Manual for *vnd* version 0.5

April 9th, 2014
Damien Magoni

1. General information

The *virtual network device* (*vnd*) is a program for emulating some network components such as links, hubs and switches. It is controlled via a command line interface. It can be used to interconnect the emulators of virtual machines. It has successfully been used with QEMU (http://wiki.qemu.org) and Dynamips (http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator).

2. Convention

Throughout this manual, the times font is used for the explanatory text, the `courier` font is used for exact input commands and output results and the arial font is used as a placeholder for arguments and variables.

Important note: the command line interface uses the *space* character as its token delimiter, thus never use spaces in arguments and identifiers.

3. Compilation

The program is written in C++ and requires the Boost (http://www.boost.org/) and OpenSSL (http://www.openssl.org/) libraries to be compiled.
On UNIX, the libvdeplug library is also needed (http://vde.sourceforge.net/).
A Makefile is provided to ease compilation. The BP and VP variables need to be edited with your proper Boost and VDE paths.
The program has been successfully compiled on x64 Debian Squeeze and Wheezy and on both x86 and x64 Windows 7 by using Visual C++ 9.0.

4. Program start

The program can be launched with the following arguments:

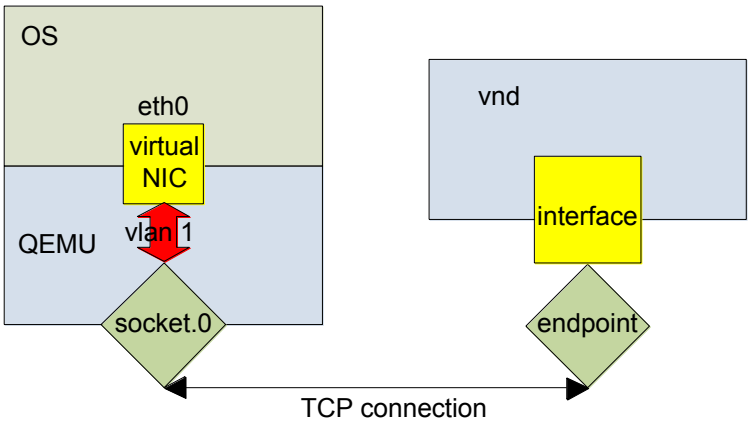
`vnd -n` <device_name> `-f` <conf_file> `-d` <log_dir>

If only some or no arguments are given, default arguments are used for the missing ones.
The effects of the flags are:
`-n` <device_name> : defines the name/identifier of the switch
`-f` <conf_file> : defines the name of a file containing commands to be executed at startup
`-d` <log_dir> : defines the directory where the logs of the program are written

After launch, a `%` prompt is presented to the user who can then enter commands.

5. Brief tutorial

In this brief tutorial, we will create a virtual 10Mbps link between a QEMU virtual machine and a *vnd* by using a TCP connection between them. We assume that everything runs on the same physical host and we use the loopback interface as the only IP address.



The *vnd* is started first and the following commands are issued:

```
% mode switch
% add intf1 nic eth
% insr endp1 len tcp 127.0.0.1 5001 ? ?
% bind endp1 intf1
% trace intf1 hex console
```

The QEMU emulator is launched afterwards with the following command line:

```
$ kvm -name <name> -hda <image> -boot c -k en-us -usb -usbdevice tablet \
-net nic,vlan=1,macaddr=01:02:03:04:05:06 \
-net socket,vlan=1,connect=127.0.0.1:5001
```

Now you should see all the Ethernet frames, sent by QEMU's guest OS to the *vnd*, being printed to the *vnd* console. You must wait for the traffic to stop in order to switch off the trace mode.

