

Open vSwitch Test Plan

Mick Tarsel
LTC Networking Team

April 14, 2017

Contents

1	Introduction and Methodology	4
2	Installation	5
2.1	RHEL: Installing .rpm	5
2.2	Ubuntu: Installing .deb	5
3	Verifying Installation	5
4	Testing Basic OvS Configurations	6
4.1	OvS Database	6
4.2	Create a Bridge	7
4.2.1	Verify the Bridge Has Been Created	7
4.3	Adding Flow to Bridge	8
4.3.1	Verify a New Flow is Added	8
4.4	Removing a Bridge	8
5	Connecting NS to OvS Bridge	9
5.1	Setup	9
5.1.1	Create Bridge	9
5.1.2	Create a NS with IP Address	9
5.1.3	Attach NS Port to OvS Bridge	10
5.2	Verifying NS Connection to Bridge	10
5.3	Ping Verification	11
5.4	Clean Up	11
6	VLAN Testing	12
6.1	Setup	12
6.1.1	Assigning Tags to Ports	14
6.2	Ping Verification	14
6.3	Creating a Fake Bridge	15
6.3.1	Verifying Fake Bridge	16
6.4	Clean Up	17
7	Bonding - Active Backup	17
7.1	Setup	18
7.1.1	Verifying Bond Interface	19
7.2	Ping Verification	20
7.3	Netcat Tests	20
7.4	Clean Up	21
8	SPAN Port	21
8.1	Setup	21
8.2	Verifying Traffic Mirror	23
8.3	Clean Up	23
9	QoS	24
9.1	Setup	24
9.2	Verifying QoS	25
9.3	Clean Up	26
10	Multi-Host Tests	27
10.1	GRE Tunnel	27
10.2	Setup	27
10.2.1	Host 1 Setup	27
10.2.2	Host 2 Setup	29
10.3	Ping Verification	30

10.4 Clean Up	31
11 TODO: VXLAN	31
Appendices	32
A Troubleshooting Installation	32
A.1 OvS Kernel Module	32
A.2 Starting OvS Service	32
A.2.1 RHEL	32
A.2.2 Ubuntu	32
B Name Spaces	33
B.1 Creating Name Spaces	33
B.2 Deleting Name Space	33
B.3 Creating Veth Interfaces	33
B.4 Attaching Veth Device to Name Space	34
B.5 Executing Commands in a Name Space	34
C Setting Up Name Spaces for Test Environment	34
C.1 Troubleshooting: Adding Port to Bridge	35
D Installing Netperf	36
D.1 Troubleshooting Installation	36
E Installing netcat	36
F Installing tcpdump	37
G Multiple Host Trouble Shooting	37
G.1 GRE Tunnels	37

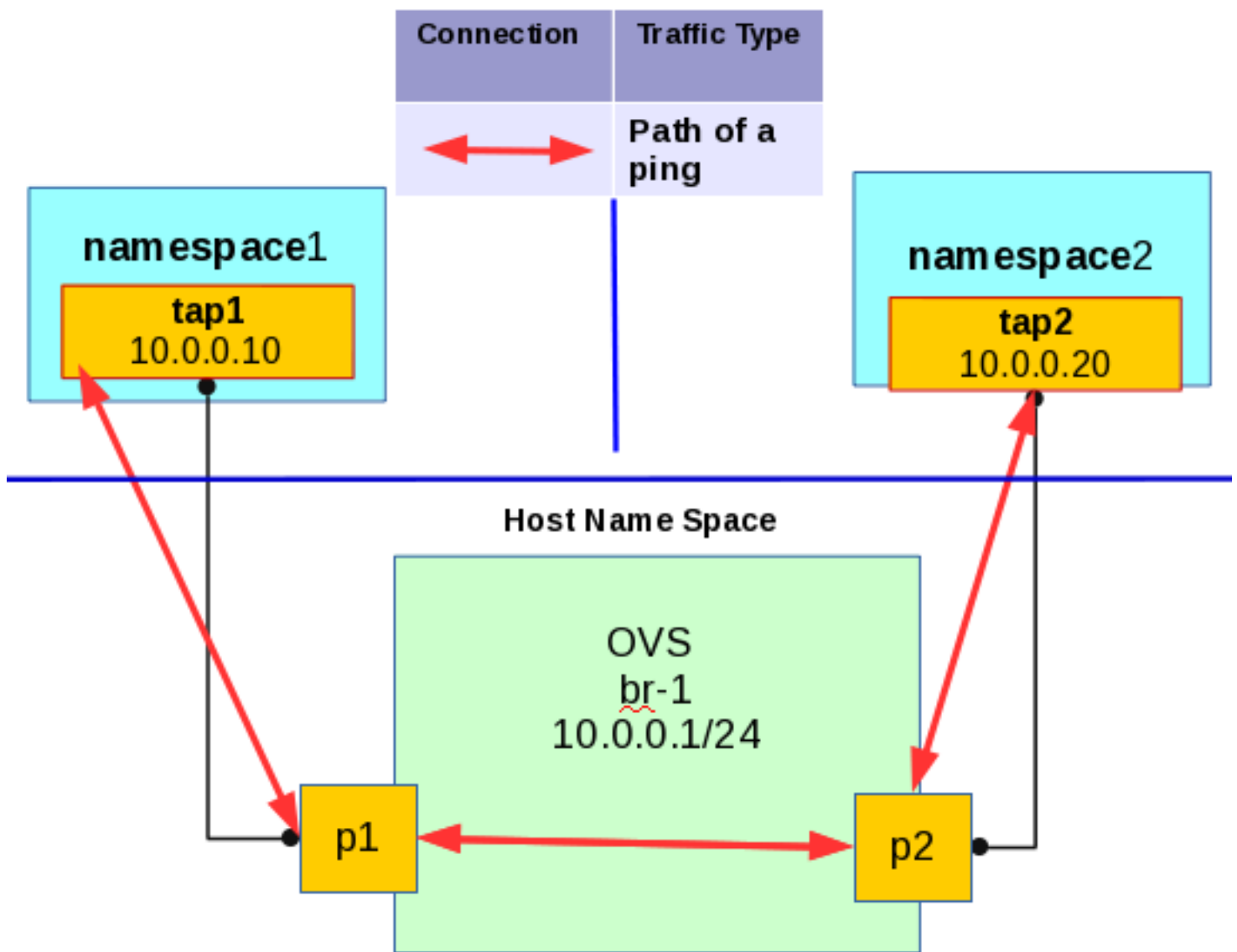
1 Introduction and Methodology

The purpose of this test plan is to test Open vSwitch (OvS). The document provides commands and the expected output. This test plan aims to provide a pass/failure reference point based on the output of the command. Output may change over time due to the nature of this project, however certain information should still be retained from output to signify a pass or failure. Important information from output will be **highlighted**. Verify that your output has something similar to the highlighted parts of the output.

Each section is self-containing such that you may skip around the document. After setup and execution of tests, the test concludes with a verification section. Highlighted output missing from verification section would indicate a failure. If there is a possibility of a failure refer to the Appendix first. Make sure you remove previous bridges or interfaces before moving to the next test in case there is a naming conflict.

This test plan utilizes network name spaces (ns) to create virtual network endpoints in order to test the features of OvS. Each network name space will be an endpoint on our virtual switch. The picture below represents the basic network topology this test plan will use.

Figure 1: Path of ICMP packet from namespace1 through br-1 to namespace2



Looking at the diagram above we see 3 name spaces. One root name space, and 2 network name spaces. The veth peers for namespace1 are tap1 and p1. tap1 device exists only inside namespace1. Much similar to a VM having a different

Ethernet interface than the host machine. *Please note that root namespace and host are used interchanbly throughout the document.* p1 exists on the host and in this picture it has been added as a port to the OvS bridge named br-1. Additional information about name spaces may be found in appendix B.

Commands below are going to be used most often in this test plan

- `ovs-vsctl` : Used for configuring the ovs-vswitchd configuration database (known as ovs-db)
- `ovs-ofctl` : A command line tool for monitoring and administering OpenFlow switches
- `ovsdb-tool` : Open vSwitch database management utility
- `netperf` : a network performance benchmark
- `tcpdump` : dump traffic on a network
- `netcat` : create, concatenate, and redirect sockets.

Any reference to another section will be a clickable link to that section.

