Networking with Linux® on System z
(Part 2 of 2)

Session L42
IBM System z9 and zSeries Expo Orlando 2006
Oct. 9-13

Ursula Braun (braunu@de.ibm.com)
IBM Development Lab, Boeblingen, Germany
Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

Enterprise Storage Server
ESCON*

FICON
FICON Express

HiperSockets
IBM*

IBM logo*
IBM eServer

Netfinity*
S/390*

VM/ESA*
WebSphere*

z/VM
zSeries

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Intel is a trademark of the Intel Corporation in the United States and other countries.

Oracle 9i is a trademark of the Oracle Corporation in the United States and other countries.

Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries.

Lotus, Notes, and Domino are trademarks or registered trademarks of Lotus Development Corporation.

Linux is a registered trademark of Linus Torvalds

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

Penguin (Tux) compliments of Larry Ewing.

SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

UNIX is a registered trademark of The Open Group in the United States and other countries.

* All other products may be trademarks or registered trademarks of their respective companies.
Agenda

- Router setup for Linux on System z
- Failover and availability solutions:
  - Virtual IP Addresses (VIPA)
  - Channel Bonding (layer2 mode)
  - IP Address Takeover
  - Proxy ARP
- The qethconf tool
- The qetharp tool
- HiperSockets Network Concentrator (HSNC)
- SNMP support: osasnmpd
Routing Types

<table>
<thead>
<tr>
<th>zSeries</th>
<th>LPAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>LG10</td>
<td>LH10</td>
</tr>
</tbody>
</table>

```
|          | eth0 10.2.1.10 | eth1 10.2.1.1 | eth0 10.1.1.1 | hsi0 10.3.1.1 |
| NIC      | B00A-B00C     | B000-B002    | A000-A002     | C000-C002     |
```

- GuestLAN (Type QDIO) 10.2.0.0
- HiperSockets 10.3.0.0

```
|          | et 0 10.2.1.10 |
| NIC      | B00A-B00C     |
```

|          | et 1 10.2.1.1 |
| NIC      | B000-B002    |

|          | et 0 10.1.1.1 |
| NIC      | A000-A002    |

|          | hsi 0 10.3.1.1 |
| iQDIO    | C000-C002    |

- OSA Express
- VM GuestLAN
- HiperSockets

| primary router | X |
| secondary router | X |
| multicast router | X |
| primary connector | Hiper (IPv4 only) |
| secondary connector | Hiper (IPv4 only) |
How OSA handles Router Settings

Ethernet packet

<table>
<thead>
<tr>
<th>Dest. MAC</th>
<th>Src MAC</th>
<th>prot</th>
<th>...</th>
<th>Src IP</th>
<th>Dest. IP</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ethernet Header

IP Header

Packet reaches OSA Card with destination MAC address

OSA Card

Is the packet's destination IP address set by any client OS image?

- YES
  - Is there a primary router?
    - YES
      - Give packet to OS image
    - NO
      - Is there a secondary router?
        - YES
        - NO

- NO

OSA Address Table (OAT)

<table>
<thead>
<tr>
<th>Device Number</th>
<th>IP Address</th>
<th>Router?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000</td>
<td>10.1.1.1</td>
<td>primary</td>
</tr>
<tr>
<td>A00A</td>
<td>10.1.1.10</td>
<td>secondary</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>A0AA</td>
<td>10.1.1.100</td>
<td></td>
</tr>
</tbody>
</table>

discard packet

Linux

IP packet

<table>
<thead>
<tr>
<th>...</th>
<th>Src IP</th>
<th>Dest. IP</th>
<th>payload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A multicast router also receives all multicast packets
How HiperSockets handles Router Settings

IP packet

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dest. IP</th>
<th>payload</th>
</tr>
</thead>
</table>

IP Header

OS Image sends IP packet via its iQDIO device

HiperSockets Microcode

Is the packet's destination IP address set by any client OS image?

<table>
<thead>
<tr>
<th>Device Number</th>
<th>IP Address</th>
<th>Connector?</th>
</tr>
</thead>
<tbody>
<tr>
<td>C000</td>
<td>10.3.1.1</td>
<td>primary</td>
</tr>
<tr>
<td>C0A0</td>
<td>10.3.1.30</td>
<td>secondary</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>C0AA</td>
<td>10.1.1.100</td>
<td></td>
</tr>
</tbody>
</table>

GuestLAN is analogous

Is there a primary connector?