6. Overview of commands

Entering `help` at the prompt gives the *vnd* version and the list of available commands:

```
add <if> {nic|wic|ral} {raw|eth|p11}
rem {<if>|all}
{up|down} {<if>|all} {in|out|all}
{set|unset} {<if>|all} {in|out|all}
      {vli <id>|bw <bps>|dl <s>|dv <%>|ber <bep>|qs <maxpkt>}
{stat|unstat} {<if>|all} {in|out|all}
{trace|untrace} {<if>|all} {txt|hex|pcap} {console|<trace_file>}
inj {<if>|all} {in|out|all} {txt|hex} <data>
{tie|untie} <if1> <if2>
incn <ep> {raw|len} {tcp|udp} {<laddr>|*} {<lport>|*} <raddr> <rport>
insr <ep> {raw|len} {tcp|udp} {<laddr>|*} {<lport>|*} {<raddr>|?} {<rport>|?}
{uncn|unsr} <ep> {raw|len} {stm|dgm} <socket_pathname>
tap <ep> <tap_ifname>
vde <ep> <switch_pathname> <port> <group> <mode>
disc {<ep>|all}
{par|unpar} {<ep>|all} {in|out|all} bs <buffer_size>
{bind|unbind} <ep> <if>
```

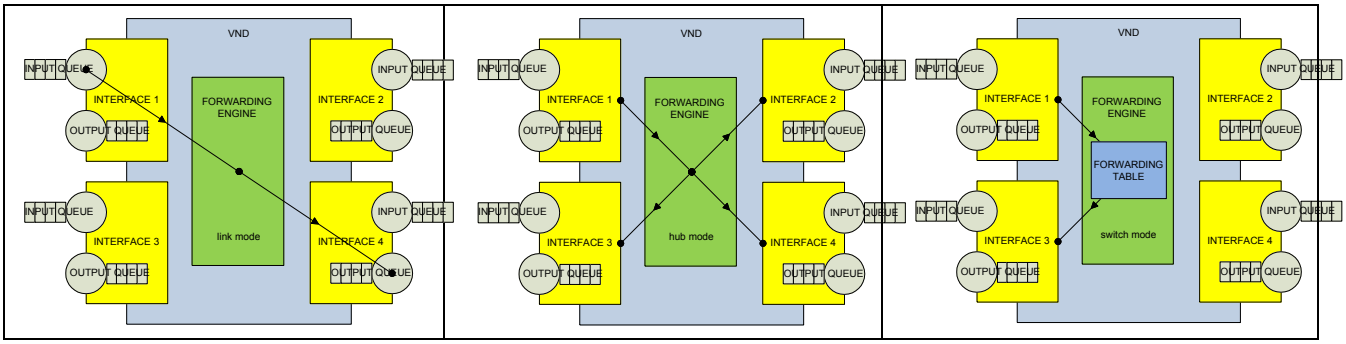
```
snd <ep> {txt|hex} <data>
show {if|ep|lk|fw|op}
clear {lk|fw}
load <conf_file>
dump {if|ep|lk|fw|st} <dump_file>
mode {link|hub|switch|access-point|mob-infra|mob-adhoc}
debug {on|off}
name <device_name>
exit
```

7. Modes

```
mode {link|hub|switch|access-point|mob-infra|mob-adhoc}
```

This command defines the mode in which the *vnd* operates:

- `link` : each interface is directly wired to another interface, which means that any data going into the input of the first interface is forwarded to the output of the second interface in this given direction (i.e., it is one way).
- `hub` : each interface is bound to all others, which means that any data going into the input of an interface is forwarded to the output of all the other interfaces except itself .
- `switch` : any frame going into the input of an interface is forwarded to the switch engine which uses a forwarding table to determine the output interface leading to the device having the same address as the frame's destination address.



- `access-point` : acts as a switch for `nic` interfaces and as a hub for `ral` interfaces.
- `mobile-infra` : acts as a pseudo WIC in infrastructure mode (BSS) for emulating a wireless node (as no WIC is emulated by QEMU yet), one `wic` interface is bound to all `ral`, which means that any data going into the input of a `wic` interface is forwarded to the output of all the other `ral` interfaces. In reverse any data going into the input of a `ral` interface is sent to the `wic` interface. A pseudo 802.11 header is used for `ral` to `ral` connections.
- `mobile-adhoc` : same as above in ad hoc mode.

