



Department of Telecommunications

BSD Network Stack Virtualization

Marko Zec

zec@tel.fer.hr

University of Zagreb, Faculty of Electrical
Engineering and Computing

BSDCon Europe, Amsterdam, November 2002.

Session contents:



Department of Telecommunications

- ◆ Introduction
- ◆ Design
- ◆ Implementation
- ◆ Performance implications
- ◆ Application scenarios
- ◆ Future work
- ◆ Discussion / questions

- ◆ Traditional OS architecture
 - General-purpose operating systems (OS) provide support for a **single** instance of network stack or protocol family within the kernel

- ◆ New concept
 - Network stack **virtualization** – a set of kernel code modifications and extensions which allow simultaneous support for **multiple** independent network stack instances within a single kernel

- ◆ Research application:
 - Network simulation
 - *Berkeley NS, OPNet modeler (“offline” simulators)*
 - *ENTRAPID, Alpine (network stack implementation in userland)*
 - *Harvard network simulator (address remapping middleware)*

- ◆ Production applications:
 - Virtual hosting
 - *IBM S/390, VMware, BSD jail*
 - VPN provisioning
 - *Cisco VRF, Linux VRF, FreeBSD 4.4 VPN patch...*

Introduction: *design objectives*



Department of Telecommunications

- ◆ Take a “general-purpose” approach
 - The network stack extensions must fit equally well in diverse application scenarios

- ◆ Compatibility with existing userland applications
 - Preserve both the application programming and binary interfaces (API / ABI)

- ◆ Avoid significant performance degradations
 - The users / applications shouldn't be able to notice the difference between the standard and modified network stack

- ◆ Virtualize the entire network stack, not just the selected portions
 - Network interfaces
 - Packet queues
 - Forwarding path, routing tables
 - Socket interfaces, protocol control blocks, hash tables
 - Statistics / counters
 - Sysctl tunable variables
 - Advanced features (firewall, traffic shaper...)
 - Support for multiple protocol families (not only IPv4)

