Hands-on Lab Description

CS-NET-00006 –
Open Virtual Switch (OVS) Installation and Basic Setup

Category:

```
CS-NET: Computer Networking
```

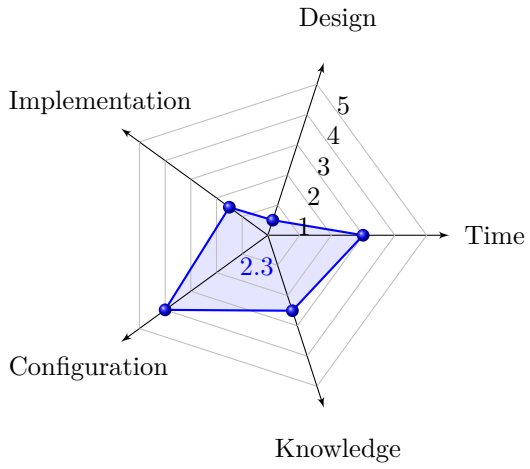
Objectives:

```
Install Open Virtual Switch (OVS)
Basic configuration of OVS on linux virtual machine
```

Estimated Lab Duration:

```
Expert: 40 minutes
Novice: 120 minutes
```

Difficulty Diagram:



Difficulty Table.

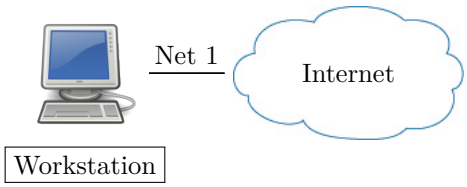
Measurements	Values (0-5)
Time	3
Design	0.5
Implementation	1.5
Configuration	4
Knowledge	2.5
Score (Average)	2.3

Required OS:

```
Linux: Ubuntu 18.04 LTS
```

Lab Running Environment:

```
  VirtualBox https://www.virtualbox.org/ (Reference Labs: CS-SYS-00101)
```




```
1 Workstations: Linux (Ubuntu 18.04 LTS)
2 Network Setup:
  Internet: Accessible
```

Lab Preparations:

```
1 Know how to use Linux OS (Reference Labs: CS-SYS-00001)
2 Basic knowledge about computer networking (Reference Labs: CS-NET-00001 and CS-NET
      -00002)
```

Task 1.0 Install and verify the dependencies for OVS

In this task, you will need to verify whether your system meet the prerequisites for Open Virtual Switch (OVS). If your VM had already installed OVS, please skip this task.

Task 1.1 OVS Installation Requirements

Before installing OVS properly on your machine, it is important to check whether your system have required and recommended software installed on your vm. To verify the specified software in Linux, please use the following command:

```
$    software_name --version
```

Note that Linux is case sensitive. The following software packages are required to build the OVS from its source codes.

1. GNU *make* is a build automation tool that automatically builds executable programs and libraries from source code by reading files called Makefiles which specify how to derive the target program.
2. *gcc* 4.6 or later. The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages.
3. *libssl* is the portion of OpenSSL which supports TLS (SSL and TLS Protocols), and depends on *libcrypto*. It is optional but recommended if you plan to connect the OVS to an OpenFlow controller.
4. *libcap-ng* is optional but recommended. It is required to run OVS daemons as a non-root user with dropped root privileges.
5. *python* 3.4 or later. Note that in Linux system, *python* command is usually running python 2.7 and *python3* command is running python 3.5.
6. *unbound* library, from `http://www.unbound.net`, is optional but recommended if you want to enable *ovs-vswitchd* and other utilities to use DNS names when specifying OpenFlow and OVSDB remotes.
7. *autoconf* version 2.63 or later. Autoconf is an extensible package of M4 macros that produce shell scripts to automatically configure software source code packages.
8. *automake* version 1.10 or later. Automake is a tool for automatically generating *Makefile.in* files compliant with the GNU Coding Standards.

The following software are not needed for compiling and building OVS, but nice to have in later stage of using OVS and performing network diagnostics.

1. *libtool* version 2.4 or later. GNU libtool is a generic library support script.
2. *pyftpdlib*, version 1.2.0 is known to work. Python FTP Server library is a FTP server library providing a portable interface to easily write very efficient asynchronous FTP servers with Python.
3. *wget*, version 1.16 is known to work. Earlier versions may also work.
4. *netcat*, several common implementations are known to work.
5. *curl*, version 7.47.0 is known to work. Earlier versions may also work.
6. *tftpy*, version 0.6.2 is known to work. Earlier versions may also work.
7. *netstat*, it is available from various distribution specific packages.

