Networking with Linux® on zSeries
(Part 1 of 2)

Session 9267

SHARE
Technology · Connections · Results

Steffen Thoss (thoss@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

- Linux 2.6 device model
- Networking example
- Linux on zSeries network device drivers:
  - QETH
  - LCS
  - CTC
  - IUCV
- Summary

Linux 2.6 Device Model

- Integrated uniform device model that reflects a system's hardware structure
- Simplified device reference counting and locking
- Unified user interface via sysfs
  - **Hierarchical, tree-like** representation of system's hardware
  - Several subsystems provide **different views** of the hardware
  - **Configuration of devices via attribute files**
  - **Dynamic attach/detach** of devices possible

Linux 2.6 Device Model (cont.)

```
/sys
|--block
|   |...
|--bus
|   |...
|--class
|   |...
|   |--net
|   |...
|--devices
|   |...
|...
```

Block subsystem (view):
Block devices and partitions (dasda, ram0)

Bus subsystem (view):
Device drivers and devices sorted by bus (ccw)

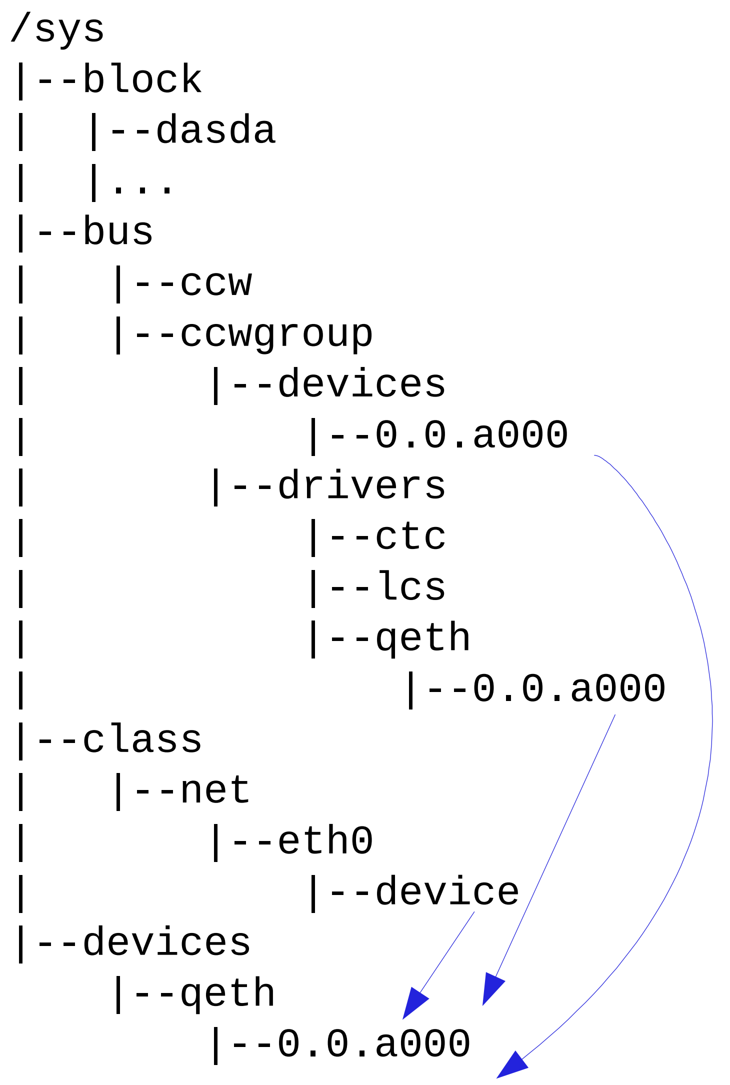
Class subsystem (view):
Logical devices sorted by type, i.e. to which class they belong;
Logical devices have link to hardware device

Devices subsystem (view):
All the devices of a system

Linux 2.6 Device Model - zSeries

- **Fully integrated into common device model** (sysfs and underlying kernel structures)
- Bus and device types:
  - Channel subsystem bus / I/O subchannel devices
    - ID: subchannel number
    - Attributes: channel paths, detach state, path masks
  - CCW device bus / CCW devices
    - ID: device number
    - Attributes: CU type, device type, online state [, group_device] + driver specific
  - CCW group device bus / CCW group devices
    - Groups of single CCW devices make up a functional unit
    - ID: device number of first device in group
    - Attributes: CCW devices, CHPID, aggregate online state, ungroup + driver specific

Linux 2.6 Device Model – zSeries Examples



```
/sys
|--block
|   |--dasda
|   |...
|--bus
|    |--ccw
|    |--ccwgroup
|        |--devices
|            |--0.0.a000
|        |--drivers
|            |--ctc
|            |--lcs
|            |--qeth
|                |--0.0.a000
|--class
|    |--net
|        |--eth0
|            |--device
|--devices
    |--qeth
        |--0.0.a000
```

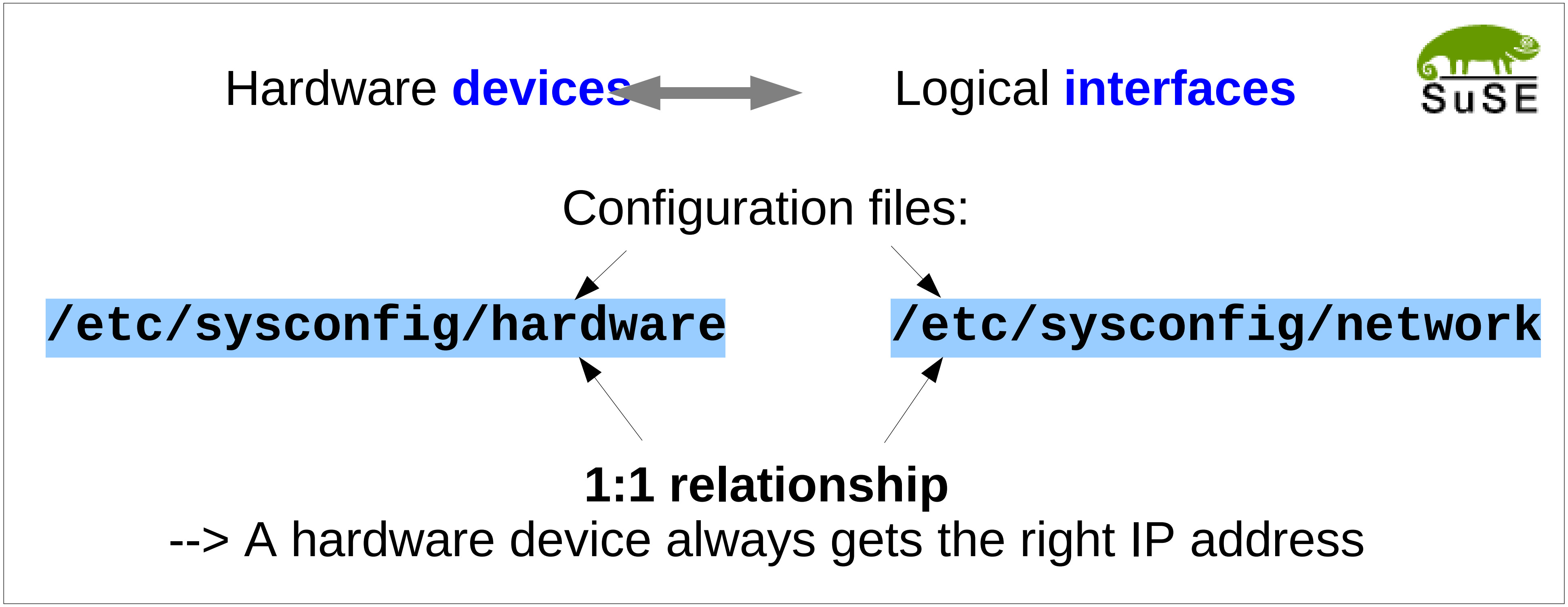
Block Devices:
DASD, RAM-Disk, Minidisk
SCSI, Loopback

