

Network emulation using the virtualized network stack in FreeBSD

Marko Zec

University of Zagreb

zec@freebsd.org

Krakow, July 3rd 2010.

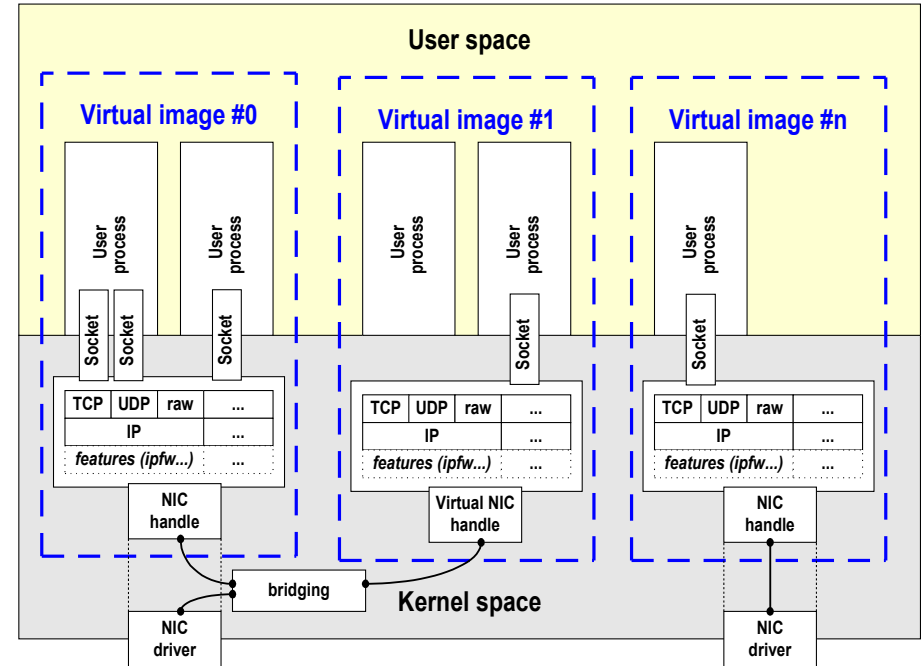
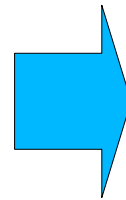
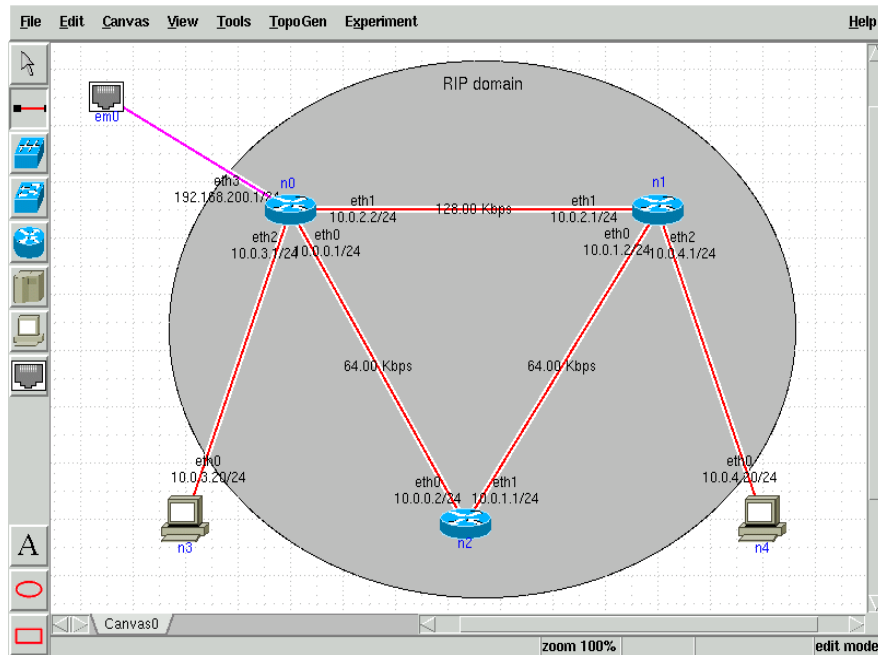
Talk outline

- Constructing arbitrary complex virtual network topologies
- Using lightweight virtual nodes: *vnet jails*
- Connected via virtual links: *netgraph*
- Running on a virtualized FreeBSD kernel

Virtualization: the art of illusion

- Provide *illusion* of exclusive use of shared hardware resources
- XEN, VMware, z/VM: multiple independent OS instances running on virtualized hardware
- FreeBSD jails, Solaris zones, Linux OpenVZ: single OS instance, provides multiple isolated hosting environments ← *in our focus today*
- Similar goals (server consolidation, application isolation, delegation of authority) with different tradeoffs

