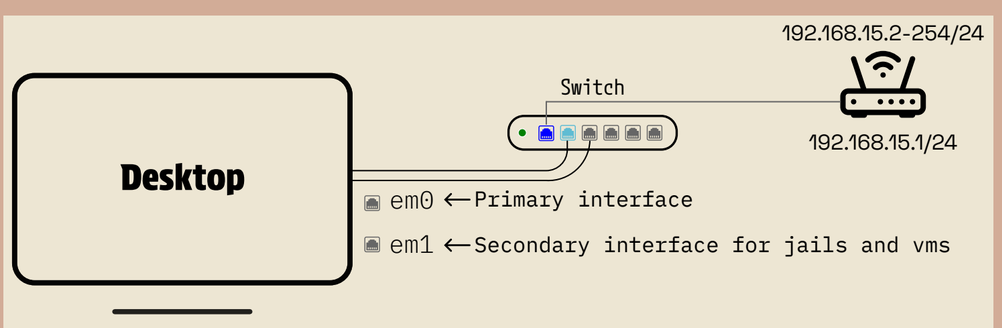
Fiddling with FreeBSD - Netgraph Part-II

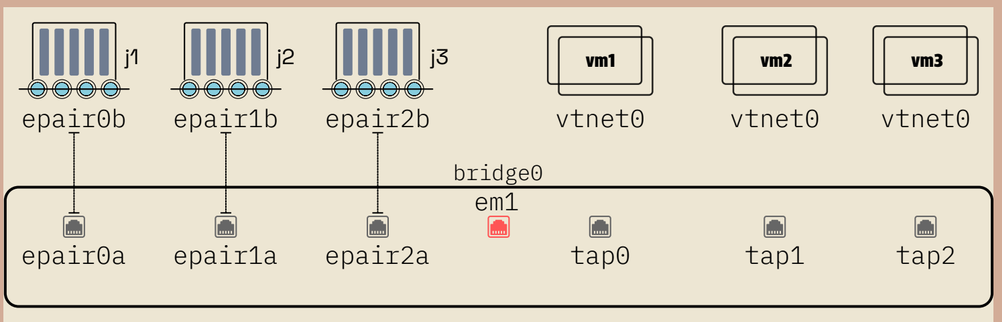
Jail and bhyve vm setup with Netgraph

In the previous post, we explored what a netgraph is, how to create a bridge, and connect it to an Ethernet interface. We also looked at how to create an ng_eiface(4) and attach it to link2 of bridge0. However, I feel that further explanation is needed related to jail setup. Let's dive into that.

Current Network Setup - Example

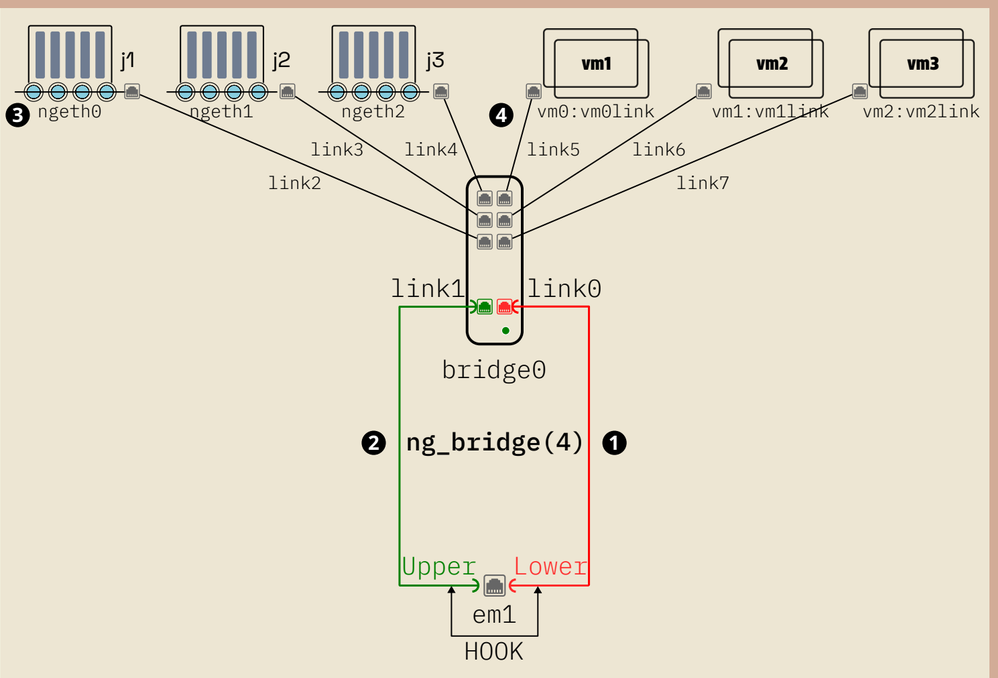


Standard bridge example setup for jails and vms




```
$ ifconfig
em0: ...
        inet xxx.xx.xx.x ...
        ...
em1: ...
        ...
...
bridge0: ...
        member: em1 ...
        member: epair0a ...
        member: epair1a ...
        member: epair2a ...
        member: tap0 ...
        member: tap1 ...
        member: tap2 ...
epair0a: ...
epair1a: ...
epair2a: ...
tap0 ...
tap1 ...
tap2 ...
```

The setup in the example above may vary depending on the infrastructure the user has and the configuration they want to implement. Now, for testing purposes, we will replicate the same configuration using Netgraph to gain a deeper understanding of it.



