To understand the Virtual Network feature (vnet), not to be confused with vtnet(4) for VirtIO Ethernet driver, let’s begin with an extract from the vnet(9) man page:

**DESCRIPTION**

vnet is the name of a technique to virtualize the network stack.

(...).

Each (virtual) network stack is attached to a prison, with vnet0 being the unrestricted default network stack of the base system.

As a related prison feature, let’s check the jail(8) man page section about vnet:

vnet    Create the jail with its own virtual network stack, with its own network interfaces, addresses, routing table, etc. The kernel must have been compiled with the VIMAGE option for this to be available. Possible values are "inherit" to use the system network stack, possibly with restricted IP addresses, and "new" to create a new network stack.

To resume, it’s a feature that allows each jail to have its own routing table / ARD & NDP cache and interfaces

**Vocabulary**

- Host: The system hosting your jails

**Examples**

These examples use “empty” jails based on the host ‘/’ to focus only on the vnet feature. They will all be in “persist” mode (because no processes are running).

Concerning the OS requirements:

- The shell used is /bin/sh.
- FreeBSD 12.1 minimum (can be 12-STABLE or even better a -head)
Useless Isolated vnet Jail

This useless example shows how to create an isolated vnet jail.

Command line parameters details:
- `-c`: create a new jail
- `name`: Name of the jail to avoid using its jail ID (JID) later
- `host.hostname`: For this example, only used to have a pretty output with command `jls`
- `persist`: There is no process running on this jail, so force it running
- `vnet`: Enable the virtual network stack

```
jail -c name=useless host.hostname=jvnet persist vnet
jls
```

<table>
<thead>
<tr>
<th>JID</th>
<th>IP Address</th>
<th>Hostname</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>jvnet</td>
<td></td>
<td>/</td>
</tr>
</tbody>
</table>

Here is the default network interface assigned and the content of its routing table for this new jail.

```
jexec useless ifconfig
lo0: flags=8008<LOOPBACK,MULTICAST> metric 0 mtu 16384
    options=680003<RXCSUM,TXCSUM,LINKSTATE,RXCSUM_IPV6,TXCSUM_IPV6>
    groups: lo
    nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>

jexec useless netstat -rn
Routing tables
```

There is an unconfigured loopback (disabled and no IP addresses assigned) and an empty routing table. Let's fix that.

```
jexec useless service netif restart
Stopping Network: lo0.
lo0: flags=8008<LOOPBACK,MULTICAST> metric 0 mtu 16384
    options=680003<RXCSUM,TXCSUM,LINKSTATE,RXCSUM_IPV6,TXCSUM_IPV6>
    groups: lo
    nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
Starting Network: lo0.
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
    options=680003<RXCSUM,TXCSUM,LINKSTATE,RXCSUM_IPV6,TXCSUM_IPV6>
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
    inet 127.0.0.1 netmask 0xff000000
    groups: lo
    nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>

jexec useless netstat -rn
Routing tables
```

Internet:
<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Flags</th>
<th>Netif</th>
<th>Expire</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>link#1</td>
<td>UH</td>
<td>lo0</td>
<td></td>
</tr>
</tbody>
</table>
This is a lot better! But we only have the loopback interface running. The next step is to create a virtual Ethernet tap interface and assign it to the jail. The ifconfig(8) man page extract:

```
vnet jail
Move the interface to the jail(8), specified by name or JID. If the jail has a virtual network stack, the interface will disappear from the current environment and become visible to the jail.
```

Let’s do this:

```
# TAP=$(ifconfig tap create)
# ifconfig $TAP
tap0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
    options=80000<LINKSTATE>
    ether 00:bd:70:98:00:00
    groups: tap
    media: Ethernet autoselect
    status: no carrier
    nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
# ifconfig $TAP vnet useless
# ifconfig $TAP
ifconfig: interface tap0 does not exist
```