Is there a secondary connector?

Copy packet to OS image storage

discard packet

Linux

<table>
<thead>
<tr>
<th>IP packet</th>
<th>...</th>
<th>Src IP</th>
<th>Dest. IP</th>
<th>payload</th>
</tr>
</thead>
</table>

Setting up a Router

1. Enable IP forwarding:

```
# > sysctl -w net.ipv4.conf.all.forwarding=1
# > sysctl -w net.ipv6.conf.all.forwarding=1
```

or enter the following lines in `/etc/sysconfig/sysctl`:

```
IP_FORWARD="yes"
IPV6_FORWARD="yes"
```

to keep the setting persistent.
Setting up a Router (cont.)

2. Set the router status for your device, e.g. eth0 of the Router (see above):

```bash
#> echo primary_router > /sys/class/net/eth0/device/route4
#> echo primary_router > /sys/class/net/eth0/device/route6
```

or enter the following line in the appropriate SuSE SLES9 hwcfg-file to keep the setting persistent:

```bash
QETH_OPTIONS='route4=primary_router
route6=primary_router'
```

*) Other possible values: secondary_router
multicast_router
primary_connector
secondary_connector
no_router (to reset)
Querying the Router Status

Router status is displayed in `/proc/qeth`:

```
#> cat /proc/qeth
```

<table>
<thead>
<tr>
<th>devices</th>
<th>CHPID</th>
<th>interface</th>
<th>cardtype</th>
<th>...</th>
<th>rtr4</th>
<th>rtr6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.a000/0.0.a001/0.0.a002</td>
<td>xA0</td>
<td>eth0</td>
<td>OSD_1000</td>
<td></td>
<td>pri</td>
<td>pri</td>
</tr>
<tr>
<td>0.0.b000/0.0.b001/0.0.b002</td>
<td>x01</td>
<td>eth1</td>
<td>GuestLAN QDIO</td>
<td></td>
<td>pri</td>
<td>pri</td>
</tr>
<tr>
<td>0.0.c000/0.0.c001/0.0.c002</td>
<td>xC0</td>
<td>hsi0</td>
<td>HiperSockets</td>
<td></td>
<td>p+c</td>
<td>no</td>
</tr>
</tbody>
</table>

or can be retrieved from sysfs:

```
#> cat /sys/class/net/eth0/device/route4
#> cat /sys/devices/qeth/0.0.a000/route6
```

note the alternative ways to your device

*) All possible values:

<table>
<thead>
<tr>
<th>procfs</th>
<th>sysfs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pri</td>
<td>primary</td>
<td>Primary Router</td>
</tr>
<tr>
<td>sec</td>
<td>secondary</td>
<td>Secondary Router</td>
</tr>
<tr>
<td>mc</td>
<td>multicast</td>
<td>Multicast Router</td>
</tr>
<tr>
<td>mc+</td>
<td>multicast+</td>
<td>Multicast Router with broadcast filtering</td>
</tr>
<tr>
<td>p.c</td>
<td>primary</td>
<td>Primary Connector</td>
</tr>
<tr>
<td>p+c</td>
<td>primary+</td>
<td>Primary Connector with broadcast filtering</td>
</tr>
<tr>
<td>s.c</td>
<td>secondary</td>
<td>Secondary Connector</td>
</tr>
<tr>
<td>s+c</td>
<td>secondary+</td>
<td>Secondary Connector with broadcast filtering</td>
</tr>
</tbody>
</table>
Virtual LAN (VLAN) support

- Risk of big switched LANs: flooded with broadcast traffic
- Devide LANs logically into subnets
  >>> fewer waste of bandwidth
- IEEE Standard 802.1Q
Virtual LAN (VLAN) support (cont.)