The last three modes are used for the emulation of wireless and mobile devices. In this case, NEmu is used for managing the virtualized mobile network (http://nemu.valab.net/). NEmu uses *vnd* for emulating devices and *nemo* as the scheduling engine (http://www.labri.fr/perso/magoni/nemo/). Please refer to the NEmu website for all the necessary information.

8. Interfaces

`add <if> {nic|wic|ral} {raw|eth|p11}`

This command adds a new interface having the unique name/identifier <if> which can be defined as a network interface card (`nic`), a wireless interface card (`wic`) or a radio link (`ral`, which is used to emulate radio broadcasting). The protocol used by the data flowing through the interface is defined by the last parameter. The currently supported protocols are:

- `raw`: the data is a flow of bytes that will not be interpreted by the *vnd*
- `eth`: the data is interpreted as Ethernet II frames
- `p11`: the data is interpreted as pseudo-IEEE 802.11 frames

Important note: the `wic` and `ral` interfaces are only useful when using the *vnd* in the access point or mobile-* modes, which is only useful if you use the network mobilizer (*nemo*) scheduling software.

`rem {<if>|all}`

This command removes an existing interface having the unique name/identifier <if> or all interfaces if the argument `all` is given.

`{up|down} {<if>|all} {in|out|all}`

This command starts or stops an existing interface having the unique name/identifier <if> or all interfaces if the argument `all` is given. The command can be applied to the input queue with the argument `in`, the output queue with the argument `out` or both with the argument `all`.

`{set|unset} {<if>|all} {in|out|all} {vli <id>|bw <bps>|dl <s>|dv <%>|ber <bep>|qs <maxfr>}`

This command sets or unsets the parameters of an existing interface having the unique name/identifier <if> or all interfaces if the argument `all` is given. The command can be applied to the input queue with the argument `in`, the output queue with the argument `out` or both with the argument `all`. The command can be applied to set a port-based VLAN ID with the argument `vli` (specified as a number between 0 and 128, 0 being the default), to set the bandwidth with the argument `bw` (specified in bits/s), to set the delay with the argument `dl` (specified in s), to set the delay variation/jitter with the argument `dv` (specified in percentage), to set the bit error rate (BER) with the argument `ber` (specified as a probability value between 0 and 1) or to set the queue size (defined as the maximum number of frames stored in the queue) with the argument `qs` (specified without units).

Note: the port-based VLAN IDs only operate in `hub` or `switch` modes. *vnd* does not support the IEEE 802.1Q standard.

`{stat|unstat} {<if>|all} {in|out|all}`

This command provides or resets statistics on an existing interface having the unique name/identifier <if> or all interfaces if the argument `all` is given. The command can be applied to the input queue with the argument `in`, the output queue with the argument `out` or both with the argument `all`. The statistics, calculated since boot or last reset, are:

- Total number of frames
- Total number of bytes
- Number of frames/s

- Number of bytes/s
- Total number of lost frames
- Total number of lost bytes
- Number of lost frames/s
- Number of lost bytes/s

`{trace|untrace} {<if>|all} {txt|hex|pcap} {console|<trace_file>}`

This command activates or deactivates the trace mode on any or all interfaces of the *vnd*. In trace mode, data flowing through the interface can be printed in the console (without any way to get the prompt back unless the data flow stops!) or in a file named <trace_file>. The format can be set to plain text (i.e., ASCII), to hexadecimal notation (with a presentation similar to *Wireshark*) or to the `pcap` standard packet capture file format of *Wireshark*. A trace saved in a file with the `pcap` option can be opened later on by *Wireshark* for inspection.

Important note: tracing network traffic has a big negative impact on the performances of the *vnd*.

`inj {<if>|all} {in|out|all} {txt|hex} <data>`

This command injects data provided either in plain text (e.g., ASCII) or in hexadecimal notation interpreted as binary to the input and/or output queue of the interface named <if>. Or all interfaces if `all` is specified. The <data> may contain space characters that are kept in text and removed in hexadecimal notation.

