Présentation d’Open vSwitch

[Open vSwitch](http://openvswitch.org/) est une implémentation logicielle d’un switch ethernet. Concrètement il est constitué d’un service (ovs-vswitchd) et d’un module kernel (openvswitch\_mod). Le service permet de commuter effectivement les paquets vers les bons ports virtuels, alors que le module kernel permet de capturer le trafic provenant des interfaces réseau, et d’y ré-injecter le trafic légitime.

Open vSwitch supporte beaucoup de fonctionnalités d’un switch L2 et même d’un switch L3. Voici une liste non-exhaustive: 802.1q, 802.1ag, LACP, STP, Netflow et sFlow (L3), SPAN et RSPAN (L2)

Pour fonctionner comme n’importe quel switch, Open vSwitch utilise la notion de ports. Ces ports peuvent être en mode trunk (ils transportent par défaut tous les VLANs), ou en mode access (le trafic tagué avec le vlan correspondant y sera affecté). Ce qui caractérise avant tout un port est son adresse MAC.

Chaque port est constitué d’une ou plusieurs interfaces, qui correspondent à des interfaces du système hôte (logiques ou physiques).

Installation sous Debian [¶](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#installation_sous_debian)

Open vSwitch n’est pas disponible dans squeeze, mais il l’est dans testing:

# apt-get install -t testing openvswitch-switch openvswitch-datapath-source

openvswitch-switch contient le daemon vswitchd, alors que datapath-source contient les sources du module kernel (openvswitch n’est pas encore intégré nativement au kernel Linux, mais il le sera en 3.3).

**Construction du datapath** [**¶**](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#construction_du_datapath)

# apt-get install -t testing linux-headers-$(uname -r)

# apt-get install -t testing module-assistant

# m-a build -f openvswitch-datapath

# dpkg -i /usr/src/openvswitch-datapath-module-$(uname -r)\_\*.deb

**Conflit avec le module bridge de Linux** [**¶**](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#conflit_avec_le_module_bridge_de_linux)

Le code d’Open vSwitch peut rentrer en conflit avec le support du bridging apporté par Linux. Le plus simple est de décharger ce module, puis de blacklister.

# rmmod bridge

# cat /etc/modprobe.d/bridge-blacklist.conf

# This file blacklists the bridge driver since we want OpenVswitch instead.

blacklist bridge

**Démarrage du service** [**¶**](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#d.C3.A9marrage_du_service)

On démarre maintenant le daemon fournissant le switch virtuel:

# /etc/init.d/openvswitch-switch start

Configuration d’Open vSwitch [¶](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#configuration_d.27open_vswitch)

On imagine que nous sommes sur un hyperviseur KVM, dans lequel sont démarrées 2 machines virtuelles: les VMs A et B. Le réseau de A est fourni par l’interface virtuelle vnet0 et B par vnet1. Ces deux VMs n’ont pour l’instant pas accès au réseau physique.

Cet hyperviseur possède par ailleurs 4 interfaces physiques (eth0,1,4,8) qui sont connectées à un switch capable de faire du LACP et du 802.1q. Le port-channel est défini côté switch, mais rien n’est défini côté hyperviseur (ces 4 interfaces sont up mais inutilisées).

On doit avoir ceci dans /etc/network/interfaces:

auto eth0

iface eth0 inet manual

 up ip link set $IFACE up

auto eth1

iface eth1 inet manual

 up ip link set $IFACE up

auto eth4

iface eth4 inet manual

 up ip link set $IFACE up

auto eth8

iface eth8 inet manual

 up ip link set $IFACE up

A ce stade, l’administration de l’hyperviseur se fait soit par la **console** (réelle ou iLo), soit par le réseau à l’aide d’une **autre interface** que ces 4 là.



**Création du switch virtuel** [**¶**](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#cr.C3.A9ation_du_switch_virtuel)

Nous allons tout d’abord créer un switch virtuel, dans lequel nous ajouterons par la suite des interfaces. Nous l’appellons *vswitch*

# ovs-vsctl add-br vswitch

# ovs-vsctl show

f5050003-57de-4e68-831b-83277b329fe5

 Bridge vswitch

 Port vswitch

 Interface vswitch

 type: internal

 ovs\_version: "1.4.2"

On note qu’un bridge (un switch) a été crée. Il contient un seul port, lui-même constitué d’une seule interface (nommée elle aussi vswitch). On remarque que cette interface existe également dans Linux (*ip link show vswitch*).

