Improve Your Multi-Homed Servers with Policy Routing

JONATHON ANDERSON

Jonathon Anderson has been an HPC Sysadmin since 2006 and believes that everything would be a lot easier if we just spent more time figuring out the correct way to do things. He's currently serving as HPC Engineer at the University of Colorado and hopes to stick around Boulder for a long time to come.
jonathon.anderson@colorado.edu

Traditional IP routing systems route packets by comparing the destination address against a predefined list of routes to each available subnet; but when multiple potential routes exist between two hosts on a network, the preferred route may be dependent on context that cannot be inferred from the destination alone. The Linux kernel, together with the iproute2 suite [1], supports the definition of multiple routing tables [2] and a routing policy database [3] to select the preferred routing table dynamically. This additional expressiveness can be used to avoid multiple routing pitfalls, including asymmetric routes and performance bottlenecks from suboptimal route selection.

Background

The CU-Boulder Research Computing environment spans three datacenters, each with its own set of special-purpose networks. A traditionally routed host simultaneously connected to two or more of these networks compounds network complexity by making only one interface (the default gateway) generally available across network routes. Some cases can be addressed by defining static routes, but even this leads to asymmetric routing that is at best confusing and at worst a performance bottleneck.

Over the past few months we've been transitioning our hosts from a single-table routing configuration to a policy-driven, multi-table routing configuration. The end result is full bi-directional connectivity between any two interfaces in the network, irrespective of underlying topology or a host's default route. This has reduced the apparent complexity in our network by allowing the host and network to Do the Right Thing™ automatically, unconstrained by an otherwise static route map.

Linux policy routing has become an essential addition to host configuration in the University of Colorado Boulder "Science Network." It's so useful, in fact, that I'm surprised a basic routing policy isn't provided by default for multi-homed servers.

The Problem with Traditional Routing

The simplest Linux host routing scenario is a system with a single network interface.

```
# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
       pfifo_fast state UP qlen 1000
    link/ether 00:50:56:88:56:1f brd ff:ff:ff:ff:ff:ff
    inet 10.225.160.38/24 brd 10.225.160.255 scope global dynamic ens192
       valid_lft 60184sec preferred_lft 60184sec
```

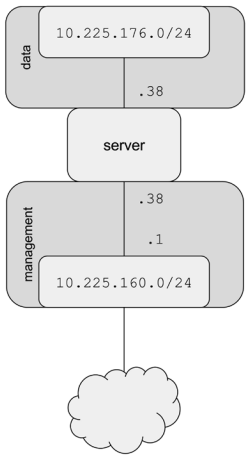

Figure 1: A simple dual-homed server with a traditional default route

Such a typically configured network with a single uplink has a single default route in addition to its link-local route.

```
# ip route list
default via 10.225.160.1 dev ens192
10.225.160.0/24 dev ens192  proto kernel  scope link  src
10.225.160.38
```

Traffic to hosts on 10.225.160.0/24 is delivered directly, while traffic to any other network is forwarded to 10.225.160.1.

A dual-homed host adds a second network interface and a second link-local route, but the original default route remains (see Figure 1).

```
# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
        pfifo_fast state UP qlen 1000
    link/ether 00:50:56:88:56:1f brd ff:ff:ff:ff:ff:ff
    inet 10.225.160.38/24 brd 10.225.160.255 scope global
dynamic ens192
    valid_lft 86174sec preferred_lft 86174sec
3: ens224: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
        pfifo_fast state UP qlen 1000
    link/ether 00:50:56:88:44:18 brd ff:ff:ff:ff:ff:ff
    inet 10.225.176.38/24 brd 10.225.176.255 scope global dynamic
ens224
       valid_lft 69193sec preferred_lft 69193sec

# ip route list
default via 10.225.160.1 dev ens192
```

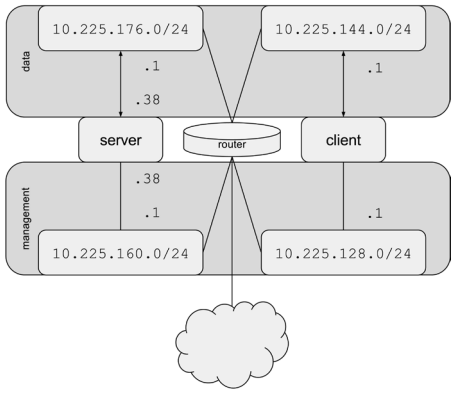

Figure 2: A server and a client, with static routes between their data interfaces

```
10.225.160.0/24 dev ens192  proto kernel  scope link  src
10.225.160.38
10.225.176.0/24 dev ens224  proto kernel  scope link  src
10.225.176.38
```

The new link-local route provides access to hosts on 10.225.176.0/24 and is sufficient for a private network connecting a small cluster of hosts. In fact, this is the configuration that we started with in our Research Computing environment: .160.0/24 is a low-performance "management" network, while .176.0/24 is a high-performance "data" network.

