Network stack virtualization for FreeBSD 7.0

Marko Zec

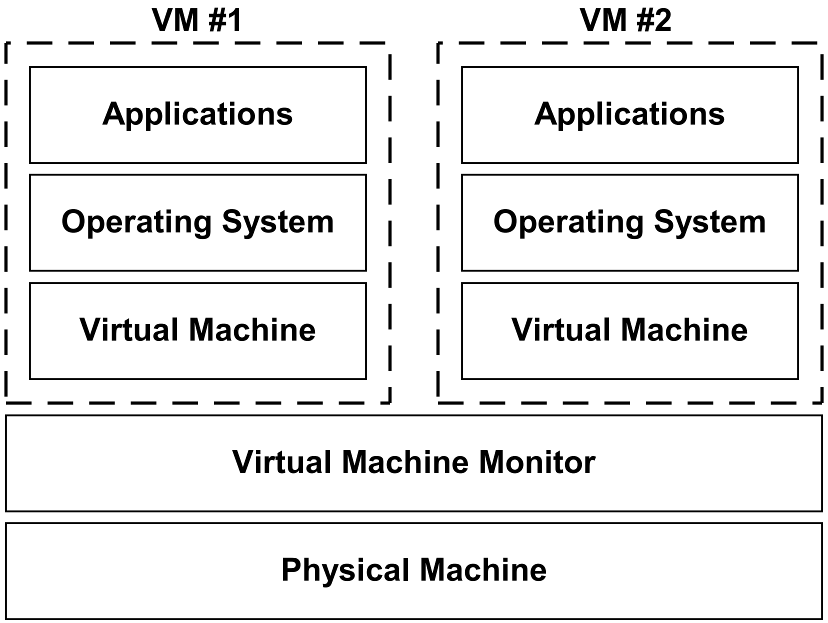
zec@fer.hr

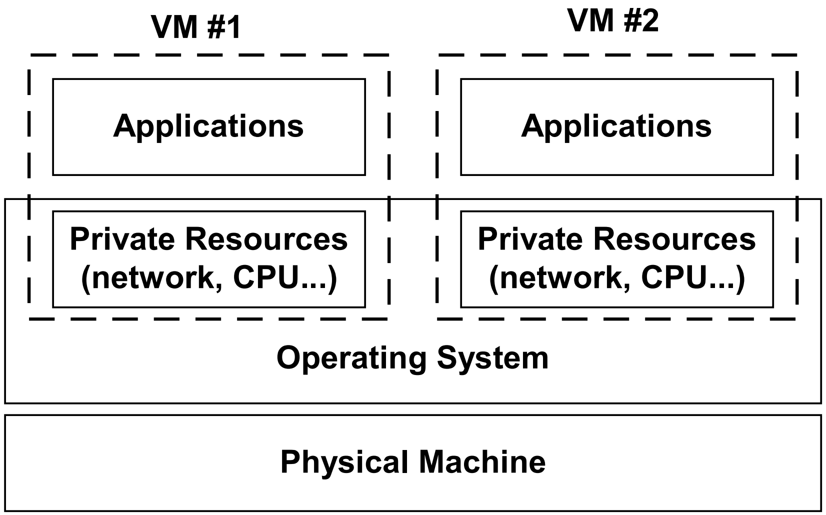
University of Zagreb

Talk outline

- Network stack virtualization – what, why, and how?
- Who needs this?
- Implementation: FreeBSD 4.x vs. 7.0
- Generalizing OS-level resource virtualization?

Server virtualization: two sides of the spectrum




VM #1
VM #2
Applications
Applications
Private Resources (network, CPU...)
Private Resources (network, CPU...)
Operating System
Physical Machine