Design: *the concepts (continued)*



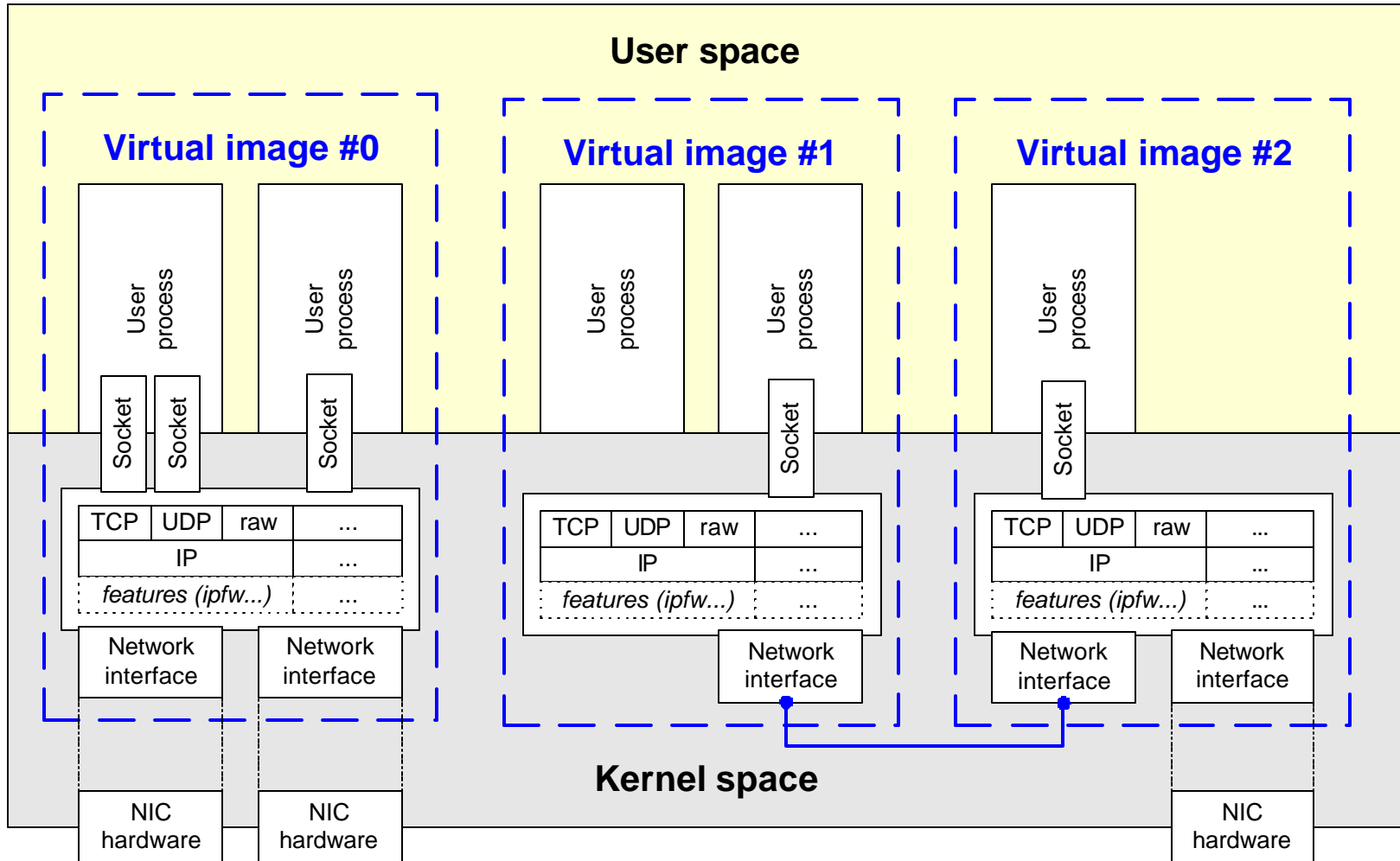
Department of Telecommunications

- ◆ Implement the functional extensions *entirely* within the kernel
 - Performance
 - Resource protection

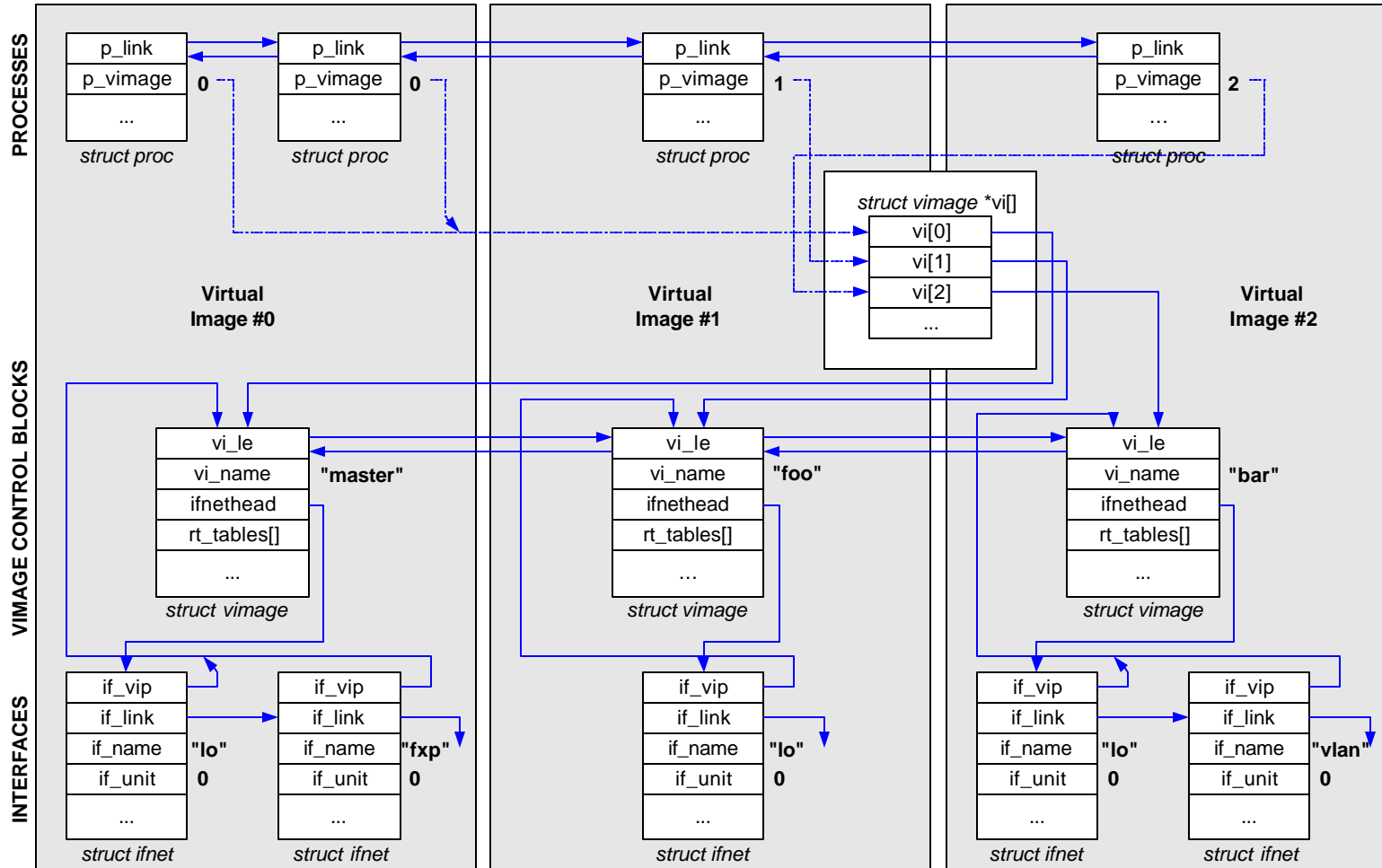
- ◆ *Kernel* support for transparent compatibility with the userland binaries (API/ABI)

