Postbank





# Virtual Labs with User Mode Linux

Armin M. Warda, Postbank Systems AG, Germany

IBM eServer pSeries AIX & Linux 2004 Technical Conference, Munich/DE, 26-29 Oct

Deutsche Post World Net
MAIL EXPRESS LOGISTICS FINANCE

# Agenda

Postbank

# Deutsche Postbank AG

- **Postbank – Multi-Channel-Bank**
  - **Leading German Retail-Bank**
    - 11.8 m Customers, 22.5 m Accounts, 9.3 m Cards *
    - 27,011 Customer Care Systems, 14,046 LAN workplaces *
    - 2,079 ATMs, 1,423 bank statement printers *
  - **Leading German Online-Bank**
    - 1.8 m Accounts use Online Banking *
    - 2.8 m Accounts use Telephone Banking *
  - **Leading German Transaction-Bank**
    - 10 m Transactions/day Payment-Processing *
    - plus Insourcing: Deutsche Bank, Dresdner Bank,..


com! Testsieger

10/2004

- **Armin M. Warda**
  - Diplom Informatiker
  - Certified AIX Specialist; pSeries, RS/6000 Tech. Expert
  - Unix-Security Expert, High-Availability Expert
  - Senior IT Infrastructure Architect at Postbank
  - Armin(dot)Warda(at)ePost(dot)De

* 06/2004

# What is User Mode Linux?



- „LinuxPPC“ is a port of the Linux-Kernel on PowerPC
- „zLinux“ is a port of the Linux-Kernel on zSeries
- „UML“ is a port of the Linux-Kernel on Linux

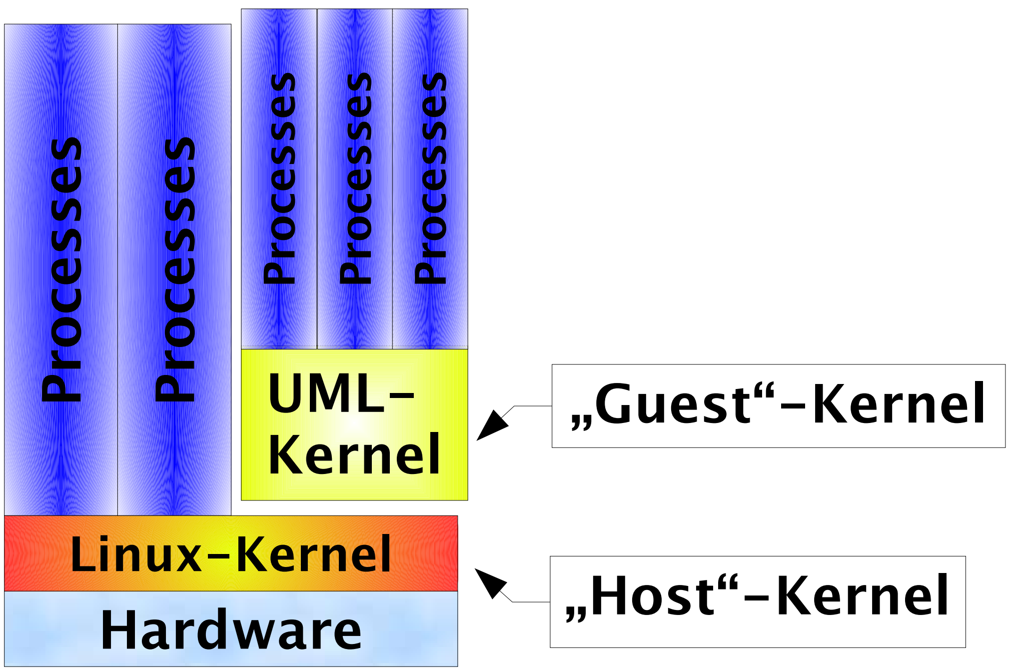
- UML-Creator & -Maintainer: Jeff Dike

- http://www.usermodelinux.org
- http://user-mode-linux.sourceforge.net

- Mailing-List *user-mode-linux-devel*

# What is User Mode Linux?

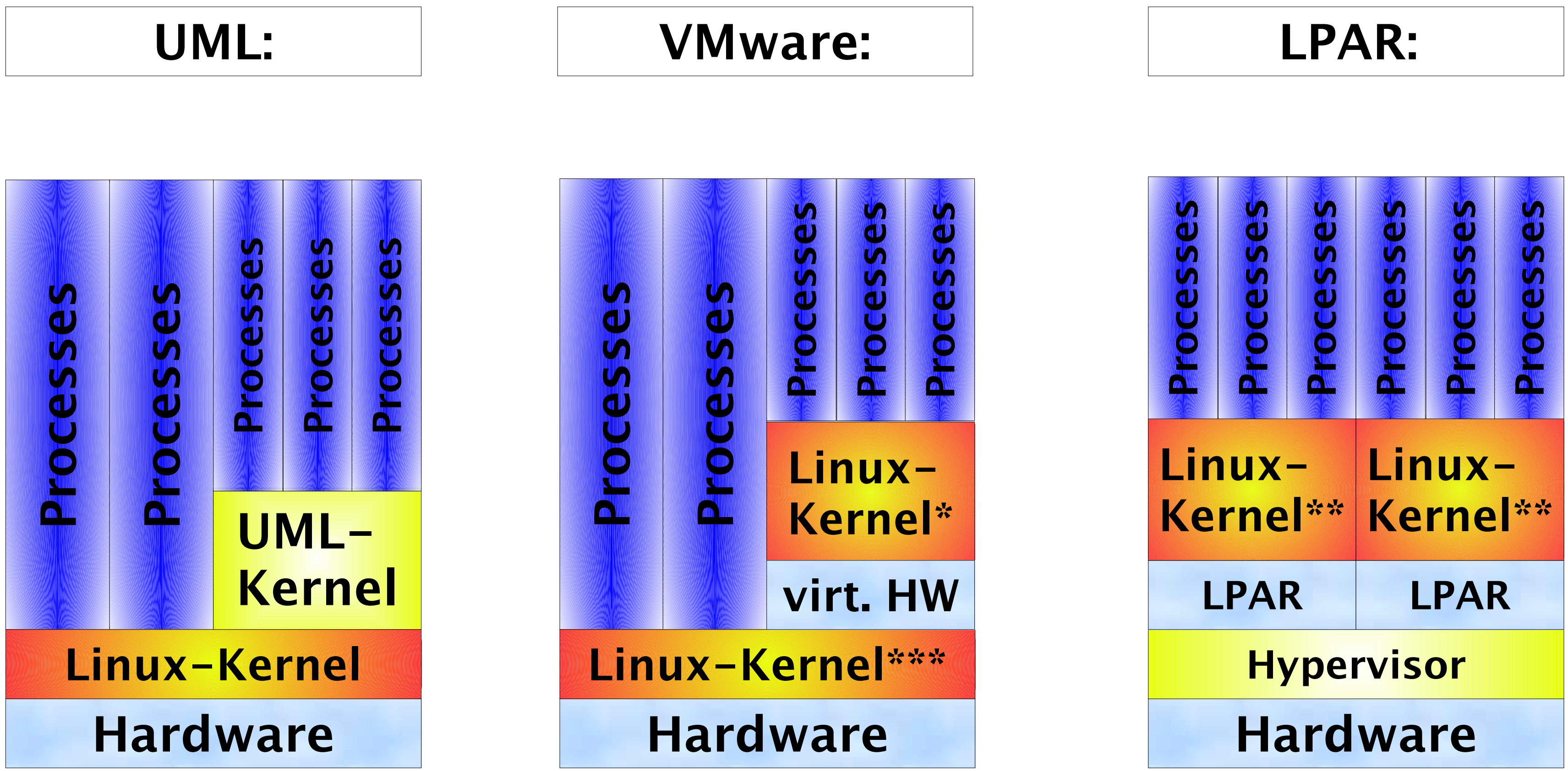

UML = Linux-Kernel running
- as regular Linux-User-Process
- without Root-Privileges




- Host-Kernel is regular Linux-Kernel, e.g. from ftp.kernel.org, Redhat, SuSE,..
- Guest-Kernel is a Kernel with Jeff Dike's UML-patches
- Versions are unrelated, everything can be mixed, e.g.:
  - Host-Kernel = 2.4.21-192 of SuSE 9.0
  - Guest-Kernel = 2.6.4-SMP from ftp.kernel.org with Jeff Dike's patches
- For best performance, the Host-Kernel should contain the SKAS- and /dev/anon-Patch (standard in SuSE, but not in kernel.org 2.4.27)

# What is User Mode Linux?






* or any other supported OS, e.g. Windows, DOS, OS/2, Solaris, BSD, Unixware, Netware,..
** or AIX, i5/OS (PowerPC), or z/OS, z/VM (zSeries)
*** or Windows

# Agenda

Postbank

Agenda

# What is UML good for?

Postbank

- originally developed to ease Kernel-Debugging

- Running UMLs allows instant access to
  - different Distributions (SuSE, Redhat)
  - different Versions (SuSE 8.2, 9.0, 9.1)
  - different Kernels (2.4.25, 2.6.4; UP, SMP)

- and allows to simulate
  - Cluster-Computing
  - complex Network-Setups
  - complex System-Interaction

- Proof-of-Concepts, Prototypes, Demonstration, Education, . . .
- Testing, Developing, Debugging, Virtual Labs, . . .
- Hosting, Sandboxing, Jailing, Honeypots, . . .

UML is a valid alternative to real hardware, if performance and size do not matter that much.

UML is 100% Open Source, Free Software („free“ as in beer, „free“ as in speech)

# What is UML good for?

- ... did you mention Performance? What is UML's performance-penalty?
  - Expect 50% performance of native Linux in best case.

- What is the best case?
  - Compute-intensive, not too many System Calls, not too many Context Switches.

- And what is the worst case?
  - `dd if=... of=...` gives you only 2% of the native System's performance.