2 Installation

2.1 RHEL: Installing .rpm

```
# yum install openvswitch
```

Verify the `openvswitch.service` is running with `systemctl`

```
# systemctl status openvswitch
openvswitch.service - LSB: Open vSwitch switch
  Loaded: loaded (/etc/rc.d/init.d/openvswitch; bad; vendor preset: disabled)
  Active: active (running) since Thu 2017-03-16 14:30:46 PDT; 28min ago
```

Here we can see that its Active: active (running)

2.2 Ubuntu: Installing .deb

```
# apt-get install openvswitch-common
# apt-get install openvswitch-switch
```

Verify the `openvswitch-switch.service` is active

```
# systemctl status openvswitch-switch.service
openvswitch-switch.service - Open vSwitch
  Loaded: loaded (/lib/systemd/system/openvswitch-switch.service; enabled;)
  Active: active (exited) since Wed 2017-03-22 18:08:12 EDT; 6 days ago
  Process: 5754 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
  Main PID: 5754 (code=exited, status=0/SUCCESS)
  Tasks: 0 (limit: 4915)
  CGroup: /system.slice/openvswitch-switch.service
```

```
Mar 22 18:08:12 ubu-1610 systemd[1]: Starting Open vSwitch...
Mar 22 18:08:12 ubu-1610 systemd[1]: Started Open vSwitch.
```

3 Verifying Installation

```
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
  ovs_version: "2.7.0"
```

Check that `ovs-vswitchd` and `ovsdb-server` are running. Use `ps -aef` command like so:

```
# ps -aef | grep ovs
root      6528   6482   grep --color=auto ovs
root      25748      1  ovsdb-server: monitoring pid 25749 (healthy)
root      25749  25748  ovsdb-server /etc/openvswitch/conf.db -vconsole:emer
                        -vsyslog:err -vfile:info
                        --remote=punix:/var/run/openvswitch/db.sock
                        --private-key=db:Open_vSwitch,SSL,private_key
                        --certificate=db:Open_vSwitch,SSL,certificate
                        --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert --no-chdir
                        --log-file=/var/log/openvswitch/ovsdb-server.log
                        --pidfile=/var/run/openvswitch/ovsdb-server.pid --detach
root      25759      1  ovs-vswitchd: monitoring pid 25760 (healthy)
root      25760  25759  ovs-vswitchd unix:/var/run/openvswitch/db.sock
                        -vconsole:emer -vsyslog:err -vfile:info --mlockall --no-chdir
                        --log-file=/var/log/openvswitch/ovs-vswitchd.log
                        --pidfile=/var/run/openvswitch/ovs-vswitchd.pid --detach
```

Ensure the Open vSwitch kernel module is loaded.

```
# lsmod | grep openv
openvswitch      159225  2
nf_defrag_ipv6   31537  2 openvswitch,nf_conntrack_ipv6
nf_nat_ipv6      11592  2 openvswitch,ip6table_nat
nf_nat_ipv4      10744  2 openvswitch,iptable_nat
nf_nat           22593  3 openvswitch,nf_nat_ipv4,nf_nat_ipv6
nf_conntrack     135188  7 openvswitch,nf_nat,nf_nat_ipv4,nf_nat_ipv6,
libcrc32c        1614  2 xfs,openvswitch

# modinfo openvswitch
filename:         /lib/modules/3.10.0-514.el7.ppc64le/kernel/net/
                  openvswitch/openvswitch.ko
license:          GPL
description:       Open vSwitch switching datapath
rhelversion:       7.3
srcversion:        B31AE95554C9D9A0067F935
depends:            nf_conntrack,nf_nat,libcrc32c,nf_nat_ipv6,nf_nat_ipv4
intree:           Y
vermagic:          3.10.0-514.el7.ppc64le SMP mod_unload modversions
signer:            Red Hat Enterprise Linux kernel signing key
sig_key:           12:17:2E:5E:C7:C7:70:4C:68:03:23:AD:F1:D5:8C:88:66:C0:A6:AB
sig_hashalgo:      sha256
```

For further instructions and troubleshooting refer to appendix A

4 Testing Basic OvS Configurations

4.1 OvS Database

First determine where your db is located. Print out the running OvS processes and search for the .db file location:

```
# ps -aef | grep openv | grep db
root 8521 8520 0 14:04 ? 00:00:00 ovsdb-server /etc/openvswitch/conf.db
                        -vconsole:emer -vsyslog:err -vfile:info
                        --remote=punix:/var/run/openvswitch/db.sock
                        --private-key=db:Open_vSwitch,SSL,private_key
                        --certificate=db:Open_vSwitch,SSL,certificate
                        --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert
                        --no-chdir --log-file=/var/log/openvswitch/ovsdb-server.log
                        --pidfile=/var/run/openvswitch/ovsdb-server.pid --detach --monitor
```

```
root 8532 8531 0 14:04 ? 00:00:00 ovs-vswitchd unix:/var/run/openvswitch/db.sock
-vconsole:emer -vsyslog:err -vfile:info --mlockall --no-chdir
--log-file=/var/log/openvswitch/ovs-vswitchd.log
--pidfile=/var/run/openvswitch/ovs-vswitchd.pid --detach --monitor
```

Here we can see PID: 8521 is running the ovdb-server from the database located in /etc/openvswitch/conf.db
To see the version number of the schema embedded within the database db

```
# ovsdb-tool db-version /etc/openvswitch/conf.db
7.14.0
```

View the newest logs from the database like so:

```
# ovsdb-tool show-log /etc/openvswitch/conf.db | tail
record 962: 2017-03-27 18:17:13.646
record 963: 2017-03-27 18:17:13.646
record 964: 2017-03-27 18:17:44.863 "ovs-vsctl (invoked by -bash):
        ovs-vsctl add-port br-1 p1"
record 965: 2017-03-27 18:17:44.867
record 966: 2017-03-27 19:07:17.562 "ovs-vsctl (invoked by -bash):
        ovs-vsctl del-br br-1"
record 967: 2017-03-27 19:07:17.672
record 969: 2017-03-27 21:07:51.038 "ovs-vsctl (invoked by -bash):
        ovs-vsctl add-br br-1"
record 970: 2017-03-27 21:07:51.088
record 971: 2017-03-27 21:07:51.089
```

If we add a new bridge called br-3 we will see the logs updated:

```
# ovs-vsctl add-br br-3
# ovsdb-tool show-log /etc/openvswitch/conf.db | tail
record 965: 2017-03-27 18:17:44.867
record 966: 2017-03-27 19:07:17.562 "ovs-vsctl (invoked by -bash):
        ovs-vsctl del-br br-1"
record 967: 2017-03-27 19:07:17.672
record 969: 2017-03-27 21:07:51.038 "ovs-vsctl (invoked by -bash):
        ovs-vsctl add-br br-1"
record 970: 2017-03-27 21:07:51.088
record 971: 2017-03-27 21:07:51.089
record 972: 2017-03-27 21:20:02.145 "ovs-vsctl (invoked by -bash):
        ovs-vsctl add-br br-3"
record 973: 2017-03-27 21:20:02.169
record 974: 2017-03-27 21:20:02.169
```

4.2 Create a Bridge

```
# ovs-vsctl add-br br-1
#
```

A new Open vSwitch bridge has been created.

```
# brctl show
bridge name        bridge id          STP enabled        interfaces
#
```

4.2.1 Verify the Bridge Has Been Created

```
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
```

```
Port "br-1"
    Interface "br-1"
        type: internal
    ovs_version: "2.7.0"
```

A new bridge interface has been created, in this example named br-1

```
# ip add | grep br-1
br-1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether ba:db:f2:e7:8f:4a brd ff:ff:ff:ff:ff:ff
```

New records have been written to the database at nearly the same time our bridge was created:

```
# ovsdb-tool show-log /etc/openvswitch/conf.db | tail
record 984: 2017-03-29 01:29:12.859 "ovs-vsctl (invoked by -bash):
    ovs-vsctl add-br br-1"
record 985: 2017-03-29 01:29:12.887
record 986: 2017-03-29 01:29:12.887
```

A new flow table for the new interface br-1 has been added.

```
# ovs-ofctl show br-1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000badbf2e78f4a
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
        mod_nw_tos mod_tp_src mod_tp_dst
LOCAL(br-1): addr:ba:db:f2:e7:8f:4a
    config:      PORT_DOWN
    state:       LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

4.3 Adding Flow to Bridge

```
# ovs-ofctl dump-flows br-1
NXST_FLOW reply (xid=0x4):
    cookie=0x0, duration=11262.392s, table=0, n_packets=16, n_bytes=1296,
    idle_age=11249, priority=0 actions=NORMAL
```

Add a new flow (using OpenFlow) to Table 0 that drops invalid multicast packets.

```
# ovs-ofctl add-flow br0 \
"table=0, dl_src=01:00:00:00:00:00/01:00:00:00:00:00, actions=drop"
```

We added a new flow using the OpenFlow match field dl_src. This flow will drop any broadcast or multicast packets with an invalid source MAC address and invalid mask address. The source address and mask address should not be the same.

4.3.1 Verify a New Flow is Added

```
# ovs-ofctl dump-flows br-1
NXST_FLOW reply (xid=0x4):
    cookie=0x0, duration=4.678s, table=0, n_packets=0, n_bytes=0,
    idle_age=4, dl_src=01:00:00:00:00:00/01:00:00:00:00:00 actions=drop
    cookie=0x0, duration=11350.365s, table=0, n_packets=16, n_bytes=1296,
    idle_age=11337, priority=0 actions=NORMAL
```

4.4 Removing a Bridge

```
# ovs-vsctl del-br br-1
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    ovs_version: "2.7.0"
```



```
# ip addr | grep br-1
#
```

5 Connecting NS to OvS Bridge

5.1 Setup

- 1 Bridge
- 1 Name space

5.1.1 Create Bridge

```
#!/bin/bash

# Create OVS switch
ovs-vsctl add-br br-1

#assign IP address
ip addr add 10.0.0.1/24 dev br-1

#bring the bridge interface up
ip link set br-1 up
```

Verify that you see the bridge:

```
# ip addr | grep br-1
4: br-1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    UNKNOWN qlen 1000
    inet 10.0.0.1/24 scope global br-1
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "br-1"
            Interface "br-1"
                type: internal
    ovs_version: "2.7.0"
```

5.1.2 Create a NS with IP Address

```
#!/bin/bash

#create name spaces
ip netns add ns1

# create a port pair. the tap1 exists in the ns, p1 on host
ip link add tap1 type veth peer name p1

#attach namespace dev tap1 to ns1 such that tap1 exists only in ns1
ip link set tap1 netns ns1