- Setup:

```
ifconfig eth1 9.164.160.23 netmask 255.255.224.0
vconfig add eth1 3
ifconfig eth1.3 1.2.3.4 netmask 255.255.0.0
```

- Displaying info:

```
cat /proc/net/vlan/config
VLAN Dev name   | VLAN_ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD
eth1.3           | 3    | eth1
```

- Implemented:
  VLAN tag, added to packets transmitted

- Supported by:
  real OSA-card, z/VM Guest LAN, z/VM VSWITCH
Virtual IP Addresses

- Minimize outage due to adapter or network failure
- Bind server applications to system-wide virtual IP addresses (instead of adapter specific addresses)
- Server can be reached via different routes

zSeries

Linux Application Server

Application Server "appservd"

dummy0
VIPA = 10.1.1.1

eth0
10.2.1.1

eth1
10.3.1.1

OSA A000-A002

OSA B000-B000

OSA Express

OSA Express

OSA ADDRESS TABLE

<table>
<thead>
<tr>
<th>IP Addr</th>
<th>Image</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1.1.1</td>
<td>LINUX1</td>
<td>vipa</td>
</tr>
<tr>
<td>10.3.1.1</td>
<td>LINUX1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LAN 10.3.0.0

LAN 10.2.0.0

Router

Client
Virtual IP Addresses (cont.)

- VIPAs are **registered with every physical adapter** usable to reach a system

- Inbound use of VIPA:
  - Setting a VIPA on an OSA card ensures that packets are handed to Linux image (and are not discarded)
  - Linux network stack forwards received packets to configured virtual device (e.g. dummy0)

- For outbound use of VIPA the SOURCE VIPA package is required (available at DeveloperWorks)
  - Linux user space layer between application and kernel network stack
  - Sets virtual IP address as source address for sent packets (normally packets get source IP of the interface via which they leave the system)
Virtual IP Address Device Attributes

/sys
|-- devices
 ||-- qeth
 | ||-- 0.0.<devno>
 | ||-- vipa
 | | ||-- add4
 | | ||-- add6
 | | ||-- del4
 | |||-- del6

VIPA configuration is done per device via sysfs attributes

add/display IPv4 VIPAs
add/display IPv6 VIPAs
delete IPv4 VIPAs
delete IPv6 VIPAs
Virtual IP Address Setup

1. Create a virtual interface and assign the VIPA using a dummy interface:

```bash
# > modprobe dummy
# > ifconfig dummy0 10.1.1.1 netmask 255.255.0.0
```

or using an interface alias:

```bash
# > ifconfig eth0:1 10.1.1.1 netmask 255.255.0.0
```

2. Register the virtual IP address with physical devices:

```bash
# > echo 10.1.1.1 > /sys/class/net/eth0/device/vipa/add4
# > echo 10.1.1.1 > /sys/class/net/eth1/device/vipa/add4
```

3. On the router add a route to the routing table:

```bash
# > route add -host 10.1.1.1 gw 10.2.1.1 if LAN1 works
# > route add -host 10.1.1.1 gw 10.3.1.1 if LAN2 works
```

or, better, configure the routes with a dynamic routing daemon (e.g. quagga: http://quagga.net).
Virtual IP Address Setup (cont.)

- VIPA settings can be checked by reading the `add4` attribute:

  ```bash
  #> cat /sys/class/net/eth0/device/vipa/add4
  10.1.1.1
  #> cat /sys/class/net/eth1/device/vipa/add4
  10.1.1.1
  ```

  (or make use of the qethconf tool)

- VIPAs are deleted by writing to the `del4` attribute:

  ```bash
  #> echo 10.1.1.1 > /sys/class/net/eth0/device/vipa/del4
  #> echo 10.1.1.1 > /sys/class/net/eth1/device/vipa/del4
  ```

IPv6 is analogous, using the `add6` and `del6` attributes.
Channel Bonding – Virtual IP Addresses with Layer 2

- The Linux bonding driver provides a method for aggregating multiple network interfaces into a single logical "bonded" interface
- provides failover and / or load-balancing functionality
- better performance depending on bonding mode
- transparent for LAN infrastructure
- latest setup description:

http://sourceforge.net/projects/bonding/
Channel bonding setup

- Add MAC address to eth0 & eth1 (not necessary for GuestLAN)

  ```
  #> ifconfig eth0 hw ether 00:06:29:55:2A:01
  #> ifconfig eth1 hw ether 00:05:27:54:21:04
  ```

- Load bonding module with miimon option
  (otherwise bonding will not detect link failures)

  ```
  #> modprobe bonding miimon=100 mode=balance-rr
  ```

- Bring up bonding device bond0

  ```
  #> ifconfig bond0 10.1.1.1 netmask 255.255.255.0
  ```

