VDE

From Virtualsquare

Contents

VDE components

`vde_switch`

The vde_switch is a virtual switch provided with the vde networking architecture. As vde_switch can interconnect several virtual networking devices multiple vde_switches can be connected together with vde_cables.

Main features

- **VLAN**

  It makes possible to partition available switch ports into subsets. Each subset is called a Virtual LAN or VLAN. With this logical division of the virtual network it is possible to have several indipendent logical networks within the same virtual switch. Moreover, this also may be useful to separate the

network traffic between hosts on the same VLAN and hosts that belong to the other VLANs.

- **Fast Spanning Tree Protocol**

  Implemented in vde_switch to prevent loops. Like in real switched networks the protocol finds a spanning tree for the mesh network and disables links that are not included within the spanning tree. When Fast Spanning Tree Protocol is running, ports cab be classified as their role in the network:
  - *Root*
    A forwarding port that has been elected for the spanning-tree topology
  - *Designated*
    A forwarding port for every LAN segment
  - *Alternate/Backup*
    A backup/redundant path to a segment where another bridge port already connects or an alternate path to rootswitch.
  - *Edge*
    Ports that don't take part in the process of building network topology can be marked as *Edge* ports. They are usually connected to end-systems that do not influence Spanning Tree computation.
  - *Unknown*
    Unidentifiable role for the port.

- **Command line management**

  It is possible to manage vde_switches both from a management socket (when running switch as detached process) and from standard input (when running switch as foreground process). The command line is useful to create VLANs, enable Fast Spanning Tree Protocol, monitor switch ports, switch status and data socket.

User definable options

When starting a new vde_switch instance there are several customizable options:

- number of ports
- operation mode
- switch mac address
- configuration file
- management socket and permissions
- data socket and permissions
- tap interface

Since vde_switch is the core of vde virtual networking architecture it is highly customizable to be as flexible as possible. In the following subsections are briefly treated almost all its features.

Some usage examples

Default options

```
$ vde_switch

vde:
```

Default working directory for the switch is `/tmp/vde.ctl` and default access mode of the directory is `2700`.

By pressing return the management prompt for the switch appears. With `help` command it is possible to get a list of the possible commands (the actual list may differ from here depending on the version of the vde_switch and on the options enabled at compile time).

```
vde$ help
0000 DATA END WITH '.'
COMMAND PATH        SYNTAX          HELP
------------        --------------  ------------
ds                  ============    DATA SOCKET MENU
ds/showinfo                         show ds info
help                [arg]           Help (limited to arg when specified)
logout                              logout from this mgmt terminal
shutdown                            shutdown of the switch
showinfo                            show switch version and info
load                path            load a configuration script
debug               ============    DEBUG MENU
debug/list                          list debug categories
debug/add           dbgpath         enable debug info for a given category
debug/del           dbgpath         disable debug info for a given category
plugin              ============    PLUGINS MENU
plugin/list                         list plugins
plugin/add          library         load a plugin
plugin/del          name            unload a plugin
hash                ============    HASH TABLE MENU
hash/showinfo                       show hash info
hash/setsize        N               change hash size
hash/setgcint       N               change garbage collector interval
hash/setexpire      N               change hash entries expire time
hash/setminper      N               minimum persistence time
hash/print                          print the hash table
hash/find           MAC [VLAN]      MAC lookup
fstp                ============    FAST SPANNING TREE MENU
fstp/showinfo                       show fstp info
fstp/setfstp        0/1             Fast spanning tree protocol 1=ON 0=OFF
fstp/setedge        VLAN PORT 1/0   Define an edge port for a vlan 1=Y 0=N
fstp/bonus          VLAN PORT COST  set the port bonus for a vlan
fstp/print          [N]             print fst data for the defined vlan
port                ============    PORT STATUS MENU
port/showinfo                       show port info
port/setnumports    N               set the number of ports
port/sethub         0/1             1=HUB 0=switch
port/setvlan        N VLAN          set port VLAN (untagged)
port/create         N               create the port N (inactive|notallocatable)
port/remove         N               remove the port N
port/allocatable    N 0/1           Is the port allocatable as unnamed? 1=Y 0=N
port/setuser        N user          access control: set user
port/setgroup       N user          access control: set group
port/epclose        N ID            remove the endpoint port N/id ID
port/resetcounter   [N]             reset the port (N) counters
port/print          [N]             print the port/endpoint table
port/allprint       [N]             print the port/endpoint table (including inactive port)
vlan                ============    VLAN MANAGEMENT MENU
vlan/create         N               create the VLAN with tag N
vlan/remove         N               remove the VLAN with tag N
vlan/addport        N PORT          add port to the vlan N (tagged)
vlan/delport        N PORT          add port to the vlan N (tagged)
vlan/print          [N]             print the list of defined vlan
vlan/allprint       [N]             print the list of defined vlan (including inactive port)
.
1000 Success
```

