brainscraps.wikia.com

http://brainscraps.wikia.com/wiki/Setup_Gateway_Routing_On_Multiple_Network_Interfaces

Setup Gateway Routing On Multiple Network Interfaces | BrainScraps Wiki

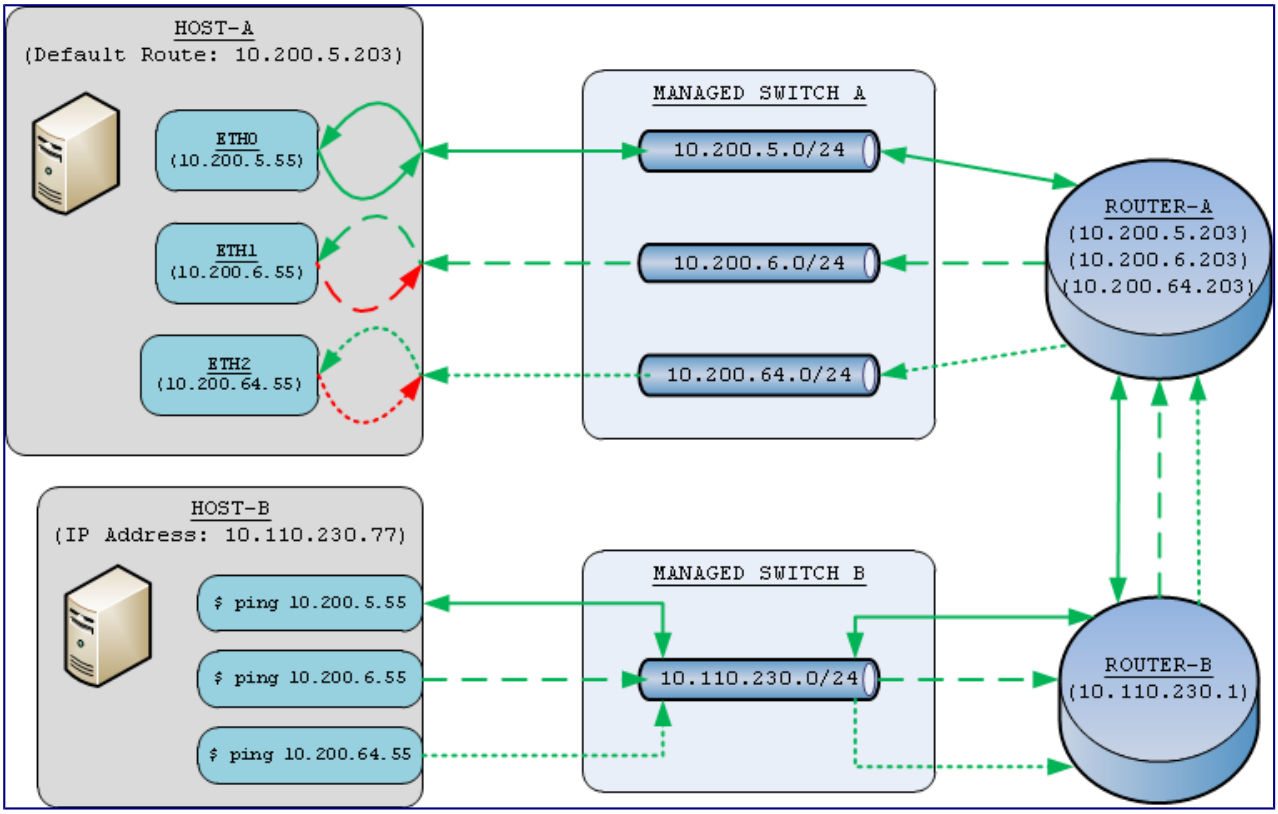
Introduction

Generally speaking, it's rare that you will need to have multiple network interfaces on a system and need to reach all the IP addresses on the system via external subnets. Most of the time, any additional IP addresses on a system will either reside on the same subnet or you will have some sort of heartbeat network in a cluster.

In the first case, all IP addresses can utilize the default gateway to get out of the system. You could even place these common-subnet IP addresses on interface aliases of a single NIC.

In the second case, you would have a NIC designated for transmission of data that would reach other subnets and, therefore, go through the default gateway. The other NICs would be for heartbeat traffic and communicate with other systems within the same subnet. Thereby, not needing to route through the default gateway.

However, there is the rare occassion that you would need to be able to reach each of the IP addresses in separate subnets on a single host from external subnets. Actually, it's not the reaching them part that is the issue; it's their response. As you can see in this diagram, With HOST-A having a default route of 10.200.5.203, you can receive a reply from the IP address within that subnet. However, with the other IP addresses on the host, you will not. The reason for this is because those interfaces do not know how to route back to you. The IP addresses on those interfaces are only aware of their own subnet, and cannot even find the subnet containing the default route out of the system. They are also unaware of the router specified for their own subnet.



