OpenStack環境で、FreeBSD Jail + VIMAGE を使った疑似インターネット実験環境の構築

株式会社インターネットイニシアティブ
山本 茂
shigeru@iij.ad.jp

自己紹介

- FreeBSDは 2.1.5から
  - BSD自体は大学で4.3BSDにふれたのが最初
- 基本的に current しか使っていない
  - もちろん業務用PCもcurrent
  - よって release 版のことはよくわからない☺
- Linuxはよくわからない
  - 必要迫られるとmanとgoogleで調べてなんとかする
- 今はFreeBSD/RaspberryPiで遊んでます
  - http://freebsd-current.os-hackers.jp/pub/FreeBSD/
  - @BsdHacker

概要

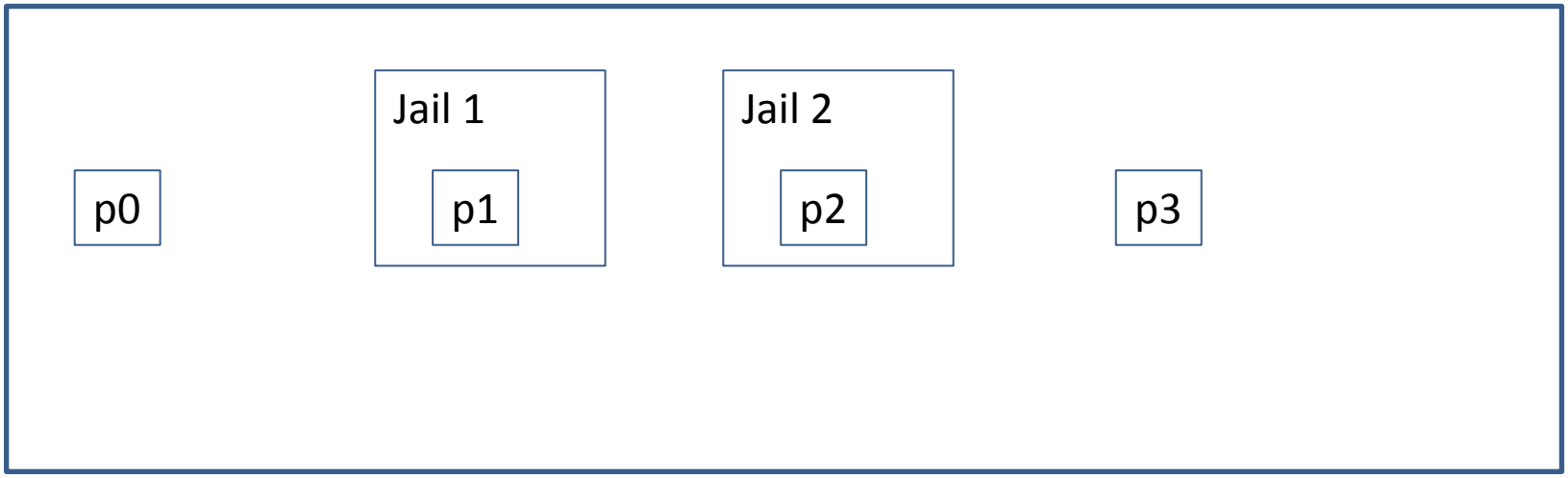
- FreBSD の Jail と VIMAGE の紹介
  - Jailの概要
  - Jailの遊び方
- OpenStack環境で疑似インターネット環境を作って実験してみた
  - なぜ軽量仮想化を使ったのか
  - OpenStackで使ってわかったTips

FREBSD の JAIL と VIMAGE の紹介

FreeBSDのJail(1)

- chroot(8)の発展系
- root directoryを変更する
  - ファイルシステムに対するアクセスを制限できる
- プロセスの実行環境を分離
  - プロセスの実行をJailの中だけに制限できる
  - Jailで実行中のプロセスはJail外のプロセスに対して影響を与えられない
- Linux emulator機能を利用すると、linux環境を作ることができる
  - ただし、現在対応しているのは32bit環境のみ

FreeBSDのJail(2)