Customizing daemon

```
$ vde_switch --daemon --sock /tmp/myvde.ctl --mgmt /tmp/myvde.mgmt
```

Now to access the command line management interface for the virtual switch it is possible to use a tool provided with vde that connect to the management unix socket `/tmp/muvde.mgmt`.

```
$ vdeterm /tmp/myvde.mgmt
```

```
VDE switch V.2.3.1
(C) Virtual Square Team (coord. R. Davoli) 2005,2006,2007 - GPLv2

vde[/tmp/muvde.mgmt]:
```

Vdeterm provides command completion and editing, and history navigation. Vdeterm also manages the visualization of asynchronous debugging messages.

The management socket is a standard PF_FILE stream socket: it is easy to interface other programs. For example socat can be used instead of vdeterm (socat is not vde specific, thus it has less features):

```
$ socat READLINE unix:/tmp/muvde.mgmt
VDE switch V.2.3.1
(C) Virtual Square Team (coord. R. Davoli) 2005,2006,2007 - GPLv2

vde$
```

Attaching a tap interface

It is possible to connect switches to tap interfaces of the hosting operating system. Usually for security reasons tun/tap interfaces creation and configuration is restricted to system administrator. To allow a user to open and use the `/dev/net/tun` device the administrator have to preconfigure a persistent tap interface. This can be done with tools like `tunctl` provided within uml-utilities from User Mode Linux (http://www.user-mode-linux.org/) .

However the user can not modify any aspect of the persistent tap interface created. This task is up to the system administrator.

```
# tunctl -u username -t tap0
Set 'tap0' persistent and owned by uid 1000

# ifconfig tap0 192.168.0.1 netmask 255.255.255.0
```

Once the tap interface is created and configured the user can start a vde_switch connected to it.

```
$ vde_switch --tap tap0

vde: port/allprint
0000 DATA END WITH '.'
Port 0001 untagged_vlan=0000 ACTIVE - Unnamed Allocatable
  -- endpoint ID 0006 module tuntap      : tap0
.
1000 Success
```

Switch management in detail

From vde_switch command line management interface is possible to tune almost every aspect of the virtual switch. Configuration command are divided in sections as can be seen from command line help.

General commands

- `help [topic]`

  prints entire help information. May be limited to specific subsection if subsection name is given as command argument

- `logout`

  used to logout from management command line when logged in from configuration socket

- `shutdown`

  causes the switch process to terminate

- `showinfo`

  prints general information about the switch such as uptime, process ID, uptime, switch's MAC address and management socket path and permissions (if any). It also tells how many unsent packets are waiting inside packet queue.

- `load filename`

  allows the user to load a script of management commands from file. The script syntax is the same of command line interface, with one command per line.

Data socket commands

- `ds/showinfo`

  prints general information about data socket, its path and access mode

Hash table commands

Storage of MAC addresses inside vde_switch is managed via an hash table.

- `hash/showinfo`

  Prints out a summary of hash table details: hash table size, garbage collector interval and expiration and minimum persistence time.

- `hash/setsize size`

  Changes hash table size to fit a larger and crowded virtual network environment with a lot of hosts or MAC address.

