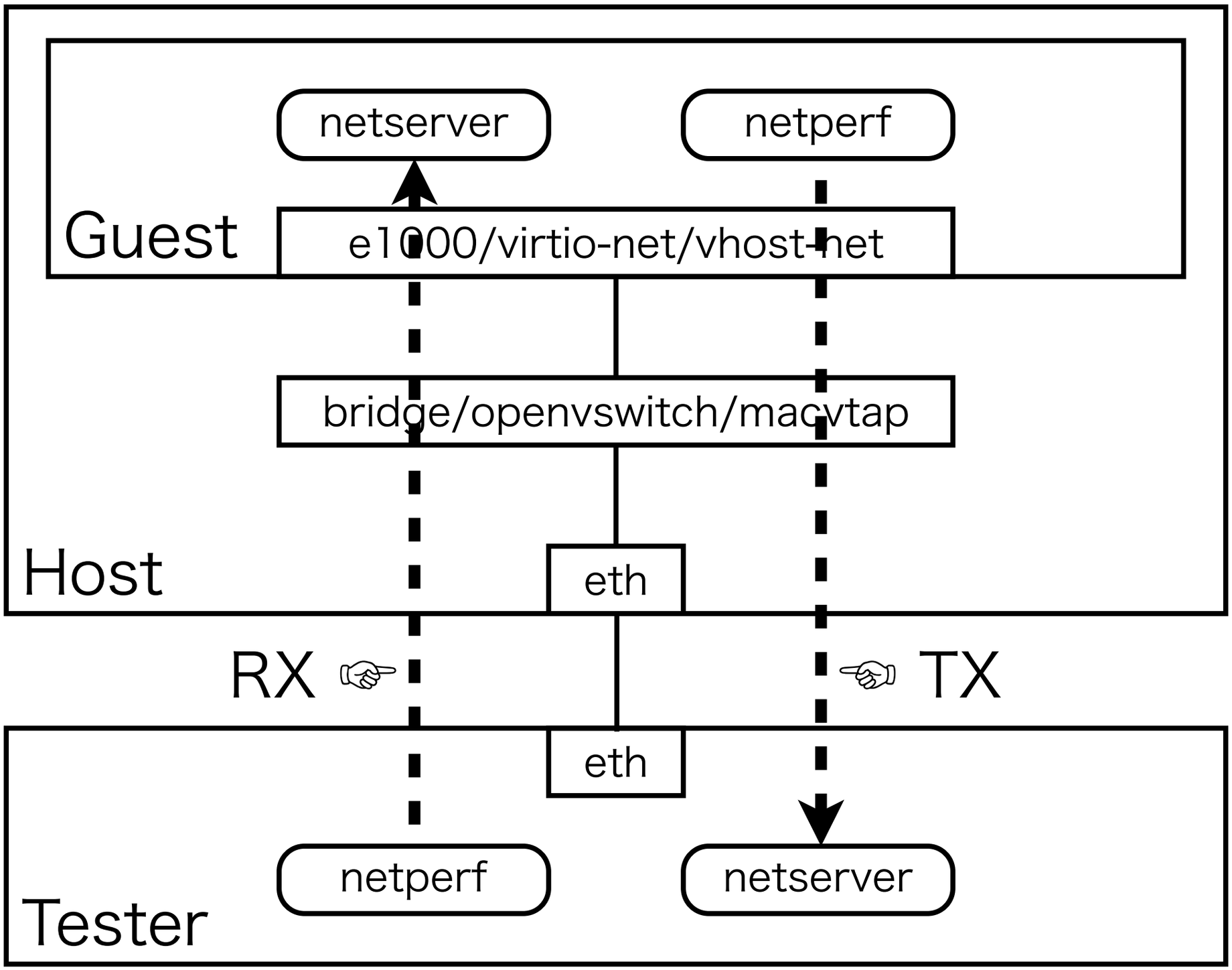
極める！ KVM

浅間 正和 有限会社 銀座堂

今日のおはなし

- 物理 NIC と仮想 NIC を接続する方法として以下の３つを説明します
  - ✓ bridge
  - ✓ openvswitch
  - ✓ macvtap
- 仮想 NIC の実装として以下の３つを説明しそれぞれでの性能計測の結果を示します
  - ✓ e1000
  - ✓ virtio-net
  - ✓ vhost-net(virtio-net w/ vhost-net)
- 性能計測の結果からいろいろ考察したりします

性能計測環境: 構成


netserver
netperf
Guest
e1000/virtio-net/vhost-net
bridge/openvswitch/macvtap
Host
eth
RX
TX
eth
netperf
netserver
Tester

性能計測環境: Host と Tester

Distribution	Fedora 17
Linux Kernel	3.6.3 (※1)(※2)
QEMU-KVM	1.2.0 (※3)
Netperf	2.6.0
H/W Model	HP ProLiant DL120 G7
CPU	Intel Xeon E31220 @ 3.1GHz x 1
Memory	DDR3 Synchronous 1333 MHz 8GB
NIC	Intel 82599EB 10GBASE-T

(※1) Fedora の patch を適用しない Vanilla Kernel を利用
(※2) configuration は Fedora のものから IOMMU を除外
(※3) Fedora の patch を適用しない素の QEMU-KVM を利用

bridge

- 太古の昔から存在する bridge 実装
- brctl command で bridge 作成や port 追加を行う

```
# brctl addbr br0   ........ bridge br0 を作成
# brctl addif br0 eth0   ... br0 に eth0 を追加
# brctl addif br0 eth1   ... br0 に eth1 を追加
```

  - ☞ 上記の例では eth0 と eth1 が接続された br0 という bridge が作成される
  - ☞ eth0 が受信した Ethernet frame は宛先 MAC address に応じて eth1 から送信される
- 基本的な機能しかもたない

openvswitch

- 比較的最近の bridge 実装
- Linux Kernel 3.3 から main line に merge
- ovs-vsctl command で bridge の作成や port の追加を行う

```
# ovs-vsctl add-br ovs0  ... bridge ovs0 を作成
# ovs-vsctl add-port ovs0 eth0
                              ... ovs0 に eth0 を追加
# ovs-vsctl add-port ovs0 eth1
                              ... ovs0 に eth1 を追加
```

☞ 上記の例では eth0 と eth1 が接続された ovs0 という bridge が作成される

openvswitch

- Ethernet frame の宛先 MAC address、送信元 MAC address、 Ethernet type field、宛先 IP address、送信元 IP address、Protocol field、宛先 port 番号、送信元 port 番号、といった情報を元に内部で flow を管理する

  ☞ datapath と呼ばれる kernel module
- 未知の flow の Ethernet frame が datapath に届いた際は daemon に判断を仰ぐ

  ☞ openvswitchd と呼ばれる daemon process
- packet header に基づく filter 機能や帯域制限のような QoS 機能、 GRE による Ethernet over IP tunnel 機能など多彩な機能を持つ

tap

- file handler と仮想 NIC を結びつける仮想 device

```
struct ifreq ifr;
int fd;
fd = open(“/dev/net/tun”, O_RDWR);
ifr.ifr_flags = IFF_TAP | IFF_NO_PI;
strncpy(ifr.ifr_name, "vnet%d", IFNAMSIZ);
ioctl(fd, TUNSETIFF, (void *) &ifr);
```

- 上記 program 例の場合 vnet0 のような仮想 NIC が生成され fd に write するとそれが vnet0 から送信され fd を read すると vnet0 が受信したものを読むことができる

```
# brctl addif br0 vnet0
```

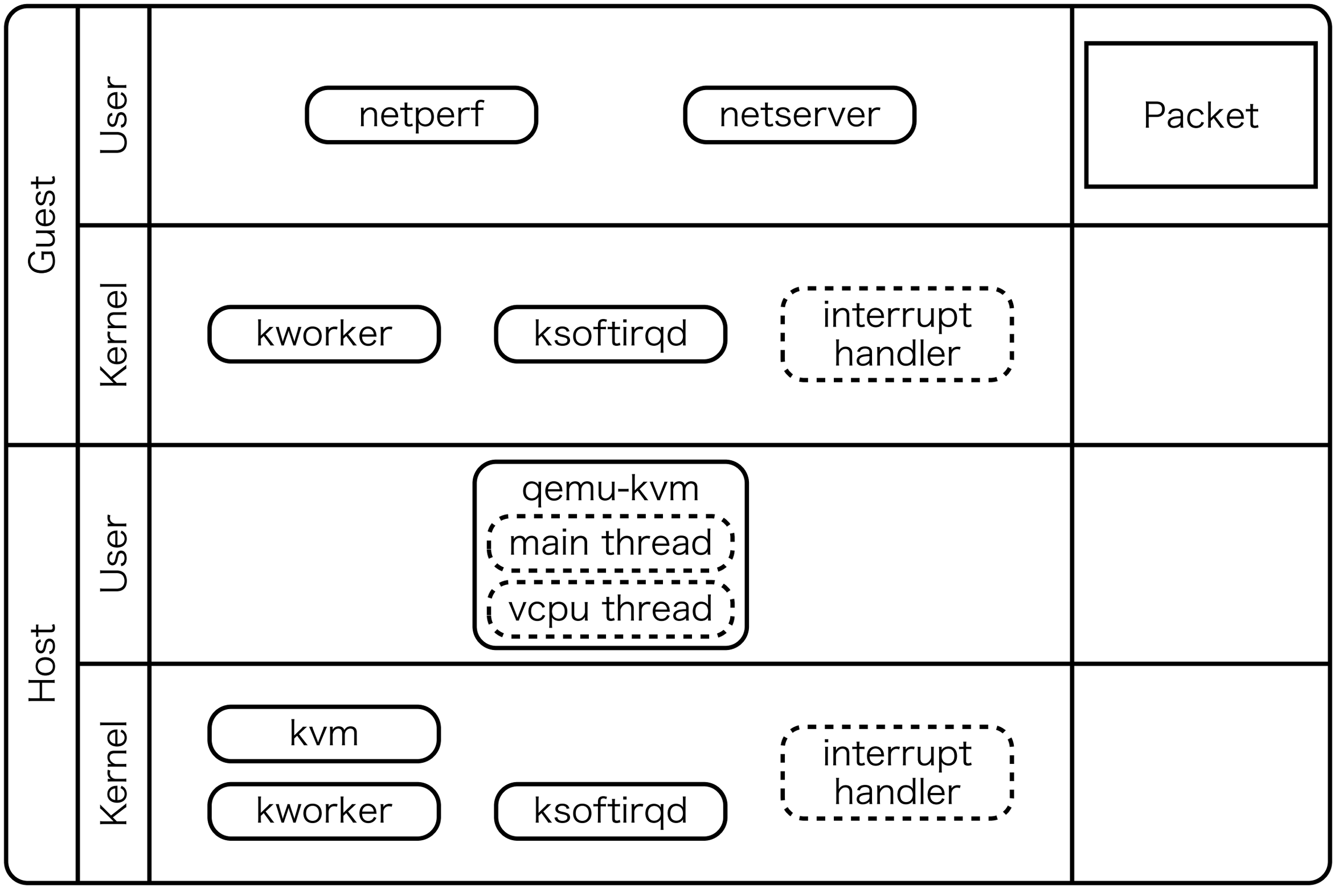
macvtap

- 比較的最近実装された仮想 NIC が物理 NIC に直接接続されているかのように出来る機能

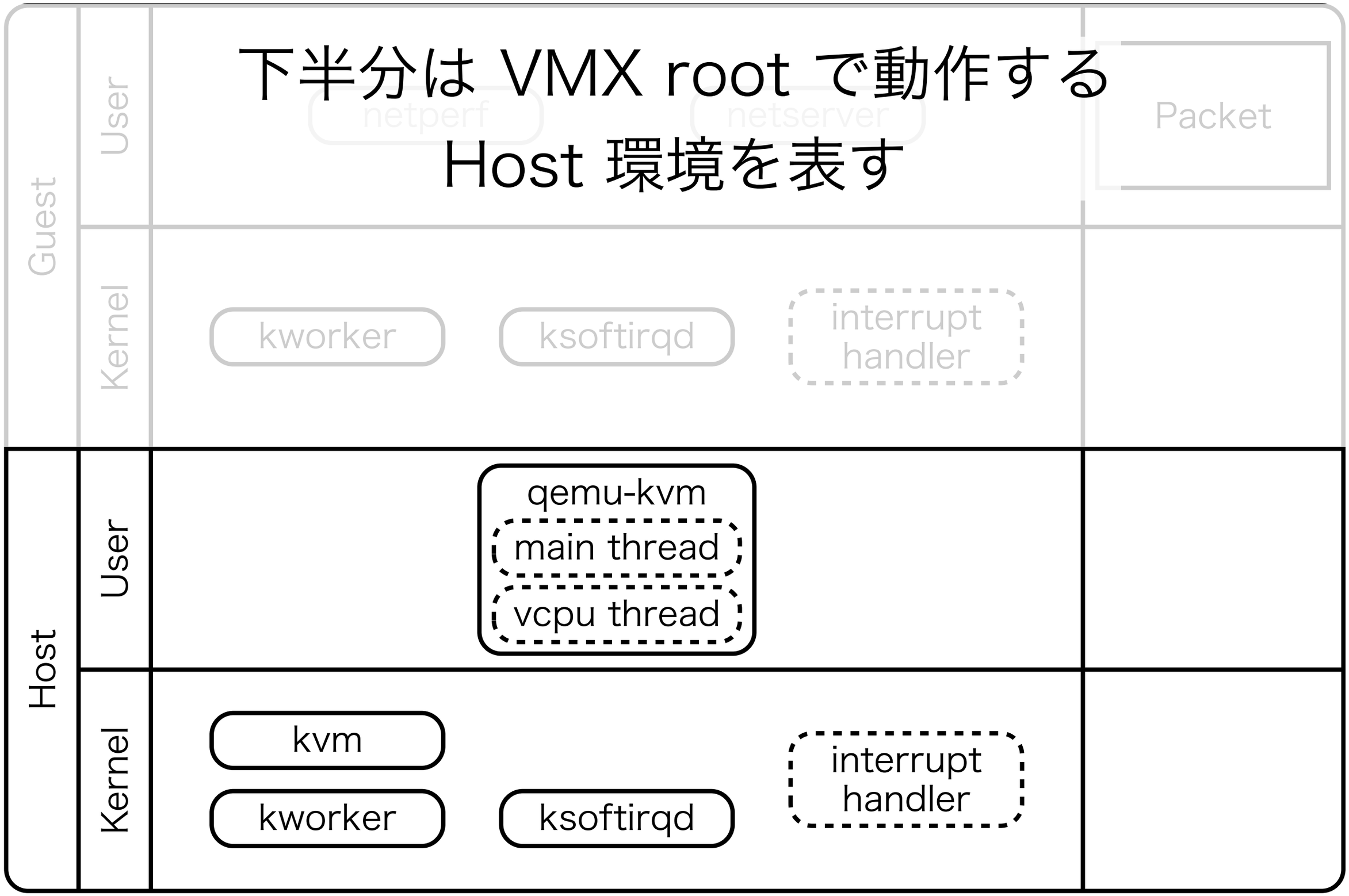
```
# ip link add link eth0 name mvtap0 ¥
                          type macvtap mode bridge
# ip link set mvtap0 address 1a:46:0b:ca:bc:7b
```

- 上記 command を実行すると udev が /dev/tap0 等を作成するのであとは tap と同様に利用する
- mode には以下の 4 種類が存在
  - ☞ bridge: 親 NIC と tap の間で通常の bridging
  - ☞ vepa: 全ての frame を親 NIC から送信
  - ☞ private: tap と tap の間の通信ができない
  - ☞ passthru: 親 NIC と tap で素通し

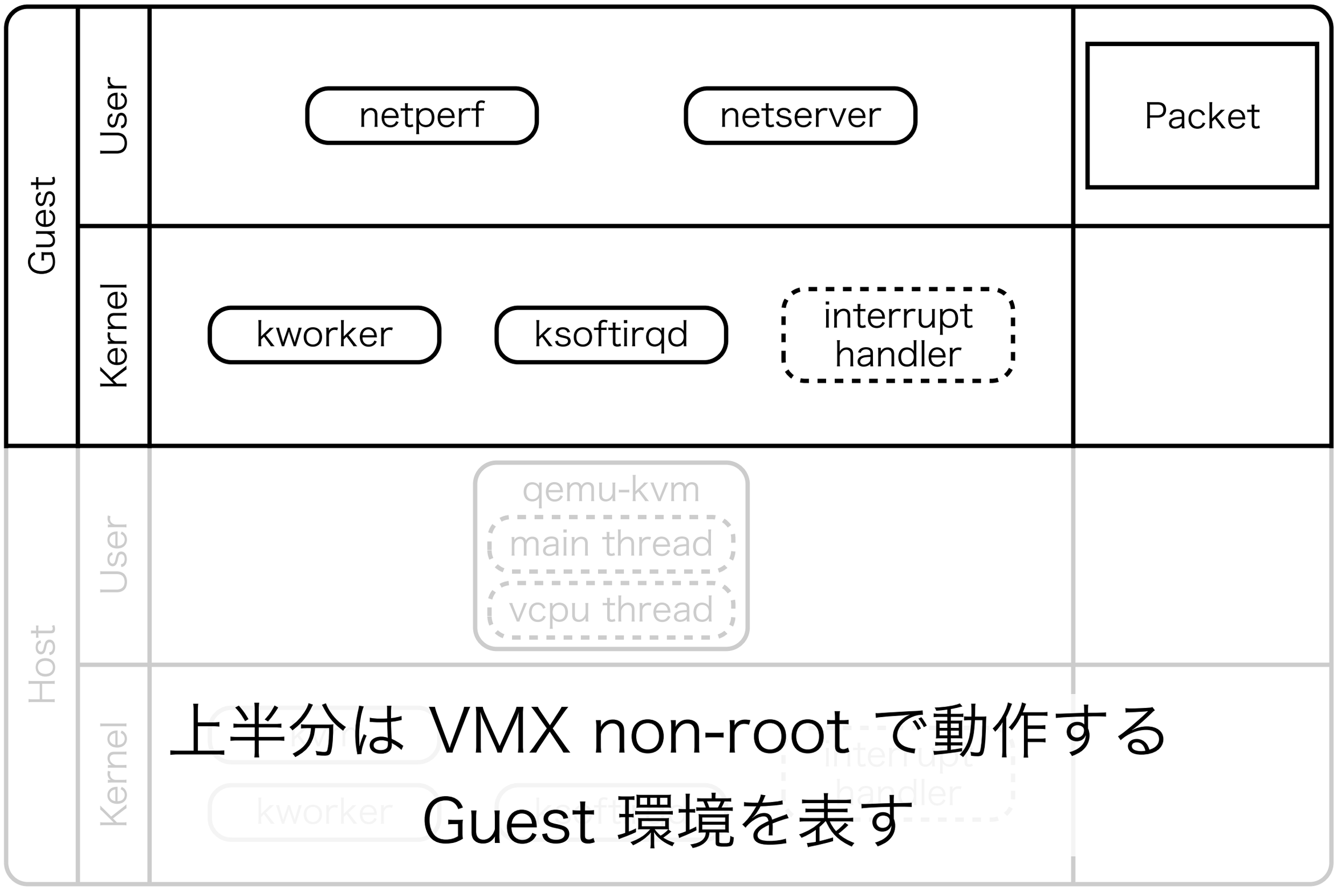
仮想 NIC driver の説明のための準備



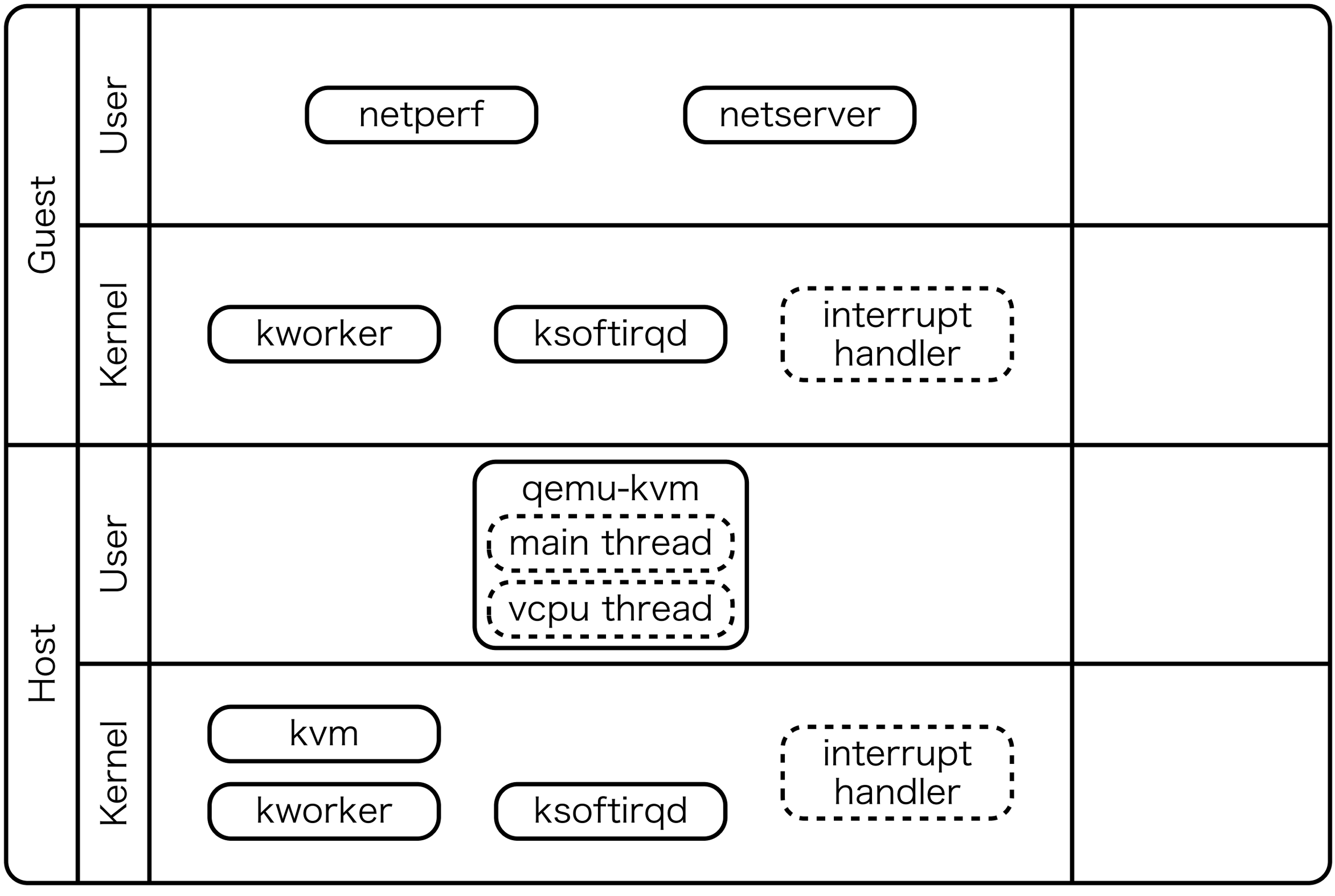
仮想 NIC driver の説明のための準備



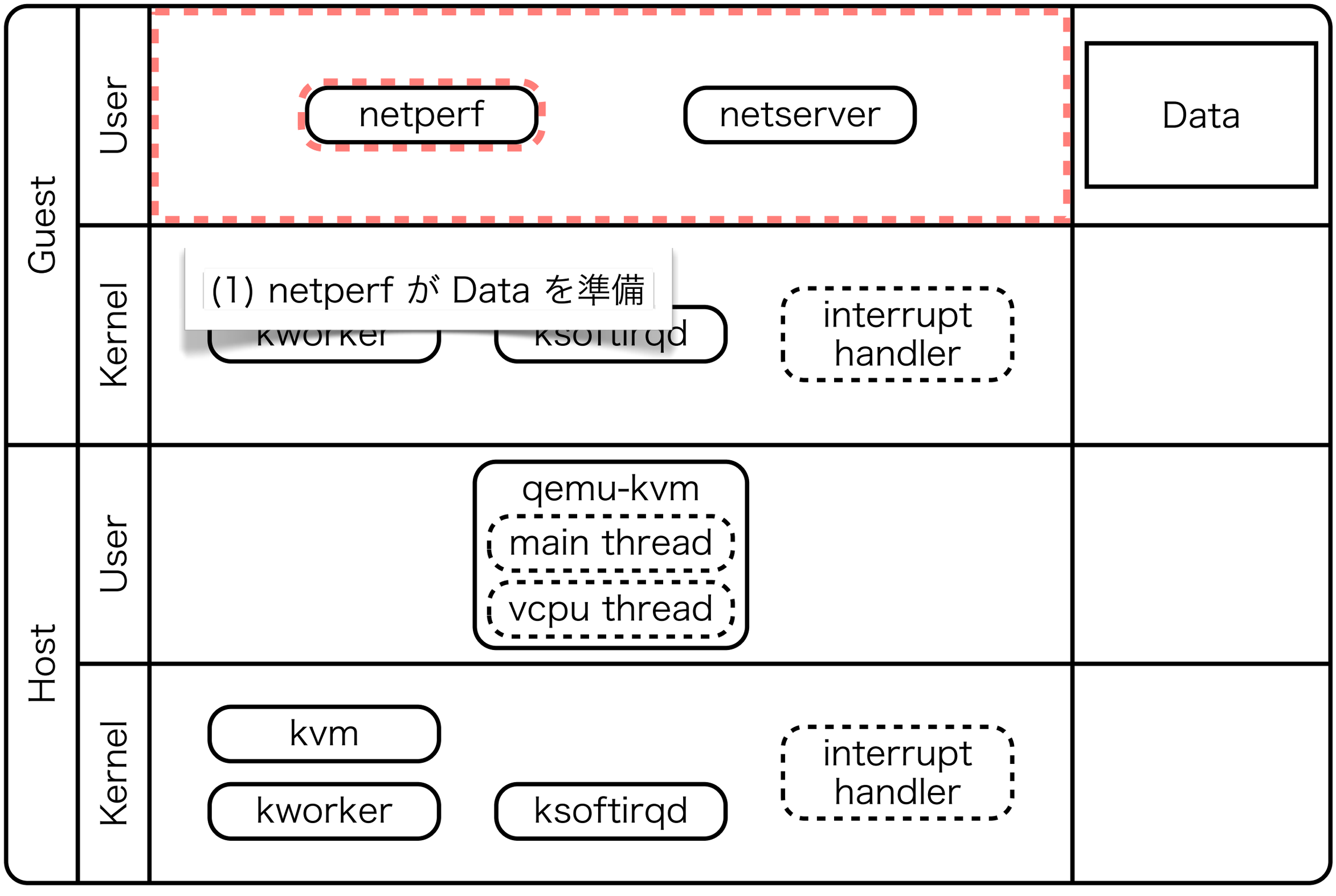
仮想 NIC driver の説明のための準備



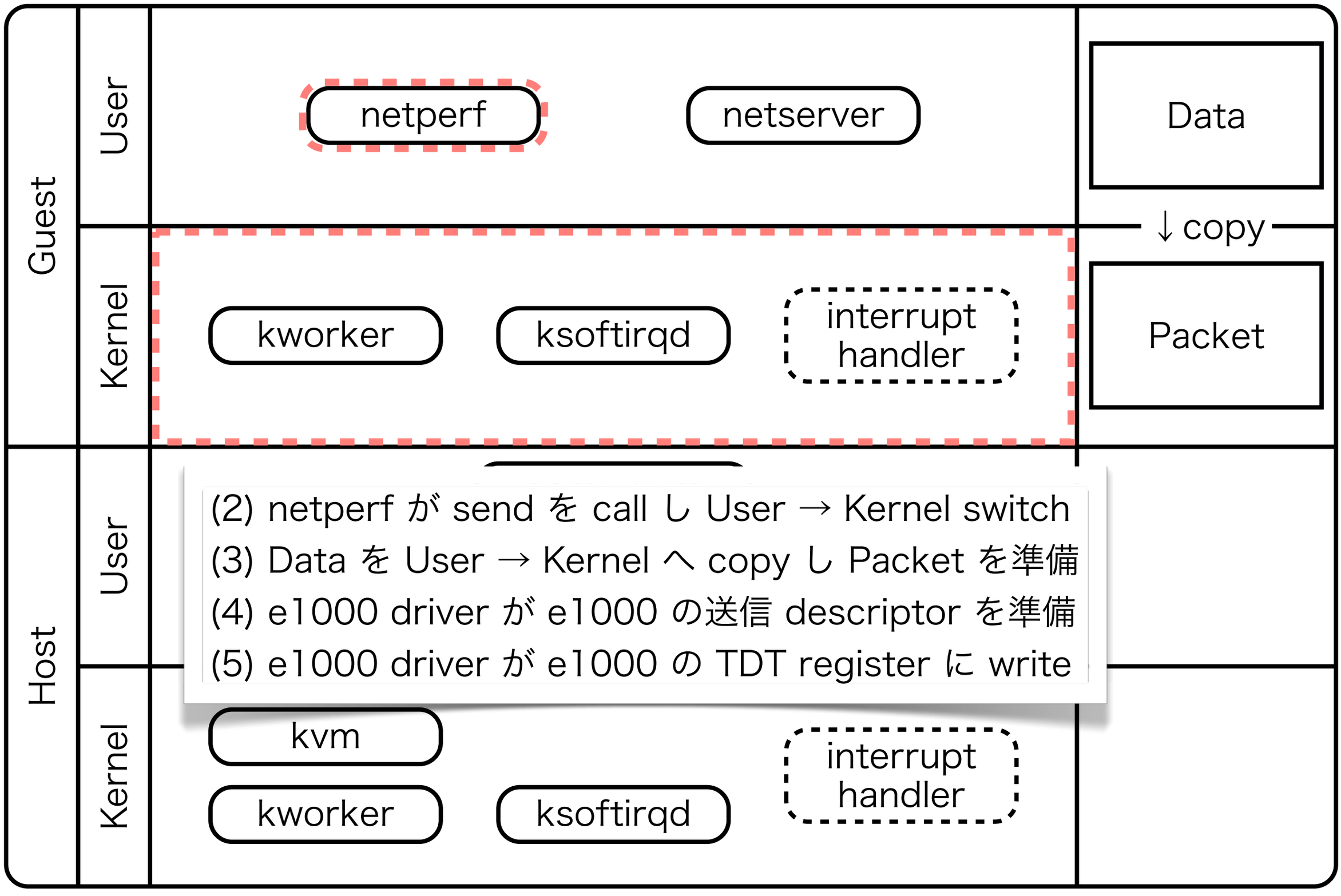
e1000: TX


Guest
User
netperf
netserver
Kernel
kworker
ksoftirqd
interrupt handler
Host
User
qemu-kvm
main thread
vcpu thread
Kernel
kvm
kworker
ksoftirqd
interrupt handler

