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 Thorvalds

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.

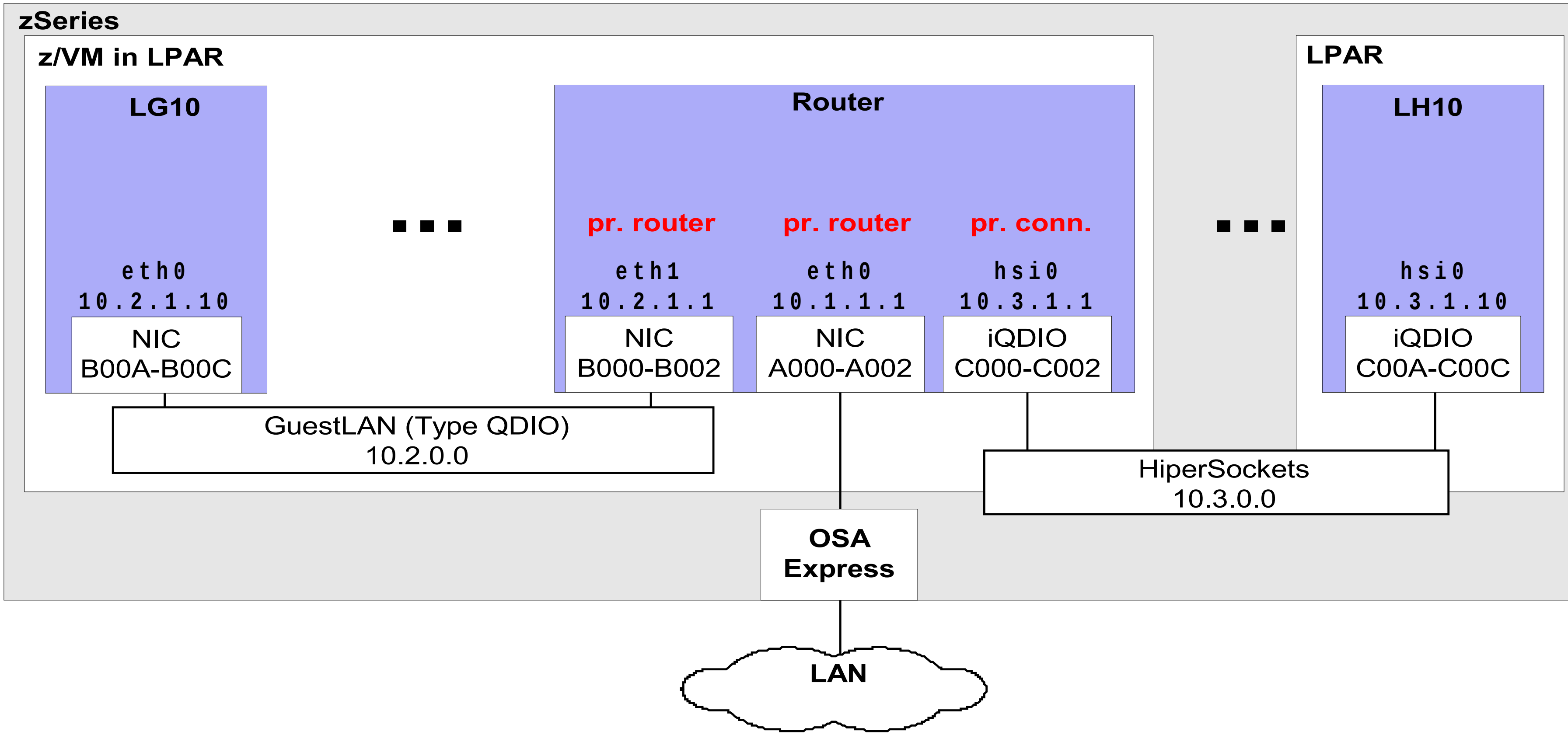
UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Agenda

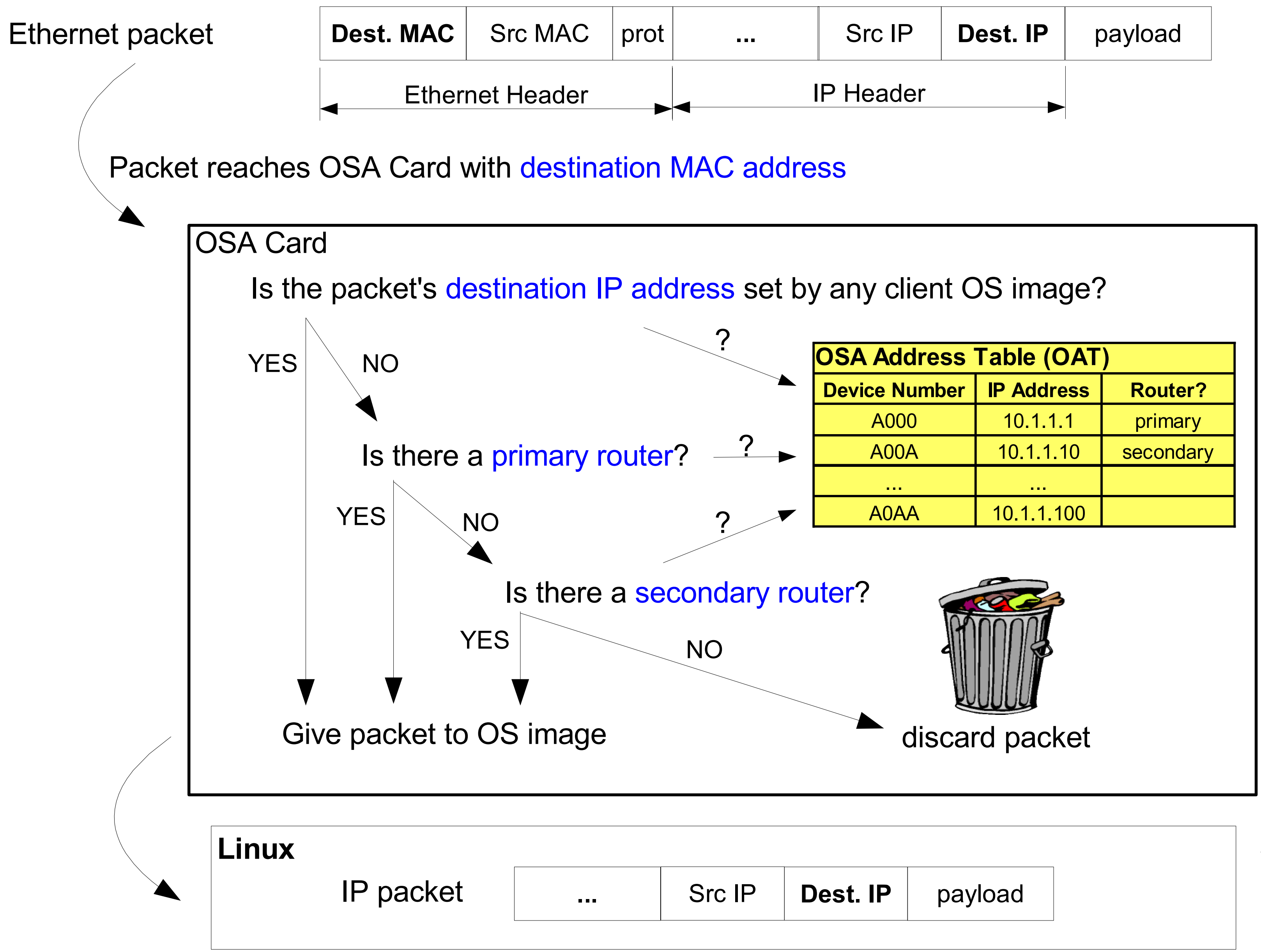
- Router setup for Linux on System z
- Failover and availability solutions:
  - Virtual IP Addresses (VIPA)
  - Channel Bonding (layer2 mode)
  - IP Address Takeover
  - Proxy ARP
- The qethconf tool
- The qetharp tool
- HiperSockets Network Concentrator (HSNC)
- SNMP support: osasnmpd