Task 1.2 Install Required Software

If all above mentioned software requirements are met, please skip the following installations. In Linux, the system require privilege escalation to perform installation. For each installation, using *sudo* before *apt-get* for privilege escalation, or issuing *sudo -i* to enter the super user/root privilege mode for all the installation.

1. Update installable components

```
$       apt-get update
```

2. Install related dependencies

```
$       apt-get install build-essential libssl-dev linux-headers-$(uname -r)
$       apt-get install graphviz autoconf automake bzip2 debhelper dh-autoreconf
          libssl-dev libtool openssl python-all python-qt4 python-twisted-conch
          python-zope.interface python-six dkms module-assistant ipsec-tools racoon
          libc6-dev module-init-tools netbase libpython2.7-stdlib uuid-runtime
```

3. Install *libssl* related components

```
$       apt-get install libssl-dev
$       apt-get install openssl
```

4. Install *libcap-ng*

```
$       apt-get install libcap-ng-utils
$       apt-get install libcap-ng-dev
```

5. Install *libtool* and *autoconf*

```
$       apt-get install libtool
$       apt-get install autoconf
```

6. Install *unbound*

```
$       apt-get update
$       apt-get install unbound
```

7. Install *tftpy*

```
$       apt-get update
$       apt-get install python-tftpy
```

8. Install *pyftpdlib*

```
$       apt-get update
$       apt-get install python3-pyftpdlib
```

Task 2.0 Install OVS

Next, let's choose one of the following two sub tasks to install OVS.

Task 2.1 Install OVS from Linux Distribution

Once the required software components are installed, we can simply use *apt-get* approach to install OVS:

```
$    apt install openvswitch-switch
```

Task 2.2 Install OVS from Source Codes

Alternatively, we can install OVS from its source codes.

1. Obtain OVS sources. The following command will clone OVS source codes in the local directory.

   ```
   $       git clone https://github.com/openvswitch/ovs.git
   ```

2. The following operations require root privilege, such as using *sudo -i* or *sudo root*. The next step is bootstrapping.

   ```
   $       cd ovs  %entering the ovs build folder
   $       ./boot.sh
   ```

3. Configure Linux kernel module by running the configure script.

   ```
   $       ./configure --with-linux=/lib/modules/$(uname -r)/build
   ```

4. Run GNU *make* in the build directory

   ```
   $       make
   ```

5. Run *make install* to install *executables* and *man* pages into the running system. By default, they will be put in the folder */usr/share*

   ```
   $       make install
   ```

6. Build the kernal modules. Note: Linux kernel is configured with *CONFIG_MODULE_SIG=y*. Thus, *make modules_install* is attempting to add a signature to each kernel module, but this is failing because the signing key certificate is not present. The signing key certificate is mostly often created as part of the base kernel build, and not included in distribution packages. During the installation, modules tried to add signatures and there are errors generated. We can ignore the errors since modules have been already installed.

   ```
   $       make modules_install
   ```

7. Load core modules and validation

   ```
   $       /sbin/modprobe openvswitch
   $       /sbin/lsmod | grep openvswitch
   ```

Task 3.0 Basic OVS Network Setup

In this task, we will need to set up an OVS and make all traffic targeting at external systems to pass through it, where the system setup is illustrated in Figure CS-NET-00006.1.

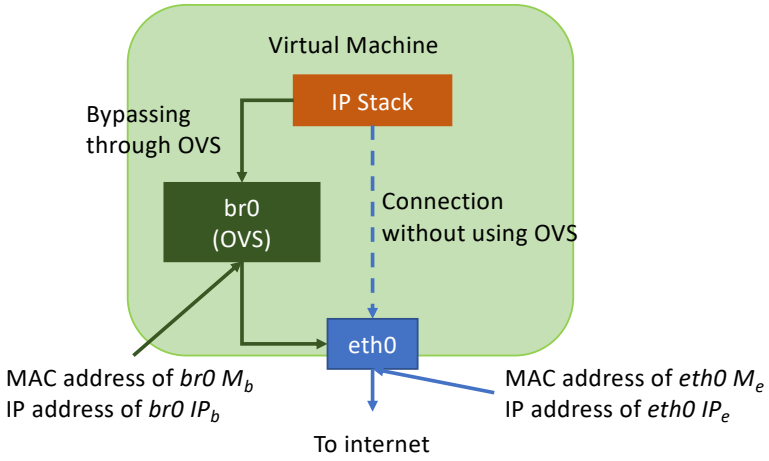


Figure CS-NET-00006.1
Network system setup.