CCW Group Devices:
QETH, LCS, CTC

Example: a QETH device

Many ways to find a device

SUSE SLES 9 Network Configuration



Hardware **devices** ⟷ Logical **interfaces**

Configuration files:

`/etc/sysconfig/hardware`

`/etc/sysconfig/network`

1:1 relationship

--> A hardware device always gets the right IP address

Naming convention:

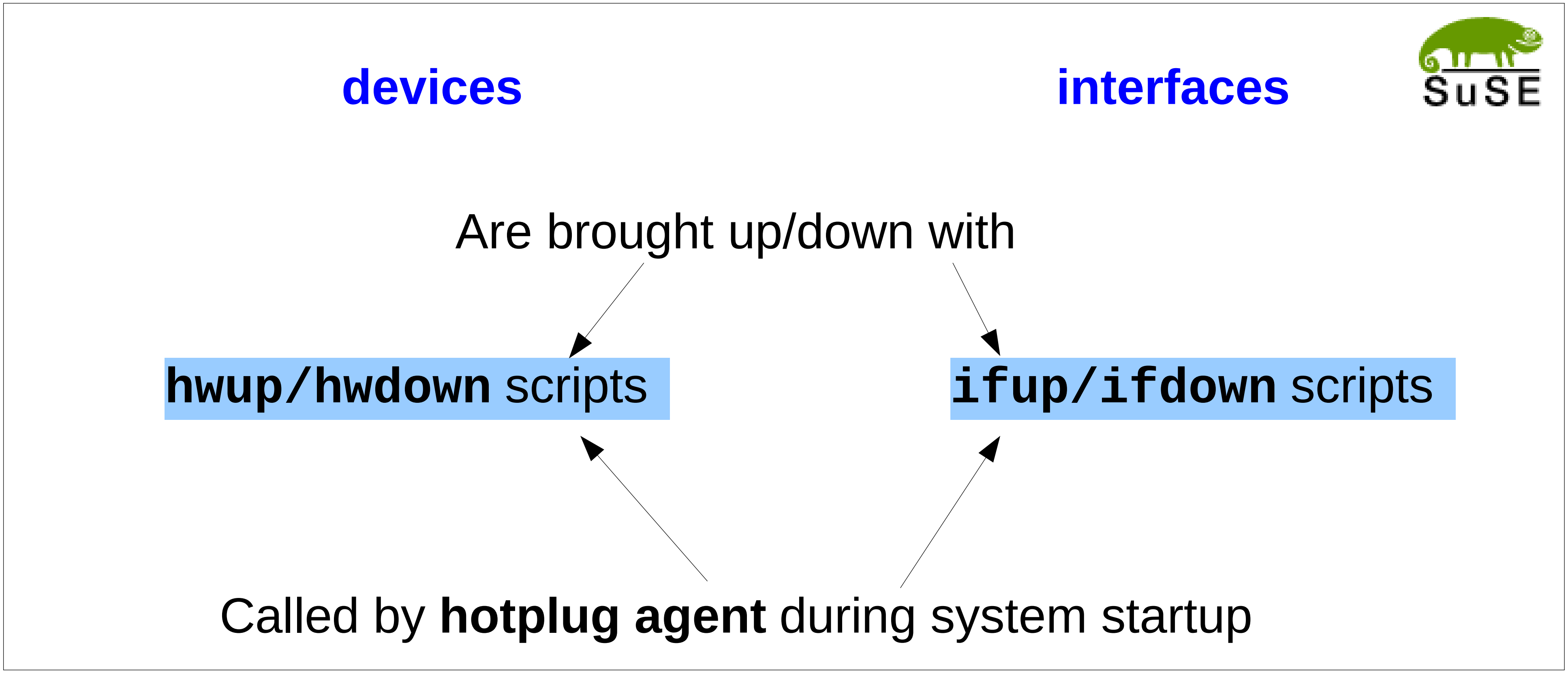
hw/ifcfg-<device type>-bus-<bus type>-<bus location> or

hw/ifcfg-<device type>-id-<identifier> (e.g. for IUCV)

e.g. **hwcfg-qeth-bus-ccw-0.0.a000**

ifcfg-qeth-bus-ccw-0.0.a000

SUSE SLES 9 Network Configuration (cont.)