- connect eth0 & eth1 to bond0

  ```
  #> ifenslave bond0 eth0
  #> ifenslave bond0 eth1
  ```
Channel bonding setup (SLES9 – config files)

- interface configuration file for a slave

```
/etc/sysconfig/network/ifcfg-qeth-bus-ccw-0.0.a000
BOOTPROTO='static'
IPADDR=''
SLAVE='yes'
STARTMODE='onboot'
```

- interface configuration file for a master

```
/etc/sysconfig/network/ifcfg-bond0
BOOTPROTO='static'
BROADCAST='10.1.255.255'
IPADDR='10.1.1.1'
NETMASK='255.255.0.0'
NETWORK='10.1.0.0'
STARTMODE='onboot'

BONDING_MASTER='yes'
BONDING_MODULE_OPTS='mode=1 miimon=1'
BONDING_SLAVE0='qeth-bus-ccw-0.0.a000'
BONDING_SLAVE1='qeth-bus-ccw-0.0.b000'
```
Channel bonding setup (cont.)

```bash
# > ifconfig
bond0 Link encap:Ethernet  HWaddr 00:06:29:55:2A:01
    inet addr:10.1.1.1  Bcast:10.255.255.255 . . .
eth0 Link encap:Ethernet  HWaddr 00:06:29:55:2A:01
    UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500 . . .
eth1 Link encap:Ethernet  HWaddr 00:06:29:55:2A:01
    UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500 . . .
```

```bash
# > cat /proc/net/bonding/bond0

Bonding Mode: load balancing (round-robin)
MII Status: up
MII Polling Interval (ms): 100

Slave Interface: eth0
MII Status: up
Permanent HW addr: 00:06:29:55:2A:01

Slave Interface: eth1
MII Status: up
Permanent HW addr: 00:05:27:54:21:04
```
IP Address Takeover

Enables implementation of failover strategies

Server2 takes over role of Server1 in case of connectivity problems
IP Address Takeover

- Idea of IP Address Takeover:
  - Setting an IP address on Linux always succeeds
  - No failure due to ARP (Address Resolution Protocol) conflicts
  - On shared OSA cards, HiperSockets or GuestLAN IP addresses are really taken away from previous owner (in OSA Address Table)

- Takeover must be enabled on both systems

- Takeover of IP addresses is secured by keys:
  Only systems with same key can take over their IP addresses
IP Address Takeover Details

- Server images are monitored and controlled by a switch image
- In case of problems the switch initiates takeover
  - Disable failing server
  - Backup server takes over the IP address of the failing server
- Management of server nodes done, e.g. via open source Heartbeat framework of the High-Availability Linux Project http://linux-ha.org
- Part of Heartbeat: STONITH ("Shoot The Other Node In The Head") (http://linux-ha.org/stonith.html)
- Adaptation of STONITH to fit actual target systems through plugins
IP Address Takeover Details (cont.)

- STONITH plugin for Linux on System z:
  - lic_vps (Linux Image Control - Virtual Power Switch)
  - part of snIPL (simple network IPL)

- snIPL is a Linux image control tool for LPAR and z/VM
  - Can boot, stop, reset Linux images, send and receive OS messages

- On LPAR
  - Uses management application programming interfaces (APIs) of HMC/SE (Hardware Management Console/Support Element)
  - communicates via SNMP (Simple Network Management Protocol)

- On z/VM
  - Utilizes system managements APIs of z/VM 4.4 (or higher)
  - Communicates with z/VM host via RPC (Remote Procedure Call) over internal network connections
IP Address Takeover Device Attributes

```
/sys
|--devices
 ||--qeth
  ||--0.0.<devno>
   ||--ipa_takeover
    |--add4
    |--add6
    |--del4
    |--del6
    |--enable
    |--invert4
    |--invert6
```

IP Address Takeover configuration is done per device via sysfs attributes.
**IP Address Takeover Setup**

1. Enable IP Address Takeover for your device by adding the following line to the appropriate hwcfg-file:

```
QETH_IPA_TAKEOVER=1
```

- Takeover will be enabled the next time the Linux Image is booted.

