

Graypaper

# Understanding routing in multi-homed HP-UX environments

---

Olivier S. Massé  
omasse@mayoxide.com  
September 9th 2008  
Revised June 24<sup>th</sup> 2009

## Abstract

Multi-homed HP-UX servers, especially ServiceGuard nodes, present a challenge in terms of routing, which is further exacerbated by the lack of documentation on the subject.

This paper tries to explain how to configure multi-homed servers to enhance the routing of IP packets and prevent asymmetric routing.

Note that I call this paper a "graypaper". I do not work for HP, nor do I have any internal knowledge of HP-UX. The information in this paper has been determined by looking at the output of tcpdump and by reading publicly accessible documentation and posts in the ITRC forums. This document is provided "as is" without any warranty.

## Revisions

<b>Date</b>	<b>Who</b>	<b>Modifications</b>
Sept 9 <sup>th</sup> 2008	O. Massé	Initial publication
Sept 18 <sup>th</sup> 2008	O. Massé	A few corrections, thanks to Rick Jones
June 24 <sup>th</sup> 2009	O. Massé	Added a section on UDP communication with extended distance clusters A few corrections, thanks to Laurent Menase

# System Setup

Our test system has HP-UX 11iv2 (11.23) with the June 2008 Quality Pack.

It has two physical interfaces:

- lan0, IP address 1.2.3.4
- lan1, IP address 5.6.7.8

It also has a logical indexed interface, which happens to be the floating IP address we'll be using:

- lan1:1, IP address 5.6.7.9

The default gateway is set to the default value, which will be the following:

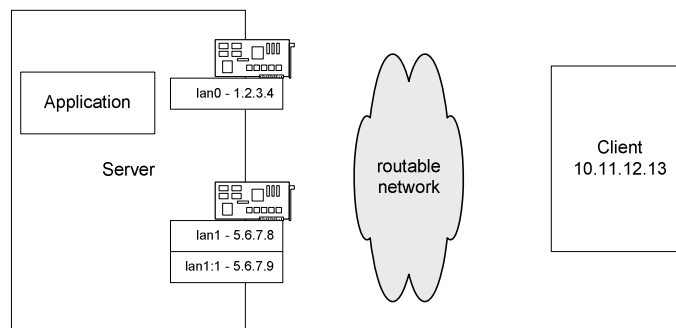
- lan0, IP address 1.2.3.1

A netstat -nr output on this system looks like this:

Destination	Gateway	Flags	Refs	Interface	Pmtu
127.0.0.1	127.0.0.1	UH	0	lo0	4136
1.2.3.4	1.2.3.4	UH	0	lan0	4136
5.6.7.8	5.6.7.8	UH	0	lan1	4136
5.6.7.9	5.6.7.9	UH	0	lan1:1	4136
1.2.3.0	1.2.3.4	U	3	lan0	1500
5.6.7.0	5.6.7.8	U	3	lan1	1500
5.6.7.0	5.6.7.9	U	3	lan1:1	1500
127.0.0.0	127.0.0.1	U	0	lo0	0
default	1.2.3.1	UG	0	lan0	0

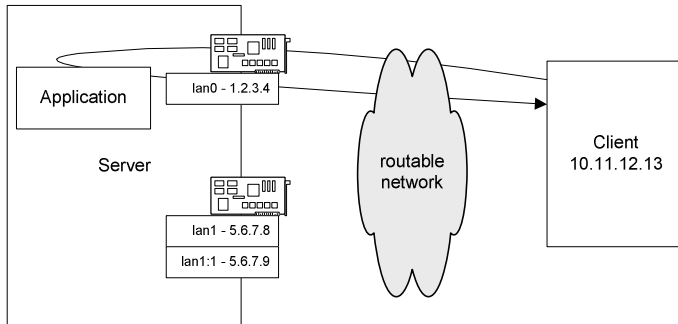
Our client is a remote device on the network, which is on a totally different subnet. Its IP address is 10.11.12.13.

This system setup is shown in the following diagram:

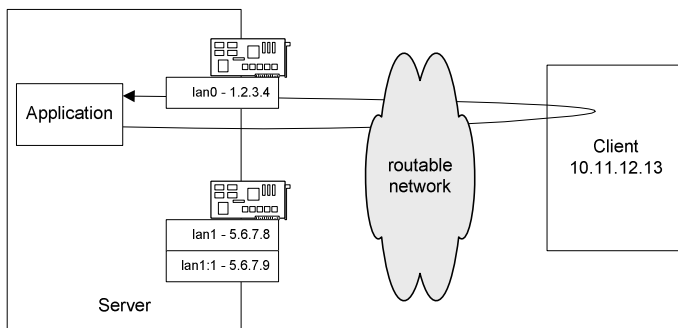


## *Introduction: a simple routing example*

In a typical scenario, where our client initiates an IP connection to 1.2.3.4, we'll get the following routing:



The same goes on if that connection is initiated from the server itself:



So far so good, everything works as we normally would expect. This is the behaviour that a server that isn't multi-homed would also exhibit.

