

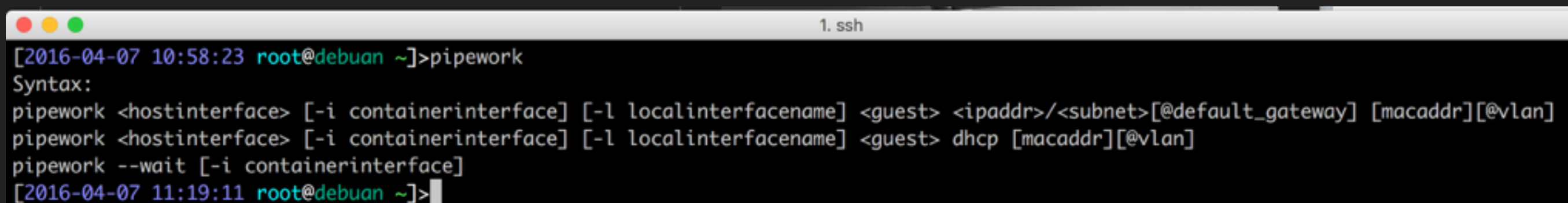
DOCKER NETWORKING

AUTHOR: SYA-KE(TWITTER: @RADICALFUNCTION)

PIPEWORK

PIPEWORKって？

- ▶ dockerコンテナにインターフェースを作ることができるシェルスクリプト
 - ▶ 公式リポジトリ： <https://github.com/jpetazzo/pipework>
 - ▶ eth0とは別のインターフェース（初回はeth1）をdockerコンテナに、それにひもづくインターフェース（brX）をdockerホストに割り振る
 - ▶ 中身は本当にただのシェルスクリプト

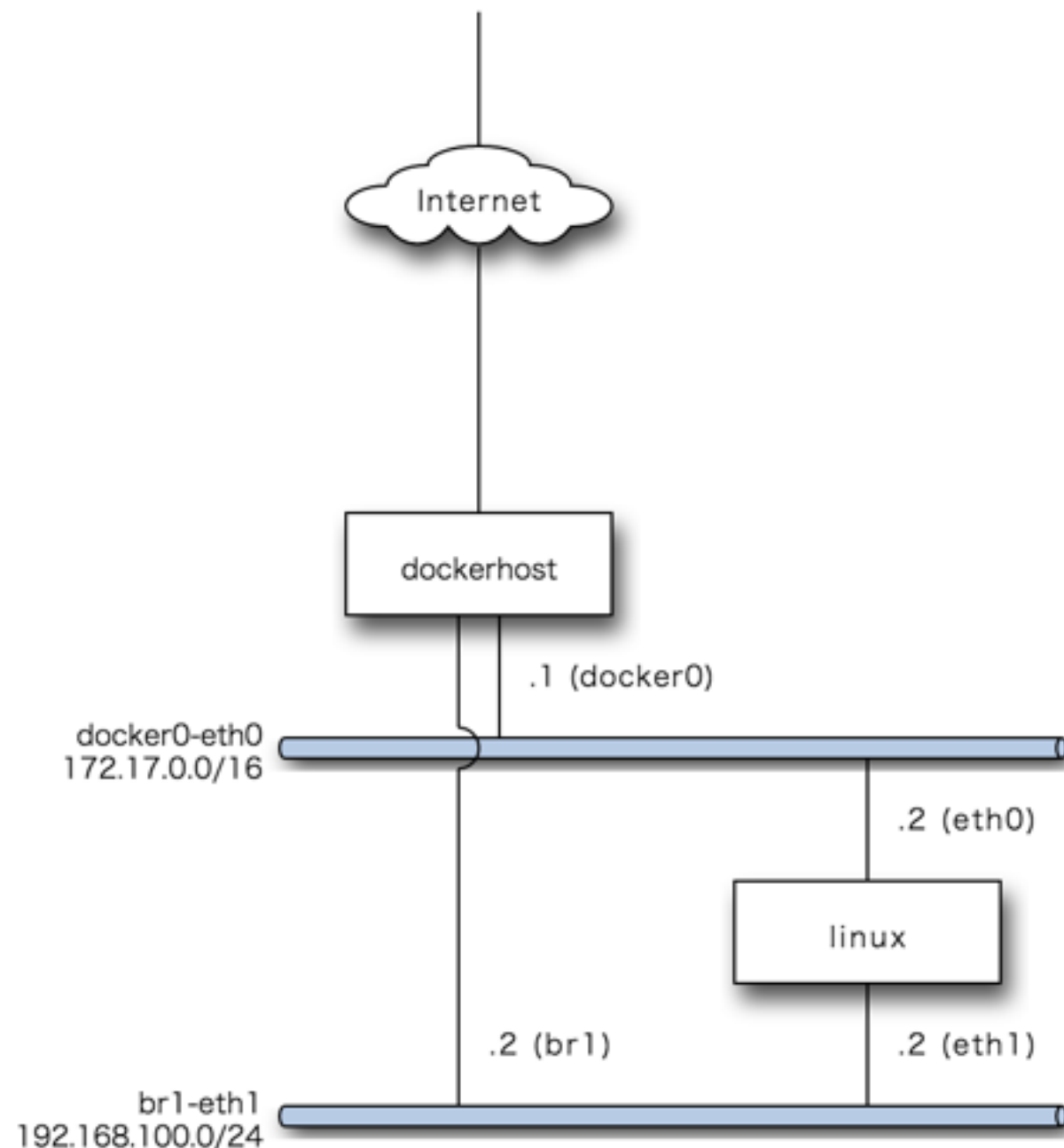
A terminal window titled "1. ssh" showing the execution of the pipework command. The prompt is [2016-04-07 10:58:23 root@debian ~]>. The command pipework is entered, and the syntax is displayed. The syntax shows three main command forms: 1) pipework <hostinterface> [-i containerinterface] [-l localinterfacename] <guest> <ipaddr>/<subnet>[@default_gateway] [macaddr][@vlan], 2) pipework <hostinterface> [-i containerinterface] [-l localinterfacename] <guest> dhcp [macaddr][@vlan], and 3) pipework --wait [-i containerinterface]. The prompt then changes to [2016-04-07 11:19:11 root@debian ~]> with a cursor.

```
[2016-04-07 10:58:23 root@debian ~]>pipework
Syntax:
pipework <hostinterface> [-i containerinterface] [-l localinterfacename] <guest> <ipaddr>/<subnet>[@default_gateway] [macaddr][@vlan]
pipework <hostinterface> [-i containerinterface] [-l localinterfacename] <guest> dhcp [macaddr][@vlan]
pipework --wait [-i containerinterface]
[2016-04-07 11:19:11 root@debian ~]>█
```