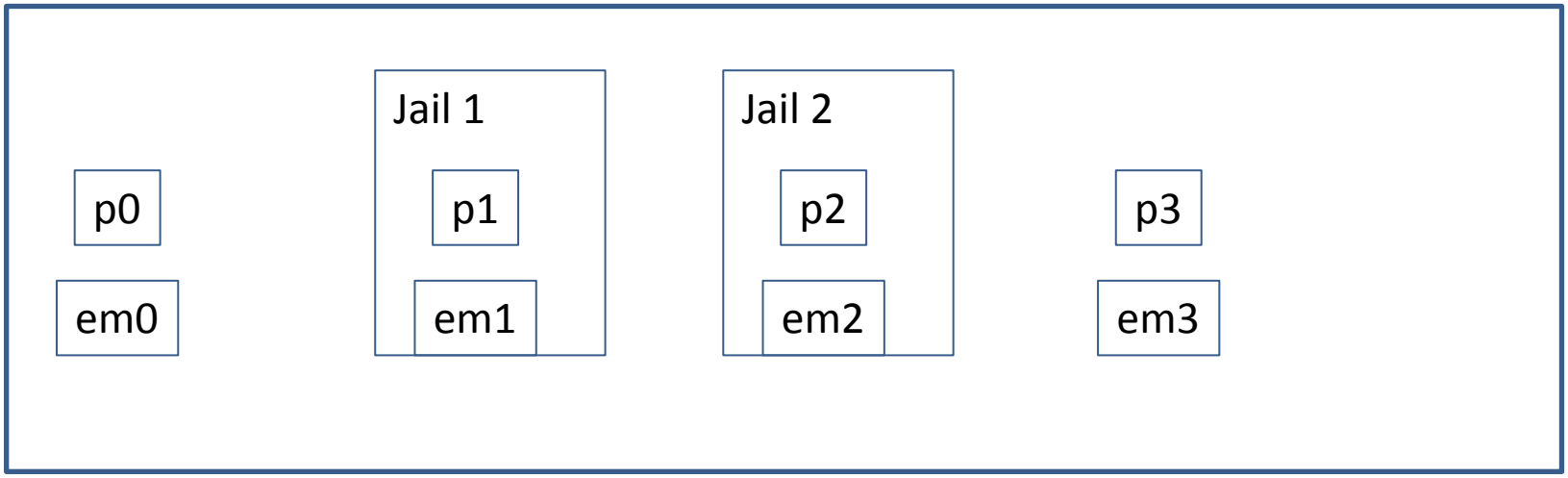


- Jail外ではすべてのプロセスが見える
- jail 1 では、p1 だけが見える
- Jail 2 では、p2 だけが見える

VIMAGE(1)

- Jail毎に異なるネットワークスタックを作成できるようにするための拡張
- FreeBSD 9.0 Releaseから普通に使える
  - ただし、kernelのre-compileは必要
    - オプションを付けずにコンパイルしたものは使えない
      - 混ぜるな危険
  - まだ対応できていないドライバ/スタックもある
- ネットワークインタフェースを自由にJailに配置できる
  - Jailに配置されたネットワークはJail専用になる

VIMAGE(2)



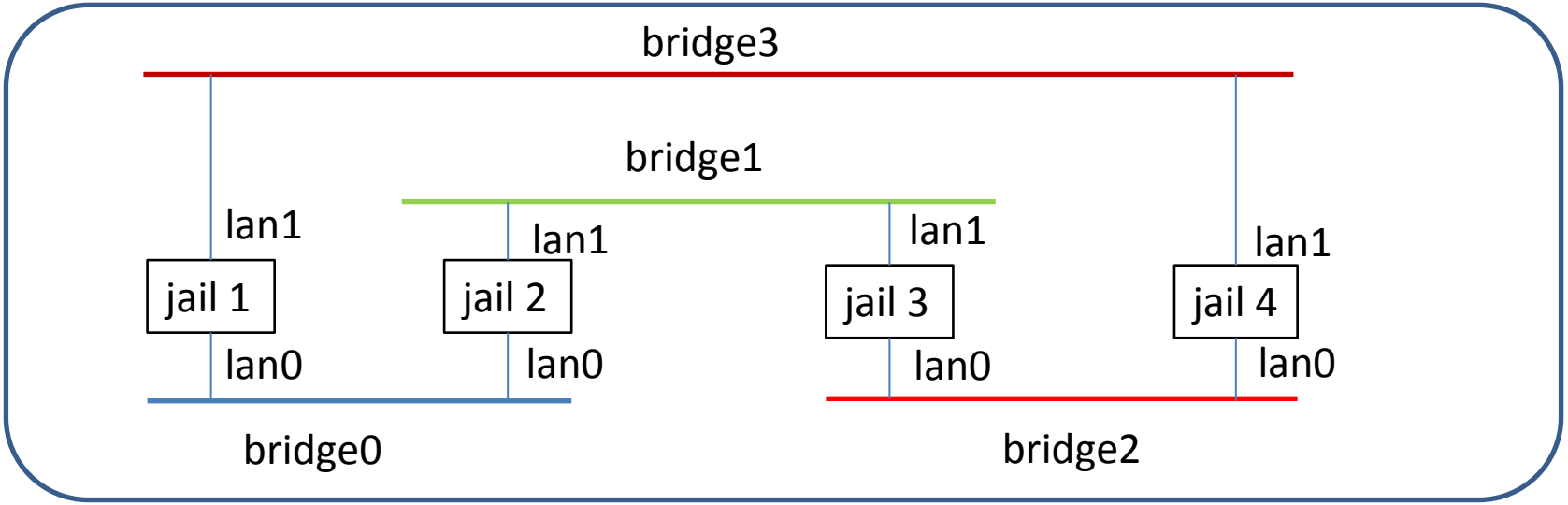
- Jail外では、em0とem3が見える
- Jail 1 では、em1 だけが見える
- Jail 2 では、em2 だけが見える