Network topology emulation demo



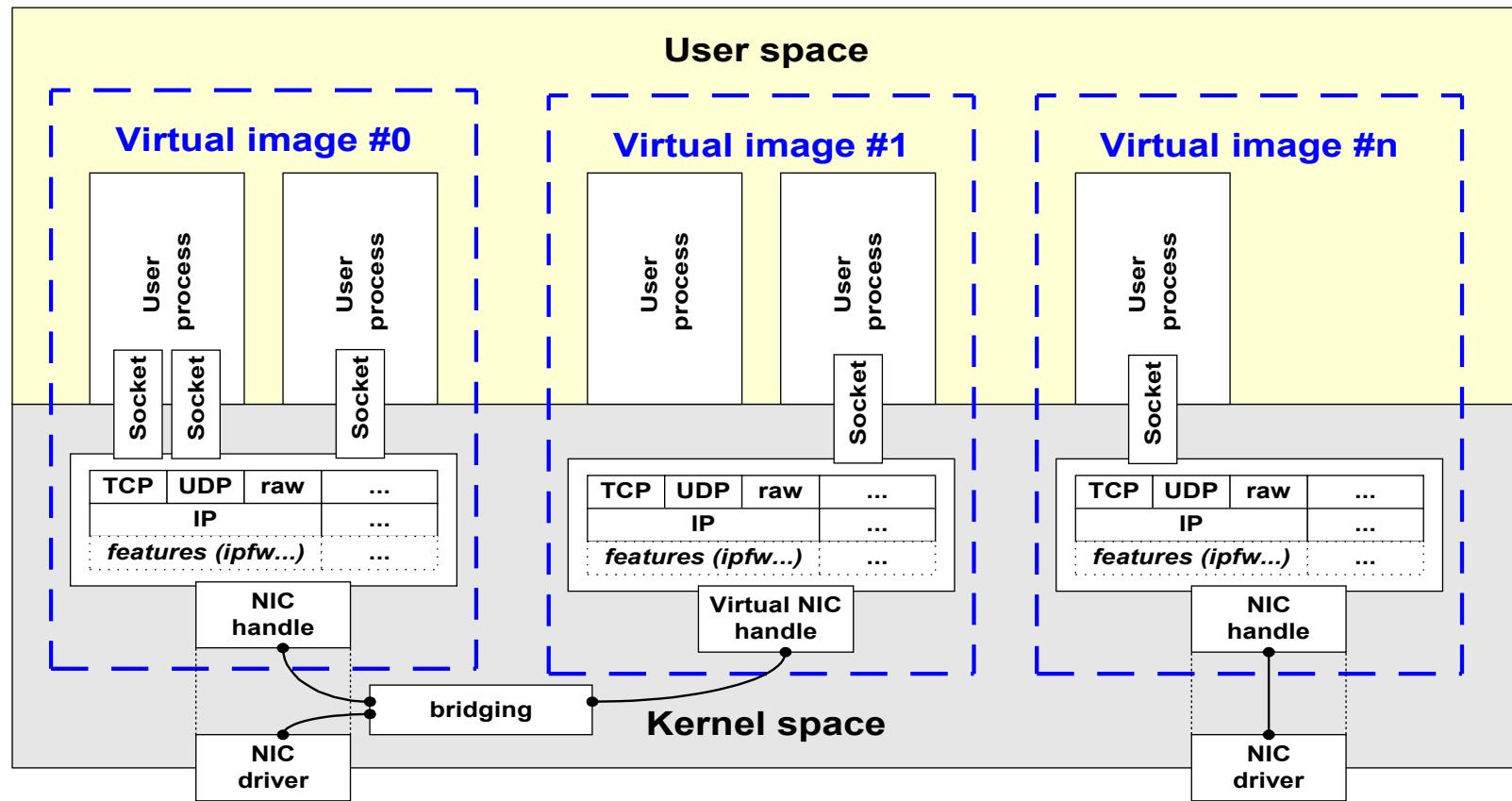
- Graphical User Interface (GUI): management plane, automates virtual node and link instantiation and configuration
- Emulator core is the OS kernel

What did we get?

- Each virtual node has its own IPv4 / IPv6 / IPSec / TCP / UDP / ipfw address space / state
- Standard UNIX APIs 100% preserved: all FreeBSD (and Linux!) binaries run *unmodified*
- Packet handoff by reference
- Efficient usage of hardware resources (RAM, CPU)
- Scalability: 100s of virtual nodes per physical machine (depending on applications)
- Fast experiment instantiation (seconds not minutes)
- Virtualized network stack in FreeBSD since 8.0-R: rebuild kernel with “options VIMAGE” enabled

Virtualized network stack

- One OS kernel instance operates on multiple instances of *state*



Key concepts to remember

- Sockets
 - Attach to a stack instance at creation time, cannot ever be reassigned / delegated to other stacks
- Network interfaces
 - Belong to only one network stack instance at a time
 - Some interface types can be reassigned to other stacks
- User processes
 - Bound to only one stack at a time
 - Started via jail / jexec

Evolution of virtualized networking on FreeBSD

- Jails (FreeBSD 4.0, around 1999)
 - Confined processes in chrooted environments
 - Each jail bound to a single IP address
 - Routing, firewalls etc. controlled externally
 - Great success in virtual hosting applications
- Vimage (proof of concept, FreeBSD 4.6+, 2002)
 - Network-related global variables placed in a huge monolithic structure / container
 - Access to virtualized variables dereferenced via a base pointer

Evolution of virtualized networking on FreeBSD

- Vimage (proof of concept, FreeBSD 4.6+, cont'd)
 - All networking functions extended with an additional argument: pointer to the virtualization container
 - Code churn confined above device driver layer and below socket layer
 - Virtualized IPv4, IPv6, IPSec, transport layer, ipfw
 - Reasonably stable, fast, got attention from ISPs
 - Problems: changes too intrusive, cannot be conditionally compiled, all fields must be reserved & anchored in the virtualization container -> kernel ABI issues, 4.x branch dead around 2005

Evolution of virtualized networking on FreeBSD

- Vimage step II (FreeBSD 7+, from late 2006 on)
 - Funding from NLNet, FreeBSD foundation
 - All changes conditionally compilable through extensive use of macros
 - Vnet context a thread-local variable: no need to pass extended arguments to networking functions
 - Broken up the huge virtualization container into smaller substructures in an attempt to relax kernel ABI constraints
 - Pioneered hierarchical management model
 - Code kept in sync with -CURRENT in p4

Evolution of virtualized networking on FreeBSD

- Merging into main FreeBSD tree (2008 - 2009)
 - Several large-scale incremental commits, proven to have no impact on existing code functionality
 - Vimage / Jail integration: vnets hang off of jails, hierarchical jails introduced
 - Virtualized variables tagged at compile time to be placed in a special contiguous linker section
 - Each vnet instance gets its own copy of that section
 - Spare space for virtualized vars from kldloadable modules
 - Solves global / static scope semantics, initializers, ABI
 - 8.0-RELEASE shipped as “options VIMAGE” capable

Evolution of virtualized networking on FreeBSD

- FreeBSD 8.1-RELEASE
 - Further funding from FreeBSD Foundation
 - Significantly improved per-subsystem destructors (fewer resource leaks & crashes)
 - Many subsystems still not virtualized (PF, CARP...)
- Future goals
 - Production quality from 9.0?
 - Single process with sockets in multiple stacks
 - Generalize the virtualization framework for application to other kernel subsystems

Vnet intercommunication

- epair(4)
 - A pair of back-to-back connected virtual Ethernet ifcs
 - Can be assigned to different vnets, bridged
- netgraph(4)
 - A modular kernel framework for network traffic manipulation, in general at link layer
 - *Nodes*: processing elements; *edge nodes* bridge to other subsystems: tty, socket, ksocket, ether, BT...
 - Nodes can be connected into arbitrary graphs via bidirectional *hooks*