#turn on port on OVS. exists on host network
ip link set dev p1 up

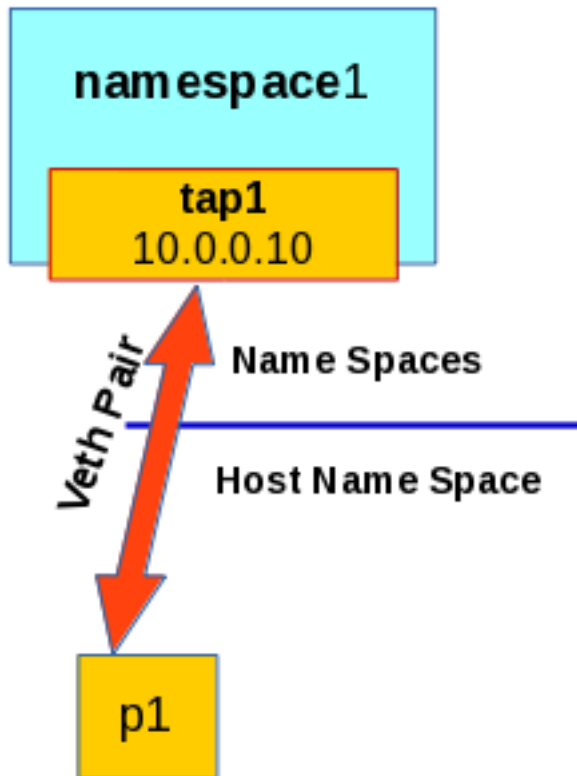
#assign ip addresses.
ip netns exec ns1 ip addr add 10.0.0.10/24 dev tap1

#turn ports on in ns. execute command inside ns
ip netns exec ns1 ip link set dev tap1 up
```

5.1.3 Attach NS Port to OvS Bridge

We have created a name space with a network interface. The network interface inside the name space is named `tap1`. This interface will be connected to the `p1` interface on the host machine.

Figure 2: Veth Pair `tap1` inside `namespace1` and `p1` on host network



Attach the interface on the host to the OvS bridge `br-1`

```
# ovs-vsctl add-port br-1 p1
#
```

Verify it exists on the bridge

```
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "p1"
            Interface "p1"
        Port "br-1"
            Interface "br-1"
                type: internal
    ovs_version: "2.7.0"
```

5.2 Verifying NS Connection to Bridge

Look at stats of `p1` interface on the bridge

```
# ovs-ofctl dump-ports br-1 p1
OFPST_PORT reply (xid=0x4): 1 ports
    port 1: rx pkts=8, bytes=648, drop=0, errs=0, frame=0, over=0, crc=0
            tx pkts=10, bytes=788, drop=0, errs=0, coll=0
```

There is a new flow table for `p1`

```
# ovs-ofctl show br-1
OFPT_FEATURES_REPLY (xid=0x2): dpid:000006cc0333ba4b
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
          mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
1(p1): addr:76:20:7f:2a:62:1a
    config:      0
    state:       0
    current:     10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
LOCAL(br-1): addr:06:cc:03:33:ba:4b
    config:      0
    state:       0
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

5.3 Ping Verification

ping bridge br-1 from ns1

```
# ip netns exec ns1 ping -c2 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp seq=1 ttl=64 time=0.166 ms
64 bytes from 10.0.0.1: icmp seq=2 ttl=64 time=0.019 ms

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.019/0.092/0.166/0.074 ms
```

Ping ns1 from root namespace

```
# ping -c2 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp seq=1 ttl=64 time=0.480 ms
64 bytes from 10.0.0.10: icmp seq=2 ttl=64 time=0.024 ms

--- 10.0.0.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.024/0.252/0.480/0.228 ms
```

Additionally, the usage stats from `ovs-ofctl dump-ports br-1 p1` should have increased.

5.4 Clean Up

```
# ovs-vsctl del-port p1
#
```

Even though we removed the interface from the bridge, p1 still exists in the root namespace

```
# ip add | grep p1
392: p1@if393: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    qlen 1000
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "br-1"
            Interface "br-1"
                type: internal
    ovs_version: "2.7.0"
```

Remove bridge.

```
# ovs-vsctl del-br br-1
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    ovs_version: "2.7.0"
# ip add | grep br-1
#
```

Remove ns

```
# ip netns delete ns1
#
```

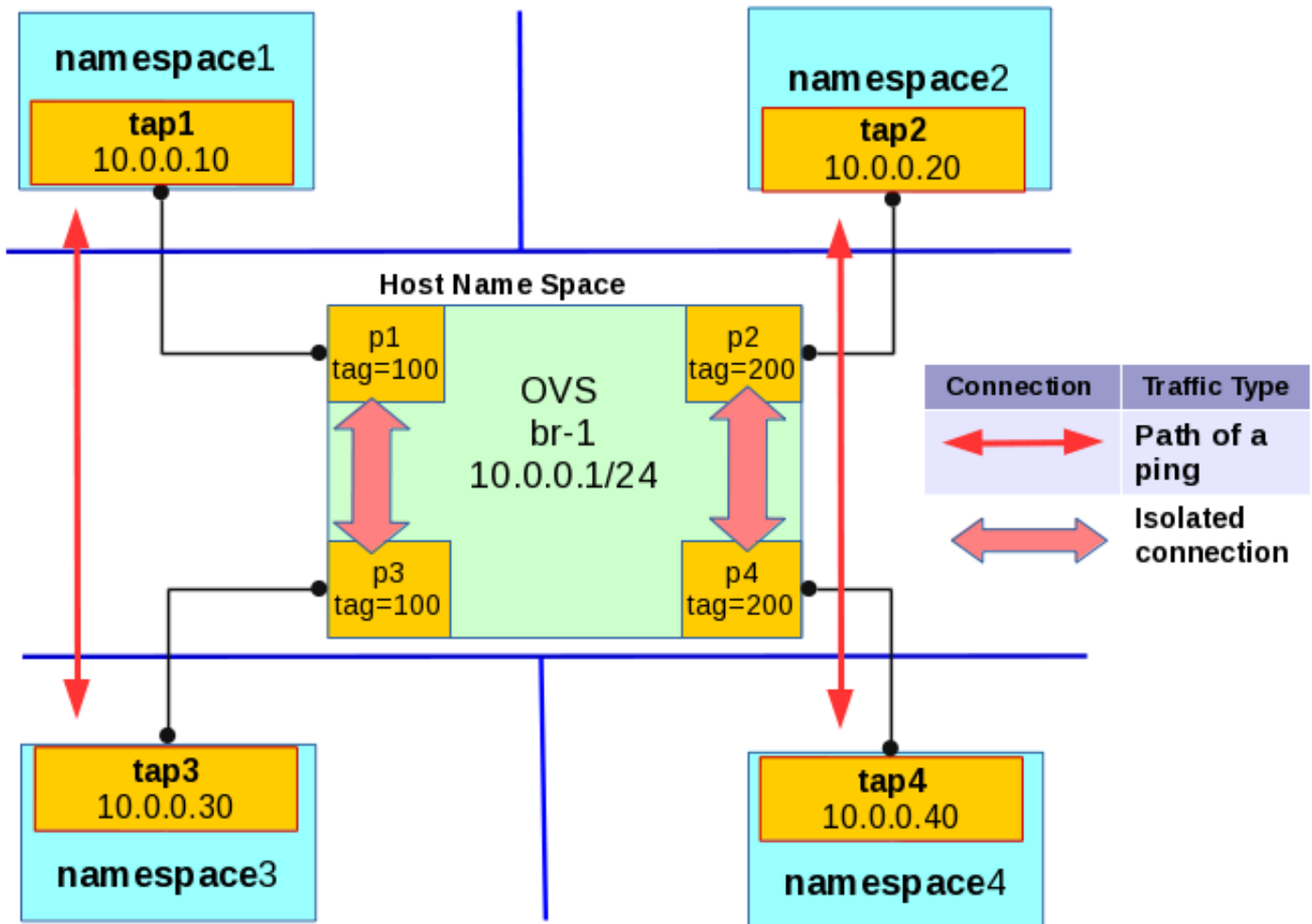
6 VLAN Testing

By default, all OvS ports are VLAN trunk ports. Packets sent and received on a trunk port are able to carry a VLAN header that says what VLAN the packet belongs to.

Using VLAN tags, we can connect 2 name spaces to each other with a specific tag. ns1 will have the same VLAN tag as ns3 such that ns1 cannot ping ns2 because it is not in the same VLAN.

These settings are ephemeral. That is, when you power off the guest, the port and its associated configuration goes away.