devices

interfaces

Are brought up/down with

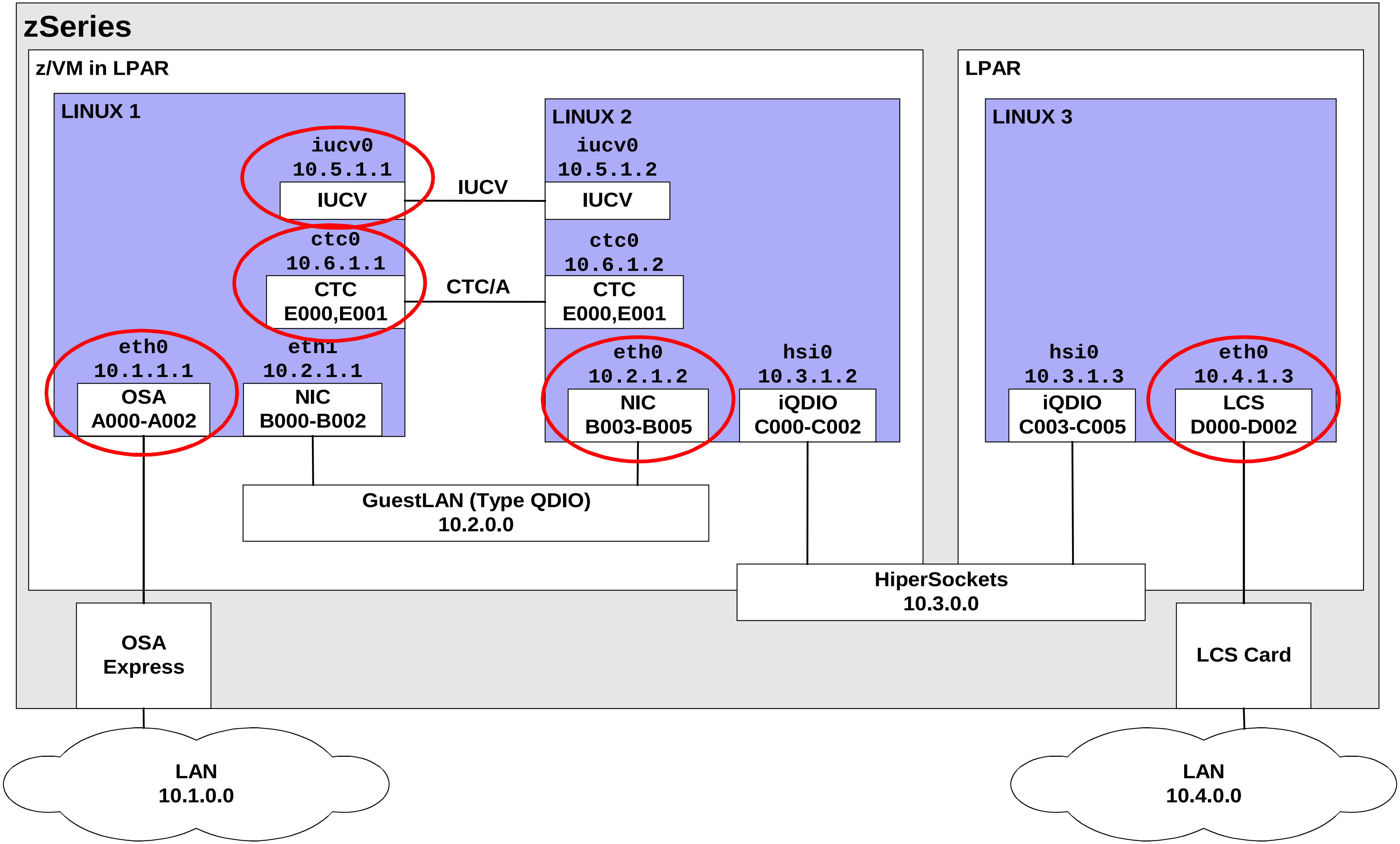
hwup/hwdown scripts

ifup/ifdown scripts

Called by **hotplug agent** during system startup

See also: /usr/share/doc/packages/sysconfig/README and README.s390

Networking Example



Linux for zSeries Network Device Drivers

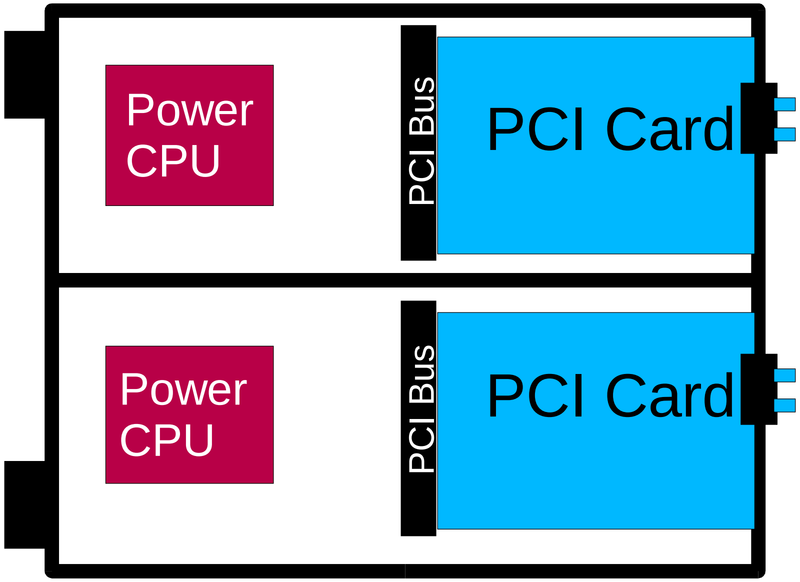
- QETH
- LCS
- CTC
- IUCV
- Major rework of existing Linux 2.4 device drivers
  - To integrate into Linux 2.6 common device model
  - To port old user interfaces to sysfs
  - Cleanup of source code --> improved readability and maintainability
  - Performance improvements

QETH Device Driver

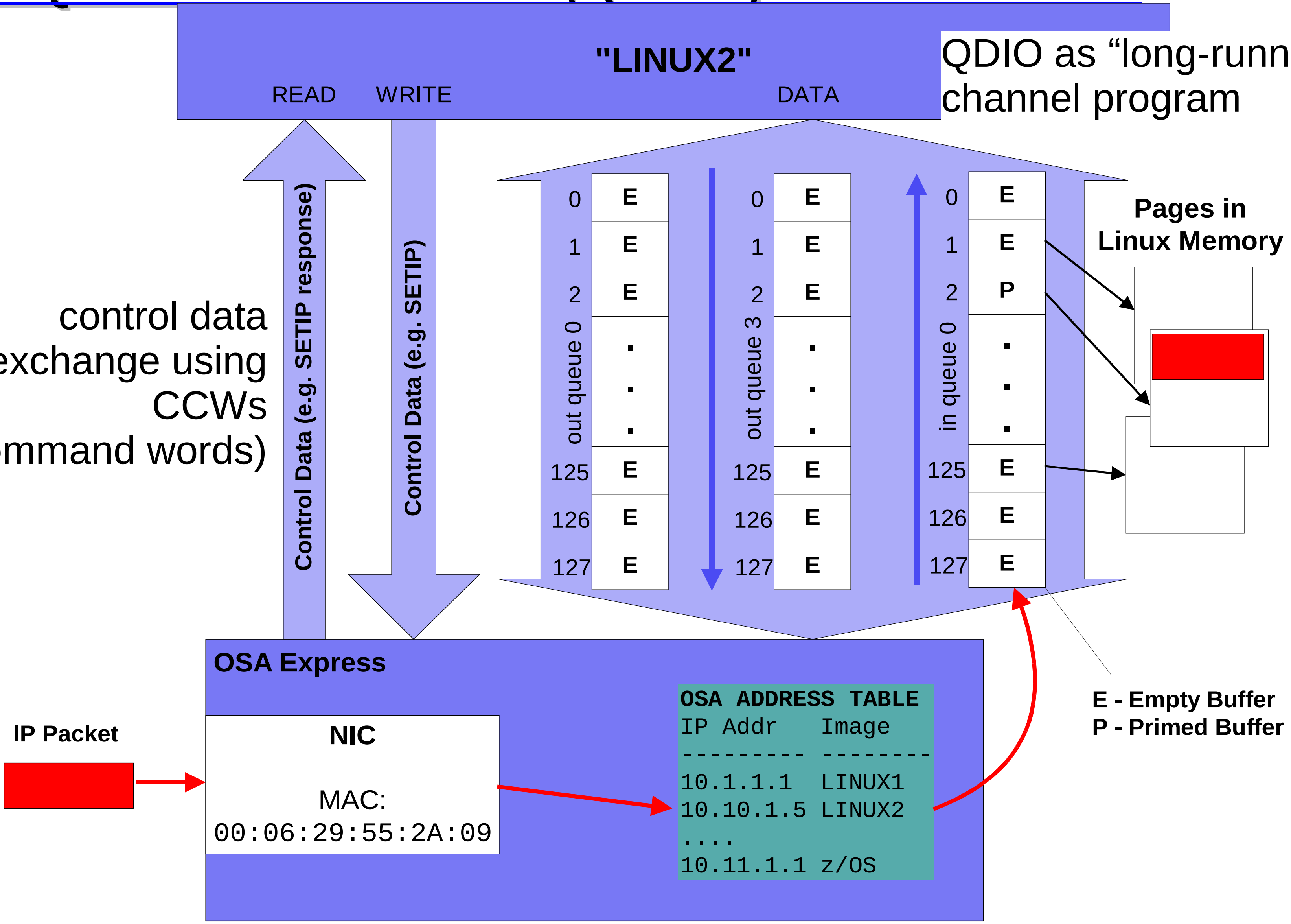
- Supports:
  - OSA Express	Fast Ethernet
    Gigabit Ethernet
    10 Gbit Ethernet
    Highspeed Tokenring
    ATM (running Ethernet LAN Emulation)
  - z/VM GuestLAN	Type QDIO
    Type Hiper
  - zSeries HiperSockets
- IPv4, IPv6, VLAN, VIPA, Proxy ARP, IP Address Takeover
- **Primary network driver for Linux on zSeries**
- **Has main focus in current and future development**