Routing Types



	OSA Express	VM GuestLAN QDIO	Hiper	HiperSockets
primary router	X	X (IPv4 only)		
secondary router	X	X (IPv4 only)		
multicast router	X			
primary connector				X
secondary connector				X

How OSA handles Router Settings



OSA Address Table (OAT)

Device Number	IP Address	Router?
A000	10.1.1.1	primary
A00A	10.1.1.10	secondary
...	...	
A0AA	10.1.1.100	

A **multicast router** also receives **all** multicast packets

How HiperSockets handles Router Settings

IP packet

...	Src IP	**Dest. IP**	payload

IP Header

GuestLAN is analogous

OS Image sends IP packet via its iQDIO device

HiperSockets Microcode

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

YES

NO

?

Address Table		
Device Number	**IP Address**	**Connector?**
C000	10.3.1.1	primary
C0A0	10.3.1.30	secondary
...	...	
C0AA	10.1.1.100	

Is there a primary connector?

?

YES

NO

?

Is there a secondary connector?

YES

NO

Copy packet to OS image storage

discard packet

Linux

IP packet

...	Src IP	**Dest. IP**	payload

Setting up a Router

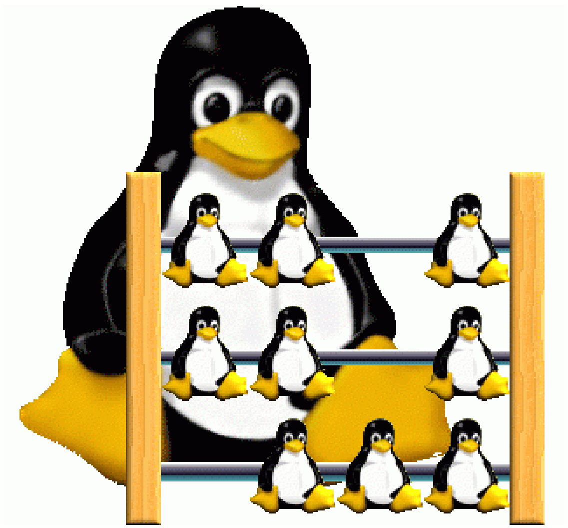
1. Enable IP forwarding:

```
#> 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):

```
                                                              *
#> 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:

```
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                                                          *
devices                    CHPID interface  cardtype       ... rtr4 rtr6
-------------------------- ----- ---------- --------------     ---- ---- ...
0.0.a000/0.0.a001/0.0.a002 xA0   eth0       OSD_1000           pri  pri
0.0.b000/0.0.b001/0.0.b002 x01   eth1       GuestLAN QDIO      pri  pri
0.0.c000/0.0.c001/0.0.c002 xC0   hsi0       HiperSockets       p+c  no
```

or can be retrieved from sysfs:

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

note the alternative ways to your device

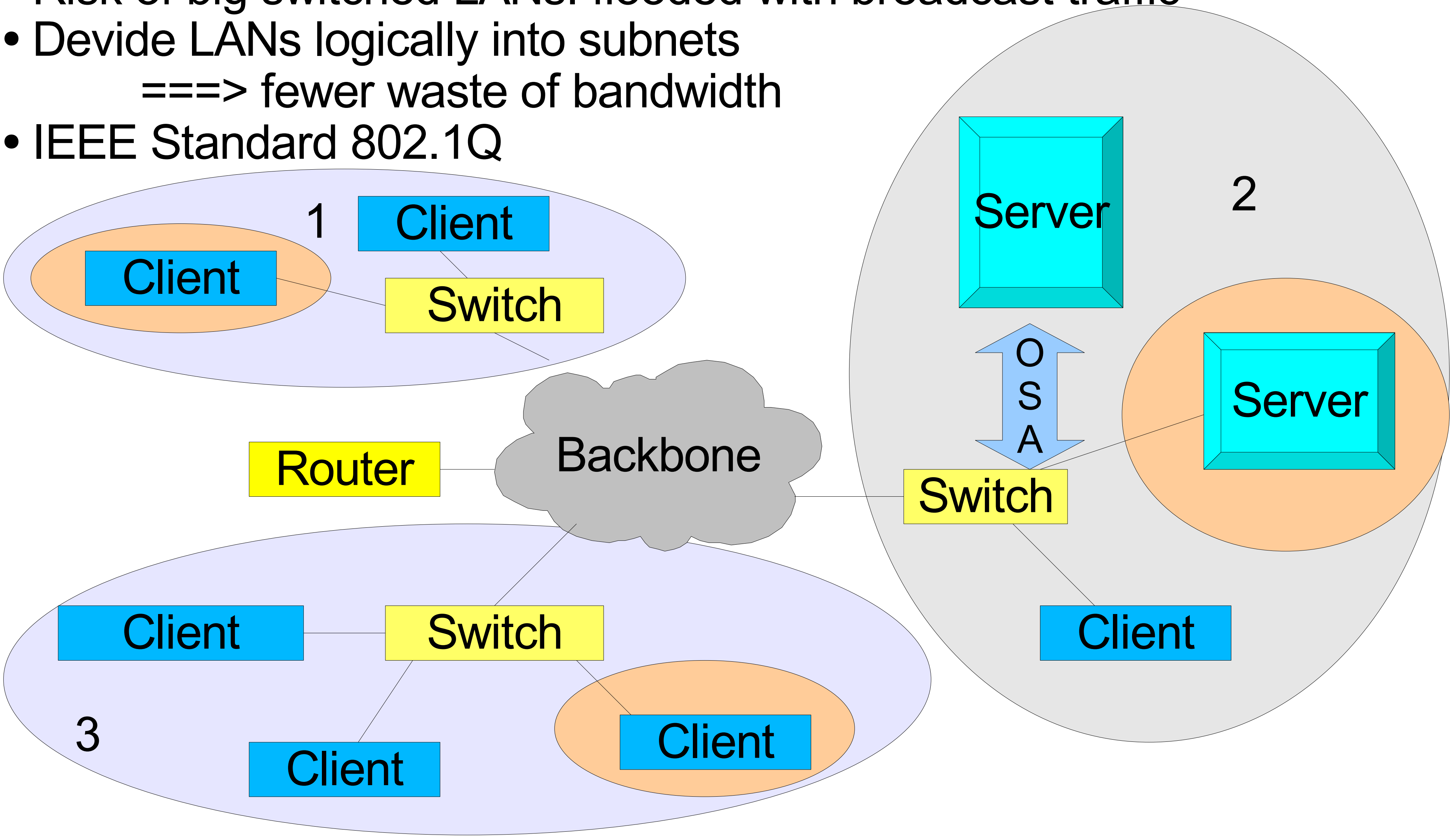
*) All possible values:

procfs	sysfs	Description
pri	primary router	Primary Router
sec	secondary router	Secondary Router
mc	multicast router	Multicast Router
mc+	multicast router+	Multicast Router with broadcast filtering
p.c	primary connector	Primary Connector
p+c	primary connector+	Primary Connector with broadcast filtering
s.c	secondary connector	Secondary Connector
s+c	secondary connector+	Secondary Connector with broadcast filtering

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
```

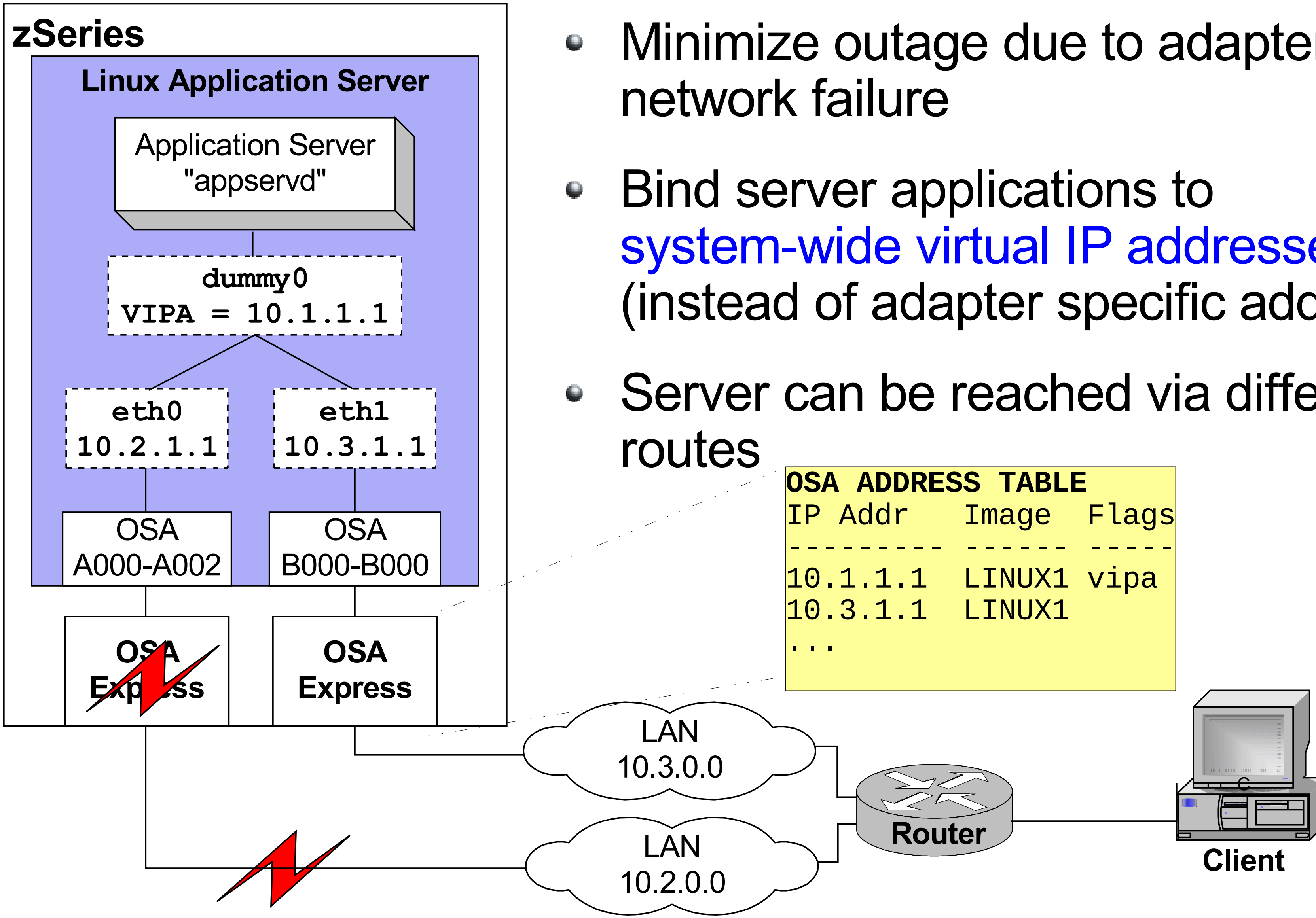
- Displaying info:

```
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



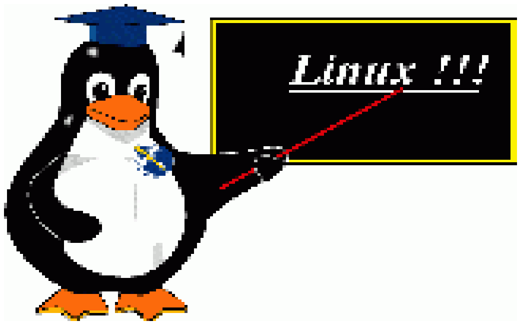
Virtual IP Addresses (cont.)

- 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     add/display IPv4 VIPAs
            |--add6     add/display IPv6 VIPAs
            |--del4     delete IPv4 VIPAs
            |--del6     delete IPv6 VIPAs
```

VIPA configuration is done per device via sysfs attributes



Virtual IP Address Setup

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

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

   or using an interface alias:

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

2. Register the virtual IP address with physical devices:

```
#> 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:

`#> 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:

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

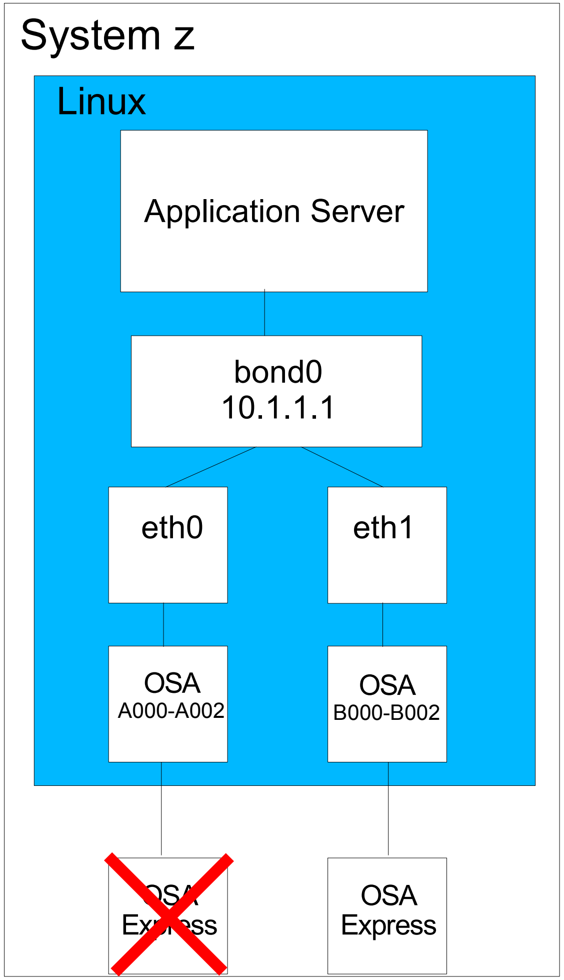
(or make use of the qethconf tool)

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

```
#> 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'
```

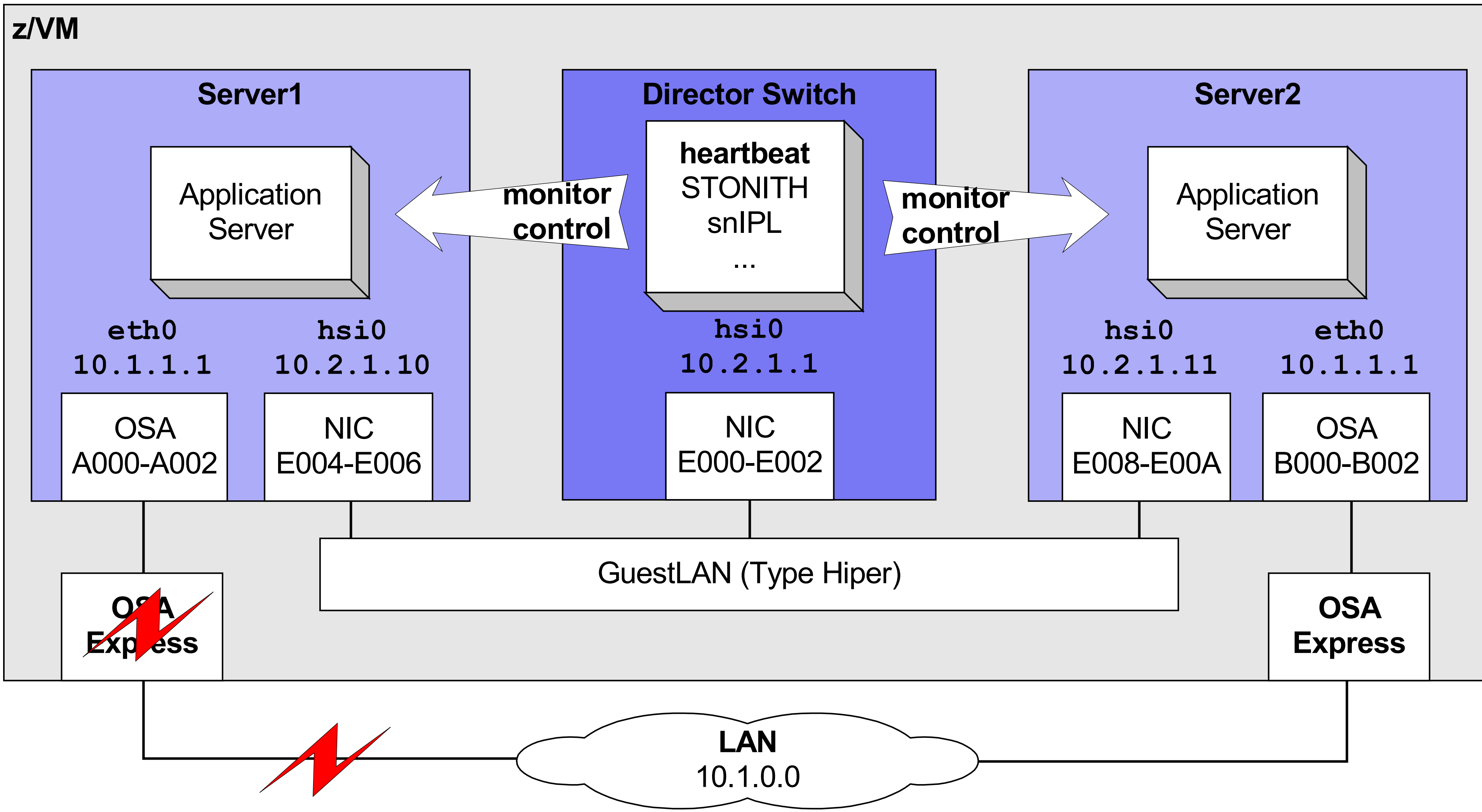
Channel bonding setup (cont.)

```
#> 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 ...
```

```
#> 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

IP Address Takeover configuration is done per device via sysfs attributes