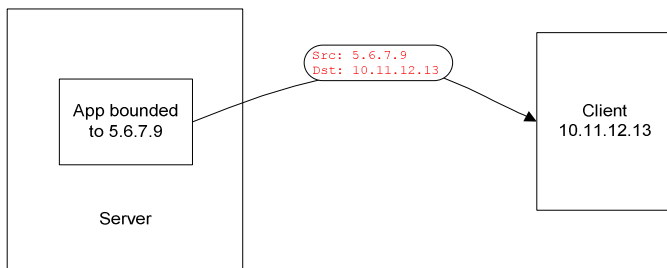
# Relocatable IP addresses

When setting up HA applications in a ServiceGuard cluster, relocatable (i.e. floating) IP addresses are used. Thus in our scenario, any client that desires to talk to our HA application is supposed to use IP address 5.6.7.9.

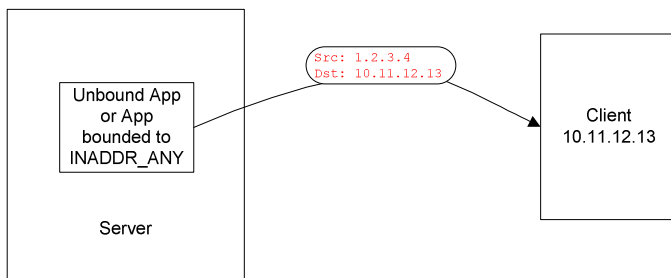
ServiceGuard best practices suggest that an HA application should bind itself to the floating IP address. This will make it behave as if the source IP of packets sent by the application is the floating address, and not the host address.

## *Bound vs. Unbound example*

With an HA application bound to IP address 5.6.7.9, any IP connection initiated from the HA application to the client will result in the following:



While applications that don't call `bind()`, or bind to `INADDR_ANY`, will do this:



This behaviour is expected, since 1.2.3.4 is our host address.

Note that the `bind()` rule is relevant only to applications that initiate TCP connections to the outside, or use UDP as it is a connectionless protocol. Applications that process incoming TCP/IP connections only, and return it in the same TCP connection such as web servers are not affected by this and will reply with a packet that has a correct source address. Some server applications that support virtual hosting also bind themselves to the address to which a user connection was initiated; an example of this is HP-UX's stock ftp daemon which is based on WU-FTPD.

## Dealing with firewalls

Binding is important when firewalls or access lists are involved, since it guarantees that only one and only source IP address will ever be associated with your application. If your application doesn't use `bind()` or binds to `INADDR_ANY`, the source IP address for outgoing TCP connections could theoretically be within a range multiple IP addresses such as, for instance:

- The physical IP address of all your hosts in your cluster
- The floating IP address of the application

When dealing with firewalls with unbound applications, if your security policy allows it, it is recommended to open access to all these possible IP addresses. If that is impossible, there are (unsupported) tricks than can be used to fool unbound applications into using a floating IP.

## Fooling unbound applications into using a floating IP

Not every application supports binding to a specific IP addresses, and in some cases requesting it to use `bind()` might be a problem if you're dealing with an obscure or legacy application. Remember that as mentioned previously, it's mostly applications that send *outgoing* IP connections that need to be dealt with. There are two tricks which can be used to circumvent this: static routes and network address translation.

Please note that the following techniques are not officially documented by HP. They may work, or may not work depending on your situation. Therefore, don't expect any frontline support from their HA team if you phone in with a support request. Consider these ideas as experimental.

### Using static routes

If (and only if) you know the destination subnet or IP address that your HA application will be talking to, static routes is an easy way to force traffic out with the floating source IP address. This can help you resolve lots of problems when dealing with firewalls. For this to work, ProxyARP must be supported on your routers.