Strong isolation model

Independent OS instances

VM migration possible

Efficient resource utilization

No extra I/O overhead

Scaling

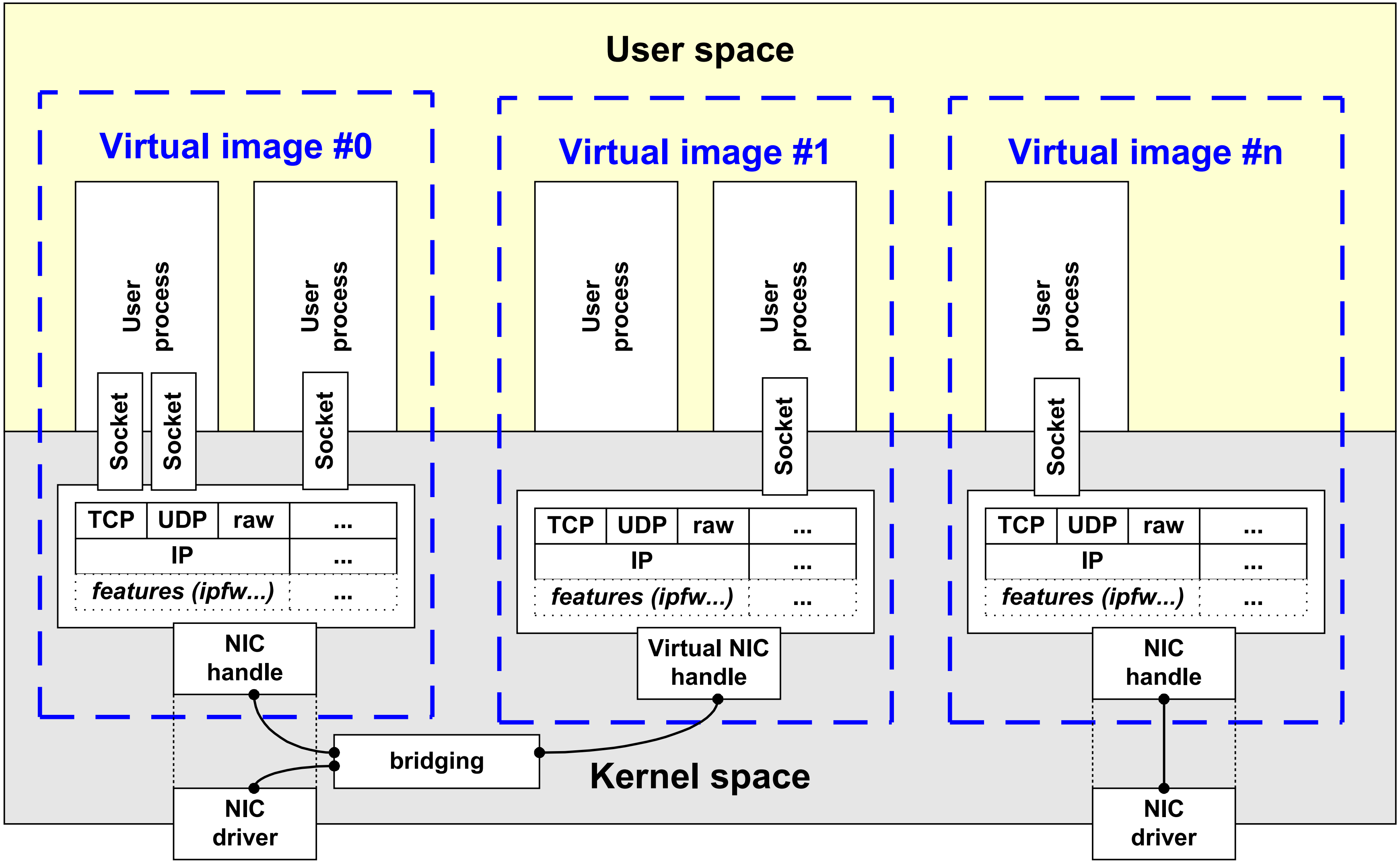
Motivation: the idea

- Traditional OS architecture
  - Support for only a single instance of network stack or protocol family within the kernel
  - *Jails*: first successful pseudo-virtualization framework
- Network stack virtualization (or *cloning)*
  - Multiple independent network stack *state* instances within a single kernel
  - Existing networking code paths and algorithms remain the same, but must be taught on how to operate on virtualized symbols / state

Applications: who needs this?

- Virtual hosting
  - Think of extending FreeBSD `jail` with its own independent network stack instance: multiple interfaces and IP addresses, private routing table, IPFW / PF, dummynet, BPF, raw sockets etc. etc.
  - Anecdotal evidence: FreeBSD 4.11 based version in production use by some US ISPs
- VPN provisioning and monitoring
  - Support for overlapping IP addressing schemes
- Network simulation / emulation
  - Each network stack instance == an independent virtual node or router -> http://www.imunes.net/

The basic idea: replicate global networking state



Implementation concepts: long time ago...