Note: currently, for technical reasons, traffic policing (i.e., input bandwidth and delay throttling) is not applied to data injected in the input queue of an interface.

`{tie|untie} <if1> <if2>`

This command ties or unties an existing interface having the unique name/identifier <if1> to another existing interface having the unique name/identifier <if2>. Such a tie is stored in the *linking* table. It is a one way internal connection, thus the data coming into the input of <if1> is sent to the output of <if2>. For a two way connection, the user must also issue the command `tie <if2> <if1>`. The same applies for the `untie` command.

9. Endpoints

`incn <ep> {raw|len} {tcp|udp} {<laddr>|*} {<lport>|*} <raddr> <rport>`

This command creates a new endpoint having the unique name/identifier <ep>.
It is created as an Internet IPv4 socket being a *client* type and thus it tries to connect to a remote host being an up and running server.
The third argument defines which layer 4 protocol is used and it can be:
- `tcp` : the TCP protocol is used to communicate with the remote host,
- `udp` : the UDP protocol is used to communicate with the remote host.

The second argument defines the encapsulation mode which can be:
- `raw` : the data is received and sent uninterpreted as a flow of bytes,
- `len` : the data is interpreted as a succession of layer 2 frames (in `udp` each packet contains one frame, in `tcp` each frame is prepended by a four-byte network byte order size field).

The fourth and fifth arguments define the local host's IP address and port number. If one or both arguments are left undefined by the use of the wildcard `*`, the host's default ANY IP address and/or a *random* port number are used.

The sixth and seventh arguments define the remote host's IP address and port number and must be fully detailed.

`insr` <ep> `{raw|len} {tcp|udp} {`<laddr>`|*} {`<lport>`|*} {`<raddr>`|?} {`<rport>`|?}`

This command creates a new endpoint having the unique name/identifier <ep>.
It is created as an Internet IPv4 socket being a *server* type and thus it starts by listening to any incoming connection from a remote host. Only *one* active connection to the server is authorized in an endpoint at any time.
The third argument defines which layer 4 protocol is used and it can be:
- `tcp` : the TCP protocol is used to communicate with the remote host,
- `udp` : the UDP protocol is used to communicate with the remote host.

The second argument defines the encapsulation mode which can be:
- `raw` : data is received and sent uninterpreted as a flow of bytes,
- `len` : data is interpreted as a succession of layer 2 frames (in `udp` each packet contains one frame, in `tcp` each frame is prepended by a four-byte network byte order size field).

The fourth and fifth arguments define the local host's IP address and port number. If left undefined by the use of the `*` wildcard, the host's default ANY IP address and a *random* port are used.
The sixth and seventh arguments define the remote host's IP address and port number. If one or both arguments are left undefined by the use of the `?` wildcard, the remote host's IP address and/or port number will be defined upon connection for `tcp` or upon reception of the first incoming packet for `udp`.

Note: if or when the remote IP address and port number are set, any subsequent incoming packet having a different IP address or port number will be discarded.

`{uncn|unsr}` <ep> `{raw|len} {stm|dgm}` <socket_pathname>

This command creates a new endpoint having the unique name/identifier <ep>.
With `uncn`, it is created as an UNIX local socket being a *client* type and thus it tries to connect to a remote server application on the same system.
With `unsr`, it is created as an UNIX local socket being a *server* type and thus it starts by listening to any incoming connection from a remote client application on the same system. Only *one* active connection to the server is authorized in an endpoint at any time.
The third argument defines which layer 4 protocol is used and it can be:
- `stm` : the stream oriented protocol is used to communicate with the remote application,
- `dgm` : the datagram oriented protocol is used to communicate with the remote application.

The second argument defines the encapsulation mode which can be:
- `raw` : the data is received and sent uninterpreted as a flow of bytes,
- `len` : the data is interpreted as a succession of layer 2 frames (in `dgm` each packet contains one frame, in `stm` each frame is prepended by a four-byte network byte order size field).