```
/sys
|--devices
    |--qeth
        |--0.0.<devno>
            |--ipa_takeover
                |--add4        add/display IPv4 takeover ranges
                |--add6        add/display IPv6 takeover ranges
                |--del4        delete IPv4 takeover ranges
                |--del6        delete IPv6 takeover ranges
                |--enable      enable takeover
                |--invert4     invert IPv4 takeover ranges
                |--invert6     invert IPv6 takeover ranges
```

IP Address Takeover Setup

1. Enable IP Address Takeover for your device by adding `QETH_IPA_TAKEOVER=1` to the appropriate hwcfg-file.

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

```
echo 1 > /sys/devices/qeth/0.0.a000/ipa_takeover/enable
```

- To activate the setting on a running Linux :
  Stop the device: `#> hwdown qeth-bus-ccw-0.0.a000`

```
echo 0 > /sys/bus/ccwgroup/devices/0.0.a000/online
```

- Re-initialize device with new settings:

```
#> hwup qeth-bus-ccw-0.0.a000
```

```
echo 1 > /sys/bus/ccwgroup/devices/0.0.a000/online
```

IP Address Takeover Setup (cont.)

2. Specify address ranges for IP Address Takeover

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

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

```
#> echo 10.1.1.0/24 >
   /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:

```
#> echo 1 >
   /sys/class/net/eth0/device/ipa_takeover/invert4
```

IP Address Takeover Setup (cont.)

3. To check current IP Address Takeover ranges, issue

```
#> 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`**'

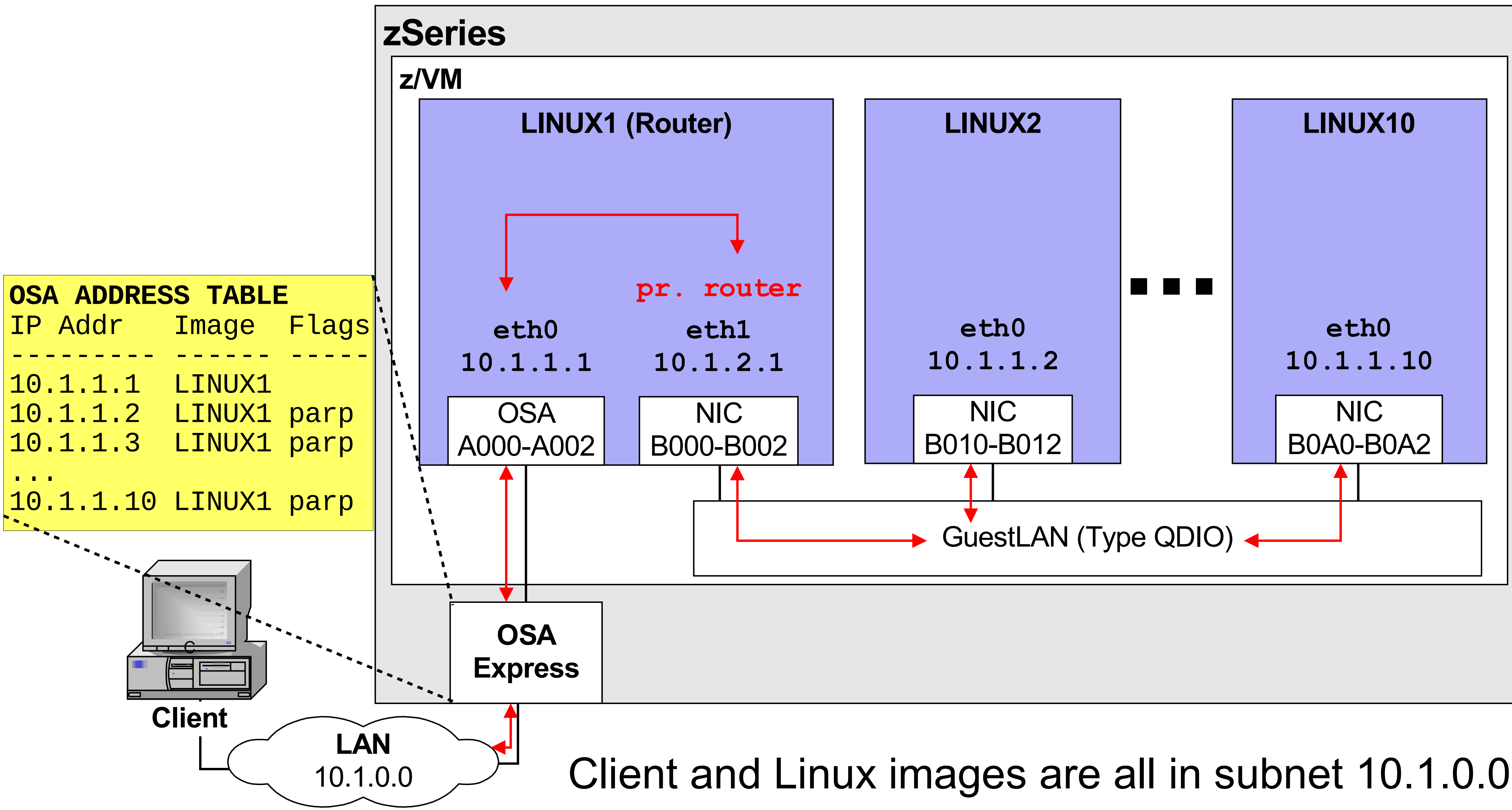
4. Takeover ranges can be deleted by writing to the del4 attribute:

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

IPv6 is analogous, i.e. add/delete ranges via `add6/del6`

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):