To activate the setting on a running Linux:

- Stop the device:
  ```
  $ hwdown qeth-bus-ccw-0.0.a000
  ```

- Re-initialize device with new settings:
  ```
  $ hwup qeth-bus-ccw-0.0.a000
  ```

- To activate the setting, execute:
  ```
  $ echo 1 > /sys/devices/qeth/0.0.a000/ipa_takeover/enable
  $ echo 0 > /sys/bus/ccwgroup/devices/0.0.a000/online
  ```

- To activate the setting, execute:
  ```
  $ echo 1 > /sys/bus/ccwgroup/devices/0.0.a000/online
  ```
2. Specify address ranges for IP Address Takeover

For example, to handle all IP addresses in the range 10.1.1.1 to 10.1.1.254 in takeover mode, issue:

```bash
# > echo <IP address>/<mask> > /sys/class/net/eth0/device/ipa_takeover/add4
```

- Ranges can be inverted, i.e. all addresses except those in a specified range are handled in takeover mode:

```bash
# > echo 10.1.1.0/24 > /sys/class/net/eth0/device/ipa_takeover/add4
```

```bash
# > echo 1 > /sys/class/net/eth0/device/ipa_takeover/invert4
```
IP Address Takeover Setup (cont.)

3. To check current IP Address Takeover ranges, issue

```bash
#> cat /sys/class/net/eth0/device/ipa_takeover/add4
10.1.1.0/24
10.1.2.0/25
```

- All IP addresses from 10.1.1.1 to 10.1.1.254 and from 10.1.2.1 to 10.1.2.127 are handled in takeover mode when being set with 'ifconfig' or 'ip addr add'

4. Takeover ranges can be deleted by writing to the del4 attribute:

```bash
#> echo 10.1.2.0/25 >
/sys/class/net/eth0/device/ipa_takeover/del4
```

IPv6 is analogous, i.e. add/delete ranges via add6/de6
Proxy ARP

Example: Integration of a virtual LAN into a real LAN

Client and Linux images are all in subnet 10.1.0.0
Proxy ARP Details

- **OSA card handles ARP requests** for all configured Proxy ARP addresses
- Gratuitous ARP packets are sent out to advertise Proxy ARP addresses
- Completely **transparent to outside clients**
- Seamless integration of arbitrary virtual networks (HiperSockets, GuestLAN, IUCV, virtual CTC) into a real LAN
Proxy ARP Device Attributes

```
/sys
|--devices
  |--qeth
    |--0.0.<devno>
    |--rxip
      |--<add4
      |--add6
      |--<del4
      |--<del6
```

Proxy ARP configuration is done per device via sysfs attributes

- add/display IPv4 Proxy ARP entries
- add/display IPv6 Proxy ARP entries
- delete IPv4 Proxy ARP entries
- delete IPv6 Proxy ARP entries
Proxy ARP Setup

1. Create Proxy ARP entries for leaf nodes (i.e. with no own connection to outside LAN):

```bash
#> echo 10.1.1.2 > /sys/class/net/eth0/device/rxip/add4
#> echo 10.1.1.3 > /sys/class/net/eth0/device/rxip/add4
#> echo 10.1.1.4 > /sys/class/net/eth0/device/rxip/add4
...
```

2. Proxy ARP settings can be checked by reading `add4` attribute:

```bash
#> cat /sys/class/net/eth0/device/rxip/add4
10.1.1.2
10.1.1.3
10.1.1.4
...
```
Proxy ARP Setup (cont.)

- Proxy ARP entries are deleted by writing to the `del4` attribute:

```
# > echo 10.1.1.2 > /sys/class/net/eth0/device/rxip/del4
# > echo 10.1.1.3 > /sys/class/net/eth0/device/rxip/del4
# > echo 10.1.1.4 > /sys/class/net/eth0/device/rxip/del4
...
```

IPv6 is analogous, using the `add6` and `del6` attributes.
The qethconf Tool

- Command line utility to configure IP Address Takeover, VIPA and Proxy ARP
- Part of the **s390-tools** package available at DeveloperWorks
- **Consistent user interface on Linux 2.4 and 2.6**

<table>
<thead>
<tr>
<th>Linux 2.4</th>
<th>Linux 2.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>qethconf <strong>script</strong></td>
<td></td>
</tr>
<tr>
<td><code>/proc/qeth_ipa_takeover</code></td>
<td><code>/sys/.../qeth/0.0.*/ipa_takeover</code></td>
</tr>
<tr>
<td></td>
<td><code>/sys/.../qeth/0.0.*/rxip</code></td>
</tr>
<tr>
<td></td>
<td><code>/sys/.../qeth/0.0.*/vipa</code></td>
</tr>
</tbody>
</table>