Figure 3: VLAN tags are used to isolate namespace1 and namespace4 from namespace2 and namespace3



6.1 Setup

- 1 Bridge

- 4 Name spaces

Create a bridge like so.

```
#!/bin/bash

# Create OVS switch
ovs-vsctl add-br br-1

#assign IP address
ip addr add 10.0.0.1/24 dev br-1

#bring the bridge interface up
ip link set br-1 up
```

Create 4 name spaces attached to our bridge

```
#!/bin/bash

#Creates name spaces with ports attached to OVS.
#Assigns an IP address to name space devices

BR=1
NUMNS=4
IP=10.0.0.0

for i in `seq 1 $NUMNS`;
do
    #create name spaces
    ip netns add ns$i

    # create a port pair. the tap$i exists in the ns, br-p$i on host
    ip link add tap$i type veth peer name p$i

    #attach namespace port to OVS
    ovs-vsctl add-port br-$BR p$i

    #attach namespace dev tap$i to ns1 such that tap$i exists only in ns1
    ip link set tap$i netns ns$i

    #turn on port on OVS. exists on host network
    ip link set dev p$i up

    #add ten to last octet of network ip addr
    IP=$(echo $IP | awk -F. '{printf "%d.%d.%d.%d", $1,$2,$3,$4+10}')

    #assign ip addresses.
    ip netns exec ns$i ip add add $IP/24 dev tap$i

    #bring the interface in ns1 up
    ip netns exec ns$i ip link set dev tap$i up
done

#ovs-vsctl show
```

Each name space has a different IP address but belongs to same subnet.

```
# ip netns exec ns1 ip add | grep 10
```

```

tap1@if415: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    inet 10.0.0.10/24 brd 10.0.0.255 scope global tap1
#
# ip netns exec ns3 ip add | grep 10
tap3@if419: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    inet 10.0.0.30/24 brd 10.0.0.255 scope global tap3

```

6.1.1 Assigning Tags to Ports

```

# ovs-vsctl set port p1 tag=100
# ovs-vsctl set port p3 tag=100

```

```

# ovs-vsctl set port p2 tag=200
# ovs-vsctl set port p4 tag=200

```

```

# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "p2"
            tag: 200
            Interface "p2"
        Port "p3"
            tag: 100
            Interface "p3"
        Port "p4"
            tag: 200
            Interface "p4"
        Port "br-1"
            Interface "br-1"
                type: internal
        Port "p1"
            tag: 100
            Interface "p1"
    ovs_version: "2.7.0"

```

br-1 is not a fake bridge. This bridge does not have 1 single VLAN ID assigned to it, thus:

```

# ovs-vsctl br-to-vlan br-1
0

```

6.2 Ping Verification

Verify name spaces are isolated with pings

```

# ip netns exec ns1 ping -c2 10.0.0.30
ip netns exec ns1 ping -c2 10.0.0.30
PING 10.0.0.30 (10.0.0.30) 56(84) bytes of data.
64 bytes from 10.0.0.30: icmp_seq=1 ttl=64 time=0.807 ms
64 bytes from 10.0.0.30: icmp_seq=2 ttl=64 time=0.032 ms

--- 10.0.0.30 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.032/0.419/0.807/0.388 ms

# ip netns exec ns2 ping -c2 10.0.0.40
PING 10.0.0.40 (10.0.0.40) 56(84) bytes of data.
64 bytes from 10.0.0.40: icmp_seq=1 ttl=64 time=0.585 ms
64 bytes from 10.0.0.40: icmp_seq=2 ttl=64 time=0.054 ms

```

```

--- 10.0.0.40 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.054/0.319/0.585/0.266 ms

```

The pings below will not work because ns2 is connected to ns4 which is at 10.0.0.40

```

# ip netns exec ns2 ping -c1 10.0.0.30
PING 10.0.0.30 (10.0.0.30) 56(84) bytes of data.

```

```

--- 10.0.0.30 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

```

```

# ip netns exec ns1 ping -c1 10.0.0.40
PING 10.0.0.40 (10.0.0.40) 56(84) bytes of data.

```

```

--- 10.0.0.40 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

```

6.3 Creating a Fake Bridge

OVS fake bridges look and act like a bridge, but are tied to a particular VLAN ID.

```

# ovs-vsctl add-br vlan100 br-1 100
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "p2"
            tag: 200
            Interface "p2"
        Port "vlan100"
            tag: 100
            Interface "vlan100"
            type: internal
        Port "p3"
            tag: 100
            Interface "p3"
        Port "p4"
            tag: 200
            Interface "p4"
        Port "br-1"
            Interface "br-1"
            type: internal
        Port "p1"
            tag: 100
            Interface "p1"
    ovs_version: "2.7.0"

```

Create a namespace and attach it to the fake bridge

```

# ip netns add ns5
# ip link add tap100 type veth peer name p100
# ip link set tap100 netns ns5
# ip netns exec ns5 ip link set dev tap100 up
# ip link set dev p100 up
# ip netns exec ns5 ip addr add 10.0.0.100/24 dev tap100

```

Add the new ns5 to fake bridge vlan100

```
# ovs-vsctl add-port vlan100 p100
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "p100"
            tag: 100
            Interface "p100"
        Port "p2"
            tag: 200
            Interface "p2"
        Port "vlan100"
            tag: 100
            Interface "vlan100"
                type: internal
        Port "p3"
            tag: 100
            Interface "p3"
        Port "p4"
            tag: 200
            Interface "p4"
        Port "br-1"
            Interface "br-1"
                type: internal
        Port "p1"
            tag: 100
            Interface "p1"
    ovs_version: "2.7.0"
```

The name space's port p100 was automatically assigned tag 100 because it is a port on the fake bridge vlan100.

6.3.1 Verifying Fake Bridge

```
# ovs-vsctl br-to-vlan vlan100
100
# ovs-vsctl br-to-parent vlan100
br-1
```

```
# ip netns exec ns5 ping -c2 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=64 time=0.278 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=64 time=0.021 ms

--- 10.0.0.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.021/0.149/0.278/0.129 ms

# ip netns exec ns5 ping -c2 10.0.0.30
PING 10.0.0.30 (10.0.0.30) 56(84) bytes of data.
64 bytes from 10.0.0.30: icmp_seq=1 ttl=64 time=0.566 ms
64 bytes from 10.0.0.30: icmp_seq=2 ttl=64 time=0.039 ms

--- 10.0.0.30 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.039/0.302/0.566/0.264 ms
```

Likewise, ns5 does not have a valid VLAN ID to ping ns2 and ns4.

```
# ip netns exec ns5 ping -c1 10.0.0.20
```



```

PING 10.0.0.20 (10.0.0.20) 56(84) bytes of data.

--- 10.0.0.20 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

# ip netns exec ns5 ping -c1 10.0.0.40
PING 10.0.0.40 (10.0.0.40) 56(84) bytes of data.

--- 10.0.0.40 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

```

6.4 Clean Up

```

#!/bin/bash
#remove bridge, namespace, and bridge interfaces
BRIDGE=1
NS=$(ip netns list | wc -l)

#delete all net name spaces
ip -all netns del

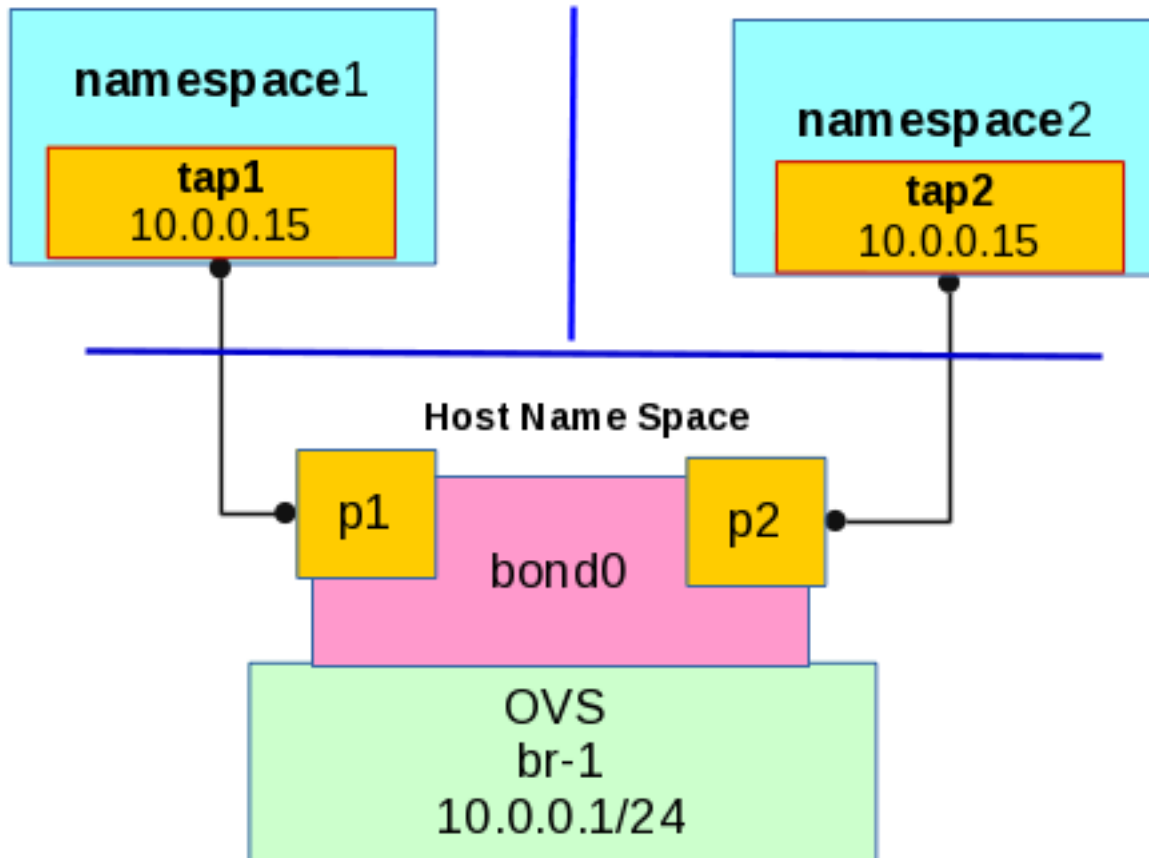
#delete all bridges
for i in `seq 1 $BRIDGE`;
do
    ip link set br-$i down
    ovs-vsctl del-br br-$i
done

```

7 Bonding - Active Backup

Bonding allows you to aggregate multiple ports into a single group, effectively combining the bandwidth into a single connection. Bonding is the same as port trunking. **active-backup** mode in OvS allows us to configure 2 interfaces such that if one device fails, the other device will then be used in its place. For this test we will configure 2 name spaces with the same IP address. We will bring one name space down but the IP address will still be routable.