Primary Network Device: OSA Express

- 'Integrated Power computer' with network daughter card
- Shared between up to 640 TCP/IP stacks
- OSA Address Table: which OS image has which IP address
- Three devices (I/O subchannels) per stack:
  - Read device (control data <-- OSA)
  - Write device (control data --> OSA)
  - Data device (network traffic)
- Network traffic Linux <--> OSA at IP or ARP level
- One MAC address for all stacks
- OSA handles ARP (Address Resolution Protocol)



The Queued Direct I/O (QDIO) Architecture



QDIO as “long-running” channel program

control data exchange using CCWs (channel command words)

	out queue 0		out queue 3		in queue 0
0	E	0	E	0	E
1	E	1	E	1	E
2	E	2	E	2	P
	.		.		.
	.		.		.
	.		.		.
125	E	125	E	125	E
126	E	126	E	126	E
127	E	127	E	127	E

```
OSA ADDRESS TABLE
IP Addr    Image
---------  --------
10.1.1.1   LINUX1
10.10.1.5  LINUX2
....
10.11.1.1  z/OS
```

E - Empty Buffer
P - Primed Buffer

Static QETH Device Setup

For LINUX 1 eth0 (see Networking Example)

1. Create a hardware device configuration file:

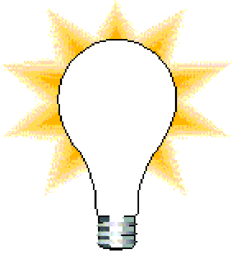
```
/etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.a000:
  CCW_CHAN_IDS='0.0.a000 0.0.a001 0.0.a002'
  CCW_CHAN_MODE='OSAPORT'
  CCW_CHAN_NUM='3'
  MODULE='qeth'
  MODULE_OPTIONS=''
  SCRIPTDOWN='hwdown-ccw'
  SCRIPTUP='hwup-ccw'
  SCRIPTUP_ccw='hwup-ccw'
  SCRIPTUP_ccwgroup='hwup-qeth'
  STARTMODE='auto'
  QETH_OPTIONS='fake_ll=1'
```

Static QETH Device Setup (cont.)

- **`CCW_CHAN_IDS`** are Read, Write, Data channels
  - Read must be even, Write must be Read + 1 (for older microcode)
  - Hexadecimal characters must be lowercase
- **`STARTMODE`** 'auto' --> started by hotplug agents
  'manual' --> manual startup
- **`QETH_OPTIONS`** allows to set optional attributes

  e.g. `QETH_OPTIONS='fake_ll=1'`

- A sample hwcfg-file for QETH can be found at `/etc/sysconfig/hardware/skel/hwcfg-qeth`



Static QETH Device Setup (cont.)

2. Create an interface configuration file:

```
/etc/sysconfig/network/ifcfg-qeth-bus-ccw-0.0.a000
   BOOTPROTO='static'
   BROADCAST='10.1.255.255'
   IPADDR='10.1.1.1'
   NETMASK='255.255.0.0'
   NETWORK='10.1.0.0'
   STARTMODE='onboot'
   PERSISTANT_NAME='interf0
```

3. Before reboot: test your config files:

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

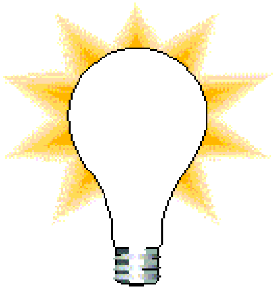
Static QETH Device Setup on Linux 2.4

Hardware configuration is in `/etc/chandev.conf`:

```
qeth0,0xa000,0xa001,0xa002
   add_parms,0x10,0xa000,0xa002,portname:OSAPORT
```

A script exists which can convert your Linux 2.4 chandev.conf into Linux 2.6 hwcfg-files (for QETH, LCS and CTC):

```
/etc/sysconfig/hardware/scripts/chandev-to-hwcfg.sh
```



Static QETH Device Setup on Linux 2.4 (cont.)

Interface configuration:

```
/etc/sysconfig/network-scripts/ifcfg-eth0
  DEVICE=eth0
  USERCTL=no
  ONBOOT=yes
  BOOTPROTO=none
  BROADCAST=10.1.255.255
  NETWORK=10.1.0.0
  NETMASK=255.255.0.0
  IPADDR=10.1.1.1
  ARP=no
```

Device – IP address mapping:

`qeth<n>` notation in chandev.conf

↕

```
ifcfg-eth<n>
   DEVICE=eth<n>
```

Dynamic QETH Device Setup

For LINUX 2 eth0 (see Networking Example)

1. In your z/VM console (if not already defined in user directory) do

   1.1. Create a GuestLAN

   ```
   #CP DEFINE LAN MY_LAN TYPE QDIO
   ```

   1.2. Create a virtual NIC

   ```
   #CP DEFINE NIC B003 TYPE QDIO
   ```

   1.3. Couple virtual NIC to GuestLAN

   ```
   #CP COUPLE B003 TO * MY_LAN
   ```

Dynamic QETH Device Setup (cont.)

2. Load the QETH device driver module:

```
#> modprobe qeth
```

3. Create a new QETH device by grouping its CCW devices:

```
#> echo 0.0.b003,0.0.b004,0.0.b005 > /sys/bus/ccwgroup/
   drivers/qeth/group
```

4. Set optional attributes:

```
#> echo 64 > /sys/bus/ccwgroup/drivers/qeth/0.0.b004/
   buffer_count

#> echo 1 > /sys/devices/qeth/0.0.b004/fake_ll
```

Note the alternative ways to your device

Dynamic QETH Device Setup (cont.)

5. Set the new device online:

```
#> echo 1 > /sys/devices/qeth/0.0.b003/online
```

6. Check your QETH devices:

```
#> cat /proc/qeth
devices                    CHPID interface  cardtype
-------------------------- ----- ---------- -------------- ...
0.0.c000/0.0.c001/0.0.c002 xC0   hsi0       HiperSockets
0.0.b003/0.0.b004/0.0.b005 x01   eth0       GuestLAN QDIO
```

7. Configure your new eth0 interface:

```
#> ifconfig eth0 10.2.1.2 netmask 255.255.0.0
```

Dynamic QETH Device Setup on Linux 2.4

1. Add definition of the new device to `/etc/chandev.conf`:

```
qeth0,0xb003,0xb004,0xb005
add_parms,0x10,0xb003,0xb005,fake_ll:1
```

can also be echoed directly to /proc/chandev

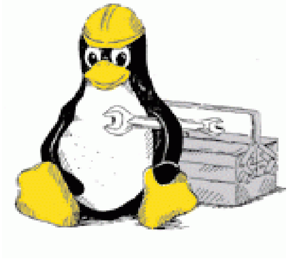
```
#> echo ‟qeth0,0xb003,0xb004,0xb005;
         add_parms,0x10,0xb003,0xb005,fake_ll:1”
         > /proc/chandev
```