- `hash/setgcint seconds`

  Changes garbage collector interval. Every interval expiration the garbage collector looks the hash table for items that have to be removed.

- `hash/setexpire seconds`

  Changes garbage collector expire time. When a new entry is added to the hash table a per-entry counter representing the age of the entry inside the hash table. This counter is increased every second and when it goes beyond expiration time the entry will be removed from garbage collector.

- `hash/setminper seconds`

  Changes minimum persistence time. (?)

- `hash/print`

  Prints out the whole content of hash table and for each entry it prints hash key, mac address, VLAN, port and current age of the entry in the hash table.

- `hash/find mac_address [VLAN_id]`

  Returns hash entry containing the given mac address.

Fast Spanning Tree Protocol commands

- `fstp/showinfo`

  Prints out general information about fast spanning tree protocol implementation inside vde_switch. This includes

- `fstp/setfstp 0/1`

  Enables or disables fast spanning tree protocol

- `fstp/setedge VLAN_id port_num 0/1`

  Sets the specified port to be an edge-port. Edge-ports are those connected directly to end-systems. Since edge-ports do no take part in the building of network topology they are rapidly switched to forward. In this way edge-ports skip listening and learning stages of the protocol resulting in a faster convergence to stable topology. Further details about fast spanning tree protocol here [1] (http://en.wikipedia.org/wiki/Spanning_tree_protocol#Rapid_Spanning_Tree_Protocol_.28RSTP.29)

- `fstp/bonus VLAN_id port_num cost`

  It is used to set the bonus port and its priproty within a VLAN (?)

- `fstp/print [VLAN_id]`

  Prints out the current state of Spanning Tree Protocol for each defined VLAN.

Port management commands

- `port/showinfo`

  Prints information about switch ports. Maximum number of currently available ports and current operating mode.

- `port/setnumports number`

  Used to change number of ports of the switch.

- `port/sethub 0/1`

  Sets switch in hub mode. Ethernet frames are broadcasted everywhere.

- `port/setvlan port_num VLAN_id`

  Adds the specified port as untagged to the specified vlan. In this way the specified port is only part of

the specified vlan.

- `port/create port_num`

  Creates a new port even if nothing is connected to it.

- `port/remove port_num`

  Removes the specified port.

- `port/allocatable port_num 0/1`

  Sets a port as reserved. In this way it is not possible to connect something to it without specifying its exact port number.

- `port/setuser port_num user`

  Port access control. Set the user for this port.

- `port/setgroup port_num group`

  Port access control. Set the group for this port. Any user of this group will be allowed to use the port. If neither the user nor the group is set for a port, port access control is disabled.

- `port/epclose port_num endpoint_id`

  Closes the endpoint associated with the specified port.

- `port/print [port_num]`

  Prints information about all active ports.

- `port/allprint [port_num]`

  Prints information about all ports.

VLAN commands

- `vlan/create VLAN_id`

  Creates a new VLAN with the given id.

- `vlan/remove VLAN_id`

  Removes the specified VLAN. A VLAN is not removable if there are ports associated with it.

- `vlan/addport VLAN_id port_num`

  Adds specified port to specified VLAN as tagged port. In this way if the specified port is already part of a VLAN it becomes also part of the specified VLAN.
  Such port have to be used when connecting VLANs distributed over multiple switches. Outgoing frames from tagged ports have an additional header containing information about the LAN where it was generated. This way of distributing VLANs over different switches is called *VLAN trunking* and it is part of the [802.1Q (http://en.wikipedia.org/wiki/IEEE_802.1Q%7CIEEE)] standard.

- `vlan/delport VLAN_id port_num`

  Removes specified tagged port from specified VLAN.

- `vlan/print [VLAN_id]`

  Prints information about VLANs.