#コマンドオプション

PIPEWORK - コマンドとイメージ図

```
dockerhost$ CPID=`docker run -ti buxybox` #C-p C-qでdetach  
dockerhost$ pipework br1 $CPID 192.168.100.2/24
```



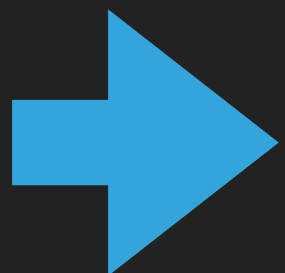
//図は以下のnwdiagにて作成

//<http://blockdiag.com/en/nwdiag/demo.html>

```
{  
  Internet [shape=cloud];  
  Internet --dockerhost;  
  network docker0-eth0 {  
    address = "172.17.0.0/16";  
    dockerhost [address = ".1 (docker0)"];  
    linux [address = ".2 (eth0)"];  
  }  
  network br1-eth1 {  
    address = "192.168.100.0/24";  
    dockerhost [address = ".2 (br1)"];  
    linux [address = ".2 (eth1)"];  
  }  
}
```

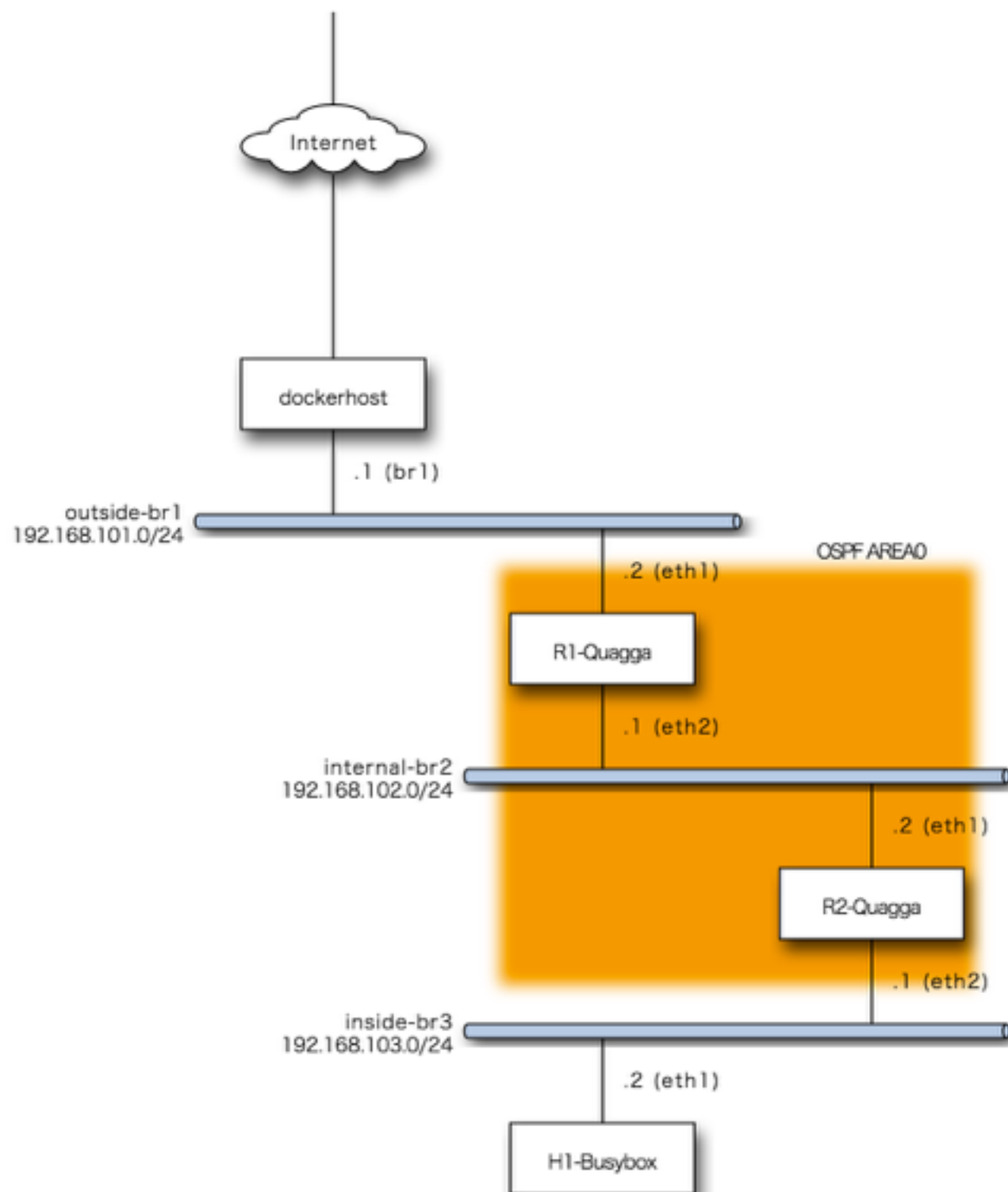
今回作成するネットワーク

- ▶ dockerhostのアップリンクでMASQUERADEしてインターネットにルーティング
- ▶ Quaggaを2台使用しOSPFで静的経路を動的にルーティング
- ▶ これらのコンテナ間ネットワークを全てpipeworkで作成