2. Activate the new configuration:

```
#> echo readconf > /proc/chandev
#> echo reprobe > /proc/chandev
```

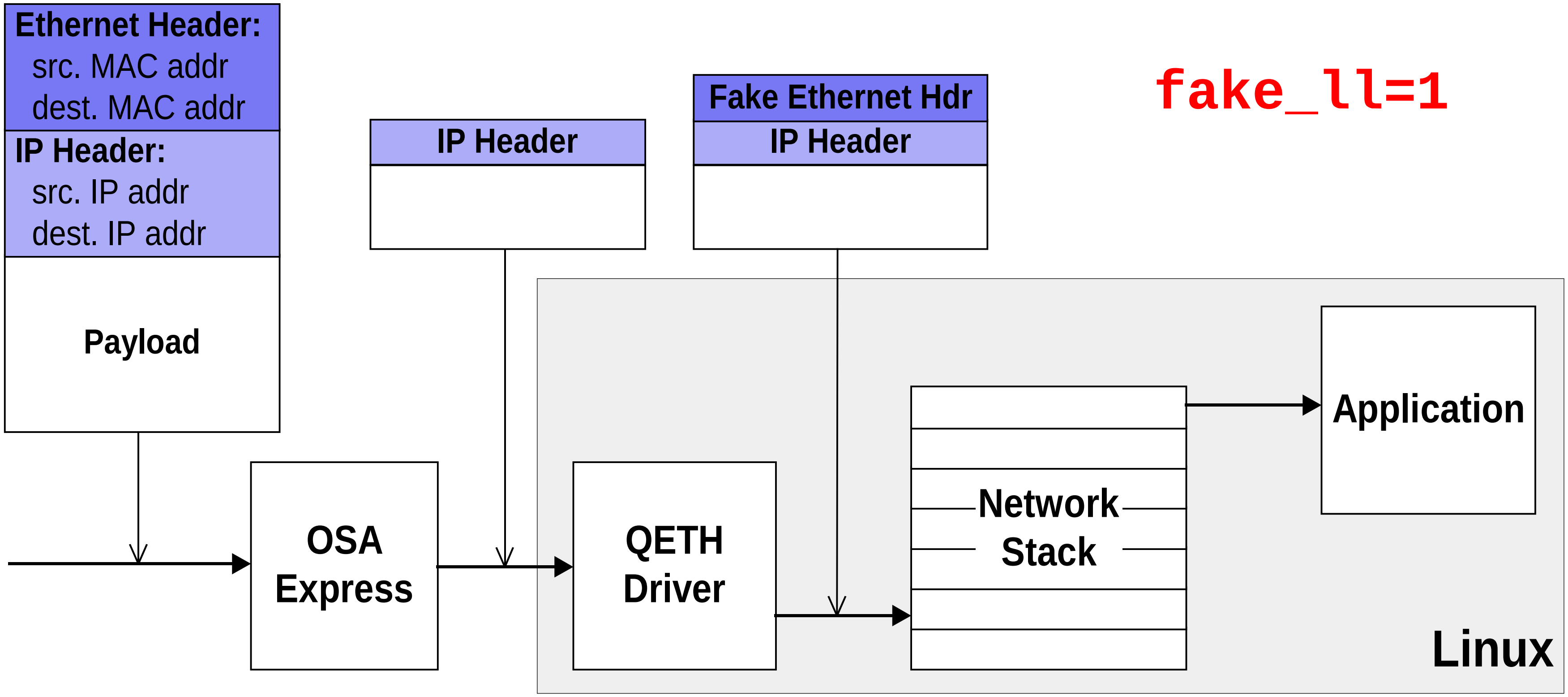
3. Configure the interface:

```
#> ifconfig eth0 10.2.1.2 netmask 255.255.0.0
```



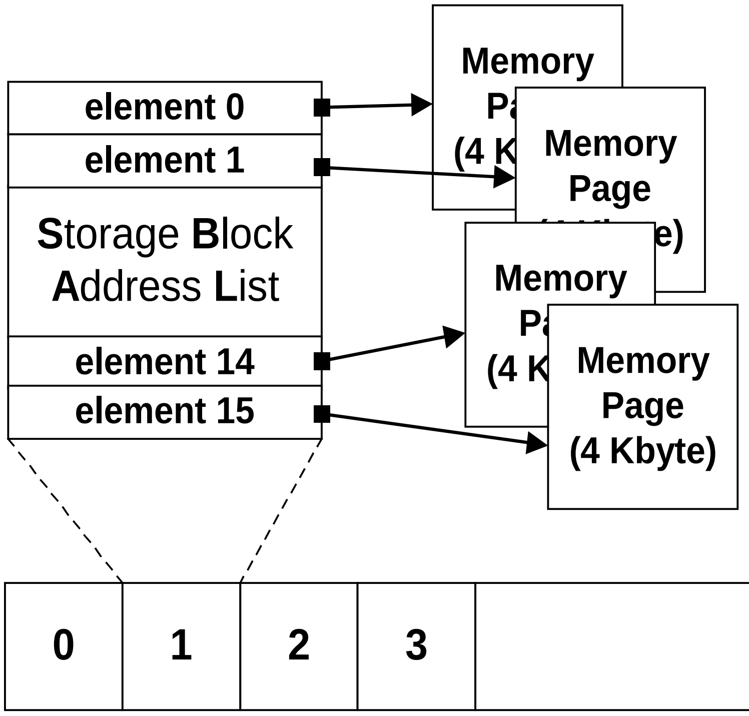
Interesting QETH Device sysfs Attributes
fake_ll

- Build fake ethernet headers before handing packets to the network stack.
- Required by some network applications, e.g. **DHCP** or TCPDUMP



Interesting QETH Device sysfs Attributes buffer_count

- The number of allocated buffers for inbound QDIO traffic --> Memory usage.

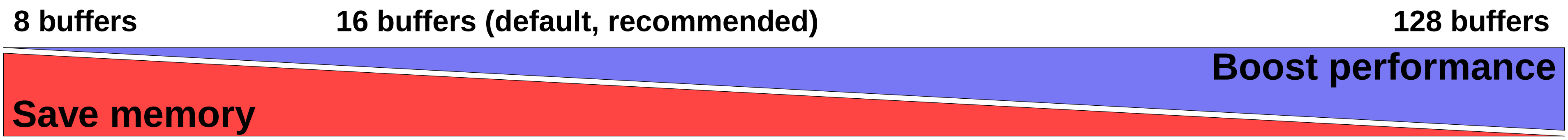


Per QETH card memory usage:

control data structures: ~ 200 KB
memory for one buffer: 64 KB

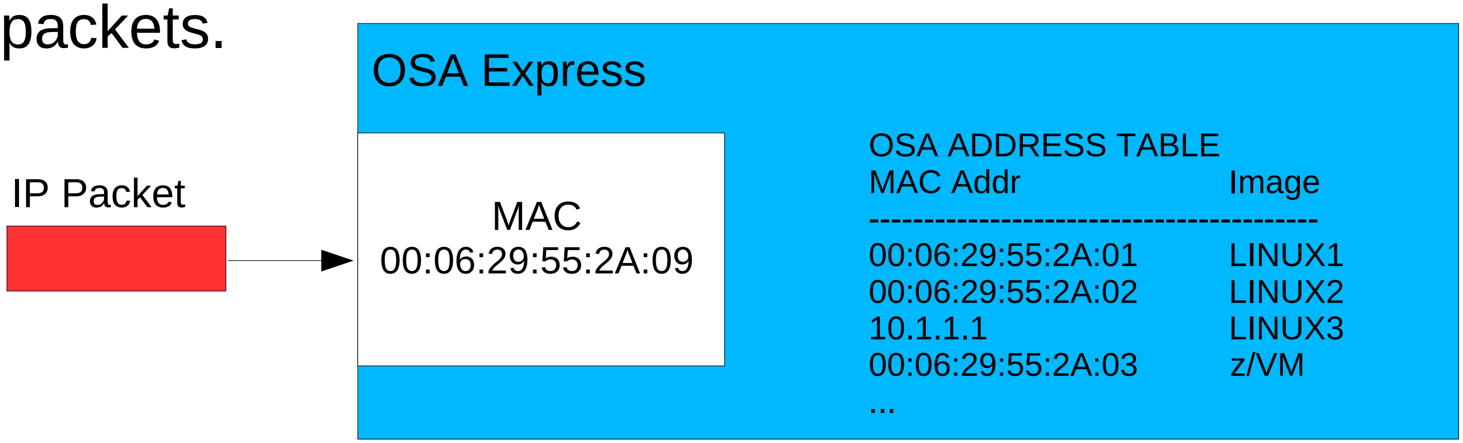
buffer_count = 8 --> ~ 712 KB

buffer_count = 128 --> ~ 8.4 MB



Interesting QETH Device sysfs Attributes - layer2

- OSA works with MAC addresses, MAC addresses no longer stripped from packets.



OSA ADDRESS TABLE

MAC Addr	Image
00:06:29:55:2A:01	LINUX1
00:06:29:55:2A:02	LINUX2
10.1.1.1	LINUX3
00:06:29:55:2A:03	z/VM
...	

- hwcfg-qeth... file : `QETH_LAYER2_SUPPORT=1`
- ifcfg-qeth... file: `LLADDR='<MAC Address>'`
- Direct attached OSA:

  MAC address must be defined with ifconfig manualy

  ```
  ifconifg eth0 hw ether 00:06:29:55:2A:01
  ```
- with VSWITCH under z/VM

  MAC address created by z/VM VSWITCH
- DHCP, tcpdump working without option fake_ll
- channel bonding possible

Interesting QETH Device sysfs Attributes – layer2 (cont.)

- Direct attach OSA and GuestLAN type QDIO supported
  GuestLAN definition for layer2:
  ```
  define lan <lanname> ... type QDIO ETHERNET
  define nic
  ```
- Prerequisites:
  - z/VM 5.1 RSU 1 + PTF VM63505, VM63506, VM 63538, PQ97436
  - OSA code level 6.25(MCL J13477.066, Bundle 28)
  - SUSE SLES8 kernel 2.4.21-266
  - SUSE SLES9 SP2

- Restrictions:
  - Layer2 and Layer3 traffic can be transmitted over the same OSA CHPID, but not between two hosts sharing the same CHPID !

LCS Device Driver

- LCS – LAN Channel Station
- Supports:
  - OSA-2 Ethernet and Tokenring
  - OSA-Express Fast Ethernet and Highspeed Tokenring (in non-QDIO mode)
  - Since z990: OSA-Express Gigabit Ethernet (incl. 1000Base-T) (in non-QDIO mode)
- May be preferred instead of QETH for security reasons
  - Administrator defines OSA Address Table, whereas with QETH each Linux registers its own IP address --> restricted access

But: performance is inferior to QETH's performance!!!

Static LCS Device Setup

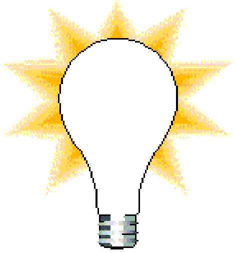
For LINUX 3 eth0 (see Networking Example)

1. Create a hardware device configuration file:

```
/etc/sysconfig/hardware/hwcfg-lcs-bus-ccw-0.0.d000:
  CCW_CHAN_IDS='0.0.d000 0.0.d001'
  CCW_CHAN_MODE='0'
  CCW_CHAN_NUM='2'
  MODULE='lcs'
  MODULE_OPTIONS=''
  SCRIPTDOWN='hwdown-ccw'
  SCRIPTUP='hwup-ccw'
  SCRIPTUP_ccw='hwup-ccw'
  SCRIPTUP_ccwgroup='hwup-lcs'
  STARTMODE='auto'
```

Static LCS Device Setup (cont.)

- **`CCW_CHAN_IDS`** are Read and Write channels
  - Read must be even, Write must be Read + 1
  - Hexadecimal characters must be lowercase
- **`CCW_CHAN_MODE`** selects the card's relative port
  - Applies to OSA-Express ATM cards only
  - Possible values: 0 .. 15
  - Default is 0
- **`STARTMODE`** 'auto' --> started by hotplug agents
  'manual' --> manual startup
- A sample hwcfg-file for QETH can be found at
  `/etc/sysconfig/hardware/skel/hwcfg-lcs`



Static LCS Device Setup (cont.)

2. Create an interface configuration file:

```
/etc/sysconfig/network/ifcfg-lcs-bus-ccw-0.0.d000:
  BOOTPROTO='static'
  BROADCAST='10.4.255.255'
  IPADDR='10.4.1.3'
  NETMASK='255.255.0.0'
  NETWORK='10.4.0.0'
  STARTMODE='onboot'
```

3. Before reboot: test your config files:

```
#> hwup lcs-bus-ccw-0.0.d000
```

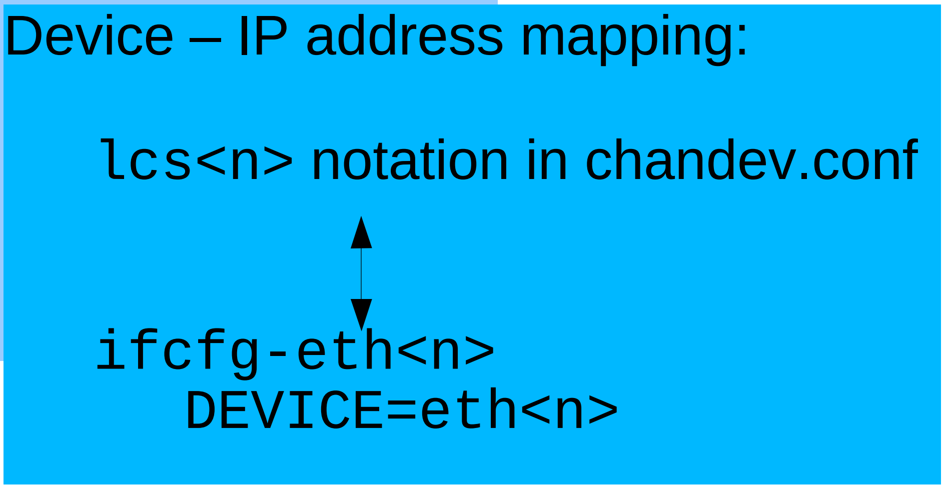
Static LCS Device Setup on Linux 2.4

Hardware configuration is in `/etc/chandev.conf`:

```
 lcs0,0xd000,0xd001
```

Interface configuration:

```
/etc/sysconfig/network-scripts/ifcfg-eth0
  DEVICE=eth0
  USERCTL=no
  ONBOOT=yes
  BOOTPROTO=none
  BROADCAST=10.4.255.255
  NETWORK=10.4.0.0
  NETMASK=255.255.0.0
  IPADDR=10.4.1.3
  ARP=no
```