- ◆ A stable development platform
 - FreeBSD 4.x-RELEASE branch selected

Design: *virtual images*



Implementation: *kernel data structures*



Implementation: *struct vimage*



```
struct vimage {
    LIST_ENTRY(vimage) vi_le;          /* linked list of all vimages */
/* sys/net */
    struct radix_node_head *rt_tables[AF_MAX+1]; /* from net/route.c */
    struct ifnethead ifnet; /* from net/if.c */
    struct ifaddr **ifnet_addrs; /* from net/if.c */
    struct ifnet **ifindex2ifnet; /* from net/if.c */
    struct rawcb_list_head rawcb_list; /* from net/raw_cb.c */
    struct ifnet loif; /* from net/if_loop.c */
    struct ifqueue ipintrq;
/* sys/netinet */
    struct route ipforward_rt; /* from netinet/ip_input.c */
    struct in_ifaddrhead in_ifaddrhead; /* from netinet/ip_input.c */
    int ipforwarding;
    struct inpcbhead tcb; /* from netinet/tcp_input.c */
    struct inpcbinfo tcbinfo; /* from netinet/tcp_input.c */
    struct tcp_syncache tcp_syncache; /* from netinet/tcp_syncache.c */
    struct inpcbhead udb; /* from netinet/udp_usrreq.c */
    struct inpcbinfo udbinfo; /* from netinet/udp_usrreq.c */
    struct ipfw_dyn_rule **ipfw_dyn_v; /* from netinet/ip_fw.c */
/* sys/netipx */
    struct ipx_ifaddr *ipx_ifaddr; /* from netipx/ipx.c */
    . . .
}
```

Implementation: *handling network traffic*

- ◆ Typical event types
 - Reception of incoming network frames
 - Socket operations / data transmission
 - Timeout operations
- ◆ Handling incoming network frames
 - For received frames, the *mbuf* header contains the pointer to ingress network interface (struct *ifnet*)

```
struct vimage *vip = m->m_pkthdr.rcvif->if_vp;
```

Implementation: *handling network traffic*



Department of Telecommunications

- ◆ netisr processing
 - Processing packets from inbound queues
 - example: `ipintr()`

```
struct mbuf *m; struct vimage *vip;
LIST_FOREACH(vip, &vi_head, vi_le)
    while(1) {
        IF_DEQUEUE(&vip->ipintrq, m);
        if (m == 0)
            break;
        ip_input(m);
    }
```

Implementation: *handling network traffic*

- ◆ requests from userland processes

```
xxx_connect(foo, bar, struct proc *p) {  
    struct vimage *vip = vi[p->p_vimage];
```

- ◆ periodic/timeout processing
 - slowtimo, fasttimo handlers modified to traverse all virtual images, similar to netisr processing
- ◆ userland process grouping (hiding)
 - jail framework reuse (PRISON_CHECK macro extension in kern/proc.h)

Implementation: *creation of virtual images*

- ◆ System startup / autoconfiguration
 - Only virtual image #0 (*master*) exists by default
 - Dynamic creation of additional virtual images

- ◆ Modifications in `domain_attach` handlers
 - standard stack: `pr_init(void)`
 - virtualized stack: `pr_init(struct vimage *)`

- ◆ Similar modifications in `mod_event` handlers
 - `ipfw`, `dummynet`, `ng_ether`...