インターネットへのアップリンク以外は
DOCKERが作成するネットワークを一切使用しない

完成イメージ図



```
{
  Internet [shape=cloud];
  Internet --dockerhost;
  group OSPF {
    R1-Quagga;
    R2-Quagga;
    label="OSPF AREA0";
  }
  network outside-br1 {
    address = "192.168.101.0/24";
    dockerhost [address = ".1 (br1)"];
    R1-Quagga [address = ".2 (eth1)"];
  }
  network internal-br2 {
    address = "192.168.102.0/24";
    R1-Quagga [address = ".1 (eth2)"];
    R2-Quagga [address = ".2 (eth1)"];
  }
  network inside-br3 {
    address = "192.168.103.0/24";
    R2-Quagga [address = ".1 (eth2)"];
    H1-Busybox [address = ".2 (eth1)"];
  }
}
```


WANの作成

```
dockerhost$ echo 1 > /proc/sys/net/ipv4/ip_forward #ルータ化
```

```
dockerhost$ echo 0 > /proc/sys/net/bridge/bridge-nf-call-iptables #bridgeでNATチェーンを消化しないように
```

```
dockerhost$ echo 0 > /proc/sys/net/bridge/bridge-nf-call-ip6tables #bridgeでNATチェーンを消化しないように
```

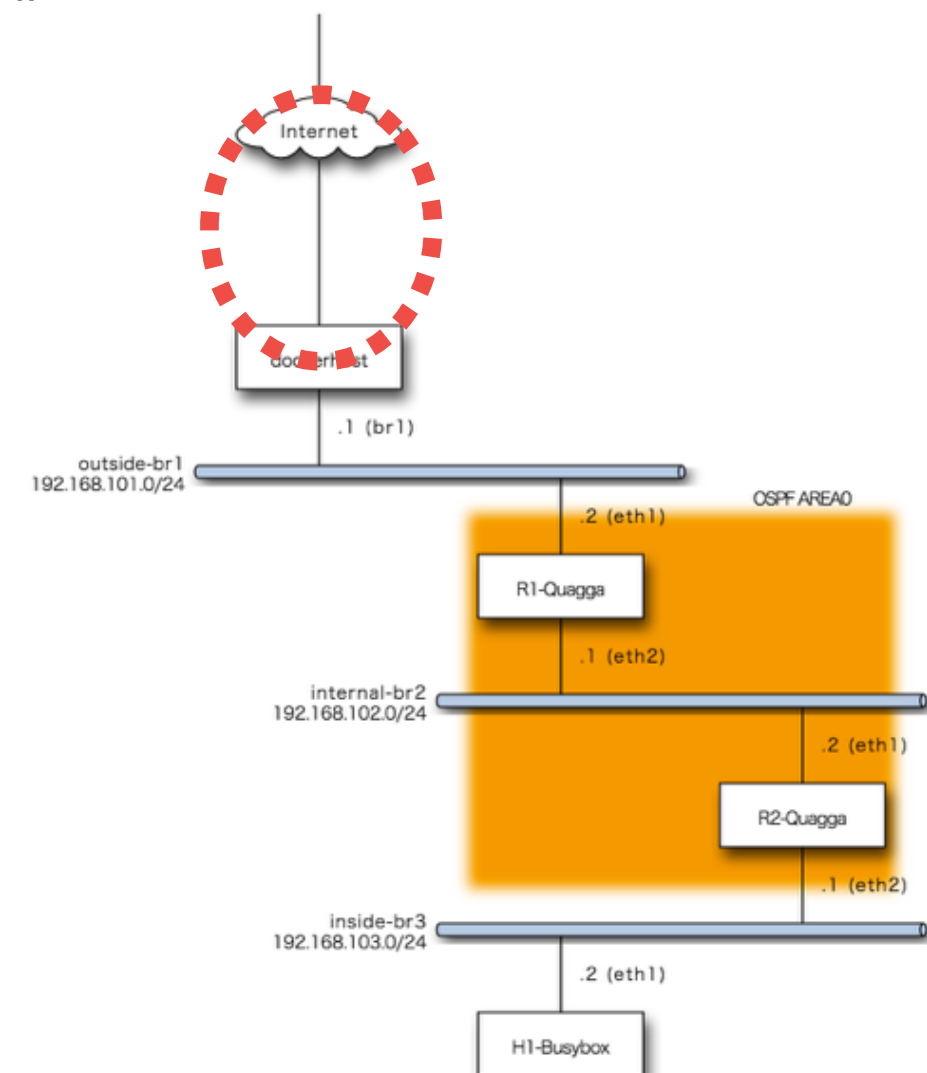
はまるトコロ！

```
dockerhost$ echo 0 > /proc/sys/net/bridge/bridge-nf-call-arptables #bridgeでNATチェーンを消化しないように
```

```
dockerhost$ iptables -A FORWARD -i br1 -o eth0 -j ACCEPT #こっち向きのSYNは通す
```