- **Especially useful:**
  - list commands iterate over all devices found in Linux 2.6 sysfs
qethconf – Examples

- IP Address Takeover configuration:
  
  ```bash
  #> qethconf ipa add 10.1.1.0/24 eth0
  ```

- Proxy ARP configuration:
  
  ```bash
  #> qethconf rxip add 10.1.1.2 eth0
  ```

- VIPA configuration:
  
  ```bash
  #> qethconf vipa add 10.1.1.1 eth0
  #> qethconf vipa add 10.1.1.1 eth1
  #> qethconf vipa list
  vipa add 10.1.1.1 eth0
  vipa add 10.1.1.1 eth1
  ```

- Display all IP Address Takeover, VIPA and Proxy ARP settings:
  
  ```bash
  #> qethconf list_all
  ipa add 10.1.1.0/24 eth0
  ipa add 10.1.1.0/24 eth1
  rxip add 10.1.1.2 eth0
  vipa add 10.1.1.1 eth0
  vipa add 10.1.1.1 eth1
  ```
The qetharp Tool

- Command line utility to query the ARP cache of OSA and HiperSockets devices
- Part of **s390-tools** package available on DeveloperWorks
- Query of OSA devices returns real ARP cache and entries of local OSA Address Table (OAT)
- Query of HiperSockets devices returns entries of address table associated to a HiperSockets CHPID, i.e. all IP addresses that are currently set on that CHPID
- Currently not supported on VM GuestLAN
- Not working on hosts running with layer2 – use arp instead
### qetharp – Examples

#### Query ARP cache of an OSA Express card:

```bash
# > qetharp -nq eth1
Address     HWaddress          HWTtype  Iface
10.30.151.31 02:00:00:00:29:29 ether  eth1
10.30.30.13   00:09:6b:1a:0c:ed ether  eth1
10.30.30.7    00:09:6b:1a:0c:ed ether  eth1
10.30.130.17  00:09:6b:1a:0c:ed ether  eth1
10.30.30.9    00:09:6b:1a:0c:ed ether  eth1
10.30.130.111 00:09:6b:1a:0c:ed ether  eth1
```

#### Query address table of a HiperSockets CHPID:

```bash
# > qetharp -nq hsi0
Address     HWaddress          HWTtype  Iface
10.1.3.1     hiper              hsi0
10.1.3.2     hiper              hsi0
10.1.3.8     hiper              hsi0
10.1.3.10    hiper              hsi0
```
HSNC – Topology Example 1

**OSA ADDRESS TABLE**

<table>
<thead>
<tr>
<th>IP Addr</th>
<th>Image Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1.1.1</td>
<td>LINUX1</td>
</tr>
<tr>
<td>10.1.1.2</td>
<td>LINUX1 parp</td>
</tr>
<tr>
<td>10.1.1.3</td>
<td>LINUX1 parp</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>10.1.1.10</td>
<td>LINUX1 parp</td>
</tr>
</tbody>
</table>

**zSeries**

- **LPAR1**
  - LINUX1 (Connector)
  - mc router
  - pri. conn.
  - eth0 10.1.1.1
  - hsi0 0.0.0.0
  - OSA A000-A002
  - iQDIO B000-B002