For example, to force all traffic to 10.11.12.13 to use floating IP 5.6.7.9 as the source address, you can add the following route:

```
# route add host 10.11.12.13 5.6.7.9 0
```

This will create an entry in the route table like the one in red below:

Routing tables	Destination	Gateway	Flags	Refs	Interface	Pmtu
	127.0.0.1	127.0.0.1	UH	0	lo0	4136
	1.2.3.4	1.2.3.4	UH	0	lan0	4136
	5.6.7.8	5.6.7.8	UH	0	lan1	4136
	5.6.7.9	5.6.7.9	UH	0	lan1:1	4136
	10.11.12.13	5.6.7.9	UH	0	lan1:1	0
	1.2.3.0	1.2.3.4	U	3	lan0	1500
	5.6.7.0	5.6.7.8	U	3	lan1	1500
	5.6.7.0	5.6.7.9	U	3	lan1:1	1500

127.0.0.0	127.0.0.1	U	0	lo0	0
default	1.2.3.1	UG	0	lan0	0

Beware, however, that all traffic to 10.11.12.13, even some that could be bound to another source IP at that point, will go out on that path from now on.

You can also extend this technique to a whole subnet if you prefer:

```
# route add net 10.11.12.0 netmask 255.255.255.0 5.6.7.9 0
```

Resulting in the following route entry instead:

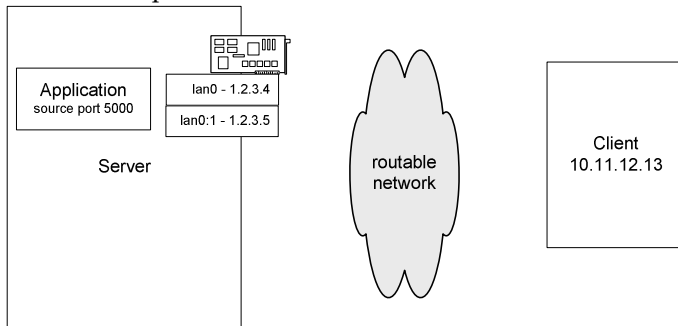
Destination	Gateway	Flags	Refs	Interface	Pmtu
127.0.0.1	127.0.0.1	UH	0	lo0	4136
1.2.3.4	1.2.3.4	UH	0	lan0	4136
5.6.7.8	5.6.7.8	UH	0	lan1	4136
5.6.7.9	5.6.7.9	UH	0	lan1:1	4136
1.2.3.0	1.2.3.4	U	3	lan0	1500
5.6.7.0	5.6.7.8	U	3	lan1	1500
5.6.7.0	5.6.7.9	U	3	lan1:1	1500
10.11.12.0	5.6.7.9	U	0	lan1:1	1500
127.0.0.0	127.0.0.1	U	0	lo0	0
default	1.2.3.1	UG	0	lan0	0

Note that these route commands cannot be executed at the startup of your server, since they're dependent on a relocatable IP addresses which is not available at startup. You will need to add them to your ServiceGuard startup scripts.

## Using network address translation

If you don't have a clue of the destination of your outgoing packets, IPFilter's NAT (network address translator) which is built into HP-UX can be used instead of static routes, to some degree. But to make this work, you really need to know what you're doing.

Let's start with an example where your server is configured slightly differently than before. In fact, it is simpler:



In this example, we'll use the NAT to rewrite IP packets from an unbound application to use source address 1.2.3.5 instead of 1.2.3.4.

This is done by putting the following in ipnat.conf:

```
map lan0 from 1.2.3.4/32 port = 5000 to any -> 1.2.3.5/32
```

This NAT rule will result in all outbound packets sent on lan0, with source IP 1.2.3.4 and source port 5000, to be rewritten as 1.2.3.5 before being put on the wire. It is important to specify the source port of the HA application here, or else you could end-up natting *all* the traffic from your host to your floating address, something you definitely do not want to do.

If you know the destination IP address or subnet, it might be easier to use a static route (see above). But if you insist on using a NAT, a safer and more fitting NAT rule would consist of applying the NAT rule to only the known remote IPs. For example, this will rewrite the source address only for TCP connections sent to system 10.11.12.13:

```
map lan0 from 1.2.3.4/32 port = 5000 to 10.11.12.13/32 -> 1.2.3.5/32
```

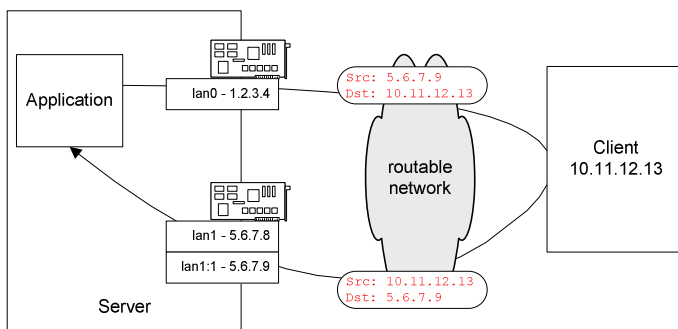
Natting in a way such as described above is a nasty workaround which has some drawbacks. First, it can cause some problems if you're filtering inbound packets with IP Filter. Since NAT rules are processed *after* filter rules for outgoing packets, no session state will be kept for your rewritten IP address, and you'll need to adapt your inbound filters accordingly to accept traffic that isn't part of an established session.