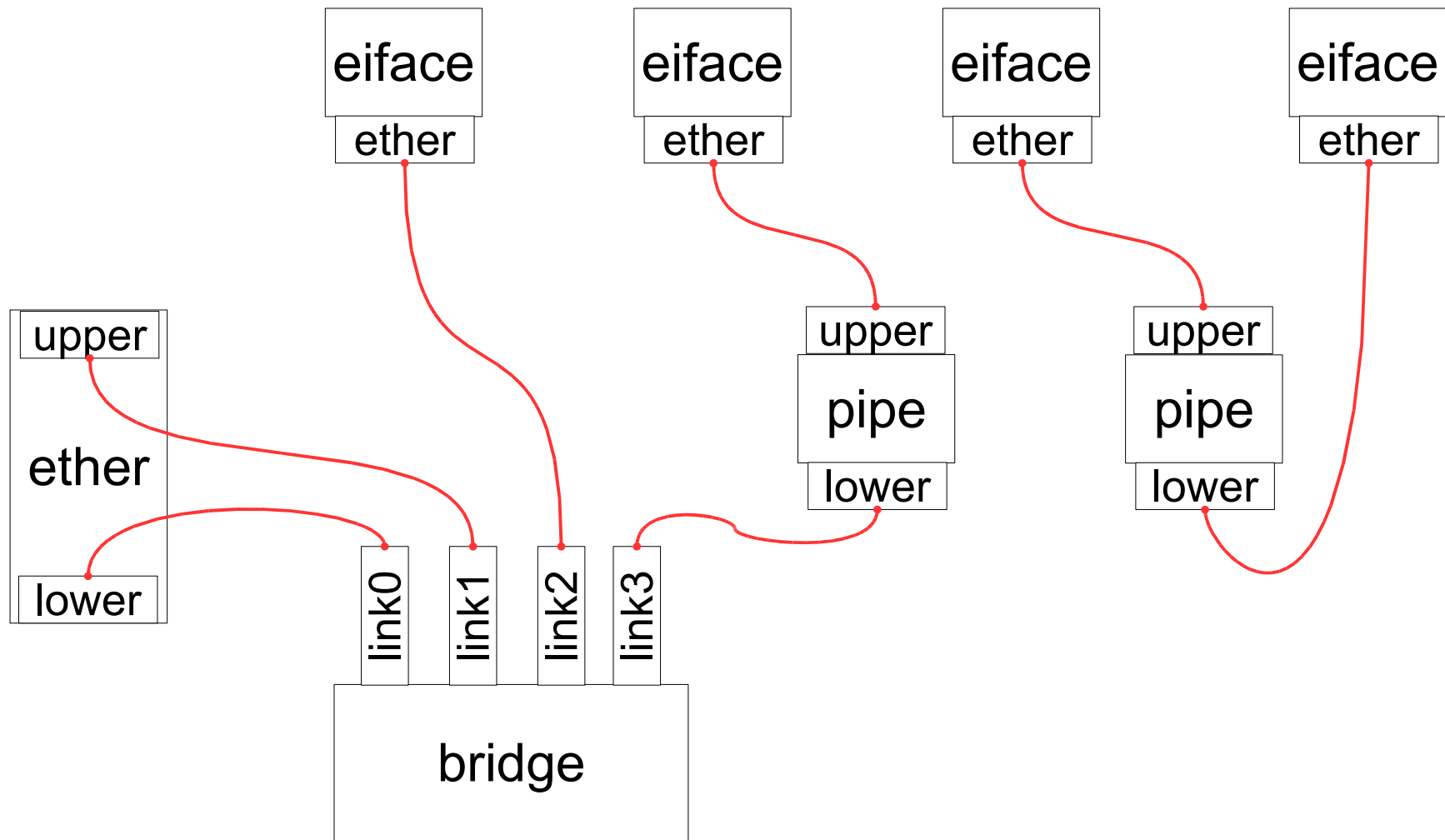
Netgraph node types

- `ng_ether(4)`
 - Hooks: upper, lower
 - Diverts / accepts real ethernet traffic to / from another netgraph node
 - Always associated with real ethernet / 802.11 interfaces
 - Automatic naming
- `ng_eiface(4)`
 - Only one hook: ether
 - Virtual interface providing ethernet framing

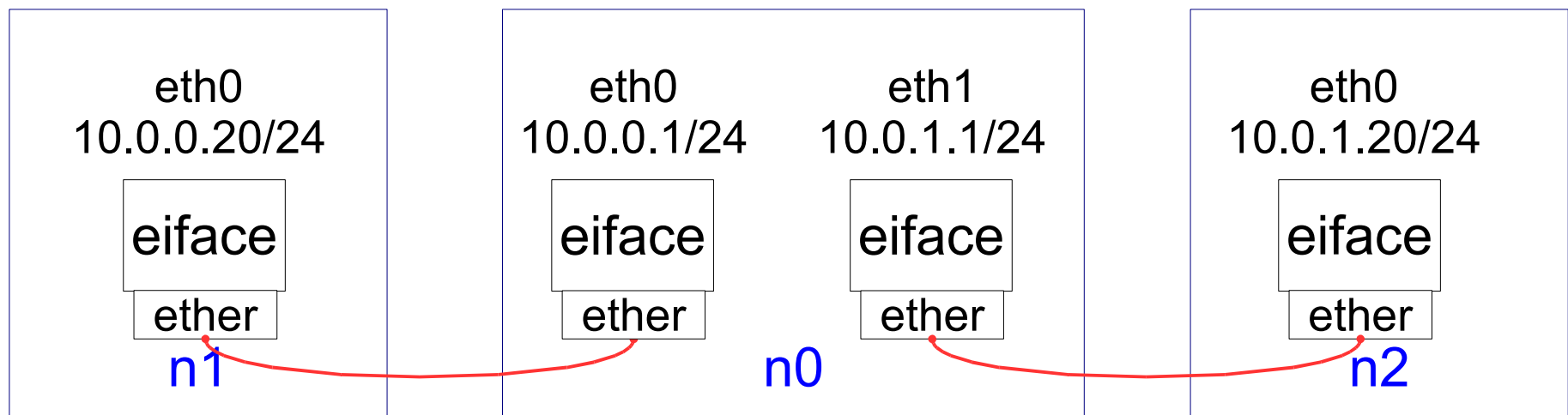
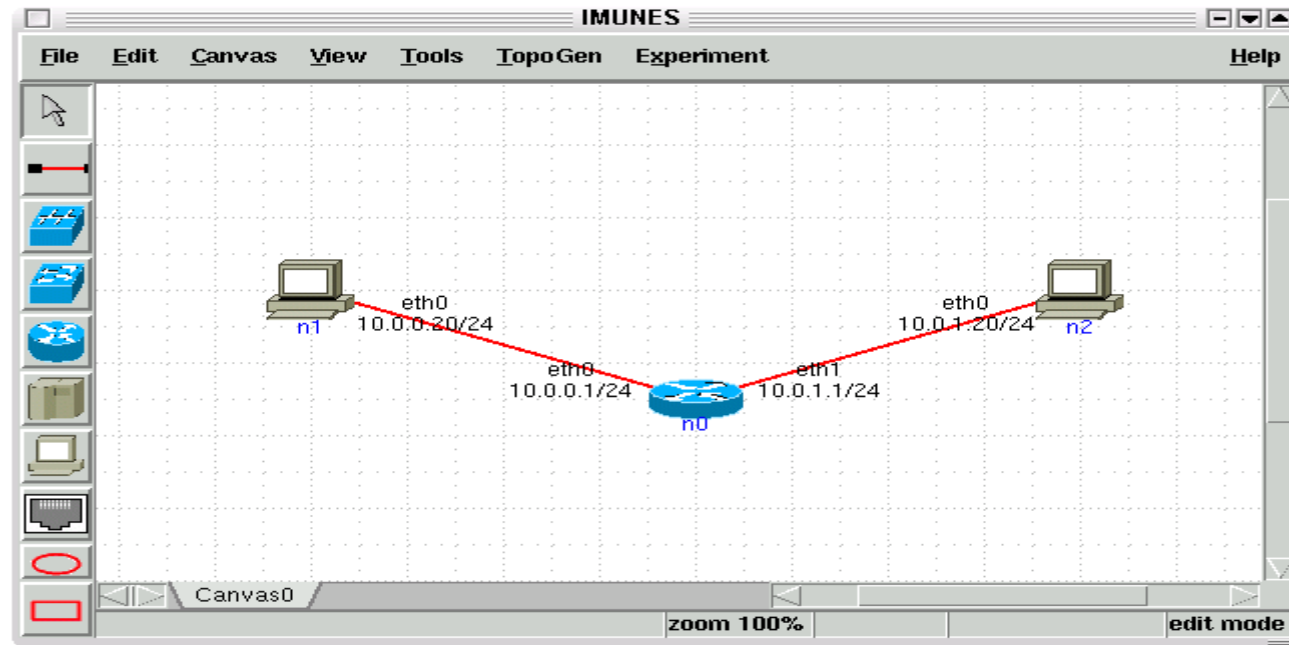
Netgraph node types