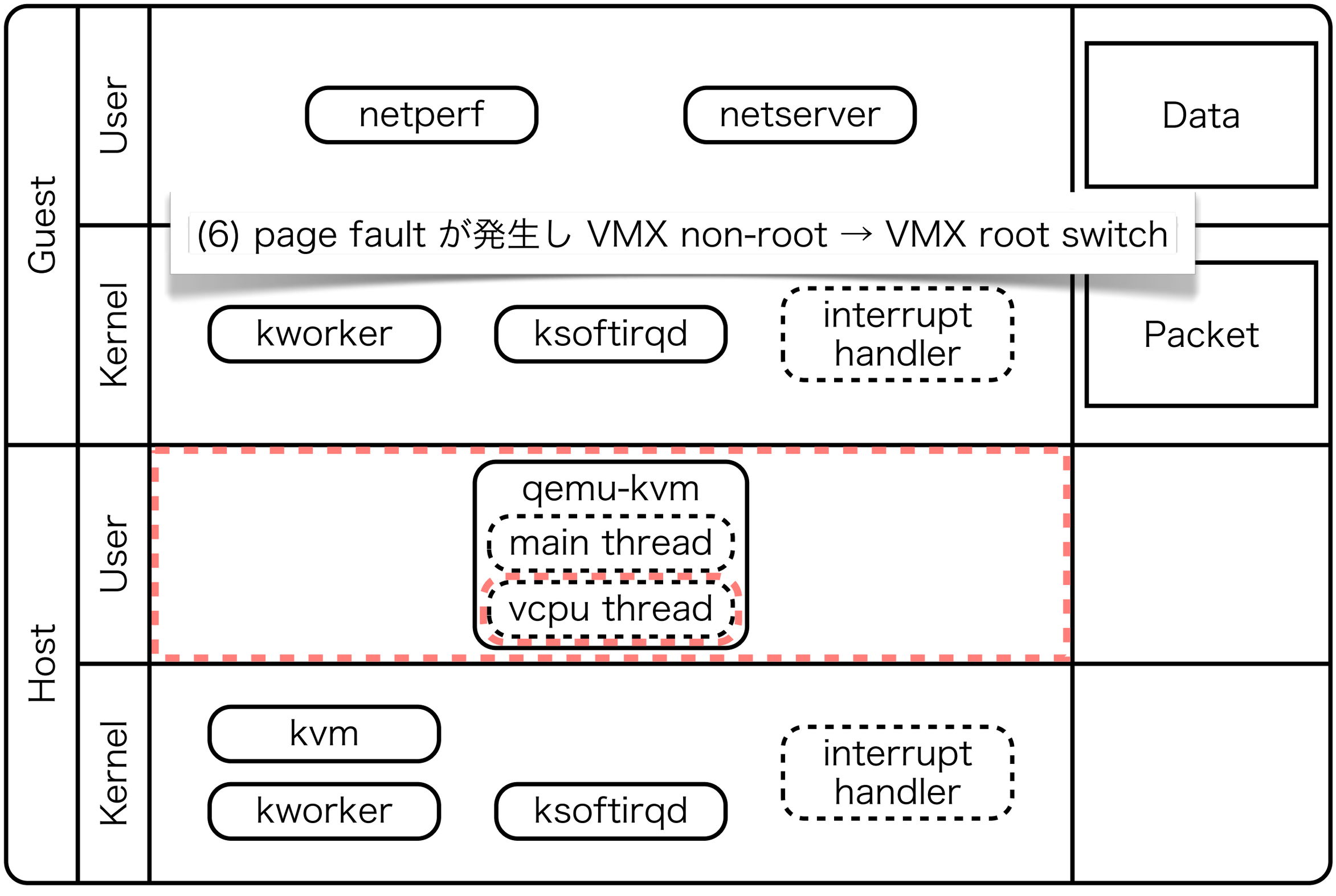
e1000: TX


Guest
User
netperf
netserver
Data
Kernel
(1) netperf が Data を準備
kworker
ksoftirqd
interrupt handler
Host
User
qemu-kvm
main thread
vcpu thread
Kernel
kvm
kworker
ksoftirqd
interrupt handler

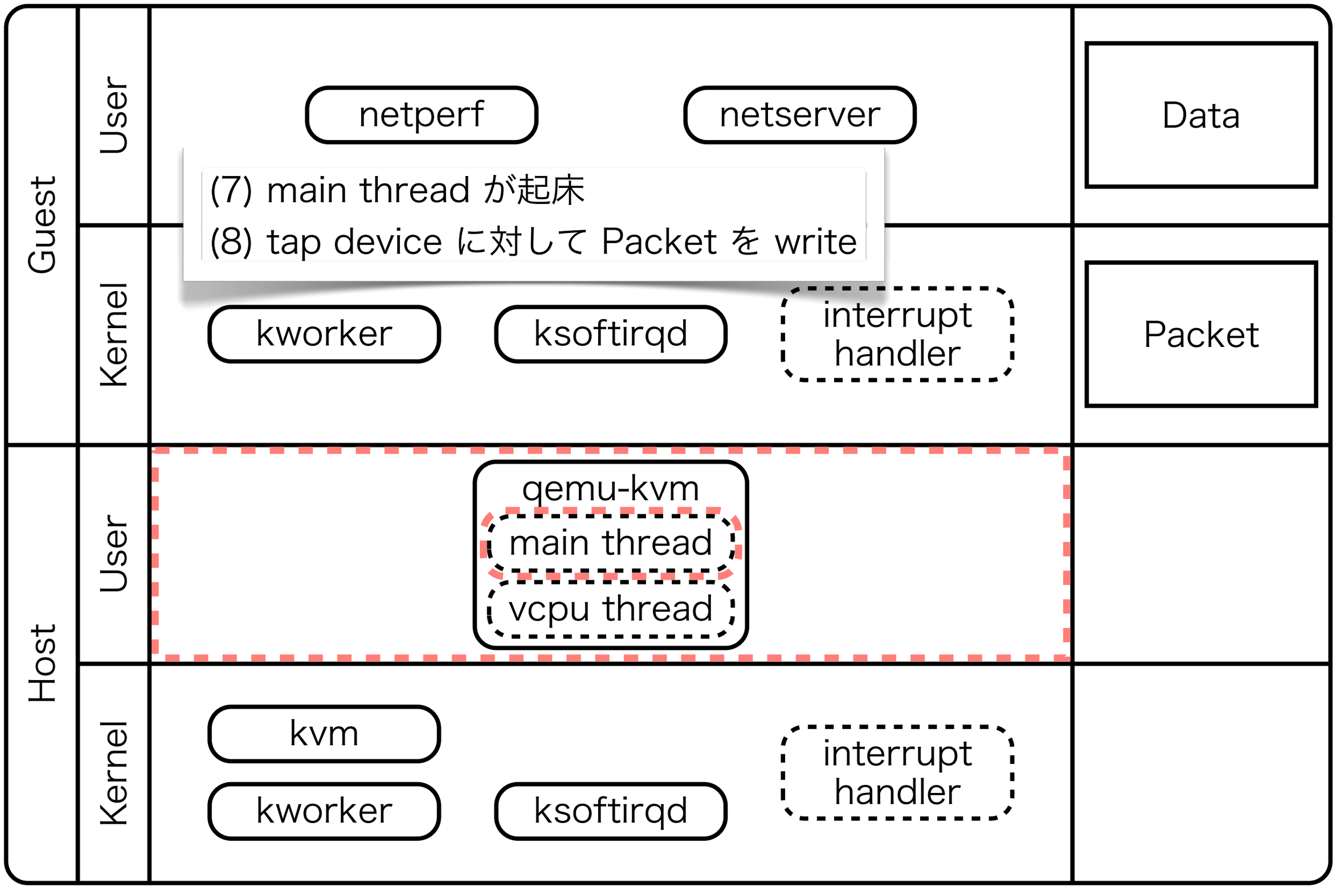
e1000: TX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
Data
↓ copy
kworker
ksoftirqd
interrupt handler
Packet
(2) netperf が send を call し User → Kernel switch
(3) Data を User → Kernel へ copy し Packet を準備
(4) e1000 driver が e1000 の送信 descriptor を準備
(5) e1000 driver が e1000 の TDT register に write
kvm
kworker
ksoftirqd
interrupt handler

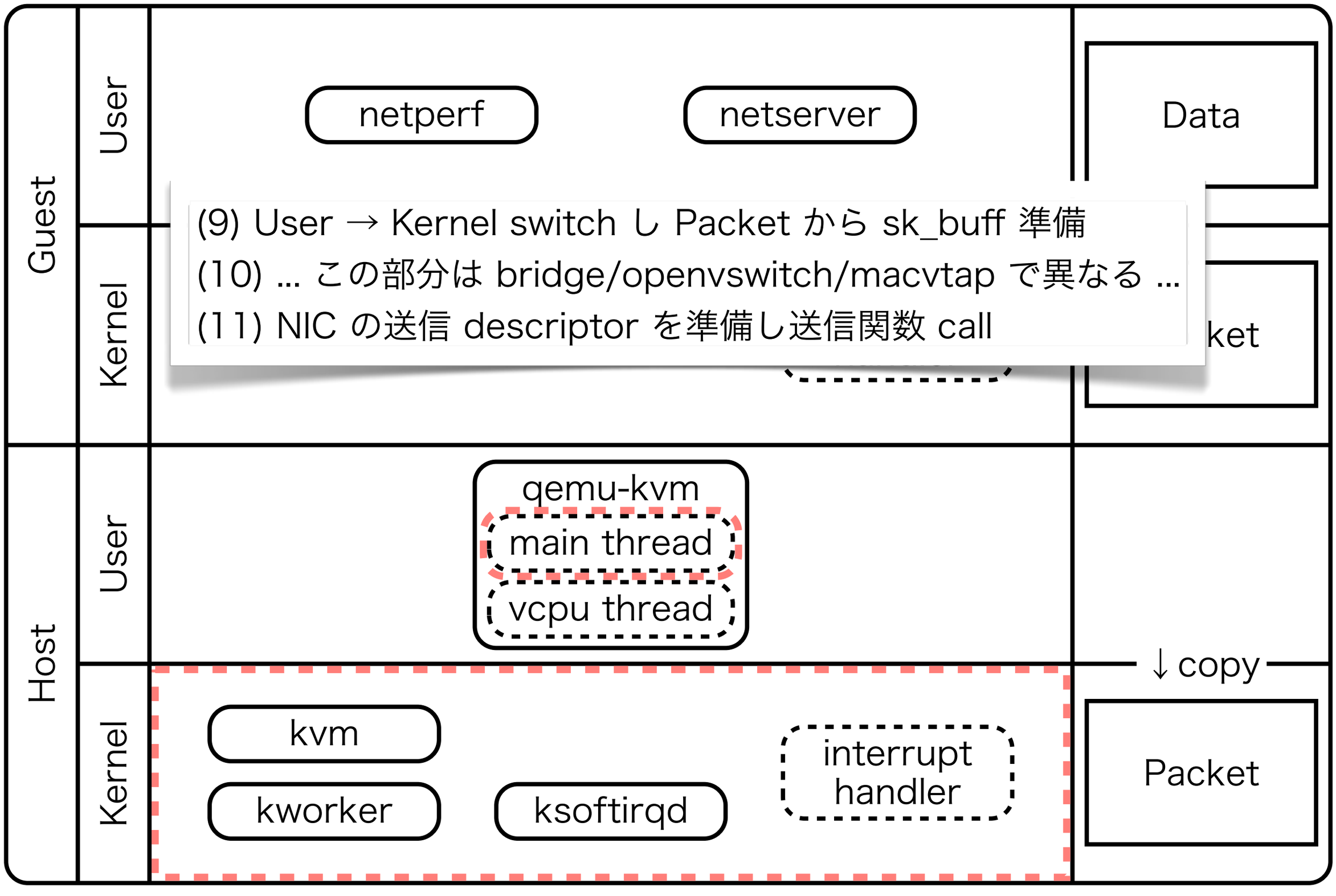
e1000: TX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
Data
(6) page fault が発生し VMX non-root → VMX root switch
kworker
ksoftirqd
interrupt handler
Packet
qemu-kvm
main thread
vcpu thread
kvm
kworker
ksoftirqd
interrupt handler

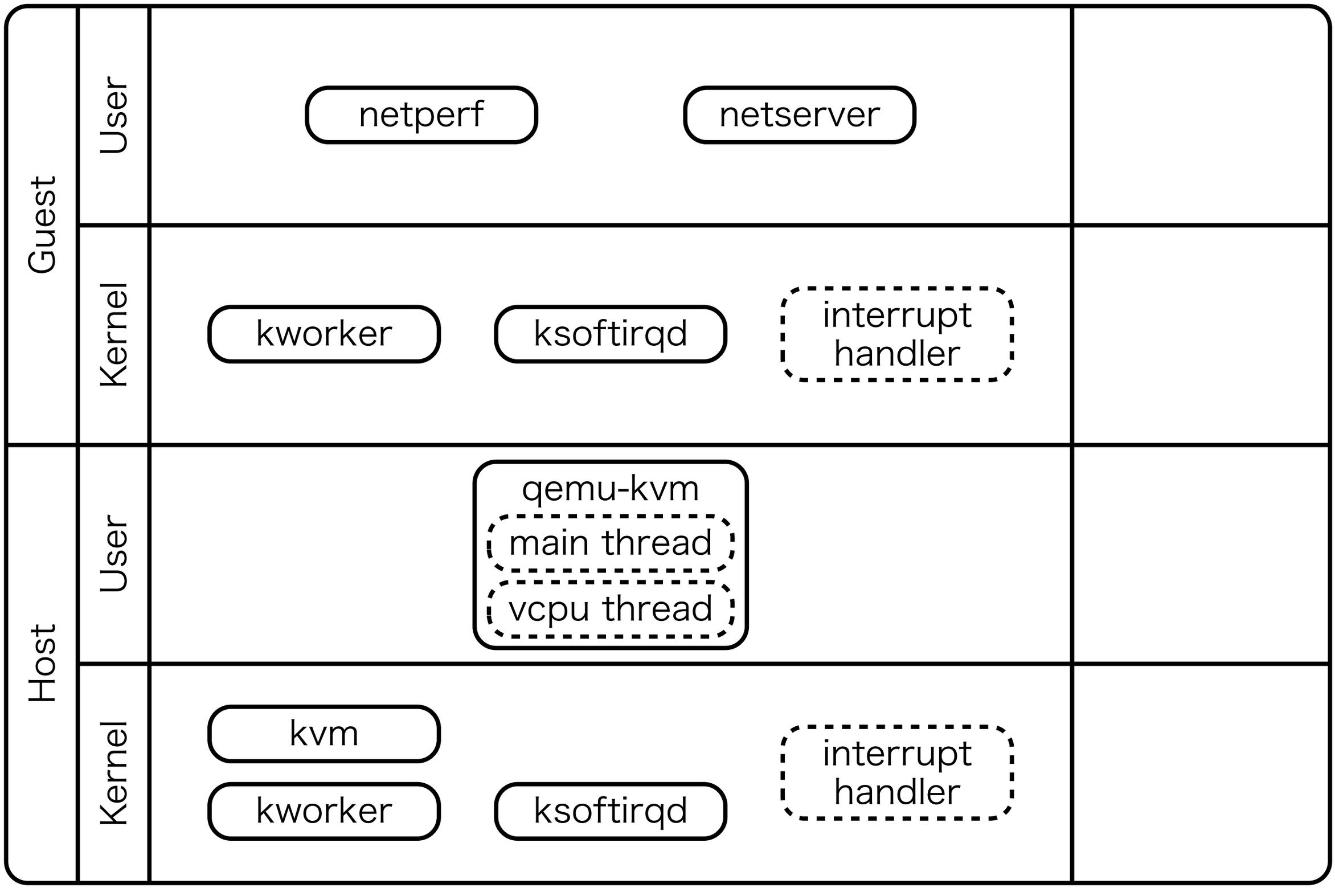
e1000: TX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
(7) main thread が起床
(8) tap device に対して Packet を write
kworker
ksoftirqd
interrupt handler
qemu-kvm
main thread
vcpu thread
kvm
kworker
ksoftirqd
interrupt handler
Data
Packet

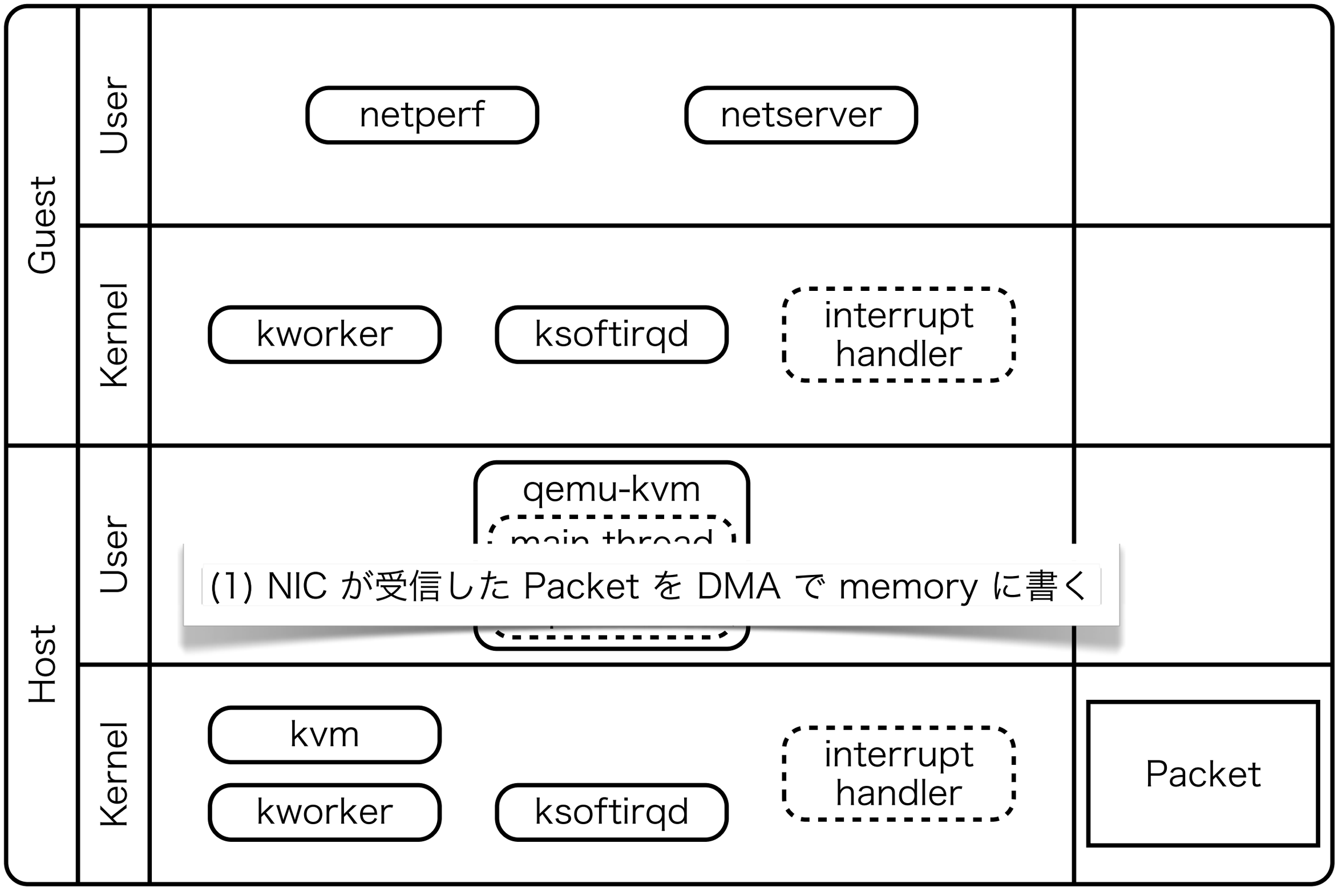
e1000: TX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
Data
(9) User → Kernel switch し Packet から sk_buff 準備
(10) ... この部分は bridge/openvswitch/macvtap で異なる ...
(11) NIC の送信 descriptor を準備し送信関数 call
ket
qemu-kvm
main thread
vcpu thread
↓ copy
kvm
kworker
ksoftirqd
interrupt
handler
Packet

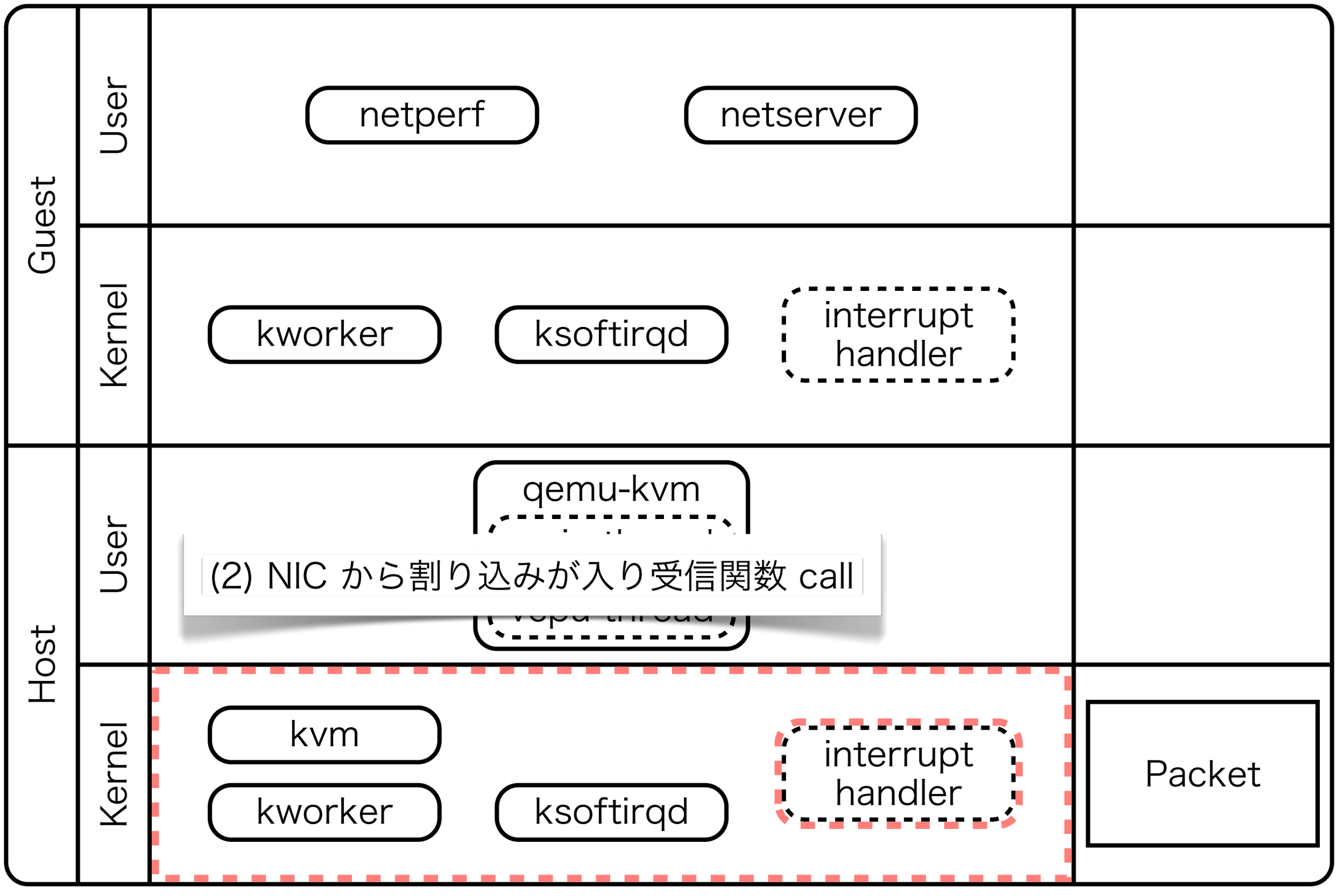
e1000: RX


Guest
User
netperf
netserver
Kernel
kworker
ksoftirqd
interrupt handler
Host
User
qemu-kvm
main thread
vcpu thread
Kernel
kvm
kworker
ksoftirqd
interrupt handler

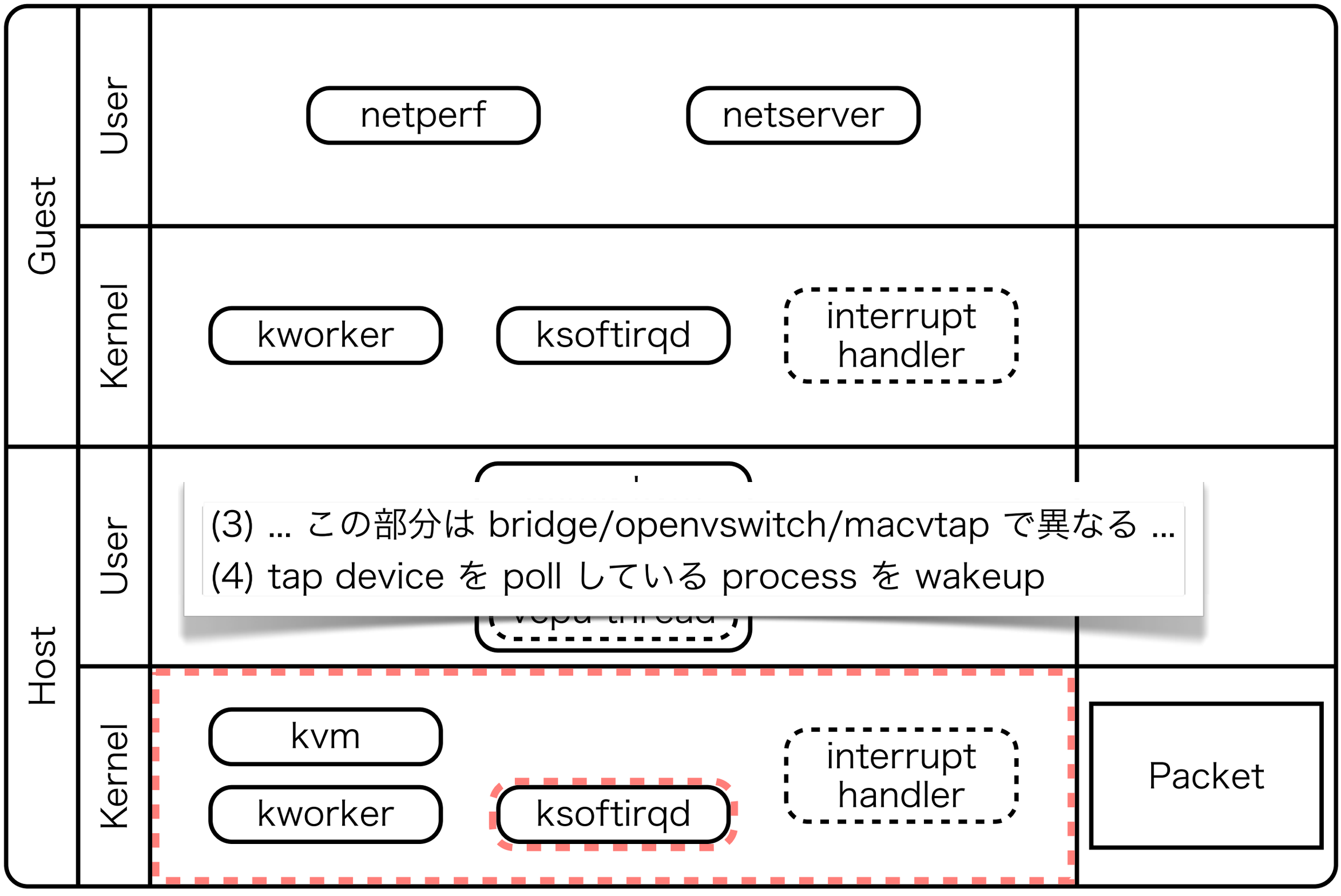
e1000: RX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
qemu-kvm
(1) NIC が受信した Packet を DMA で memory に書く
kvm
kworker
ksoftirqd
interrupt handler
Packet