```
#> 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:

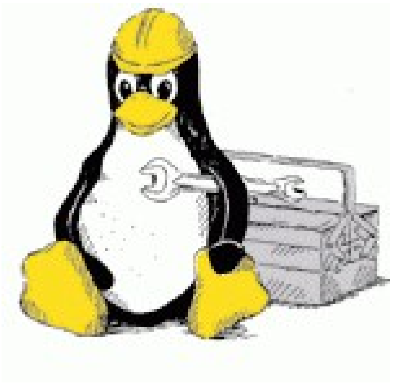
```
#> cat /sys/class/net/eth0/device/rxip/add4
10.1.1.2
10.1.1.3
10.1.1.4
...
```

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

Linux 2.4	Linux 2.6
`qethconf` script	

`/proc/qeth_ipa_takeover`	`/sys/.../qeth/0.0.*/ipa_takeover`
	`/sys/.../qeth/0.0.*/rxip`
	`/sys/.../qeth/0.0.*/vipa`

- Especially useful:
  list commands iterate over all devices found in Linux 2.6 sysfs

qethconf – Examples

- IP Address Takeover configuration:

```
#> qethconf ipa add 10.1.1.0/24 eth0
```

- Proxy ARP configuration:

```
#> qethconf rxip add 10.1.1.2 eth0
```

- VIPA configuration:

```
#> 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:

```
#> 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:

```
#> qetharp -nq eth1
Address                 HWaddress            HWType     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
```