Figure 4: Bond device bond0 with ports p1 and p2 connected



7.1 Setup

- 1 Bridge
- 2 Name spaces
- netperf

For netcat install instructions refer to Appendix E

Create a bridge

```
#!/bin/bash

# Create OVS switch
ovs-vsctl add-br br-1

#assign IP address
ip addr add 10.0.0.1/24 dev br-1

#bring the bridge interface up
ip link set br-1 up
```

Create 2 name spaces with the **same IP address**.

```
# ip netns add ns1
# ip link add tap1 type veth peer name p1
# ip link set tap1 netns ns1
# ip netns exec ns1 ip link set dev tap1 up
# ip link set dev p1 up
# ip netns exec ns1 ip addr add 10.0.0.15/24 dev tap1
```

```
# ip netns add ns2
# ip link add tap2 type veth peer name p2
# ip link set tap2 netns ns2
# ip netns exec ns2 ip link set dev tap2 up
# ip link set dev p2 up
# ip netns exec ns2 ip addr add 10.0.0.15/24 dev tap2
```

Create the bond interface named bond0 with interfaces p1 and p2.

```
# ovs-vsctl add-bond br-1 bond0 p1 p2 bond_mode=active-backup
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "bond0"
            Interface "p1"
            Interface "p2"
        Port "br-1"
            Interface "br-1"
                type: internal
    ovs_version: "2.7.0"
```

7.1.1 Verifying Bond Interface

View the bond information. Determine which interface is active. Both interfaces are considered slaves.

```
# ovs-appctl bond/show bond0
---- bond0 ----
bond_mode: active-backup
bond may use recirculation: no, Recirc-ID : -1
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
lacp_status: off
active slave mac: 02:6c:ee:36:41:8f(p1)

slave p1: enabled
    active slave
    may_enable: true

slave p2: enabled
    may_enable: true

# ovs-ofctl show br-1
OFPPT_FEATURES_REPLY (xid=0x2): dpid:000036cd4e136441
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst
1(p1): addr:02:6c:ee:36:41:8f
    config: 0
    state: 0
    current: 10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
2(p2): addr:aa:f0:3d:f2:a5:15
    config: 0
    state: 0
    current: 10GB-FD COPPER
    speed: 10000 Mbps now, 0 Mbps max
LOCAL(br-1): addr:36:cd:4e:13:64:41
```

```

config:      0
state:       0
speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0

```

This output confirms that interface p1 is the active slave - meaning we can send traffic to p1.
 Lets send traffic to p1 then bring the interface down such that traffic will then be routed to p2.

7.2 Ping Verification

From the root namespace, make sure the bridge and bond is configured correctly by pinging 10.0.0.15 from the host.

```

# ping -c2 10.0.0.15
PING 10.0.0.15 (10.0.0.15) 56(84) bytes of data.
64 bytes from 10.0.0.15: icmp_seq=1 ttl=64 time=0.277 ms
64 bytes from 10.0.0.15: icmp_seq=2 ttl=64 time=0.014 ms

--- 10.0.0.15 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.014/0.145/0.277/0.132 ms

```

7.3 Netcat Tests

From the output above we know that p1 is active. Start the listening nc server on port 8080 in ns1 and echo a message to the connection.

```
# ip netns exec ns1 nc -l 8080 -c "echo hello from ns1"
```

The server is now listening. From the root namespace , run the following command:

```
# nc 10.0.0.15 8080
hello from ns1
```

We were able to connect to ns1. Hit CTRL+C to kill the client.
 Bring ns1 tap1 interface down and kill netserver.

```

# ip netns exec ns1 ip link set tap1 down
# ip netns exec ns1 ip add
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
40: tap1@if39: <BROADCAST,MULTICAST> mtu 1500 qdisc noqueue state DOWN qlen 1000
   link/ether 1a:7a:bc:2f:c2:5c brd ff:ff:ff:ff:ff:ff link-netnsid 0
   inet 10.0.0.15/24 brd 10.0.0.255 scope global tap1
       valid_lft forever preferred_lft forever

# ovs-ofctl show br-1
OFPT_FEATURES_REPLY (xid=0x2): dpid:000036cd4e136441
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst
1(p1): addr:02:6c:ee:36:41:8f
   config:      0
   state:       LINK_DOWN
   current:     10GB-FD COPPER
   speed: 10000 Mbps now, 0 Mbps max
2(p2): addr:aa:f0:3d:f2:a5:15
   config:      0
   state:       0
   current:     10GB-FD COPPER
   speed: 10000 Mbps now, 0 Mbps max

```

```

LOCAL(br-1):  addr:36:cd:4e:13:64:41
    config:      0
    state:      0
    speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0

```

Start nc in ns2 and run same echo something different

```
# ip netns exec ns2 nc -l 8080 -c "echo hello from ns2"
```

Connect to ns2 from root namespace with *the same* nc command:

```

# nc 10.0.0.15 8080
hello from ns2
^C
#

```

Hit CTRL+C to kill the client.

7.4 Clean Up

```

#!/bin/bash
#remove bridge, namespace, and bridge interfaces
BRIDGE=1
NS=$(ip netns list | wc -l)

#delete all net name spaces
ip -all netns del

#delete all bridges
for i in `seq 1 $BRIDGE`;
do
    ip link set br-$i down
    ovs-vsctl del-br br-$i
done

```

8 SPAN Port

SPAN (Switched Port Analyzer) or also known as Port Mirroring, is a method of monitoring network traffic. OvS will send a copy of all network packets seen on one port (or an entire VLAN) to another port, where the packet can be analyzed.

8.1 Setup

- 1 Bridge
- 2 Name spaces
- tcpdump
- 2 terminal windows

For tcpdump install instructions refer to Appendix F

Create a bridge

```

#!/bin/bash

# Create OVS switch
ovs-vsctl add-br br-1

#assign IP address
ip addr add 10.0.0.1/24 dev br-1

```

```
#bring the bridge interface up
ip link set br-1 up
```

Create 2 name spaces with IP addresses

```
#!/bin/bash
```

```
#Creates name spaces with ports attached to OVS.
```

```
#Assigns an IP address to name space devices
```

```
BR=1
```

```
NUMNS=2
```

```
IP=10.0.0.0
```

```
for i in `seq 1 $NUMNS`;
do
```

```
    #create name spaces
```

```
    ip netns add ns$i
```

```
    # create a port pair. the tap$i exists in the ns, br-p$i on host
```

```
    ip link add tap$i type veth peer name p$i
```

```
    #attach namespace port to OVS
```

```
    ovs-vsctl add-port br-$BR p$i
```

```
    #attach namespace dev tap$i to ns1 such that tap$i exists only in ns1
```

```
    ip link set tap$i netns ns$i
```

```
    #turn on port on OVS. exists on host network
```

```
    ip link set dev p$i up
```

```
    #add ten to last octet of network ip addr
```

```
    IP=$(echo $IP | awk -F. '{printf "%d.%d.%d.%d", $1,$2,$3,$4+10}')
```

```
    #assign ip addresses.
```

```
    ip netns exec ns$i ip add add $IP/24 dev tap$i
```

```
    #bring the interface in ns1 up
```

```
    ip netns exec ns$i ip link set dev tap$i up
```

```
done
```

```
# ovs-vsctl add-port br-1 p1
```

```
# ovs-vsctl show
```

```
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
```

```
    Bridge "br-1"
```

```
        Port "br-1"
```

```
            Interface "br-1"
```

```
                type: internal
```

```
        Port "p2"
```

```
            Interface "p2"
```

```
        Port "p1"
```

```
            Interface "p1"
```

```
    ovs_version: "2.7.0"
```

Mirror p2 traffic to p1

```
# ovs-vsctl add-port br-1 p2 -- --id=@p get port p2 \
```

```
-- --id=@m create mirror name=m0 select-all=true output-port=@p \
-- set bridge br-1 mirrors=@m
0dde3367-0192-4068-ab47-ce6b77b442a9

# ovs-vsctl list Bridge br-1 | grep mirror
mirrors          : [0dde3367-0192-4068-ab47-ce6b77b442a9]
```

This mirror is called m0 and routes all traffic for p1 to the output-port which is p2.

8.2 Verifying Traffic Mirror

Because we created a SPAN port on p2, any traffic arriving on p2 is dropped. However, we can still use tcpdump to see the mirrored traffic.

You will need 2 terminal windows for this test.

Terminal 1:

```
tcpdump -i p2
# tcpdump: WARNING: p2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on p2, link-type EN10MB (Ethernet), capture size 65535 bytes1
```

Terminal 2:

```
# ip netns exec ns1 ping -c3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.627 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.627/0.627/0.627/0.000 ms
```

Terminal 1 output should now be:

```
# tcpdump -i p2
tcpdump: WARNING: p2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on p2, link-type EN10MB (Ethernet), capture size 65535 bytes
14:50:14.244389 IP 10.0.0.10 > myhostname.pok.stglabs.ibm.com: ICMP echo request,
id 4084, seq 1, length 64
14:50:14.244470 IP myhostname.pok.stglabs.ibm.com > 10.0.0.10: ICMP echo reply,
id 4084, seq 1, length 64
```

Even though traffic is only traveling from ns1 to br-1 we can use tcpdump and still see the traffic mirrored to p2.