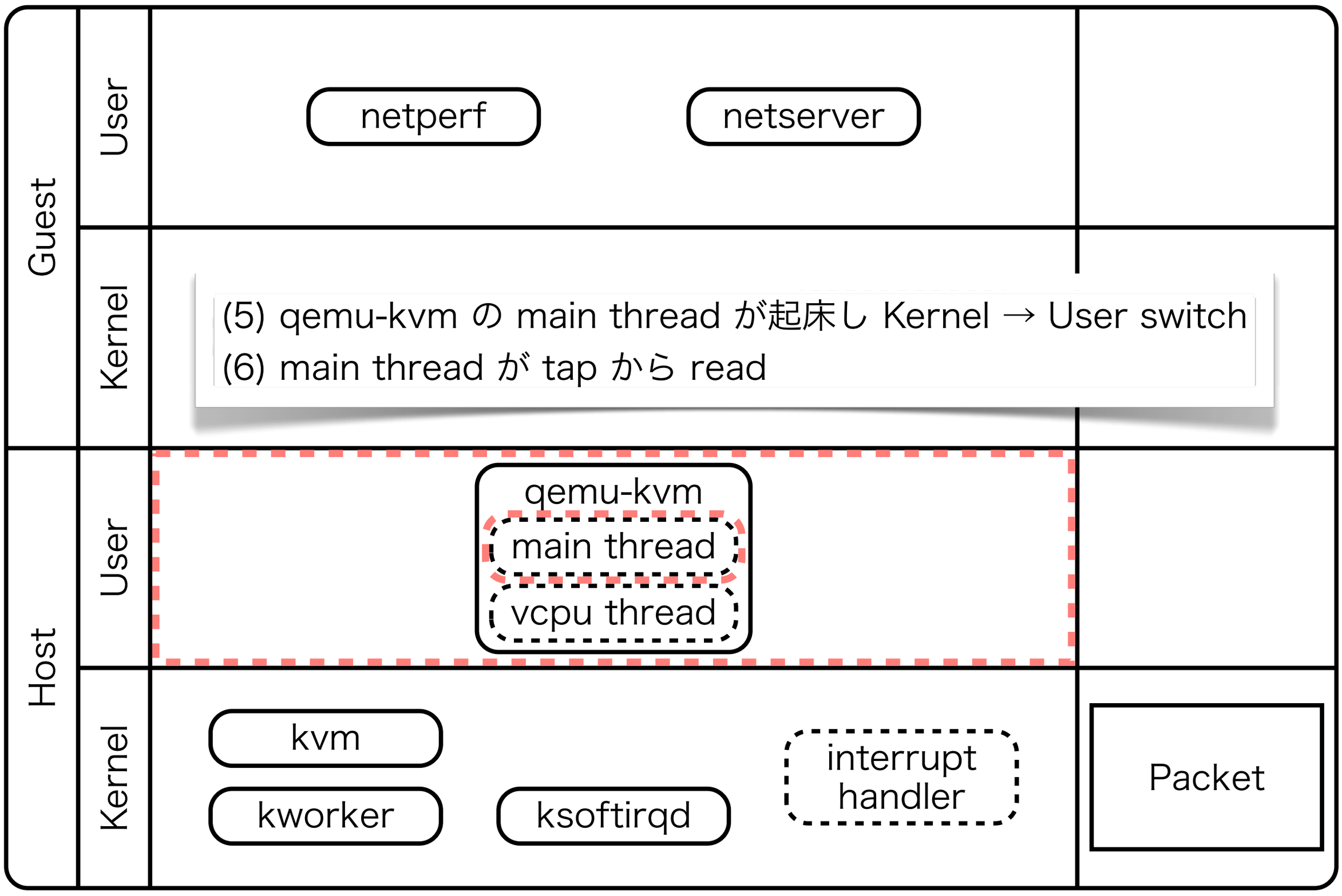
e1000: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Host
User
Kernel
qemu-kvm
(2) NIC から割り込みが入り受信関数 call
kvm
kworker
ksoftirqd
interrupt handler
Packet

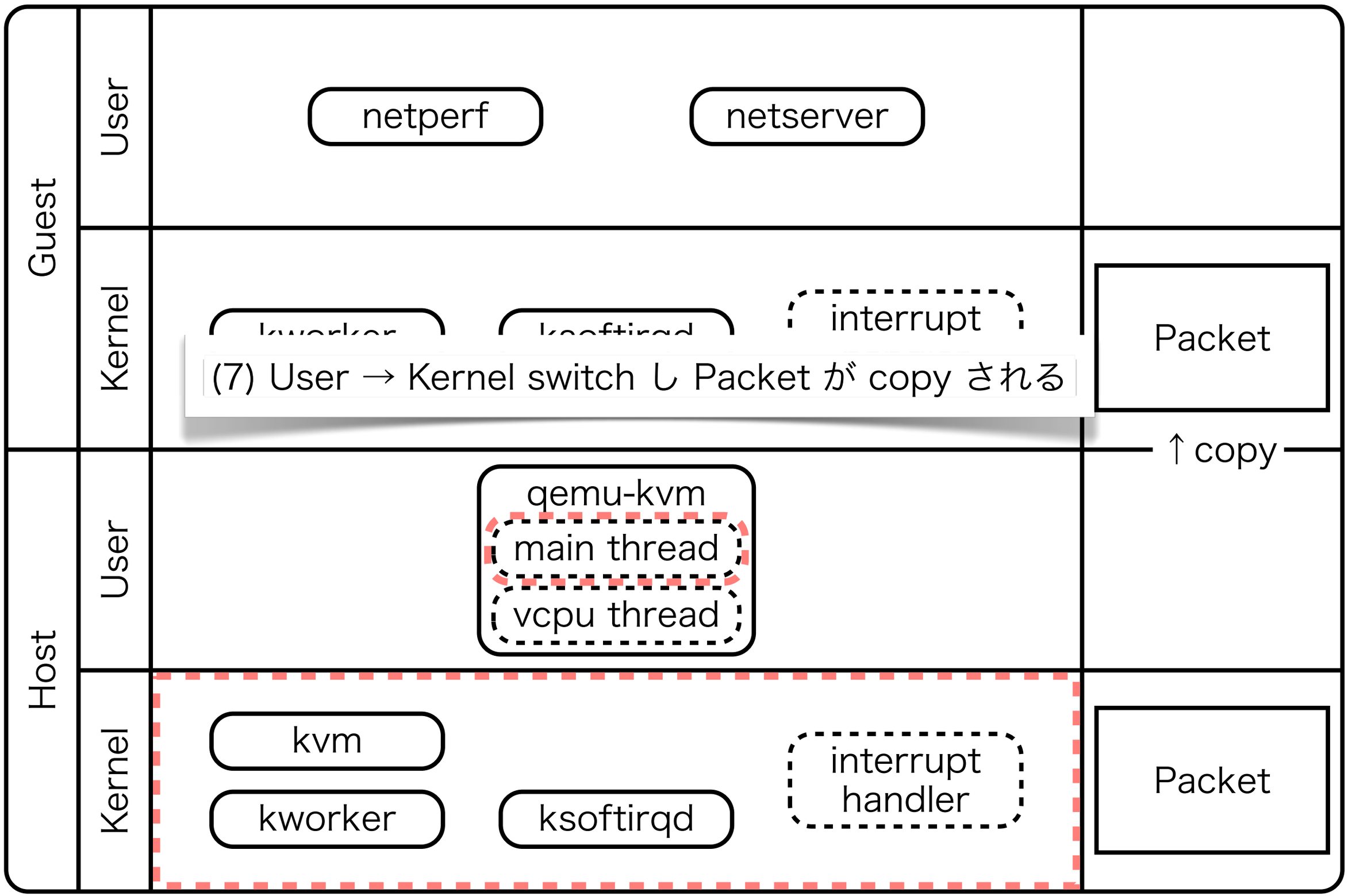
e1000: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Host
User
Kernel
kvm
kworker
ksoftirqd
interrupt handler
Packet
(3) ... この部分は bridge/openvswitch/macvtap で異なる ...
(4) tap device を poll している process を wakeup

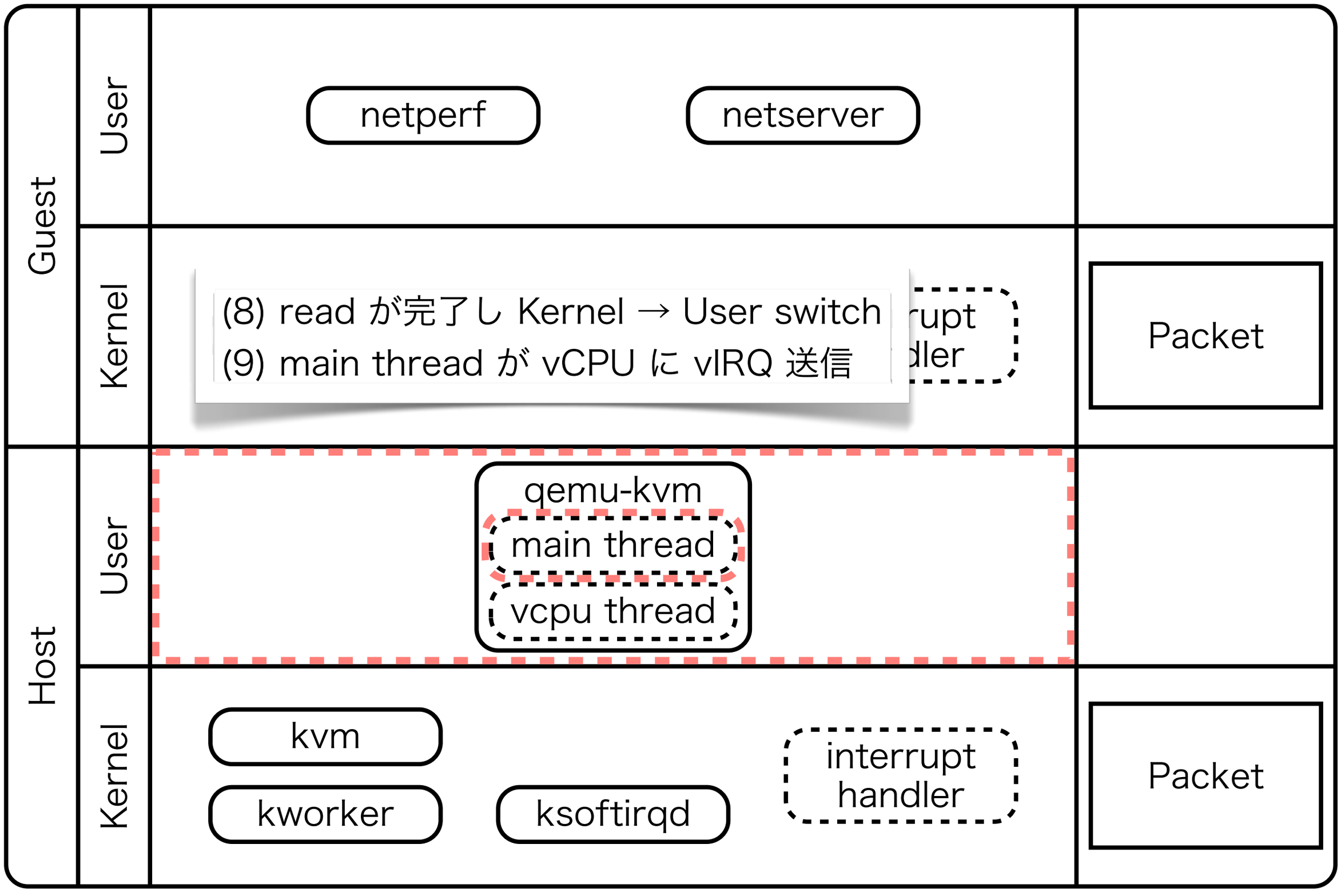
e1000: RX


Guest
User
netperf
netserver
Kernel
(5) qemu-kvm の main thread が起床し Kernel → User switch
(6) main thread が tap から read
Host
User
qemu-kvm
main thread
vcpu thread
Kernel
kvm
kworker
ksoftirqd
interrupt handler
Packet

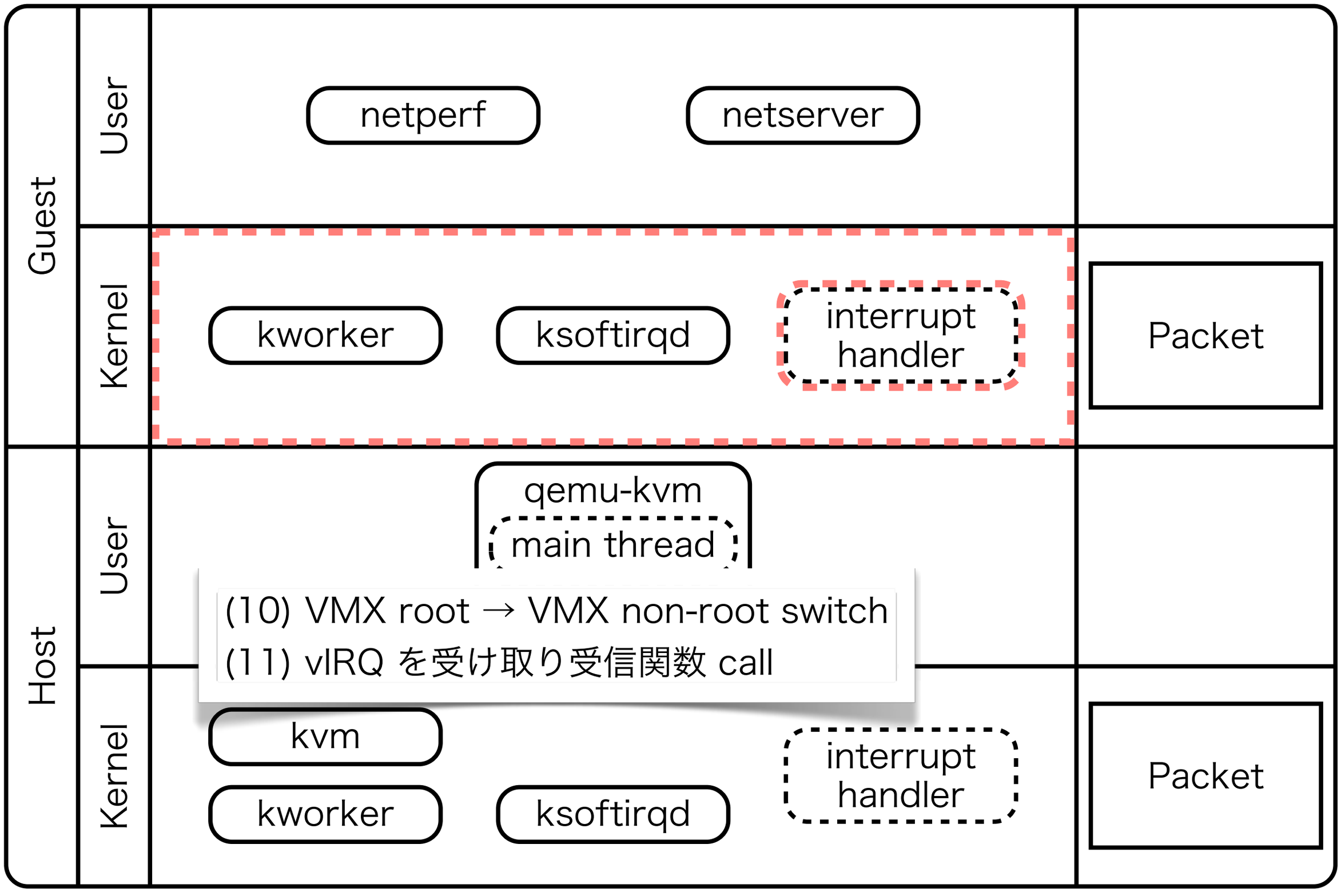
e1000: RX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt
(7) User → Kernel switch し Packet が copy される
Packet
↑ copy
qemu-kvm
main thread
vcpu thread
kvm
kworker
ksoftirqd
interrupt handler
Packet

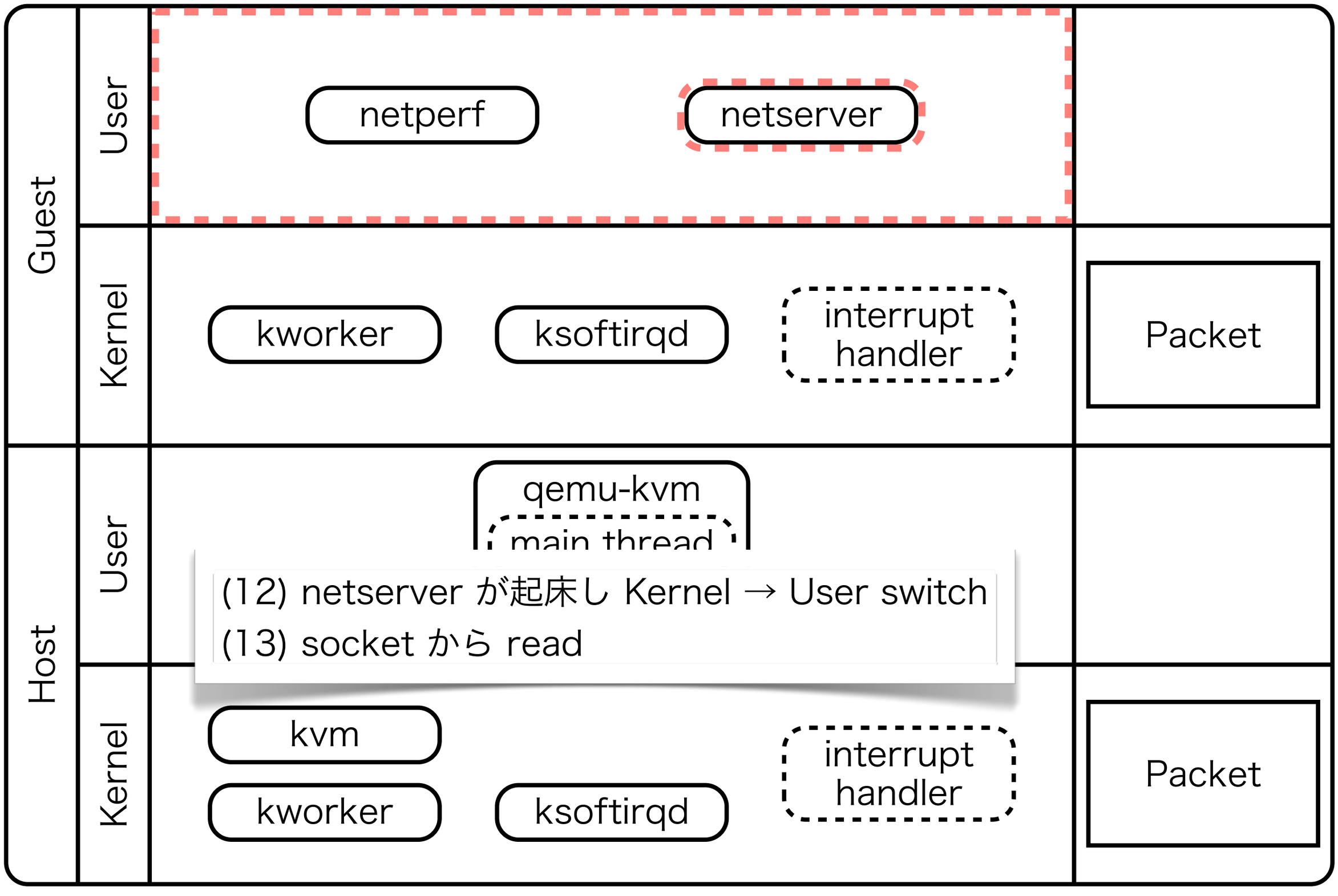
e1000: RX


Guest
User
Kernel
netperf
netserver
(8) read が完了し Kernel → User switch
(9) main thread が vCPU に vIRQ 送信
rupt
dler
Packet
Host
User
qemu-kvm
main thread
vcpu thread
Kernel
kvm
kworker
ksoftirqd
interrupt
handler
Packet

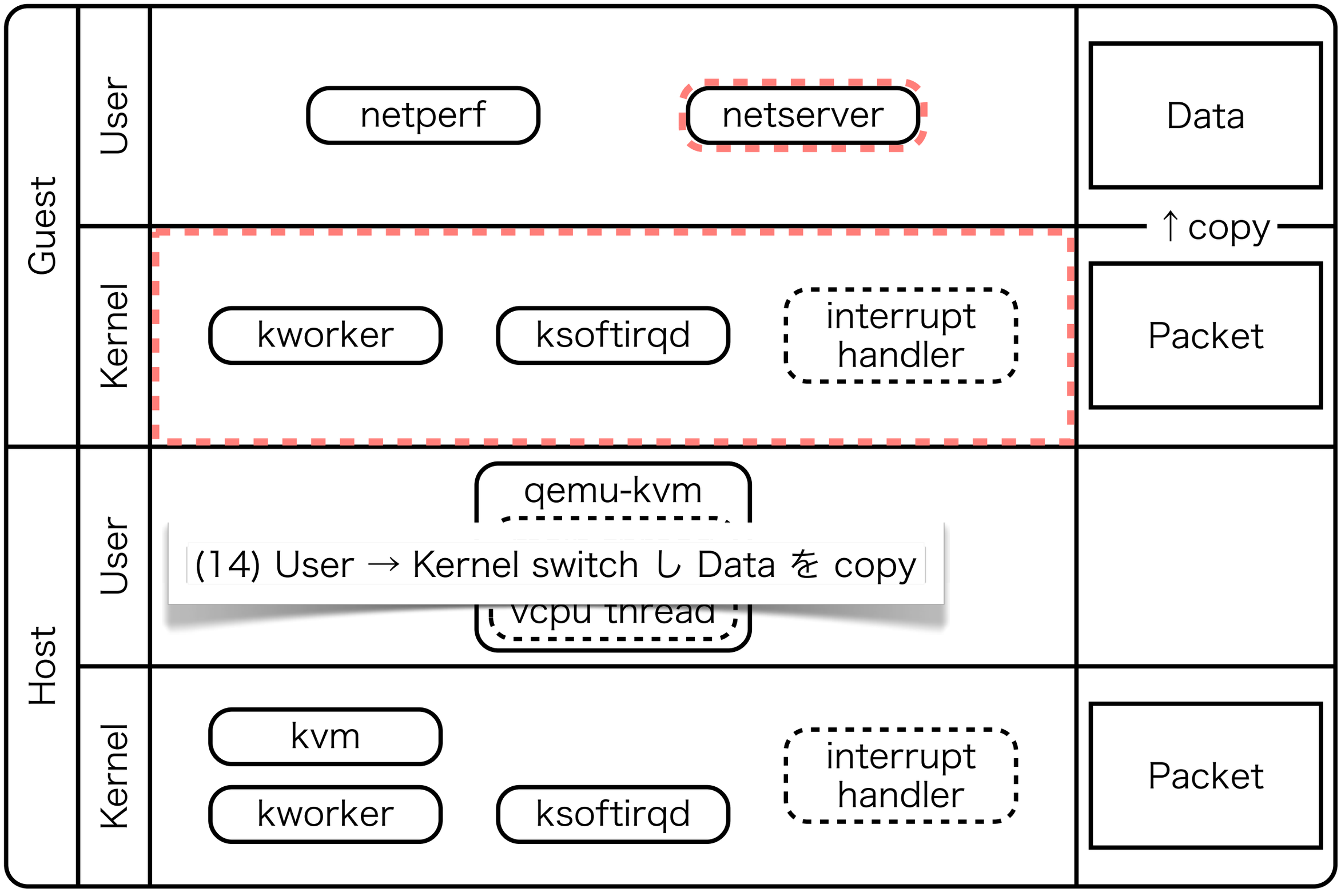
e1000: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Packet
Host
User
Kernel
qemu-kvm
main thread
(10) VMX root → VMX non-root switch
(11) vIRQ を受け取り受信関数 call
kvm
kworker
ksoftirqd
interrupt handler
Packet

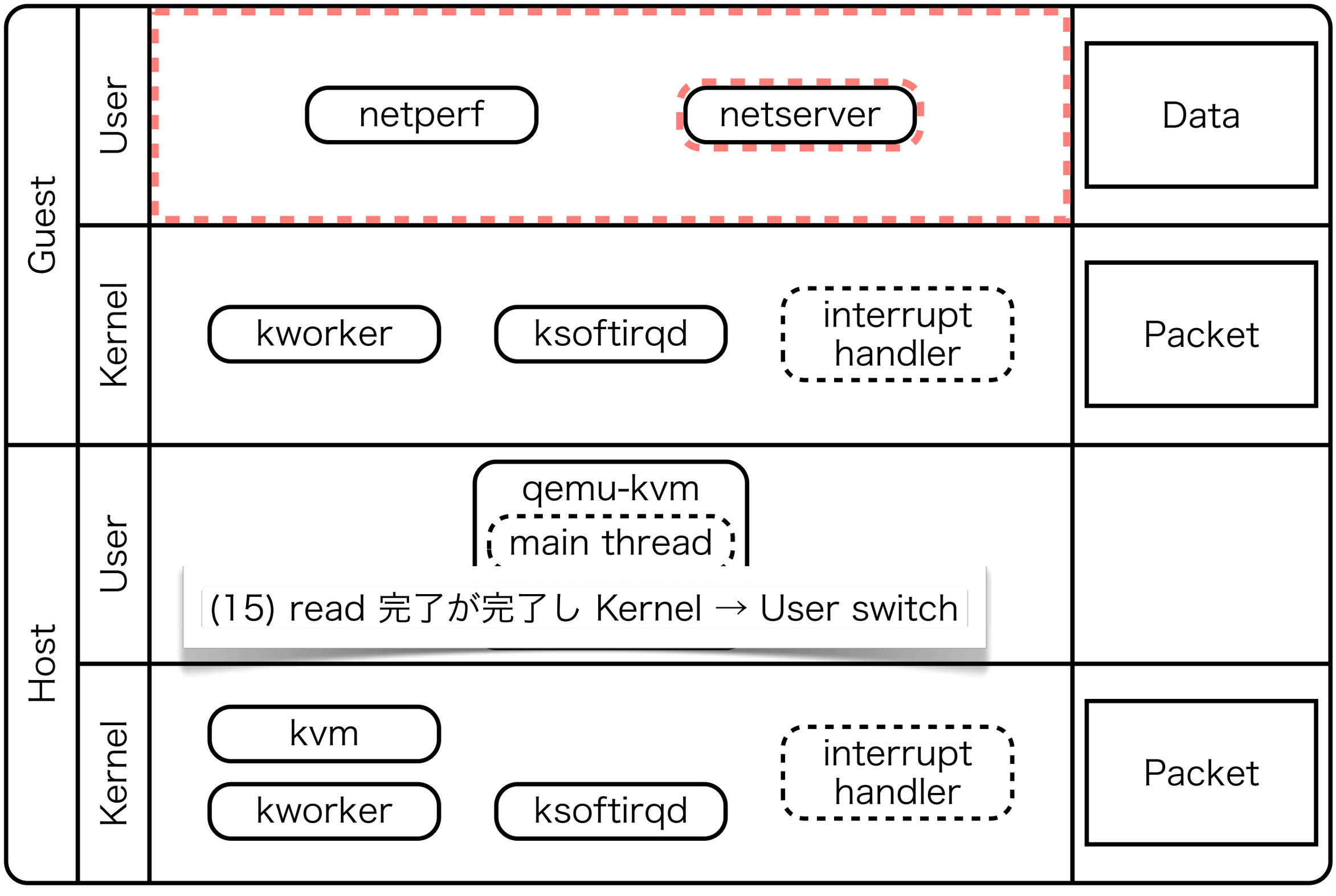
e1000: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Packet
Host
User
Kernel
qemu-kvm
main thread
(12) netserver が起床し Kernel → User switch
(13) socket から read
kvm
kworker
ksoftirqd
interrupt handler
Packet

e1000: RX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
qemu-kvm
vcpu thread
kvm
kworker
ksoftirqd
interrupt handler
Data
↑ copy
Packet
Packet
(14) User → Kernel switch し Data を copy