- `ng_iface(4)`
 - Hooks: `inet`, `inet6`, `ipx`, `atalk`, `ns`, `atm`, `natm`
 - Virtual interface emitting / accepting raw frames on protocol-specific hooks
- `ng_bridge(4)` / `ng_hub(4)`
 - Hooks: `link0`, `link1`...
- `ng_tee(4)`
 - Hooks: `left`, `right`, `left2right`, `right2left`
 - Snooping / tapping
- Many more -> `man 4 netgraph`

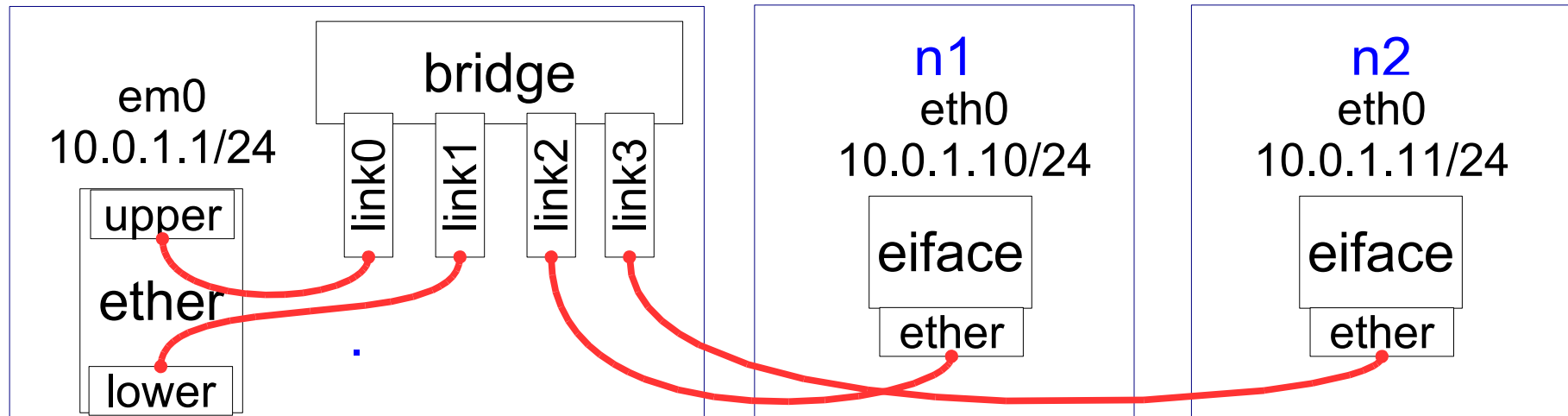
Netgraph: arbitrary topology interconnection