Furthermore, both filters and NAT rules are applied after the packets have been routed to the interface, so you can't easily rewrite the source IP to be on a totally different LAN as this will inadvertently cause asymmetric routing, i.e. sending outbound packets on the wrong interface.

To show this problem, let's consider this NAT rule which uses our initial server setup, where we need to change the source IP from 1.2.3.4 to 5.6.7.9:

```
map lan0 from 1.2.3.4/32 port = 5000 to 10.11.12.13/32 -> 5.6.7.9/32
```

When we enable this NAT rule, all packets destined to 10.11.12.13 will go out on lan0 with source IP address 5.6.7.9. However, lan0 is on subnet 1.2.3.0/24. We'll end up with outbound traffic being routed on lan0, while responses will be routed back to lan1:1, as illustrated in the following figure:



This is called **asymmetric routing**. The packets might get to their destination depending on your network topology, but it can create problems, which will be described later. Even if using the techniques that I suggest below to overcome asymmetric routing, there is no easy way, to my knowledge, to prevent it from happening when you're using a NAT to rewrite addresses to another subnet. This reckless configuration is for the most adventurous only!

Asymmetric routing doesn't happen only when we're natting. As a matter of fact, the default HP-UX routing algorithm does just that in many situations with multi-homed servers. The following chapter will describe these issues, and how they can be resolved.

# How routing is handled by HP-UX

HP-UX uses a static routing table to determine how to route packets. The routing(7) man page describes the default routing algorithm as following:

The routing table entries are of three types:

- + Entries for a specific host.
- + Entries for all hosts on a specific network.
- + Wildcard entries for any destination not matched by entries of the first two types.

To select a route for forwarding an IP packet, the network facilities select the complete set of "matching" routing table entries from the routing table.

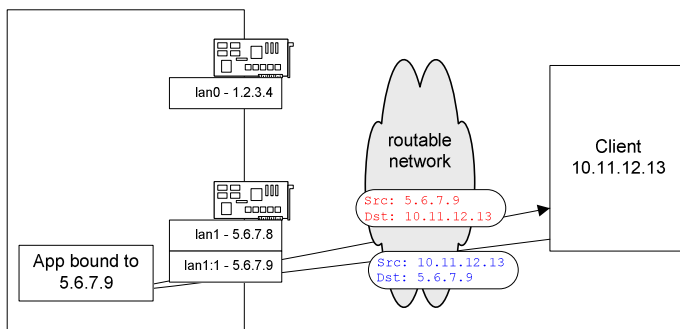
This routing algorithm is straightforward, but it can cause asymmetric routing in multi-homed environments.

## Asymmetric routing

We'll show here how asymmetric routing is created. First, let's recap the default routing table of our test system:

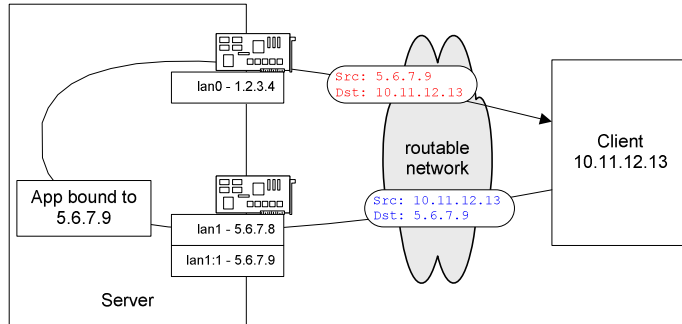
Destination	Gateway	Flags	Refs	Interface	Pmtu
127.0.0.1	127.0.0.1	UH	0	lo0	4136
1.2.3.4	1.2.3.4	UH	0	lan0	4136
5.6.7.8	5.6.7.8	UH	0	lan1	4136
5.6.7.9	5.6.7.9	UH	0	lan1:1	4136
1.2.3.0	1.2.3.4	U	3	lan0	1500
5.6.7.0	5.6.7.8	U	3	lan1	1500
5.6.7.0	5.6.7.9	U	3	lan1:1	1500
127.0.0.0	127.0.0.1	U	0	lo0	0
default	1.2.3.1	UG	0	lan0	0

Assuming your application binds to the floating IP address, the routing of responses to incoming packets should be as following, right?



Well, you're wrong!

You'll rather get this kind of routing:



In a nutshell, for any connection initiated from the client to the server, all packets get in through lan1:1, but the response comes out through lan0, even though the source address is 5.6.7.9. This is called **asymmetric routing**.