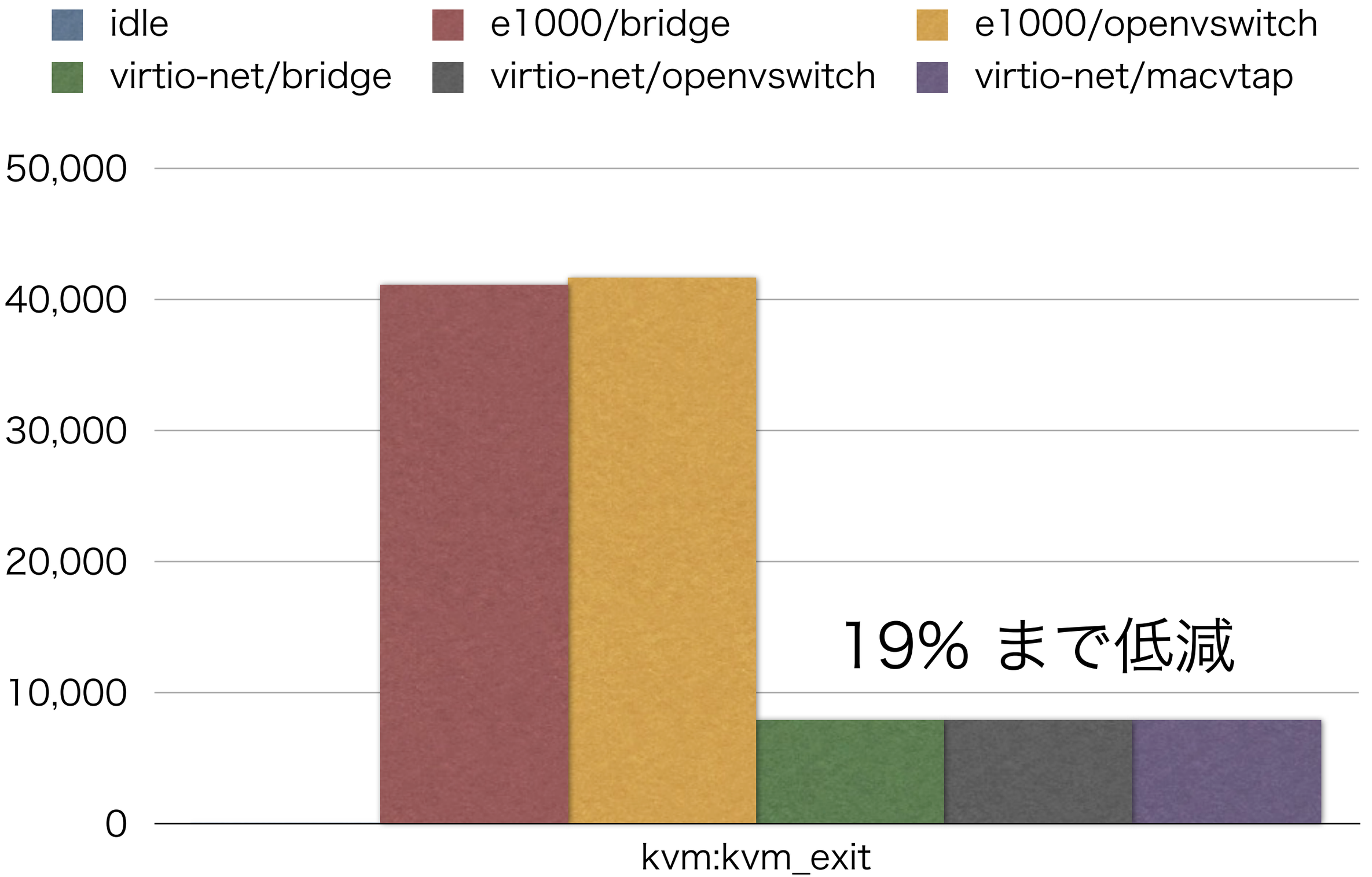
e1000: RX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
Data
kworker
ksoftirqd
interrupt handler
Packet
qemu-kvm
main thread
(15) read 完了が完了し Kernel → User switch
kvm
kworker
ksoftirqd
interrupt handler
Packet

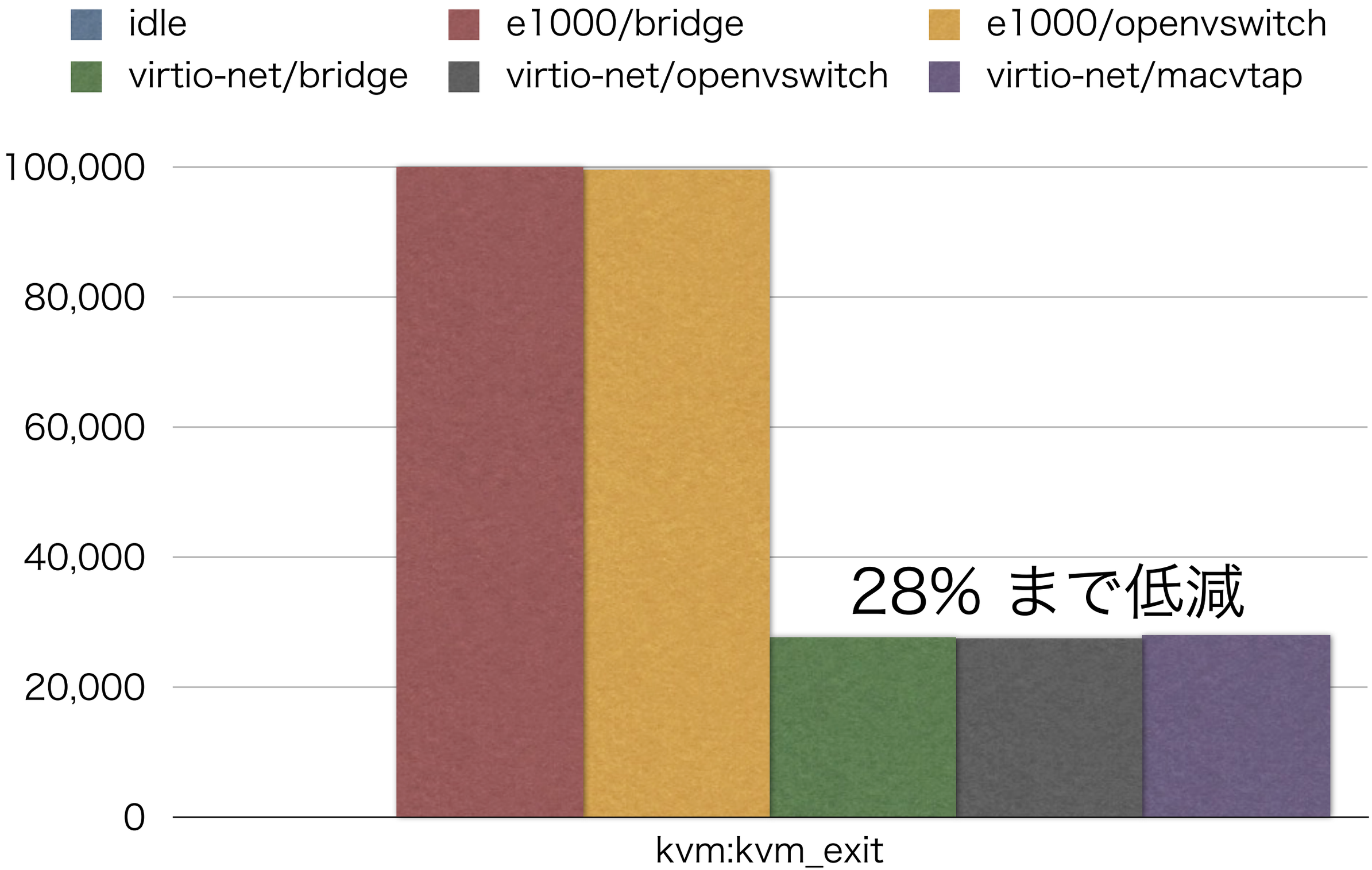
e1000 から virtio-net へ

- e1000 の問題点
  - ☞ register を読み書きするたびに VMX non-root ⇄ VMX root switch が発生する
  - ☞ Hardware を Software で完全に仮想化する為実装が非効率
- virtio-net で解決
  - ☞ 最低限の VMX context switch で実行できるような実装
  - ☞ ふつうに Software で実装する Queue 構造体でデータのやり取りを行う

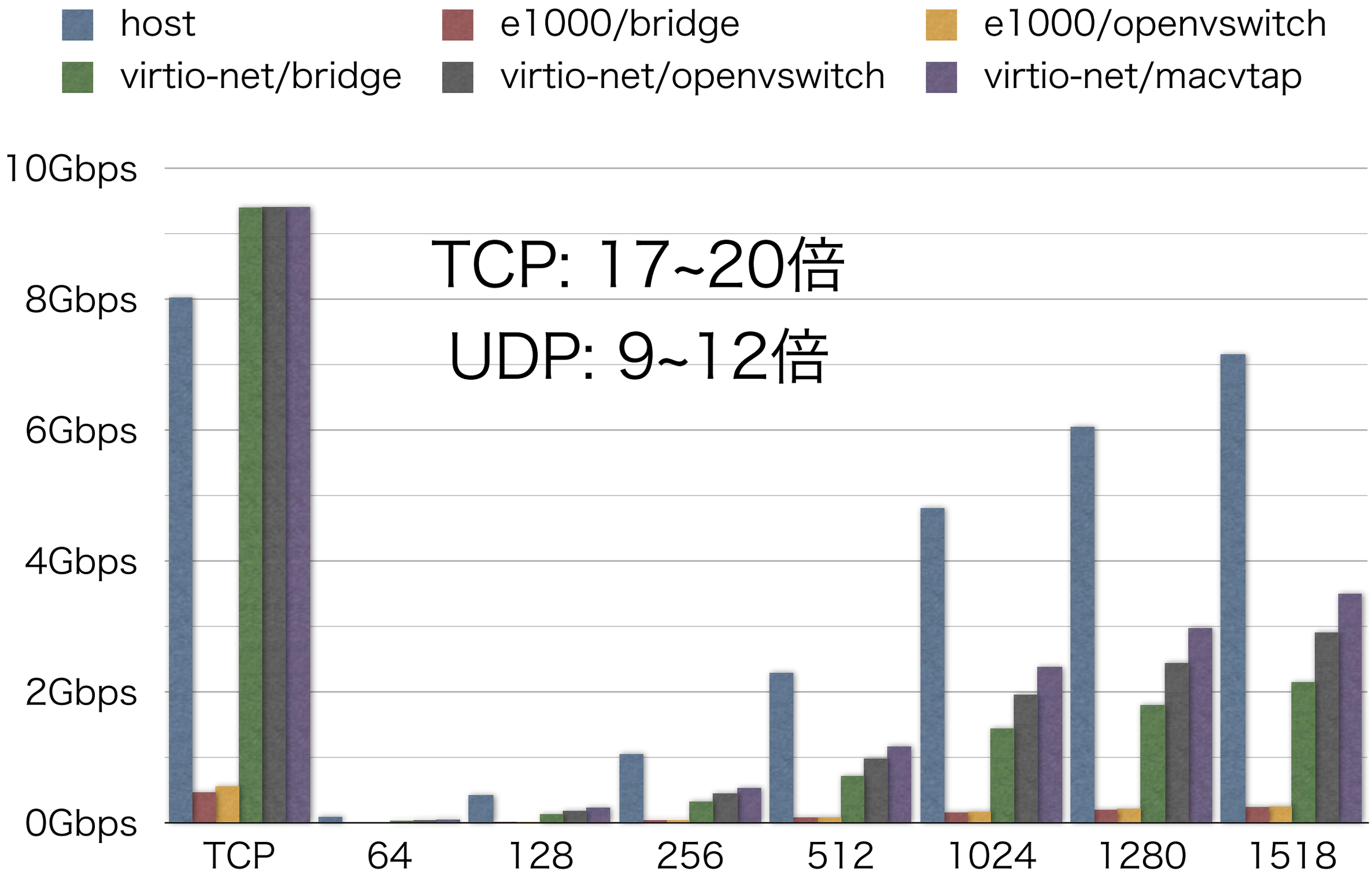
kvm:kvm_exit: Perf Counters: TX



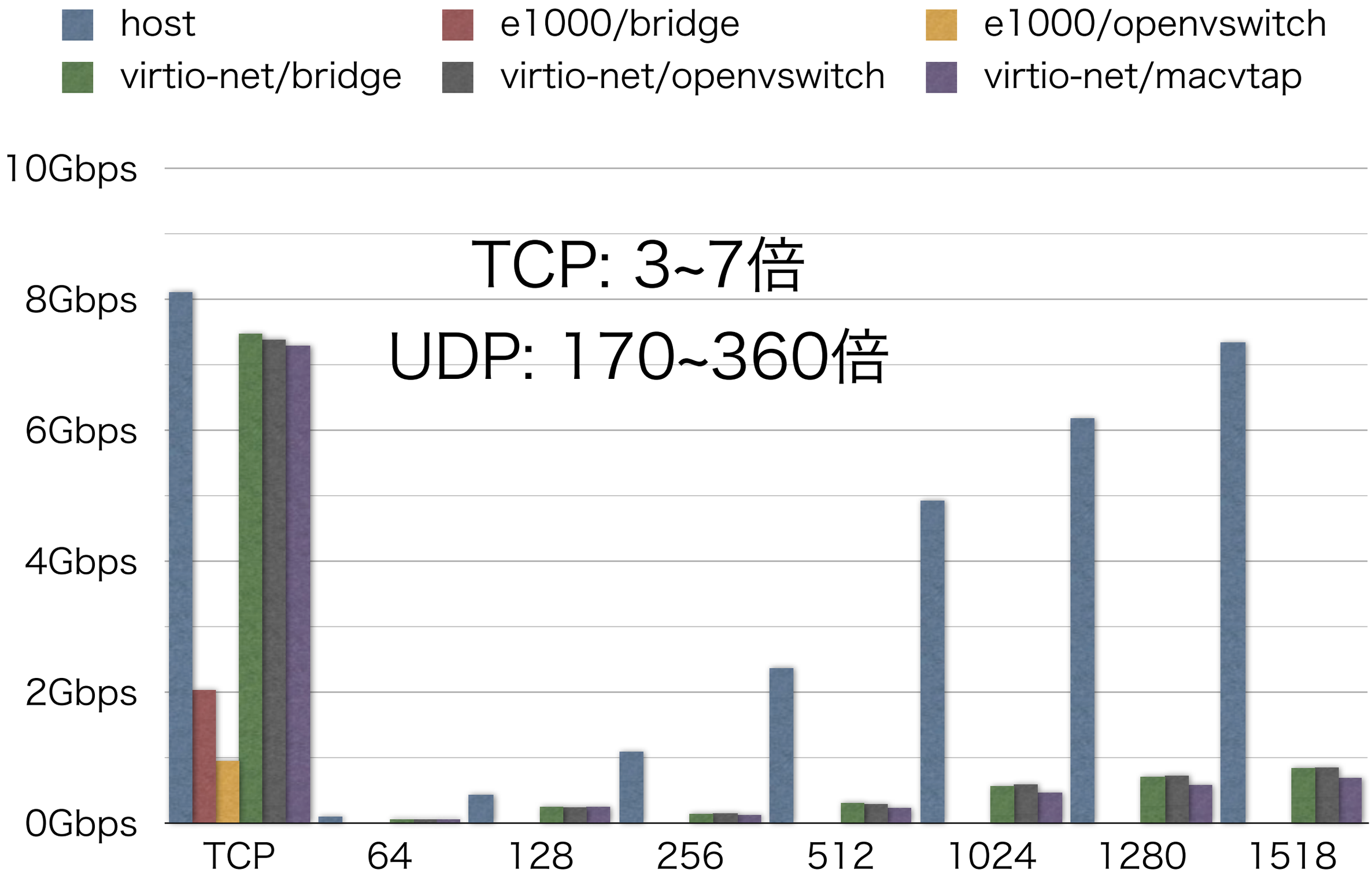
kvm:kvm_exit: Perf Counters: RX



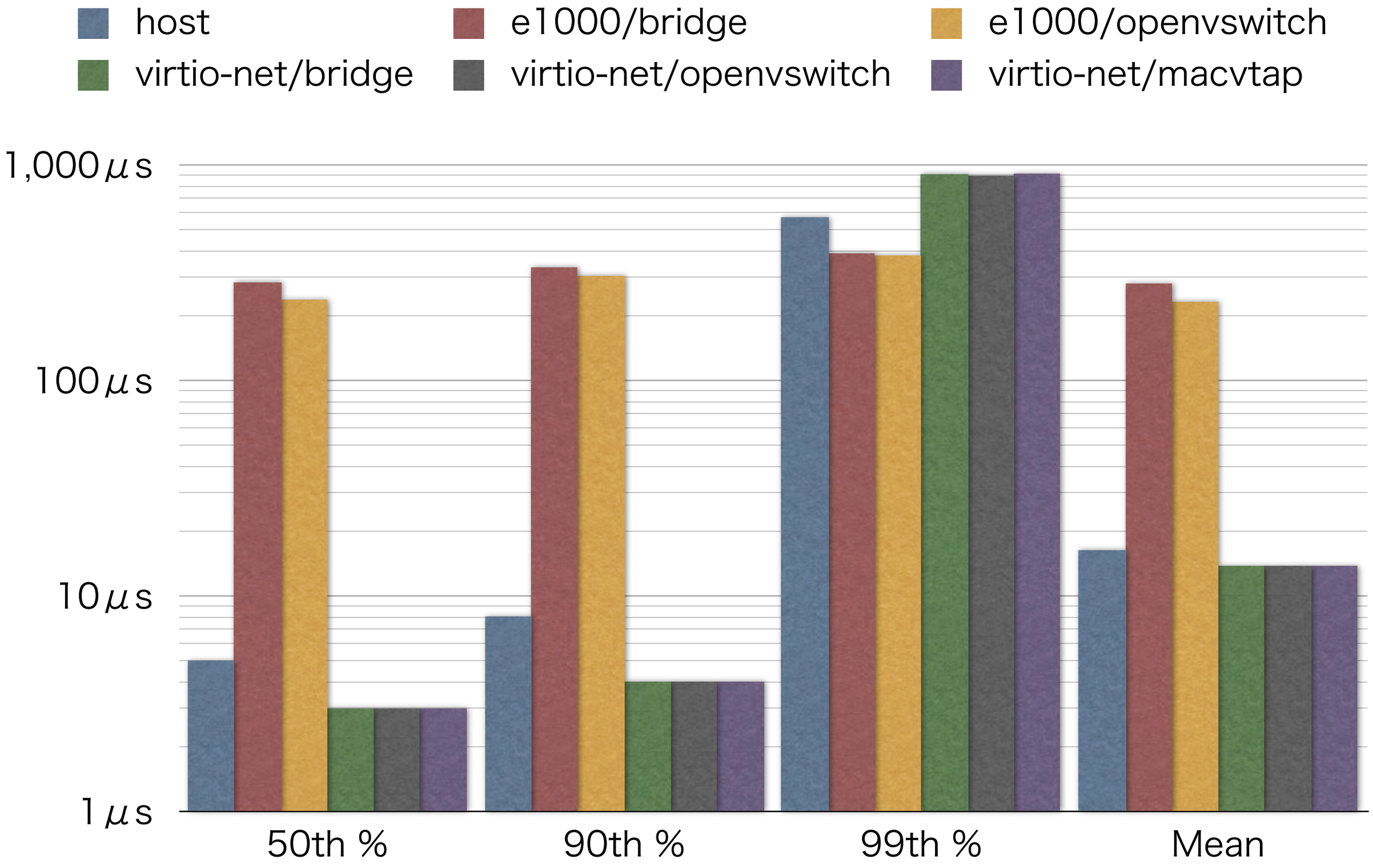
e1000 v.s. virtio-net: Throughput: TX



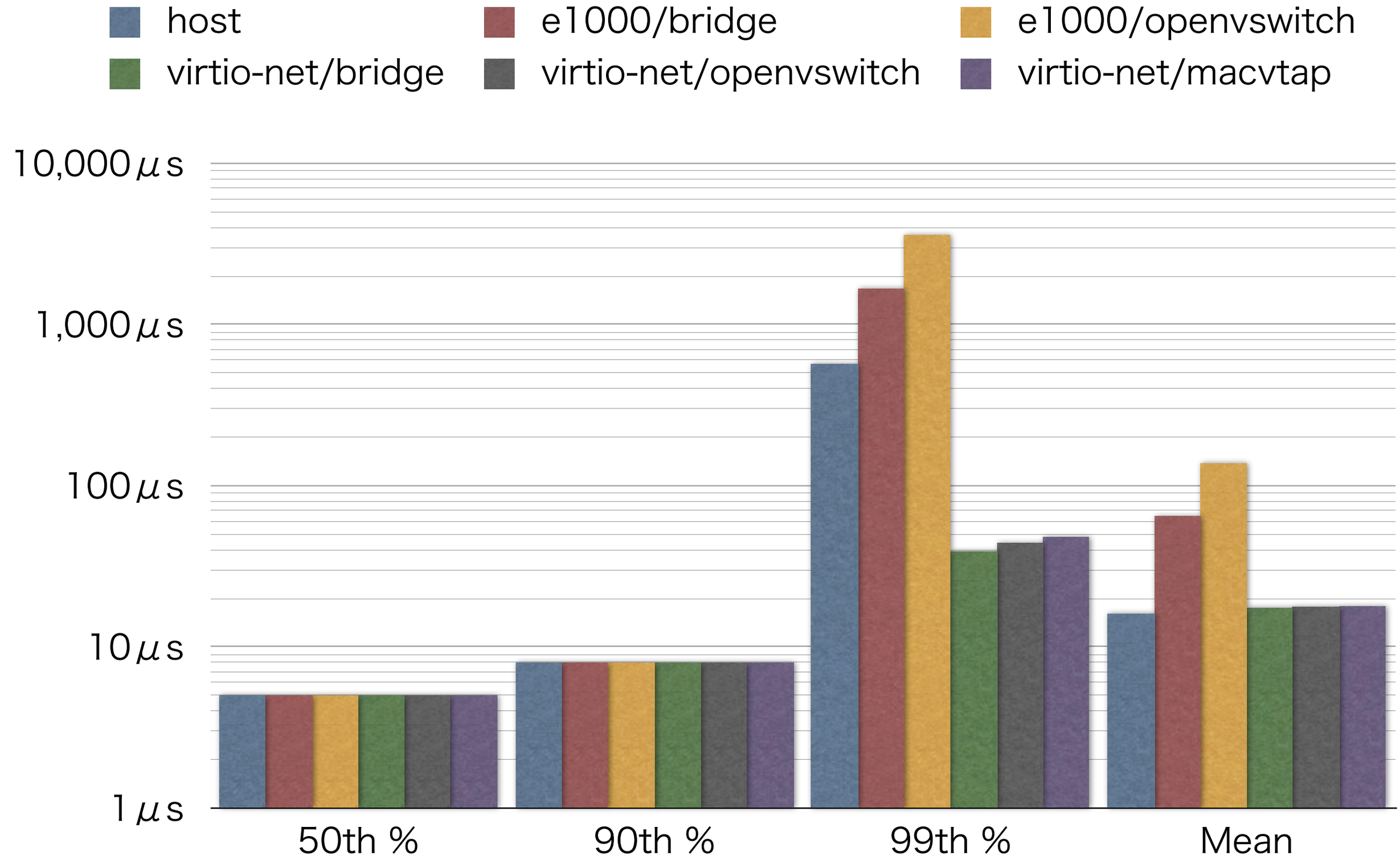
e1000 v.s. virtio-net: Throughput: RX



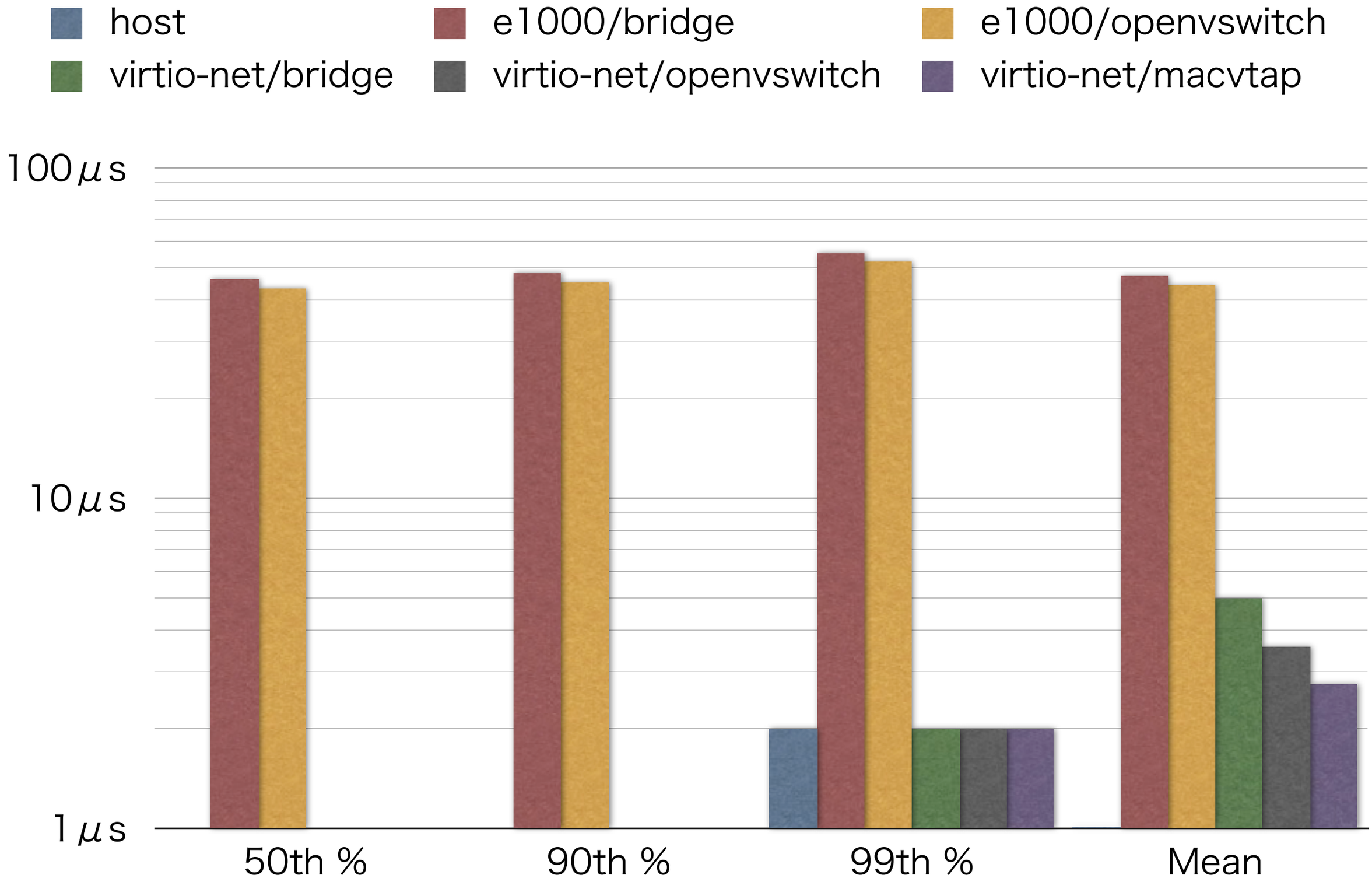
e1000 v.s. virtio-net: Latency: TCP: TX



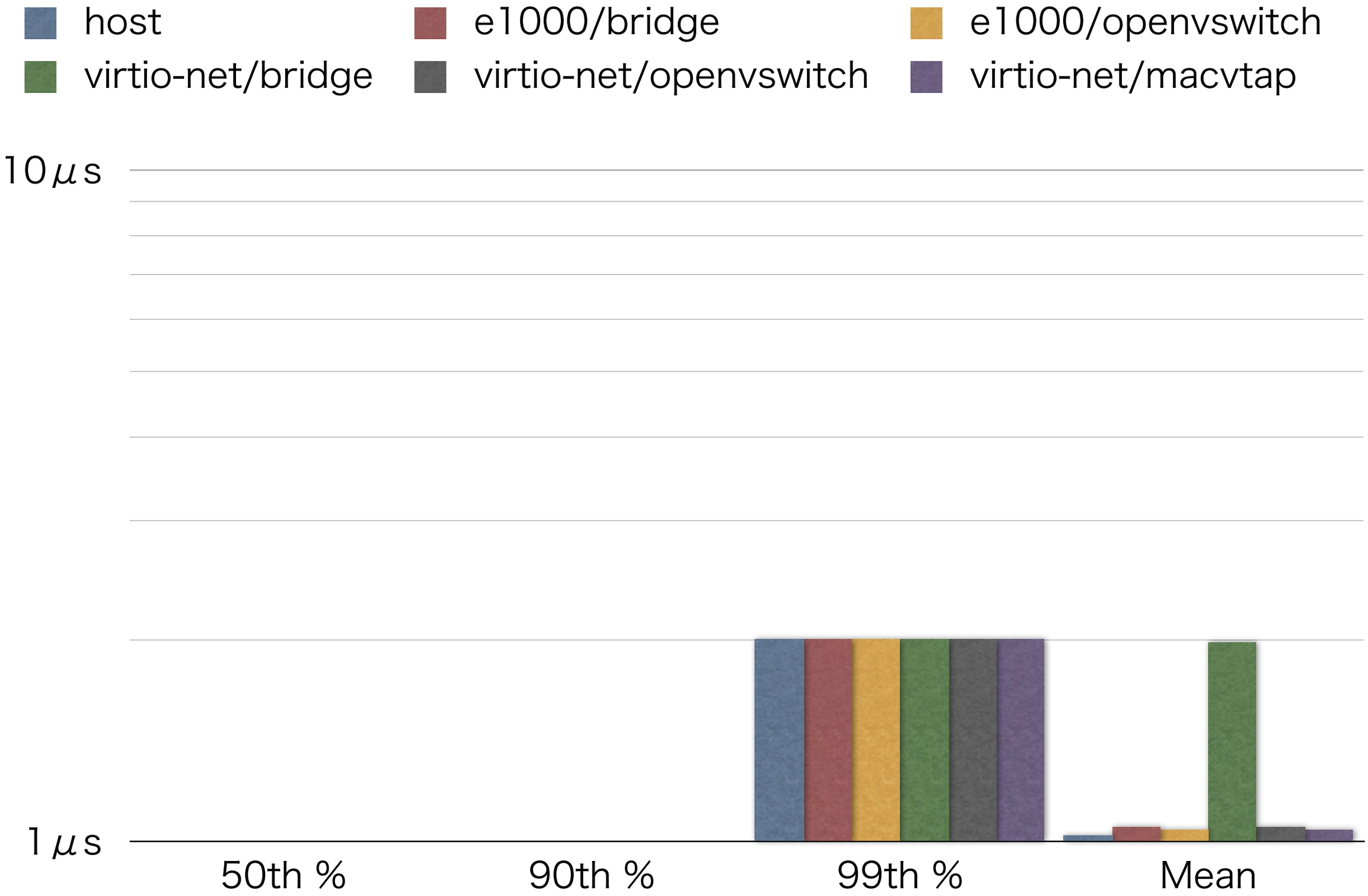
e1000 v.s. virtio-net: Latency: TCP: RX



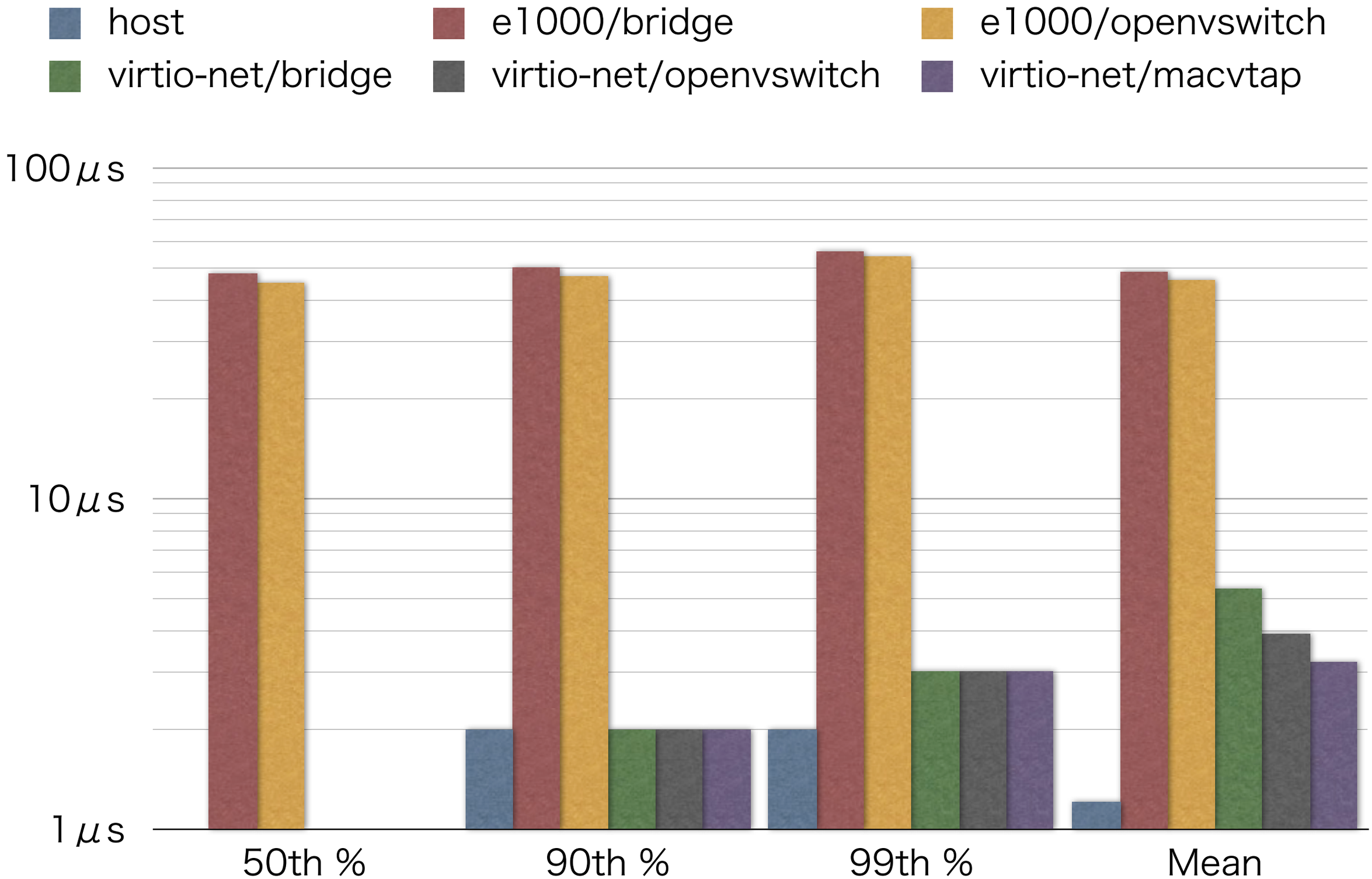
e1000 v.s. virtio-net: Latency: UDP64: TX



