VDE Basic Networking

From Virtualsquare

VDE is the Virtual Distributed Ethernet. You can use it to connect virtual machines or linux boxes or any combination of the two. Like a real modern Ethernet network a VDE is composed by switches and cables. Each switch has several sockets where machines can be "plugged-in".

Ethernet and Virtual Distributed Ethernet

Contents

Two virtual machines connected by a VDE switch

Step 1: start the switch

Each switch has a directory where it keeps all its temporary files. This directory is also used as the *name* of the switch.

```
$ vde_switch -s /tmp/switch1
```

This switch runs as a foreground process, so the shell prompt will not be returned. If you type *Enter* you'll have the prompt of the configuration interface of the switch itself. Type *help* if you want to see the command list and syntax.

Step 2: run a QEMU/KVM virtual machine

You can connect QEMU/KVM to VDE using two different transports: the old *vdeq* wrapper and native support.

Step 2a: Connecting with natively:

If you use qemu > 0.9.1 or kvm >= 72, both compiled with *--enable-vde* you can use native VDE support. Please refer to official qemu/kvm documentation to setup the network with VDE.

Step 2b: Connecting old vdeq wrapper:

```
$ vdeq qemu -hda qemu-image -m 128
```

The command vdeq runs qemu and automagically defines a vde network interface. Change *-hda qemu-image -m 128* with your favourite qemu flags. If it is installed in your system (AFAIK all distribution supporting vde do it) you can also use the shorten form:

```
$ vdeqemu -hda qemu-image -m 128
```

Set the networking support of the operating system running on the qemu virtual machine. For example if it is some flavour of GNU-Linux you can type:

```
qemu-linux$ ifconfig eth0 10.0.0.1 netmask 255.255.255.0 up
```

Step 3: run a User-Mode Linux

You can connect connect User-Mode Linux to VDE using two different transports: the old *daemon backend*, originally written for uml_switch or the new *vde backend* which offers much more configuration flexibility.

Step 3a: Connecting with new vde backend:

If you're planning to use new vde backend you should use at least a 2.6.24-rc1 guest kernel.

Note: Linux vanilla 2.6.25 and newer versions need a bugfix (we submitted it several times but it has not been committed to the mainstream yet):

```
--- linux-2.6.28.2/arch/um/drivers/vde_user.c   2009-01-25 01:42:07.000000000 +0100
+++ linux-2.6.28.2-vde/arch/um/drivers/vde_user.c       2009-02-01 11:07:40.000000000 +0100
@@ -78,7 +78,7 @@
 {
        struct vde_open_args *args;

-       vpri->args = kmalloc(sizeof(struct vde_open_args), UM_GFP_KERNEL);
+       vpri->args = uml_kmalloc(sizeof(struct vde_open_args), UM_GFP_KERNEL);
        if (vpri->args == NULL) {
                printk(UM_KERN_ERR "vde_init_libstuff - vde_open_args "
                       "allocation failed");
```

User-Mode-Linux with VDE requires a larger stack for kernel processes. During the kernel configuration phase (make ARCH=um menuconfig) set the option named "Kernel stack size order" (under the "UML-specific options" menu) to 1 or higher. Alternatively you can edit by hand the .config file: search the tag 'CONFIG_KERNEL_STACK_ORDER' and change it as follows:

```
CONFIG_KERNEL_STACK_ORDER=1
```

When the kernel is ready start the UML machine in this way:

```
$ linux ubd0=uml-image mem=128k eth0=vde,/tmp/switch1
```

Obviously change 'ubd0=uml-image mem=128k' with your user-mode linux flags.

Set the networking of the User-Mode linux machine as follows:

```
uml-linux$ ifconfig eth0 10.0.0.2 netmask 255.255.255.0 up
```

Whole syntax of vde backend is:

ethN=vde,<vde_switch>,<mac addr>,<port>,<group>,<mode>,<description>

Step 3b: Connecting with old daemon backend:

```
$ linux ubd0=uml-image mem=128k eth0=daemon,,,/tmp/switch1/ctl
```

Obviously change 'ubd0=uml-image mem=128k' with your user-mode linux flags. Remember to add ctl to the name of the switch. Set the networking of the User-Mode linux machine as follows:

```
uml-linux$ ifconfig eth0 10.0.0.2 netmask 255.255.255.0 up
```

Step 4: test the results

The two virtual machines are now virtually connected. You can try the following:

```
qemu-linux$ ping 10.0.0.2
uml-linux$ ping 10.0.0.1
```

In the same way you can add several virtual machines to the virtual network. Some virtual machines (e.g. qemu) have built-in MAC addresses for their virtual Ethernet Interfaces, so if you start several virtual machines of the same kind they cannot communicate together (they have all the same physical address!). For Qemu the solution is to manually set up their mac address as follows:

```
$ vdeqemu -net vde,vlan=0 -net nic,vlan=0,macaddr=52:54:00:00:AA:02 -hda qemu-image1 -m 128
$ vdeqemu -net vde,vlan=0 -net nic,vlan=0,macaddr=52:54:00:00:AA:04 -hda qemu-image2 -m 128
```

NB: The first byte of all MAC addresses assigned to hosts must be **even**. An address with odd value in the first byte is by definition a broadcast or multicast address, thus the switches cannot work properly. This limitation holds both on VDE and on real Ethernets (although it is very rare to assign by hand a MAC address on real ethernet interfaces). e.g. a mac address of the form xy:xx:xx:xx:xx:xx can be assigned to a host only if y gets one of the following values 0,2,4,6,8,a,c,e.