- Patches against FreeBSD 4.7 .. 4.11 kernels
  - Obsolete platform today
- `struct vnet`
  - One huge structure / container; each network stack instance operates on its private copy
  - Contains ifnet lists, IPv4 / IPv6 / firewall state etc.
- Sockets
  - Each socket is assigned to a network stack instance during creation time
  - Cannot move / change until socket closed

Implementation concepts: how it was done (cont'd)

- Network interfaces
  - Each interface can belong to only one network stack instance at a time
  - Demultiplexing of incoming traffic based on on the new `if_vnet` tag in `struct ifnet`
  - Network communication between stack instances only through explicit links: **netgraph**
- User processes
  - Bound to only one stack at a time, can reassociate
  - Jail–style separation (reused existing jail code)

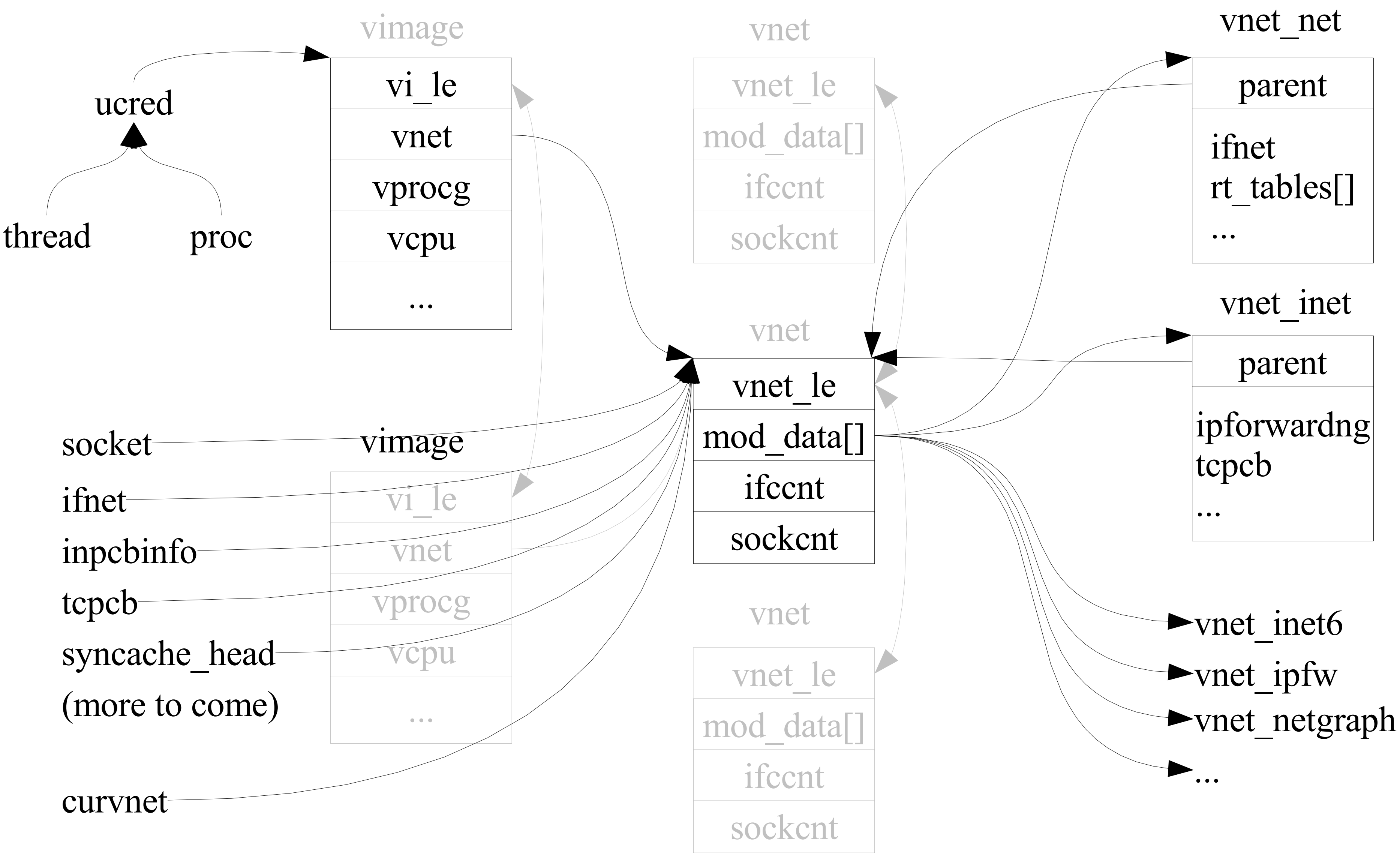
Implementation concepts: API / ABI compatibility