How can that be? That's because by default, the HP-UX routing algorithm doesn't consider the *source* address when determining what route to use; it only uses the *destination* address. So it doesn't matter if the source IP for the returning packet is 5.6.7.9; in this example, the destination address is 10.11.12.13, and there is no route to this destination. The routing algorithm will gladly pick the default route, which is gateway 1.2.3.1, and 1.2.3.1 is accessible only through lan0.

## Is asymmetric routing that bad?

Asymmetric routing is desirable in some cases, but the bottom line is that you have to know that it exists.

While often unnoticeable, it has a few gotchas. It makes you *think* that all traffic goes through a specific wire, yet it does *not*. This can lead to a performance hit, depending on your network.

For example:

- If you decide set lan1 at gigabit speed and put lan0 on a 100Mbit network, you'll be hit by a performance penalty for returned packets that you did not anticipate.
- If you run a host-based firewall such as IP Filter and filter by lan card, you also need to know precisely what interfaces will be used by your applications<sup>1</sup>.
- Lan1 might have a redundant network connection to enhance reliability of the network that has your floating IP, while lan0 does not. Unpredictable results might happen if all your traffic is returned through lan0 by default, and lan0's link goes down.

---

<sup>1</sup> The author also experienced problems with IP Filter hanging when there is asymmetric routing with a heavy load.

## How can I check if my routing is asymmetric?

You'll need a packet-capture software such as wireshark or tcpdump which are open-source products (HP-UX comes bundled with nettl but I don't know how it works). These tools depend on the libpcap library. All of these are available on the Internet Express DVDs or the HP-UX Porting Center. The examples in this document will use tcpdump.

First, run "tcpdump -D" to identify the index numbers of your interfaces.

Example:

```
# tcpdump -D
1.lan0
2.lan1
```

Then run "tcpdump -lni [interface index] | grep [IP Address]" to see if the packets are being sent on the right interface. The three options mean this:

- l : buffered output, to feed grep quickly
- n: don't resolve IP addresses to hostnames
- i: interface index

As an example, to capture all packets with the address 5.6.7.9, on lan0 (interface #1), one would type:

```
# tcpdump -lni 1 | grep 5.6.7.9
```

In this example, we'll do a simple ping that will issue ICMP packets. By alternating tcpdump on lan0 and lan1, you'll see that packets come in on lan1:

```
# tcpdump -lni 2 | grep 5.6.7.9
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lan1, link-type EN10MB (Ethernet), capture size 96 bytes
10:53:52.781438 IP 10.11.12.13 > 5.6.7.9: ICMP echo request, id 512, seq 14336, length 40
10:53:53.781635 IP 10.11.12.13 > 5.6.7.9: ICMP echo request, id 512, seq 14592, length 40
10:53:54.785520 IP 10.11.12.13 > 5.6.7.9: ICMP echo request, id 512, seq 14848, length 40
10:53:55.786487 IP 10.11.12.13 > 5.6.7.9: ICMP echo request, id 512, seq 15104, length 40
38 packets captured
38 packets received by filter
0 packets dropped by kernel
```

Yet they go out on lan0:

```
# tcpdump -lni 1 | grep 5.6.7.9
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lan0, link-type EN10MB (Ethernet), capture size 96 bytes
10:54:35.668922 IP 5.6.7.9 > 10.11.12.13: ICMP echo reply, id 512, seq 15360, length 40
10:54:36.668849 IP 5.6.7.9 > 10.11.12.13: ICMP echo reply, id 512, seq 15616, length 40
10:54:37.669846 IP 5.6.7.9 > 10.11.12.13: ICMP echo reply, id 512, seq 15872, length 40
10:54:38.669819 IP 5.6.7.9 > 10.11.12.13: ICMP echo reply, id 512, seq 16128, length 40
3373 packets captured
3373 packets received by filter
0 packets dropped by kernel
```

If you see packets coming on one interface, and going out another like in this example, then you're doing asymmetric routing.

# Resolving asymmetric routing with `ip_strong_es_model`

## *Introduction to `ip_strong_es_model`*

HP-UX 11iv1 introduced with the TOUR add-on a new ndd option named `ip_strong_es_model` that helps resolve asymmetric routing. This setting is an integral part of 11iv2 onwards.

`Ip_strong_es_model` is not well documented, but the TOUR release notes do give a brief description on what it does:

You can use the ndd parameter `ip_strong_es_model` to enable the default gateway for a physical IPv4 interface. The `ip_strong_es_model` parameter controls support for the strong end-system model. When this parameter is enabled, packet source addresses (and therefore interfaces on a multihomed host) affect selection of a gateway for outbound packets.