GUI, virtual nodes & netgraph in action

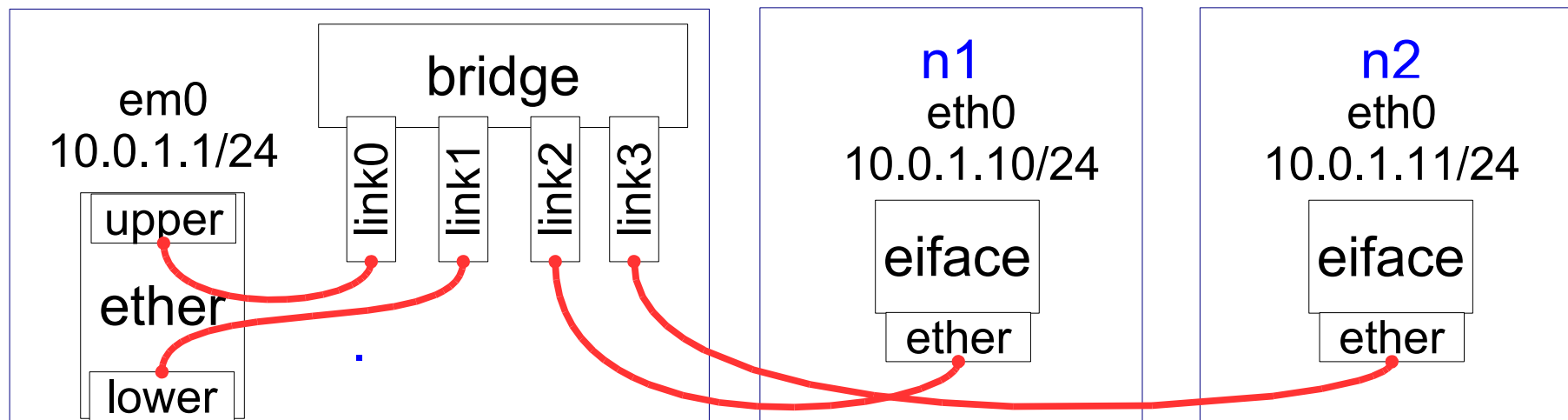


Example: vnet jails + netgraph



```
tpx32# jail -c name=n1 path=/ host.hostname=n1 vnet persist
tpx32# jail -c name=n2 path=/ host.hostname=n1 vnet persist
tpx32# jls
      JID  IP Address      Hostname      Path
      ---  -              -             -
      1    -              n1            /
      2    -              n1            /
tpx32# jexec n1 ifconfig lo0 localhost
tpx32# jexec n1 ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
      options=3<RXCSUM,TXCSUM>
      inet 127.0.0.1 netmask 0xff000000
      inet6 ::1 prefixlen 128
      inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
```

Example: vnet jails + netgraph



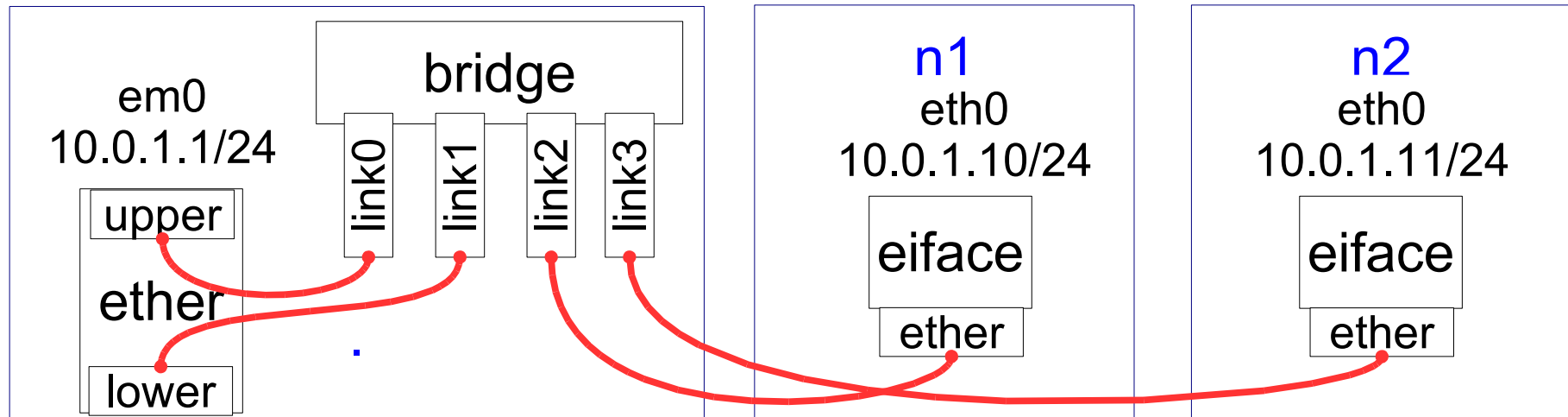
```
tpx32# ngctl mkpeer eiface ether ether
tpx32# ngctl mkpeer eiface ether ether
tpx32# ngctl l
```

There are 4 total nodes:

Name: em0	Type: ether	ID: 00000002	Num hooks: 0
Name: ngeth0	Type: eiface	ID: 00000004	Num hooks: 0
Name: ngeth1	Type: eiface	ID: 00000006	Num hooks: 0
Name: ngctl2874	Type: socket	ID: 00000007	Num hooks: 0

```
tpx32# ifconfig ngeth0 vnet n1
tpx32# ifconfig ngeth1 vnet n2
Tpx32# ifconfig em0 promisc
```

Example: vnet jails + netgraph

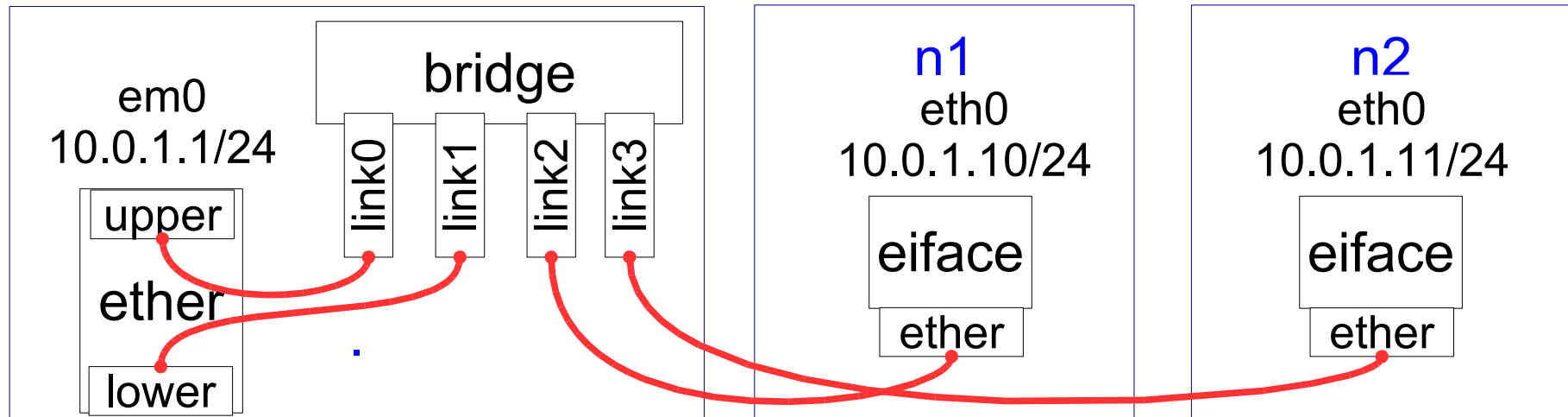


```
tpx32# ngctl mkpeer em0: bridge upper link0
tpx32# ngctl name em0:upper bridge
tpx32# ngctl connect em0: bridge: lower link1
tpx32# ngctl connect ngeth0: bridge: ether link2
tpx32# ngctl connect ngeth1: bridge: ether link3
tpx32# ngctl l
```