Nous obtenons alors l’état suivant:



**Ajout du LACP** [**¶**](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#ajout_du_lacp)

L’étape suivante est l’ajout d’un port constitué de nos 4 interfaces physiques, agrégées en LACP. De plus, nos VMs A et B souhaitant être présentes respectivement dans les VLANs 1 et 2, nous ajouterons ces deux VLANs dans le trunk de ce port.

* Création du port LACP (bond0):

# ovs-vsctl add-bond vswitch bond0 eth0 eth1 eth4 eth8 lacp=active trunks=1,2

# ovs-vsctl show

f5050003-57de-4e68-831b-83277b329fe5

 Bridge vswitch

 Port "bond0"

 trunks: [1, 2]

 Interface "eth8"

 Interface "eth0"

 Interface "eth1"

 Interface "eth4"

 Port vswitch

 Interface vswitch

 type: internal

 ovs\_version: "1.4.2"

L’état obtenu est le suivant:



On peut surveiller l’état du port-channel avec la commande suivante:

# ovs-appctl bond/show bond0

bond\_mode: balance-slb

bond-hash-algorithm: balance-slb

bond-hash-basis: 0

updelay: 0 ms

downdelay: 0 ms

next rebalance: 4496 ms

lacp\_negotiated: true

slave eth0: enabled

 active slave

 may\_enable: true

 hash 186: 147 kB load

slave eth4: enabled

 may\_enable: true

slave eth8: enabled

 may\_enable: true

slave eth1: enabled

 may\_enable: true

**Remarque**: il peut se passer plusieurs (longues) secondes avant que les 4 interfaces soient *bundled* au sein du port-channel. Cela fait l’objet d’une négociation entre le switch physique et le vswitch.

**Ajout des VLANs** [**¶**](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#ajout_des_vlans)

L’étape suivante est facultative quand on souhaite seulement fournir du réseau aux VMs. Elle consiste à ajouter 2 *fake-bridges* pour chaque VLAN que l’on souhaite pouvoir:

* soit utiliser directement depuis l’OS hôte (par exemple affecter une adresse IP à l’hyperviseur dans un VLAN, afin de pouvoir l’administrer via cette IP)
* soit utiliser avec un logiciel comme la libvirt, afin de bridger une VM directement dans le bon VLAN (on ne peut pas pour l’instant spécifier un VLAN dans le fichier XML d’un domain, uniquement une interface à bridger).

# ovs-vsctl add-br vlan1 vswitch 1

# ovs-vsctl add-br vlan2 vswitch 2

# ovs-vsctl show

f5050003-57de-4e68-831b-83277b329fe5

 Bridge vswitch

 Port "vlan1"

 tag: 1

 Interface "vlan1"

 type: internal

 Port "vlan2"

 tag: 2

 Interface "vlan2"

 type: internal

 Port "bond0"

 trunks: [1, 2]

 Interface "eth8"

 Interface "eth0"

 Interface "eth1"

 Interface "eth4"

 Port vswitch

 Interface vswitch

 type: internal

 ovs\_version: "1.4.2"

On obtient alors:



**Remarque**: Comme dans lors de la création du vswitch, on obtient deux interfaces sous Linux, vlan1 et vlan2. Ces deux interfaces sont taguées automatiquement: tout le trafic qui y sera injecté sera acheminé dans le vswitch avec le tag du VLAN correspondant.

De plus, si on affecte une IP à ces interfaces, l’hyperviseur sera alors joignable via cette IP dans le VLAN correspondant.

**Ajout des VMs** [**¶**](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#ajout_des_vms)

L’étape suivante est la finale: nous allons ajouter les VMs dans les bons VLANs:

On veut VM A dans le vlan1 et et VM B dans le vlan2. Nous faisons donc:

# ovs-vsctl add-port vswitch vnet0 tag=1

# ovs-vsctl add-port vswitch vnet1 tag=2

Note que si l’on a créé les *fake-bridges* à l’étape précédente, nous aurions pu aussi bien faire:

# ovs-vsctl add-port vlan1 vnet0

# ovs-vsctl add-port vlan2 vnet1

On obtient alors:



# ovs-vsctl show

f5050003-57de-4e68-831b-83277b329fe5

 Bridge vswitch

 Port "vlan1"

 tag: 1

 Interface "vlan1"

 type: internal

 Port "vlan2"

 tag: 2

 Interface "vlan2"

 type: internal

 Port "vnet0"

 tag: 1

 Interface "vnet0"

 Port "vnet1"

 tag: 2

 Interface "vnet1"

 Port "bond0"

 trunks: [1, 2]

 Interface "eth8"

 Interface "eth0"

 Interface "eth1"

 Interface "eth4"

 Port vswitch

 Interface vswitch

 type: internal

 ovs\_version: "1.4.2"

A ce stade, les VM A et B peuvent communiquer avec n’importe quel hôte présent respectivement dans les VLANs 1 et 2.