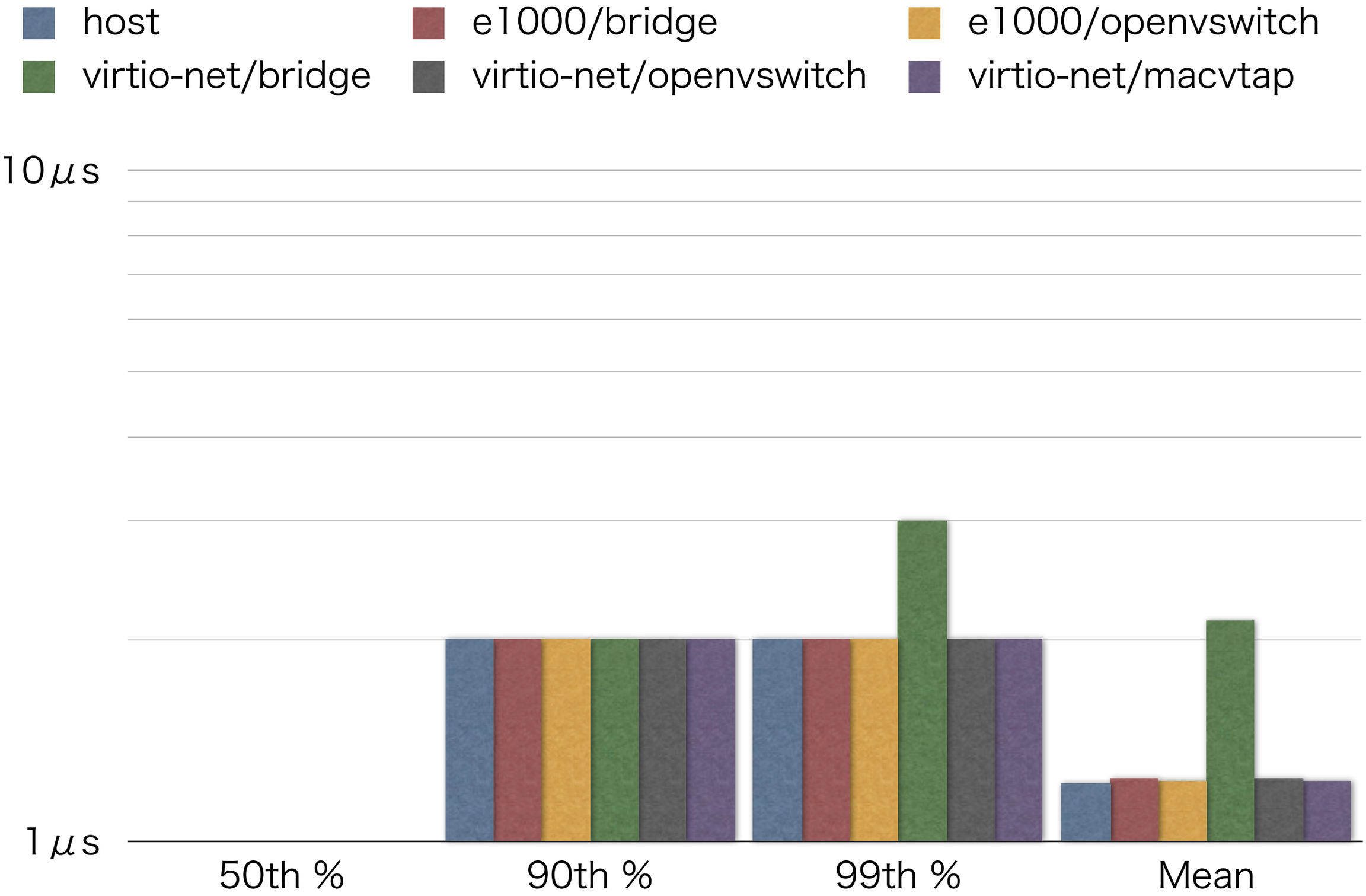
e1000 v.s. virtio-net: Latency: UDP64: RX



e1000 v.s. virtio-net: Latency: UDP1518: TX



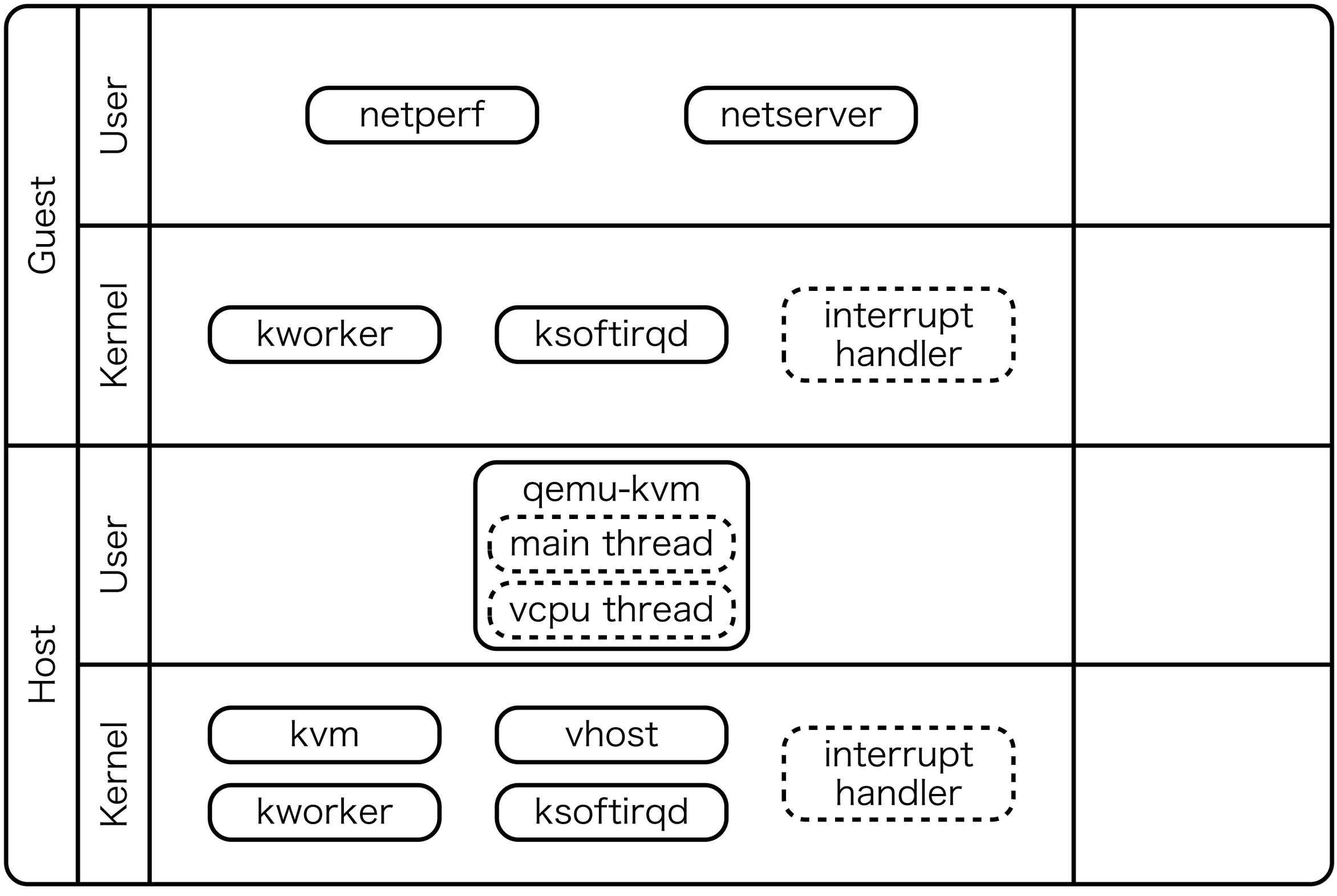
e1000 v.s. virtio-net: Latency: UDP1518: RX



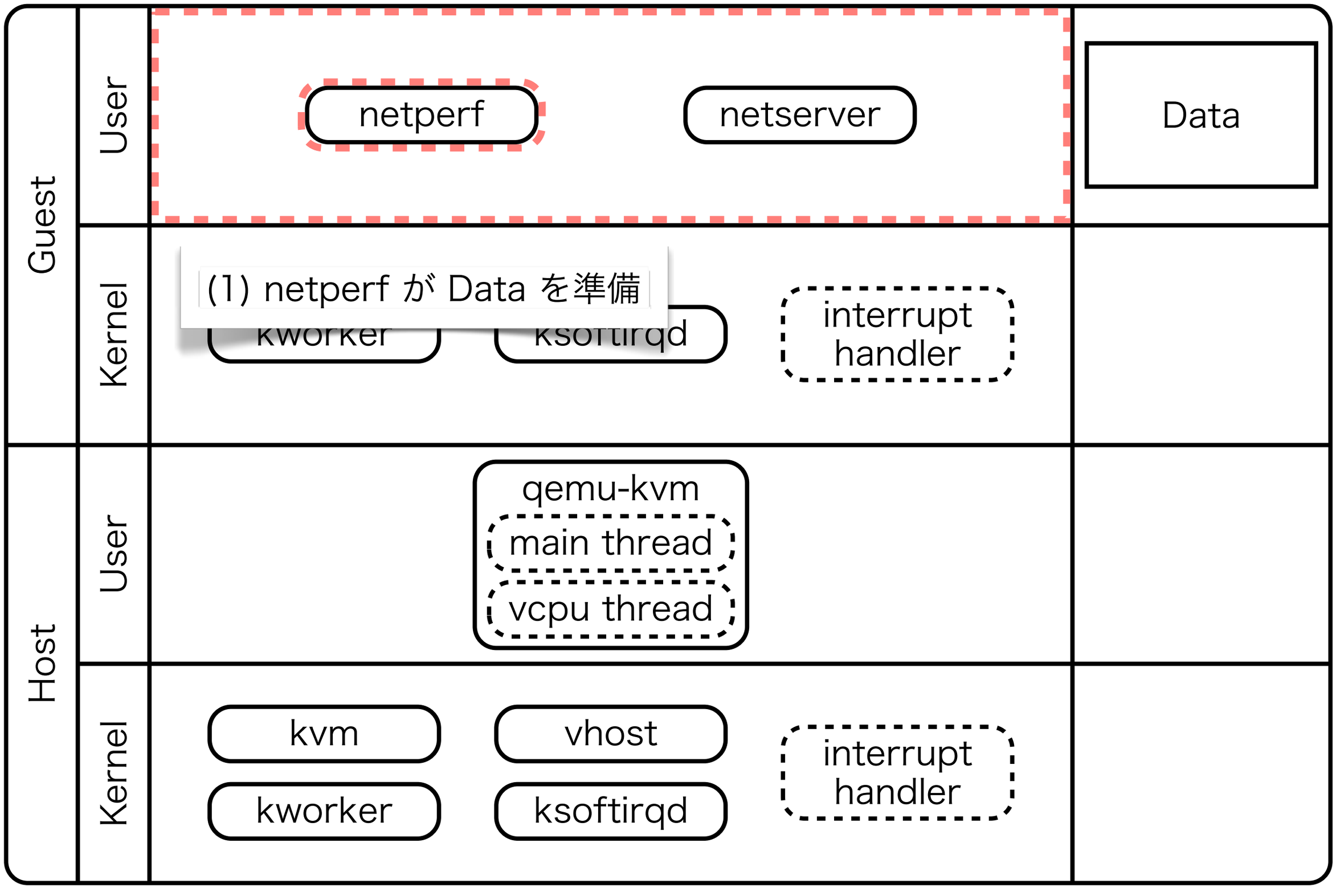
virtio-net から vhost-net へ

- virtio-net の問題点
  - ☞ qemu-kvm が device emulation をする際に User ⇄ Kernel switch が頻繁に発生する
    (例：qemu-kvm が tap から Packet を読む際や書く際)
- vhost-net で解決
  - ☞ kernel thread にこれらの処理をさせて User ⇄ Kernl switch の回数を低減

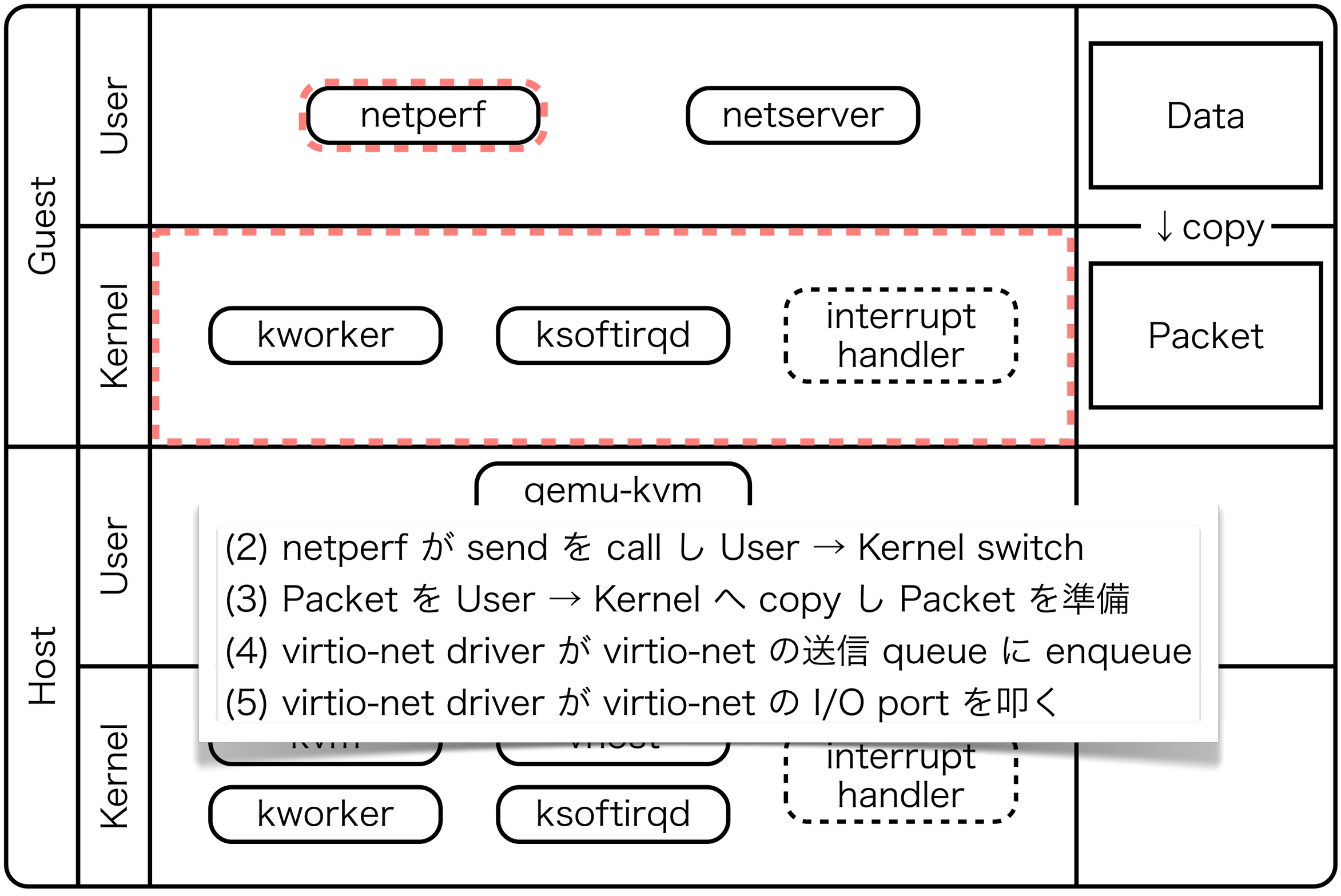
vhost-net: TX


Guest
User
netperf
netserver
Kernel
kworker
ksoftirqd
interrupt handler
Host
User
qemu-kvm
main thread
vcpu thread
Kernel
kvm
vhost
kworker
ksoftirqd
interrupt handler

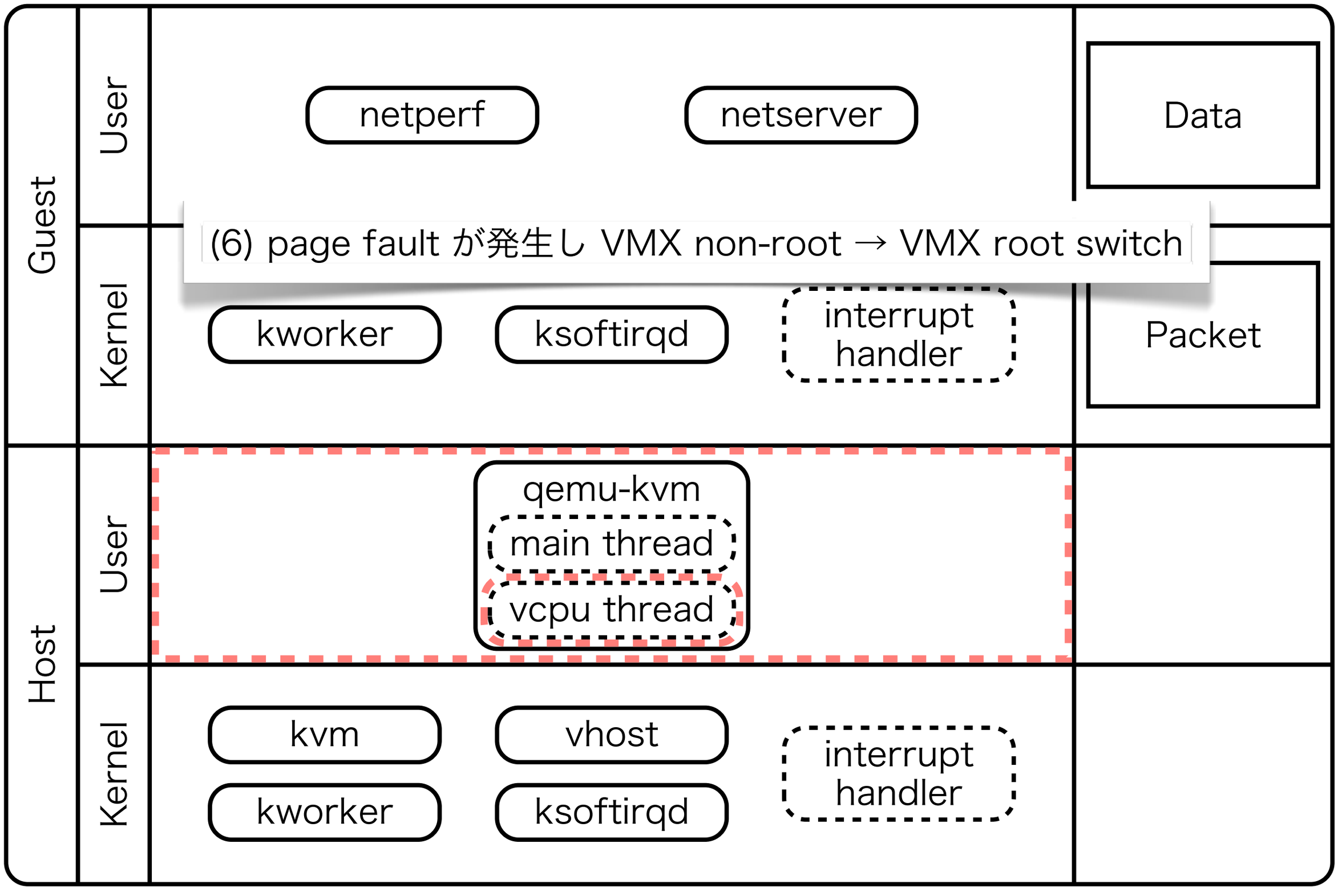
vhost-net: TX


Guest
User
netperf
netserver
Data
Kernel
(1) netperf が Data を準備
kworker
ksoftirqd
interrupt handler
Host
User
qemu-kvm
main thread
vcpu thread
Kernel
kvm
vhost
kworker
ksoftirqd
interrupt handler

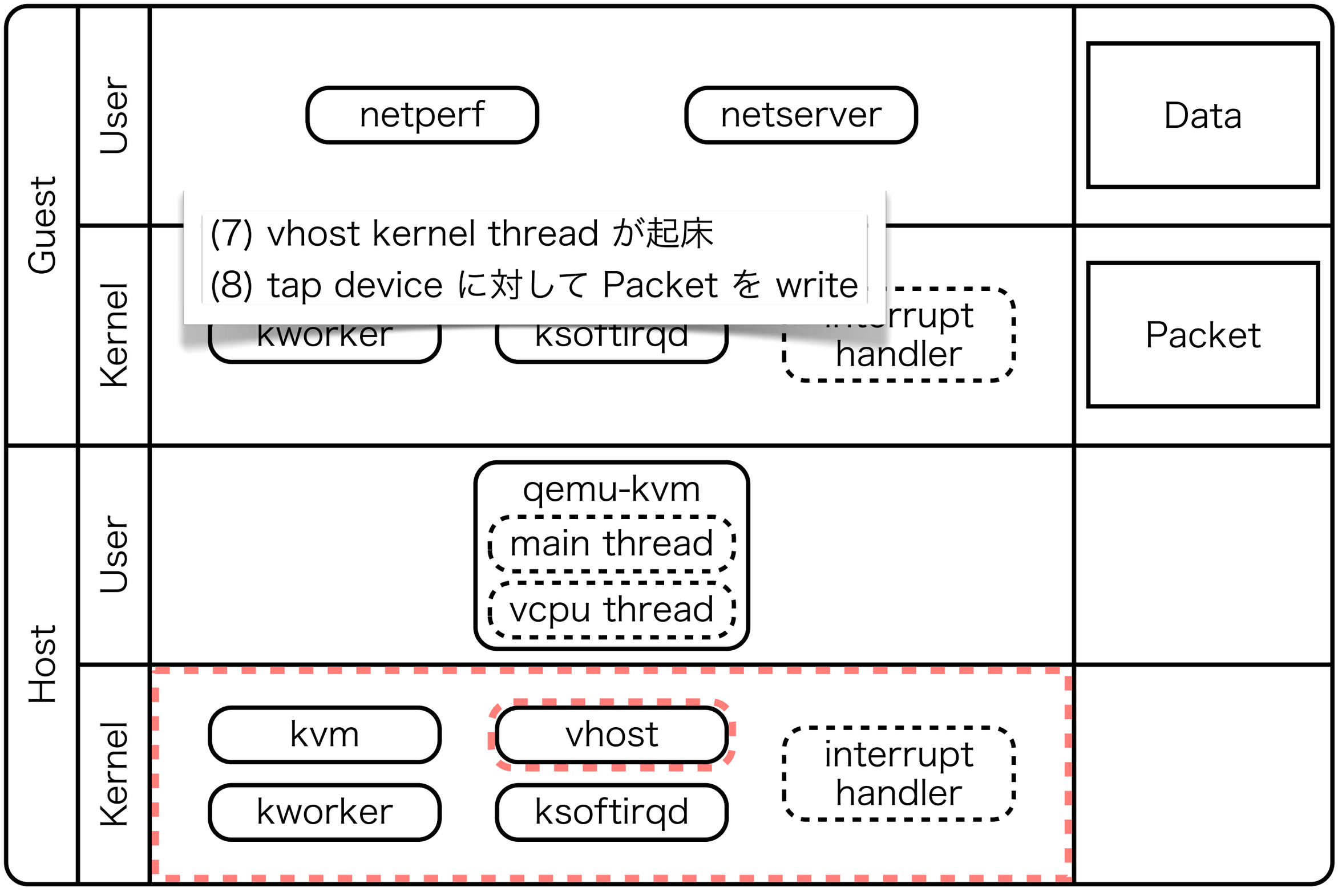
vhost-net: TX


Guest
User
Kernel
netperf
netserver
Data
↓ copy
kworker
ksoftirqd
interrupt handler
Packet
Host
User
Kernel
qemu-kvm
(2) netperf が send を call し User → Kernel switch
(3) Packet を User → Kernel へ copy し Packet を準備
(4) virtio-net driver が virtio-net の送信 queue に enqueue
(5) virtio-net driver が virtio-net の I/O port を叩く
kworker
ksoftirqd
interrupt handler