There are 5 total nodes:

Name: em0	Type: ether	ID: 00000002	Num hooks: 2
Name: ngeth0	Type: eiface	ID: 00000004	Num hooks: 1
Name: ngeth1	Type: eiface	ID: 00000006	Num hooks: 1
Name: ngctl2919	Type: socket	ID: 00000010	Num hooks: 0
Name: bridge	Type: bridge	ID: 0000000a	Num hooks: 4

Example: vnet jails + netgraph



```
ttpx32# jexec n1 csh
n1# ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
    options=3<RXCSUM,TXCSUM>
    inet 127.0.0.1 netmask 0xff000000
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
ngeth0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
    ether 00:00:00:00:00:00
n1# ifconfig ngeth0 name eth0
n1# ifconfig eth0 link 2:a:b:c:d:e
n1# ifconfig eth0 192.168.0.123
n1# ping ...
```

Resources

- Install 8.1-RELEASE (or 9-CURRENT), rebuild the kernel with “options VIMAGE” enabled
- If interested in network emulation, fetch the GUI tool from <http://imunes.net/imunes>
- Alternatively look at CORE, a fork of IMUNES from Boeing: <http://cs.itd.nrl.navy.mil/work/core>
- <http://wiki.freebsd.org/Image>
- FreeBSD-virtualization mailing list: virtualization@freebsd.org
- If in doubt: zec@freebsd.org

Backup & junk slides

Backup & junk slides

Global variable virtualization magic

```
--- releng_7/sys/net/if.c   Fri Feb  5 17:30:15 2010
+++ releng_8/sys/net/if.c   Fri Jul   2 10:54:26 2010
```

```
+#include <net/vnet.h>
```

```
-static int if_indexlim = 8;
+static VNET_DEFINE(int, if_indexlim) = 8;
+#define V_if_indexlim VNET(if_indexlim)
```

```
@@ -369,39 +385,19 @@ if_grow(void)
```

```
-     if_indexlim <= 1;
-     n = if_indexlim * sizeof(*e);
+     V_if_indexlim <= 1;
+     n = V_if_indexlim * sizeof(*e);
```

In sys/net/vnet.h:

```
#define VNET_SETNAME "set_vnet"
__asm__( ".section " VNET_SETNAME " , \"aw\" , @progbits\n"
         "\t.previous" );
#define VNET_NAME(n) vnet_entry_##n
#define VNET_DEFINE(t, n) t VNET_NAME(n) __section(VNET_SETNAME) __used
```

Global variable virtualization magic

```
+#define V_if_indexlim VNET(if_indexlim)

@@ -369,39 +385,19 @@ if_grow(void)

-     if_indexlim <<= 1;
-     n = if_indexlim * sizeof(*e);
+     V_if_indexlim <<= 1;
+     n = V_if_indexlim * sizeof(*e);
```

In sys/net/vnet.h:

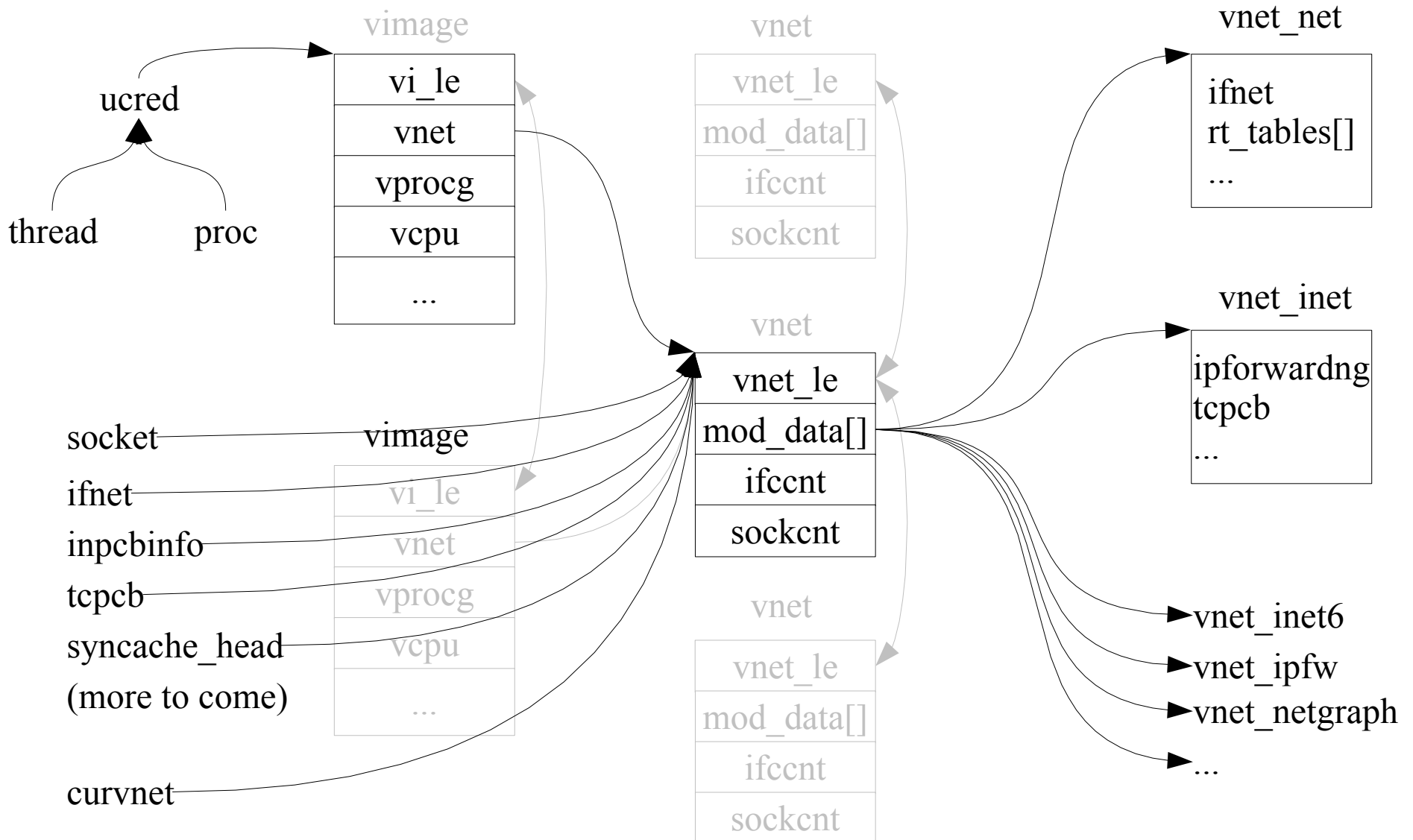
```
#define VNET_NAME(n) vnet_entry_##n

#define _VNET_PTR(b, n)          ( __typeof(VNET_NAME(n))*          \
                                ((b) + (uintptr_t)&VNET_NAME(n)) )
#define _VNET(b, n)            (*_VNET_PTR(b, n))

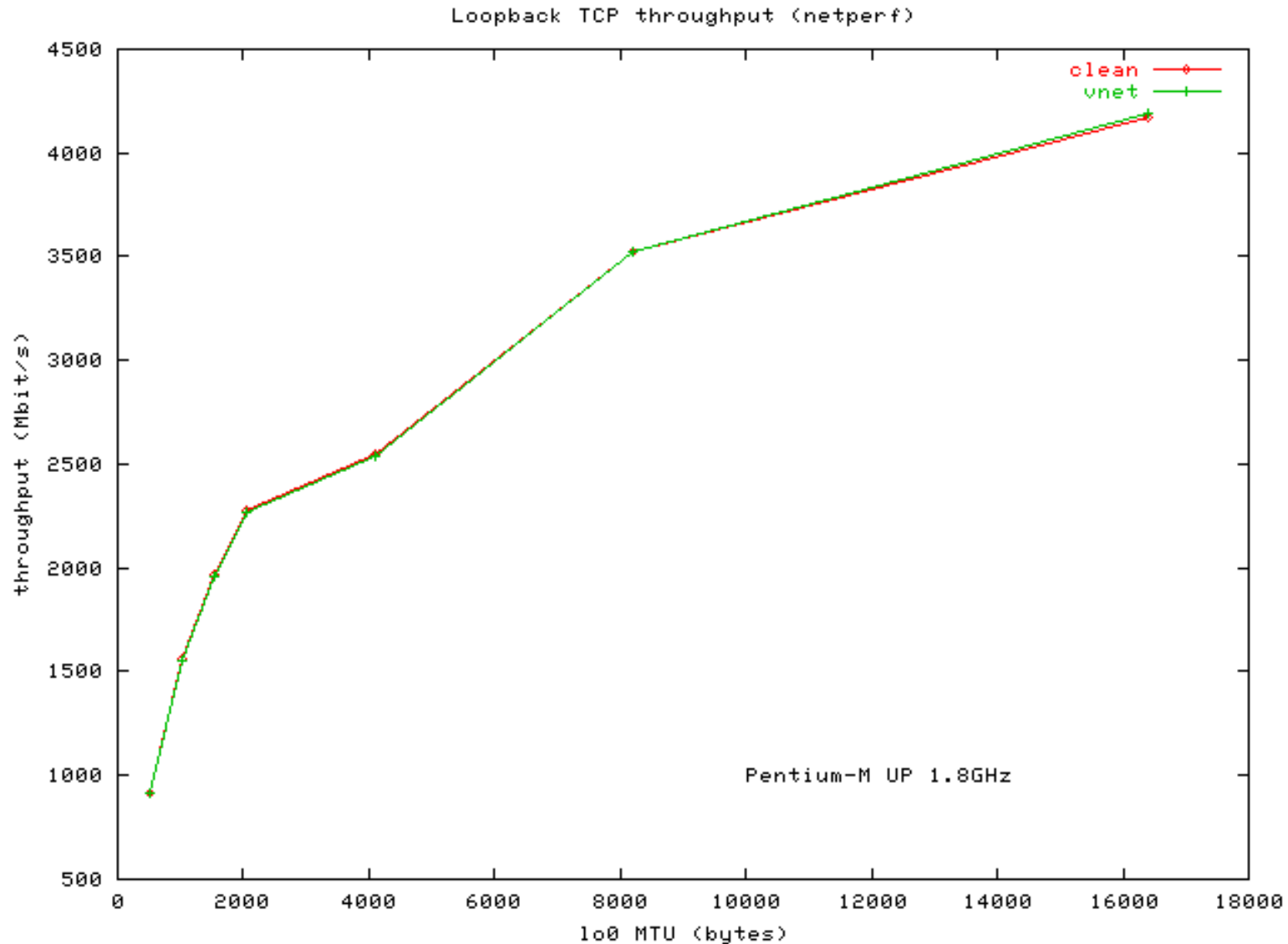
#define VNET_VNET_PTR(vnet, n)  _VNET_PTR((vnet)->vnet_data_base, n)
#define VNET_VNET(vnet, n)     (*VNET_VNET_PTR((vnet), n))

#define VNET_PTR(n)             VNET_VNET_PTR(curvnet, n)
#define VNET(n)                 VNET_VNET(curvnet, n)
```