8.3 Clean Up

Remove the mirror port

```
# ovs-vsctl clear bridge br-1 mirrors

#!/bin/bash
#remove bridge, namespace, and bridge interfaces
BRIDGE=1
NS=$(ip netns list | wc -l)

#delete all net name spaces
ip -all netns del

#delete all bridges
for i in `seq 1 $BRIDGE`;
do
```

```
ip link set br-$i down
ovs-vsctl del-br br-$i
done
```

9 QoS

OvS supports policing trafficking that ingresses (enters) into a switch. Policing is a simple form of quality-of-service (QoS) that simply drops packets received in excess of the configured rate. For this specific test, we change the policy algorithm to only allow a throughput less than 1Mbps for one specific interface.

9.1 Setup

- 1 Bridge
- 2 Name spaces
- netperf

For netperf install instructions refer to Appendix D
Create a bridge

```
# ovs-vsctl add-br br-1
#
```

Create 2 name spaces with IP addresses

```
#!/bin/bash

#Creates name spaces with ports attached to OVS.
#Assigns an IP address to name space devices

BR=1
NUMNS=2
IP=10.0.0.0

for i in `seq 1 $NUMNS`;
do
    #create name spaces
    ip netns add ns$i

    # create a port pair. the tap$i exists in the ns, br-p$i on host
    ip link add tap$i type veth peer name p$i

    #attach namespace port to OVS
    ovs-vsctl add-port br-$BR p$i

    #attach namespace dev tap$i to ns1 such that tap$i exists only in ns1
    ip link set tap$i netns ns$i

    #turn on port on OVS. exists on host network
    ip link set dev p$i up

    #add ten to last octet of network ip addr
    IP=$(echo $IP | awk -F. '{printf "%d.%d.%d.%d", $1,$2,$3,$4+10}')

    #assign ip addresses.
    ip netns exec ns$i ip add add $IP/24 dev tap$i
```



```
#bring the interface in ns1 up
ip netns exec ns$i ip link set dev tap$i up

done
```

```
# ovs-vsctl add-port br-1 p1
# ovs-vsctl add-port br-1 p2
```

```
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "p1"
            Interface "p1"
        Port "br-1"
            Interface "br-1"
                type: internal
        Port "p2"
            Interface "p2"
    ovs_version: "2.7.0"
```

```
# ip netns list
ns2 (id: 3)
ns1 (id: 0)
```

```
# ovs-vsctl list interface p1 | grep ingress
ingress_policing_burst: 0
ingress_policing_rate: 0
```

Start a listening netserver in ns2

```
# ip netns exec ns2 netserver
#
```

Send some traffic to listening netserver in ns2

```
# ip netns exec ns1 netperf -H 10.0.0.20 -P0 -v0 -l5
13484.52
#
```

9.2 Verifying QoS

Change ingress policy for interface p1 to accept only 1Mb of traffic limited by OvS.

```
# ovs-vsctl set interface p1 ingress_policing_rate=1000
# ovs-vsctl set interface p1 ingress_policing_burst=100

# ovs-vsctl list interface p1 | grep ingress
ingress_policing_burst: 100
ingress_policing_rate: 1000
```

Run the same netperf test. The netserver is still listening.

```
# ip netns exec ns1 netperf -H 10.0.0.20 -P0 -v0 -l5
0.51
#
```

Your results may be different and thats ok. The through put number should be less than your first results and should be less than 1.

9.3 Clean Up

```
#!/bin/bash
#remove bridge, namespace, and bridge interfaces
BRIDGE=1
NS=$(ip netns list | wc -l)

#delete all net name spaces
ip -all netns del

#delete all bridges
for i in `seq 1 $BRIDGE`;
do
    ip link set br-$i down
    ovs-vsctl del-br br-$i
done
```

Make sure the netserver has been killed.

```
# ps -aef | grep netserver
root      8234      1  0 12:38 ?        00:00:00 netserver
root      8307    6360  0 12:45 pts/0    00:00:00 grep --color=auto netserver
# kill 8234
```

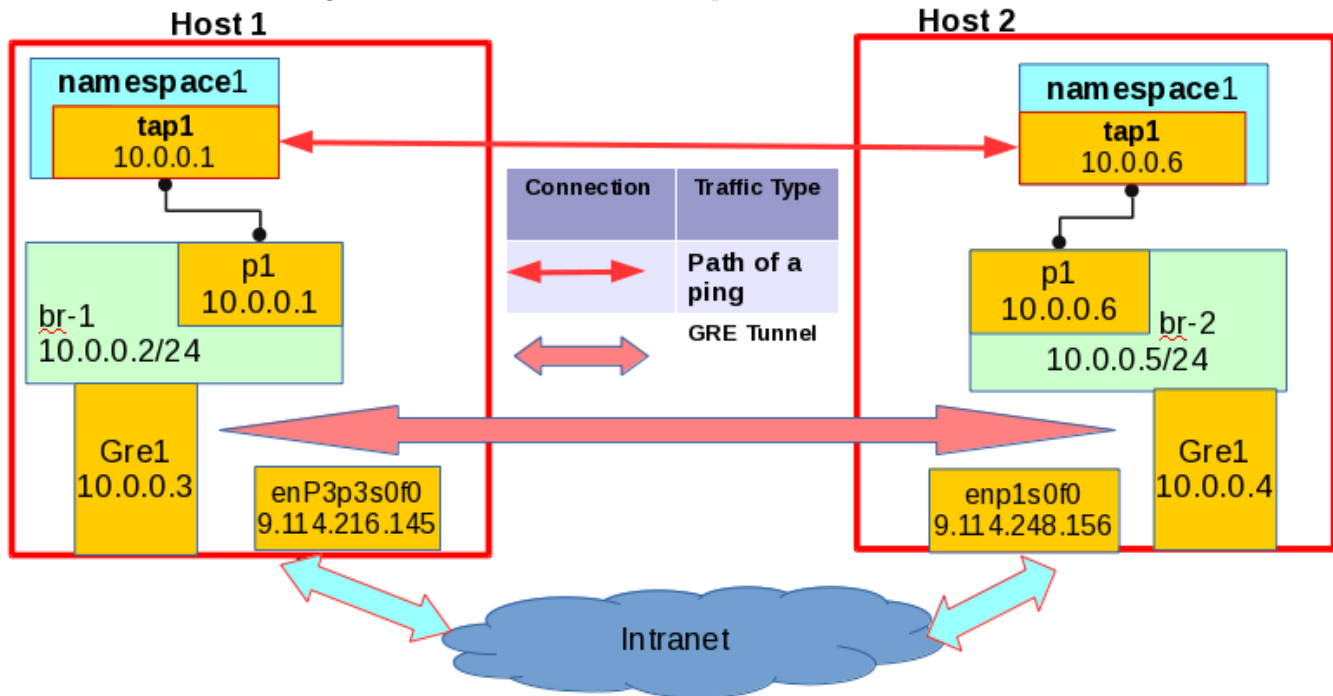
10 Multi-Host Tests

Tests outlined below will utilize 2 different kernel instances. I will be using 2 different VMs but you may also use 2 physical machines as well.

This test plan is for Open vSwitch and not a network setup tutorial. You will be responsible for making sure your 2 machines (virtual or not) are already configured properly and can ping each other **before** we add Open vSwitch to the setup.

I recommend using VMs to consume less resources. This test plan was written using VMs and there is more information about OvS and libvirt in appendix.

Figure 5: GRE tunnel between 2 separate hosts, Host 1 and Host 2



10.1 GRE Tunnel

A Generic Routing Encapsulation tunnel is a method to connect 2 separate networks on different physical hosts. This is accomplished by encapsulating packets into a GRE header. I will be setting up a GRE tunnel between 2 VMs running on top of libvirt and QEMU.