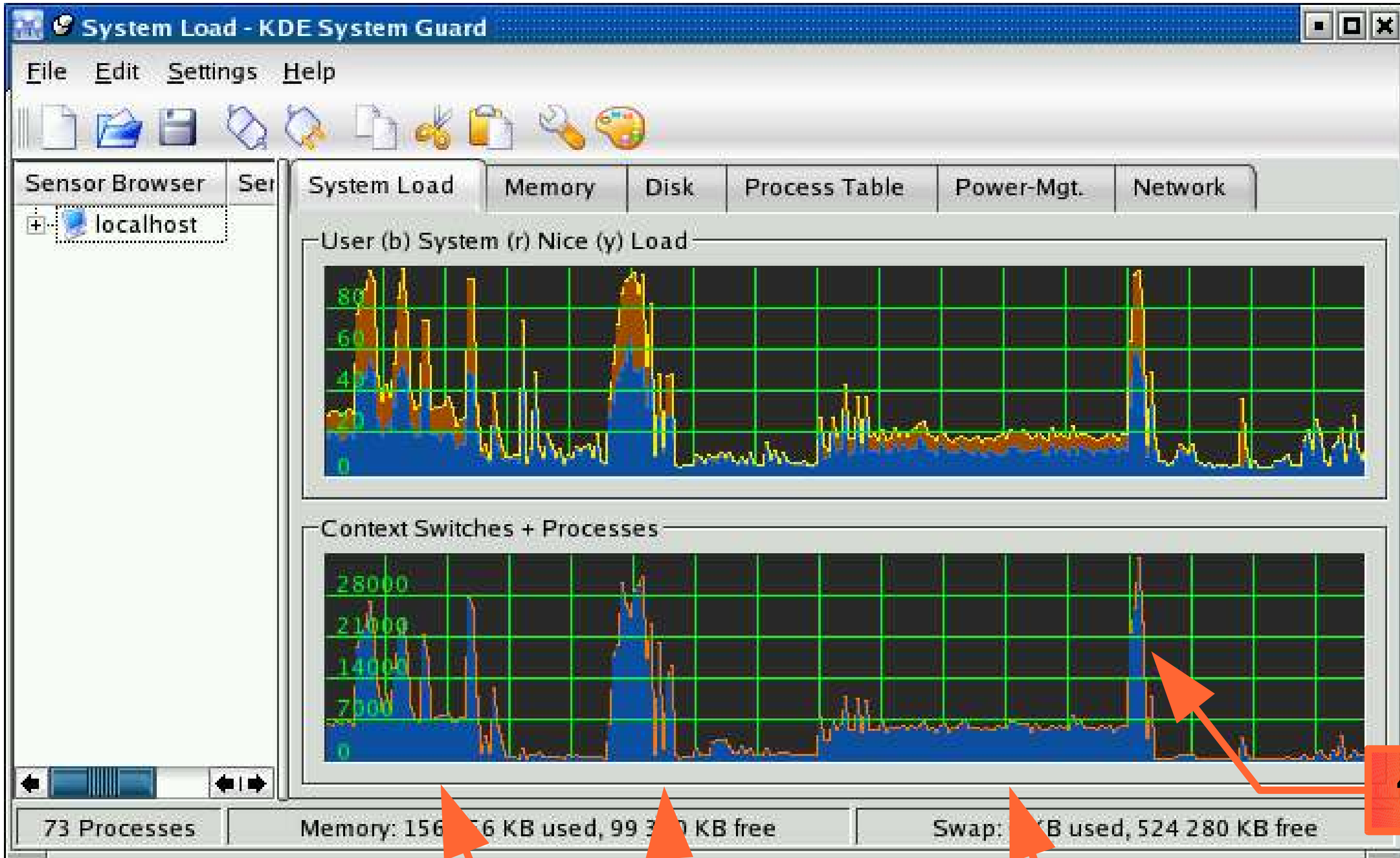
Use real hardware or LPARs, if performance and size do matter.

- *VMware has better performance than UML, but it uses some hacks implemented by Kernel-Modules (vmmon.o + vmnet.o), thus, VMware does not entirely live in User-Mode.*

- *The low performance-penalty of LPAR, typical 2-5%, is still unmatched.*

# What is UML good for?

Postbank




**750 MHz**
**256 KB L1**

**Pentium III**
**(Coppermine)**

**4 UMLs halting**

**3 UMLs idling in Heartbeat + DRBD + NFS plus 2 virtual Switches**

**4 UMLs booting**

**4 UMLs idling in Quagga (OSPF) plus 5 virtual Switches**

# Agenda

Postbank

# How do I setup an UML? (Quick.)

Download a pre-build RPM with UML Kernel and Utilities, e.g.:

- http://prdownloads.sourceforge.net/user-mode-linux/user_mode_linux-2.4.19-5um-0.i386.rpm

Download a pre-build Root-Filesystem, e.g.:

- http://prdownloads.sourceforge.net/user-mode-linux/Debian-3.0r0.ext2.bz2

Download size
- RPM ~1.7 MB
- Root-FS ~22.5 MB

Install RPM, unpack Root-FS, boot your UML!

- `host# rpm -ivh user_mode_linux-2.4.19.5um-0.i386.rpm`
- `host> bunzip2 Debian-3.0r0.ext2.bz2`
- `host> linux ubd0=Debian-3.0r0.ext2`

# Agenda

Postbank

# How do I setup an UML? (from scratch)

Postbank

## Build your own Root-Filesystem!

This is the absolute minimal Root-FS to boot a Linux Kernel:

- `host> dd if=/dev/zero of=minidisk bs=1024k count=5`
- `host> mke2fs minidisk`
- `host# mount -o loop minidisk /mnt`
- `host# mkdir /mnt/dev`
- `host# cp /bin/sash /mnt/sh`

(„sash“ is the Stand-Alone Shell with built-in commands)

- `host# umount /mnt`

```
/
|
+--dev
|
+--lost+found
|
+--sh
```

Ready to go!

- `host> linux ubd0=minidisk init=/sh`

...can't do too many useful things with this system (type „help“ to get list of build-in commands) ...

# How do I setup an UML? (from scratch)

Postbank

... but we can use it to explore the uml_mconsole:

First, at boot, assign an ID to the UML:

- `host> linux umid=mini ubd0=minidisk init=/sh`

Call the uml_mconsole with that ID:

- `host> uml_mconsole mini`
- `(mini) config ubd0`
- `(mini) config ubd1=another.disk`
- `(mini) stop`
- `(mini) go`
- `(mini) sysrq t`
- `(mini) cad`
- `(mini) reboot`
- `(mini) halt`
- `(mini) quit`

(Did you ever work with a Hardware Management Console of an IBM eServer pSeries or zSeries?)

# How do I setup an UML? (from scratch)

Postbank

## Build your own *real* Root-Filesystem!

- **Insert your Linux Distribution Install Media...**
  *e.g. SuSE Linux Professional 9.0 DVD #1*

- **Which RPMs have to be installed?**
  **Look at `/suse/setup/descr/Minimal.sel` on DVD #1, copy that file to `/tmp/Minimal.sel`, edit it to create a minimal list of RPMs you want to install:**

  1. **Delete all lines enclosing the list of RPMs**
  2. **Delete all RPMs that aren't essential (e.g. all yast-*), this should reduce the list to approximately 100 RPMs.**
  3. **cd into /suse/i586/ of DVD #1 and execute**

```
while read f; do ls -1 $f-[0-9]*.rpm; done
 < /tmp/Minimal.sel | sort \
 > /tmp/Minimal.rpms
```

```
...
+Ins:
SuSEfirewall2
aaa_base
aaa_skel
...
...
yast2-users
yast2-xml
zlib
-Ins:
...
```

```
aaa_base
aaa_skel
ash
at
bash
bc
bzip2
...
...
zlib
```

```
aaa_base-9.0-6.i586.rpm
aaa_skel-2003.9.18-4.i586.rpm
ash-0.2-798.i586.rpm
at-3.1.8-782.i586.rpm
bash-2.05b-207.i586.rpm
bc-1.06-643.i586.rpm
bzip2-1.0.2-224.i586.rpm
...
...
zlib-1.1.4-225.i586.rpm
```

# How do I setup an UML? (from scratch)

Create the virtual disk, format it and initialize the RPM database on it:

```
host> dd if=/dev/zero of=myrootfs \
  bs=1024k count=350
host> mke2fs myrootfs
host> mkdir /tmp/altroot
host# mount -o loop myrootfs /tmp/altroot
```

Initialize the RPM db and test-install the RPMs:

```
host# mkdir -p /tmp/altroot/var/lib/rpm
host# rpm --root /tmp/altroot --initdb
host# rpm -ivh --root /tmp/altroot \
  --test $(cat /tmp/Minimal.rpms)
```

Edit list of RPMs and repeat this process until not too many critical errors remain, then:

```
host# rpm -ivh --root /tmp/altroot \
  --nodeps $(cat /tmp/Minimal.rpms)
```

**Install RPMs.**

# How do I setup an UML? (from scratch)

Postbank

A little bit of manual pre-first-boot customization is necessary for the Root-FS:

Of cause, you need the modules compiled for the UML-kernel!

- `host# cd /tmp/altroot`
- `host# cp -a /lib/modules/2.4.22-6um \`
  `lib/modules/`
- `host# mv sbin/blogd sbin/blogd.orig`
- `host# cp  bin/true  sbin/blogd`
- In etc/inittab deactivate mingetty entries with „#“, and insert this enty instead:
  „`C0:2345:respawn:/sbin/agetty ttys/0 9600 xterm`“
- `host# rm etc/rc.d/rc3.d/*`
- `host# rm etc/boot.d/*boot.clock`
- In etc/securetty insert „ `vc/0`“ to allow Root-Login
- `host# umount /tmp/altroot`

**Pre-First-Boot Customization.**

Now we finally may try to boot the UML with our homegrown Root-FS:

- `host> linux umid=myuml ubd0=myrootfs`

Good Luck!

# How do I setup an UML? (from scratch)

After Login as root post-first-boot customization is needed for the Root-FS

- `(none)# echo myuml > /etc/HOSTNAME`
- `(none)# dd if=/dev/zero of=/swapfile \`
  `bs=1024k count=32`
- `(none)# mkswap /swapfile`
- `(none)# echo /swapfile swap swap defaults 0 0 \`
  `>> /etc/fstab`
- `(none)# echo /dev/ubd/0 / ext2 defaults 1 1 \`
  `>> /etc/fstab`
- `(none)# echo "echo 1 > /proc/sys/kernel/sysrq" >>`
  `/etc/rc.d/boot.local`
- `(none)# reboot`

After the Reboot the system can be further customized as you like.

- `myuml#`

**Post-First-Boot Customization.**

# How do I setup an UML? (from scratch)

Postbank

**Build your own UML-Kernel!**

Get Jeff Dike's latest UML-Patch:

- http://prdownloads.sourceforge.net/user-mode-linux/uml-patch-2.4.26-3.bz2

Get Linus Torvald's unpatched kernel-sources:

- ftp://ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.26.tar.bz2

Unpack and patch the kernel-sources:

```
host> mkdir ~/uml; cd ~/uml
host> tar -xjf linux-*.tar.bz2
host> cd ~/uml/linux
host> bzcat uml-patch-*.bz2 | patch -p1
```

**Do not build your UML kernel in /usr/src/linux!**

**And no need to build it as root.**

# How do I setup an UML? (from scratch)

Postbank

Configure the kernel:

- `host> make clean menuconfig ARCH=um`

Don't build a regular kernel, but a UML- (= um) Kernel!