A VDE-Switch connected to the Internet

Slirp: a virtual NAT router as a process

The siplest way to connect your virtual network to the Internet is slirp. Slirp is a process that appear as a router on the virtual network. The slirp process forwards all the communications coming from the virtual network to the Internet.

Start slirpvde prior to boot the virtual machine as follows:

```
$ slirpvde -s /tmp/switch1 --dhcp
```

Now boot your virtual machines and let them keep their address using automatic configuration (dhcp). They'll receive addresses like 10.0.2.xxx and you'll be able to use your favourite networking commands to access the Internet (provided that they are IPv4 clients, IPv6 is unsupported yet).

Connecting the switch to a tap interface

This feature requires access to the tuntap facility (/dev/net/tun on linux), usually this means root access.

```
# vde_switch -s /tmp/switch1 -tap tap0 -m 666
```

The switch starts with a port connected to the virtual interface tap0. The -m flag is required to allow user virtual machines to join the network, otherwise all the virtual machines are required to run as root. The host machine is now connected to the virtual network. It is possible to use all the bridging/routing/firewalling /dhcp/tracing support provided by the hosting operating system for the virtual network.

Several distributions provide startup scripts or ifup/ifdown configurations to start switch at boot time.

Connecting VDE-switched together

Step 1: run several switches

On the same computer you can run several switches, provided they have different names:

```
$ vde_switch -s /tmp/switch1
$ vde_switch -s /tmp/switch2
```

It is possible to run switches on different computers:

```
host1$ vde_switch -s /tmp/switch
host2$ vde_switch -s /tmp/switch
```

Step2: connect them together

It is straightforward simple:

```
$ dpipe vde_plug /tmp/switch1 = vde_plug /tmp/switch2
```

dpipe is a double pipe: the two commands separated by a = sign are mutually interconnected: the output of the first is the input for the second and viceversa. In this way the two plugs *plugged-in* the two switches exchange their packets...

If the switch are running on different computers we need a wire, i.e. a program able to forward a stream connection from a computer to the other. *ssh* is the simplest (safe and quite fast) example

```
host1$ dpipe vde_plug /tmp/switch = ssh host2 vde_plug /tmp/switch
```

when the switch are connected all the virtual (and real) machines connected to one can communicate with those connected to the other. It has exactly the same effect for real (physical, not-virtual) ethernet of connecting a cross-cable between two switches.

Obviously several switches can be conntected in the same way. Remember that all the problems of the Ethernet do exist in the virtual Ethernet. For example cycles on the network are allowed only if the fast spanning tree protocol is enable otherwise the packets loop on the network saturating the channels.

Dump or Monitor switch traffic

You can dump or monitor traffic using **pdump** plugin. Nowadays if you want to use VDE with plugins support you must compile passing *--enable-experimental* flag to *./configure*.

1. Dump traffic

Start *vde_switch*, then from its console load and activate *pdump*:

```
$ vde_switch -F -s /tmp/switch1


vde$ plugin/add /usr/local/lib/vde2/plugins/pdump.so
1000 Success

vde$ pdump/active 1
0000 DATA END WITH '.'
.
1000 Success

vde$
```

Then, when you want to stop your dump:

```
vde$ pdump/active 0
0000 DATA END WITH '.'
.
1000 Success

vde$
```

You will find a **vde_dump.cap** file in your working directory.

2. Monitor traffic

We're going to write switch traffic into a FIFO and then use *wireshark* to monitor.

First, create the FIFO:

```
$ mkfifo /tmp/myfifo
```

Then, instead of type the commands directly into the switch, we create a configuration file (let's call it

vde_pdump.conf) containing the following lines:

```
plugin/add /usr/local/lib/vde2/plugins/pdump.so
pdump/filename /tmp/myfifo
pdump/buffered 0
pdump/active 1
```

Now start *vde_switch*:

```
$ vde_switch -F -s /tmp/switch1 -f vde_pdump.conf
```

At this point the switch will hang waiting for the other side of the FIFO to be opened, so type

```
$ wireshark -i /tmp/myfifo
```

and then start a new live capture (if you don't want the switch to hang simply start a new live capture first).

Retrieved from "http://wiki.v2.cs.unibo.it/wiki/index.php?title=VDE_Basic_Networking&oldid=63"
Category: Pages with broken file links