Implementation: *userland binary compatibility*



Department of Telecommunications

- ◆ `kvm_read` support for virtualized symbols
 - extensions to `kldsym()` in `kern/kern_linker.c`
 - if the symbol requested cannot be resolved, try to find it in the appropriate *struct vimage*
- ◆ `sysctl` framework virtualization
 - New macros/hooks for manipulating virtualized symbols – examples:

```
int sysctl_handle_v_int()  
SYSCTL_V_INT
```

- ◆ CPU time and load accounting virtualization
 - system load
 - process priority calculation
 - idle / interrupt time accounting

- ◆ CPU usage limiting
 - run queues – skipping active processes
 - time quantum scaling
 - returning to `cpu_idle`

Implementation: *management*



Department of Telecommunications

```
vmbsd# vimage -c bsdcon #create a new virtual image
```

```
vmbsd# vimage -l #list the current virtual images
```

```
"master":
```

```
30 processes, load averages: 0.15, 0.03, 0.01
```

```
CPU usage: 1.81% (0.00% user, 0.00% nice, 1.81% system)
```

```
Nice level: 0, no CPU limit, no process limit,
```

```
child limit: 7
```

```
2 network interfaces, 1 child vimages
```

```
"bsdcon":
```

```
0 processes, load averages: 0.00, 0.00, 0.00
```

```
CPU usage: 0.00% (0.00% user, 0.00% nice, 0.00% system)
```

```
Nice level: 0, no CPU limit, no process limit
```

```
1 network interfaces, parent vimage: "master"
```

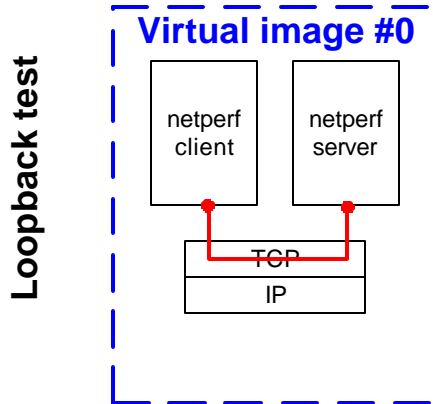
Implementation: *management*



Department of Telecommunications

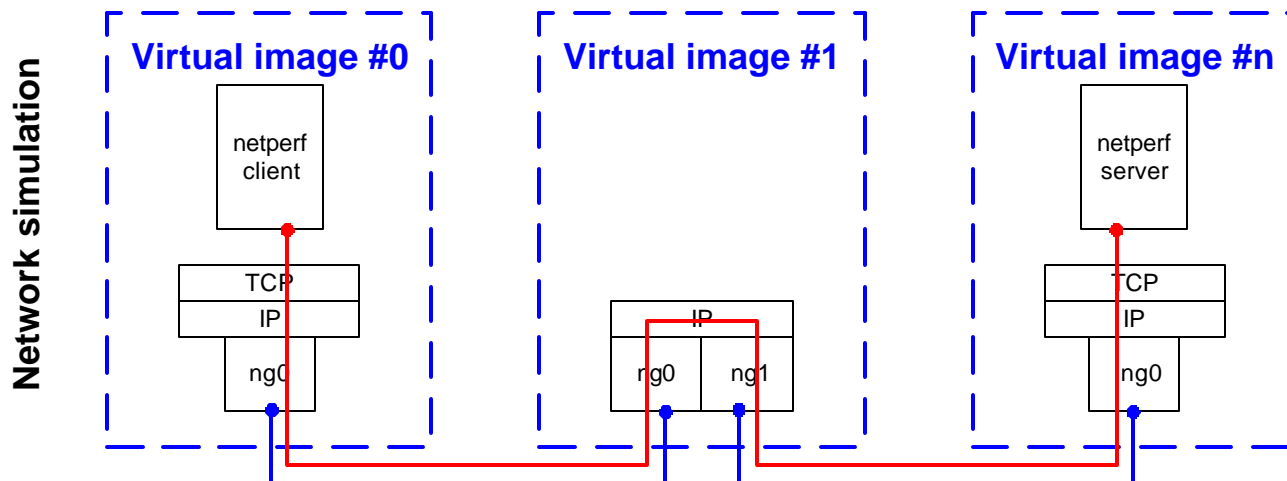
```
vmbsd# ifconfig #we are still in the "master" vimage
lnc0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> mtu 1500
    ether 00:50:56:40:00:47
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
    inet 127.0.0.1 netmask 0xff000000
vmbsd# vimage bsdcon ifconfig #exec ifconfig in "bsdcon"
lo0: flags=8008<LOOPBACK,MULTICAST> mtu 16384
vmbsd# vimage -i bsdcon lnc0 #move lnc0 to "bsdcon"
vmbsd# vimage bsdcon #start a new shell in "bsdcon"
Switched to vimage bsdcon
# ifconfig
lnc0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> mtu 1500
    ether 00:50:56:40:00:47
lo0: flags=8008<LOOPBACK,MULTICAST> mtu 16384
```