Must use some special kernel-configuration options:

- `CONFIG_USERMODE=y`
- `CONFIG_MCONSOLE=y`
- `CONFIG_NEST_LEVEL=0`
- `...`

Don't build a bzImage, but a (statically linked) ELF executable

Build the kernel:

- `host> make dep linux modules ARCH=um`

See appendix for sample UML kernel configuration!

Strip & copy the kernel:

- `host> strip linux`
- `host> cp linux ~/bin/linux-2.4.26-3um`

Copy the kernel-modules:

- `host> make modules_install INSTALL_MOD_PATH=...`

# How do I setup an UML? (from scratch)

Postbank

**Build your own UML-Utilities!**

Get Jeff Dike's latest UML-Utilities:

- http://prdownloads.sourceforge.net/user-mode-linux/uml_utilities_20040406.tar.bz2

Unpack:

```
host> cd ~/uml
host> tar -xjf uml_utilities-*.tar.bz2
host> cd ~/uml/tools
```

Edit the Makefile to adjust BIN_DIR and LIB_DIR, then:

```
host> make
host> make install
```

**However, the UML-utilities from the rpm should be good enough . . .**

# Summary: How do I setup an UML?

UML setup from scratch:

- build UML-Kernel from Linus' kernel-sources plus Jeff's UML-Patches
- build UML-utilities from Jeff's sources
- build Root-Filesystem, e.g. from Distribution-Media

Alternatively, if you are lazy,
have a big pipe to the internet or lots of time,
you can simply download, install and use:

- a pre-build UML-Kernel plus modules
- a rpm with the UML-utilities
- a pre-build Root-Filesystem

Build

or

Download

# Agenda

Postbank

# How can I simulate Ethernets, Disks,..?

The „uml_switch“ daemon creates a virtual ethernet switch, it must be started prior any UML guest attaching to it:

- `host> uml_switch -unix /tmp/switch-1`

Attach the UML guest to the switch via boot commandline:

- ```
  host> linux umid=myuml ubd0=myrootfs \
              eth0=daemon,,unix,/tmp/switch-1
  ```

unique Unix-Socket for each uml_switch


If you want to attach the UML host to the virtual switch your need root-access to configure a tap-device:

- `host# tunctl -t tap1`
- `Set 'tap1' persistent and owned by uid 0`
- `host# ifconfig tap1 192.168.1.99 down up`
- `host> uml_switch -unix /tmp/switch-1 -tap tap1`

If you want to snoop all packets on the virtual ethernet switch, you should configure it to behave like a hub:

- `host> uml_switch -unix /tmp/switch-1 -tap tap1 -hub`
- `host# tcpdump -i tap1`
  ```

# How can I simulate Ethernets, Disks,..?

Postbank

You already saw:

- Virtual Disk in UML guest = plain file in UML host
- When a UML guest opens a disk, the file is locked in the UML host

  => looks like „AIX-style“ SCSI-Reservation

  ☹ no concurrent access possible

  ☺ but shared (alternating) access is possible

  => enough to build simple failover cluster with UML . . .

# How can I simulate Ethernets, Disks,..?



**You already saw:**

- Virtual Disk in UML guest = plain file in UML host
- When a UML guest opens a disk, the file is locked in the UML host

  => looks like „AIX-style“ SCSI-Reservation

  ☹ no concurrent access possible

  ☺ but shared (alternating) access is possible

  => enough to build simple failover cluster with UML . . .

**Copy-On-Write (COW) Disks:**

- a read-only <u>background</u> file with fixed contents, plus
- a read/write <u>sparse</u> COW file recording the changes


**Start UML guest with a COW Disk:**

- `host> linux ubd0=myrootfs.cow,myrootfs.bg`

# How can I simulate Ethernets, Disks,..?

## What is a sparse file?

To understand this, create a sparse file:

- `host> dd if=/dev/zero of=sparsefile seek=99999 count=1`
- `1+0 records in`
- `1+0 records out`

With „seek=99999 count=1“ we instructed „dd“ to write only the 100.000th block of the file.

The nominal file size is 51.200.000 bytes = 100.000 blocks:

- `host> ls -la sparsefile`
- `-rw-r--r--    1 armin    users    51200000 Sep 23 22:14 sparsefile`

But the actual disk utilization is only 20 blocks (= 10240 bytes):

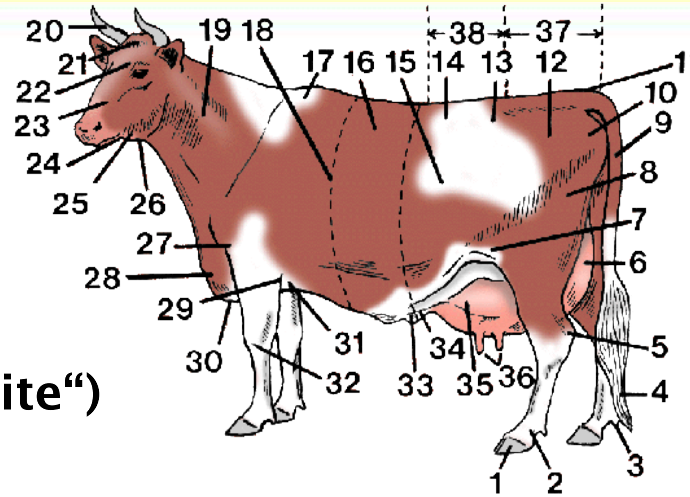
- `host> du sparsefile`
- `20      sparsefile`

(Obviously, the 99.999 empty blocks are „compressed“ into 19 blocks.)

# How can I simulate Ethernets, Disks,..?

Postbank

## How does this COW work?




- **Writes** are always directed to the sparse COW file. If necessary the block if copied from the background file before the write happens. (= „Copy-On-Write")

- **Reads** are satisfied from the sparse file, if the block exists there (= has been written before). Otherwise the Read is satisfied from the background file.

Multiple sparse COW files can share the same background file.
=> this is very space-efficient for virtual clusters of similar nodes!
=> this even improves disk performance (file-caching in the UML host)

It is possible to merge a sparse COW file with its background file to create a consolidated background file:

- `host> uml_moo -b myrootfs.bg myrootfs.cow myrootfs.bg.new`

# How can I simulate Ethernets, Disks,..?

Multiple sparse COW files can **share the same background file**:

```
host> ls -la rootfs.node0[0-5]
-r--------    1 armin    402653184 Feb 27  2004 rootfs.node00
-rw-r--r--    1 armin    402759680 Sep 23 21:29 rootfs.node01
-rw-r--r--    1 armin    402759680 Sep 22 23:11 rootfs.node02
-rw-r--r--    1 armin    402759680 Sep 22 23:11 rootfs.node03
-rw-r--r--    1 armin    402759680 Sep 22 23:11 rootfs.node04
-rw-r--r--    1 armin    402759680 Mar 12  2004 rootfs.node05

host> du rootfs.node0[0-5]
393604  rootfs.node00
13988   rootfs.node01
13896   rootfs.node02
13592   rootfs.node03
13560   rootfs.node04
10768   rootfs.node05
```