chroot/jail/VIMAGEのまとめ

	chroot	jail	VIMAGE
ファイルシステム	分離	分離	分離
プロセス	共通	分離	分離
ネットワークスタック	共通	共通	分離

Jail + VIMAGE で遊んでみよう

- jailを四つ作る
- それぞれのjailにはネットワークインタフェースを二つ作る
- ループ状に接続



材料

- FreeBSD 9.1 Relase
  - “optin VIMAGE”をつけてkernelを再構築
- jail(8), jls(8), jexec(8)
- epair(4)
- if_bridge(4)
- Ifconfig(8)
  - Nameオプション
- tmpfs(5)
- nullfs(5)

作り方

1. VIMAGE付きのFreeBSD環境を用意する
2. Jail用のファイルシステムを準備する
3. “vnet”オプション付きでjailを作成
4. 作ったjailにインタフェースを作成
5. 各jailのインタフェースを接続
6. 各jailのネットワーク設定を行う

VIMAGE付きのFreeBSD環境を用意

1. FreeBSD 9.1 Release をインストールする
2. source code を取得する
   - インストール時にsrcを入れておくと楽
3. “option VIMAGE”を追加してkernelを再構築する

```
# cd /usr/src/sys/amd64/conf
# cat > VIMAGE <<EOM
include GENERIC
ident VIMAGE
option VIMAGE
EOM
# cd /usr/src
# make buildworld
# make buildkernel KERNCONF=VIMAGE
# make installkernel KERNCONF=VIMAGE
```

Jail用のファイルシステムを準備する

- chroot と同じように各Jail用のrootを用意する
  - “cp –r”, tar, dump なのでファイルをすべてコピーする
  - ZFSを使ってcloning
  - nullfsを使って必要なディレクトをmount

Jail用のファイルシステムの作成

```
# mkdir -p /jail
# mount -t tmpfs tmpfs /jail
# sysctl security.jail.mount_devfs_allowed=1
# sysctl security.jail.mount_procfs_allowed=1
# for i in 1 2 3 4
do
  mkdir  -p  /jail/jail${i}
  mkdir  -p  /jail/jail${i}/dev
  mount  -t  devfs  devfs   /jail/jail${i}/dev
  mkdir  -p   /jail/jail${i}/proc
  mount  -t procfs  proc   /jail/jail${i}/proc
  for d in  lib  libexec  etc  bin  sbin  usr  var
  do
    mkdir  -p  /jail/jail${i}/${d}
    mount  -t  nullfs   /${d}   /jail/jail${i}/${d}
  done
done
```

“vnet”オプション付きでjailを作成

指定オプション	目的
-c	Jailの作成
persist	恒常的なJail環境の作成
vnet	VIMAGEを利用する
jid	JailのIDを指定する
path	Jailのroot filesystemを指定する
host.hostname	Hostnameを設定する

```
# for i in  1  2  3  4
do
  jail  -c  persist  vnet  jid=${i}  path=/jail/jail${i}  host.hostname=jail-${i}
done
# jls
   JID  IP Address      Hostname                      Path
     1  -                jail-1                        /jail/jail1
     2  -                jail-2                        /jail/jail2
     3  -                jail-3                        /jail/jail3
     4  -                jail-4                        /jail/jail4
```