The first step is to load the ng_ether kernel module, which will create nodes of the ether type for us, though no hooks will be connected initially.

```
# ngctl ls; kldload -nq ng_ether.ko; kldstat
```

Tips and tricks to remember before you proceed...

You don't have to run these commands individually; you can include all of them in a single startup file and execute it at once. However, keep in mind that bhyve creates sockets for you, so you only need ng_bridge(4). i would suggest practice commands first then use this option.

Let's create file called `jail_start.do` and mention below contents

```
mkpeer em1: bridge lower link0
name em1:lower bridge0
connect em1: bridge0: upper link1
mkpeer eiface ether ether
rmhook ngeth0: ether
connect ngeth0: bridge0: ether link2
```

```
$ doas ngctl read jail_start.do
```

The commands are executed interactively, which is why the rmhook is necessary. When you create and connect the ether hook to a local node (socket), it will throw a "file exists" error without removing the existing hook first.

`ngctl msg em1: setpromisc 1` is necessary for vm interface to work. read `ng_ether(4)` for more details.

Let's Setup...

After executing the last command, you should see **em0** and **em1** nodes of the **ether** type created. However, if you check the hooks column, you will notice that no hooks are connected.

As we know, ng_ether(4) supports three hooks: **lower**, **upper**, and **orphans**. The first step is to connect em1's **lower** hook to the bridge's **link0**. However, we haven't created the bridge yet, which is where the `$ doas ngctl mkpeer` command comes in. Let's review its syntax.

```
$ doas ngctl mkpeer [path] <type> <hook> <peerhook>
```

mkpeer - Create and connect a new node to an existing node

[path] - Every netgraph node is addressable called a **node address** or **path,** addresses are used to send control messages.

→ In our case it would be **em1:**

<type> - type of node we want to create and connect to, this will load it's kernel module dynamically verify that with `$ kldstat`.

→ In our case it would be **bridge**

<hook> - hook of the node what's mentioned in the **[path]**

→ this <hook> is the hook of em1: which we want to connect is **lower** hook

<peerhook> - hook of the node what's mentioned in the **<type>**

→ <peerhook> is the hook of **bridge** in our case is **link0** hook

With that we should now be able to create our bridge and connect hooks

connect these two hooks

```
$ doas ngctl mkpeer em1: bridge lower link0          ❶
```

```
$ doas ngctl list
$ doas ngctl show em1:
```

Let's rename our "<unnamed>" bridge.

Assuming you have ID: **000000b** of the bridge

```
$ doas ngctl name "[00b]:" bridge0
```

OR

```
$ doas ngctl name em1:lower bridge0
$ doas ngctl list
```

Now we are going to connect our ng_ether(4) **upper** hook to ng_bridge(4) **link1**, note we are not creating anything now but connecting and syntax changes.

connect these two hooks

```
$ doas ngctl connect em1: bridge0: upper link1       ❷
```

```
$ doas ngctl show bridge0: OR $ doas ngctl show em1:
$ doas ngctl ls -ln
```

The bridge setup with em1 is complete. Now, let's create another node type called ng_eiface(4) supports single **hook called ether**, which is a generic Ethernet interface, and named as **ngeth0, ngeth1**, and so on.

This can be any name

```
$ doas ngctl mkpeer .: eiface ether ether
```

here "." OR ".:" is local node which is basically is ng_socket(4), supports hooks with arbitrary names. you should see ngeth0 of type eiface created with **ifconfig command.**

Create rest of the interfaces for 2 jails

```
$ for i in $(jot 2); do doas ngctl mkpeer eiface ether ether;done
$ doas ngctl list
```

you should now have ngeth0, ngeth1 and ngeth2, check with ifconfig command.

let's connect our ngeth0 ether hook to bridge0 link2

```
$ doas ngctl connect ngeth0: bridge0: ether link2   ❸
$ doas ngctl connect ngeth1: bridge0: ether link3
$ doas ngctl connect ngeth2: bridge0: ether link4
$ doas ngctl msg em1: setpromisc 1        ⟵ Read ng_ether(4) man page
```

Let's create our jails

```
$ for i in $(jot - 0 2)
do doas jail -c name=j$i host.hostname=j$i.home.arpa vnet
vnet.interface=ngeth$i persist
done
```

Assign ip address to our jails

```
$ for i in $(jot - 0 2)
do doas ifconfig -j j$i ngeth$i xxx.xx.xx.x$i/xx
done
```

Bring up our em1 interface

```
$ doas ifconfig em1 up
```

Try pinging from host, you should now have working jail setup with netgraph.

Let's Setup bhyve vm to use Netgraph, we will create 3 vms... ❹

```
# sh /usr/share/examples/bhyve/vmrun.sh -c 1 -m 1024M \
> -t netgraph,socket=vm0,path=bridge0:,hook=vm0link,peerhook=link5 \
> -d disk0.img \
> -i -I FreeBSD-14.2-RELEASE-amd64-disc1.iso vm0
```

```
# sh /usr/share/examples/bhyve/vmrun.sh -c 1 -m 1024M \
> -t netgraph,socket=vm1,path=bridge0:,hook=vm1link,peerhook=link6 \
> -d disk0.img \
> -i -I FreeBSD-14.2-RELEASE-amd64-disc1.iso vm1
```

```
# sh /usr/share/examples/bhyve/vmrun.sh -c 1 -m 1024M \
> -t netgraph,socket=vm2,path=bridge0:,hook=vm2link,peerhook=link7 \
> -d disk0.img \
> -i -I FreeBSD-14.2-RELEASE-amd64-disc1.iso vm2
```