What happened? Just after we enabled the interface and assigned it to the jail, it disappeared! That is the expected behavior because this interface doesn’t belong to your host network stack anymore. You can check its status on the jail and even assign an IP address to it:

```
# jexec useless ifconfig $TAP
tap0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
    options=80000<LINKSTATE>
    ether 00:bd:70:98:00:00
    groups: tap
    media: Ethernet autoselect
    status: no carrier
    nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
# jexec useless ifconfig $TAP inet 192.0.2.1/24 up
# jexec useless ifconfig $TAP inet
tap0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
    options=80000<LINKSTATE>
    inet 192.0.2.1 netmask 0xffffff00 broadcast 192.0.2.255
# jexec useless ping -c 2 192.0.2.1
```
PING 192.0.2.1 (192.0.2.1): 56 data bytes
64 bytes from 192.0.2.1: icmp_seq=0 ttl=64 time=0.248 ms
64 bytes from 192.0.2.1: icmp_seq=1 ttl=64 time=0.525 ms

--- 192.0.2.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.248/0.387/0.525/0.139 ms

# jexec useless arp -na
? (192.0.2.1) at 00:bd:70:98:00:00 on tap0 permanent [ethernet]
# arp -na | grep 192.0.2.1
#

The jail can ping its own interface, its own ARP cache is populated with the corresponding entry, and all of these are isolated from the host networking stack.

It's the same for the routing table:

# jexec useless route add -net 198.51.100.0/24 192.0.2.2
add net 198.51.100.0: gateway 192.0.2.2
# jexec useless netstat -4rn
Routing tables

Internet:
Destination   Gateway       Flags Netif Expire
127.0.0.1     link#1       UH    lo0
192.0.2.0/24   link#2       U    tap0
192.0.2.1     link#2       UHS   lo0
198.51.100.0/24 192.0.2.2  UGS   tap0
# netstat -4rn | grep 198.51.100.0
#

Before continuing to the next example, we will clean up the existing jail and destroy the tap interface. We need to use the -R (upper case) option to remove a jail created without a configuration file. Using option -r (lower case), the vnet interface will not be removed to the host automatically.

# jail -R useless
# ifconfig $TAP destroy

**vnet Jail Connected to the Host**

This example shows how to communicate between the jail and the host itself. The main problem is that any interface put into a vnet will disappear from the host networking stack.

So, we could imagine this setup:
1. Create a bridge and assign an IP address to it
2. Create a tap interface and add it into the bridge
3. Assign the tap interface to the jvnet jail

But this will not work. The TAP interface will move from the host (=disappear from the host) to join the jail’s vnet, and so it will disappear from the bridge too!
To solve this problem, the epair(4) interface (Ethernet pair) was created. This special network interface represents two interfaces (epairXa and epairXb) that will behave like two Ethernet interfaces cross-connected between them. By assigning each side to a different vnet, they will still exchange frames between them.

Start by creating a new epair pair.

```bash
# ifconfig epair create
epair0a
# ifconfig -g epair
epair0b
epair0a
```

The host is showing its two new interfaces: epair0a and epair0b.

Create a new jail, named “jvnet” and assign interface epair0b to it.

```bash
# jail -c name=jvnet host.hostname=jvnet persist vnet vnet.interface=epair0b
```

Interface epair0b no longer belongs to the host system network stack, but the other epair0a still does! Let's configure an IP address on epair0a.

```bash
# ifconfig -g epair
epair0a
# ifconfig epair0a inet 192.0.2.1/24 up
```

Then do the same on epair0b belonging to the jail and check their connectivity.

```bash
# jexec jvnet ifconfig epair0b inet 192.0.2.2/24 up
# ping -c 2 192.0.2.2
PING 192.0.2.2 (192.0.2.2): 56 data bytes
64 bytes from 192.0.2.2: icmp_seq=0 ttl=64 time=0.285 ms
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.532 ms

--- 192.0.2.2 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.285/0.408/0.532/0.124 ms
# arp -na | grep 192.0.2.
? (192.0.2.2) at 02:77:29:04:9e:0b on epair0a expires in 1139 seconds [ethernet]
? (192.0.2.1) at 02:77:29:04:9e:0a on epair0a permanent [ethernet]
```
By displaying their MAC address, you will notice that epair are using specific MAC address.