- `vlan/allprint [VLAN_id]`

  Prints information about VLANs including in the output also inactive ports.

vde_plug

A vde_plug is like an ethernet plug and was designed to be connected to vde_switches. Everything that is injected into the plug from standard input is sent into the vde_switch which it is connected to. On the other hand everything that comes from the virtual network to that plug goes to the vde_plug standard output. In the following subsection is presented and explained that enables the creation of so called vde_cables.

dpipe

Two vde_plug can be connected together with a simple but powerful tool developed to work in virtual distributed ethernet environment. `dpipe` also known as bi-directional pipe is able to run two or more commands diverting standard output of the first command into the standard input of the second command and vice-versa.

```
$ dpipe vde_plug /tmp/vde1.ctl = vde_plug /tmp/vde2.ctl
```

This simple example shows how is possible to connect two vde_switches together by running two vde_plugs connected to respective vde control sockets via the bi-directional channel provided by `dpipe`.

```
$ dpipe vde_plug /tmp/vde.ctl = ssh foo@remote.host.org vde_plug /tmp/vde_remote.ctl
```

Here can be seen how virtual distributed ethernet can become distributed for real. In this example a vde_switch running locally is connected to a remote vde_switch using a secure shell channel. This is done simply running a vde_plug connected to the remote vde control socket on the remote host. Potentially any program able to provide a bi-directional channel both remotely or locally can be used as a vde_wire to connect vde networking components. Another example could be an UDP unencrypted channel built with `netcat` utility.

One instance of netcat connected to a vde_switch waiting for incoming connections on the remote machine:

```
$ dpipe vde_plug /tmp/vde.ctl = nc -l -u -p 8000
```

That a netcat client connecting to the remote one and with standard input and output connected with dpipe to the local vde_switch:

```
$ dpipe vde_plug /tmp/vde_local.ctl = nc -u 8000 remove.host.address.org
```

vde_cryptcab

In previous examples have been used tools like ssh or netcat to interconnect remote vde_switches. Although these two tools are very simple and intuitive to use they are both troublesome: netcat creates unencrypted connections and for this reason does not protect from traffic sniffing and intrusion; on the other hand ssh gives protection from traffic sniffing because the traffic transferred with ssh is encrypted but it has poor performances. The reason is that when overlaying two TCP transport control layers they work concurrently to keep the stream connection interfering almost all the time.

vde_cryptcab have been developed within the virtual square project to provide a secure and efficient tool to interconnect vde networking components distributed over different machines or different underlying networks. vde_cryptcab uses ssh to exchange a secret key and then creates an encrypted UDP connection.

Let's start a vde_cryptcab server connected to a vde_switch on a remote machine:

```
$ vde_cryptcab -s /tmp/vde.ctl -p 12000
```

This vde_cryptcab server will accept UDP datagrams on port 12000 and multiple connections authenticated via ssh. So it is possible to connect multiple remote vde_cryptcab clients to the same vde_cryptcab server. All datagrams are sent to udp port 12000.

```
$ vde_cryptcab -s /tmp/vde_local.ctl -c username@remote.host.org:12000
username@remote.host.org's password:
.blowfish.key                                 100%   24     0.0KB/s   00:00
```

Note that during initialization a blowfish secret key has been transferred to remote cryptcab server. The key will be used to encrypt UDP datagrams from and to the server.

`wirefilter`

Another useful tool for testing purposes could be a program that simulates problems, limitations and errors of real wired connections, like noise, bandwidth and so on. A tool that does this has been developed within virtual distributed ethernet to operate in its environment. This tool can be inserted into a bi-directional pipeline say between two vde_plugs that interconnect two vde_switches, and can introduce virtual errors or limits on the line. `wirefilter` can control several connection parameters:

- percentage of packet loss
- extra delay on packet transmisison
- percentage of duplicated packets
- channel bandwidth
- interface speed
- maximum capacity of packet queue
- maximum transmission unit
- corrupted bits per megabyte
- packet sorting

Since wirefilter works on bi-directional channels it is also possible to fine-tune the filtering by choosing which direction of the stream is affected by wirefilter settings. A typical example of wirefilter usage could be

```
$ dpipe vde_plug /tmp/vde1.ctl = wirefilter -M /tmp/wiremgmt = vde_plug /tmp/vde2.ctl
```