```
host> strings rootfs.node01 | head -2 | tail -1
/guests/rootfs.node00
```

(obviously, rootfs.node00 ist the background file for the sparse files node0[1-5])

# How can I simulate Ethernets, Disks,..?

Postbank

**What COWs can't do . . .**

- You cannot layer multiple sparse files.
- You cannot „moo“ multiple sparse files into the same background file.
- When you modify a background file, then all related sparse COW files become invalid!

**Example:** if I would „moo“ rootfs.node01 back into rootfs.node00, then rootfs.node0[2-5] would become invalid (and rootfs.node01 would have to be replaced by a new, empty sparse COW file)

**Finally, how can I simulate serial lines (ttys)?**

- `host> linux ... ssl=pts ...`

serial line in the guest = pts in the host

- `guest# stty -F /dev/serial/1 9600`
- `guest# dmesg | grep Serial`
  `Serial line 1 assigned device '/dev/pts/10'`
- `guest# cat < /dev/serial/1`
- `host> echo Hello UML > /dev/pts/10`

# Agenda

Postbank

# How can I clone, rollback, share, . . ?



**Clone** =

- Copy the file representing the virtual disk
- Or copy the sparse COW file

**Rollback** =

- Move the copy back over the original file
- Or remove the sparse COW file

**Share** =

- Create an archive of all virtual disk files and a script to start the UML guests + switches

- Clone
- Rollback
- Share

**Attention!** If you use COW disks you have to be sure that your archiver knows how to handle sparse files correctly! e.g. „tar“ needs the „-S“ option to handle sparse files

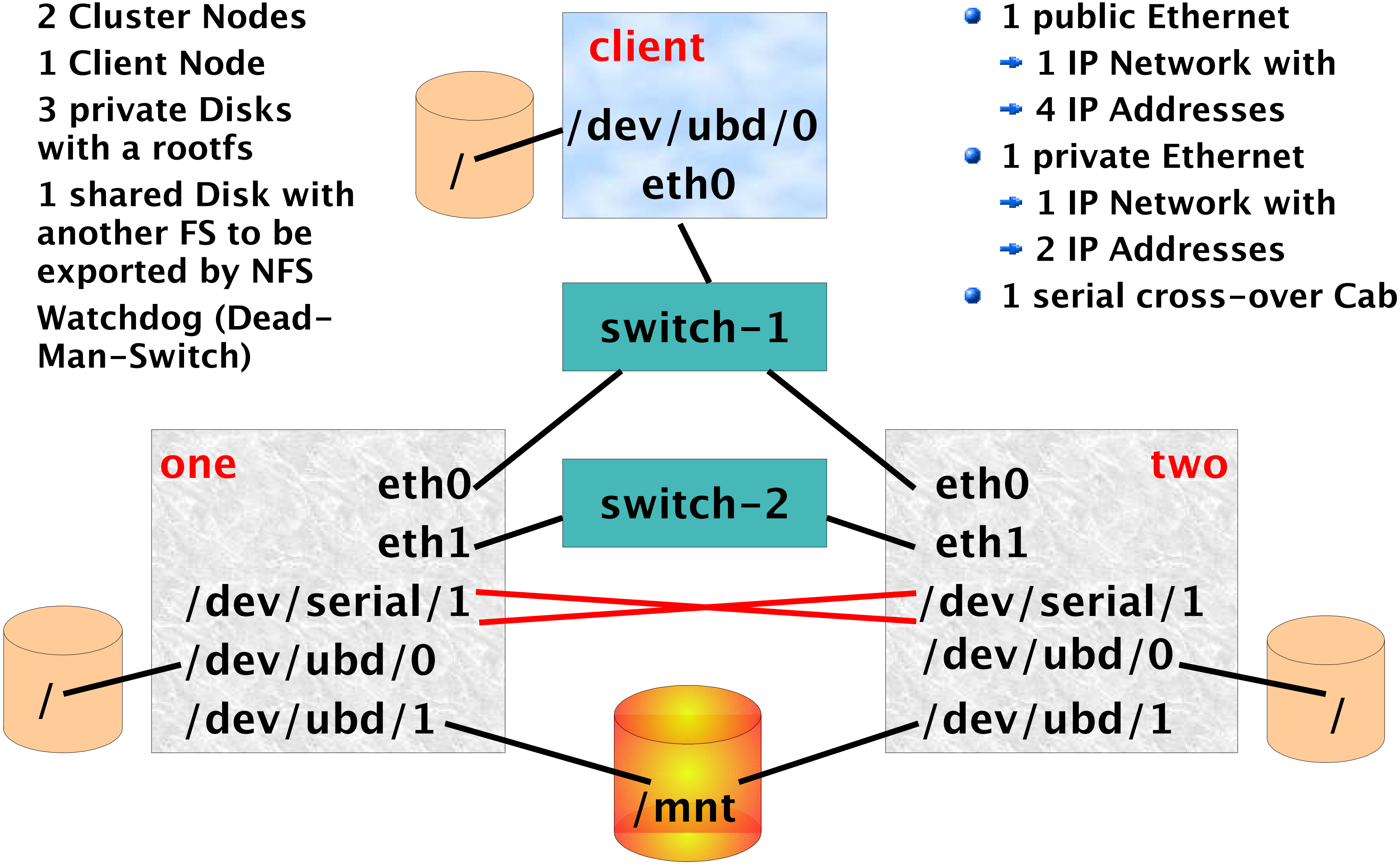
(IBM Tivoli Storage Manager handles sparse files correctly by default.)

# Agenda

Postbank

# Case Study 1: High-Availability Cluster

- 2 Cluster Nodes
- 1 Client Node
- 3 private Disks with a rootfs
- 1 shared Disk with another FS to be exported by NFS
- Watchdog (Dead-Man-Switch)




- 1 public Ethernet
  - 1 IP Network with
  - 4 IP Addresses
- 1 private Ethernet
  - 1 IP Network with
  - 2 IP Addresses
- 1 serial cross-over Cable

# Case Study 1: High-Availability Cluster

Postbank

- Start the virtual Ethernet switches:
  - `host> uml_switch -unix /tmp/switch-1`
  - `host> uml_switch -unix /tmp/switch-2`
- Start the Cluster Nodes:
  - ```
    host> linux umid=one ubd=2 \
      ubd0=rootfs.one,rootfs.node00 \
      ubd1=shared.disk \
      eth0=daemon,,unix,/tmp/switch-1 \
      eth1=daemon,,unix,/tmp/switch-2 \
      ssl=pts mem=32m ro
    ```
- Replace „one“ by „two“ to start the other Cluster Node.
- Start the Client Node:
  - ```
    host> linux umid=client ubd=1 \
      ubd0=rootfs.client,rootfs.node00 \
      eth0=daemon,,unix,/tmp/switch-1 \
      mem=32m ro
    ```
- Now logon to all three Nodes as root!

**Start all virtual hardware components in dedicated windows!**

# Case Study 1: High-Availability Cluster

Postbank

- Initialize serial interface of the Cluster Node and find out the name of the serial's pts-backend at the host:
  - `one# stty -F /dev/serial/1 9600`
  - `one# dmesg | grep Serial`
    `Serial line 1 assigned device '/dev/pts/10'`
- Do the same on the other Cluster Node. Note the pts.
- On the Host start the virtual cross-over cable:
  - `host> cat < /dev/pts/10 >> /dev/pts/11`
  - `host> cat < /dev/pts/11 >> /dev/pts/10`
- Start the Software-Watchdog-Timer (Dead-Man-Switch)
  - `one# modprobe softdog soft_margin=9`
  - `one# grep -w misc /proc/devices`
  - `one# grep -w watchdog /proc/misc`
  - `one# mknod /dev/watchdog c 10 130`
- All virtual Hardware is now up and running!
- Next we should test out Setup!

**Setup and connect the virtual serial interfaces.**

**Initialize the SWT (alias DMS).**

# Case Study 1: High-Availability Cluster

- Configure the IP interfaces:
  - `one# ifconfig eth0 192.168.1.101 up`
  - `one# ifconfig eth1 192.168.2.101 up`
  - `two# ifconfig eth0 192.168.1.102 up`
  - `two# ifconfig eth1 192.168.2.102 up`
  - `client# ifconfig eth0 192.168.1.103 up`
- Test IP connectivity with ping.
- Test the serial interface:
  - `one# cat < /dev/serial/1`
  - `two# echo Hello UML >> /dev/serial/1`

  (You must interrupt the cat with Ctrl-C.)
- Do the same the other way around

**Test IP- & Serial-Connectivity**

# Case Study 1: High-Availability Cluster

- Final Preparation, start the RPC-portmapper on all nodes (NFS needs it):
  - `one# /etc/rc.d/portmap start`

- Now you should manually test all commands that later the Cluster-Manager is supposed to do:

- On Node one, configure the service IP address, mount the filesystem and start the NFS server:
  - `one# ifconfig eth0:0 192.168.1.100 up`
  - `one# mount /dev/ubd/1 /mnt`
  - `one# /etc/rc.d/nfsserver start`

**Manually bring resources online and access them form the Client.**

- Access the filesystem from the Client Node:
  - `client# mount 192.168.1.100:/mnt /mnt`
  - `client# while true; do date; ls /mnt; sleep 1; done`

- Now we want to do a manual failover to the other Node!

# Case Study 1: High-Availability Cluster



- Manual failover:
  - `one# /etc/rc.d/nfsserver stop`
  - `one# umount /mnt`
  - `one# ifconfig eth0:0 down`

  - `two# ifconfig eth0:0 192.168.1.100 up`
  - `two# mount /dev/ubd/1 /mnt`
  - `two# /etc/rc.d/nfsserver start`

Manually failover the resoures to the other Node.

- The Client receives a „nfs server not responding, still trying“ message and hangs for a while.
- Finally the Client receives a „nfs server ok“ message and continues.
- If you mix-up the order of the commands, then you risk a „stale NFS file handle“ message.

- Now we are set to transfer control to the Cluster-Manager!

# Case Study 1: High-Availability Cluster

- Bring all resources offline on all Cluster Nodes: nfsserver, IP-address, filesystem (you can leave the Client hanging with its „nfs server not responding, still trying“ message)

- Start the cluster manager on both nodes:
  - `one# /etc/rc.d/heartbeat start`
  - `two# /etc/rc.d/heartbeat start`

  Wait for stabilization...

- Initiate a failover:
  - `one# hb_standby`
- Wait for stabilization... and initiate a failback:
  - `two# hb_standby`
- Wait for stabilization... and stop the cluster:
  - `two# /etc/rc.d/heartbeat stop`
  - `one# /etc/rc.d/heartbeat stop`

**Regular Cluster Operations**

# Case Study 1: High-Availability Cluster

Postbank

- Start the Cluster-Manage on both Nodes, wait for stabilization, then kill the active Cluster Node, e.g.:
  - `one# halt -f`
- ✯ Do the resources failover to Node two?
- Boot the killed Node, then look at /dev/ubd/, what is missing?
- ✯ Can you explain this?
- Thus, we cannot failback the resources by simply starting the Cluster Manager on Node one and calling hb_standby, we need a slightly extended downtime for this:
- Stop resources on Node two:
  - `two# /etc/rc.d/heartbeat stop`
- Reconfigure Node one via the UML-Machine-Console:
  - `host> uml_mconsole one`
  - `(one) remove ubd1`
  - `(one) config ubd1=shared.disk`
- Now we can restart the Cluster:
  - `one# /etc/rc.d/heartbeat start`
  - `two# /etc/rc.d/heartbeat start`

Simulating Faults:

Node failure.

# Case Study 1: High-Availability Cluster

Postbank

- From within the UML:
  - `one# reboot -f`
- With the Dead-Man-Switch / Software-Watchdog-Timer:
  - `one# while true; do : ; done &`
    `[1] 4711`
  - `one# set_sched_fifo 4711 99`
- UML-Machine-Console:
  - `host> uml_mconsole one`
  - `(one) stop`
  - `(one) reboot`
  - `(one) sysrq b`
- Have to have enabled Magic System Request Key Hacks:
  - `one# echo 1 >> /proc/sys/kernel/sysrq`
- From the host:
  - Send SIGTERM, SIGKILL,.. to pids of UML, attach debugger,...

More ways to die...

# Case Study 1: High-Availability Cluster

Postbank

## Simulating Network faults:

- Note the pids of the virtual Ethernet switches:
  - `one# ps -ef | grep switch`

  You can suspend and resume the processes implementing the virtual Ethernet switches by sending them Signals SIGSTOP and SIGCONT.
- Suspend the private Ethernet switch!
- Resume the private Ethernet switch!
- Suspend the public Ethernet switch!
- Suspend all Ethernet switches and the serial cross-over cable!
- ★ Did you know how a Split-Brain or Node-Isolation feels like?

**Simulating Faults:**

**Network failure.**

# Case Study 1: High-Availability Cluster

Postbank

- Sniffing can be very important to diagnose problems or to gain a deeper understanding!

- Real Sniffer are expensive
(at least non-Ethernet-Sniffers)

- Start the virtual Ethernet Switches as Hubs
and tap the switch with tcpdump...
  - `host> uml_switch -hub -tap tap0 -unix /tmp/switch-1`
  - `host> tcpdump -i tap0`

  ... or – even better – use Ethereal!

- To sniff the serial lines, replace
  - `host> cat < /dev/pts/10 >> /dev/pts/11`

  with
  - `host> cat < /dev/pts/10 | tee -a /dev/pts/11`

**Sniffing**
- Ethernet
- Serial Lines

# Agenda

Postbank

# Case Study 2: Dynamic OSPF Routing

- static routing = admin defines routes manually
  dynamic routing = daemons communicate & calculate routes

- OSPF = Open Shortest Path First
- OSPV v2 = RFC2328 (OSPF for IPv4 unicast routing)

- is an IGP = Interior Gateway Protocol:
  designed to be run internal to a single
  Autonomous System
- (an EGP/BGP = Exterior/Border Gateway Protocol
  is designed to be run at the border between
  Autonomous Systems)

- OSPF supports Equal-Cost Multipath Routes
  (but the kernel must support it, too)

- RFC2328
- IGP
- Multipathing

# Case Study 2: Dynamic OSPF Routing

- OSPF is a Link-State Routing Protocol:
  each OSPF router maintains an identical database
  describing the Autonomous System's topology

- neighbor routers exchange „Hellos“ (= Heartbeats)
- routers exchange „Link-State-Advertisements“

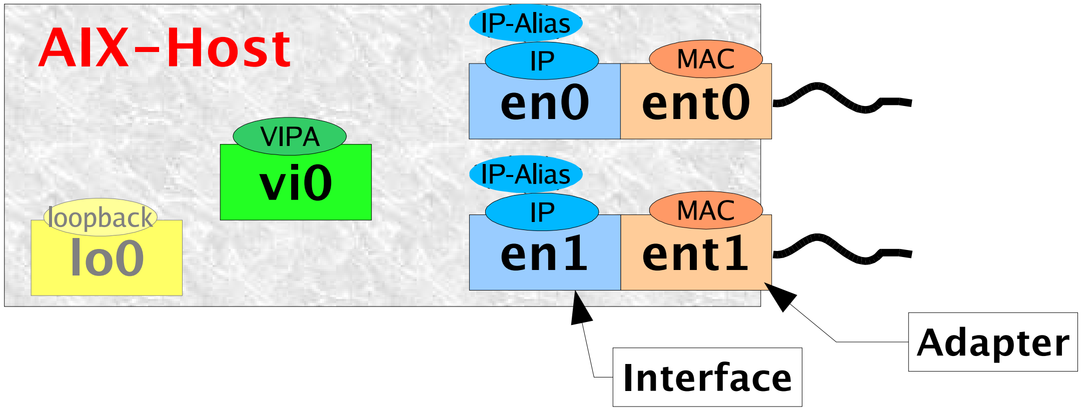
- By applying the Shortest-Path-First Algorithm (E. Dijkstra) to the link-state database, paths with least „cost“ are computed and routes are derived for injection into the kernel forwarding table

- relevant OSPF-Implementations for IBM eServer operating systems:
  - AIX: gated
  - z/OS: OMPROUTE
  - Linux: Quagga (Zebra)

- Link-State
- Dijkstra
- Quagga

# Case Study 2: Dynamic OSPF Routing

- OSPF is most useful together with VIPA:
  - Virtual IP Address
  - not attached to a physical adapter
  - best used as Source and Destination IP Address



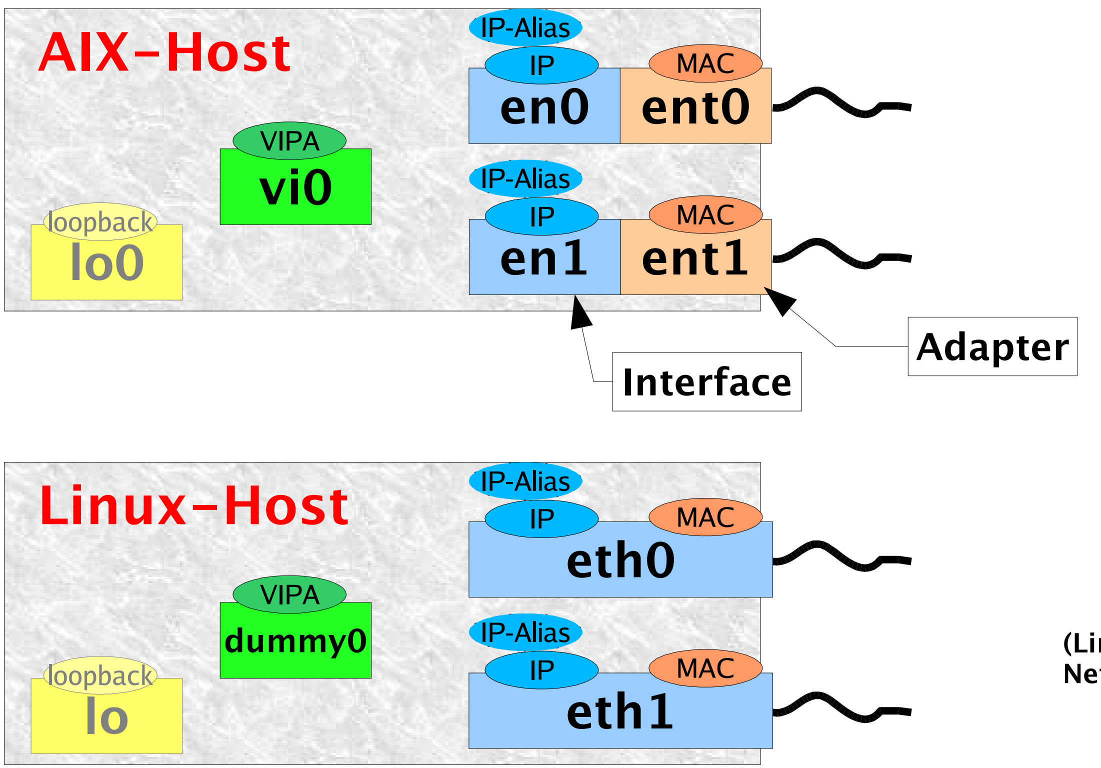

VIPA
- AIX
- . . .

# Case Study 2: Dynamic OSPF Routing

Postbank

- OSPF is most useful together with VIPA:
  - Virtual IP Address
  - not attached to a physical adapter
  - best used as Source and Destination IP Address

VIPA
- AIX
- Linux

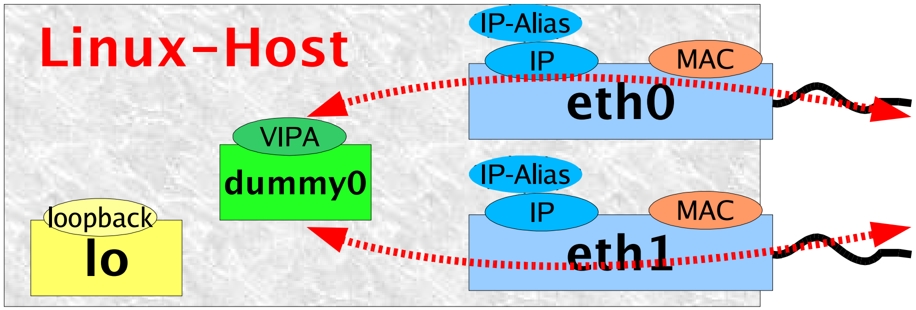



(Linux does not distinguish between Network- „Adapter“ and „Interface“)

# Case Study 2: Dynamic OSPF Routing

- **VIPAs support multipathing*:**

* Kernel must support it, too



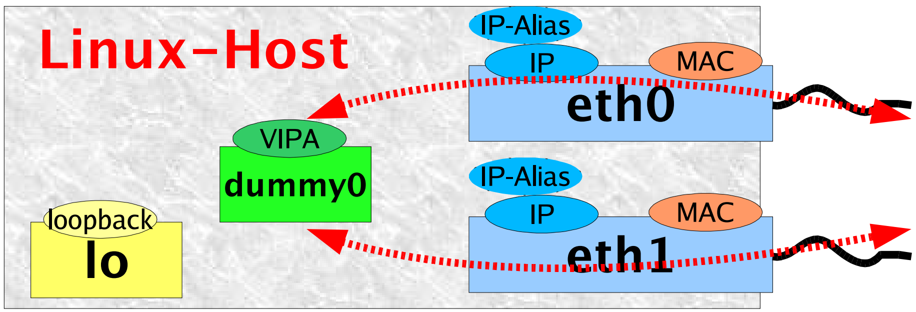

- VIPA
- Multipathing
- ...
- ...

# Case Study 2: Dynamic OSPF Routing

Postbank

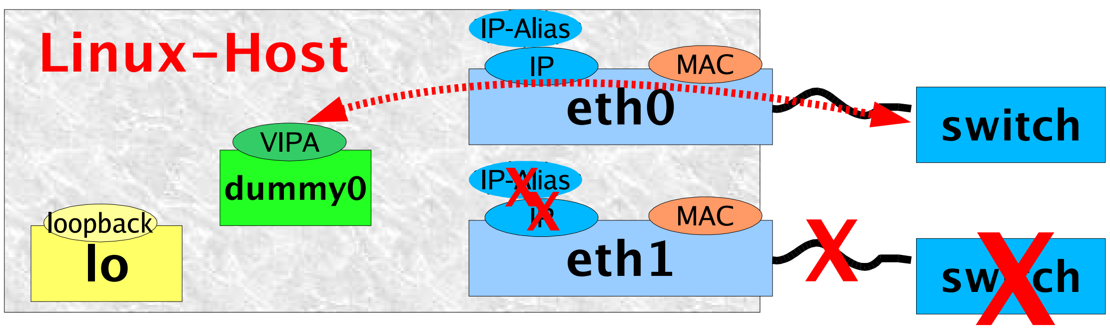
- VIPAs support multipathing*:

* Kernel must support it, too




- VIPA
- Multipathing
- Redundancy
- zSeries GDPS

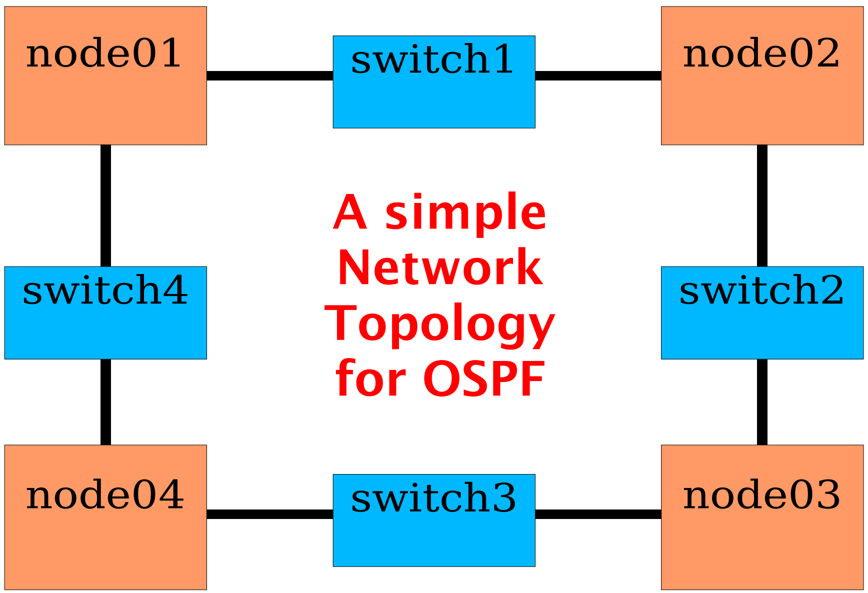
- VIPAs can survive single adapter/switch failures:




- IBM zSeries GDPS (Geographically Dispersed Parallel Sysplex = „z/OS-Mainframe-Cluster“) is based on OSPF + VIPA

# Case Study 2: Dynamic OSPF Routing



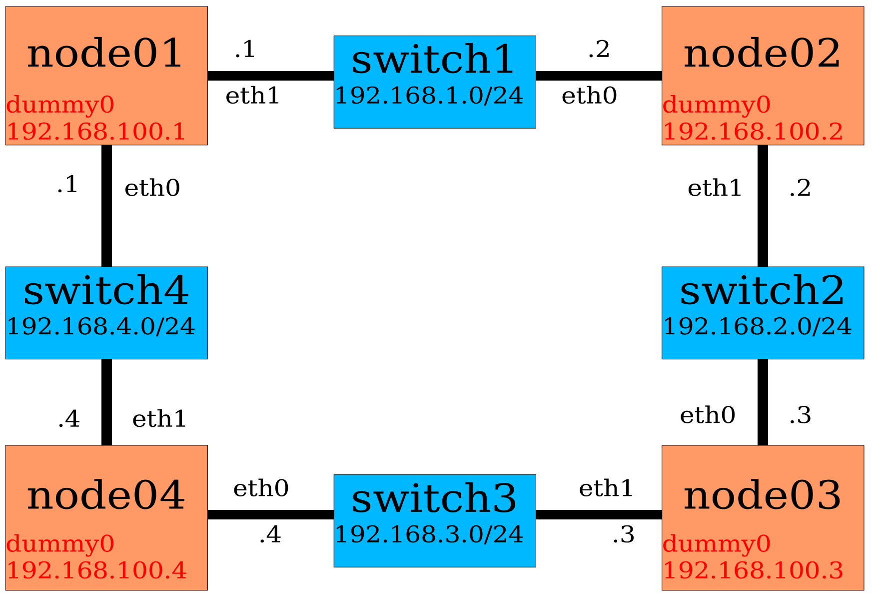



TCP/IP over Ethernet and ISO/OSI Layers:

7 \
6 } Application
5 /
4 TCP/UDP/ICMP
3 IP
2 Ethernet
1 Wire

- All components participating in IP-Forwarding (= Layer 3) are called routers, thus the nodes are routers,
- but the Ethernet-Switches (= Layer 2) are <u>not</u> routers!

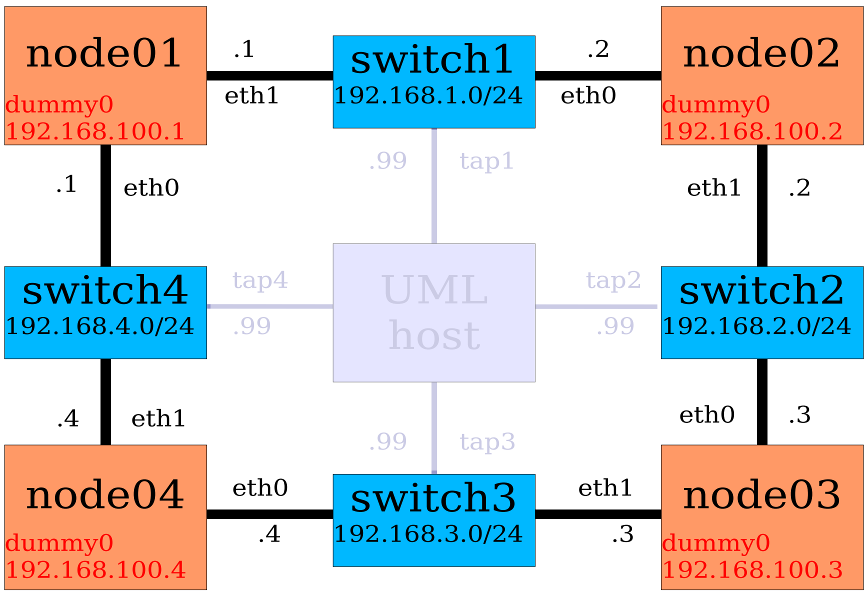
# Case Study 2: Dynamic OSPF Routing




**Planning the IP network topology**

- note names of interfaces
- assign IP-networks to switches
- assign IP-addresses to interfaces
- define VIPAs

# Case Study 2: Dynamic OSPF Routing




**Planning the taps for IP snooping**

- use taps to attach the UML host to all switches
- the switches will be configured to behave like hubs, thus we will be able to snoop <u>all</u> packets from the UML host
- (the host will be passive, it will not run an OSPF daemon)

# Case Study 2: Dynamic OSPF Routing



## Configure the taps:

```
sudo tunctl -u 500 -t tap1; sudo ifconfig tap1 192.168.1.99 up
sudo tunctl -u 500 -t tap2; sudo ifconfig tap2 192.168.2.99 up
sudo tunctl -u 500 -t tap3; sudo ifconfig tap3 192.168.3.99 up
sudo tunctl -u 500 -t tap4; sudo ifconfig tap4 192.168.4.99 up
```


## Start the virtual switches and attach the taps:

```
xterm -geometry 40x5+010+100 -bg     green  +sb -title switch-1 \
                              -e uml_switch -unix /tmp/switch-1 -tap tap1 -hub &
xterm -geometry 40x5+010+200 -bg     green  +sb -title switch-2 \
                              -e uml_switch -unix /tmp/switch-2 -tap tap2 -hub &
xterm -geometry 40x5+010+300 -bg     green  +sb -title switch-3 \
                              -e uml_switch -unix /tmp/switch-3 -tap tap3 -hub &
xterm -geometry 40x5+010+400 -bg     green  +sb -title switch-4 \
                              -e uml_switch -unix /tmp/switch-4 -tap tap4 -hub &
```

## Start sniffers on the taps:

```
xterm -geometry 65x5+275+100 -bg lightgreen +sb -title sniffer-1 \
                                                 -e sudo tcpdump -i tap1 &
xterm -geometry 65x5+275+200 -bg lightgreen +sb -title sniffer-2 \
                                                 -e sudo tcpdump -i tap2 &
xterm -geometry 65x5+275+300 -bg lightgreen +sb -title sniffer-3 \
                                                 -e sudo tcpdump -i tap3 &
xterm -geometry 65x5+275+400 -bg lightgreen +sb -title sniffer-4 \
                                                 -e sudo tcpdump -i tap4 &
```

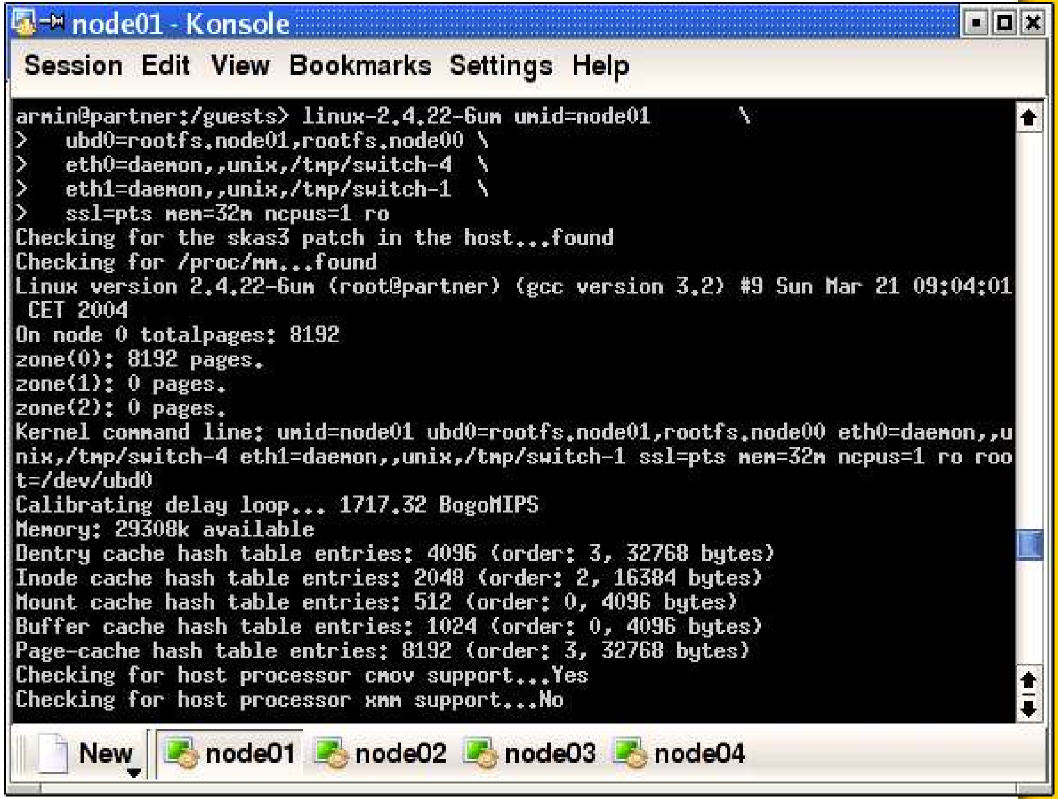
# Case Study 2: Dynamic OSPF Routing

Start a KDE-Konsole with four sessions using a predefined profile:

- `konsole --profile ospf &`

Start the UML-guests in these four session of KDE-Konsole:

- ```
  linux-2.4.22-6um umid=node01         \
    ubd0=rootfs.node01,rootfs.node00 \
    eth0=daemon,,unix,/tmp/switch-4  \
    eth1=daemon,,unix,/tmp/switch-1  \
    mem=32m ncpus=1 ro
  ```
- ```
  linux-2.4.22-6um umid=node02         \
    ubd0=rootfs.node02,rootfs.node00 \
    eth0=daemon,,unix,/tmp/switch-1  \
    eth1=daemon,,unix,/tmp/switch-2  \
    mem=32m ncpus=1 ro
  ```
- ```
  linux-2.4.22-6um umid=node03         \
    ubd0=rootfs.node03,rootfs.node00 \
    eth0=daemon,,unix,/tmp/switch-2  \
    eth1=daemon,,unix,/tmp/switch-3  \
    mem=32m ncpus=1 ro
  ```
- ```
  linux-2.4.22-6um umid=node04         \
    ubd0=rootfs.node04,rootfs.node00 \
    eth0=daemon,,unix,/tmp/switch-3  \
    eth1=daemon,,unix,/tmp/switch-4  \
    mem=32m ncpus=1 ro
  ```

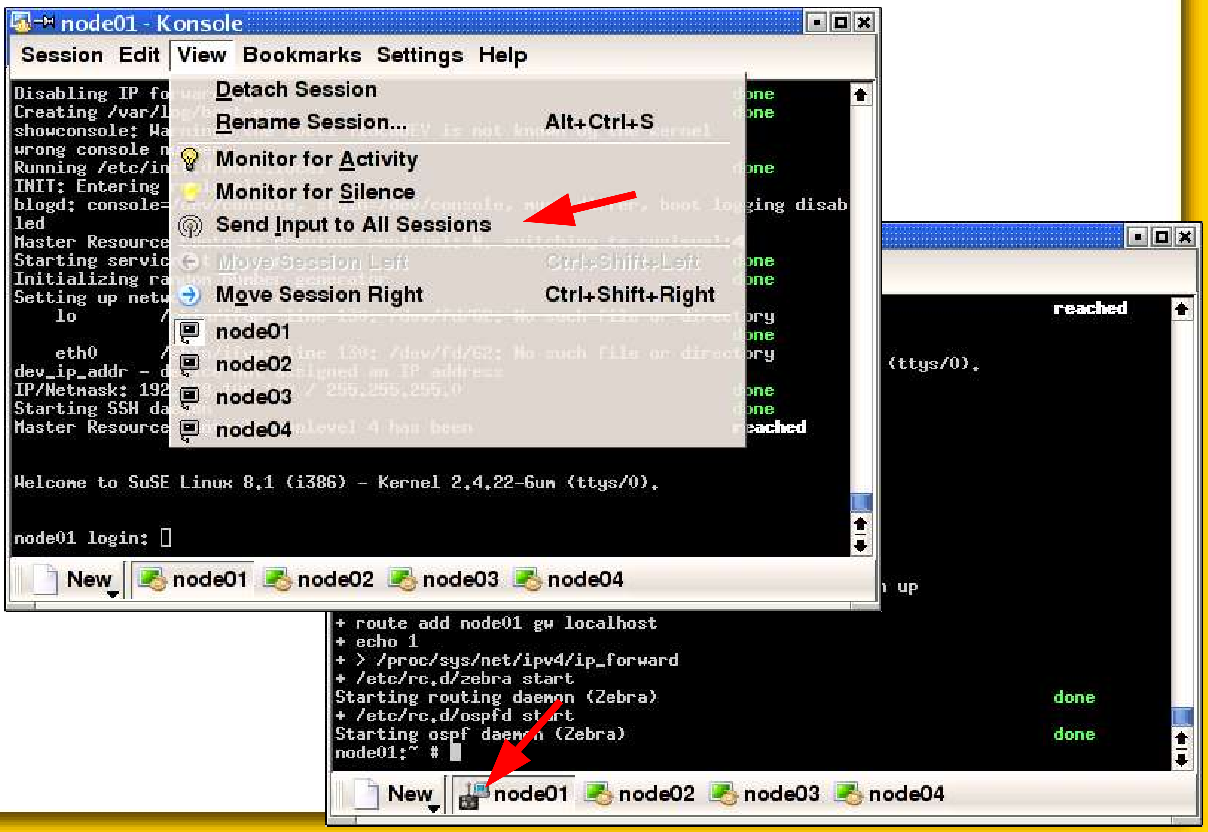


# Case Study 2: Dynamic OSPF Routing

## Log into all sessions and configure the network and start the daemons:

- `ifconfig eth0   $(hostname)-eth0 down up`
- `ifconfig eth1   $(hostname)-eth1 down up`
- `ifconfig dummy0 $(hostname)      netmask 255.255.255.255 down up # VIPA`
- `route add $(hostname) gw localhost`
- `echo 1 > /proc/sys/net/ipv4/ip_forward`
- `/etc/rc.d/zebra start`
- `/etc/rc.d/ospfd start`

**It's very convenient to use the „Send Input to all Sessions“ Mode of the KDE-Konsole:**



# Case Study 2: Dynamic OSPF Routing

```
sniffer-3
id node04 backbone dr node03-eth1 [tos 0xc0]  [ttl 1]
23:00:31.256679 node03-eth1 > AllSPFrouters: OSPFv2-hello 48: rtr
id node03 backbone dr node03-eth1 bdr node04-eth0 [tos 0xc0]  [tt
l 1]
```

## Observe the sniffers!

## Have a look at the kernel forwarding tables after OSPF convergence (= stabilization):

- `node01:~ # netstat -r`

Kernel IP routing table

| Destination | Gateway | Genmask | Flags | MSS | Window | irtt | Iface |
|---|---|---|---|---|---|---|---|
| node01 | localhost | 255.255.255.255 | UGH | 0 | 0 | 0 | lo |
| node02 | node02-eth0 | 255.255.255.255 | UGH | 0 | 0 | 0 | eth1 |
| node03 | node04-eth1 | 255.255.255.255 | UGH | 0 | 0 | 0 | eth0 |
| node04 | node04-eth1 | 255.255.255.255 | UGH | 0 | 0 | 0 | eth0 |
| switch4 | * | 255.255.255.0 | U | 0 | 0 | 0 | eth0 |
| switch3 | node04-eth1 | 255.255.255.0 | UG | 0 | 0 | 0 | eth0 |
| switch2 | node02-eth0 | 255.255.255.0 | UG | 0 | 0 | 0 | eth1 |
| switch1 | * | 255.255.255.0 | U | 0 | 0 | 0 | eth1 |

**As you see, packets from node01 to node03 should be routed through node04.**

(obviously, no multipathing is in use here)

# Case Study 2: Dynamic OSPF Routing

Postbank

## Ping node03 from node01 with option „record-route“:

- `node01:~ # ping -R node03`

```
PING node03 (192.168.100.3) from 192.168.4.1 : 56(124) bytes of data.
64 bytes from node03 (192.168.100.3): icmp_seq=1 ttl=63 time=4.07 ms
RR:     node01-eth0 (192.168.4.1)
        node04-eth0 (192.168.3.4)
        node03 (192.168.100.3)
        node03 (192.168.100.3)
        node04-eth1 (192.168.4.4)
        node01-eth0 (192.168.4.1)
64 bytes from node03 (192.168.100.3): icmp_seq=2 ttl=63 time=2.60 ms    (same route)
64 bytes from node03 (192.168.100.3): icmp_seq=3 ttl=63 time=3.30 ms    (same route)
...
```

**This confirms that node04 is indeed the transit node for IP packets from node01 to node03.**

**Now take node04-eth0 down and observe what happens to the active pings on node01!**

- `node04:~ # sleep 10; ifconfig eth0 down   # ...or: halt -f`

# Case Study 2: Dynamic OSPF Routing

Postbank

**Observe how packets are rerouted after a short delay via node02 to bypass the failed transit node:**

```
...
64 bytes from node03 (192.168.100.3): icmp_seq=12 ttl=63 time=2.50 ms   (same route)
64 bytes from node03 (192.168.100.3): icmp_seq=13 ttl=63 time=2.60 ms   (same route)
From node04-eth1 (192.168.4.4) icmp_seq=14 Destination Net Unreachable
From node04-eth1 (192.168.4.4) icmp_seq=15 Destination Net Unreachable
From node04-eth1 (192.168.4.4) icmp_seq=16 Destination Net Unreachable
64 bytes from node03 (192.168.100.3): icmp_seq=17 ttl=63 time=3.28 ms
RR:     node01-eth1 (192.168.1.1)
        node02-eth1 (192.168.2.2)
        node03 (192.168.100.3)
        node03 (192.168.100.3)
        node02-eth0 (192.168.1.2)
        node01-eth1 (192.168.1.1)

64 bytes from node03 (192.168.100.3): icmp_seq=18 ttl=63 time=2.65 ms   (same route)
64 bytes from node03 (192.168.100.3): icmp_seq=19 ttl=63 time=2.45 ms   (same route)
...
```

**Convergence is very fast in this situation, only 3s, because communication is still possible. If you kill the node (instead of the interface only), convergence will be significantly longer, because no communication with the killed node is possible, but still <1min.**

Postbank

# The End.

# Agenda

Postbank

# Appendix: List of SuSE-RPMs for Root-FS

../noarch/netcfg-9.0-5.noarch.rpm
../noarch/suse-build-key-1.0-460.noarch.rpm
aaa_base-9.0-6.i586.rpm
aaa_skel-2003.9.18-4.i586.rpm
acl-2.2.15-23.i586.rpm
ash-0.2-798.i586.rpm
at-3.1.8-782.i586.rpm
attr-2.4.8-23.i586.rpm
bash-2.05b-207.i586.rpm
bc-1.06-643.i586.rpm
bzip2-1.0.2-224.i586.rpm
coreutils-5.0-75.i586.rpm
cpio-2.5-209.i586.rpm
cpp-3.3.1-24.i586.rpm
cracklib-2.7-895.i586.rpm
cron-3.0.1-814.i586.rpm
curl-7.10.5-37.i586.rpm
cyrus-sasl-2.1.15-57.i586.rpm
db-4.1.25-70.i586.rpm
devs-9.0-4.i586.rpm
diffutils-2.8.1-203.i586.rpm
e2fsprogs-1.34-30.i586.rpm
ed-0.2-762.i586.rpm
ethtool-1.8-33.i586.rpm
fbset-2.1-680.i586.rpm
file-4.03-40.i586.rpm
filesystem-9.0-6.i586.rpm
fillup-1.42-9.i586.rpm
findutils-4.1.7-710.i586.rpm
gawk-3.1.3-53.i586.rpm
gdbm-1.8.3-119.i586.rpm
glibc-2.3.2-87.i586.rpm
glibc-locale-2.3.2-87.i586.rpm
gpg-1.2.2-80.i586.rpm
grep-2.5.1-294.i586.rpm
groff-1.17.2-685.i586.rpm
gzip-1.3.5-47.i586.rpm
heimdal-lib-0.6-67.i586.rpm
info-4.5-88.i586.rpm
insserv-1.00.1-15.i586.rpm
iproute2-2.4.7-655.i586.rpm
iptables-1.2.8-71.i586.rpm
iputils-ss021109-56.i586.rpm
isapnp-1.26-390.i586.rpm
kbd-1.08-35.i586.rpm
ldapcpplib-0.0.1-851.i586.rpm
less-381-28.i586.rpm
libacl-2.2.15-23.i586.rpm
libattr-2.4.8-23.i586.rpm
libgcc-3.3.1-24.i586.rpm
libstdc++-3.3.1-24.i586.rpm
libxcrypt-2.0-32.i586.rpm
libxml2-2.5.10-25.i586.rpm
liby2util-2.8.15-1.i586.rpm
logrotate-3.6.6-91.i586.rpm
lsof-4.68-33.i586.rpm
mailx-10.5-37.i586.rpm
man-2.4.1-60.i586.rpm
mktemp-1.5-633.i586.rpm
modutils-2.4.25-50.i586.rpm
ncurses-5.3-110.i586.rpm
net-tools-1.60-444.i586.rpm
netcat-1.10-764.i586.rpm
openldap2-client-2.1.22-65.i586.rpm
openssh-3.7.1p2-1.i586.rpm
openssl-0.9.7b-68.i586.rpm
pam-0.77-124.i586.rpm
pam-modules-9.0-5.i586.rpm
pcre-4.4-20.i586.rpm
perl-5.8.1-46.i586.rpm
permissions-2003.9.18-4.i586.rpm
popt-1.7-70.i586.rpm
portmap-5beta-617.i586.rpm
postfix-2.0.14-41.i586.rpm
procmail-3.15.1-479.i586.rpm
ps-2003.9.20-3.i586.rpm
readline-4.3-207.i586.rpm
recode-3.6-391.i586.rpm
rpm-4.1.1-71.i586.rpm
sash-3.6-105.i586.rpm
sed-4.0.6-69.i586.rpm
shadow-4.0.3-182.i586.rpm
src_vipa-1.0.1-153.i586.rpm
suse-release-9.0-6.i586.rpm
sysconfig-0.23.30-17.i586.rpm
syslogd-1.4.1-418.i586.rpm
sysvinit-2.82-362.i586.rpm
tar-1.13.25-199.i586.rpm
terminfo-5.3-110.i586.rpm
timezone-2.3.2-87.i586.rpm
utempter-0.5.2-287.i586.rpm
util-linux-2.11z-91.i586.rpm
vim-6.2-74.i586.rpm
zlib-1.1.4-225.i586.rpm

**94 RPMs from 1st DVD of SuSE 9.0 Professional**

# Appendix: Sample UML kernel config.