Destroy this jail and the epair pair interfaces before continuing to the next example:

```
# jail -R jvnet
# ifconfig epair0a destroy
```

Now that we’ve got a basic setup working, let’s complexify a little more using multiple jails configured in serial and with routing between them.

**Chained Routed vnet Jails**

In this example we will create four jails to route traffic from the host toward a fifth jail.

```
static route
192.0.2.0/24 → 192.0.2.1
```

```
static route
default → 192.0.2.3
192.0.2.0/31 → 192.0.2.2
```

```
static route
default → 192.0.2.5
192.0.2.0/31 → 192.0.2.2
```

Start to generate the five epairs with a first loop.

```
# for i in $(jot 5); do ifconfig epair create; done
epair0a
epair1a
epair2a
epair3a
epair4a
```

Then generate five jails with epair assigned to them with a mix of loop and manual assignment.

```
# for i in $(jot 4); do jail -c name=hop$i host.hostname=hop$i persist vnet
  vnet.interface=epair$((i-1))b vnet.interface=epair$i;a; done
# jail -c name=hop5 host.hostname=hop5 persist vnet vnet.interface=epair4b
# jls

<table>
<thead>
<tr>
<th>JID</th>
<th>IP Address</th>
<th>Hostname</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>hop1</td>
<td></td>
<td>/</td>
</tr>
<tr>
<td>4</td>
<td>hop2</td>
<td></td>
<td>/</td>
</tr>
</tbody>
</table>
```
Now configure IP addresses, enable routing on some jails, and set the static routes.

```bash
# ifconfig epair0a inet 192.0.2.0/31 up
# jexec hop1 ifconfig epair0b inet 192.0.2.1/31 up
# jexec hop1 ifconfig epair1a inet 192.0.2.2/31 up
# jexec hop2 ifconfig epair1b inet 192.0.2.3/31 up
# jexec hop2 ifconfig epair2a inet 192.0.2.4/31 up
# jexec hop3 ifconfig epair2b inet 192.0.2.5/31 up
# jexec hop3 ifconfig epair3a inet 192.0.2.6/31 up
# jexec hop4 ifconfig epair3b inet 192.0.2.7/31 up
# jexec hop4 ifconfig epair4a inet 192.0.2.8/31 up
# jexec hop5 ifconfig epair4b inet 192.0.2.9/31 up
# for i in $(jot 4); do jexec hop$i sysctl net.inet.ip.forwarding=1; done
test.net.inet.ip.forwarding: 0 -> 1
test.net.inet.ip.forwarding: 0 -> 1
test.net.inet.ip.forwarding: 0 -> 1
# route add 192.0.2.0/24 192.0.2.1
add net 192.0.2.0: gateway 192.0.2.1
# jexec hop1 route add default 192.0.2.3
add net default: gateway 192.0.2.3
# jexec hop2 route add default 192.0.2.5
add net default: gateway 192.0.2.5
# jexec hop2 route add 192.0.2.0/31 192.0.2.2
add net 192.0.2.0: gateway 192.0.2.2
# jexec hop3 route add default 192.0.2.4
add net default: gateway 192.0.2.4
# jexec hop3 route add 192.0.2.8/31 192.0.2.7
add net 192.0.2.8: gateway 192.0.2.7
# jexec hop4 route add default 192.0.2.6
add net default: gateway 192.0.2.6
# jexec hop5 route add default 192.0.2.8
add net default: gateway 192.0.2.8
```

Test your setup by pinging the fifth jail from the host network stack.

```bash
# ping -c 2 192.0.2.9
PING 192.0.2.9 (192.0.2.9): 56 data bytes
64 bytes from 192.0.2.9: icmp_seq=0 ttl=60 time=0.265 ms
64 bytes from 192.0.2.9: icmp_seq=1 ttl=60 time=0.482 ms
--- 192.0.2.9 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.265/0.373/0.482/0.108 ms
```
# traceroute -n 192.0.2.9
traceroute to 192.0.2.9 (192.0.2.9), 64 hops max, 40 byte packets
1 192.0.2.1  0.060 ms  0.243 ms  0.244 ms
2 192.0.2.3  0.180 ms  0.202 ms  0.263 ms
3 192.0.2.5  0.050 ms  0.159 ms  0.205 ms
4 192.0.2.7  0.194 ms  0.197 ms  0.191 ms
5 192.0.2.9  0.261 ms  0.201 ms  0.188 ms

Optionally, add a little more fun to this setup (with a big IPv6 range it should be easy to automate a text-to-traceroute script populating DNS configuration file).