Performance: *measurement scenarios*

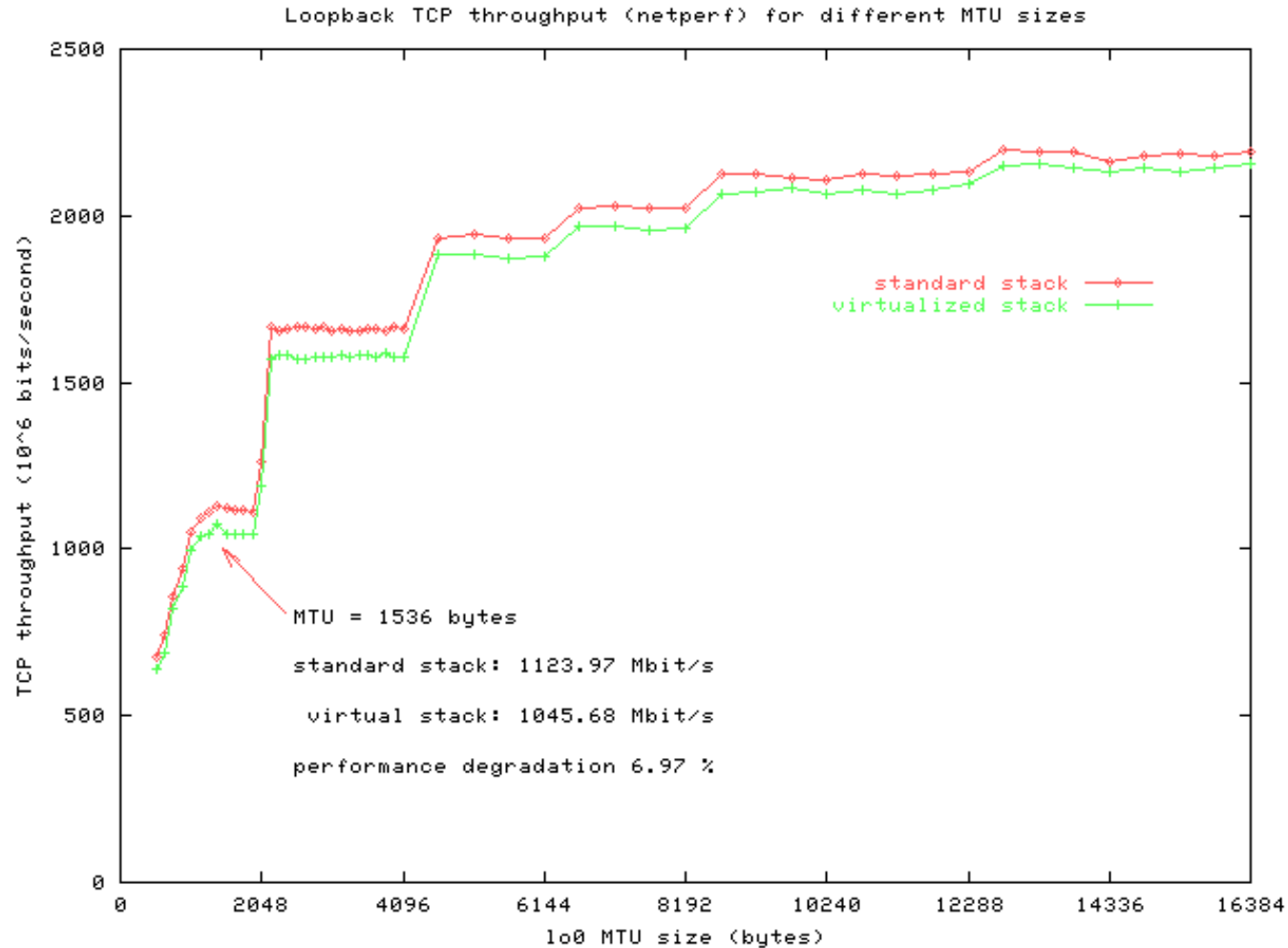


Referent machine:

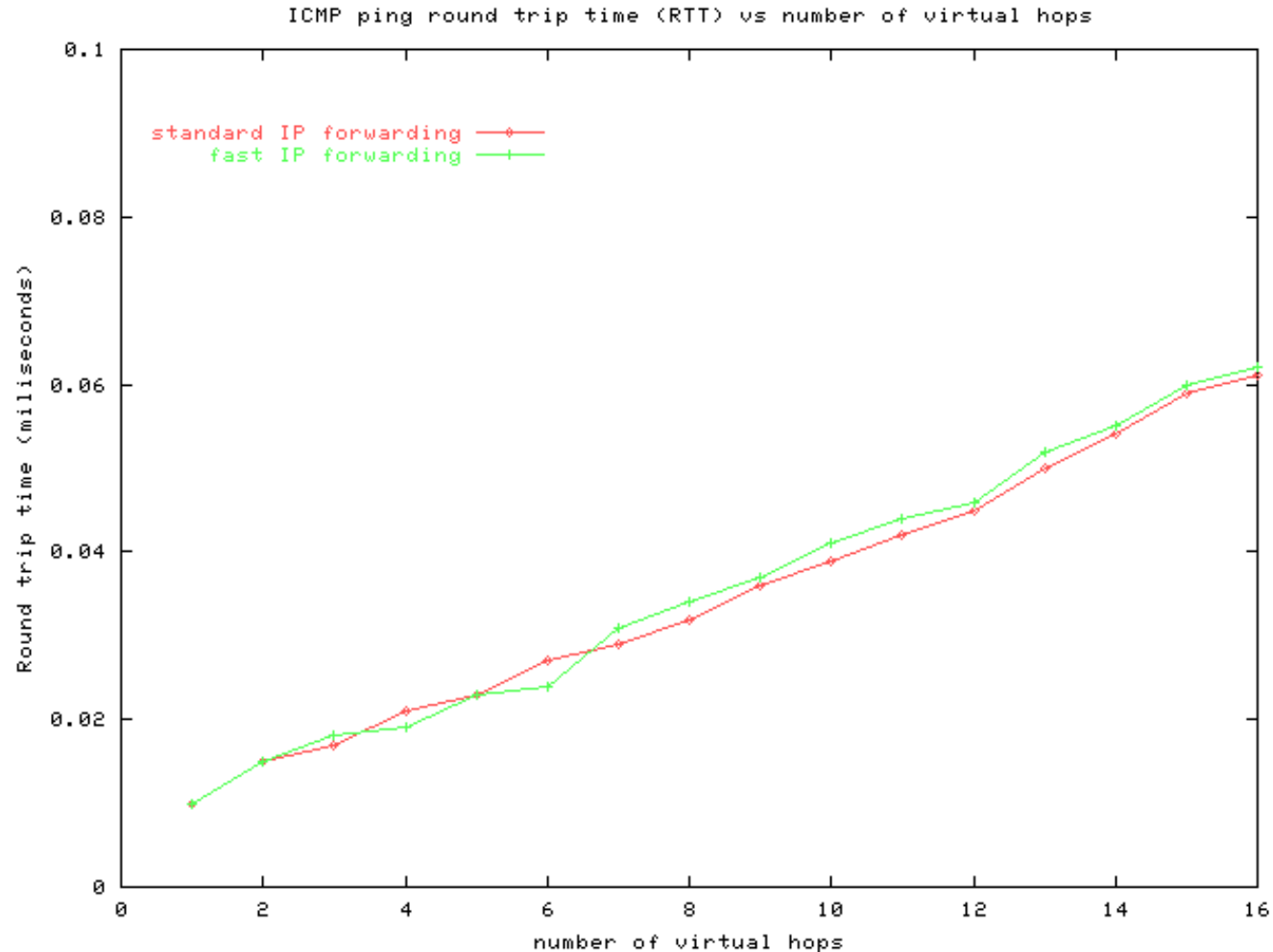
AMD Athlon @ 1200 MHz, 100 MHz FSB
256 MByte SDRAM
FreeBSD 4.7-RELEASE