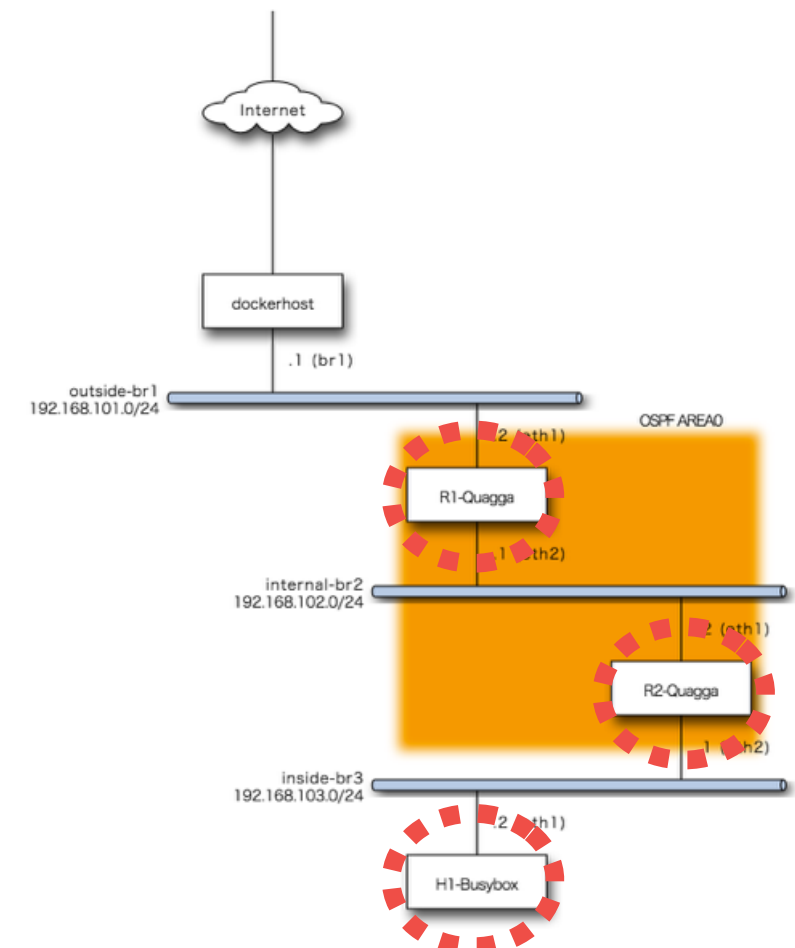
```
dockerhost$ iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT #あとはSYN通さない(flag偽装できるかな)
```

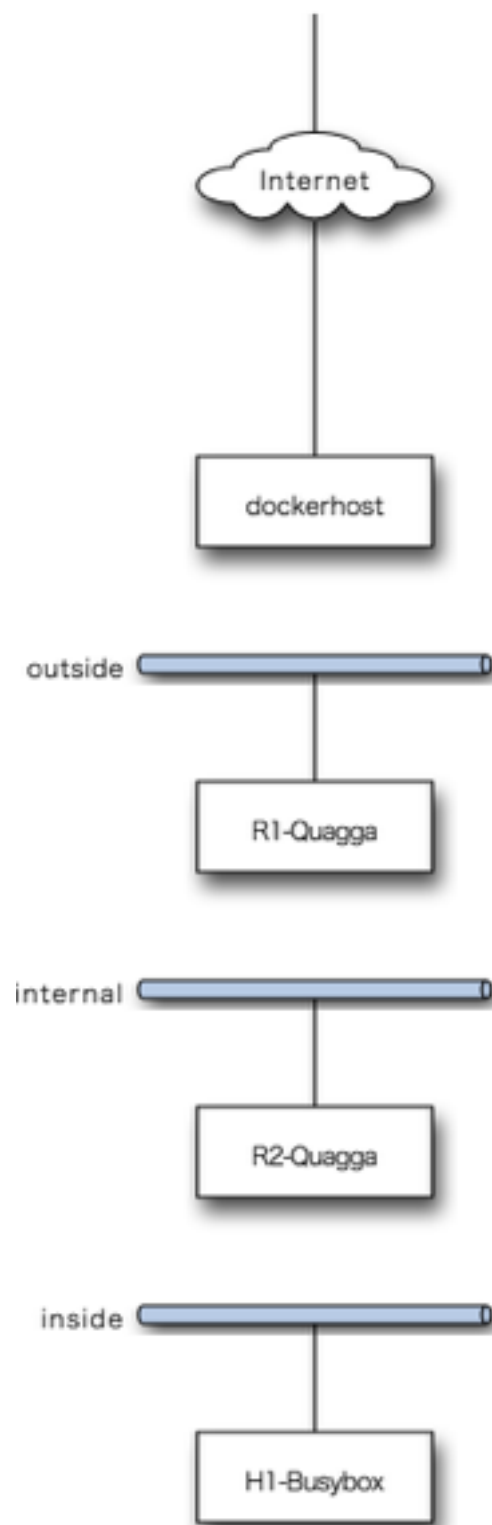
```
dockerhost$ iptables -t nat -A POSTROUTING -o <WAN IF> -j SNAT --to-source <WAN IP>
```



各種コンテナの作成

```
dockerhost$ docker pull syakesaba/quagga ; docker pull busybox;  
dockerhost$ docker run -ti --rm --privileged --name R1 --hostname R1 --net=none syakesaba/quagga;  
R1# service quagga start; #C-p C-q  
dockerhost$ docker run -ti --rm --privileged --name R2 --hostname R2 --net=none syakesaba/quagga;  
R2# service quagga start; #C-p C-q  
dockerhost$ docker run -ti --rm --privileged --name H1 --hostname=H1 --net=none busybox;  
H1# #C-p C-q
```