# cat >> /etc/hosts <<EOF
? 192.0.2.1 once.upon.a.time.in.the.middle
? 192.0.2.3 of.winter.when.the.flakes.of
? 192.0.2.5 snow.were.failling.like
? 192.0.2.7 feathers.from.the.sky
? 192.0.2.9 a.queen.sat.at.a.window.sewing
? EOF
# traceroute 192.0.2.9
traceroute to 192.0.2.9 (192.0.2.9), 64 hops max, 40 byte packets
1 once.upon.a.time.in.the.middle (192.0.2.1) 0.237 ms  0.508 ms  0.540 ms
2 of.winter.when.the.flakes.of (192.0.2.3) 0.489 ms  0.361 ms  0.373 ms
3 snow.were.failling.like (192.0.2.5) 0.343 ms  0.337 ms  0.285 ms
4 feathers.from.the.sky (192.0.2.7) 0.255 ms  0.296 ms  0.271 ms
5 a.queen.sat.at.a.window.sewing (192.0.2.9) 0.328 ms  0.271 ms  0.242 ms

Going Further
Connecting Jails with the Outside World
Multiple choice here:
1. With SR-IOV compliant NIC, generate multiple Virtual NIC and assign them to the jails.
2. Virtual-Interface (drivers specific)
3. Using VLAN and assigning VLAN interfaces to each jail, the limitation is one jail per VLAN maximum per Ethernet port.
4. Using if_bridge interface (and it’s possible to mix with VLAN, too) is the easiest setup but there is some performance penalty when using if_bridge.

SR-IOV
This feature, initially designed for virtual machine use, creates multiple Virtual Function (VF = Virtual NIC in our case). And by using the default non-passthrough mode, it will present multiple virtual NIC to the host, each of which could be attached to vnet-jail.

Here is an example using two Chelsios interfaces (cxl0 and cxl1) to create 10 VF for each.

# sysrc iovctl_files="/etc/iovctl.cxl0.conf /etc/iovctl.cxl1.conf"
# sysrc kld_list+=if_cxgbev
# cat > /etc/iovctl.cxl0.conf <<EOF
? PF {
? device : “cxl0”;
? num_vfs : 10;
? EOF
# cat > /etc/iovctl.cxl1.conf <<EOF
# PF {
# device : "cxl1";
# num_vfs : 10;
# }
# EOF
# iovctl -C -f /etc/iovctl.cxl0.conf
# iovctl -C -f /etc/iovctl.cxl1.conf
# kldload if_cxgbev
# tail /var/log/messages
(....) kernel: t5vf18: <Chelsio T540-CR VF> at device 0.41 on pci4
(....) kernel: cxlv18: Ethernet address: 06:44:2e:e5:90:18
(....) kernel: cxlv18: 2 txq, 1 rxq (NIC)
(....) kernel: t5vf19: <Chelsio T540-CR VF> at device 0.45 on pci4
(....) kernel: cxlv19: 1 ports, 2 MSI-X interrupts, 4 eq, 2 iq
(....) kernel: cxlv19: Ethernet address: 06:44:2e:e5:90:19
(....) kernel: cxlv19: 2 txq, 1 rxq (NIC)
(....) kernel: t5vf19: 1 ports, 2 MSI-X interrupts, 4 eq, 2 iq
# ifconfig -l
  cxl0 cxl1 igb0 lo0 cxl0v cxl1v cxl2v cxl3v cxl4v cxl5v cxl6v cxl7v cxl8v cxl9v cxl10v cxl11v cxl12v cxl13v cxl14v cxl15v cxl16v cxl17v cxl18v cxl19v

Now we can keep cxl0 (=physical interface) for the host and all cxlvX interfaces can be assigned to a different jvnet-jail.