10.2 Setup

- 2 Different hosts (2 VMs or physical hosts)
- 2 OvS Bridges
- 1 network namespace in each host

More information about GRE tunnels in Appendix G.1

10.2.1 Host 1 Setup

Create a gre interface pointing to Host 2's network interface. We additionally assign an IP address for the intra-net. First, create a gre interface

```
# ip tunnel add gre1 mode gre local 9.114.216.145 remote 9.114.248.156 ttl 255
# ip netns exec ns1 ip add add 10.0.0.3/24 dev gre1
# ip netns exec ns1 link set gre1 up
```

Create an OvS bridge and verify IP addresses.

```
# ovs-vsctl add-br br-1
# ip add add 10.0.0.2/24 dev br-1
# ip link set br-1 up
# ovs-vsctl show
2cfb3863-56ed-4ea5-89ee-607a64a72b13
    Bridge "br-1"
        Port "br-1"
            Interface "br-1"
                type: internal

# ip add
gre1@NONE: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1476 state UNKNOWN qlen 1
    link/gre 9.114.216.145 peer 9.114.248.156
    inet 10.0.0.3/24 scope global gre1
        valid_lft forever preferred_lft forever
    inet6 fe80::200:5efe:972:d891/64 scope link
        valid_lft forever preferred_lft forever
br-1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 state UNKNOWN qlen 1000
    link/ether e6:df:5d:03:af:4e brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.2/24 scope global br-1
        valid_lft forever preferred_lft forever
    inet6 fe80::e4df:5dff:fe03:af4e/64 scope link
        valid_lft forever preferred_lft forever
```

Create ns1 with a veth pair p1 in root ns and tap1 in ns. Additionally assign an IP address to p1

```
# ip netns add ns1
# ip link add tap1 type veth peer name p1
# ip link set tap1 netns ns1
# ip link set dev p1 up
# ip netns exec ns1 ip add add 10.0.0.1/24 dev tap1
# ip netns exec ns1 ip link set dev tap1 up
# ip add add 10.0.0.1/24 dev p1
# ip netns exec ns1 ip add
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN qlen 1
    link/gre 0.0.0.0 brd 0.0.0.0
3: gretap0@NONE: <BROADCAST,MULTICAST> mtu 1462 qdisc noop state DOWN qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
37: tap1@if36: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 state UP qlen 1000
    link/ether a6:48:98:46:52:ce brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.1/24 scope global tap1
        valid_lft forever preferred_lft forever
    inet6 fe80::a448:98ff:fe46:52ce/64 scope link
        valid_lft forever preferred_lft forever

# ip add
p1@if37:<BROADCAST,MULTICAST,UP,LOWER_UP>mtu 1500 master state UP qlen 1000
    link/ether de:ad:cc:fb:a6:20 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.1/24 scope global p1
        valid_lft forever preferred_lft forever
    inet6 fe80::dcad:ccff:fefb:a620/64 scope link
        valid_lft forever preferred_lft forever
```

Attach p1 and gre1 to bridge.

```
# ovs-vsctl add-port br-1 p1
# ovs-vsctl add-port br-1 gre1 \
```

```
-- set Interface gre1 type=gre options:remote_ip=9.114.248.156

# ovs-vsctl show
2cfb3863-56ed-4ea5-89ee-607a64a72b13
Bridge "br-1"
  Port "p1"
    Interface "p1"
  Port "gre1"
    Interface "gre1"
    type: gre
    options: {remote_ip="-- set Interface gre1 type=gre options:remote_ip=9.114.248.156"}
  Port "br-1"
    Interface "br-1"
    type: internal
```

Make sure all interfaces have been set to UP with `ip link set` command.

10.2.2 Host 2 Setup

Create a gre interface pointing to Host 1's network interface. We additionally assign an IP address for the intra-net. First, create a gre interface

```
# ip tunnel add gre1 mode gre remote 9.114.216.145 local 9.114.248.156 ttl 255
# ip add add 10.0.0.4/24 dev gre1
# ip link set gre1 up
```

Create an OvS bridge and verify IP addresses.

```
# ovs-vsctl add-br br-1
# ip add add 10.0.0.5/24 dev br-1
# ip link set br-1 up
# ovs-vsctl show
2cfb3863-56ed-4ea5-89ee-607a64a72b13
Bridge "br-1"
  Port "br-1"
    Interface "br-1"
    type: internal

# ip add
gre1@NONE: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1476 state UNKNOWN qlen 1
  link/gre 9.114.248.156 peer 9.114.216.145
  inet 10.0.0.4/24 scope global gre1
    valid_lft forever preferred_lft forever
  inet6 fe80::200:5efe:972:d891/64 scope link
    valid_lft forever preferred_lft forever
br-1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 state UNKNOWN qlen 1000
  link/ether e6:df:5d:03:af:4e brd ff:ff:ff:ff:ff:ff
  inet 10.0.0.5/24 scope global br-1
    valid_lft forever preferred_lft forever
  inet6 fe80::e4df:5dff:fe03:af4e/64 scope link
    valid_lft forever preferred_lft forever
```

Create ns1 with a veth pair p1 in root ns and tap1 in ns. Additionally assign an IP address to p1

```
# ip netns add ns1
# ip link add tap1 type veth peer name p1
# ip link set tap1 netns ns1
# ip link set dev p1 up
# ip netns exec ns1 ip add add 10.0.0.6/24 dev tap1
```

```
# ip netns exec ns1 ip link set dev tap1 up
# ip add add 10.0.0.6/24 dev p1
# ip netns exec ns1 ip add
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN qlen 1
   link/gre 0.0.0.0 brd 0.0.0.0
3: gretap0@NONE: <BROADCAST,MULTICAST> mtu 1462 qdisc noop state DOWN qlen 1000
   link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
37: tap1@if36: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 state UP qlen 1000
   link/ether a6:48:98:46:52:ce brd ff:ff:ff:ff:ff:ff link-netnsid 0
   inet 10.0.0.6/24 scope global tap1
       valid_lft forever preferred_lft forever
   inet6 fe80::a448:98ff:fe46:52ce/64 scope link
       valid_lft forever preferred_lft forever

# ip add
p1@if37:<BROADCAST,MULTICAST,UP,LOWER_UP>mtu 1500 master state UP qlen 1000
   link/ether de:ad:cc:fb:a6:20 brd ff:ff:ff:ff:ff:ff link-netnsid 0
   inet 10.0.0.6/24 scope global p1
       valid_lft forever preferred_lft forever
   inet6 fe80::dcad:ccff:fefb:a620/64 scope link
       valid_lft forever preferred_lft forever
```

Attach p1 and gre1 to bridge.

```
# ovs-vsctl add-port br-1 p1
# ovs-vsctl add-port br-1 gre1 \
-- set Interface gre1 type=gre options:remote_ip=9.114.216.145

# ovs-vsctl show
2cfb3863-56ed-4ea5-89ee-607a64a72b13
Bridge "br-1"
  Port "p1"
    Interface "p1"
  Port "gre1"
    Interface "gre1"
    type: gre
    options: {remote_ip="9.114.216.145"}
  Port "br-1"
    Interface "br-1"
    type: internal
```

Make sure all interfaces have been set to UP with `ip link set` command.

10.3 Ping Verification

On Host 2 ping ns1 on Host 1

```
# ip netns exec ns1 ping -c2 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.800 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.295 ms

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.295/0.547/0.800/0.253 ms
```

If your pings are not working, first try flushing iptables on *both* hosts. Ensure all interfaces are up on *both* hosts.

```
# iptables -F
# ip link set gre1 up
# ip link set br-1 up
# ip netns exec ns1 ip link set tap1 up
```

Likewise, on Host 1 ping ns1 on Host 2

```
# ip netns exec ns1 ping -c2 10.0.0.6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=0.851 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=0.293 ms

--- 10.0.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.295/0.547/0.800/0.253 ms
```

10.4 Clean Up

On Host 1

```
# rmmod ip_gre
# ovs-vsctl del-br br-1
# ip -all netns del
```

On Host 2

```
# rmmod ip_gre
# ovs-vsctl del-br br-2
# ip -all netns del
```

11 TODO: VXLAN

VXLAN stands for Virtual eXtensible Local Area Network. Read RFC 7348 for more info.

Appendices

A Troubleshooting Installation

A.1 OvS Kernel Module

If kernel module is not loaded, use modprobe command:

```
# modprobe openvswitch
```

Verify that the kernel object exists on your system for your currently booted kernel

```
# ls /lib/modules/$(uname -r)/kernel/net/openvswitch
```

A.2 Starting OvS Service

If you use `yum install` for RHEL or `apt` for Ubuntu, OvS should be started automatically. If not follow these commands:

A.2.1 RHEL

Check the status of the OvS service and confirm it is not running:

```
# systemctl status openvswitch
openvswitch.service - LSB: Open vSwitch switch
  Loaded: loaded (/etc/rc.d/init.d/openvswitch; bad; vendor preset: disabled)
  Active: inactive (dead) since Wed 2017-03-22 14:42:55 PDT; 29s ago
  Docs: man:systemd-sysv-generator(8)
```

Output verifies that it is inactive (dead)

Then start the service and verify there are new processes started:

```
# systemctl start openvswitch
# systemctl status openvswitch
openvswitch.service - LSB: Open vSwitch switch
  Loaded: loaded (/etc/rc.d/init.d/openvswitch; bad; vendor preset: disabled)
  Active: active (running) since Wed 2017-03-22 14:45:00 PDT; 4 days ago
  Docs: man:systemd-sysv-generator(8)
  Process: 25684 ExecStop=/etc/rc.d/init.d/openvswitch stop
           (code=exited, status=0/SUCCESS)
  Process: 25725 ExecStart=/etc/rc.d/init.d/openvswitch start
           (code=exited, status=0/SUCCESS)
  CGroup: /system.slice/openvswitch.service
           25748 ovsdb-server: monitoring pid 25749 (healthy)
           25749 ovsdb-server /etc/openvswitch/conf.db -vconsole:emer
           -vsyslog:err -vfile:info --remote=punix:/...
           25759 ovs-vswitchd: monitoring pid 25760 (healthy)
           25760 ovs-vswitchd unix:/var/run/openvswitch/db.sock -vconsole:
           -vsyslog:err -vfile:info --mlock...
```

A.2.2 Ubuntu

Check the status of the OvS service:

```
# systemctl status openvswitch-switch
openvswitch-switch.service - Open vSwitch
  Loaded: loaded (/lib/systemd/system/openvswitch-switch.service;
           enabled; vendor preset: enabled)
  Active: inactive (dead) since Wed 2017-03-22 18:07:10 EDT; 22s ago
  Main PID: 5079 (code=exited, status=0/SUCCESS)
```