So to counter act this issue, with Linux, you can setup additional routing tables, so when we setup a routing table for each interface we have, we can allow the IP address on that interface to have its own default route. This will allow the IP address attempting a reply to route the reply through a router that can get the response back to the host initiating the connection.

Creating Routes & Rules

To accomplish this, we will need to setup the routing tables on each additional interface excluding the interface managing the default gateway for the system.

First we will create an entry in the rt_tables file. This will allow us to use that table name with our further policy routes.

```
# echo "1 eth1" >>/etc/iproute2/rt_tables
```

Next we add two routes. One for the subnet so that the IP address can find its gateway. Another for specifying the default gateway for that interface.

```
# ip route add 10.200.6.0/24 dev eth1 src 10.200.6.55 table eth1
# ip route add default via 10.200.6.203 dev eth1 table eth1
```

Finally, we add policy route rules for ONLY that IP address that we want to use that table. This will not only ensure that the IP address we're trying to communicate with on that one interface can respond properly, but it will also ensure that we don't route information between subnets. The goal of this task is to communicate with individual IP addresses on other interfaces, not turn the system into another router allowing communication between subnets.

```
# ip rule add from 10.200.6.55/32 table eth1
# ip rule add to 10.200.6.55/32 table eth1
```

Automating on Startup

These 5 commands will need to be run for each interface we have on the system. To accomplish this task efficiently, I've added a "ROUTER=" line to the corresponding /etc/sysconfig/network-scripts/ifcfg-ethX file and placed a script in /etc/rc.local on the system. By deriving the table name and number after the interface, I've eliminated the need to keep this information somewhere additional.

/etc/sysconfig/network-scripts/ifcfg-eth1

```
# Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE]
DEVICE=eth1
BOOTPROTO=static
ONBOOT=yes
HWADDR=00:50:56:98:01:55
NETWORK=10.200.6.0
BROADCAST=10.200.6.255
NETMASK=255.255.255.0
ROUTER=10.200.6.203
IPADDR=10.200.6.55
```

/etc/rc.d/rc.local

```
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.

touch /var/lock/subsys/local

#mbm - 04/16/2012 - add additional routes
for interface_file in $(ls /etc/sysconfig/network-scripts/ifcfg-eth* | grep -v ifcfg-
eth0) ;do
  . ${interface_file}
  prefix=$(ipcalc -p ${IPADDR} ${NETMASK} | awk -F= '{print $2}')
  tablenum=$(echo ${DEVICE} | sed 's/eth//g')
  if [ ${ONBOOT} != 'yes' ] ;then
    continue
  fi
  if ! grep "^${tablenum} ${DEVICE}$" /etc/iproute2/rt_tables >/dev/null ;then
    echo "${tablenum} ${DEVICE}" >>/etc/iproute2/rt_tables
  fi
  ip route add ${NETWORK}/${prefix} dev ${DEVICE} src ${IPADDR} table ${DEVICE}
  ip route add default via ${ROUTER} dev ${DEVICE} table ${DEVICE}
  ip rule add from ${IPADDR}/32 table ${DEVICE}
  ip rule add to ${IPADDR}/32 table ${DEVICE}
done
```

Displaying Routes & Rules

These additional routes and rules will not show up in the normal routing table via "netstat -rn". You will need to use the "ip" command to see them. For reference, I've included the additions to the rt_tables file, the commands to get the routing tables and their expected output below:

Route Tables List

```
# cat /etc/iproute2/rt_tables
#
# reserved values
#
255     local
254     main
253     default
0       unspec
#
# local
#
#1      inr.ruhep
1 eth1
2 eth2
```

Routing Tables

```
# ip route show table all
10.200.6.0/24 dev eth1  table eth1  scope link  src 10.200.6.55
default via 10.200.6.203 dev eth1  table eth1
10.200.64.0/24 dev eth2  table eth2  scope link  src 10.200.64.55
default via 10.200.64.203 dev eth2  table eth2
10.200.6.0/24 dev eth1  proto kernel  scope link  src 10.200.6.55
10.200.5.0/24 dev eth0  proto kernel  scope link  src 10.200.5.55
10.200.64.0/24 dev eth2  proto kernel  scope link  src 10.200.64.55
default via 10.200.5.203 dev eth0
broadcast 10.200.6.255 dev eth1  table 255  proto kernel  scope link  src
10.200.6.55
broadcast 10.200.5.0 dev eth0  table 255  proto kernel  scope link  src 10.200.5.55
broadcast 10.200.64.255 dev eth2  table 255  proto kernel  scope link  src
10.200.64.55
broadcast 127.255.255.255 dev lo  table 255  proto kernel  scope link  src
127.0.0.1
local 10.200.5.55 dev eth0  table 255  proto kernel  scope host  src 10.200.5.55
broadcast 10.200.6.0 dev eth1  table 255  proto kernel  scope link  src 10.200.6.55
broadcast 10.200.5.255 dev eth0  table 255  proto kernel  scope link  src
10.200.5.55
broadcast 10.200.64.0 dev eth2  table 255  proto kernel  scope link  src
10.200.64.55
local 10.200.6.55 dev eth1  table 255  proto kernel  scope host  src 10.200.6.55
broadcast 127.0.0.0 dev lo  table 255  proto kernel  scope link  src 127.0.0.1
local 10.200.64.55 dev eth2  table 255  proto kernel  scope host  src 10.200.64.55
local 127.0.0.1 dev lo  table 255  proto kernel  scope host  src 127.0.0.1
local 127.0.0.0/8 dev lo  table 255  proto kernel  scope host  src 127.0.0.1
```