Device – IP address mapping:

`lcs<n>` notation in chandev.conf

```
ifcfg-eth<n>
   DEVICE=eth<n>
```

Dynamic LCS Device Setup

1. Load the LCS device driver module:

```
#> modprobe lcs
```

2. Create a new LCS device by grouping its CCW devices:

```
#> echo 0.0.d000,0.0.d001 > /sys/bus/ccwgroup/drivers/
    lcs/group
```

3. Set optional attributes:

```
#> echo 2 > /sys/bus/ccwgroup/drivers/lcs/0.0.d000/
   portno
```

4. Set the new device online:

```
#> echo 1 > /sys/devices/lcs/0.0.d000/online
```

Note the alternative ways to your device

Dynamic LCS Device Setup (cont.)

5. Find out the interface for your new device:

   At the moment only possible by checking the 'device' link of each `/sys/class/net` entry:

```
#>ls -Al /sys/class/net/*/device
lrwxrwxrwx  1 root root 0 Jul  6 11:17
    /sys/class/net/eth0/device -> ../../../devices/lcs/0.0.d000
lrwxrwxrwx  1 root root 0 Jul 12 15:14
    /sys/class/net/hsi0/device -> ../../../devices/qeth/0.0.c0
```

6. Configure your new eth0 interface:

```
#> ifconfig eth0 10.4.1.3 netmask 255.255.0.0
```

Dynamic LCS Device Setup on Linux 2.4

1. Add definition of the new device to `/etc/chandev.conf`:

```
lcs0,0xd000,0xd001
```

2. Activate the new configuration:

```
#>echo reandconf > /proc/chandev
#>echo reprobe > /proc/chandev
```

3. Configure the interface:

```
#> ifconfig eth0 10.4.1.3 netmask 255.255.0.0
```

CTC Device Driver

- CTC – Channel-to-Channel connection
- Direct intra- or inter-mainframe communication
- Supports:
  - ESCON
  - FICON
  - Virtual CTC/A (VM)

Static CTC Device Setup

For LINUX 1 ctc0 (see Networking Example)

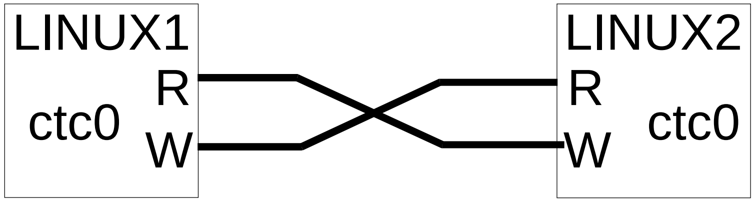
1. Create a virtual CTC connection on your VM console

   1.1. Create virtual CTC devices in both LINUX1 and LINUX2

```
#CP DEFINE CTC E000
#CP DEFINE CTC E001
```

   1.2. Couple CTC devices cross-over, i.e. LINUX1's Read device with LINUX2's Write device ...

```
#CP COUPLE E000 TO LINUX2 E001
#CP COUPLE E001 TO LINUX2 E000
```



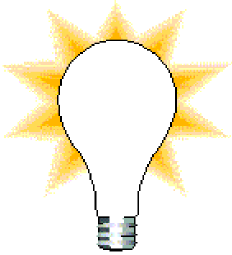
Static CTC Device Setup (cont.)

2. Create a hardware device configuration file:

```
/etc/sysconfig/hardware/hwcfg-ctc-bus-ccw-0.0.e000:
  CCW_CHAN_IDS='0.0.e000 0.0.e001'
  CCW_CHAN_MODE='0'
  CCW_CHAN_NUM='2'
  MODULE='ctc'
  MODULE_OPTIONS=''
  SCRIPTDOWN='hwdown-ccw'
  SCRIPTUP='hwup-ccw'
  SCRIPTUP_ccw='hwup-ccw'
  SCRIPTUP_ccwgroup='hwup-ctc'
  STARTMODE='auto'
```

Static CTC Device Setup (cont.)

- **`CCW_CHAN_IDS`** are Read and Write channels
  - Hexadecimal characters must be lowercase
- **`CCW_CHAN_MODE`** selects protocol for CTC
  - 0 – compatibility with peers other than OS/390 and z/OS (default)
  - 1 – extended mode for Linux peers
  - 2 – for CTC tty based connections to Linux peers
  - 3 – compatibility with OS/390 and z/OS
- **`STARTMODE`** 'auto' --> started by hotplug agents
  'manual' --> manual startup
- A sample hwcfg-file for QETH can be found at
  `/etc/sysconfig/hardware/skel/hwcfg-ctc`



Static CTC Device Setup (cont.)

3. Create an interface configuration file:

```
/etc/sysconfig/network/ifcfg-ctc-bus-ccw-0.0.e000:
  BOOTPROTO='static'
  BROADCAST='10.6.255.255'
  IPADDR='10.6.1.1'
  MTU=''
  NETMASK='255.255.0.0'
  NETWORK='10.6.0.0'
  REMOTE_IPADDR='10.6.1.2'
  STARTMODE='onboot'
```

4. Before reboot: test your config files:

```
#> hwup lcs-bus-ccw-0.0.e000
```

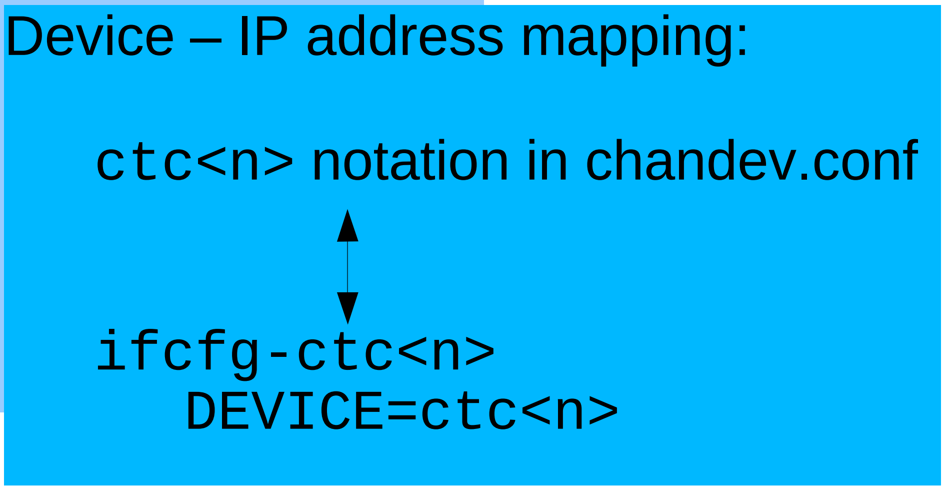
Static CTC Device Setup on Linux 2.4

Hardware configuration is in `/etc/chandev.conf`:

```
ctc0,0xe000,0xe001
```

Interface configuration:

```
/etc/sysconfig/network-scripts/ifcfg-ctc0
  DEVICE=ctc0
  USERCTL=no
  ONBOOT=yes
  BOOTPROTO=none
  BROADCAST=10.6.255.255
  NETWORK=10.6.0.0
  NETMASK=255.255.0.0
  IPADDR=10.6.1.1
  REMOTE_IPADDR=10.6.1.2
  ARP=no
```