In this example wirefilter is in the middle of a bi-directional pipe that connects vde_switches together via two vde_plugs. It is possible to differentiate filtering for left-to-right or right-to-left channel. Note that like

in vde_switch also in wirefilter it is possible to specify a unix socket to manage filter settings at runtime via vdeterm.

```
$ vdeterm /tmp/wiremgmt

VDE wirefilter V.2.3.1
(C) R.Davoli 2005,2006 - GPLv2

VDEwf[/tmp/wiremgmt]:help

COMMAND      HELP
------------ ------------
help         print a summary of mgmt commands
load         load a configuration file
showinfo     show status and parameter values
loss         set loss percentage
lostburst    mean length of lost packet bursts
delay        set delay ms
dup          set dup packet percentage
bandwidth    set channel bandwidth bytes/sec
speed        set interface speed bytes/sec
noise        set noise factor bits/Mbyte
mtu          set channel MTU (bytes)
chanbufsize  set channel buffer size (bytes)
fifo         set channel fifoness
shutdown     shut the channel down
logout       log out from this mgmt session
...
Success
```

vde_plug2tap

vde_plug2tap is another plug tool that can be connected to vde_switches. Instead of using standard input and standard output for network I/O everything that come from vde_switch to the plug is redirected to the specified tap interface. In the same way everything injected into the tap interface is redirected to the vde_switch.

```
$ vde_plug2tap --daemon -s /tmp/myvde.ctl tap0
```

It is also possible to attach a tap interface during vde_switch creation, obtaining the same result.

vdeqemu - vdekvm

N.B. These tools are obsolete: qemu and kvm have already builtin vde support. Using recent versions of kvm/qemu the example below becomes:

```
$ vde_switch -d -s /tmp/vde.ctl -t tap0 -M /tmp/mgmt
$ qemu -hda /path/to/image.img -net nic -net vde,sock=/tmp/vde.ctl
```

kvm can be used instead of qemu:

```
$ vde_switch -d -s /tmp/vde.ctl -t tap0 -M /tmp/mgmt
$ kvm -hda /path/to/image.img -net nic -net vde,sock=/tmp/vde.ctl
```

These tools are wrappers for running qemu/kvm virtual machines and get them connected to a vde_switch. Substantially they have the role of calling qemu/kvm with right network parameters by re-writing the command line. The only thing to know is the path for the desired vde_switch to connect to: vdeqemu/vdekvm launch qemu/kvm with the desired number of emulated network interfaces connected to

the specified vde_switch(es).

First of all a vde_switch must be running and ready to accept connections on its control socket. Note that the vde_switch is connected to a preconfigured tap interface to make guest and host networks easier to reach.

```
$ vde_switch -d -s /tmp/vde.ctl -t tap0 -M /tmp/mgmt
```

Once vde_switch is started a new instance of qemu can be connected to it via vdeqemu wrapper

```
$ vdeqemu -hda /path/to/image.img -net nic -net vde,sock=/tmp/vde.ctl
```

Taking a look in the vde_switch management console it is possible to check what is connected to vde_switch after vdeqemu/vdekvm has been launched:

```
$ vdeterm /tmp/mgmt
...
vde[tmp/mgmt]: port/print
0000 DATA END WITH '.'
Port 0001 untagged_vlan=0000 ACTIVE - Unnamed Allocatable
  -- endpoint ID 0006 module tuntap      : tap0
Port 0002 untagged_vlan=0000 ACTIVE - Unnamed Allocatable
  -- endpoint ID 0007 module unix prog   : vdeqemu user=render PID=14422  SOCK=/tmp/vde.14422-00000
.
Success
```

The usage of vdekvm is the same of vdeqemu: they are both simple links to vdeq.

By default qemu uses the same MAC address for every virtual machine, so if you plan to use several instances of qemu be sure to explicitly set a different MAC address for each virtual machine.