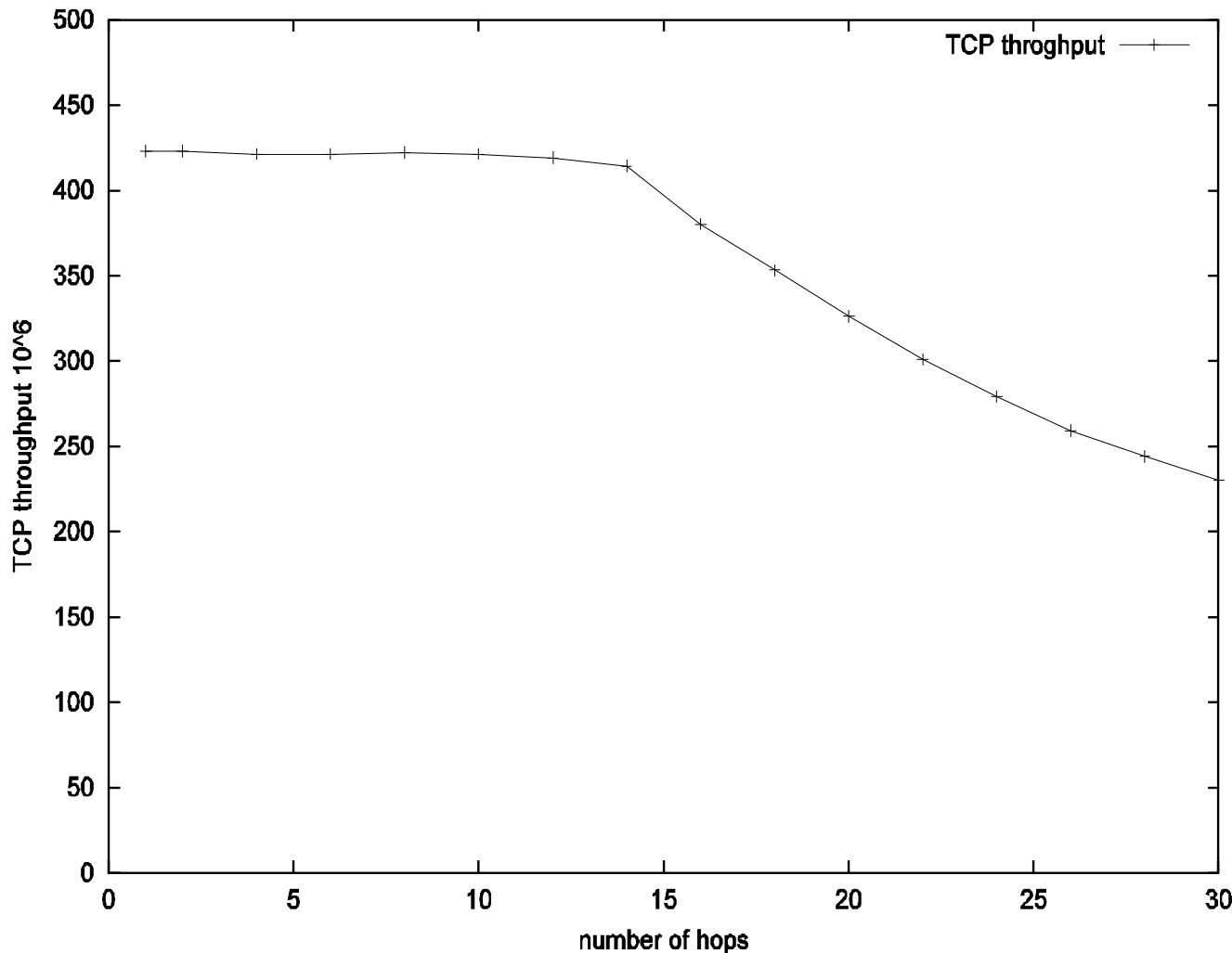
vnet container structure in FreeBSD 7.x



Loopback TCP throughput in FreeBSD 7



TCP throughput – multihop routing in FreeBSD 4.x



External:

420 Mbps TCP

36k+18k = 54 Kpps

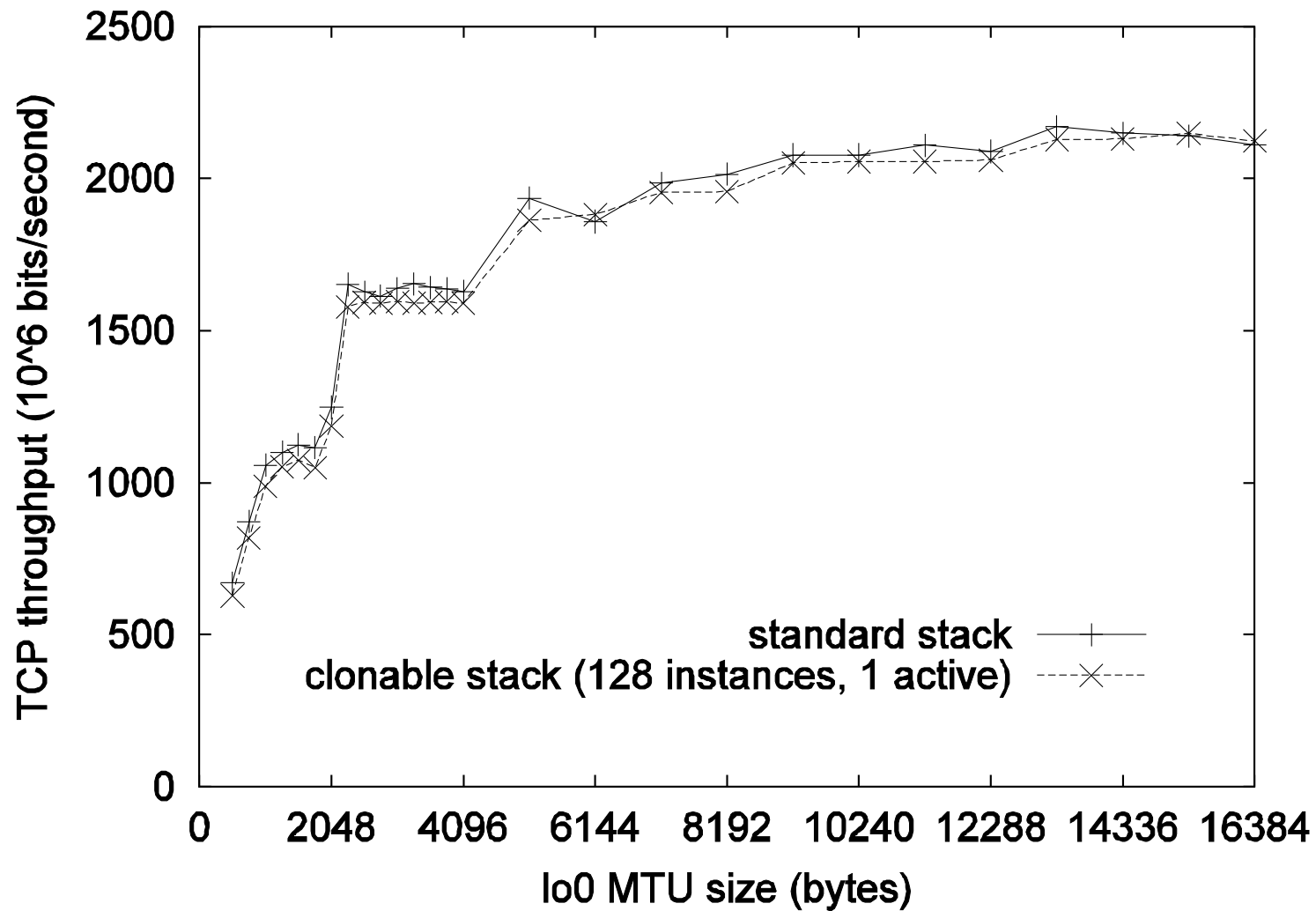
Internal:

14 hops * 54 Kpps

= **~800.000 pps**

P4 @ 2.8 GHz

Loopback TCP throughput in FreeBSD 4.x



Loopback ICMP packet rate in FreeBSD 4.x