In a more complex network, however, link-local routes quickly become insufficient. In the CU Science Network, for example, each datacenter is considered a discrete network zone with its own set of "management" and "data" networks. For hosts in different network zones to communicate, a static route must be defined in each direction to direct performance-sensitive traffic across the high-performance network route (see Figure 2).

```
server # ip route add 10.225.144.0/24 via 10.225.176.1
client # ip route add 10.225.176.0/24 via 10.225.144.0
```

Although managing these static routes can be tedious, they do sufficiently define connectivity between the relevant network pairs: "data" interfaces route traffic to each other via high-performance networks, while "management" interfaces route traffic to each other via low-performance networks. Other networks (e.g., the Internet) can only communicate with the hosts on their default routes; but this limitation may be acceptable for some scenarios.

Even this approach is insufficient, however, to allow traffic between "management" and "data" interfaces. This is particularly problematic when a client host is not equipped with a symmetric set of network interfaces (see Figure 3). Such a client may only have a "management" interface but should still

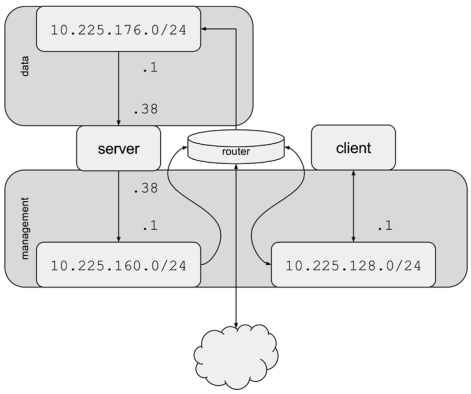

Figure 3: In a traditional routing configuration, the server would try to respond to the client via its default route, even if the request arrived on its data interface.

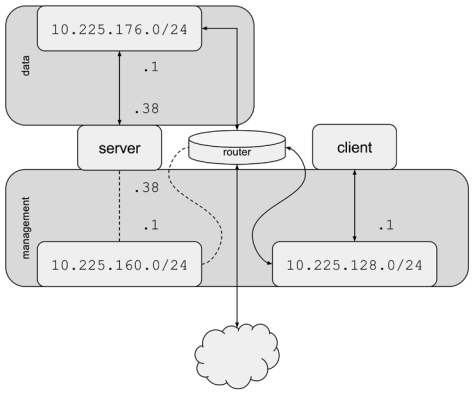

Figure 4: Routing policy allows the server to respond using its data interface for any request that arrives on its data interface, even if it has a different default route.

communicate with the server's high-performance interface for certain types of traffic. (For example, a dual-homed NFS server should direct all NFS traffic over its high-performance "data" network, even when being accessed by a client that itself only has a low-performance "management" interface.) By default, the Linux `rp_filter` [4] blocks this traffic, as the server's response to the client targets a different route than the incoming request; but even if `rp_filter` is disabled, this asymmetric route limits the server's aggregate network bandwidth to that of its lower-performing interface.

The server's default route could be moved to the "data" interface—in some scenarios, this may even be preferable—but this only displaces the issue: clients may then be unable to communicate with the server on its "management" interface, which may be preferred for certain types of traffic. In Research Computing, for example, we prefer that administrative access and monitoring not compete with IPC and file system traffic.

Routing Policy Rules

Traditional IP routing systems route incoming packets based solely on the the intended destination; but the Linux iproute2 stack supports route selection based on additional packet metadata, including the packet source. Multiple discrete routing tables, similar to the virtual routing and forwarding (VRF) support found in dedicated routing appliances [5], define contextual routes, and a routing policy selects the appropriate routing table dynamically based on a list of rules.