- Userland to kernel: both API and ABI 100% preserved
  - Support for accessing the virtualized symbols added to the `kldsym` interface (needed for `netstat, systat, top` and similar utilities)
  - Similar extensions added to the `sysctl` interface
- Within the kernel: API is *NOT* preserved
  - Many networking functions extended with an additional argument: `struct vnet *`
  - Generally, no changes at device driver layer

(Re)implementation: 7.0

- Goals:
- Conditional compilation
- Better support for kernel loadable modules
- Scope of changes is huge: reduce code churn
- SMP must work
- Otherwise, no chances for including the changes into main FreeBSD tree

Replicate global networking state: how?



ucred

thread

proc

vimage

vi_le
vnet
vprocg
vcpu
...

vnet

vnet_le
mod_data[]
ifccnt
sockcnt

vnet_net

parent
ifnet
rt_tables[]
...

vnet_inet

parent
ipforwardng
tcpcb
...

socket
ifnet
inpcbinfo
tcpcb
syncache_head
(more to come)

curvnet

vimage

vi_le
vnet
vprocg
vcpu
...

vnet

vnet_le
mod_data[]
ifccnt
sockcnt

vnet

vnet_le
mod_data[]
ifccnt
sockcnt

vnet_inet6
vnet_ipfw
vnet_netgraph
...

vnet modules: registration / deregistration

```
static struct vnet_symmap vnet_net_symmap[] = {
        VNET_SYMMAP(net, ifnet),
        VNET_SYMMAP(net, rt_tables),
        ...
        VNET_SYMMAP_END
};

static struct vnet_modinfo vnet_net_modinfo = {
        .id             = VNET_MOD_NET,
        .flags          = VNET_MFLAG_ORDER_1ST,
        .name           = "net",
        .symmap         = vnet_net_symmap,
        .i_attach       = vnet_net_iattach,
        .i_detach       = vnet_net_idetach
};

if_init(void *dummy __unused)
{
#ifdef VIMAGE
        vnet_mod_register(&vnet_net_modinfo);
#else
        vnet_net_iattach();
#endif
        ...
```

Conditional compilation: `option VIMAGE`

- Dereference virtualized symbols: how?
  - Use macros for this. Example:
    - `if_addrhead` becomes **`V_if_addrhead`**
  - Standard kernel:
    - `V_if_addrhead` expands back to `if_addrhead`
  - Virtualized kernel:
    - `V_if_addrhead` expands to `vnet_net->_if_addrhead`
  - Sysctl and kldsym interfaces extended to support access to virtualized symbols