The OVS setup procedure is given as follows:

1. Before starting OVS, we need to make some preparations.

```
$       mkdir -p /usr/local/etc/openvswitch % create a directory for OVS database
          and schema
$       ovsdb-tool create /usr/local/etc/openvswitch/conf.db
          vswitchd/vswitch.ovsschema % Initialize the configuration database using
          ovsdb-tool
```

2. (Option 1) Start the Open vSwitch suite of daemons is a simple process. Open vSwitch includes a shell script, and helpers, called *ovs-ctl* which automates much of the tasks for starting and stopping *ovsdb-server*, and *ovs-vswitchd*. The *ovs-ctl* utility is located in *$(pkgdatadir)/scripts*, and defaults to */usr/local/share/openvswitch/scripts*

```
$       export PATH=$PATH:/usr/local/share/openvswitch/scripts % set up path to
          rung scripts. Note: you can put export statement into the file .profile
          to make sure the path is set after restarting the VM.
$       ovs-ctl start  % start the ovs and DB server
$       ovs-ctl stop  % stop the ovs and DB server as needed
```

3. (Option 2) Additionally, the *ovs-ctl* script allows starting and stopping the daemons individually using specific options. To start just the *ovsdb-server*:

```
$       export PATH=$PATH:/usr/local/share/openvswitch/scripts
$       ovs-ctl --no-ovs-vswitchd start
```

4. (Option 3) Start just the *ovs-vswitchd*

```
$       export PATH=$PATH:/usr/local/share/openvswitch/scripts
$       ovs-ctl --no-ovsdb-server start
```

5. Start Open *vSwitch* by using automated script, you may wish to manually start the various daemons. Before starting *ovs-vswitchd* itself, you need to start its configuration database, *ovsdb-server*. Each machine on which Open vSwitch is installed should run its own copy of *ovsdb-server*. Configure ovsdb-server to use

database created above, to listen on a Unix domain socket, to connect to any managers specified in the database itself, and to use the SSL configuration in the database.

```
$       mkdir -p /usr/local/var/run/openvswitch
$       ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock
          --remote=db:Open_vSwitch,Open_vSwitch,manager_options
          --private-key=db:Open_vSwitch,SSL,private_key
          --certificate=db:Open_vSwitch,SSL,certificate
          --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert --pidfile --detach
          --log-file
```

6. Initialize the database by using *ovs-vsctl*

```
$       ovs-vsctl --no-wait init
```

7. Now, create bridges on OVS

```
$       ovs-vsctl add-br br0  % adding a new bridge into br0
$       ovs-vsctl add-br br1  % adding a new bridge into br1
$       ovs-vsctl show       % show our new virtual switch
```

Check bridge setup

```
$       ifconfig br0 up  %turn on the port br0, same for bridge br1
$       ifconfig           %show interface of br0
$       ovs-vsctl del-br br1 %delete a bridge as needed
```

8. Next, we need to connect the bridge to *eth0* (or other interfaces such as *enp0s3* on the testing VM). Otherwise, the bridges are isolated from external networks. To connect the bridge to external network:

```
$       ovs-vsctl add-br br0 % adding a new bridge into br0
$       ifconfig br0 up
$       ovs-vsctl add-port br0 enp0s3 %connecting br0 to the network interface to
          outside e.g., enp0s3, eth0, which may connect to the Internet
$       ovs-vsctl show
```

Now, let's ping external network to see if the connection is successfully established

```
$       ping 8.8.8.8  % ping google server to test the connectivity
```

However, it cannot make the connection. The problem is that we need to remove external interface (*enp0s3* or *eth0*) IP address and assign that IP to *br0* and reset the default gateway network to *br0*. In this way, the IP stack will use *br0* as the network to send traffic to external networks.

```
$       ifconfig enp0s3 0  % removing enp0s3 IP address
$       ifconfig br0 10.0.2.33 % assigning the a static IP address to br0
```