Device – IP address mapping:

`ctc<n>` notation in chandev.conf

```
ifcfg-ctc<n>
   DEVICE=ctc<n>
```

Dynamic CTC Device Setup

1. Load the CTC device driver module:

```
#> modprobe ctc
```

2. Create a new CTC device by grouping its CCW devices:

```
#> echo 0.0.e000,0.0.e001 > /sys/bus/ccwgroup/drivers/
   ctc/group
```

3. Set optional attributes:

```
#> echo 0 > /sys/bus/ccwgroup/drivers/ctc/0.0.e000/
   protocol
```

4. Set the new device online:

```
#> echo 1 > /sys/bus/ccwgroup/drivers/ctc/0.0.e000/
   online
```

Dynamic CTC Device Setup (cont.)

5. Find out the interface for your new device:

   At the moment only possible by checking the 'device' link of each `/sys/class/net/ctc*` entry:

```
#>ls -Al /sys/class/net/ctc*/device
lrwxrwxrwx  1 root root 0 Jul  6 11:17
  /sys/class/net/ctc0/device -> ../../../devices/cu3088/0.0.e000
lrwxrwxrwx  1 root root 0 Jul 12 15:14
  /sys/class/net/ctc1/device -> ../../../devices/cu3088/0.0.f000
```

6. Configure your new ctc0 interface:

```
#> ifconfig ctc0 10.6.1.1 pointopoint 10.6.1.2
```

Dynamic CTC Device Setup on Linux 2.4

1. Add definition of the new device to `/etc/chandev.conf`:

```
ctc0,0xe000,0xe001
```

2. Activate the new configuration:

```
#>echo readconf > /proc/chandev
#>echo reprobe > /proc/chandev
```

3. Configure the interface:

```
#> ifconfig ctc0 10.6.1.1 pointopoint 10.6.1.2
```

IUCV Device Driver

- IUCV – Inter User Communication Vehicle
- VM communication facility for inter guest data exchange
- Point to point communication
- Linux device driver builds IP networking semantics on top of IUCV --> NETIUCV

- Recommendation:
  Use GuestLAN where possible

Static IUCV Device Setup

For LINUX 1 iucv0 (see Networking Example)

1. Create a hardware device configuration file:

```
/etc/sysconfig/hardware/hwcfg-iucv-id-linux2:
  STARTMODE="auto"
  MODULE="netiucv"
  MODULE_OPTIONS=""
  MODULE_UNLOAD="yes"
  SCRIPTUP="hwup-iucv"
  SCRIPTDOWN="hwdown-iucv"
```

Note, that the peer user “LINUX2” is specified solely via the file name.

Static IUCV Device Setup (cont.)

2. Create an interface configuration file:

```
/etc/sysconfig/network/ifcfg-iucv-id-linux2:
  BOOTPROTO='static'
  BROADCAST='10.5.255.255'
  IPADDR='10.5.1.1'
  MTU=''
  NETMASK='255.255.0.0'
  NETWORK='10.5.0.0'
  REMOTE_IPADDR='10.5.1.2'
  STARTMODE='onboot'
```

3. Before reboot: test your config files:

```
#> hwup iucv-id-linux2
```

Static IUCV Device Setup on Linux 2.4

Peer VM guests to connect to are specified as kernel parameters:

```
iucv=<vm guest ID>[{:vm guest ID}]
```

e.g.

```
iucv=LINUX2:VMTCPIP
```

Interface configuration:

```
/etc/sysconfig/network-scripts/ifcfg-iucv0
  DEVICE=iucv0
  USERCTL=no
  ONBOOT=yes
  BOOTPROTO=none
  BROADCAST=10.5.255.255
  NETWORK=10.5.0.0
  NETMASK=255.255.0.0
  IPADDR=10.5.1.1
  REMOTE_IPADDR=10.5.1.2
  ARP=no
```

Device – IP address mapping:

position in kernel parameter line

↕

```
ifcfg-iucv<n>
   DEVICE=iucv<n>
```

Dynamic IUCV Device Setup

1. Load the IUCV network device driver module:

```
#> modprobe netiucv
```

2. Create a connection to the peer user:

```
#> echo linux2 > /sys/bus/iucv/drivers/netiucv/
   connection
```

This creates the following sysfs entries:

```
/sys/bus/iucv/devices/netiucv<n>
/sys/devices/iucv/netiucv<n>
/sys/class/net/iucv<n>
```

where *n* is the first free index assigned to the new iucv device
(in our example 0).

Dynamic IUCV Device Setup (cont.)

3. Verify which iucv interface is connected to which user:

```
#> cat /sys/bus/iucv/devices/iucv0/user
linux2
```

4. Configure your new iucv interface:

```
#> ifconfig iucv0 10.5.1.1 pointopoint 10.5.1.2
```

Dynamic IUCV Device Setup on Linux 2.4

Only possible if netiucv is compiled as a loadable module

1. Unload the module. This removes all current connections!

```
#> rmmod netiucv
```

2. Load module and specify all peers as module parameters:

```
#> modprobe netiucv iucv=LINUX2:VMTCPIP
```

3. Configure the interfaces:

```
#> ifconfig iucv0 10.5.1.1 pointopoint 10.5.1.2
#> ifconfig iucv1 ...
```

Summary of Linux Network Device Drivers

	QETH				LCS	CTC	IUCV
	OSA	HiperSockets	GuestLAN QDIO	GuestLAN Hiper			
Adapters	100 Mbps, 1Gbps, 1000 Base-T, HSTR				100 Mbps, 1000 Base-T, HSTR	ESCON, FICON, Virtual CTC/A	
Connection type	LAN	LAN	LAN	LAN	LAN	point-to-point	point-to-point
Protocols	IPv4, IPv6	IPv4	IPv4, IPv6	IPv4	IPv4	IPv4	IPv4
Max bandwith	0.9 Gbps	9.6 Gbps	3.6 Gbps	4.8 Gbps	0.48 Gbps (1000Base-T)	1.2 Gbps (VCTC)	1.44 Gbps
Avg response time	0.9 ms	0.09 ms	0.28 ms	0.24 ms	4.4 ms (1000Base-T)	0.39 ms (VCTC)	0.28 ms
Remarks	**Primary network device driver for Linux on zSeries**				restricted access (admin defines OSA Address Table)		

References

- Linux for zSeries and S/390 on DeveloperWorks

  http://www-128.ibm.com/developerworks/linux/linux390/index.html

- Linux for zSeries and S/390 Documentation

  http://www-128.ibm.com/developerworks/linux/linux390/april2004_documentation.html

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

  http://www-128.ibm.com/developerworks/linux/linux390/useful_add-ons.html

Outlook for Session 2

- Router setup for Linux on zSeries
- Failover and availability solutions:
  - IP Address Takeover
  - Virtual IP Addresses (VIPA)
  - Proxy ARP
- The qethconf tool
- The qetharp tool
- HiperSockets Network Concentrator (HSNC)