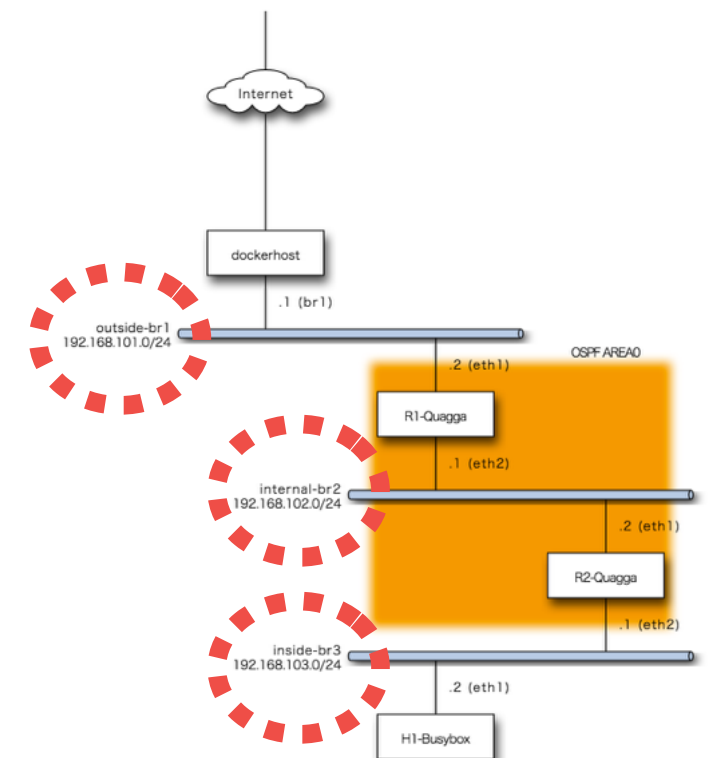


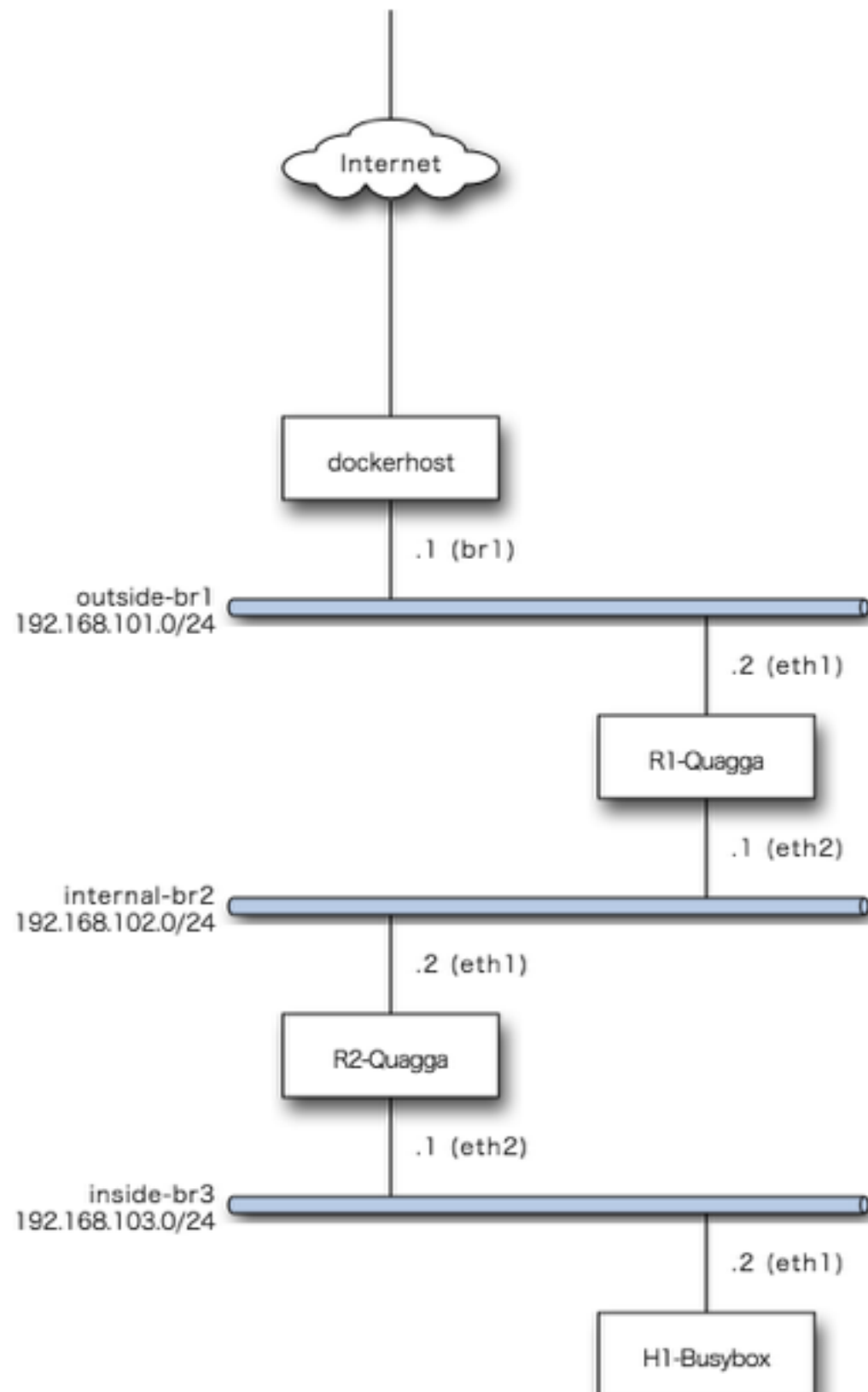


```
{  
  Internet [shape=cloud];  
  Internet --dockerhost;  
  network outside {  
    R1-Quagga [address = ""];  
  }  
  network internal {  
    R2-Quagga [address = ""];  
  }  
  network inside {  
    H1-Busybox [address = ""];  
  }  
}
```