作ったjailにインタフェースを作成

- epairインタフェースを作成する
  - A/Bの二つの口をもつ仮想インタフェース
    - イメージはEthernetケーブル
  - AからB、BからAにパケットが送信される

```
# kldload if_epair
# for i in  1 2 3 4
do
 for n in  0 1
 do
  ifconfig  epair${i}${n}  create
  ifconfig  epair${i}${n}b  vnet  ${i}
  jexec  ${i} ifconfig  epair${i}${n}b  name  lan${n}
  jexec  ${i} ifconfig  lan${n}  inet6  -ifdisabled  accept_rtadv up
  ifconfig  epair${i}${n}a  up
 done
done
```

各jailのインタフェースを接続

- 仮想ブリッジインタフェース(if_bridge)を作成し、b側のepairインタフェースを接続していく
  - 複数のインタフェースをL2で接続するための仮想インタフェース
    - イメージとしては仮想HUB

```
# kldload   if_bridge
# ifconfig   bridge0   create   up
# ifconfig   bridge0   addm   epair40a   addm   epair10a
# ifconfig   bridge1   create   up
# ifconfig   bridge1   addm   epair11a   addm   epair21a
# ifconfig   bridge2   create   up
# ifconfig   bridge2   addm   epair20a   addm   epair30a
# ifconfig   bridge3   create   up
# ifconfig   bridge3   addm   epair31a   addm   epair41a
```

各jailのネットワーク設定を行う

- 後はjexecコマンドを使って操作する以外は、通常のネットワーク設定作業と同じ

```
# jexec  1  ifconfig  lan0  inet 192.168.4.1/24
# jexec  1  ifconfig  lan1  inet 192.168.1.1/24
# jexec  2  ifconfig  lan1  inet 192.168.1.2/24
# jexec  2  ifconfig  lan0  inet 192.168.2.2/24
# jexec  3  ifconfig  lan0  inet 192.168.2.3/24
# jexec  3  ifconfig  lan1  inet 192.168.3.3/24
# jexec  4  ifconfig  lan1  inet 192.168.3.4/24
# jexec  4  ifconfig  lan0  inet 192.168.4.4/24
# for i in  1 2 3 4
do
 jexec  ${i}  sysctl net.inet.ip.forwarding=1
 jexec  ${i}  sysctl  net.inet6.ip6.forwarding=1
 jexec  ${i}  route  add  default  192.168.${i}.$((${i}%4)+1))
done
```

Tips

- epairをEthernetケーブル、bridgeをHUBとみなすことで、ケーブルのつなぎ変えを再現できる
- Jail側のインタフェースはどのJailでも同じ構成にしておくと設計が楽
- できるだけ計算で決められるようなJailのホスト名、インタフェース名、アドレス設定、ネットワーク構成にしておくことで、大規模なネットワークを構成しやすい