The fourth argument defines the name of the local socket defined as an absolute path name in the system.

`tap` <ep> <tapifname>

This command creates a new endpoint having the unique name/identifier <ep>.
This command enables the connection of the new endpoint to an existing Linux tap identified by <tap_ifname> (e.g., `tap0`).

Note: the access rights on the tap must have been properly set at the tap creation by the administrator if regular (i.e., non-root) users are expected to use it.

`vde` <ep> <switch> [<port> <group> <mode>]

This command creates a new endpoint having the unique name/identifier <ep>.
This command enables the connection of the new endpoint to an existing VDE switch identified by <switch> (e.g., `/tmp/myswitch`). The other parameters are optional. Please look at the VDE documentation for details if you do not want the default values. The access rights on the switch must have been properly set at the switch creation if regular (i.e., non-root) users shall use it.

`disc {`<ep>`|all}`

This command disconnects and destroys the existing endpoint having the unique name/identifier <ep> or all endpoints if the argument `all` is given.

Note: any endpoint can also be disconnected from its remote host. In this case, if the endpoint is a *client* type, it is destroyed. If the endpoint is a *server* type, it returns to its listening state.

`{par|unpar} {`<ep>`|all} {in|out|all} bs` <buffer_size>

This command sets or unsets the parameters of an existing endpoint having the unique name/identifier <ep> or all endpoints if the argument `all` is given. The command can be applied to the *send* side with the argument `in`, the *receive* side with the argument `out` or both with the argument `all`. For the moment, this command can only be used to modify the buffer size with the argument `bs` (specified in bytes).

Note: although they could be different, the buffer size of endpoints is currently equal to the buffer size of interface queues. Any change to the endpoint in/out buffer size is propagated to the interface queue in/out buffer size (if the interface is bound to the endpoint).

`{bind|unbind}` <ep> <if>

This command binds or unbinds an existing endpoint having the unique name/identifier <ep> to an existing interface having the unique name/identifier <if>.

`snd` <ep> `{txt|hex}` <data>

This command sends data provided either in plain text (e.g., ASCII) or in hexadecimal notation interpreted as binary to the remote side of the endpoint named <ep>. The data <data> may contain space characters that are kept in text and removed in hexadecimal notation.

10. Operations

`clear {lk|fw}`

This command clears the *linking* table with the option `lk` or the *forwarding* table with the option `fw`.

`load` <conf_file>

This command loads a list of proper *vnd* commands stored in a file named <conf_file>. The commands are then executed sequentially in the order in which they appear in the file.

`debug {on|off}`

This command sets the *vnd* debug mode on or off. In debug mode, the *vnd* is more verbose andalso printed in the console (without any way to get the prompt back unless data flow stops!).

`name` <device_name>

This command sets the name of the *vnd*. The name is always shown at the left of the prompt.

`exit`

This command destroys all endpoints, removes all interfaces and terminates the *vnd*.

11. Informations

`show {if|ep|lk|fw|op}`

This command shows/prints information on various objects of the *vnd* to the console, namely:
- `if` : list of the interfaces including the values of their parameters
- `ep` : list of the endpoints including the IP@/port# of the local and remote host
- `lk` : list of the existing ties between interfaces
- `fw` : content of the forwarding table (i.e., list of the MAC address/output interface associations)
- `op` : list of the current values of the global parameters

`dump {if|ep|lk|fw|st}` <dump_file>

This command writes information on various objects of the *vnd* to a file named <dump_file>.
The meaning of the first argument is the same as for the `show` command above except for the last option `st` which stands for statistics.

12. References

More information can be found in the reference below. Please cite it if ever you are using *vnd* in your research or educational work.

Network Emulator: a Network Virtualization Testbed for Overlay Experimentations.
Vincent Autefage, Damien Magoni.
CAMAD'12 - IEEE International Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks,
pp. 38-42, September 17-19, 2012, Barcelona, Spain.

That's all folks!