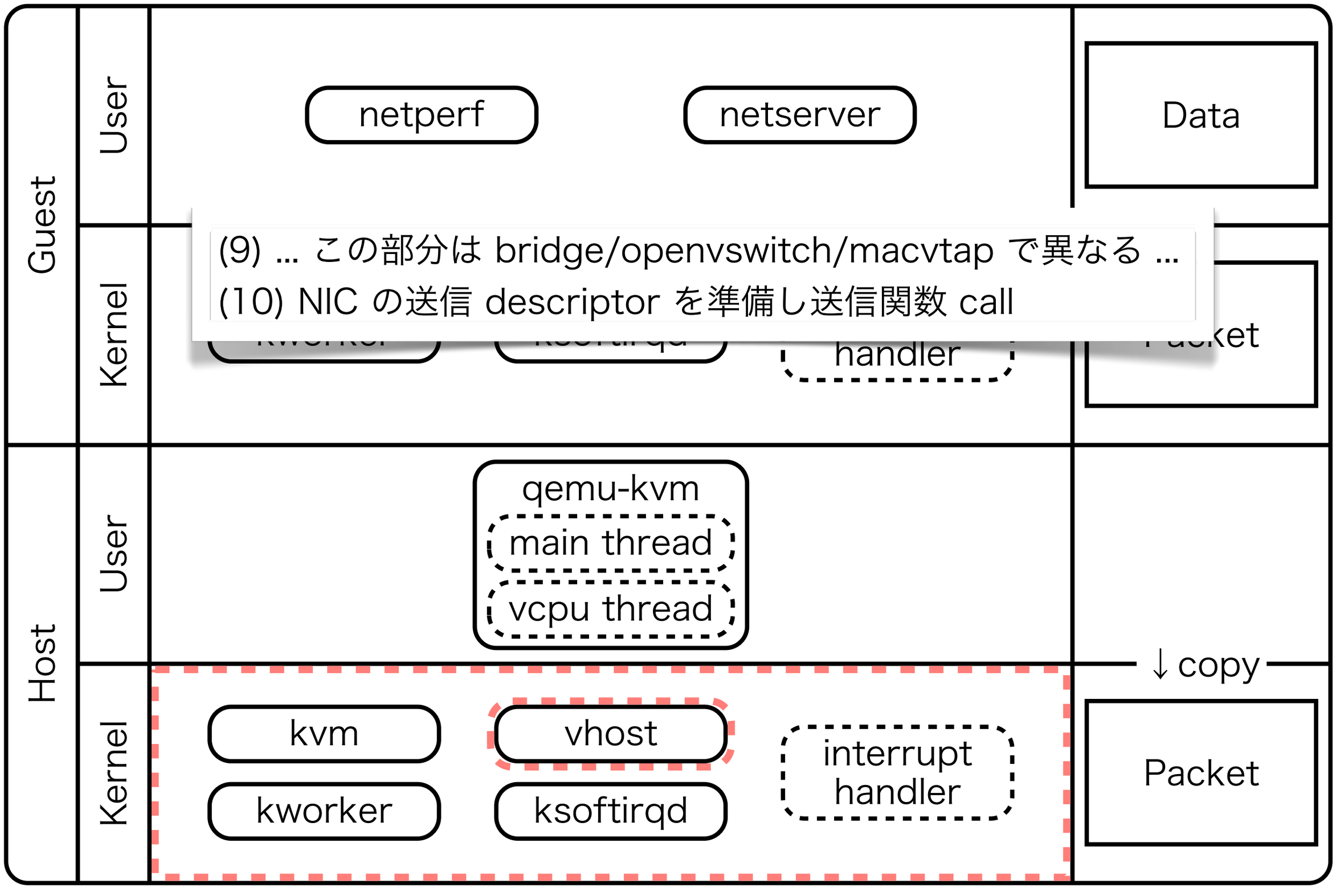
vhost-net: TX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
Data
(6) page fault が発生し VMX non-root → VMX root switch
kworker
ksoftirqd
interrupt handler
Packet
qemu-kvm
main thread
vcpu thread
kvm
vhost
kworker
ksoftirqd
interrupt handler

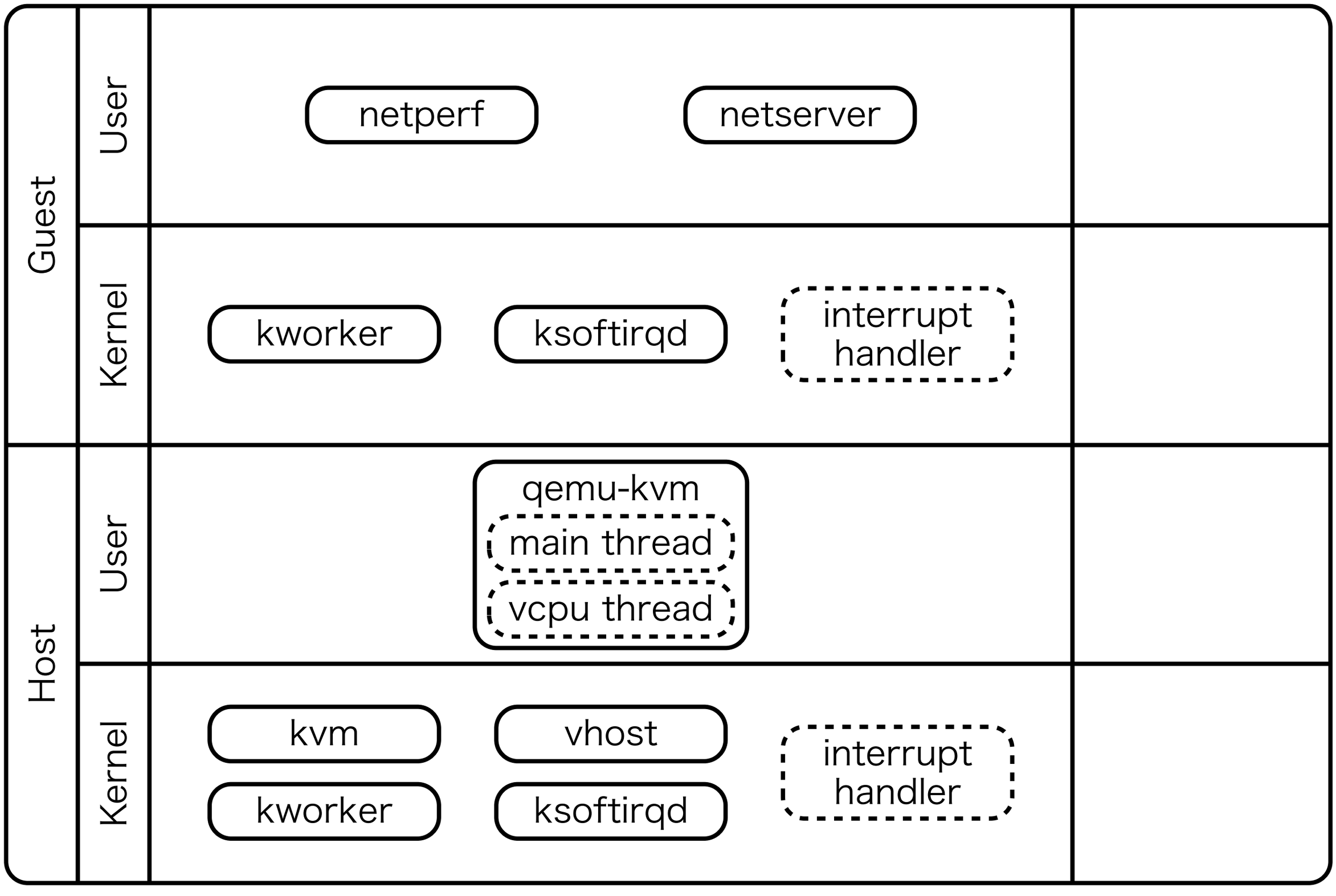
vhost-net: TX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
Data
(7) vhost kernel thread が起床
(8) tap device に対して Packet を write
kworker
ksoftirqd
interrupt handler
Packet
qemu-kvm
main thread
vcpu thread
kvm
vhost
kworker
ksoftirqd
interrupt handler

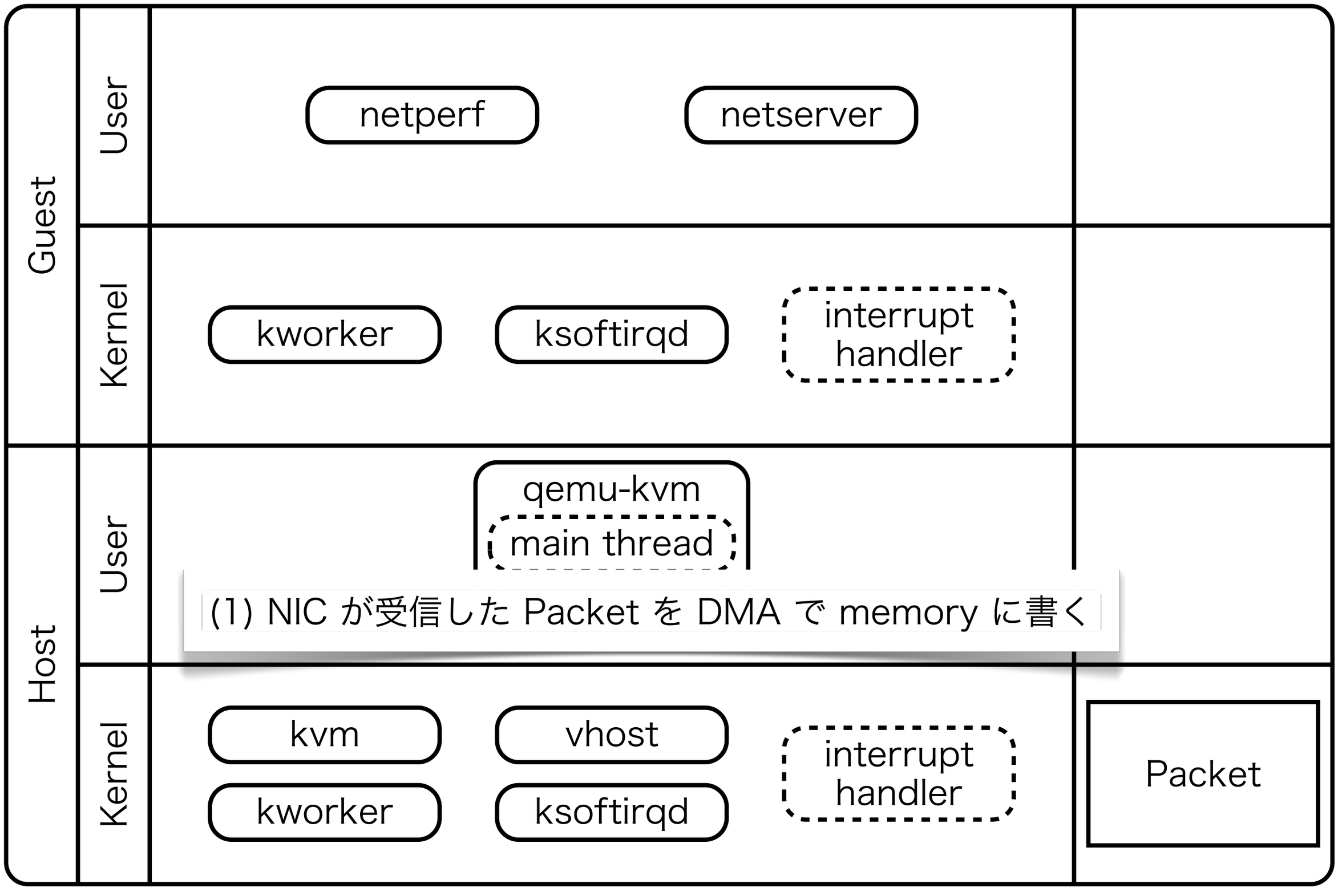
vhost-net: TX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
Data
(9) ... この部分は bridge/openvswitch/macvtap で異なる ...
(10) NIC の送信 descriptor を準備し送信関数 call
handler
qemu-kvm
main thread
vcpu thread
kvm
kworker
vhost
ksoftirqd
interrupt
handler
↓ copy
Packet

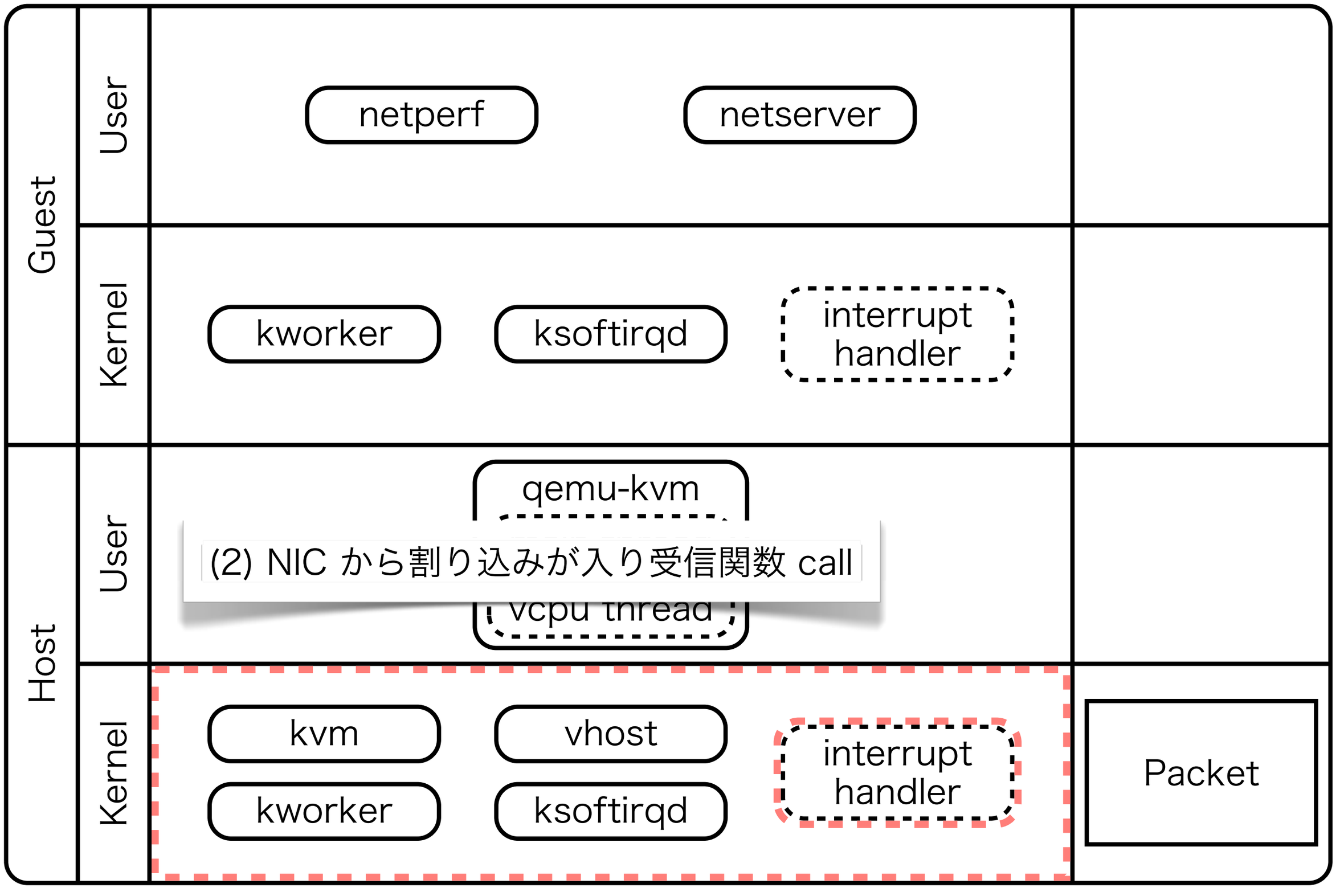
vhost-net: RX


Guest
User
netperf
netserver
Kernel
kworker
ksoftirqd
interrupt handler
Host
User
qemu-kvm
main thread
vcpu thread
Kernel
kvm
vhost
kworker
ksoftirqd
interrupt handler

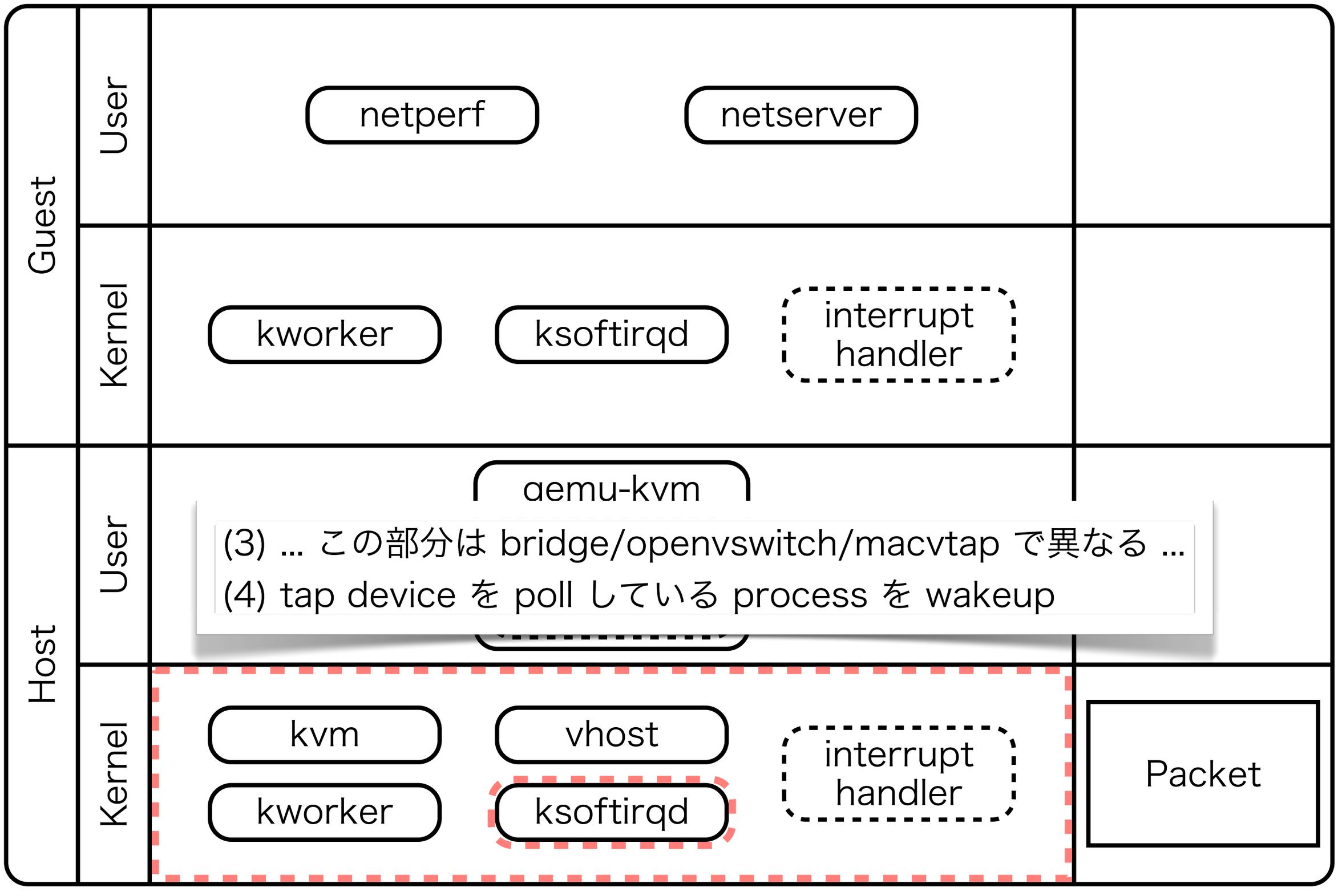
vhost-net: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Host
User
Kernel
qemu-kvm
main thread
(1) NIC が受信した Packet を DMA で memory に書く
kvm
vhost
kworker
ksoftirqd
interrupt handler
Packet

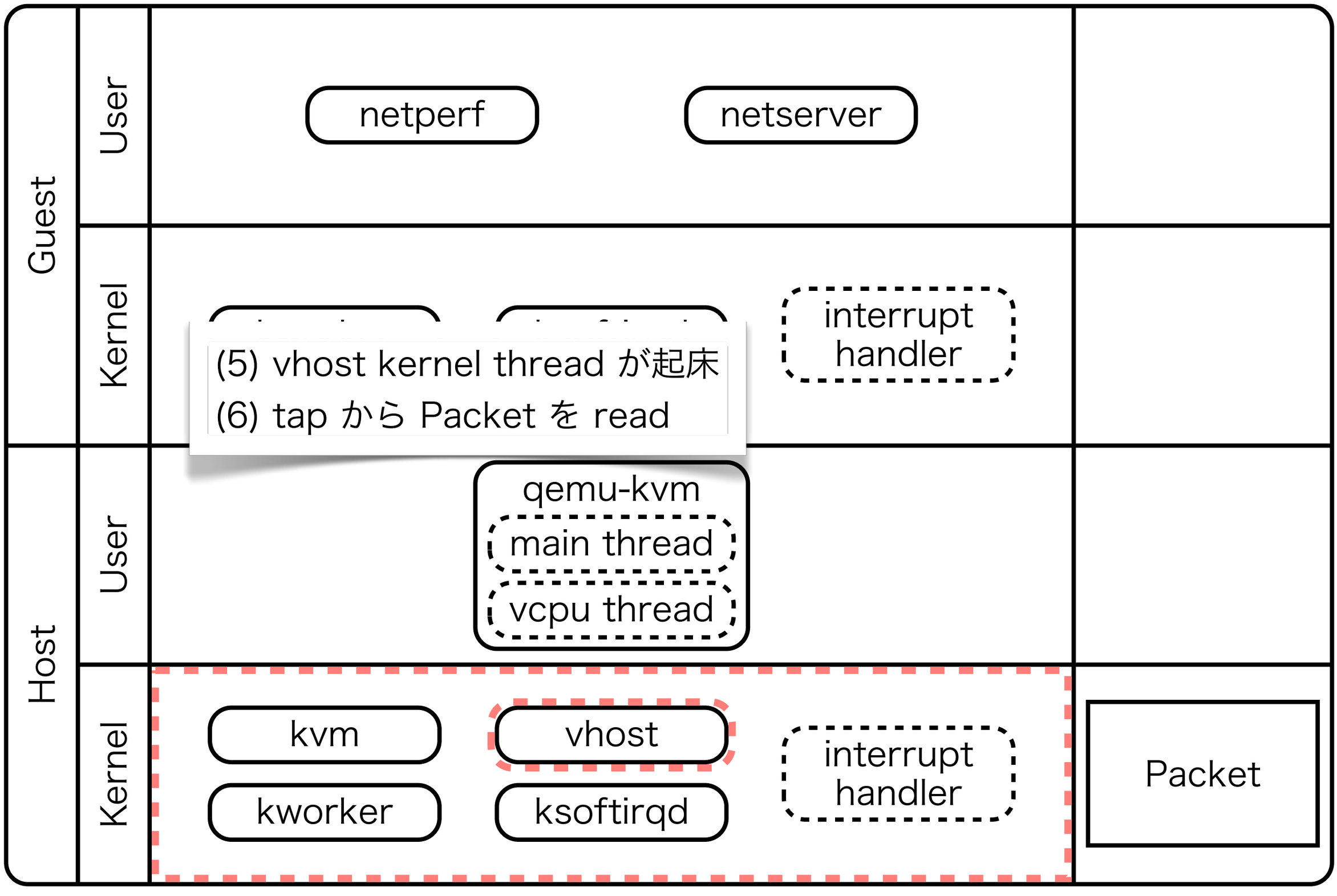
vhost-net: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Host
User
Kernel
qemu-kvm
vcpu thread
(2) NIC から割り込みが入り受信関数 call
kvm
vhost
kworker
ksoftirqd
interrupt handler
Packet

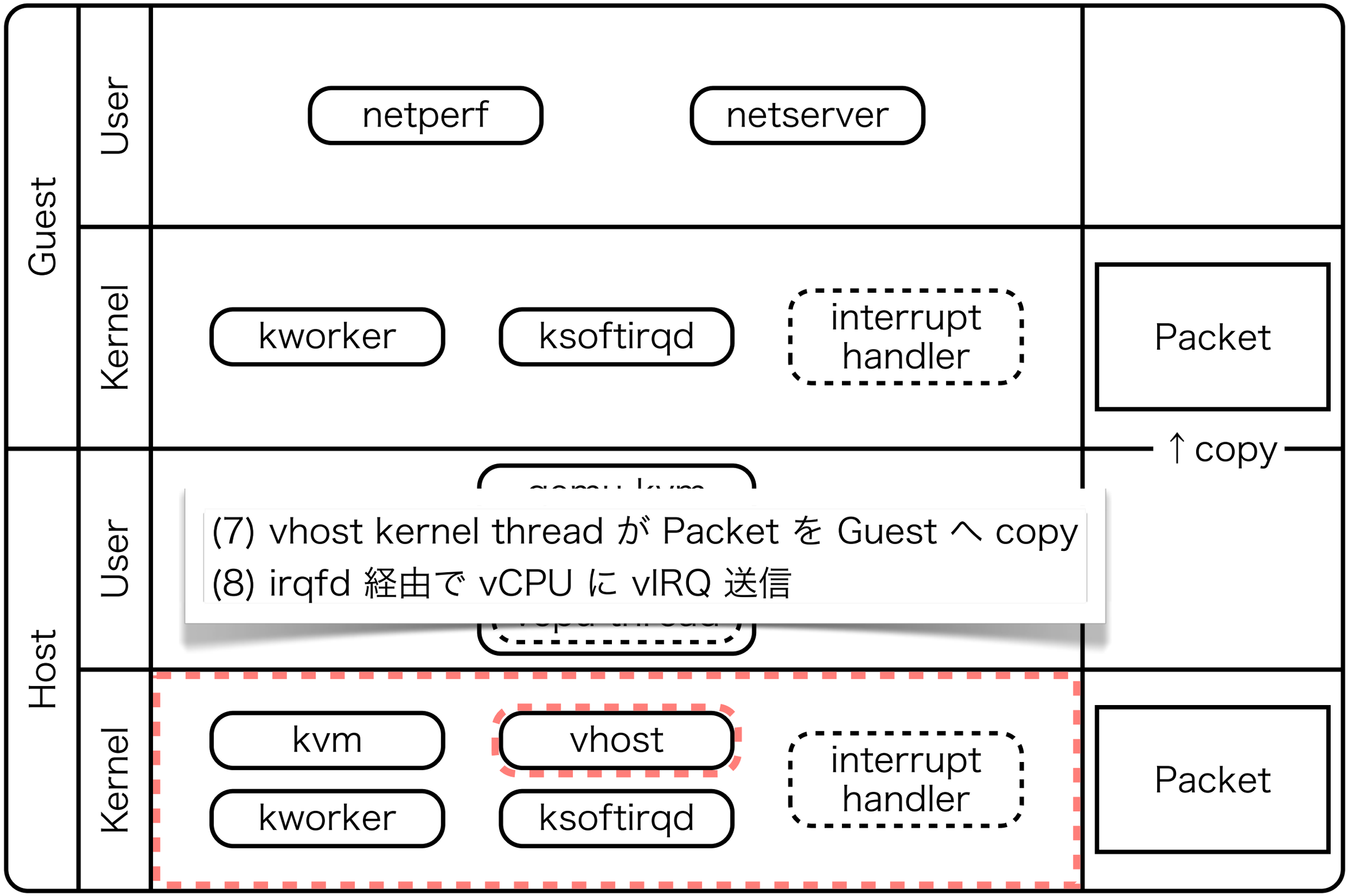
vhost-net: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Host
User
Kernel
qemu-kvm
(3) ... この部分は bridge/openvswitch/macvtap で異なる ...
(4) tap device を poll している process を wakeup
kvm
vhost
kworker
ksoftirqd
interrupt handler
Packet

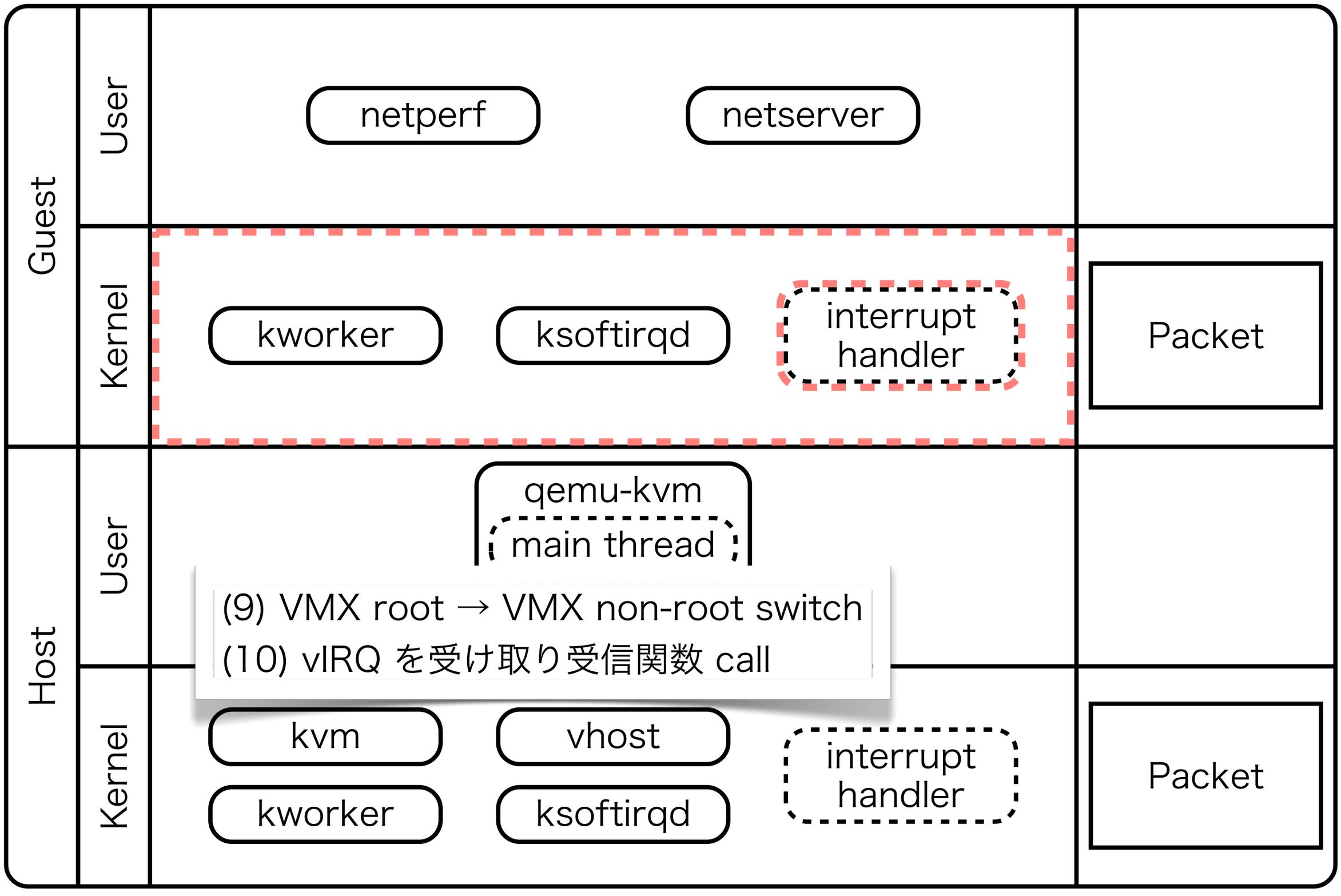
vhost-net: RX


Guest
User
Kernel
netperf
netserver
interrupt handler
(5) vhost kernel thread が起床
(6) tap から Packet を read
Host
User
qemu-kvm
main thread
vcpu thread
Kernel
kvm
vhost
kworker
ksoftirqd
interrupt handler
Packet