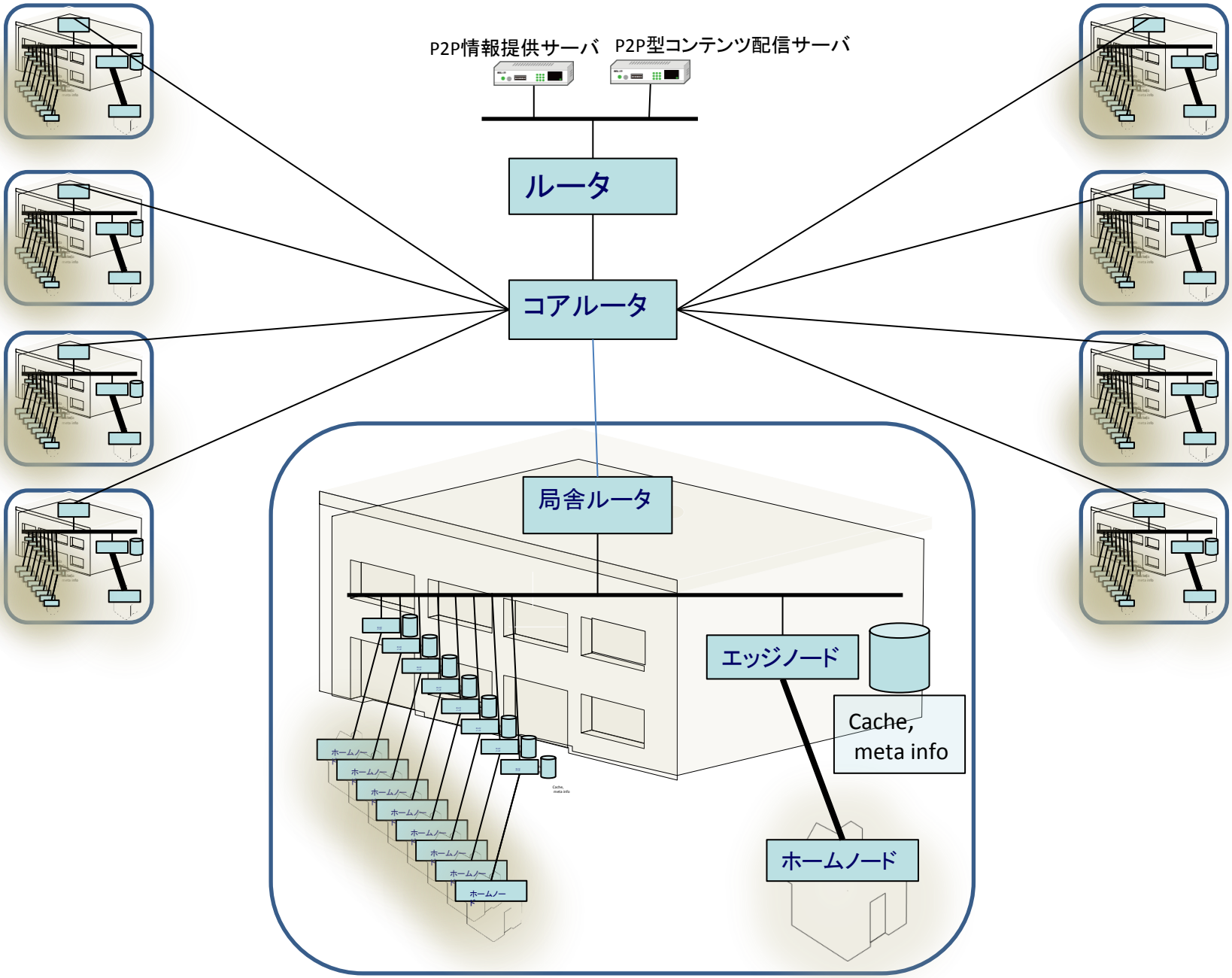
参考資料

- http://people.allbsd.org/~hrs/FreeBSD/sato-FBSD20120608.pdf
  - たぶん、こちらの方がわかりやすいです

OpenStack環境で疑似インターネット環境を作って実験してみた

OpenStackでFreeBSD Jail/VIMAGEを使った疑似インターネット環境の構築

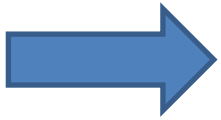
- NICTの委託研究にかかわっていた
  - 大規模コンテンツ配信基盤を実現するアクセス網のクラウド化
  - 大規模コンテンツ配信基盤を実現するアクセス網のクラウド化


P2P情報提供サーバ
P2P型コンテンツ配信サーバ
ルータ
コアルータ
局舎ルータ
エッジノード
Cache,
meta info
ホームノード

実験ネットワークに対する要求

- エンドノードをいっぱい作りたい
- ルータもいっぱい作りたい
- ネットワーク構成も自由に変更したい
- エンドノードやルータの実装に手を入れたい

物理的な機器を使うにはお金が…



なら仮想化環境を使おう

一般的な仮想化環境を考えてみる

- エンドノード一つにVM一つ
- ルーター一つにVM一つ
- VMのメモリ2Gとすると、Host側のメモリが32Gだと、16VM/1Hostぐらいか?
- エンドノードを100ノードぐらい作りたい場合、7Hostぐらい必要
  - 1000ノードだと63Hostぐらい…
- ネットワーク構成を自由にしたいならVLANが使えるL2 switchが必要

これってStarBet…

そうだ、JailとVIMAGEがあった

- Jailは軽量仮想化なので、いっぱいノードが作成できる
  - Jailを作るだけなら
- VIMAGEを使うとネットワークスタックも別々にできる
- FreeBSDならコードもあるていどわかっている
  - lxc… Linuxはよく知らないんです…