各種ネットワークの作成

```
dockerhost$ git clone https://github.com/jpetazzo/pipework.git; cd pipework;  
dockerhost$ pipework br1 -i eth1 R1 192.168.101.2/24@192.168.101.1;  
dockerhost$ pipework br2 -i eth2 R1 192.168.102.1/24;  
dockerhost$ pipework br2 -i eth1 R2 192.168.102.2/24;  
dockerhost$ pipework br3 -i eth2 R2 192.168.103.1/24;  
dockerhost$ pipework br3 -i eth1 H1 192.168.103.2/24@192.168.103.1;  
dockerhost$ ifconfig br1 192.168.101.1/24 up  
dockerhost$ route add -net 192.168.102.0/24 gw 192.168.101.2  
dockerhost$ route add -net 192.168.103.0/24 gw 192.168.101.2
```



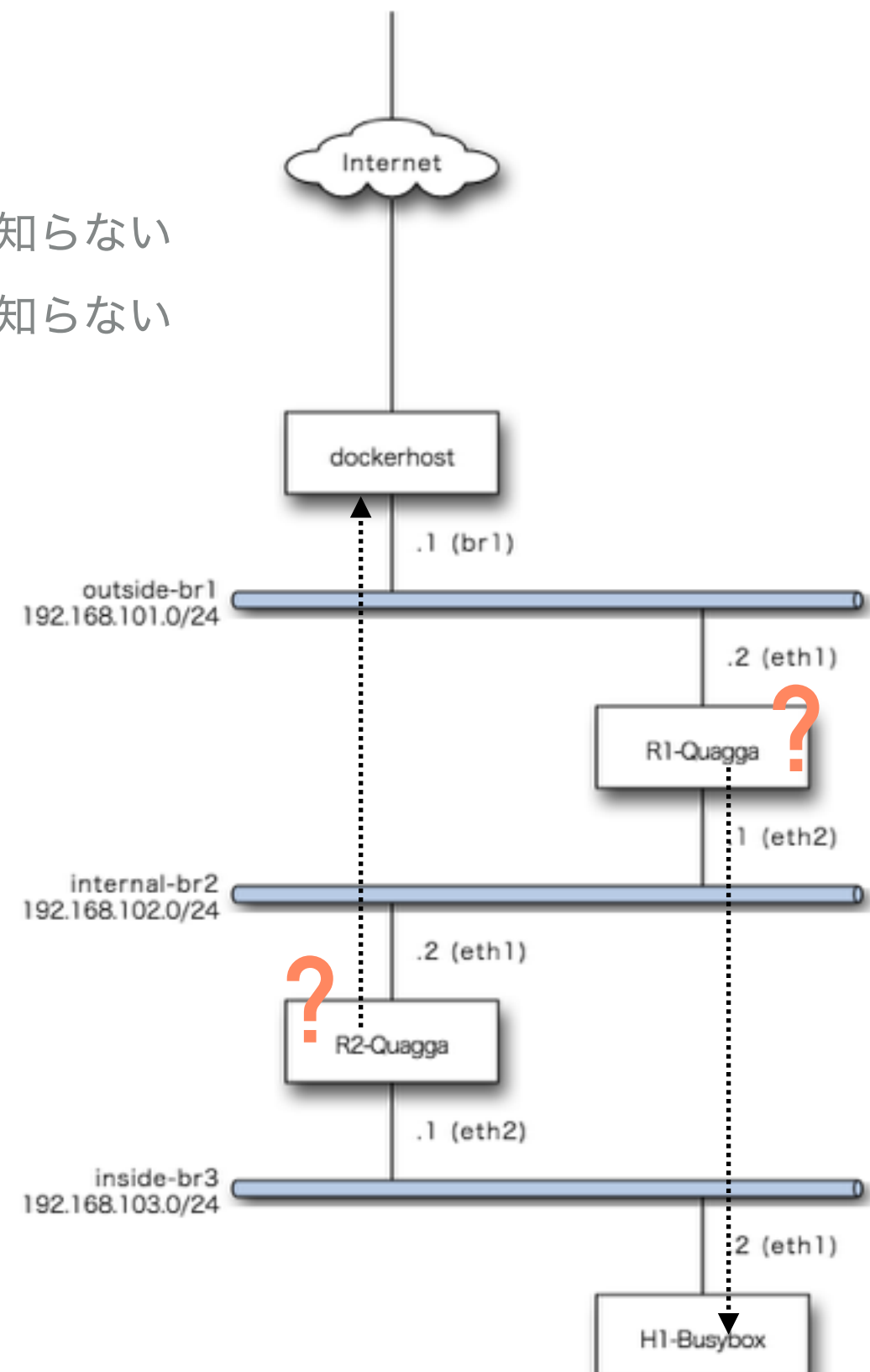


```
{
  Internet [shape=cloud];
  Internet --dockerhost;
  network outside-br1 {
    address = "192.168.101.0/24";
    dockerhost [address = ".1 (br1)"];
    R1-Quagga [address = ".2 (eth1)"];
  }
  network internal-br2 {
    address = "192.168.102.0/24";
    R1-Quagga [address = ".1 (eth2)"];
    R2-Quagga [address = ".2 (eth1)"];
  }
  network inside-br3 {
    address = "192.168.103.0/24";
    R2-Quagga [address = ".1 (eth2)"];
    H1-Busybox [address = ".2 (eth1)"];
  }
}
```