C’est ce que montre ce schéma de synthèse:



A, C et E peuvent communiquer ensemble dans le vlan 1, tandis que B, D et F sont dans le vlan2. Les deux VLANs sont étanches et il faudra ajouter un routeur entre eux quelque-part dans cette architecture pour que tout ce petit monde puisse communiquer ensemble.

Intégration fine avec le service networking de Debian [¶](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#int.C3.A9gration_fine_avec_le_service_networking_de_debian)

Si comme moi vous choisissez d’affecter une IP à un *fake-bridge* afin d’administrer l’hyperviseur par ce biais (et donc d’économiser une interface physique), vous aurez comme moi envie que cette IP soit ajoutée au démarrage du serveur, et donc de la spécifier dans */etc/network/interfaces* .

Problème: le service openvswitch-switch dépend du service network. Le poisson se mort la queue…

Une solution glanée sur [internet](http://openvswitch.org/pipermail/discuss/2011-October/005886.html) consiste à utiliser udev (merci Hans van Kranenburg). On ajoute un script déclenché au chargement des interfaces vlan par le kernel, et ce script provoque la configuration de la dite interface depuis */etc/network/interfaces*

# cat /etc/udev/rules.d/openvswitch.rules

KERNEL=="vswitch\*", SUBSYSTEM=="net", RUN+="/etc/openvswitch-switch/udev.sh %k"

KERNEL=="vlan\*", SUBSYSTEM=="net", RUN+="/etc/openvswitch-switch/udev.sh %k

Puis le script *udev.sh* :

#!/bin/sh

logger -t ovs-udev "$ACTION $1"

case $ACTION in

 add)

 ifup --force $1 | logger -t ovs-udev

 ;;

 remove)

 # hm, this won't ever work, because this script gets

 # called after the device is already removed...

 ifdown --force $1 | logger -t ovs-udev

 ;;

esac

exit 0

Il reste alors à spécifier correctement l’interface dans // /etc/network/interfaces /:

iface vlan1 inet static

 address a.b.c.d

 netmask 255.255.255.0

 gateway a.b.c.e

Intégration avec la libvirt [¶](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#int.C3.A9gration_avec_la_libvirt)

Il faut modifier le domaine pour qu’il ressemble à la section suivante:

# virsh edit VM-A

[...]

<interface type='bridge'>

 <mac address='52:54:00:ba:e9:f7'/>

 <source bridge='vlan1'/>

 <virtualport type='openvswitch'/>

 <model type='virtio'/>

 <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>

</interface>

[...]

Il n’est pas possible de faire la même chose *facilement* avec virt-manager. Il faut d’abord passer l’interface en mode macvtap, spécifier openvswitch dans le type du virtualport, sauver, puis repasser l’interface en mode bridge sur le *fake-bridge* adéquat (ici vlan1), avant de sauver à nouveau. Assez fastidieux mais la libvirt va évoluer pour mieux s’interfacer avec Open vSwitch.

Quelques commandes utiles [¶](http://tech.covoiturage.fr/2012/08/22/open-vswitch-un-switch-logiciel-pour-des-reseaux-virtuels/#quelques_commandes_utiles)

* Afficher la forwarding db (l’équivalent d’un *show mac address-table* sur cisco)

# ovs-appctl fdb/show vswitch

 port VLAN MAC Age

 2 2 a0:cf:5b:ed:98:c7 206

 2 1 00:21:5a:a6:44:a6 169

 2 1 98:4b:e1:66:a5:1c 161

 2 2 88:f0:77:d6:d4:88 144

 2 1 78:e7:d1:f6:fc:54 133

 2 1 52:54:00:d0:24:f5 108

 2 2 64:d9:89:65:6a:c1 33

 2 2 78:e7:d1:f6:bc:bc 10

[...]

* Afficher l’état du LACP pour le port bond0:

# ovs-appctl bond/show bond0

bond\_mode: balance-slb

bond-hash-algorithm: balance-slb

bond-hash-basis: 0

updelay: 0 ms

downdelay: 0 ms

next rebalance: 6207 ms

lacp\_negotiated: true

slave eth0: enabled

 active slave

 may\_enable: true

 hash 186: 129 kB load

slave eth4: enabled

 may\_enable: true

slave eth8: enabled

 may\_enable: true

slave eth1: enabled

 may\_enable: true