Notice that:
1. some NIC (like Intel) need more parameters (allow-promisc, allow-set-mac, mac-anti-spoof) to allow specific usage like CARP with the VF.
2. A FreeBSD 12 with Intel ix(4) drivers couldn’t attach drivers to these VF.

ixv0: <Intel(R) PRO/10GbE Virtual Function Network Driver> at device 0.128 on pci4
ixv0: ...reset_hw() failure: Reset Failed!
ixv0: IFDI_ATTACH_PRE failed 5
device_attach: ixv0 attach returned 5

**Virtual-Interface (driver specific)**
Chelsio driver supports another mode called Virtual Interface; here is an example asking to create four VI per port.

# echo hw.cxgbe.num_vis="4" >> /boot/loader.conf

After a reboot the new interfaces (vcxlIX) will be available and show in dmesg as:
vcx10: <port 0 vi 1> on cxl0
vcx10: Ethernet address: 00:07:43:2e:e5:91
vcx10: netmap queues/slots: TX 2/1023, RX 2/1024
vcx10: 1 txq, 1 rxq (NIC); 2 txq, 2 rxq (netmap)
vcx11: <port 0 vi 2> on cxl0
vcx11: Ethernet address: 00:07:43:2e:e5:92
vcx11: netmap queues/slots: TX 2/1023, RX 2/1024
vcx11: 1 txq, 1 rxq (NIC); 2 txq, 2 rxq (netmap)
(...)

Now you can assign these Virtual interfaces (vxclX) to the vnet-jail.

**VLAN**

Without a NIC supporting SR-IOV or Virtual-Interface features, one other possibility is to create multiple VLAN and assign the VLAN interface to the vnet-jail. The restriction is that VLAN ID are unique per interface, so if two vnet-jails need to be assigned a vlan sub-interface in the same VLAN, you need to use two physical interfaces.

```
# ifconfig igb0.6 create vlan 6 vlandev igb0 up
# ifconfig igb0.7 create vlan 7 vlandev igb0 up
# ifconfig igb1.6 create vlan 6 vlandev igb1 up
```

In this example, new interfaces igb0.6, igb0.7, and igb1.6 are available for the vnet-jails.

**Bridge + epair**

To remove the restriction of unique VLAN ID per physical interface, there is still the classical approach, but with some performance impact using bridge and epair setup.

This small diagram is generated with these commands:

```
# ifconfig bridge create up
bridge0
# for i in $(jot 3); do ifconfig epair$i create up; done
```
epair1a
epair2a
epair3a
# ifconfig bridge0 inet 192.0.2.254/24 addm igb1 addm epair1a addm epair2a \
addm epair3a
# for i in $(jot 3); do jail -c name=jail$i host.hostname=jail$i persist vnet \n\vnet.interface=epair${i}b; jexec jail$i ifconfig epair${i}b inet 192.0.2.\${i}/24 up; jexec jail$i ifconfig epair${i}b inet; done
epair1b: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500 
\options=8<VLAN_MTU>
\inet 192.0.2.1 netmask 0xffffff00 broadcast 192.0.2.255
epair2b: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500 
\options=8<VLAN_MTU>
\inet 192.0.2.2 netmask 0xffffff00 broadcast 192.0.2.255
epair3b: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500 
\options=8<VLAN_MTU>
\inet 192.0.2.3 netmask 0xffffff00 broadcast 192.0.2.255
# ping -c 2 192.0.2.1
PING 192.0.2.1 (192.0.2.1): 56 data bytes
64 bytes from 192.0.2.1: icmp_seq=0 ttl=64 time=0.158 ms
64 bytes from 192.0.2.1: icmp_seq=1 ttl=64 time=0.103 ms