While generating your address beware to not use broadcast/multicast reserved MACs, ethernet rules say: the multicast bit is the low-order bit of the first byte, which is "the first bit on the wire". For example 34:12:de:ad:be:ef is an unicast address, 35:12:de:ad:be:ef is a multicast address (see ETHERNET MULTICAST ADDRESSES section in ethertypes (http://www.iana.org/assignments/ethernet-numbers) for more informations).

slirpvde

slirpvde is a slirp interface for VDE networks. It acts like a networking router connected to a vde_switch and provides connectivity from the host where it is running to virtual machines inside the virtual network. slirpvde is not the only way for virtual machines within a virtual distributed ethernet network to communicate with the outside world but its main feature is that it can be run using standard user privileges.

Every connection from a machine within the virtual network to slirpvde internal address is translated, masqueraded and re-generated by slirpvde and redirected to host machine stack. Like most of the intermediate systems it provides basic functionalities like dhcp service, port forwarding and dns requests re-mapping.

```
$ slirpvde -d -s /tmp/vde.ctl -dhcp
```

Launching slirpvde and just specifying the vde_switch control socket where virtual machines are connected is enough to provide access to external network to all virtual machines connected to that vde_switch. The additional `-dhcp` option tells slirpvde to provide also dynamic network addresses assignment.

```
$ vdeterm /tmp/mgmt
...
vde[/tmp/mgmt]: port/print
0000 DATA END WITH '.'
Port 0001 untagged_vlan=0000 ACTIVE - Unnamed Allocatable
  -- endpoint ID 0006 module tuntap      : tap0
Port 0002 untagged_vlan=0000 ACTIVE - Unnamed Allocatable
  -- endpoint ID 0007 module unix prog   : vdeqemu user=render PID=14422  SOCK=/tmp/vde.14422-00000
Port 0003 untagged_vlan=0000 ACTIVE - Unnamed Allocatable
  -- endpoint ID 0009 module unix prog   : slirpvde: user=render PID=14554  SOCK=/tmp/vde.14554-00000
.
Success
```

With a look to vde_switch management interface it is possible to see that slirpvde is connected to port 3 of the vde_switch. On the other ports can be seen a qemu virtual machine and a tap interface. Nothing forbids to use dhcp service provided by slirpvde to use it also to configure the tap interfaces connected to the switch.

slirpvde6

2011 January: in the experimental branch

Slirpvde6 is the new implementation of slirpvde based on LWIPv6. It supports both IPv4 and IPv6.

The same example can use slirpvde6 instead of slirpvde:

```
$ slirpvde6 -d -s /tmp/vde.ctl -dhcp
```

Here, slirpvde6 provides an IPv4 slirp service. The default gateway ad- dress is 10.0.2.1/24. slirpvde6 uses the same interface for all its services such as the dns forwarder or the dhcp server. In the port list on the switch slirpvde6 appears as a LWIPv6 service:

```
Port 0002 untagged_vlan=0000 ACTIVE - Unnamed Allocatable
  Current User: renzo Access Control: (User: NONE - Group: NONE)
  IN: pkts              482            bytes                     141526
  OUT: pkts            2893            bytes                     189153
   -- endpoint ID 0011 module unix prog            : LWIPv6 if=vd0
   user=renzo PID=5260 SOCK=/var/run/vde.ctl/.05260-00000
```

slirpvde6 supports several addresses, the user may specify any mix of IPv4 and IPv6 addresses:

```
$ slirpvde6 -d -H10.0.2.1/24 -H2001::1/64 -s /tmp/vde.ctl -dhcp -r
```

the -r option activates the router advertisement daemon (for autoconfiguration).

slirpvde6 is able to act as a stateless translator: if the VDE network is IPv6 only (no IPv4 addresses), all the IPv6 requests to IPv4 mapped hosts (::ffff:0:0/96) are converted by slirpvde6 into IPv4. Unfortunately the standard on how to support this conversion is still unsettled; the "IPv6 Addressing of IPv4/IPv6 Translators" working group of IETF is still open. LWIPv6 supports the management of IPv4 mapped addresses which is one of the working group's proposals. Unfortunately this feature is not currently implemented in the networking stacks provided by many (all?) of the primary popular and commercial operating systems.

Retrieved from "http://wiki.v2.cs.unibo.it/wiki/index.php?title=VDE&oldid=13"