Reducing code churn

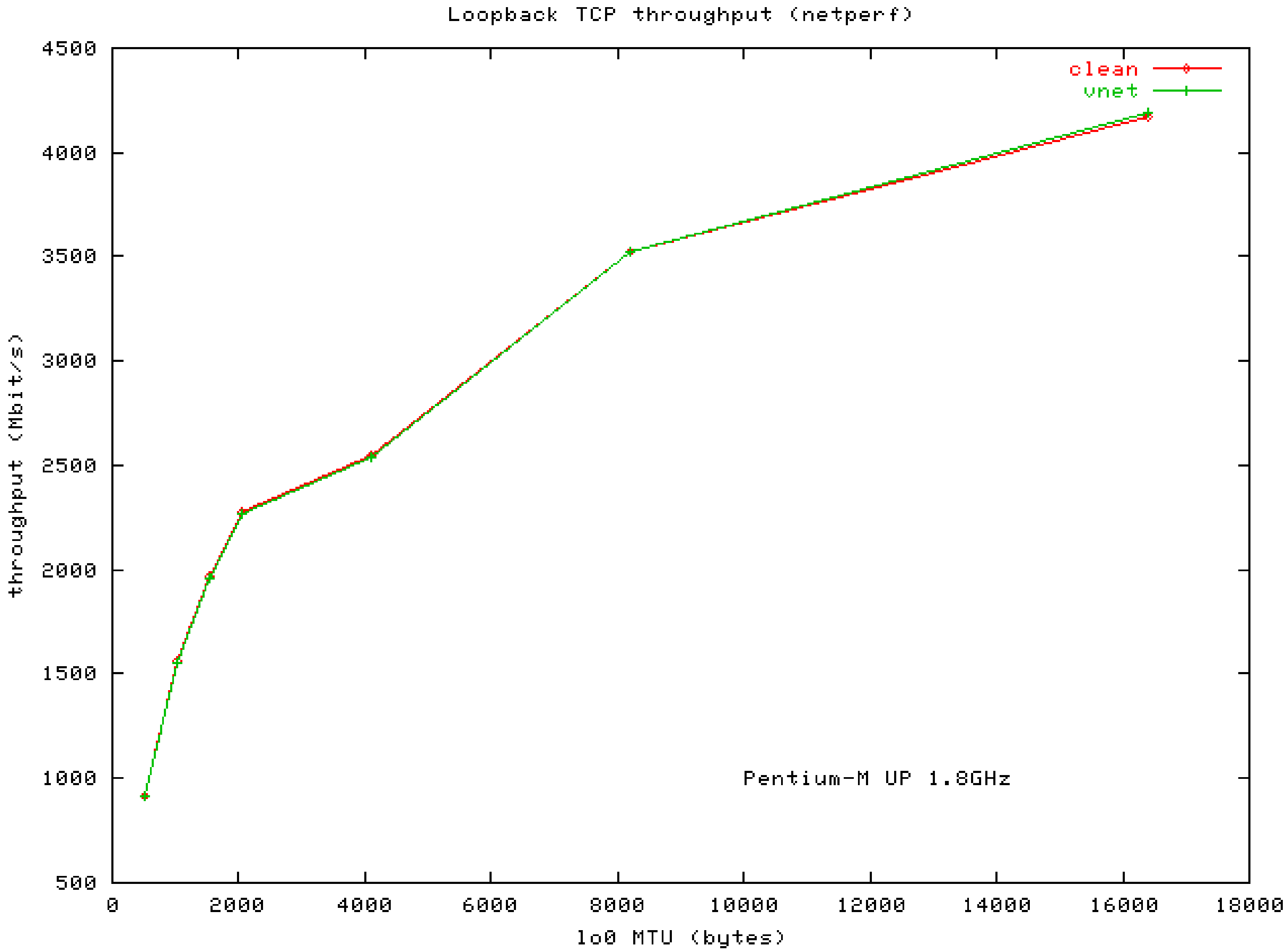
- Implicitly pass the `vnet` context to operate on:
  - Thread-local `curvnet` variable

```
void if_attach(struct ifnet *ifp)
{
        INIT_VNET_NET(curvnet);
        ...
}
```

`INIT_VNET_NET(x)` (x is a `struct vnet *`) expands to

```
struct vnet_net *vnet_net = x->mod_data[VNET_MOD_NET];
```

Performance: loopback TCP throughput



Generalizing OS-level virtualization

- Management concepts / API
  - Top-level resource container `struct vimage`
  - Contains freely combinable subsystem-specific state
    - `vnet, vcpu, vprocg, vfs...`
  - Single process with sockets in multiple stacks
    - Extend socket interface -> multi-table routing daemons
  - Hierarchy of vimages – follow UNIX process model?
  - Permissions, restrictions, inheritance...
  - How to best integrate those new concepts / features with the rest of the system?

Project status

- Supported by NLNet and FreeBSD foundation
  - Started in August 2006, should have already finished...
- In sync with -CURRENT: p4 projects/vimage
  - Snap-in replacement kernel – no userspace changes!
  - http://imunes.tel.fer.hr/virtnet/ : CVSup
- Reasonably stable already
  - Lots to be done: locking, management API & housekeeping
- Most important networking subsystems virtualized:
  - IPv4, IPv6, NFS, IPFW / PF firewalls, BPF, raw / routing sockets...
- Outside the tree until 7.0-RELEASE, merging in 8.0?

To conclude...

- Do we need all this?
  - the community has to provide that answer.
- If yes, what's next to virtualize?
  - CPU time (scheduler)
  - Filesystems (ZFS?) / disk I/O bandwidth
  - Memory
  - ...
- We need a generalized OS-level virtualization model

http://imunes.tel.fer.hr/virtnet/