The ndd internal help for the parameter also shows that by setting `ip_strong_es_model` to 1, you get the following routing algorithm:

- (A) A host MUST silently discard an incoming datagram whose destination address does not correspond to the physical interface through which it is received.
- (B) A host MUST restrict itself to sending (non-source-routed) IP datagrams only through the physical interface that corresponds to the IP source address of the datagrams.

Okay, so by tuning `ip_strong_es_model`, we should now be able to route packets based not only on the destination address, but the source address as well. Let's give it a try.

## *Setting `ip_strong_es_model` to 1*

First, log in on the console (in case you get disconnected) and set `ip_strong_es_model` to 1:

```
# ndd -set /dev/ip ip_strong_es_model 1
```

For this to work as expected, though, there is catch that seems to be documented only in ITRC doc #NETUXKBRC00005778: You have to add a default route entry for your new interface, like this:

```
# route add net default 5.6.7.1 1
```

This will create this route entry in red:

```
Routing tables
Destination      Gateway          Flags   Refs  Interface  Pmtu
127.0.0.1        127.0.0.1       UH      0     lo0         4136
1.2.3.4          1.2.3.4         UH      0     lan0        4136
```

5.6.7.8	5.6.7.8	UH	0	lan1	4136
5.6.7.9	5.6.7.9	UH	0	lan1:1	4136
1.2.3.0	1.2.3.4	U	3	lan0	1500
5.6.7.0	5.6.7.8	U	3	lan1	1500
5.6.7.0	5.6.7.9	U	3	lan1:1	1500
127.0.0.0	127.0.0.1	U	0	lo0	0
default	5.6.7.1	UG	0	lan1	0
default	1.2.3.1	UG	0	lan0	0

Now that this is done, let's ping 5.6.7.8 (not 5.6.7.9 yet) from 10.11.12.13 and see if the routing is still asymmetric:

```
# tcpdump -lni 2 | grep 5.6.7.8
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lan1, link-type EN10MB (Ethernet), capture size 96 bytes
13:09:31.371581 IP 10.11.12.13 > 5.6.7.8: ICMP echo request, id 512, seq 30976, length 40
13:09:31.371611 IP 5.6.7.8 > 10.11.12.13: ICMP echo reply, id 512, seq 30976, length 40
13:09:32.372442 IP 10.11.12.13 > 5.6.7.8: ICMP echo request, id 512, seq 31232, length 40
13:09:32.372496 IP 5.6.7.8 > 10.11.12.13: ICMP echo reply, id 512, seq 31232, length 40
34 packets captured
34 packets received by filter
0 packets dropped by kernel
```

Nope, we're no longer asymmetric. So setting ip\_strong\_es\_model did the trick.

Up until now, we've only considered lan1. But what about lan1:1? Are they both seen as two different interfaces, or a single and only interface? Let's try again by pinging 5.6.7.9 this time:

```
# tcpdump -lni 2 | grep 5.6.7.9
13:20:27.431884 IP 10.11.12.13 > 5.6.7.9: ICMP echo request, id 512, seq 35328, length 40
13:20:32.758382 IP 10.11.12.13 > 5.6.7.9: ICMP echo request, id 512, seq 35584, length 40
33 packets captured
33 packets received by filter
0 packets dropped by kernel
```

That didn't work. Snooping on interface lan0 does not show any returned packet to 10.11.12.13 as well. Why is that? From my understanding, that's because there is no default gateway through lan1:1 as well. So now comes the BIG question: how can I add a default route that uses lan1:1?

The answer is in the (thinly documented) "source" option to the route command. This option appeared only the man page of HP-UX 11iv1.6, so my guess is that it was either unavailable, or unsupported before that release. The description in the route(1M) man page doesn't clearly tell what it does, but a PHNE patch has been released in Summer 2008 to clarify this (see ITRC doc #ttr\_na-SSB\_1000589089-2).

According to the ITRC document, here's what this option does:

This option is to allow traffic directed to multiple IP addresses assigned to a physical interface to be accepted when Strong ES Model is set to 1.