こんな環境で実験してました

- 場所: IIJのコンテナデータセンターの一つ

- 仮想環境: Jail/VIMAGE + KVM + OpenStack

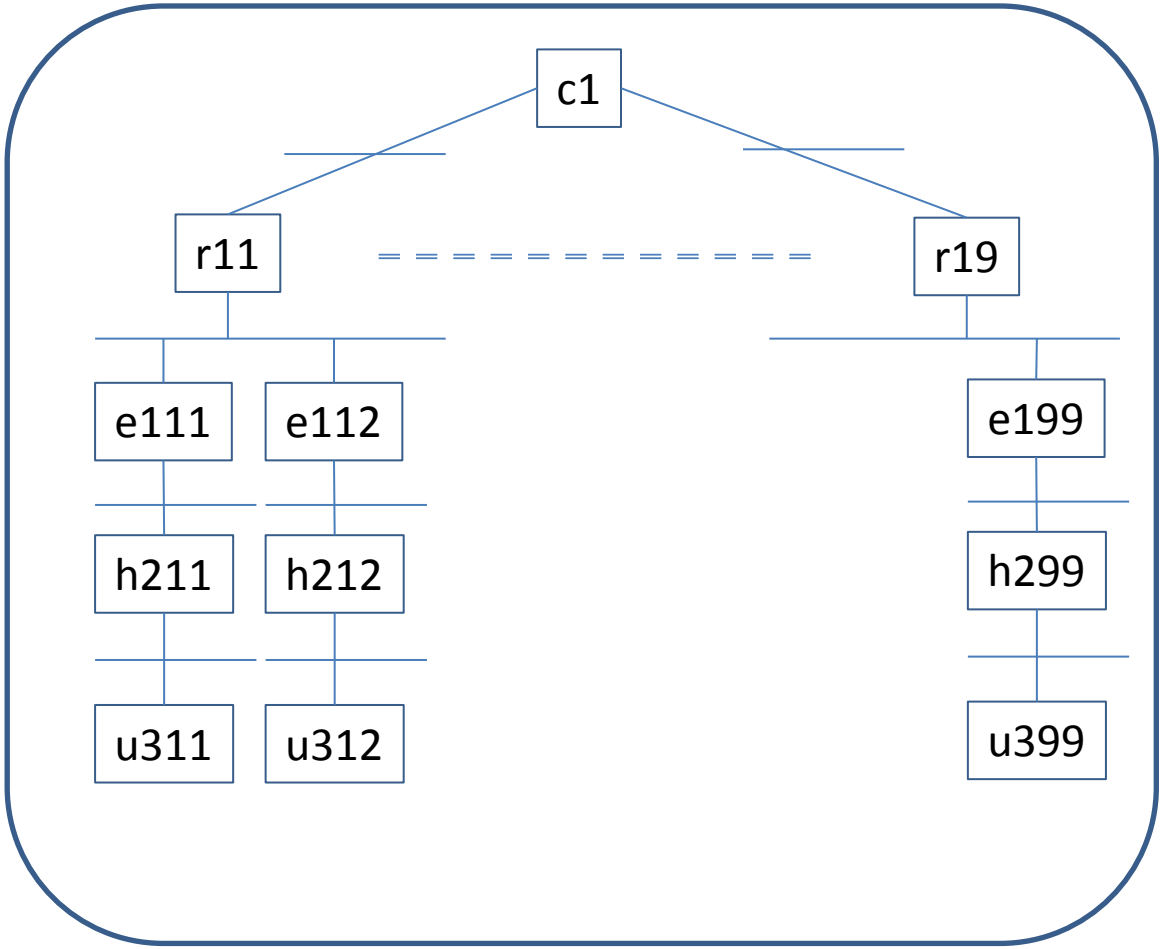
つまり、コンテナ in コンテナ!