remote

local OAT

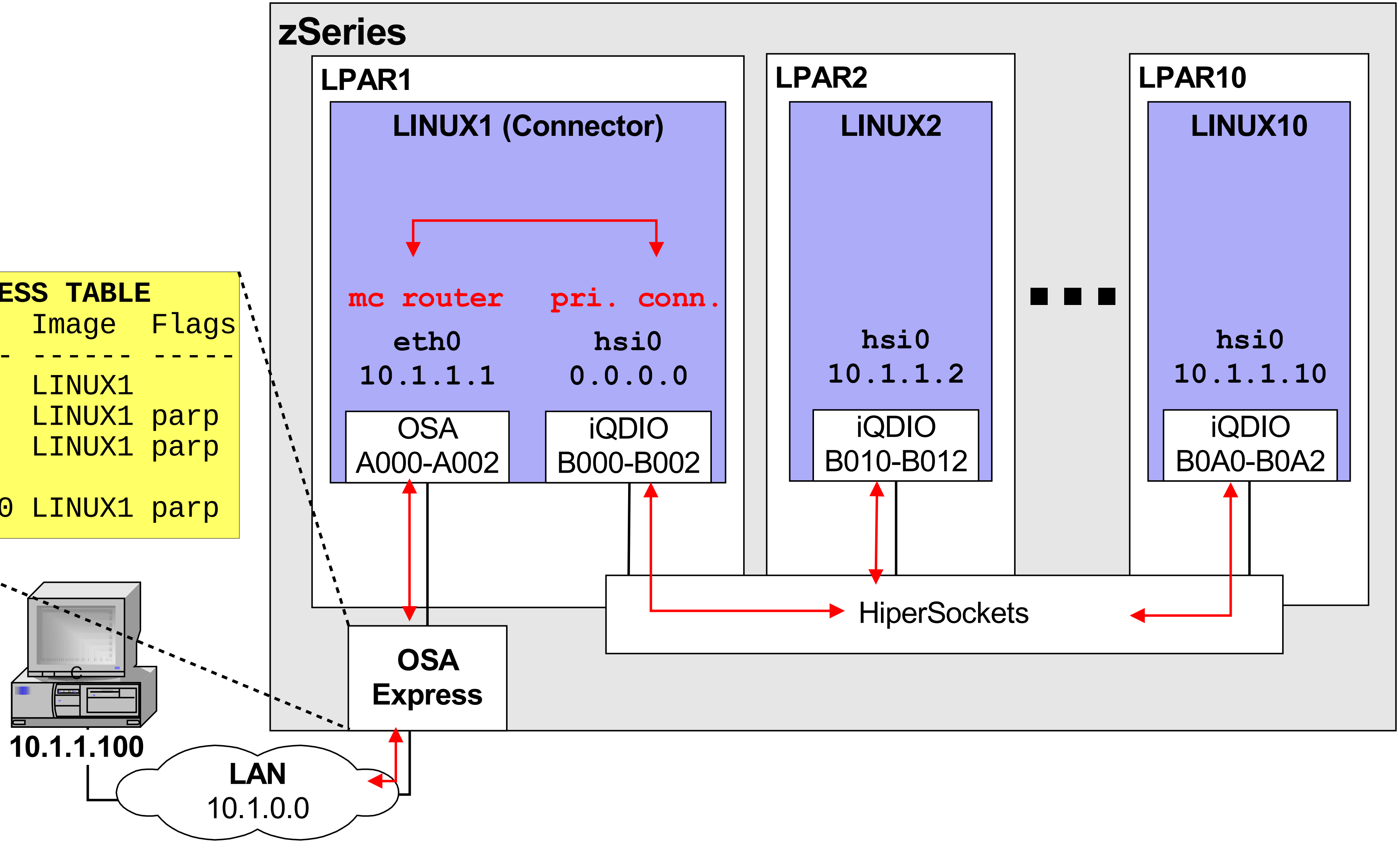
Query address table of a HiperSockets CHPID:

```
#> qetharp -nq hsi0
Address                 HWaddress            HWType     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

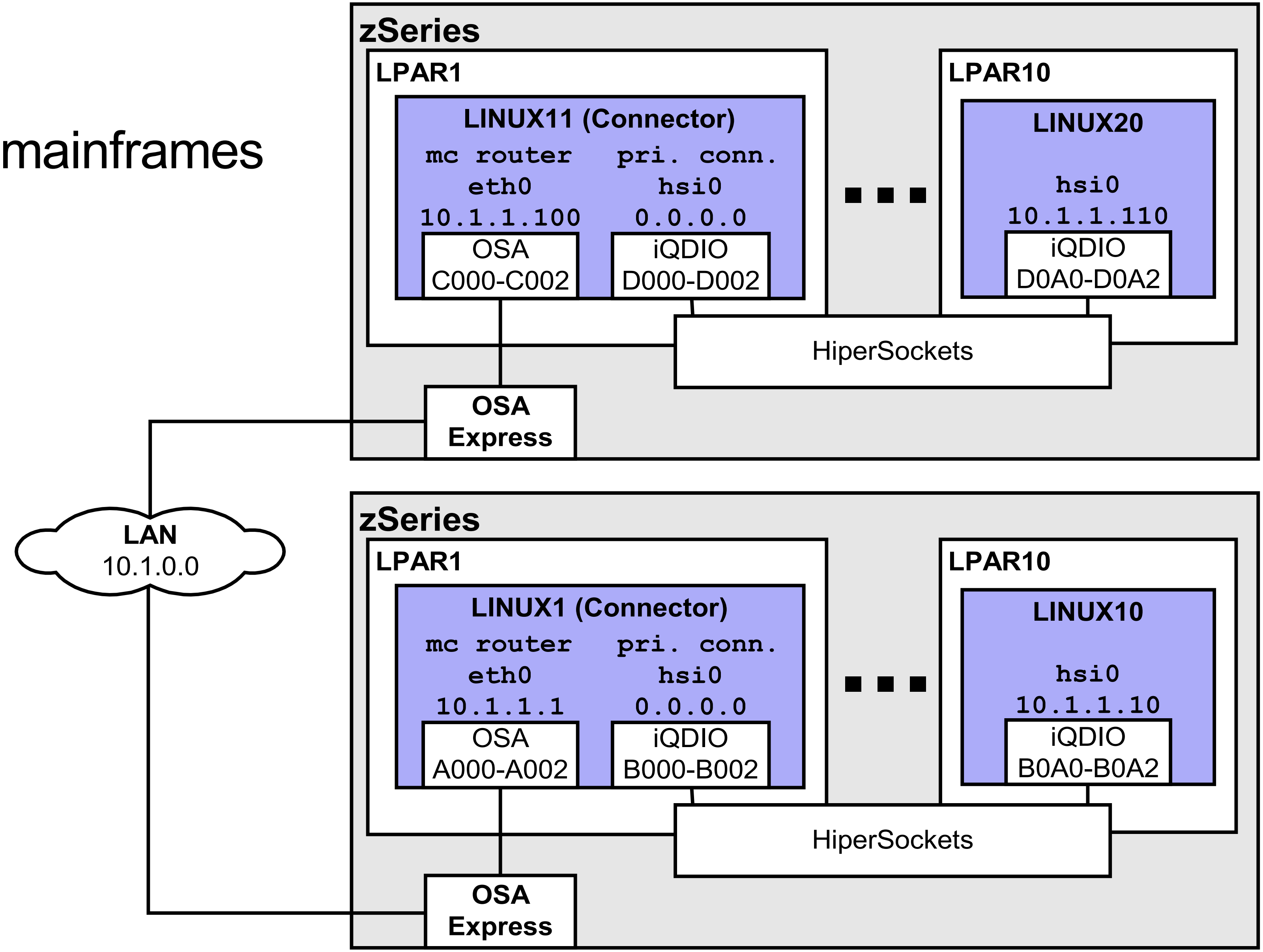
OSA ADDRESS TABLE

IP Addr	Image	Flags
10.1.1.1	LINUX1	
10.1.1.2	LINUX1	parp
10.1.1.3	LINUX1	parp
...		
10.1.1.10	LINUX1	parp



HSNC – Topology Example 2

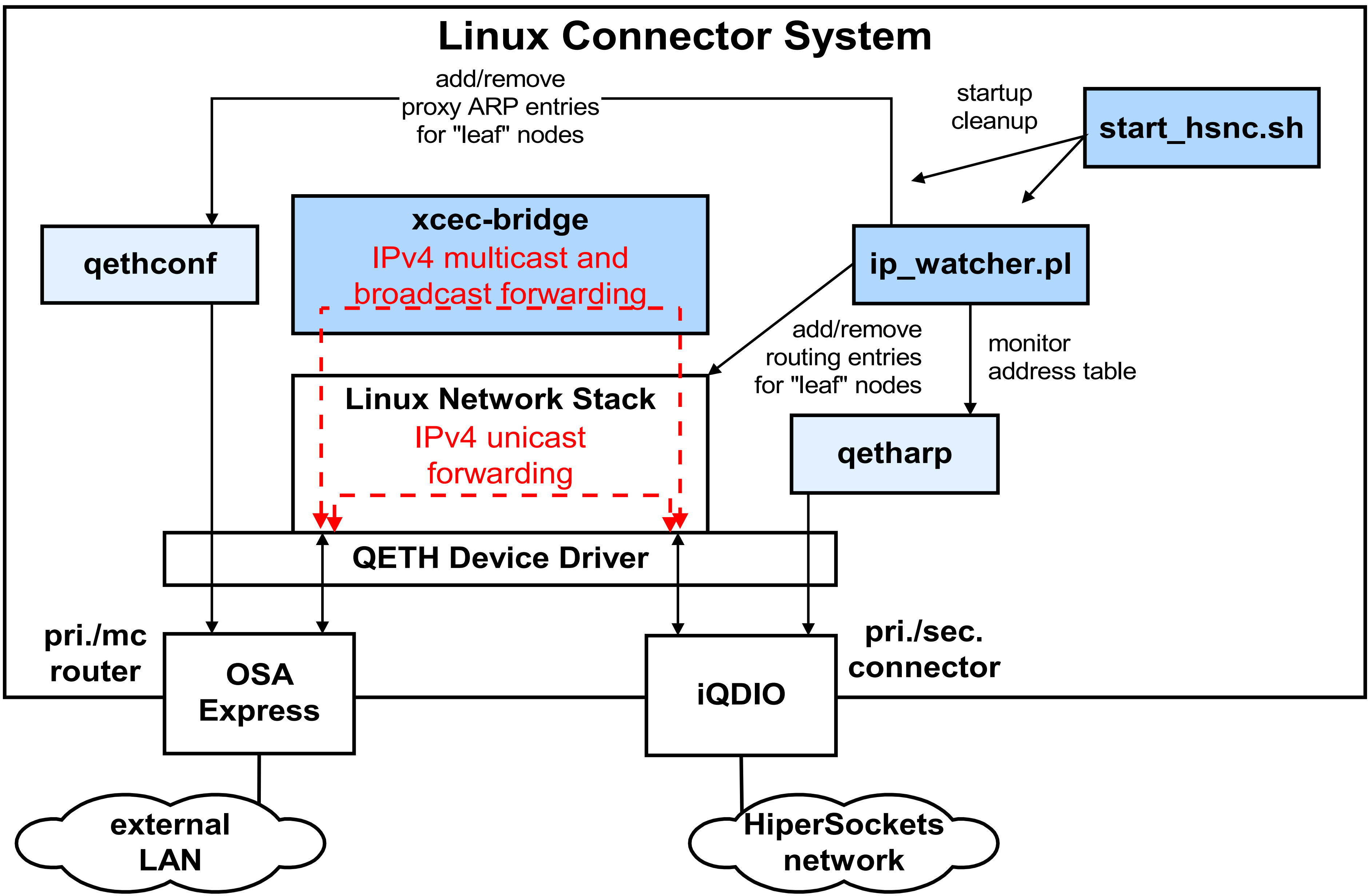
Different mainframes or XCEC



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



HSNC – Setup of Connector System

1. Configure your OSA device as multicast router: *