That looks like our problem, right? So let's make a route using the source option.

```
# route add net default 5.6.7.1 1 source 5.6.7.9
```

Then the following route entry will appear in your routing table:

Routing tables	Destination	Gateway	Flags	Refs	Interface	Pmtu
	127.0.0.1	127.0.0.1	UH	0	lo0	4136
	1.2.3.4	1.2.3.4	UH	0	lan0	4136
	5.6.7.8	5.6.7.8	UH	0	lan1	4136
	5.6.7.9	5.6.7.9	UH	0	lan1:1	4136
	1.2.3.0	1.2.3.4	U	3	lan0	1500

5.6.7.0	5.6.7.8	U	3	lan1	1500
5.6.7.0	5.6.7.9	U	3	lan1:1	1500
127.0.0.0	127.0.0.1	U	0	lo0	0
default	5.6.7.1	UG	0	lan1:1	0
default	5.6.7.1	UG	0	lan1	0
default	1.2.3.1	UG	0	lan0	0

Now when you try your ping test, it should work:

```

topdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lan1, link-type EN10MB (Ethernet), capture size 96 bytes
13:34:07.467910 IP 10.11.12.13 > 5.6.7.9: ICMP echo request, id 512, seq 55808, length 40
13:34:07.467973 IP 5.6.7.9 > 10.11.12.13: ICMP echo reply, id 512, seq 55808, length 40
13:34:08.468548 IP 10.11.12.13 > 5.6.7.9: ICMP echo request, id 512, seq 56064, length 40
13:34:08.468598 IP 5.6.7.9 > 10.11.12.13: ICMP echo reply, id 512, seq 56064, length 40
16 packets captured
16 packets received by filter
0 packets dropped by kernel

```

We've now fixed our asymmetric routing using `ip_strong_es_model` to 1, and setting up appropriate default routes.

## Setting `ip_strong_es_model` to 2

Setting `ip_strong_es_model` to 2 gives us the following routing algorithm, which is more relaxed than when it is set to 1:

- (A) A host MUST NOT silently discard an incoming datagram whose destination address does not correspond to the physical interface through which it is received.
- (B) A host SHOULD restrict itself to sending (non-source-routed) IP datagrams only through the physical interface that corresponds to the IP source address of the datagrams.

The TOUR 2.4 notes also say the following:

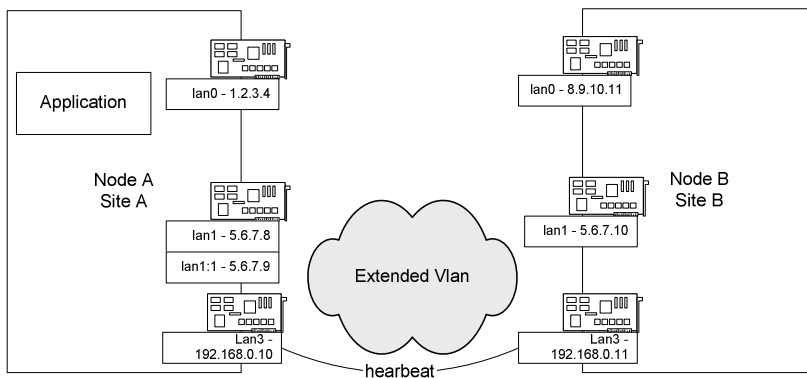
Starting with TOUR 2.4, the `ip_strong_es_model` parameter supports another value, that is 2. If you set `ip_strong_es_model` to 2, the system attempts to match the best gateway (discussed in Case 1) when multiple default gateways are set up on a single IPv4 subnet.

So from my understanding, you would want to set this parameter to 2 if you have multiple default gateways, on different lan cards, in the same subnet.

## *A gotcha with Extended Distance/Metrocluster nodes using the strong ES model*

Some ServiceGuard services use UDP to transmit information between nodes. These include hacl-probe, hacl-hb and hacl-cfg. These UDP packets can cause communication problems between nodes when the strong ES model is set. I don't understand fully what's happening but from what I can see, since there are potentially multiple possible routes between the nodes, sometimes HP-UX will pick the wrong one and UDP packets *might* be returned with a relocatable address instead of the host address. It doesn't happen all the time.

Let's say you have this setup:



Node A and B are on two different sites. They share an extended Vlan, 5.6.7.0/24. They also have another interface which is used for system administration means such as backups. Since we're on different sites, they don't use an extended vlan, their subnets are 1.2.3.4/0 and 8.9.10.11/0, respectively. These subnets are in a different site, so they can't be set as STATIONARY\_IP's and ServiceGuard won't care about them. All inter-node traffic will use 5.6.7.0/24. Heartbeats are also made on a dedicated subnet which is 192.168.0/24.

Node A and B exchange UDP packets for their SG services on 5.6.7.0/24. That in itself is fine. However, when Node B inquires Node A with UDP packets, the replies between Node A and Node B can possibly have their source address set to either 5.6.7.8 or 5.6.7.9. It doesn't happen all the time<sup>2</sup>.

Running cmcheckconf in this situation will give you inconsistency errors, and you could have trouble moving packages as well.

To prevent this, the only workaround I found is to put static routes between my nodes.