--- 192.0.2.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.103/0.130/0.158/0.027 ms
# ping -c 2 192.0.2.2
PING 192.0.2.2 (192.0.2.2): 56 data bytes
64 bytes from 192.0.2.2: icmp_seq=0 ttl=64 time=0.189 ms
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.104 ms

--- 192.0.2.2 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.104/0.146/0.189/0.042 ms
# ping -c 2 192.0.2.3
PING 192.0.2.3 (192.0.2.3): 56 data bytes
64 bytes from 192.0.2.3: icmp_seq=0 ttl=64 time=0.201 ms
64 bytes from 192.0.2.3: icmp_seq=1 ttl=64 time=0.091 ms

--- 192.0.2.3 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.091/0.146/0.201/0.055 ms

And to clean up this example.

# for i in $(jot 3); do jail -R jail$i; ifconfig epair${i}a destroy; done
# ifconfig bridge0 destroy
Final Exercise
Now you should be able to set up this kind of lab on your own.

**FreeBSD host**

```
bridge2  
epair1a  
Jail1  
epair1b  
Jail2  
epair51b
epair51a

bridge5  
epair5a  
Jail5

bridge3  
epair2a  
Jail3  
epair3a

bridge0  
epair1a
```

How Light Is a Jail Compared to a VM?
Let’s use a small device like PC Engines APU2 (4 core AMD GX-412TC SOC 1Ghz, 4GB of RAM, 16Gb flash). On this device, how many jails executing a real process (bird using OSPF to announce each jail’s loopback) can we start?

```
jail1  
lo1  198.51.100.1/32
epair1b  192.0.2.1/24
epair1a

jail2  
lo1  198.51.100.2/32
epair2b  192.0.2.2/24
epair2a

jail254  
lo1  198.51.100.254/32
epair254b  192.0.2.254/24
epair2a
```
This small shell script is used to start 480 jails with a bird OSPF tuned to use large numbers of neighbors on a shared link:

- MTU increased to 9000, allowing large numbers of neighbors (you can only have about 350 maximum OSPF neighbors with the default 1500 bytes MTU)
- Hello and dead interval increased to reduce multicast storm on the bridge interface

```bash
#!/bin/sh
set -eu
dec2dot () {
    # $1 is a decimal number
    # output is pointed decimal (IP address format)
    printf '%d.%d.%d.%d\n' $(printf "%x
" $1 | sed 's../0x& /g')
}
# Need to increase some network value a little bit
# to avoid "No buffer space available" messages
# maximum number of mbuf clusters allowed
sysctl kern.ipc.mmbclusters=1000000
sysctl net.inet.raw.maxdgram=16384
sysctl net.inet.raw.recvspace=16384
# Start addressing shared LAN at 192.0.2.0 (in decimal to easily increment it)
ipepairbase=3221225984
# start addressing loopbacks at 198.51.100.0
iplobase=3325256704
ifconfig bridge create name vnetdemobridge mtu 9000 up
for i in $(jot 480); do
    ifconfig epair$i create mtu 9000 up
    ifconfig vnetdemobridge addm epair$i a edge epair$i a
    jail -c name=jail$i host.hostname=jail$i persist \
        vnet vnet.interface=epair$i b
    ipdot=$( dec2dot $(( iprobase + i)) )
    jexec jail$i ifconfig lo1 create inet $ipdot/32 up
    ipdot=$( dec2dot $(( ipepairbase + i)) )
    jexec jail$i ifconfig epair$i b inet $ipdot/20 mtu 9000 up
    cat > /tmp/bird.${i}.conf <<EOF
    protocol device {}
    protocol kernel { ipv4 { export all; }; };
    protocol ospf {
        area 0 {
            interface "epair$i b" {
                hello 60;
                dead 240;
            }
            interface "lo1" {
                stub yes;
            }
        }
    }
EOF
```
Install net/bird2, execute this script, and after the OSPF DR/BDR election and database synchronization, network traffic on the bridge interface should still be quite high with only the OSPF keep-alives on the bridge interface:

```
# netstat -ihw 1 -I vnetdemobridge

input vnetdemobridge          output
packets  errs idrops      bytes    packets  errs      bytes colls
490     0     0       826K        29k     0       7.0M     0
981     0     0       1.7M        65k     0        22M     0
1.5k     0     0       3.3M        92k     0        25M     0
596     0     0       337K       102k     0        35M     0
732     0     0       479K       100k     0        33M     0
```