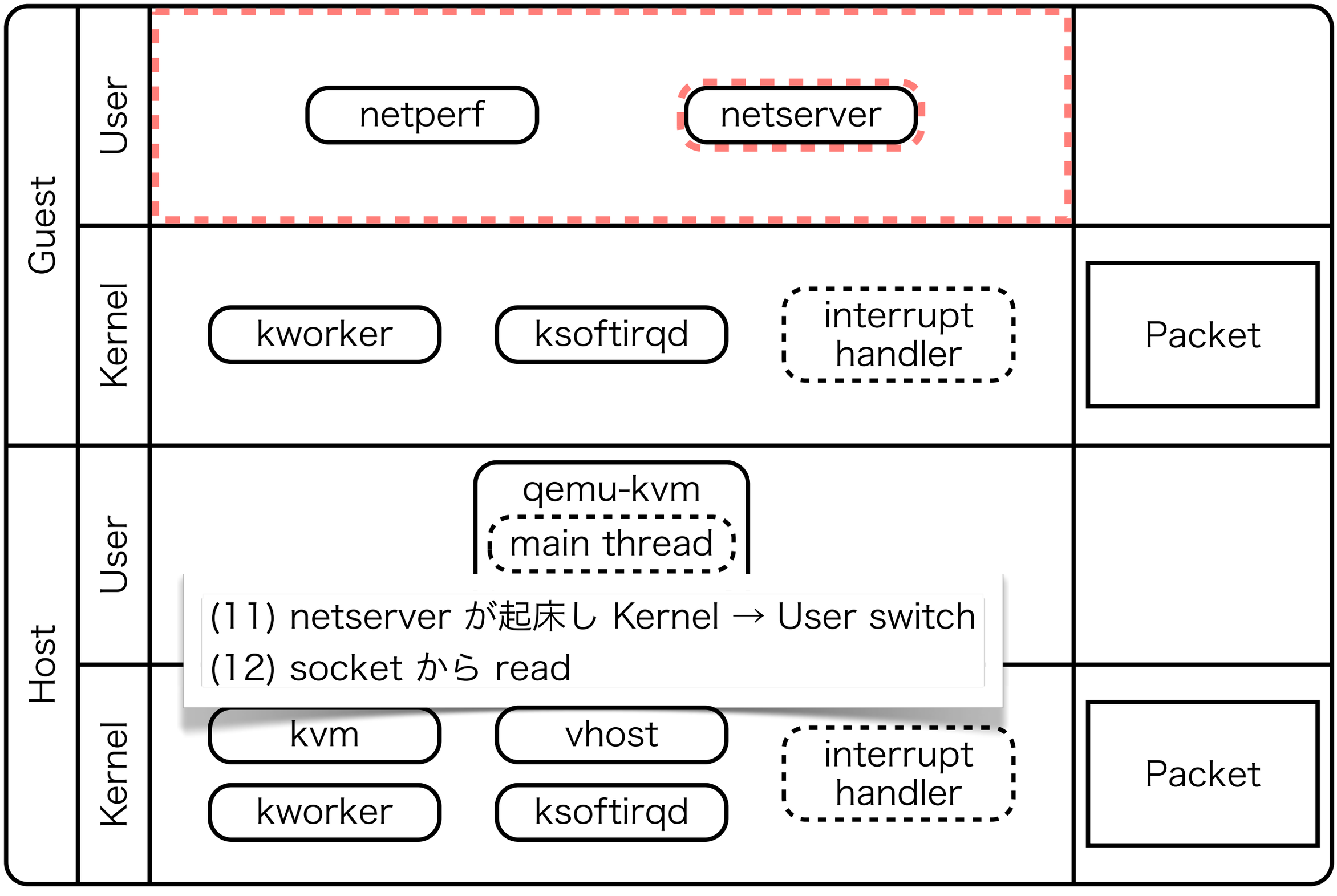
vhost-net: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Packet
↑ copy
Host
User
(7) vhost kernel thread が Packet を Guest へ copy
(8) irqfd 経由で vCPU に vIRQ 送信
Kernel
kvm
vhost
kworker
ksoftirqd
interrupt handler
Packet

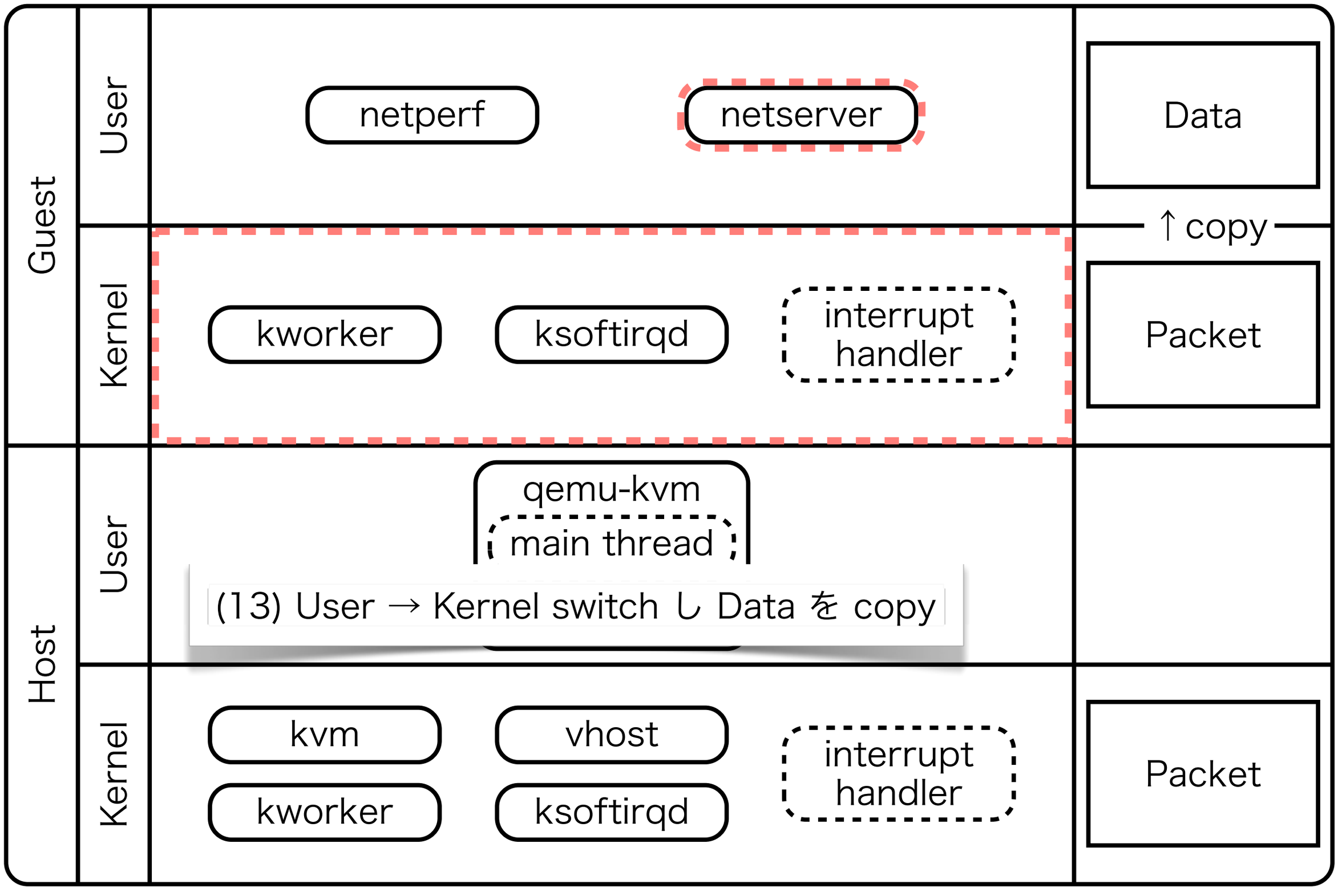
vhost-net: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Packet
Host
User
Kernel
qemu-kvm
main thread
(9) VMX root → VMX non-root switch
(10) vIRQ を受け取り受信関数 call
kvm
vhost
kworker
ksoftirqd
interrupt handler
Packet

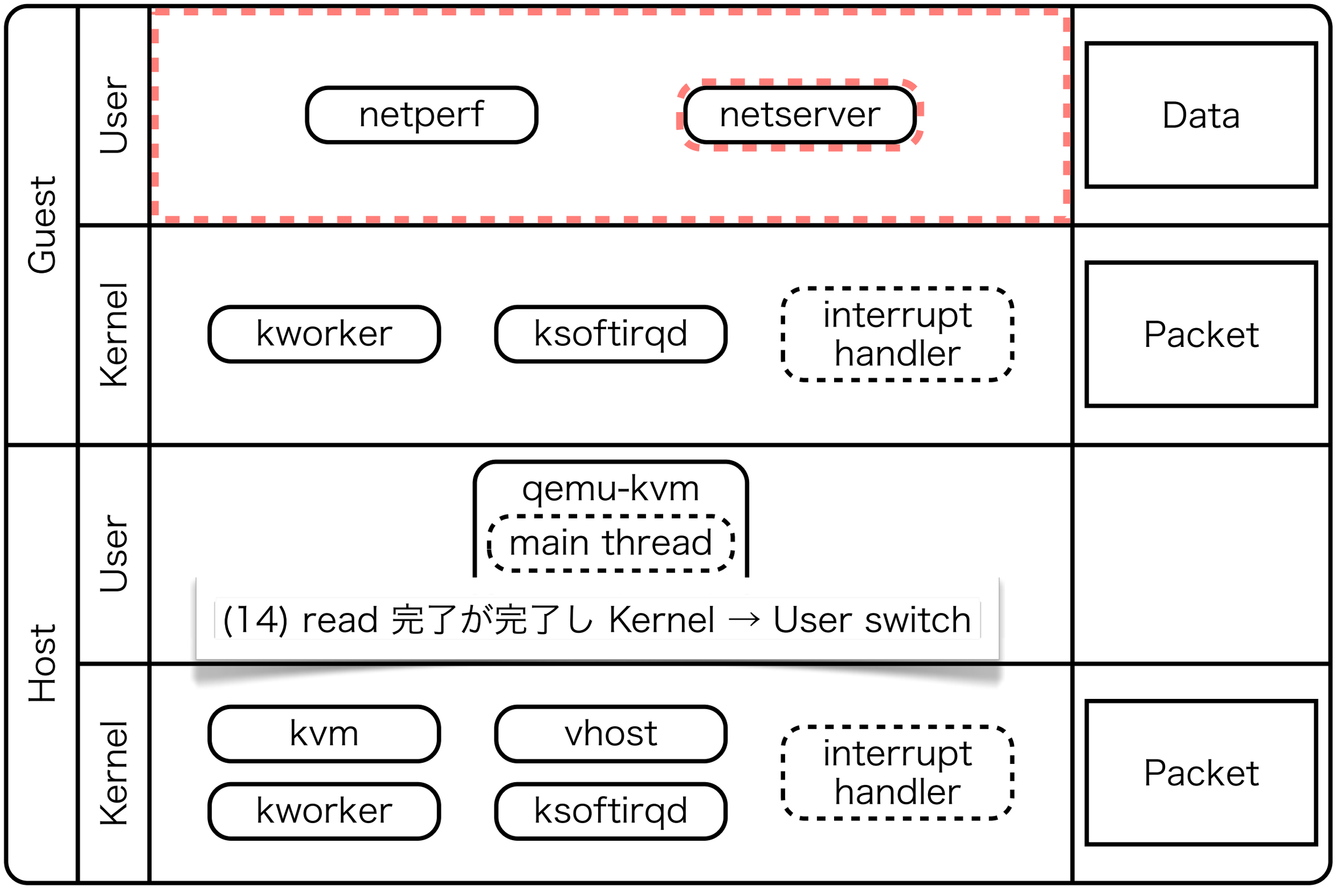
vhost-net: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Packet
Host
User
Kernel
qemu-kvm
main thread
(11) netserver が起床し Kernel → User switch
(12) socket から read
kvm
vhost
kworker
ksoftirqd
interrupt handler
Packet

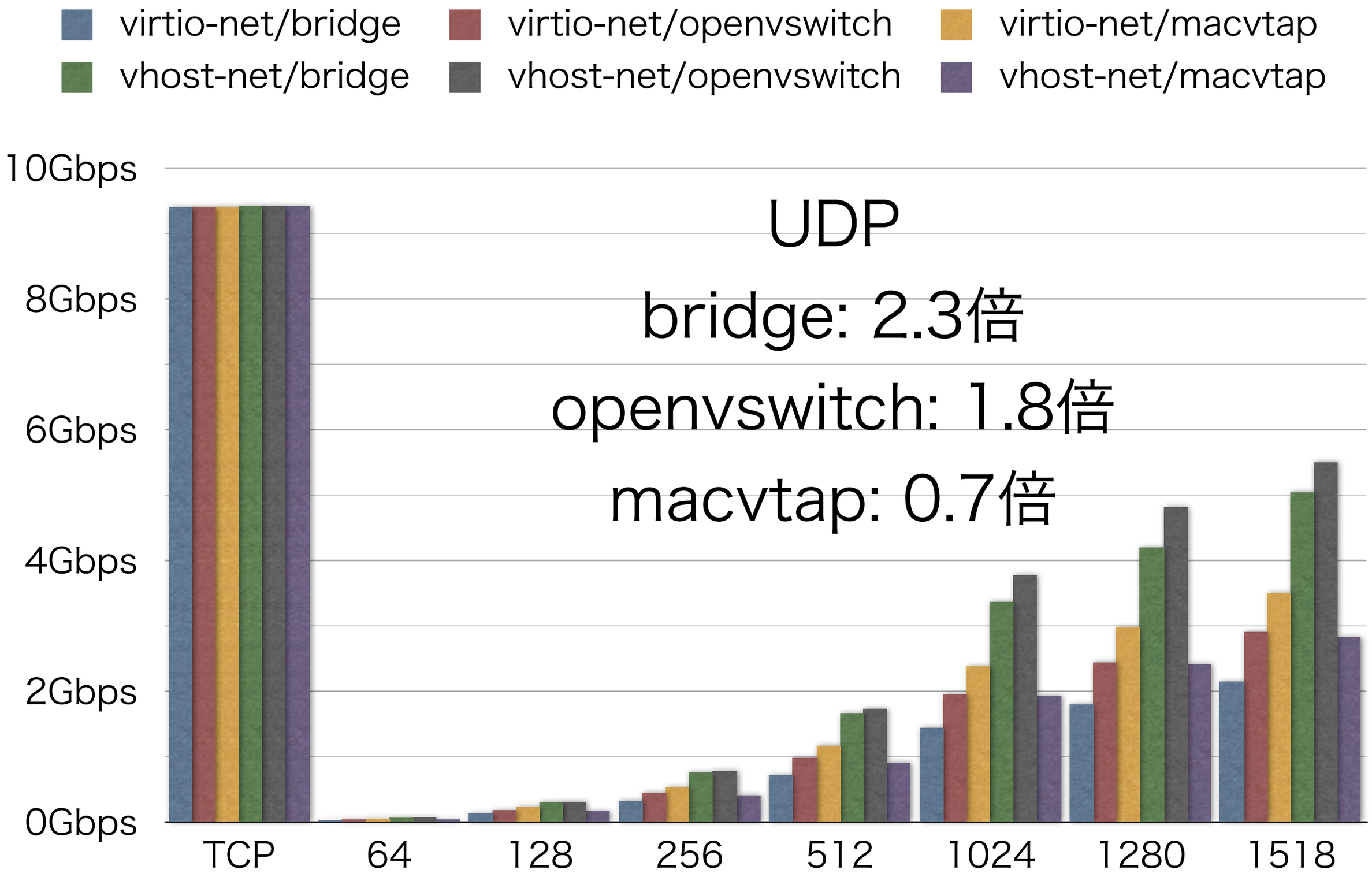
vhost-net: RX


Guest
User
Kernel
netperf
netserver
kworker
ksoftirqd
interrupt handler
Data
↑ copy
Packet
Host
User
Kernel
qemu-kvm
main thread
(13) User → Kernel switch し Data を copy
kvm
vhost
kworker
ksoftirqd
interrupt handler
Packet

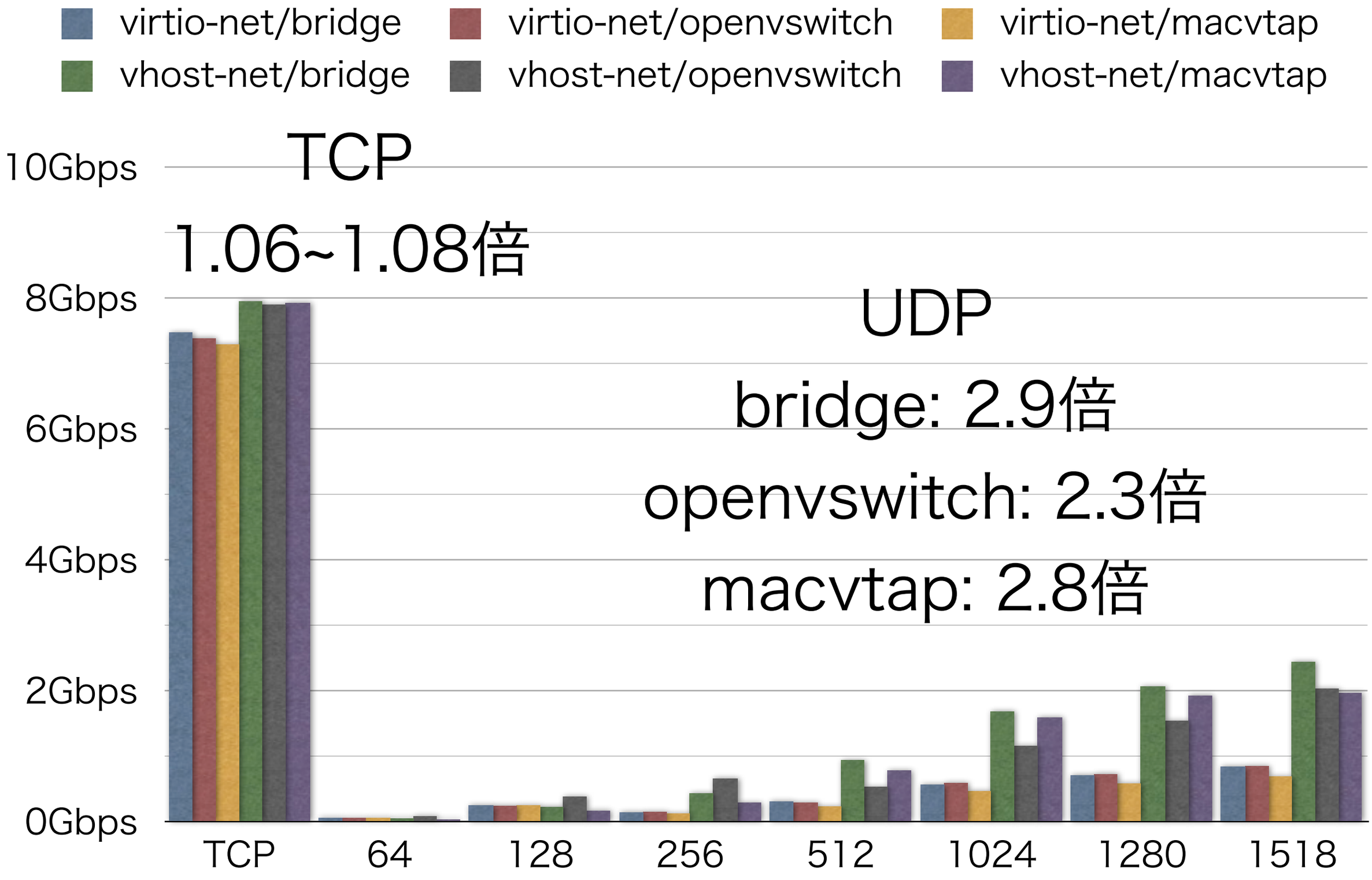
vhost-net: RX


Guest
User
Kernel
Host
User
Kernel
netperf
netserver
Data
kworker
ksoftirqd
interrupt handler
Packet
qemu-kvm
main thread
(14) read 完了が完了し Kernel → User switch
kvm
vhost
kworker
ksoftirqd
interrupt handler
Packet

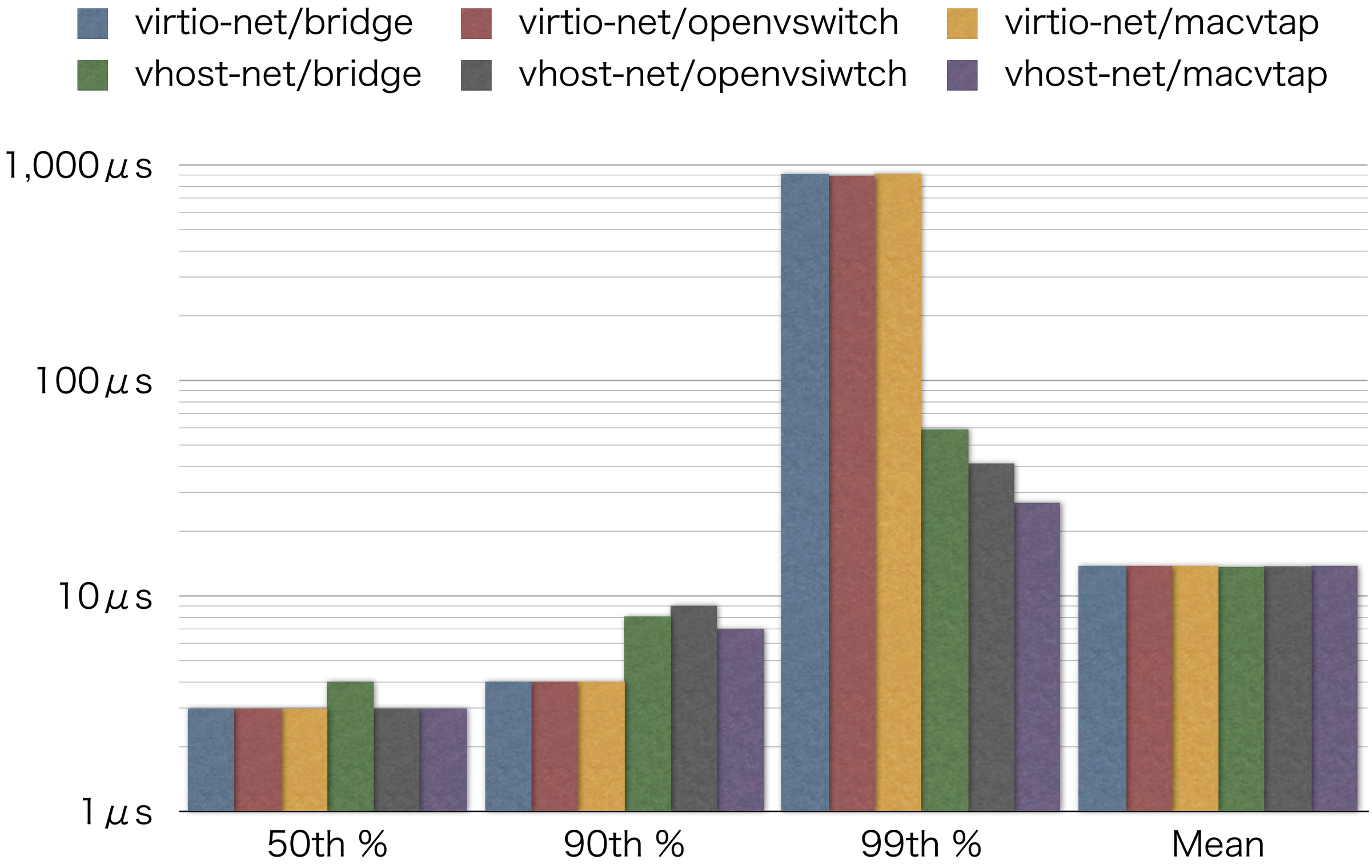
virtio-net v.s. vhost-net: Throughput: TX



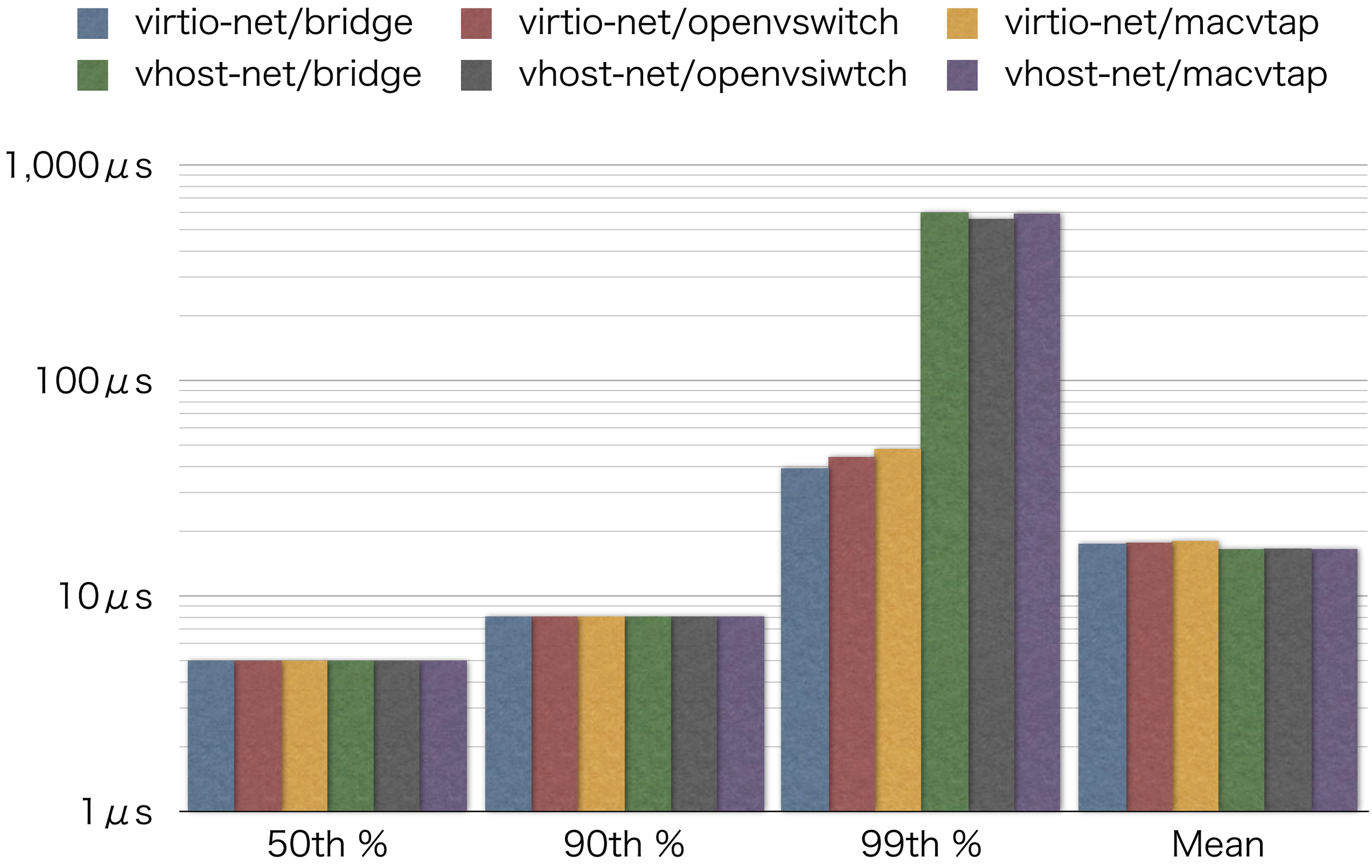
virtio-net v.s. vhost-net: Throughput: RX