---

<sup>2</sup> I'm not sure if this is a ServiceGuard bug, but there's no way I'll inquire on the ITRC with such a setup.

## *Other recommendations*

Laurent Menase mentioned to me in an ITRC post that to have success when using `ip_strong_es_model`, you need two conditions:

- Be up to date with your patches
- Disable default gateway probing

When an interface has secondary IP addresses, probing messages can be stuck when refreshing the gateway in a specific context so it is important to disable gateway probing by setting ndd parameter `ip_ire_gw_probe` to 0. It is already done for you by default if you run Bastille to harden your systems.

From what I've seen in the ITRC forums, there aren't many users who seem to change the `ip_strong_es_model` parameter. So I also strongly suggest that you install only mature PHNE patches, and make a complete test drill each time you add them to your environment. This capability, while supported by HP, does involve being in a specific situation and it could have been overseen by HP when they tested their networking patches.

## *Putting it all together*

### **Making the ndd changes permanent**

To make your changes to `ip_strong_es_model` permanent, you need to edit `/etc/rc.config.d/nddconf` and add an entry that will set it as runtime. For example:

```
TRANSPORT_NAME[10]=ip
NDD_NAME[10]=ip_strong_es_model
NDD_VALUE[10]=1
```

Don't forget to check if gateway probing is already disabled in `nddconf`. If it isn't, you'll have to add it. For example:

```
TRANSPORT_NAME[0]=ip
NDD_NAME[0]=ip_ire_gw_probe
NDD_VALUE[0]=0
```

At startup, the server will issue an "ndd -c" when configuring the network interfaces and this will set the parameter correctly<sup>3</sup>.

### **Making new default gateways permanent**

Gateways that use stationary interfaces on your hosts can be easily made permanent by adding them in `/etc/rc.config.d/netconf`, like this:

```
ROUTE_DESTINATION[1]="default"
```

---

<sup>3</sup> There used to be a maximum of 10 parameters possible in `nddconf` due to a limit of `ndd -c`, but this seems to have been fixed years ago

```
ROUTE_MASK[1]=" "  
ROUTE_GATEWAY[1]="5.6.7.8"  
ROUTE_COUNT[1]="1"  
ROUTE_ARGS[1]=" "
```

However, you won't be able to do this for your relocatable ServiceGuard IP addresses. This is caused by the fact that these floating addresses are never available at boot time.

The only workaround I could find is adding the route commands directly in `customer_defined_run_cmds` (legacy packages) or the `external_script` (modular packages). You will therefore need to do something like this before starting the application:

```
echo "Setting up a default route through lan1:1"  
route add net default 5.6.7.1 1 source 5.6.7.9
```

And this when stopping it:

```
echo "Dropping the route through lan1:1 "  
route delete net default 5.6.7.1 1 source 5.6.7.9
```

## *What about running gated?*

Network-savvy administrators might want to run a routing daemon such as `gated(1M)` on a server to do more advanced routing. However, in the scope of this document, we're dealing with **server** and not a **router**. We do not, for instance, want our multi-homed server to start learning about possible routes and acting as a router. To my knowledge, few people still use general-purpose Unix servers (let alone HP-UX servers) as routers and this task should be left to your network administrator.

I've preferred concentrating my efforts on routing tables instead. If anyone has a compelling reason to use `gated`, and would like to share his/her experiences, please drop me an e-mail and I'll be glad to add the information to this paper.

## Conclusion

In this document, we've explored the basics of HP-UX's routing with multi-homed servers, especially with HA applications that use a relocatable IP address. Binding applications to a specific IP address, when possible is the first step to resolving potential problems. For other applications, static routes or NAT can be used to some extent, but these techniques have some drawbacks.

We've also seen that asymmetric routing can be experienced on multi-homed servers. Tcpdump is a handy tool to find out how routing is effectively handled on your servers. The ndd parameter `ip_strong_es_model` and adding default gateways on your other interfaces can be used to limit asymmetric routing.

# References

TOUR 2.4 Release Notes: <http://docs.hp.com/en/5991-0782/ch01s02.html>

IPFilter NAT manual page: ipnat(5) and IP Filter documentation

ITRC Doc ID #ttr\_na-SSB\_1000589089-2

ITRC Doc ID #NETUXKBRC00005778

Route(1m) and routing(7) manual pages

Overcoming Asymmetric Routing on multi-homed servers:

<http://www.linuxjournal.com/article/7291>

Managing ServiceGuard, Fifteenth Edition, reprinted May 2008

Thanks to Rick Jones and Laurent Menase who, through their posts in the ITRC forums, helped clear things up for me. Rick also provided valuable comments that resulted in a revised version of this paper.