JID/インタフェース名

- 計算でJIDやインタフェース名を決められるようにする
  - 数を変えやすくするため
- VMの番号: VID = 1,2,3,…
- コアルータ JID(C) = VID = 1,2,3,…
- 局舎ルータ JID(R) = (JID(C) × 10 + r), r = 1,2,3,…
- ホームノード JID(H) = (JID(R) × 10 + h), h = 1,2,3,…
- エッジノード JID(E) = 100 + (JID(R) × 10 + e), e=1,2,3,…
- ユーザ JID(U) = 200 + (JID(R) × 10 + u), u = 1,2,3,…
- インタフェース番号 = JID × 10 + i, i = 0,1,2,3
  - Jail側はlan0, lan1, lan2, lan3
  - Jail外はepairNb, Nはインタフェース番号

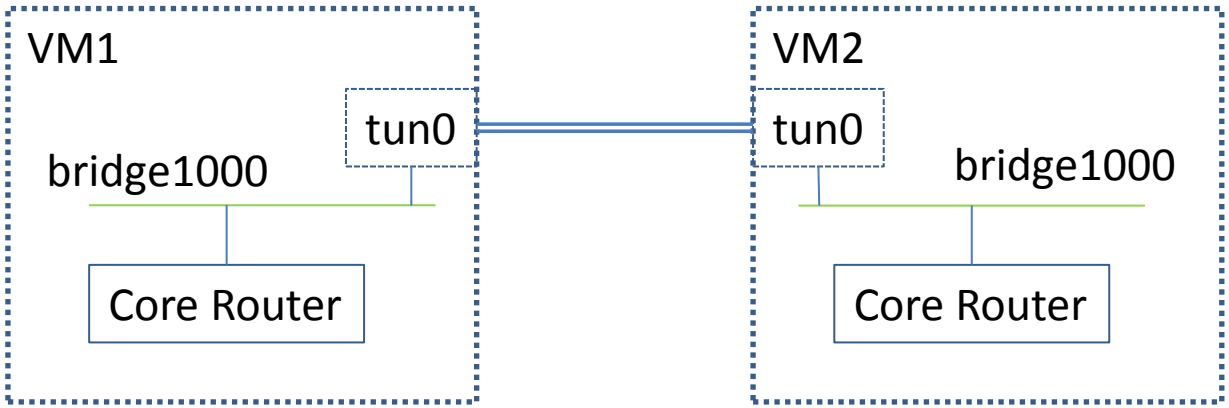
こういうJailになりました

- Core Router: 1
- 局舎ルータ: 9
- 局舎一つあたりのEdge/Home/Userの数: 9組
- 合計: $(9 \times 3) \times 9 + 1 = 243$



さらに数を増やす場合

- 同じ構成のVMを立ち上げる
- tun(4)を使ってVM間の接続回線を作る
  - IPv6アドレスを使ってtunnelを作成
  - 作ったtunnelをbridge/epairを使ってL2接続



OpenStack上で使う場合のTips 1/3

- ディスクイメージは小さい方がいい
  - 必要最小限のイメージを作る
  - 標準以外のソフトはpkgで起動後に入れる
  - GPTを使って、起動時に空き領域を確保する
- 設定などはリモートから取得できるようにしておく
  - VMを削除するとディスクイメージごと消える

OpenStack上で使う場合のTips 2/3

- 使う必要がないメモリは使わない
  - Jail自体よりJailで動くプロセスの方がメモリを使う
  - 不要なプロセスは立ち上げない
  - 経路制御もできればstaticで設定
  - ZFSを使わず、UFSを使う
    - 変更されない部分は、NULLFS
    - 設定ファイルの名前を別々にして、設定ファイルをオプションで指定
    - 必要に応じてUNIONFSの利用も考える

OpenStack上で使う場合のTips 3/3

- HostnameにVMのIDを入れておき、hostnameからVMのIDを取り出す
- 設定は、計算で決まるようにしておく
  - 設定が楽になるようにするため

本当はOpenStack対応を頑張るのが正道なんだろうけど…

まとめと妄想

まとめ

- FreeBSDのJailとVIMAGEはそこそこ使えるようになっている
  - でも、VIMAGEはまだまだ発展途上
- Jail/VIMAGE/epair/bridgeを組み合わせると実験ネットワークの構築が簡単にできる
- Jailは軽量仮想化なので数を稼ぎやすい
- Jailは完全/準仮想環境でも利用できるため、実験環境を固定できる

あったらいいなあ…

- HTML5+JavaScriptでネットワーク図を描くと、それに合わせてJailの設定を作ってくれるもの
  - Jailやインタフェースは動的に確保してくれる
  - 再起動しても自動的に立ち上がるように設定を生成してくれる
- OpenStack対応
  - LXCは対応しているらしいので
- amd64 Linux Emurator環境
  - Jailの中でamd64なLinux環境を作るため必要