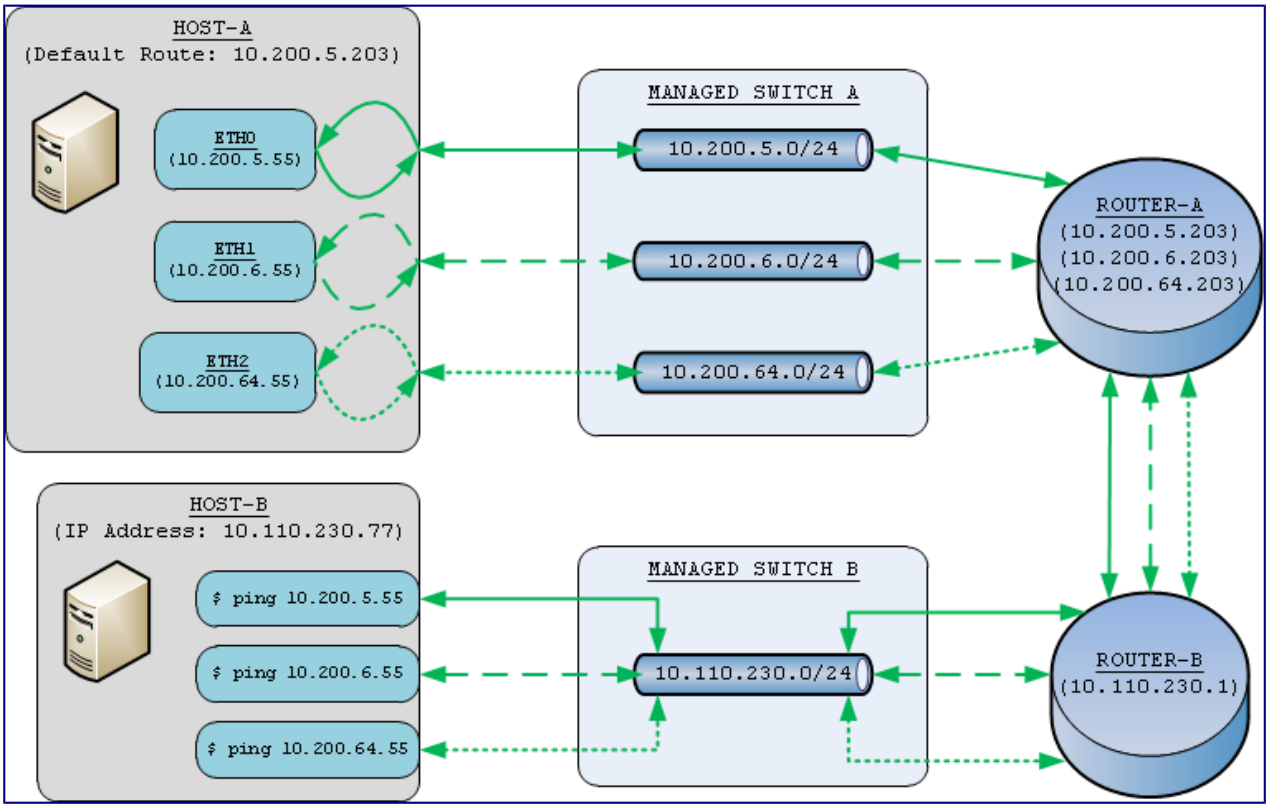
IPRoute Rules

```
# ip rule show
0:      from all lookup 255
32762:  from all to 10.200.64.55 lookup eth2
32763:  from 10.200.64.55 lookup eth2
32764:  from all to 10.200.6.55 lookup eth1
32765:  from 10.200.6.55 lookup eth1
32766:  from all lookup main
32767:  from all lookup default
```

Conclusion

After these routes and rules have been added, you should be able to fully ping each IP address on the system.

```
$ fping 10.200.5.55 10.200.6.55 10.200.64.55
10.200.5.55 is alive
10.200.6.55 is alive
10.200.64.55 is alive
```



Information gleamed from: Darien Kindlund