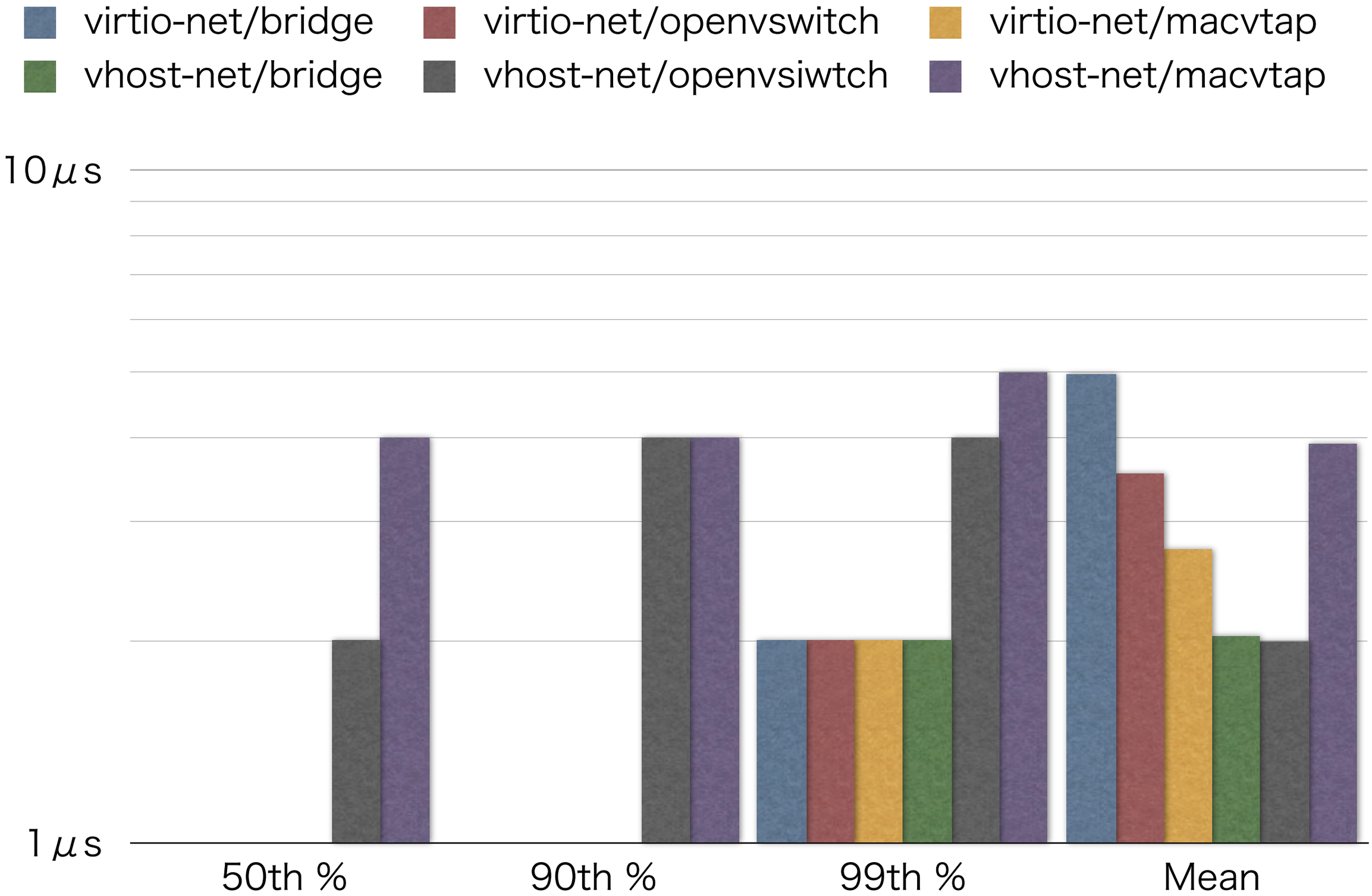
virtio-net v.s. vhost-net: Latency: TCP: TX



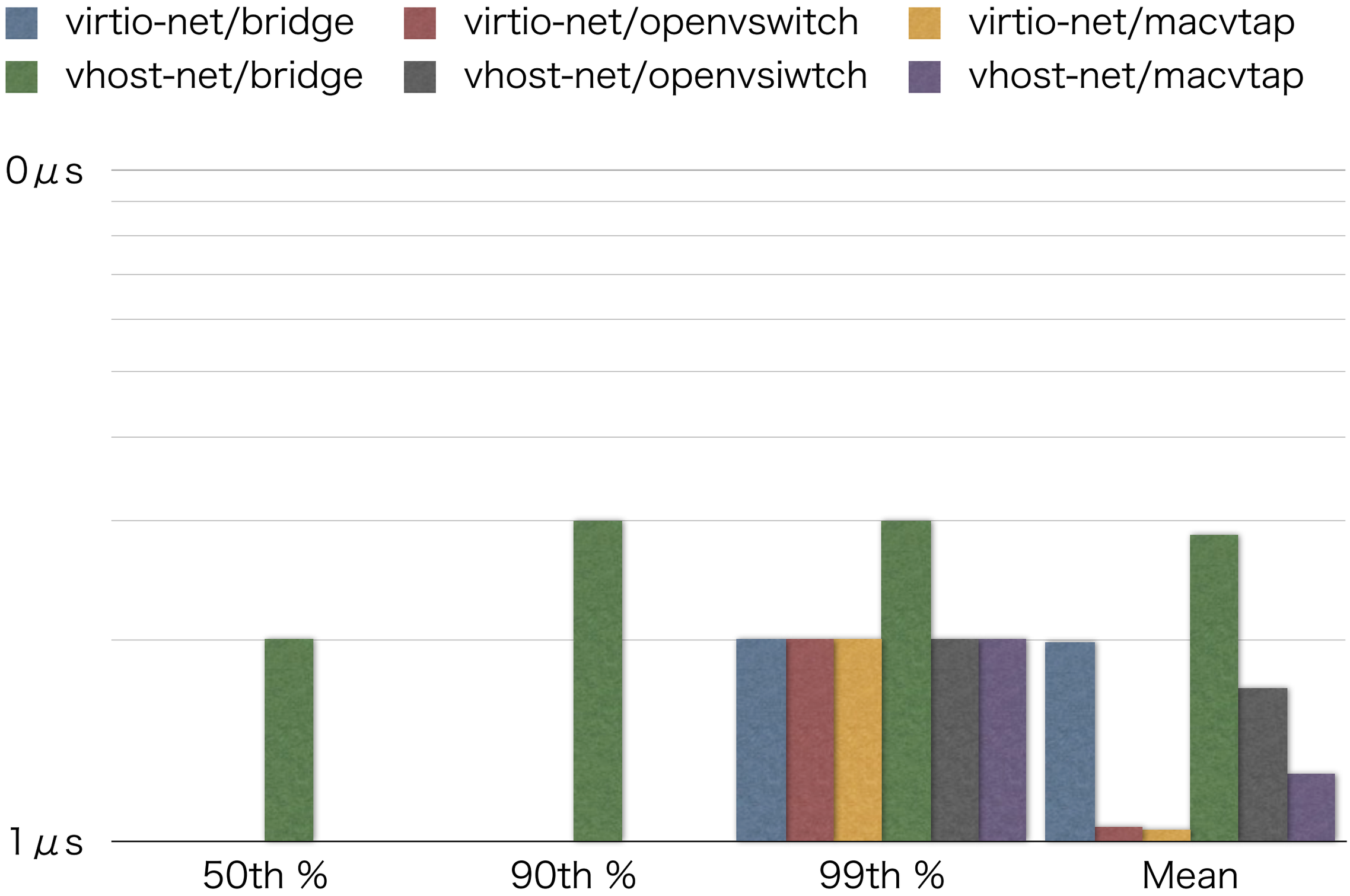
virtio-net v.s. vhost-net: Latency: TCP: RX



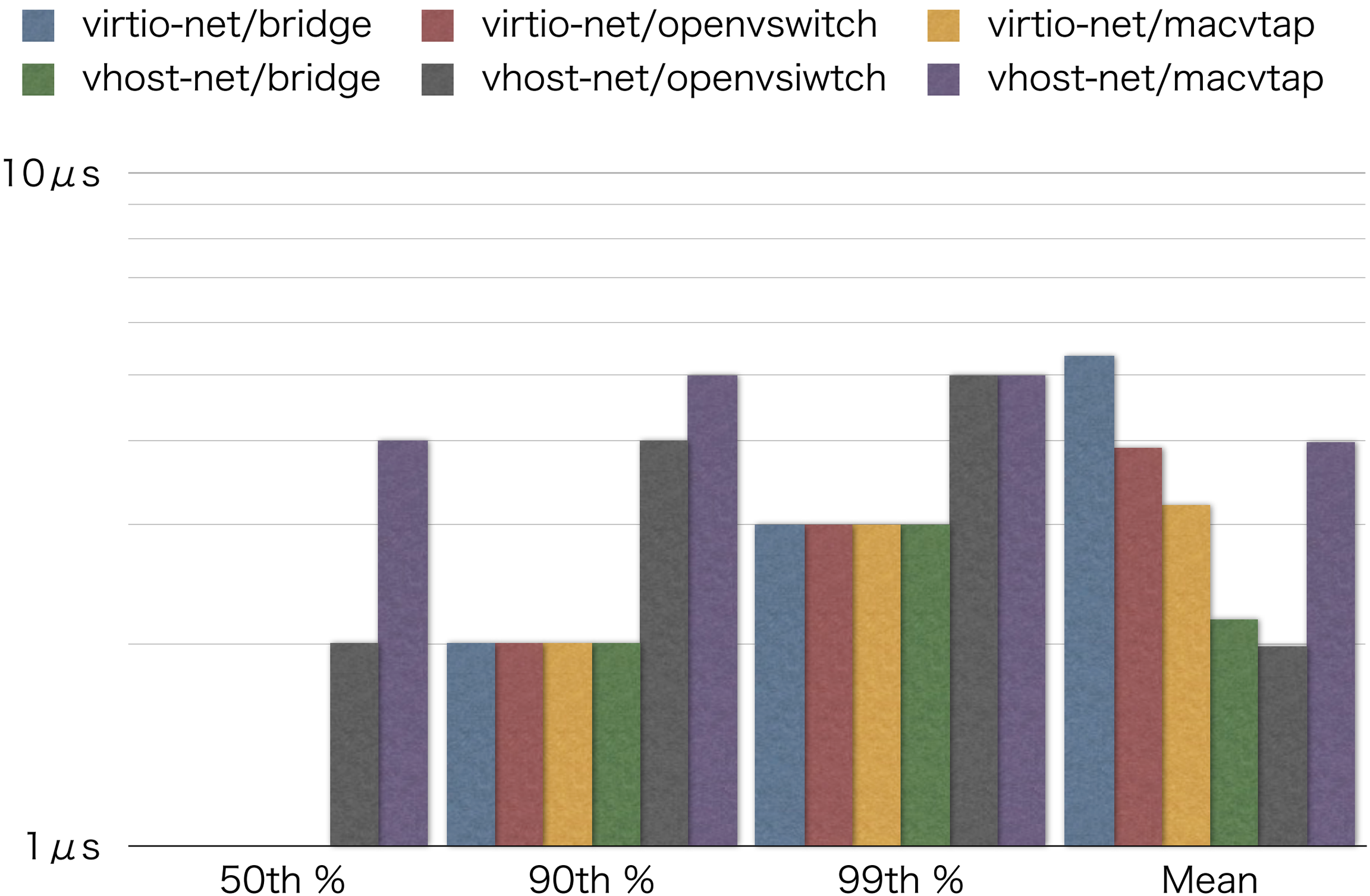
virtio-net v.s. vhost-net: Latency: UDP64: TX



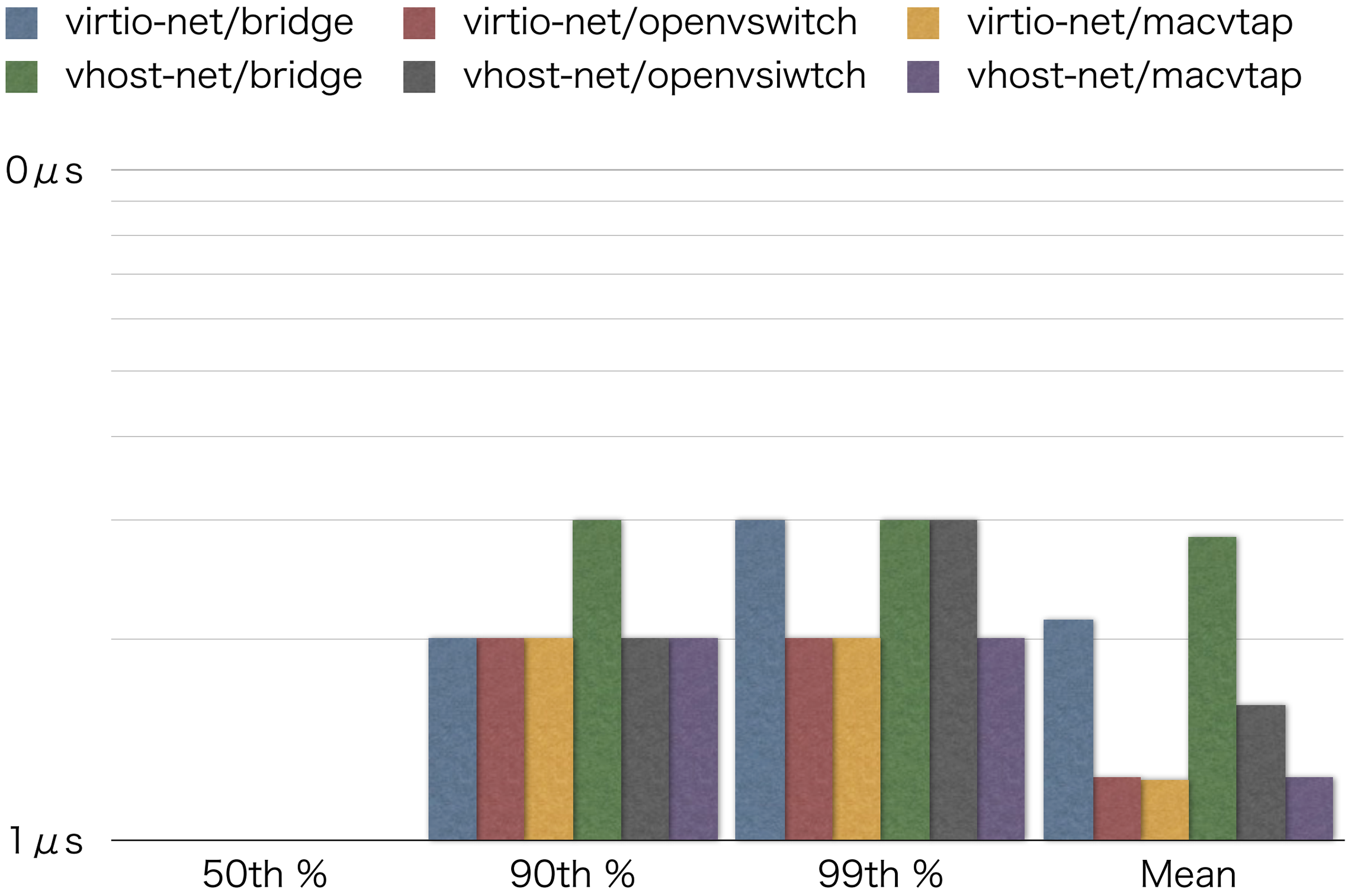
virtio-net v.s. vhost-net: Latency: UDP64: RX



virtio-net v.s. vhost-net: Latency: UDP1518: TX



virtio-net v.s. vhost-net: Latency: UDP1518: RX



まとめ

- 確実に高速化しているが packets/s 性能はまだまだしんどい？
  - ☞ 特に UDP short packet を大量に捌かなければならないような用途のサーバを仮想化環境で構築する際には性能問題に注意
- macvtap はまだ発展途上？
  - ☞ 現時点で性能面での優位性がそんなに無いのであれば機能面で優れた openvswitch を選択するのが無難なのかも
- 特に理由が無い限り virtio-net/vhost-net を使いましょう
  - ☞ なにか理由ある？

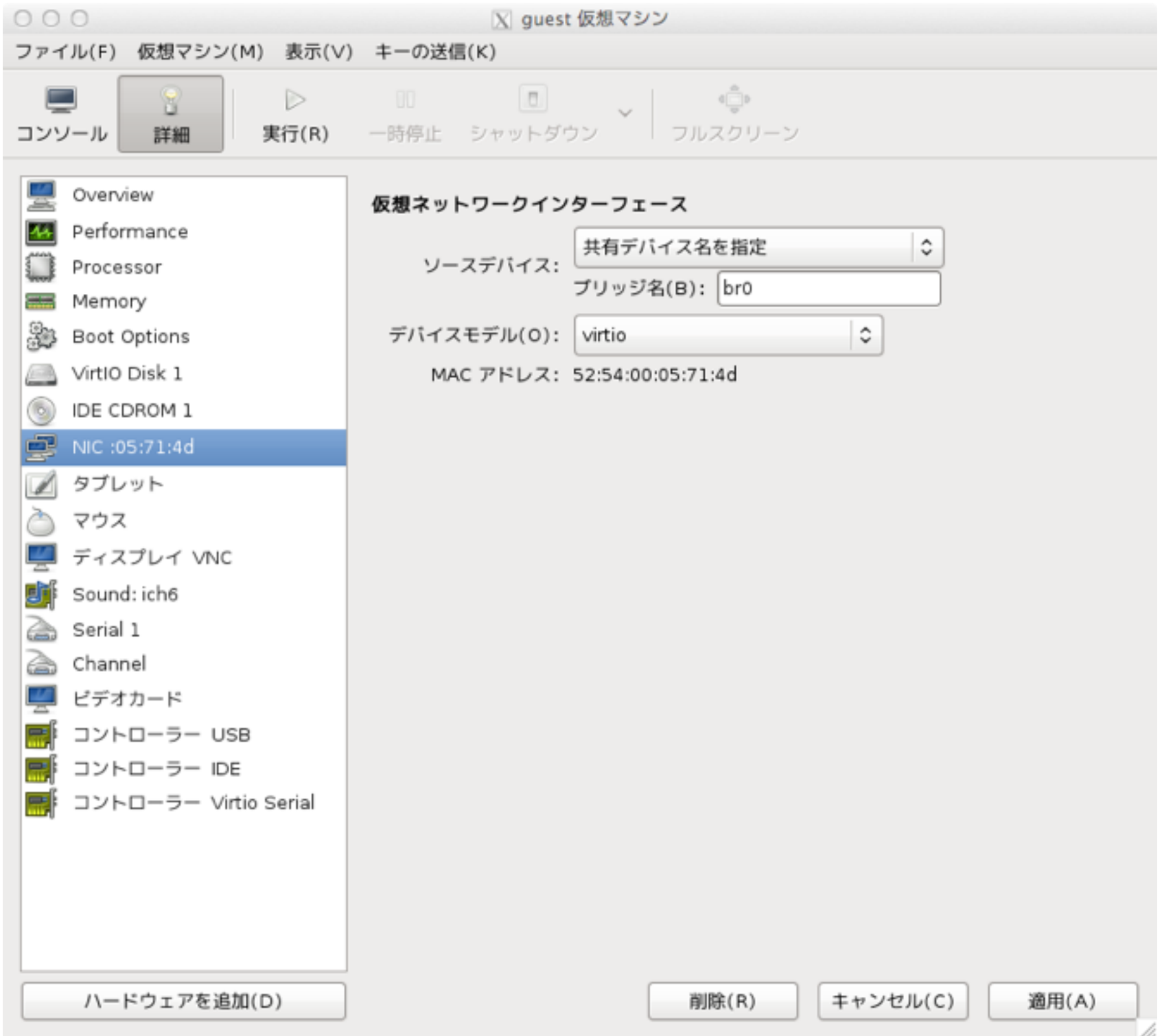
使い方: bridge

- virt-manager を使って設定するのが簡単（後述）
- libvirt の設定を直接編集して設定する場合は以下のように設定

☞ # virsh edit (guest name)

```
<interface type='bridge'>
  <mac address='52:54:00:05:71:4d'/>
  <source bridge='br0'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000'
        bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

使い方: bridge


guest 仮想マシン
ファイル(F)
仮想マシン(M)
表示(V)
キーの送信(K)
コンソール
詳細
実行(R)
一時停止
シャットダウン
フルスクリーン
Overview
Performance
Processor
Memory
Boot Options
VirtIO Disk 1
IDE CDROM 1
NIC :05:71:4d
タブレット
マウス
ディスプレイ VNC
Sound: ich6
Serial 1
Channel
ビデオカード
コントローラー USB
コントローラー IDE
コントローラー Virtio Serial
仮想ネットワークインターフェース
ソースデバイス:
共有デバイス名を指定
ブリッジ名(B):
br0
デバイスモデル(O):
virtio
MAC アドレス:
52:54:00:05:71:4d
ハードウェアを追加(D)
削除(R)
キャンセル(C)
適用(A)

使い方: openvswitch

- 現時点では virt-manager で設定できない
- libvirt が 0.9.11 以降の場合は以下のように設定することで起動時に自動的に openvswitch の bridge に接続される

  ☞ 一度 virt-manager で bridge と同様に設定してから virsh edit で編集するのが簡単

```
<interface type='bridge'>
  <mac address='52:54:00:05:71:4d'/>
  <source bridge='ovs0'/>
  <virtualport type='openvswitch'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000'
           bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

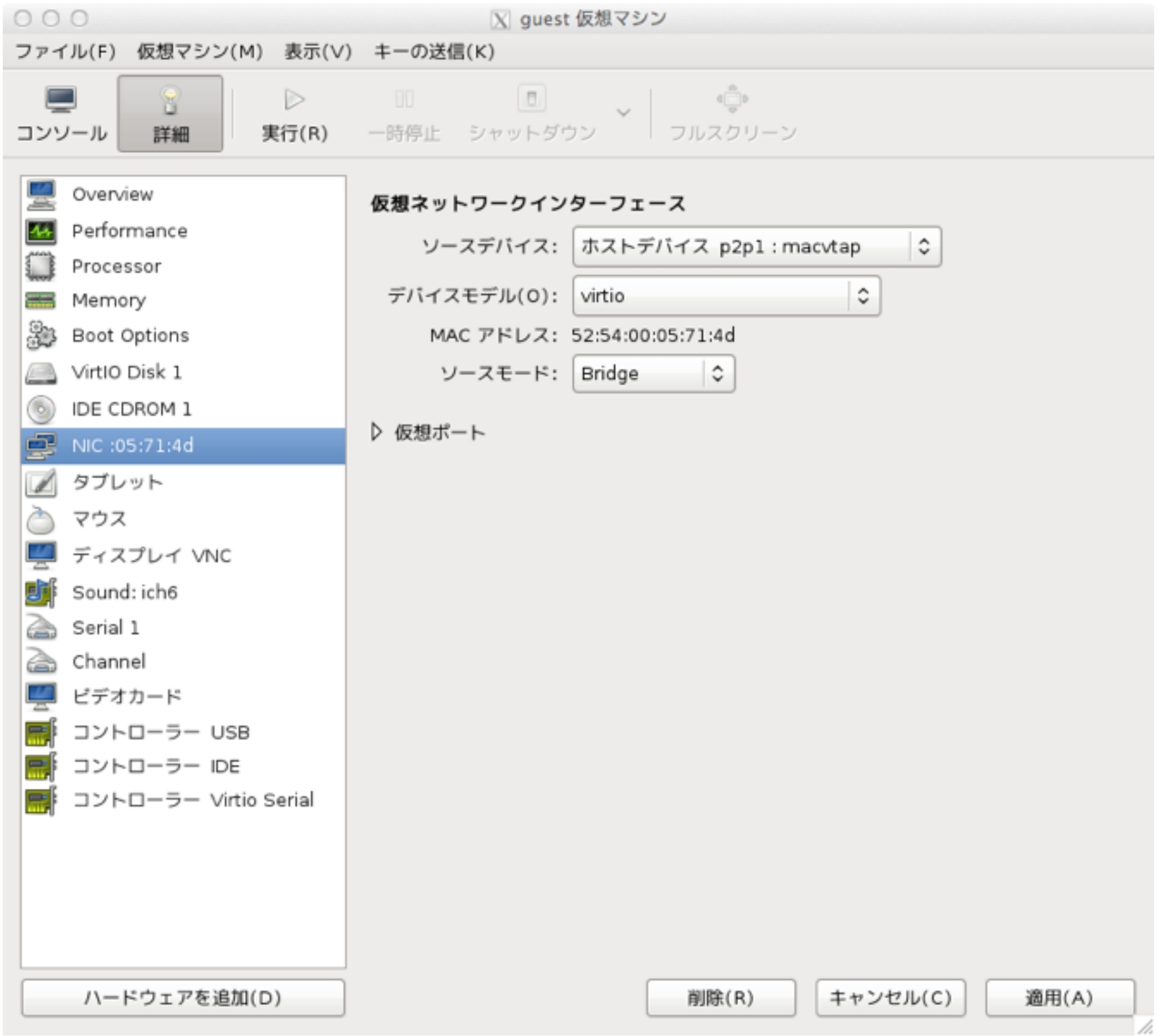
使い方: macvtap

- virt-manager を使って設定するのが簡単（後述）
- libvirt の設定を直接編集して設定する場合は以下のように設定

```
<interface type='direct'>
  <mac address='52:54:00:05:71:4d'/>
  <source dev='eth0' mode='bridge'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000'
           bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

- macvtap は Linux Kernel 2.6.34 以降で利用可能
- Linux Kernel が 2.6.34 以降でも iproute2 が macvtap に対応してないと利用できないので注意

使い方: macvtap

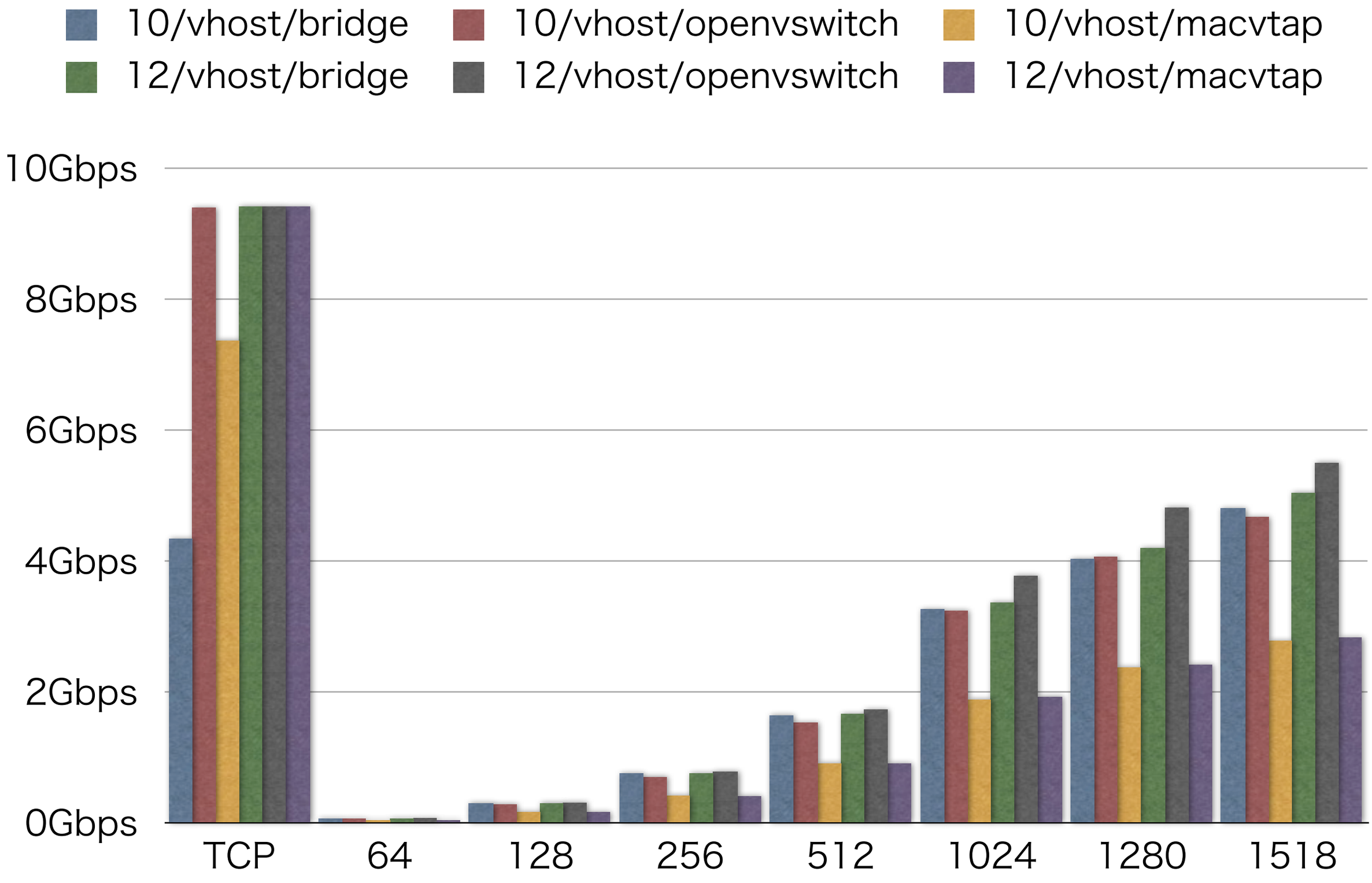

guest 仮想マシン
ファイル(F)
仮想マシン(M)
表示(V)
キーの送信(K)
コンソール
詳細
実行(R)
一時停止
シャットダウン
フルスクリーン
Overview
Performance
Processor
Memory
Boot Options
VirtIO Disk 1
IDE CDROM 1
NIC :05:71:4d
タブレット
マウス
ディスプレイ VNC
Sound: ich6
Serial 1
Channel
ビデオカード
コントローラー USB
コントローラー IDE
コントローラー Virtio Serial
仮想ネットワークインターフェース
ソースデバイス:
ホストデバイス p2p1 : macvtap
デバイスモデル(O):
virtio
MAC アドレス:
52:54:00:05:71:4d
ソースモード:
Bridge
仮想ポート
ハードウェアを追加(D)
削除(R)
キャンセル(C)
適用(A)

使い方: vhost-net

- libvirt を利用している場合は vhost_net という名前の kernel module が load されていれば自動的に vhost-net が使用される
- qemu-kvm の実行時引数を確認し vhost=on となっていて qemu-kvm の PID に対応する vhost-PID という名前の kernel thread が出来ていれば OK

```
# ps -ef
...
qemu 13740  1 46 14:21 ? 00:00:10 /usr/bin/qemu-kvm -name guest
      ... -netdev tap,fd=23,id=hostnet0,vhost=on,vhostfd=24 ...
...
root 13766  2  0 14:21 ? 00:00:00 [vhost-13740]
...
```

QEMU1.0 v.s. QEMU1.2: TX



QEMU1.0 v.s. QEMU1.2: RX