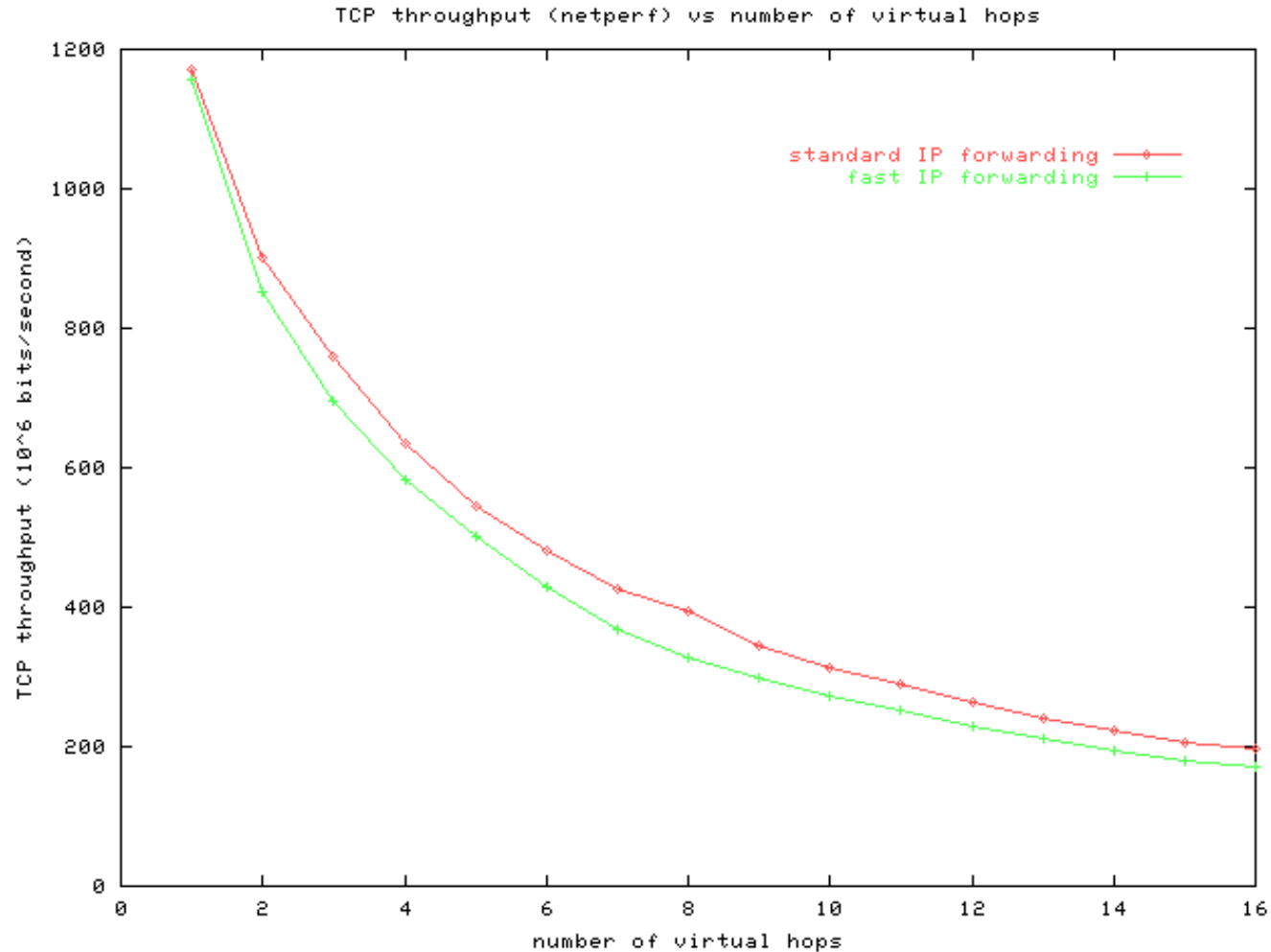
Performance: *loopback TCP throughput*



Performance: *latency (ICMP ping)*



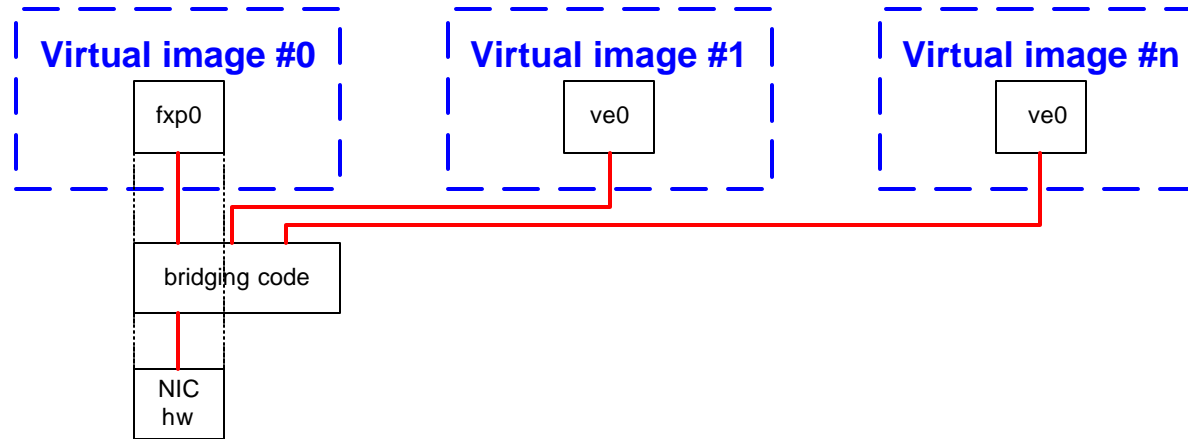
Performance: *TCP over multiple virtual hops*



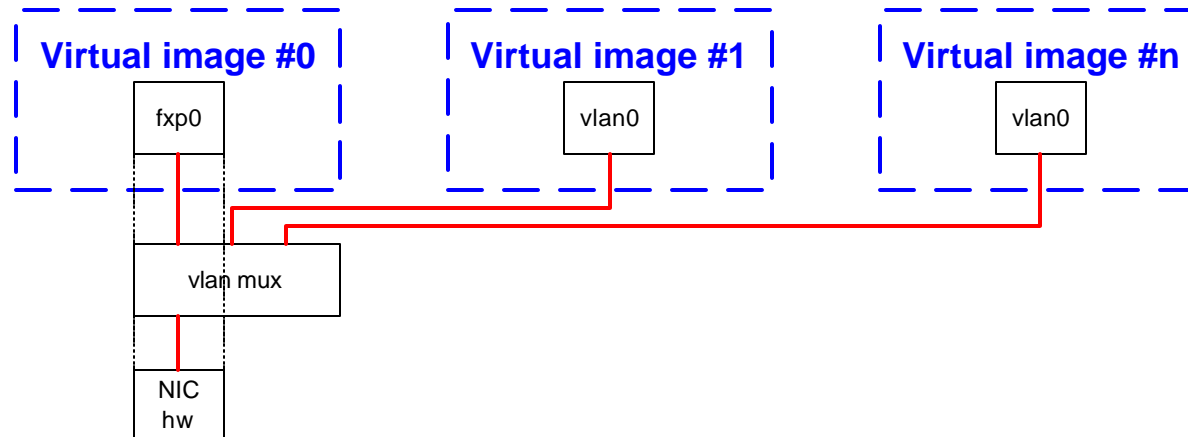
Implementation: *application scenarios*



Virtual hosting



VPN



Future work



Department of Telecommunications

- Implement removal of virtual images (`domain_detach?`)
- Proper detection of domain attach failures, with controlled rollback domain detach
- Tunnel interfaces (*gif*, *tun*, *faith...*)
- Resource protection
 - Check for correct reuse of *jail* framework
 - Mbufs, userland memory, swap, I/O...
- Migration to FreeBSD 5.0
 - Reserve the fields in *struct proc* & *ifnet* for future use NOW!
- MP adjustments / testing
- Virtualization of protocols other than IPv4
- Porting to other BSD platforms

- ◆ Experimental implementation – scope of work
 - ~190 virtualized symbols (in *struct vimage*)
 - ~5200 lines of new or modified code
 - 165 modified files in /sys tree, including new files

- ◆ Patches against FreeBSD 4.7-RELEASE available at <http://www.tel.fer.hr/zec/>

- ◆ Discussion / questions ?