- **LPAR2**
  - LINUX2
  - hsi0 10.1.1.2
  - iQDIO B010-B012

- **LPAR10**
  - LINUX10
  - hsi0 10.1.1.10
  - iQDIO B0A0-B0A2

**HiperSockets**

- **LAN** 10.1.0.0
HSNC – Topology Example 2

Different mainframes or XCEC

```
zSeries
    LPAR1
    | LINUX11 (Connector)
    | mc router
    | pri. conn.
    | eth0
    | 10.1.1.100
    | OSA
    | C000-C002
    | iQDIO
    | D000-D002
    | hsi0
    | 0.0.0.0
    | OSA
    | Express
    | HiperSockets
    | LAN 10.1.0.0

zSeries
    LPAR1
    | LINUX1 (Connector)
    | mc router
    | pri. conn.
    | eth0
    | 10.1.1.1
    | OSA
    | A000-A002
    | iQDIO
    | B000-B002
    | hsi0
    | 0.0.0.0
    | OSA
    | Express
    | HiperSockets

zSeries
    LPAR10
    | LINUX20
    | mc router
    | pri. conn.
    | hsi0
    | 10.1.1.110
    | iQDIO
    | D0A0-D0A2

zSeries
    LPAR10
    | LINUX10
    | mc router
    | pri. conn.
    | hsi0
    | 10.1.1.10
    | iQDIO
    | B0A0-B0A2

...
**HiperSockets Network Concentrator**

- **What is HSNC:**
  - A combination of tools for enhanced HiperSockets connectivity
  - Part of **s390-tools** package available at DeveloperWorks
- Enables automatic integration of nodes in a HiperSockets network into an external LAN (i.e. one IP subnet)
- Enables creation of IP subnets across multiple HiperSockets networks on different CECs (Central Electronic Complex) “XCEC HiperSockets”
- Except for connector nodes, completely transparent for attached operating system images
- **Simplification of network topologies** and server consolidation efforts
- Does not work with OSA layer2 at the moment
HSNC – How it works

Linux Connector System

- qethconf
- xcec-bridge
  IPv4 multicast and broadcast forwarding
- QETH Device Driver
- ipWatcher.pl
  add/remove routing entries for "leaf" nodes
- qetharp
  monitor address table
- start_hsnc.sh
  startup cleanup

Linux Network Stack
- IPv4 unicast forwarding

OSA Express
- pri./mc router

iQDIO
- pri./sec. connector

external LAN

HiperSockets network
HSNC – Setup of Connector System

1. Configure your OSA device as multicast router: *

```bash
#> echo multicast_router > /sys/class/net/eth0/device/route4
```

If no multicast forwarding is desired, use `primary_router`

2. Configure your HiperSockets device as primary connector: *

```bash
#> echo primary_connector > /sys/class/net/hsi0/device/route4
```

* Optional: You can configure a backup Connector System for failover strategies. Use `secondary_router` for the OSA device and `secondary_connector` for the HiperSockets device on the backup system.
HSNC – Setup of Connector System (cont.)

3. Check the routing configuration:

```bash
#> cat /proc/qeth devices
+-----------------+----------+-----------+----------------+---+---
| CHPID | interface | cardtype  | rtr4 | rtr6 |
+-----------------+----------+-----------+----------------+---+---
| 0.0.a000/0.0.a001/0.0.a002 | xA0 | eth0 | OSD_1000 | mc+ | no |
| 0.0.b000/0.0.b001/0.0.b002 | xB0 | hsi0 | HiperSockets | p+c | no |
```

The '+' sign indicates broadcast filtering capability. This is required for broadcast forwarding.

4. Enable IP forwarding:

```bash
#> sysctl -w net.ipv4.ip_forward=1
```

5. Start HSNC:

```bash
#> start_hsnce.sh
```

The OSA device can be specified as start option. This enables unicast forwarding only.
SNMP support: osasnmudp

- **SNMP** support:
  - osasnmudp

**Diagram:***
- **Network Management Station**
- **NMS**
- **SNMP managed Linux node**
- **snmpd** (master agent)
- **osasnmudp** (subagent)
- **AgentX**
- **requests responses**
  - register OIDs
  - SNMP request
  - response data
- **OSA-Express feature**
- **OSA-Express feature**
SNMP support – osasnmpd (cont.)

- Download IBM OSA-Express MIB from ibm.com/servers/resourcelink
- Tailor access control definitions for master agent
- Start master agent plus osasnmpd subagent

```
# > rcsnmpd start
```

- Check log files

```
# > cat /var/log/snmpd.log
...
# > cat /var/log/osasnmpd.log
```

- Issue queries with snmpget / snmpwalk

```
# > snmpwalk -OS localhost ibmOsaExpEthPortType.6
IBM-OSA-MIB::ibmOsaExpEthPortType.6 = INTEGER: fastEthernet(81)
```
References

- Linux for zSeries and S/390 on DeveloperWorks

- Linux for zSeries and S/390 Device Drivers, Features and Commands

- Linux for zSeries and S/390, useful add-ons
    - snIPL
    - src_vipa

- Linux High-Availability Project
  http://linux-ha.org