```
CONFIG_USERMODE=y
CONFIG_UID16=y
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_EXPERIMENTAL=y
CONFIG_MODE_SKAS=y
CONFIG_MODE_TT=y
CONFIG_NET=y
CONFIG_SYSVIPC=y
CONFIG_BSD_PROCESS_ACCT=y
CONFIG_SYSCTL=y
CONFIG_BINFMT_AOUT=y
CONFIG_BINFMT_ELF=y
CONFIG_BINFMT_MISC=y
CONFIG_HOSTFS=m
CONFIG_MCONSOLE=y
CONFIG_MAGIC_SYSRQ=y
CONFIG_NEST_LEVEL=0
CONFIG_KERNEL_HALF_GIGS=1
CONFIG_KERNEL_STACK_ORDER=2
CONFIG_MODULES=y
CONFIG_KMOD=y
CONFIG_STDIO_CONSOLE=y
CONFIG_SSL=y
CONFIG_FD_CHAN=y
CONFIG_NULL_CHAN=y
CONFIG_PORT_CHAN=y
CONFIG_PTY_CHAN=y
CONFIG_TTY_CHAN=y
CONFIG_XTERM_CHAN=y
CONFIG_CON_ZERO_CHAN="fd:0,fd:1"
CONFIG_CON_CHAN="xterm"
CONFIG_SSL_CHAN="pty"
CONFIG_UNIX98_PTYS=y
CONFIG_UNIX98_PTY_COUNT=256
CONFIG_WATCHDOG=y
CONFIG_SOFT_WATCHDOG=m
CONFIG_UML_WATCHDOG=m
CONFIG_UML_SOUND=y
CONFIG_SOUND=y
CONFIG_HOSTAUDIO=y
CONFIG_BLK_DEV_UBD=y
CONFIG_COW=y
CONFIG_COW_COMMON=y
CONFIG_BLK_DEV_LOOP=m
CONFIG_BLK_DEV_NBD=m
CONFIG_NETDEVICES=y
CONFIG_UML_NET=y
CONFIG_UML_NET_TUNTAP=y
CONFIG_UML_NET_DAEMON=y
CONFIG_UML_NET_MCAST=y
CONFIG_DUMMY=m
CONFIG_TUN=m
CONFIG_PACKET=m
CONFIG_PACKET_MMAP=y
CONFIG_NETFILTER=y
CONFIG_FILTER=y
CONFIG_UNIX=y
CONFIG_INET=y
CONFIG_IP_MULTICAST=y
CONFIG_IP_ADVANCED_ROUTER=y
CONFIG_IP_ROUTE_MULTIPATH=y
CONFIG_IP_NF_CONNTRACK=m
CONFIG_IP_NF_FTP=m
CONFIG_IP_NF_TFTP=m
CONFIG_IP_NF_QUEUE=m
CONFIG_IP_NF_IPTABLES=m
CONFIG_IP_NF_MATCH_LIMIT=m
CONFIG_IP_NF_MATCH_MAC=m
CONFIG_IP_NF_MATCH_PKTTYPE=m
CONFIG_IP_NF_MATCH_MARK=m
CONFIG_IP_NF_MATCH_MULTIPORT=m
CONFIG_IP_NF_MATCH_TOS=m
CONFIG_IP_NF_MATCH_RECENT=m
CONFIG_IP_NF_MATCH_ECN=m
CONFIG_IP_NF_MATCH_DSCP=m
CONFIG_IP_NF_MATCH_AH_ESP=m
CONFIG_IP_NF_MATCH_LENGTH=m
CONFIG_IP_NF_MATCH_TTL=m
CONFIG_IP_NF_MATCH_TCPMSS=m
CONFIG_IP_NF_MATCH_HELPER=m
CONFIG_IP_NF_MATCH_STATE=m
CONFIG_IP_NF_MATCH_CONNTRACK=m
CONFIG_IP_NF_MATCH_UNCLEAN=m
CONFIG_IP_NF_MATCH_OWNER=m
CONFIG_IP_NF_FILTER=m
CONFIG_IP_NF_TARGET_REJECT=m
CONFIG_IP_NF_TARGET_MIRROR=m
CONFIG_IP_NF_NAT=m
CONFIG_IP_NF_NAT_NEEDED=y
CONFIG_IP_NF_TARGET_MASQUERADE=m
CONFIG_IP_NF_TARGET_REDIRECT=m
CONFIG_IP_NF_NAT_SNMP_BASIC=m
CONFIG_IP_NF_NAT_FTP=m
CONFIG_IP_NF_NAT_TFTP=m
CONFIG_IP_NF_MANGLE=m
CONFIG_IP_NF_TARGET_TOS=m
CONFIG_IP_NF_TARGET_ECN=m
CONFIG_IP_NF_TARGET_DSCP=m
CONFIG_IP_NF_TARGET_MARK=m
CONFIG_IP_NF_TARGET_LOG=m
CONFIG_IP_NF_TARGET_ULOG=m
CONFIG_IP_NF_TARGET_TCPMSS=m
CONFIG_IP_NF_ARPTABLES=m
CONFIG_IP_NF_ARPFILTER=m
CONFIG_IP_NF_ARP_MANGLE=m
CONFIG_IP_NF_COMPAT_IPCHAINS=m
CONFIG_IP_NF_NAT_NEEDED=y
CONFIG_IP_NF_COMPAT_IPFWADM=m
CONFIG_IP_NF_NAT_NEEDED=y
CONFIG_NET_SCHED=y
CONFIG_NET_SCH_CBQ=m
CONFIG_NET_SCH_HTB=m
CONFIG_NET_SCH_CSZ=m
CONFIG_NET_SCH_PRIO=m
CONFIG_NET_SCH_RED=m
CONFIG_NET_SCH_SFQ=m
CONFIG_NET_SCH_TEQL=m
CONFIG_NET_SCH_TBF=m
CONFIG_NET_SCH_GRED=m
CONFIG_NET_SCH_DSMARK=m
```

1/2

# Appendix: Sample UML kernel config.

```
CONFIG_NET_SCH_INGRESS=m
CONFIG_NET_QOS=y
CONFIG_NET_ESTIMATOR=y
CONFIG_NET_CLS=y
CONFIG_NET_CLS_TCINDEX=m
CONFIG_NET_CLS_ROUTE4=m
CONFIG_NET_CLS_ROUTE=y
CONFIG_NET_CLS_FW=m
CONFIG_NET_CLS_U32=m
CONFIG_NET_CLS_RSVP=m
CONFIG_NET_CLS_RSVP6=m
CONFIG_NET_CLS_POLICE=y
CONFIG_QUOTA=y
CONFIG_REISERFS_FS=m
CONFIG_REISERFS_PROC_INFO=y
CONFIG_EXT3_FS=y
CONFIG_JBD=y
CONFIG_FAT_FS=m
CONFIG_MSDOS_FS=m
CONFIG_UMSDOS_FS=m
CONFIG_VFAT_FS=m
CONFIG_TMPFS=y
CONFIG_RAMFS=y
CONFIG_ISO9660_FS=m
CONFIG_JOLIET=y
CONFIG_ZISOFS=y
CONFIG_MINIX_FS=m
CONFIG_PROC_FS=y
CONFIG_DEVFS_FS=y
CONFIG_DEVFS_MOUNT=y
CONFIG_DEVPTS_FS=y
CONFIG_EXT2_FS=y
CONFIG_UDF_FS=m
CONFIG_UFS_FS=m
CONFIG_NFS_FS=m
CONFIG_NFS_V3=y
CONFIG_NFSD=m
CONFIG_NFSD_V3=y
CONFIG_SUNRPC=m
CONFIG_LOCKD=m
CONFIG_LOCKD_V4=y
CONFIG_SMB_FS=m
CONFIG_SMB_NLS_DEFAULT=y
CONFIG_SMB_NLS_REMOTE="cp437"
CONFIG_ZISOFS_FS=m
CONFIG_PARTITION_ADVANCED=y
CONFIG_MSDOS_PARTITION=y
CONFIG_SMB_NLS=y
CONFIG_NLS=y
CONFIG_NLS_DEFAULT="iso8859-1"
CONFIG_NLS_CODEPAGE_437=m
CONFIG_NLS_CODEPAGE_737=m
CONFIG_NLS_CODEPAGE_775=m
CONFIG_NLS_CODEPAGE_850=m
CONFIG_NLS_CODEPAGE_852=m
CONFIG_NLS_CODEPAGE_855=m
CONFIG_NLS_CODEPAGE_857=m
CONFIG_NLS_CODEPAGE_860=m
CONFIG_NLS_CODEPAGE_861=m
CONFIG_NLS_CODEPAGE_862=m
CONFIG_NLS_CODEPAGE_863=m
CONFIG_NLS_CODEPAGE_864=m
CONFIG_NLS_CODEPAGE_865=m
CONFIG_NLS_CODEPAGE_866=m
CONFIG_NLS_CODEPAGE_869=m
CONFIG_NLS_CODEPAGE_1250=m
CONFIG_NLS_CODEPAGE_1251=m
CONFIG_NLS_ISO8859_1=m
CONFIG_NLS_ISO8859_2=m
CONFIG_NLS_ISO8859_3=m
CONFIG_NLS_ISO8859_4=m
CONFIG_NLS_ISO8859_5=m
CONFIG_NLS_ISO8859_6=m
CONFIG_NLS_ISO8859_7=m
CONFIG_NLS_ISO8859_9=m
CONFIG_NLS_ISO8859_13=m
CONFIG_NLS_ISO8859_14=m
CONFIG_NLS_ISO8859_15=m
CONFIG_NLS_KOI8_R=m
CONFIG_NLS_KOI8_U=m
CONFIG_NLS_UTF8=m
CONFIG_MD=y
CONFIG_BLK_DEV_MD=m
CONFIG_MD_LINEAR=m
CONFIG_MD_RAID0=m
CONFIG_MD_RAID1=m
CONFIG_MD_RAID5=m
CONFIG_MD_MULTIPATH=m
CONFIG_BLK_DEV_LVM=m
CONFIG_ZLIB_INFLATE=m
CONFIG_ZLIB_DEFLATE=y
CONFIG_DEBUGSYM=y
CONFIG_PT_PROXY=y
```

2/2