisc use the Linux kernel’s tc implementation.
- Only HTB and HFSC classes are supported at this time.
- The class of a flow is chosen by the controller.
- The QoS policy (i.e., class) of a flow is chosen by the controller.
- Operates as an egress filter.
QoS: Port QoS policy example

Programming the Datapath

1:# ovs-vsctl set port eth1 qos=@newqos \\
2:   -- --id=@newqos create qos type=linux-htb \\
3:       other-config:max-rate=200000000 queues=0=@q0,1=@q1 \\
4:   -- --id=@q0 create queue \\
5:       other-config:min-rate=100000000 \\
6:       other-config:max-rate=100000000 \\
7:   -- --id=@q1 create queue \\
8:       other-config:min-rate=500000000 \\
9:       other-config:max-rate=500000000

Line numbers added for clarity
QoS: Port QoS policy example

Hard-coding the controller

# ovs-ofctl add-flow br0 "in_port=2 ip nw_dst=172.17.50.253 \ idle_timeout=0 actions=enqueue:1:0"
# ovs-ofctl add-flow br0 "in_port=3 ip nw_dst=172.17.50.253 \ idle_timeout=0 actions=enqueue:1:1"

Only suitable for testing
QoS: Port QoS policy example

Guest 0:
# netperf -4 -t TCP_STREAM -H 172.17.50.253 -l 30 -- -m 8972
TCP STREAM TEST from 0.0.0.0 (0.0.0.0)...
Recv    Send    Send
Socket  Socket  Message  Elapsed
Size    Size    Size     Time     Throughput
bytes   bytes   bytes    secs.    10^6bits/sec
87380   16384   8972     30.01    99.12

Guest 1:
# netperf -4 -t TCP_STREAM -H 172.17.50.253 -l 30 -- -m 8972
...
87380   16384   8972     30.14    49.56
QoS: Port QoS policy controller improvements

- Add a default queue to the Port table
- Add enqueue to the FLOOD and NORMAL ports
Per-Customer VLANs are desirable for security reasons
But there is a limit of 4094 VLANs
More VLANs

Two, apparently competing, approaches

- IETF / Cisco
  - RFC5517 — Private VLANs
- IEEE
  - 802.1ad — Provider Bridges (Q-in-Q)
  - 802.1ah — Provider Backbone Brides (MAC-in-MAC)
RFC5517 — Private VLANs

- Uses existing 802.1Q framing
  - Simple to implement (in software/firmware)
- Makes use of pairs of VIDs
  - Requires all switches to support of Private VLANs
    otherwise switch tables may not merge
- Provides L2 broadcast isolation
  - Forwarding may occur at L3
    - Requires the router to perform proxy ARP
- Currently not supported by Open vSwitch
Three VLAN classifications

- **Promiscuous**
  - May communicate with endpoints on any port
  - e.g.: Gateway, Management Host

- **Community**
  - May only communicate with endpoints on promiscuous ports or ports belonging to the same community
  - e.g.: Different hosts belonging to the same customer

- **Isolated**
  - May only communicate with endpoints on promiscuous ports
  - e.g.: Hosts that only require access to the gateway
- **Promiscous domain (P)**
  - May communicate with endpoints in the same domain and sub-domains
- **Two community sub-domains** \((C_1, C_2)\)
  - May communicate with endpoints in the same domain and parent-domain
- **Isolated sub-domain** \((I)\)
  - May communicate with endpoints in the parent domain
  - May *not* communicate with endpoints in the same domain
802.1ad — Provider Bridges (Q-in-Q)

- Current standard is 802.1ad-2005, Approved December 2005
- Builds on 802.1Q
- New Framing
  - C-VID (inner)
    - Renamed 802.1Q VID
    - There may be more than one C-VID (inner-inner, ...)
  - S-VID (outer)
    - Different ether-type to C-VID
    - May be translated
- Currently not supported by Linux Kernel / Open vSwitch
# 802.1ad Framing — Provider Bridges

<table>
<thead>
<tr>
<th>DA</th>
<th>Destination MAC address</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>Source MAC address</td>
</tr>
<tr>
<td>S-VID</td>
<td>Service VLAN ID</td>
</tr>
<tr>
<td>C-VID</td>
<td>Customer VLAN ID</td>
</tr>
<tr>
<td>VID</td>
<td>VLAN ID</td>
</tr>
</tbody>
</table>

---

![Diagram showing 802.1, 802.1Q, and 802.1ad framing models](image)
Current standard is 802.1ah-2008, Approved August 2008

Builds on 802.1ad

New Framing

- MAC encapsulation provides full Client VLAN isolation
  - Inner MAC is unknown outside of its scope
- I-SID: Up to $2^{24} \approx 16$ million backbone services
- I-VID semantics are the same as the S-VLAN
  - Only edge switches need to be Provider Backbone Bridge aware
  - Core switches need only be Provider Bridge (802.1ad) aware

Currently not supported by Linux Kernel / Open vSwitch
802.1ah Framing — Provider Backbone Bridges

<table>
<thead>
<tr>
<th>B-DA</th>
<th>Backbone Destination MAC address</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-SA</td>
<td>Backbone Source MAC address</td>
</tr>
<tr>
<td>B-VID</td>
<td>Backbone VLAN ID</td>
</tr>
<tr>
<td>I-SID</td>
<td>Service ID</td>
</tr>
<tr>
<td>DA</td>
<td>Destination MAC address</td>
</tr>
<tr>
<td>SA</td>
<td>Source MAC address</td>
</tr>
<tr>
<td>S-VID</td>
<td>Service VLAN ID</td>
</tr>
<tr>
<td>C-VID</td>
<td>Customer VLAN ID</td>
</tr>
<tr>
<td>VID</td>
<td>VLAN ID</td>
</tr>
</tbody>
</table>

802.1 | 802.1Q | 802.1ad | 802.1ah
---|---|---|---
DA | DA | DA | DA
SA | SA | SA | SA
VID | VID | VID | VID
S-VID | S-VID | S-VID | S-VID
C-VID | C-VID | C-VID | C-VID
Payload | Payload | Payload | Payload