現状の問題の確認

R1はゲートウェイへの経路を知っているがH1への経路を知らない

R2はH1への経路を知っているがゲートウェイへの経路を知らない

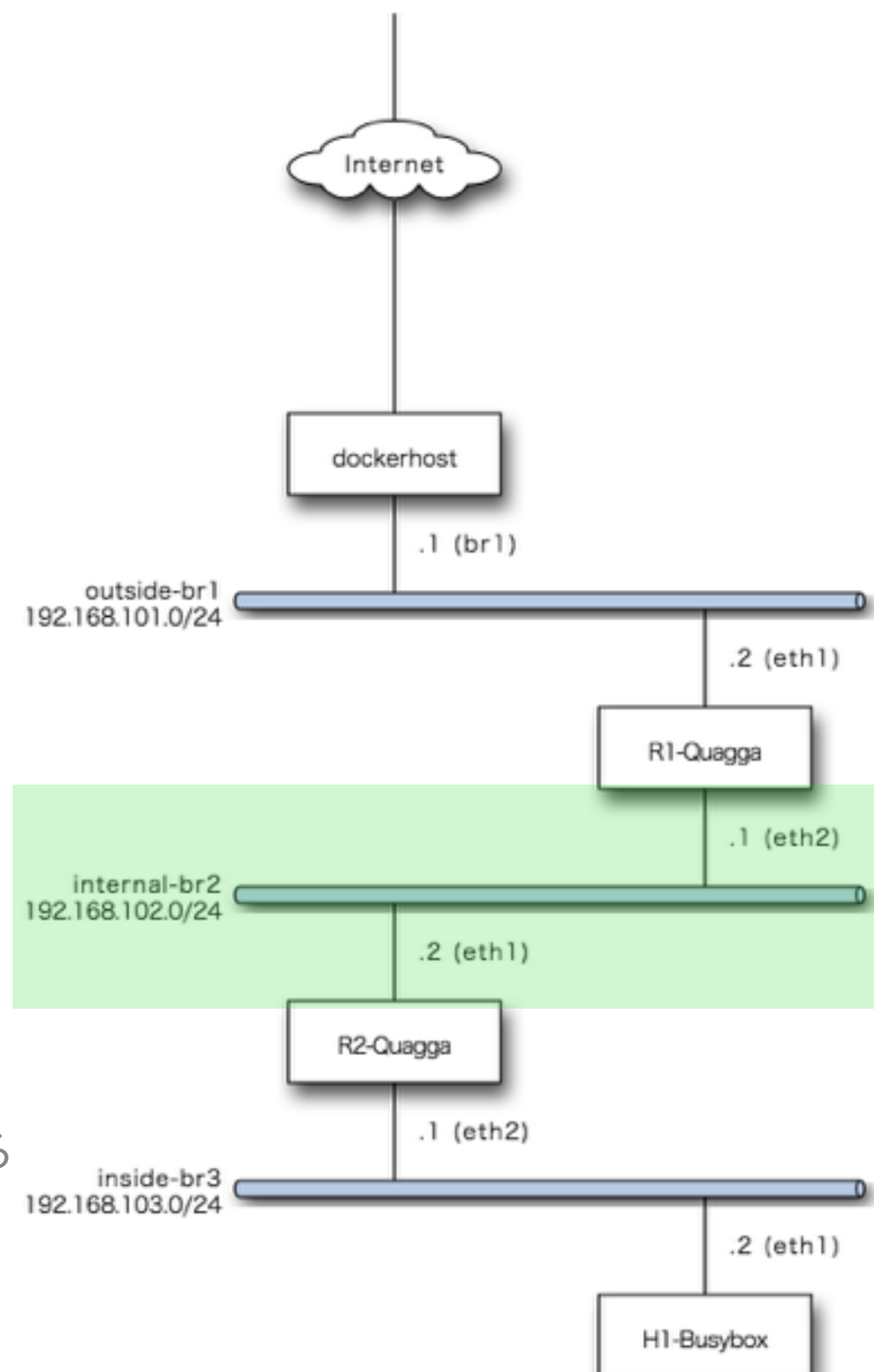


OSPFを動かそう！

OSPFにおいてバックボーンエリアへの接続は必須なのでR1-R2の間のinternal-br2を便宜上area0としinternal-br2にOSPFを流す！



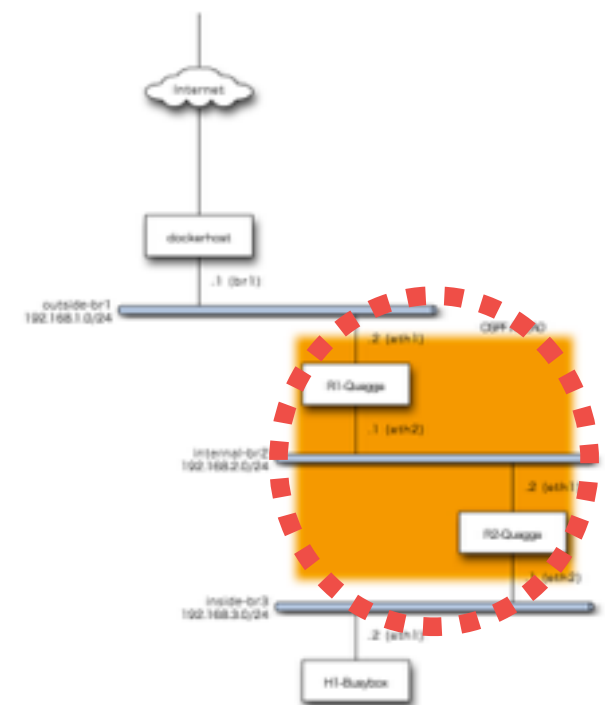
internal-br2にOSPFのマルチキャストが流れるようにする
他のネットワークにはマルチキャストは流さないようにする