Before we continue to next step, we need to do one more configuration in ThoTh Lab to assign the MAC address of *enp0s3* to *br0*, i.e., $M_b \Leftarrow M_e$ as shown in Figure CS-NET-00006.1. This is because ThoTh Lab is built on openstack, where its security group policy inspects packet sent from *enp0s3* that has *br0* MAC address, and it will be considered as a misused packet, and thus Openstack network will drop that packet. To assign *br0* with the same MAC address of *enp0s3*, we can do follows:

```
$       ovs-vsctl set bridge br0 other-config:hwaddr= MAC of enp0s3 % this command
          is needed in ThoTh Lab only
```

```
$       route add default gw 10.0.2.1 dev br0 % change default gw to br0
```

In the above operation, we assign enp0s3's original IP to *br0*, assuming original IP assigned to *enp0s3* is 10.0.2.33, i.e., i.e., $IP_b \Leftarrow IP_e$ as shown in Figure CS-NET-00006.1. An alternate approach is to use dhcp service by issuing a command *dhclient br0* to get an IP address from the dhcp server (if available). Now, we can ping outside such as `www.google.com`.
If your still can not reach outside, please modify the *nameserver* in */etc/resolv.conf* from *127.0.0.1* to *8.8.8.8* so that we can manually add the DNS server for Google DNS. Then implement the whole process again.

9. To recover the network setting from the failure of operations

```
$       ovs-vsctl del-br br0 % delete the OVS bridge
$       ifconfig enp0s3 % reassign the IP address of the ehternet interface
$       route add default gw 10.0.2.1 enp0s3 % adding enp0s3 to the routing table
```

Related Information and Resource

Other frequently used OVS command

1. Getting general information

```
$       ovs-vsctl list open_vswitch
$       ovs-vswitchd -V
```

2. Check the status of *openvswitch*

```
$       service openvswitch status
```

3. Prints a list of configured bridges

```
$       ovs-vsctl list-br
```

4. Prints a list of ports on a specific bridge

```
$       ovs-vsctl list-ports <bridge>
```

5. Prints a list of interfaces

```
$       ovs-vsctl list interface
```

6. Creates a bridge in the switch database

```
$       ovs-vsctl add-br <bridge>
```

7. Binds an interface (physical or virtual) to a bridge

```
$       ovs-vsctl add-port <bridge> <interface>
```

8. Converts port to an access port on specified VLAN (by default all OVS ports are VLAN trunks)

```
$       ovs-vsctl add-port <bridge> <interface> tag=<VLAN number>
```

9. Used to create patch ports to connect two or more bridges together

```
$       ovs-vsctl set interface <interface> type=patch options:peer=<interface>
```

10. Show mac learning table for a bridge

```
$       ovs-appctl fdb/show br0
```

11. Show flows on *ovs*

```
$       ovs-dpctl show -s
```

12. Dump flows

```
$       ovs-dpctl dump-flows br0
```

13. To list bonds

```
$       ovs-appctl bond/list
```

14. To show bond properties for a bond

```
$       ovs-appctl bond/show bond0
```

Lab Assessments (100 points)

Lab assessment for accomplishing Task 1, Task 2, and Task3 depends on the following facts:

1. (50 points) The OVS
   - can correctly show the created bridges via the open vswitch command,

     ```
     $         ovs-vsctl show
     ```

     ```
               Port "br0"
                  Interface "br0"
                     type: internal
     ```

   - can show the correct bound between the br0 and enp0s3

     ```
     $         ovs-vsctl show
     ```

     ```
            Port "enp0s3"
               Interface "enp0s3"
     ```

   - can show the correct IP address assigned to the br0

     ```
     $         ifconfig
     ```

```
    inet 10.0.2.33
```

2. (50 points) After finishing the configuration of br0 in OVS, the machine

   - can show the correct routing table for especially for the br0 and enp0s3

     ```
     $         route -n
     ```

     ```
     Destination  Gateway       Genmask   Flags Metric Ref  Use   Iface
     0.0.0.0     10.0.2.1    0.0.0.0        UG    0     0      0     br0
     10.0.2.0  0.0.0.0      255.255.255.0 U    0     0      0     br0
     ```

   - can ping outside such as ping 8.8.8.8 correctly

     ```
     $         ping 8.8.8.8
     ```

The challenging task evaluation is optional and the instructor can decide on how to evaluate students' performance.