```
#> 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: *

```
#> 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:

```
#> 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:

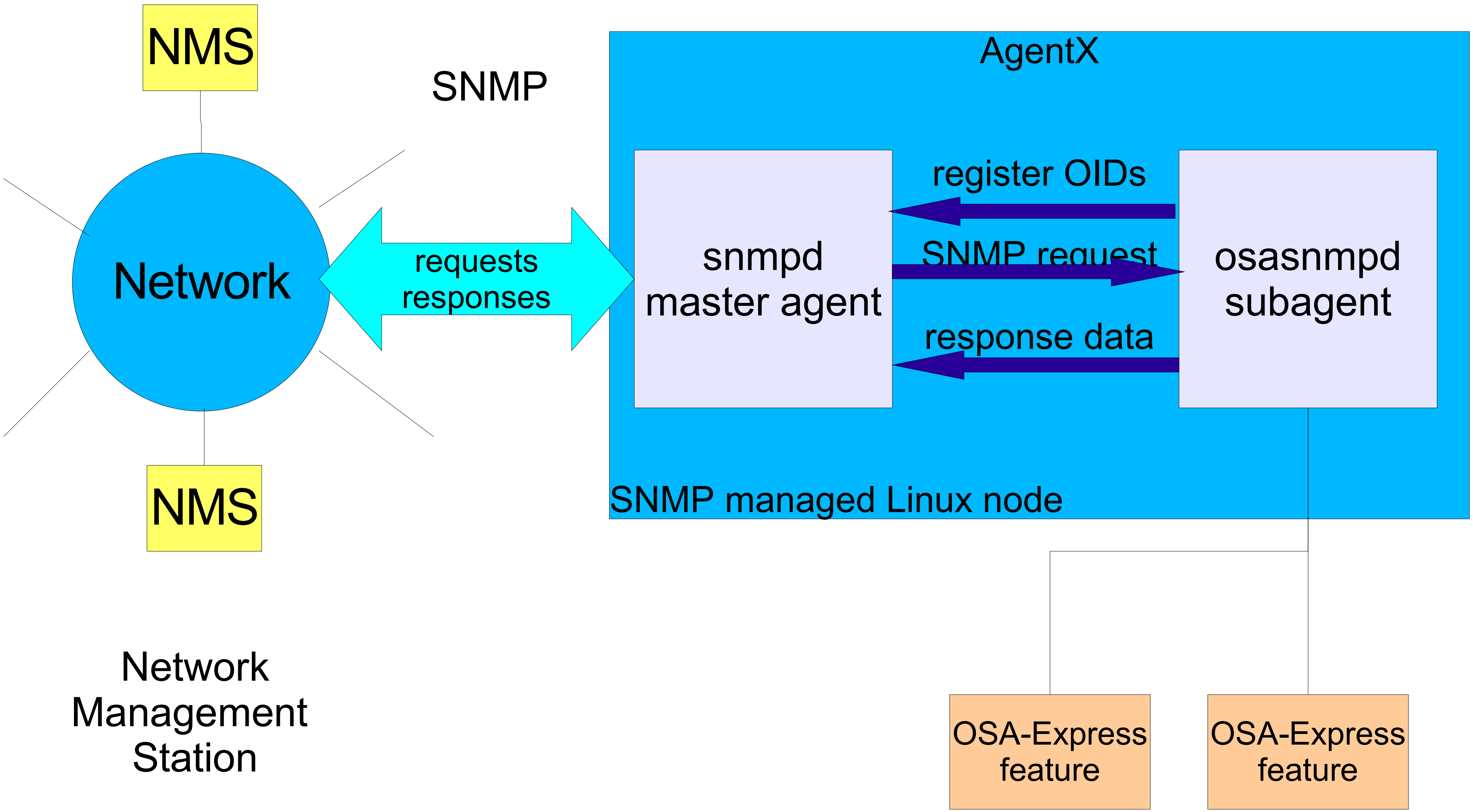
```
#> sysctl -w net.ipv4.ip_forward=1
```

5. Start HSNC:

```
#> start_hsnc.sh
```

The OSA device can be specified as start option. This enables unicast forwarding only.

SNMP support: osasnmpd



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
  http://www.ibm.com/developerworks/opensource/linux390/index.shtml
- Linux for zSeries and S/390 Device Drivers, Features and Commands
  http://www.ibm.com/developerworks/linux/linux390/april2004_documentation.html
- Linux for zSeries and S/390, useful add-ons
  http://www.ibm.com/developerworks/linux/linux390/useful_add-ons.html
  - snIPL
  - src_vipa
- Linux High-Availability Project
  http://linux-ha.org


MARIAN CERMAN