In the following example, there are three different routing contexts to consider. The first of these—the "main" routing table—defines the routes to use when the server initiates communication.

```
server # ip route list table main
10.225.144.0/24 via 10.225.176.1 dev ens224
default via 10.225.160.1 dev ens192
10.225.160.0/24 dev ens192  proto kernel  scope link  src
10.225.160.38
10.225.176.0/24 dev ens224  proto kernel  scope link  src
10.225.176.38
```

A separate routing table defines routes to use when responding to traffic on the "management" interface. Since this table is concerned only with the default route's interface in isolation, it simply reiterates the default route.

```
server # ip route add default via 10.225.160.1 table 1
server # ip route list table 1
default via 10.225.160.1 dev ens192
```

Similarly, the last routing table defines routes to use when responding to traffic on the "data" interface. This table defines a *different* default route: all such traffic should route via the "data" interface.

```
server # ip route add default via 10.225.176.1 table 2
server # ip route list table 2
default via 10.225.176.1 dev ens224
```

With these three routing tables defined, the last step is to define routing policy to select the correct routing table based on the packet to be routed. Responses from the "management" address should use table 1, and responses from the "data" address should use table 2. All other traffic, including server-initiated traffic that has no outbound address assigned yet, uses the "main" table automatically.

```
server # ip rule add from 10.225.160.38 table 1
server # ip rule add from 10.225.176.38 table 2
server # ip rule list
0:  from all lookup local
32764:  from 10.225.176.38 lookup 2
32765:  from 10.225.160.38 lookup 1
32766:  from all lookup main
32767:  from all lookup default
```

With this routing policy in place, a single-homed client (or, in fact, *any* client on the network) may communicate with both the server's "data" and "management" interfaces independently and successfully and the bi-directional traffic routes consistently via the appropriate network (see Figure 4).

Persisting the Configuration

This custom routing policy can be persisted in the Red Hat "ifcfg" network configuration system by creating interface-specific `route-` and `rule-` files.

```
# cat /etc/sysconfig/network-scripts/route-ens192
default via 10.225.160.1 dev ens192
default via 10.225.160.1 dev ens192 table mgt

# cat /etc/sysconfig/network-scripts/route-ens224
10.225.144.0/24 via 10.225.176.1 dev ens224
default via 10.225.176.1 dev ens224 table data

# cat /etc/sysconfig/network-scripts/rule-ens192
from 10.225.160.38 table mgt

# cat /etc/sysconfig/network-scripts/rule-ens224
from 10.225.176.38 table data
```

The symbolic names `mgt` and `data` used in these examples are translated to routing table numbers as defined in the `/etc/iproute2/rt_tables` file.

```
# echo "1 mgt" >>/etc/iproute2/rt_tables
# echo "2 data" >>/etc/iproute2/rt_tables
```

Once the configuration is in place, activate it by restarting the network service (e.g., `systemctl restart network`). You may also be able to achieve the same effect using `ifdown` and `ifup` on individual interfaces.

Red Hat's support for routing rule configuration has a confusing regression that merits specific mention. Red Hat (and its derivatives) has historically used a `network` init script and subscripts to configure and manage network interfaces, and these scripts support the aforementioned `rule-` configuration files. Red Hat Enterprise Linux 6 introduced NetworkManager, a persistent daemon with additional functionality; however, NetworkManager did not support `rule-` files until version 1.0, released as part of RHEL 7.1 [6]. If you're currently using NetworkManager, but wish to define routing policy in `rule-` files, you'll need to either disable NetworkManager entirely or exempt specific interfaces from NetworkManager by specifying `NM_CONTROLLED=no` in the relevant `ifcfg-` files.

In a Debian-based distribution, these routes and rules can be persisted using `post-up` directives in `/etc/network/interfaces`.

Further Improvements

We're still in the process of deploying this policy-based routing configuration in our Research Computing environment, and, as we do, we discover more cases where previously complex network requirements and special-cases are abstracted away by this relatively uniform configuration. We're simultaneously evaluating other potential changes, including the possibility of running a dynamic routing protocol (such as OSPF) on our multi-homed hosts, or of configuring every network connection as a simultaneous default route for failover. In any case, this experience has encouraged us to take a second look at our network configuration to reevaluate what we had previously thought were inherent limitations of the stack itself.

References

[1] Iproute2: http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2.

[2] Routing tables: http://linux-ip.net/html/routing-tables.html.

[3] Policy-based routing: http://linux-ip.net/html/routing-rpdb.html.

[4] rp-filter How To: http://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.kernel.rpf.html.

[5] Virtual routing and forwarding: http://www.cisco.com/c/en/us/td/docs/net_mgmt/active_network_abstraction/3-7/reference/guide/ANARefGuide37/vrf.html.

[6] Red Hat on policy-based routing persistence: https://access.redhat.com/solutions/288823.