After a few minutes, check number of neighbors detected (should be 479). DR/BDR election should have chosen jail479 as BDR and jail480 as DR and number of learned routes.

```
# birdcl -s /tmp/bird.1.ctl show ospf
BIRD 2.0.6 ready.
ospf1:
RFC1583 compatibility: disabled
Stub router: No
RT scheduler tick: 1
Number of areas: 1
Number of LSAs in DB:   481
  Area: 0.0.0.0 (0) [BACKBONE]
    Stub:   No
    NSSA:   No
    Transit:        No
    Number of interfaces:   2
    Number of neighbors:    479
    Number of adjacent neighbors:   2
```

The current system limit of this test is due to 4GB of RAM consumed by all the bird processes.
<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>THR</th>
<th>PRI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>C</th>
<th>TIME</th>
<th>WCPU</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>13529</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>32M</td>
<td>20M</td>
<td>select</td>
<td>1</td>
<td>0:45</td>
<td>2.04%</td>
<td>bird</td>
</tr>
<tr>
<td>13553</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>29M</td>
<td>17M</td>
<td>select</td>
<td>3</td>
<td>0:41</td>
<td>0.71%</td>
<td>bird</td>
</tr>
<tr>
<td>16459</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>14M</td>
<td>3568K</td>
<td>CPU1</td>
<td>1</td>
<td>0:00</td>
<td>0.62%</td>
<td>top</td>
</tr>
<tr>
<td>13512</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>20M</td>
<td>7372K</td>
<td>select</td>
<td>2</td>
<td>0:03</td>
<td>0.39%</td>
<td>bird</td>
</tr>
<tr>
<td>8003</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>20M</td>
<td>7316K</td>
<td>select</td>
<td>0</td>
<td>0:03</td>
<td>0.38%</td>
<td>bird</td>
</tr>
<tr>
<td>7913</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>20M</td>
<td>7260K</td>
<td>select</td>
<td>0</td>
<td>0:03</td>
<td>0.38%</td>
<td>bird</td>
</tr>
<tr>
<td>13466</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>20M</td>
<td>7172K</td>
<td>select</td>
<td>0</td>
<td>0:03</td>
<td>0.34%</td>
<td>bird</td>
</tr>
<tr>
<td>7887</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>20M</td>
<td>7288K</td>
<td>select</td>
<td>1</td>
<td>0:03</td>
<td>0.33%</td>
<td>bird</td>
</tr>
<tr>
<td>7832</td>
<td>root</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>20M</td>
<td>7260K</td>
<td>select</td>
<td>2</td>
<td>0:03</td>
<td>0.33%</td>
<td>bird</td>
</tr>
</tbody>
</table>

Here is the script to delete/clean up all the jails, but you should reboot because your system will panic during this cleanup:

```bash
#!/bin/sh
set -eu
for i in $(jot 480); do
    echo Deleting jail$i
    jail -R jail$i
    ifconfig epair${i}a destroy
    rm /tmp/bird.${i}.*
done
ifconfig vnetdemobridge destroy
```

**Firewalls**

pf and ipfw are vnet compliant, which allows the building of a multi-tenant firewall in an HA scenario like the one presented here:
This setup is more complex because it needs to enable specific kernel features to be useable as a jail. (A detailed example of such setup could be presented in a future article.) Meanwhile, this setup is explained as a BSD Router Project example here (using some helper scripts to configure a jail that hides complexity):
https://bsdrp.net/documentation/examples/multi-tenant_ha_pf_firewalls

OLIVIER COCHARD-LABBÉ discovered FreeBSD in 2005 by customizing m0n0wall to create FreeNAS. As a network engineer, he created the BSD Router Project in 2009 and since then has been endeavoring to bench the FreeBSD network stack. He received his port commit bit in 2016 and is currently a software developer in tests at Netflix.