Then start the service and verify there are new processes started:

```
# systemctl start openvswitch-switch
#

# ps -aef | grep ovs
root      6528   6482   grep --color=auto ovs
root      25748      1  ovsdb-server: monitoring pid 25749 (healthy)
root      25749  25748  ovsdb-server /etc/openvswitch/conf.db
                -vsyslog:err -vfile:info
                --remote=punix:/var/run/openvswitch/db.sock
                --private-key=db:Open_vSwitch,SSL,private_key
                --certificate=db:Open_vSwitch,SSL,certificate
                --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert --no-chdir
                --log-file=/var/log/openvswitch/ovsdb-server.log
                --pidfile=/var/run/openvswitch/ovsdb-server.pid --detach
root      25759      1  ovs-vswitchd: monitoring pid 25760 (healthy)
root      25760  25759  ovs-vswitchd unix:/var/run/openvswitch/db.sock
                -vconsole:emer -vsyslog:err -vfile:info --mlockall --no-chdir
                --log-file=/var/log/openvswitch/ovs-vswitchd.log
                --pidfile=/var/run/openvswitch/ovs-vswitchd.pid --detach
```

B Name Spaces

A network name space can be thought of like a container (LXC or Docker) or very tiny virtual machine (VM). Network name spaces have the same idea as a container or VM such that resources are isolated, but a network name space only isolates the network interface. Each network namespace has its own interfaces, routing tables, and forwarding tables. Think of a network name space as a virtual machine with only network functionality. For our specific usage, our name spaces will have an IP address and will be connected to our bridge to run tests.

It is also important to note that processes can be dedicated to a network name space.

Name spaces use virtual Ethernet devices, called veth. The veth device works like a tunnel. This tunnel can be thought of like a Ethernet cable which we will use to connect our name spaces to our bridge. Veth works with peers, one peer is in the name space and the other peer is a port on OvS.

B.1 Creating Name Spaces

To create a name space named `ns1` type:

```
# ip netns add ns1
```

To view your name spaces type:

```
# ip netns list
ns1 (id: 0)
```

B.2 Deleting Name Space

```
# ip netns delete ns1
#
```

B.3 Creating Veth Interfaces

To create a veth device type the following:

```
# ip link add tap1 type veth peer name p1
```

You will now have 2 new network interfaces on your machine:

```
ip addr | grep tap
p1@tap1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN qlen 1000
tap1@p1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN qlen 1000
```

B.4 Attaching Veth Device to Name Space

Attach namespace dev `tap1` to `ns1` such that `tap1` exists only in `ns1`

```
# ip link set tap1 netns ns1
```

Interface `tap1` now exists only in the name space called `ns1`

```
# ip add | grep tap
#
```

Remember that a name space is isolated similar to how a VM would be. So the `tap1` interface no longer exists in the root's name space, it is in `ns1` now.

B.5 Executing Commands in a Name Space

To view network interfaces inside `ns1`, execute the `ip address` command inside the name space like so:

```
# ip netns exec ns1 ip addr
# ip netns exec ns1 ip link set dev tap1 up
# ip netns exec ns1 ip addr
```

Now we have a name space with a network interface. The interface in the name space (`tap1`) is connected to the interface named `p1` on the host network.

C Setting Up Name Spaces for Test Environment

To create a name space named `ns1`:

```
# ip netns add ns1
# ip netns list
ns1
```

Create the veth interfaces

```
# ip link add tap1 type veth peer name p1

# ip addr | grep tap
p1@tap1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN qlen 1000
tap1@p1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN qlen 1000
```

Move the `tap1` interface into `ns1`.

```
# ip netns exec ns1 ip addr
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
# ip link set tap1 netns ns1
# ip netns exec ns1 ip addr
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
381: tap1@if380: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether ba:15:20:dc:63:ec brd ff:ff:ff:ff:ff:ff link-netnsid 0
# ip addr | grep tap
#
```

Set the `tap1` interface to UP state inside `ns1` by executing the `ip addr`. This command is executed inside the name space.

```
# ip netns exec ns1 ip link set dev tap1 up
```

Likewise, we can run the same command on the root namespace and turn on the `p1` interface

```
# ip link set dev p1 up
```

Now we have an interface that is up and a virtual environment to execute commands in.

```
# ip addr | grep p1
p1@if381: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    qlen 1000
    link/ether 72:9b:17:a0:cb:13 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::709b:17ff:fea0:cb13/64 scope link
        valid_lft forever preferred_lft forever
# ip netns exec ns1 ip addr
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
tap1@if380: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    qlen 1000
    link/ether ba:15:20:dc:63:ec brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::b815:20ff:fedc:63ec/64 scope link
        valid_lft forever preferred_lft forever
```

C.1 Troubleshooting: Adding Port to Bridge

First make sure you have a bridge:

```
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "br-1"
            Interface "br-1"
                type: internal
    ovs_version: "2.7.0"
```

Verify the flows for br-1

```
# ovs-ofctl show br-1
OFPST_FEATURES_REPLY (xid=0x2): dpid:0000c24e5f3e2f4b
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
LOCAL(br-1): addr:c2:4e:5f:3e:2f:4b
    config:      PORT_DOWN
    state:      LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
```

```
# ovs-ofctl dump-flows br-1
NXST_FLOW reply (xid=0x4):
    cookie=0x0, duration=233.241s, table=0, n_packets=0, n_bytes=0, idle_age=233,
    priority=0 actions=NORMAL
```

Now let's add some ports to our bridge

```
# ovs-vsctl add-port br-1 p1
ovs-vsctl: Error detected while setting up 'p1':
could not open network device p1 (No such device).
See ovs-vswitchd log for details.
ovs-vsctl: The default log directory is "/var/log/openvswitch".
```

We added interface p1 to bridge br-1 however p1 interface has not been created on our machine yet.

Unlike a new bridge interface, OvS will not create the network interface for a port. However, OvS will still create the port on the bridge.

```
# ip addr | grep p1
#
```

Take a look at your bridge and flows:

```
# ovs-vsctl show
d35f9ced-06b7-4651-8d1c-f8dd75ff3fc2
    Bridge "br-1"
        Port "br-1"
            Interface "br-1"
                type: internal
        Port "p1"
            Interface "p1"
                error: "could not open network device p1 (No such device)"
    ovs_version: "2.7.0"
```

The output above shows a similar error after we added the port to the bridge, no such device.

```
# ovs-ofctl show br-1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000c24e5f3e2f4b
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src
mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
    LOCAL(br-1): addr:c2:4e:5f:3e:2f:4b
        config:      PORT_DOWN
        state:       LINK_DOWN
        speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

We cannot create a flow for an interface that does not exist. The output above verifies no flows have been added for interface p1. Output from `ovs-ofctl dump-flows br-1` will remain the same as above.

D Installing Netperf

```
# wget ftp://ftp.netperf.org/netperf/netperf-2.7.0.tar.gz
# tar -xzf netperf-2.7.0.tar.gz
# cd netperf-2.7.0
# ./configure
# make
# make install
```

D.1 Troubleshooting Installation

If build is not working, you may need to download 2 patches to determine your system arch.

```
# wget -O config.guess \
'http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.guess;hb=HEAD'

# wget -O config.sub \
'http://git.savannah.gnu.org/gitweb/?p=config.git;a=blob_plain;f=config.sub;hb=HEAD'
```

E Installing netcat

```
# yum search all netcat
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager
===== Matched: netcat =====
nmap.ppc64le : Network exploration tool and security scanner
nmap-ncat.ppc64le : Nmap's Netcat replacement
socat.ppc64le : Bidirectional data relay between two data channels ('netcat++')

# yum install nc
```

```

Resolving Dependencies
--> Running transaction check
---> Package nmap-ncat.ppc64le 2:6.40-7.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

Installing:
  nmap-ncat

Installed:
  nmap-ncat.ppc64le 2:6.40-7.el7

Complete!

```

F Installing tcpdump

```

# yum search tcpdump
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager
===== N/S matched: tcpdump =====
tcpdump.ppc64le : A network traffic monitoring tool

# yum install tcpdump

```

G Multiple Host Trouble Shooting

Before you even install Open vSwitch, make sure your 2 machines (physical or virtual) can communicate with each other.

It is recommended to first draw a simple diagram of what your network will look like once configured properly. It can be difficult to think of how your network will look given command outputs and different name spaces.

It is important to approach a network problem in pieces or chunks. If you cannot communicate with your VM start from the VM's point-of-view and start pinging smaller portions of the network. Additionally there may be certain IP Tables or rules on a external switch which may be preventing internet access. First verify the problem is NOT on the host machine, then you could explore the possibility of an additional rule or firewall somewhere else. Don't assume both sides of a connection are working in a network test the problem from all angles.

Use tools like ping and tcpdump to verify connections. If you are pinging Host B from Host A, start tcpdump listening on Host B to see if the ICMP packet even gets to Host B. This will narrow down the issue in the network.

G.1 GRE Tunnels

```

# lsmod | grep gre
vport_gre          1889  1
ip_gre             23515  1 vport_gre
ip_tunnel          31010  1 ip_gre
gre                4661  1 ip_gre
openvswitch        149665  4 vport_gre

```

To remove a GRE interface you will have to unload the ip_gre module.

```

# rmmod vport_gre
# rmmod ip_gre
# ip adr | grep gre
#

```

If vport_gre module is loaded, remove that as well.

```

# rmmod ip_gre
rmmod: ERROR: Module ip_gre is in use by: vport_gre
# rmmod vport_gre

```

```
# rmmod ip_gre
# ip add | grep gre
#
```