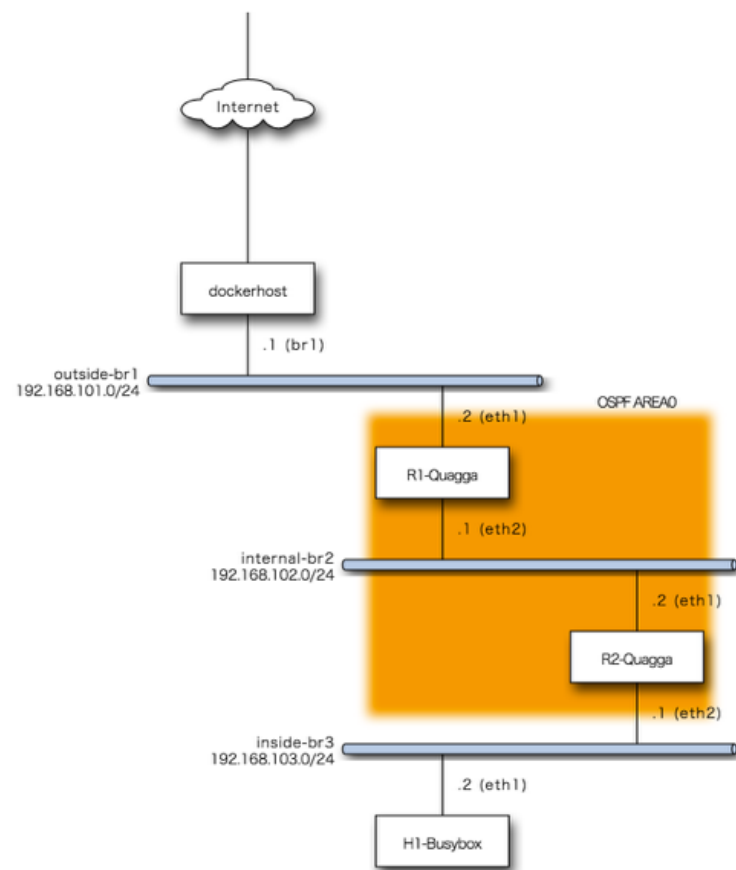
OSPFネットワークの作成

```
dockerhost$ docker attach R1
R1#telnet localhost 2604 #password=zebra
ospfd> enable
ospfd# configure terminal
ospfd(config)# router ospf
ospfd(config-router)# router-id 192.168.102.1
ospfd(config-router)# passive-interface default
ospfd(config-router)# no passive-interface eth2
ospfd(config-router)# network 192.168.101.0/24 area 101
ospfd(config-router)# network 192.168.102.0/24 area 0
ospfd(config-router)# default-information originate
ospfd(config-router)# end
ospfd# show running-config
# C-p C-qでdetach
```

```
dockerhost$ docker attach R2
R2#telnet localhost 2604 #password=zebra
ospfd> enable
ospfd# configure terminal
ospfd(config)# router ospf
ospfd(config-router)# router-id 192.168.102.2
ospfd(config-router)# passive-interface default
ospfd(config-router)# no passive-interface eth1
ospfd(config-router)# network 192.168.102.0/24 area 0
ospfd(config-router)# network 192.168.103.0/24 area 103
ospfd(config-router)# end
ospfd# show running-config
# C-p C-qでdetach
```



完成！



```
[2016-04-08 02:33:23 root@debuan ~]>ip route
default via 133.242.17.1 dev eth0
133.242.17.0/24 dev eth0 proto kernel scope link src 133.242.17.5
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
192.168.101.0/24 dev br1 proto kernel scope link src 192.168.101.1
192.168.102.0/24 via 192.168.101.2 dev br1
192.168.103.0/24 via 192.168.101.2 dev br1
```

```
root@R1:/# ip route
default via 192.168.101.1 dev eth1
192.168.101.0/24 dev eth1 proto kernel scope link src 192.168.101.2
192.168.102.0/24 dev eth2 proto kernel scope link src 192.168.102.1
192.168.103.0/24 via 192.168.102.2 dev eth2 proto zebra metric 20
```

```
root@R2:~# ip route
default via 192.168.102.1 dev eth1 proto zebra metric 10
192.168.101.0/24 via 192.168.102.1 dev eth1 proto zebra metric 20
192.168.102.0/24 dev eth1 proto kernel scope link src 192.168.102.2
192.168.103.0/24 dev eth2 proto kernel scope link src 192.168.103.1
```

```
/ # ip route
default via 192.168.103.1 dev eth1
192.168.103.0/24 dev eth1 src 192.168.103.2
```


おさらい

- ▶ pipeworkはネットワークの検証に超便利！
 - ▶ できれば物理NICを複数用意してbrctlでdockerコンテナと紐付けよう！
- ▶ linuxのbridgeを使用するので、bridgeまわりの挙動に注意しよう！
 - ▶ iptablesがどう動くか、パケットがどう動くか、STP、ループ等。
- ▶ やれば大抵動く！なんでも試してみよう！
 - ▶ QuaggaでBGP,ECMPによる経路バランシング、VXLANでオーバレイネットワーク、iptablesやnginx等を使用してロードバランシング、brctlでコンテナにグローバルIP付与、DHCP・NetBIOSの検証、各種攻撃手法の検証

お片づけ（超絶手抜き）

```
dockerhost$ docker stop {R1,R2,H1}
dockerhost$ docker rm -f {R1,R2,H1}
dockerhost$ ip link delete {br1,br2,br3} type bridge
dockerhost$ echo 0 > /proc/sys/net/ipv4/ip_forward
dockerhost$ iptables -t nat -D POSTROUTING <NUMBER>
```