Empty Promise: Zero-copy Receive for vhost

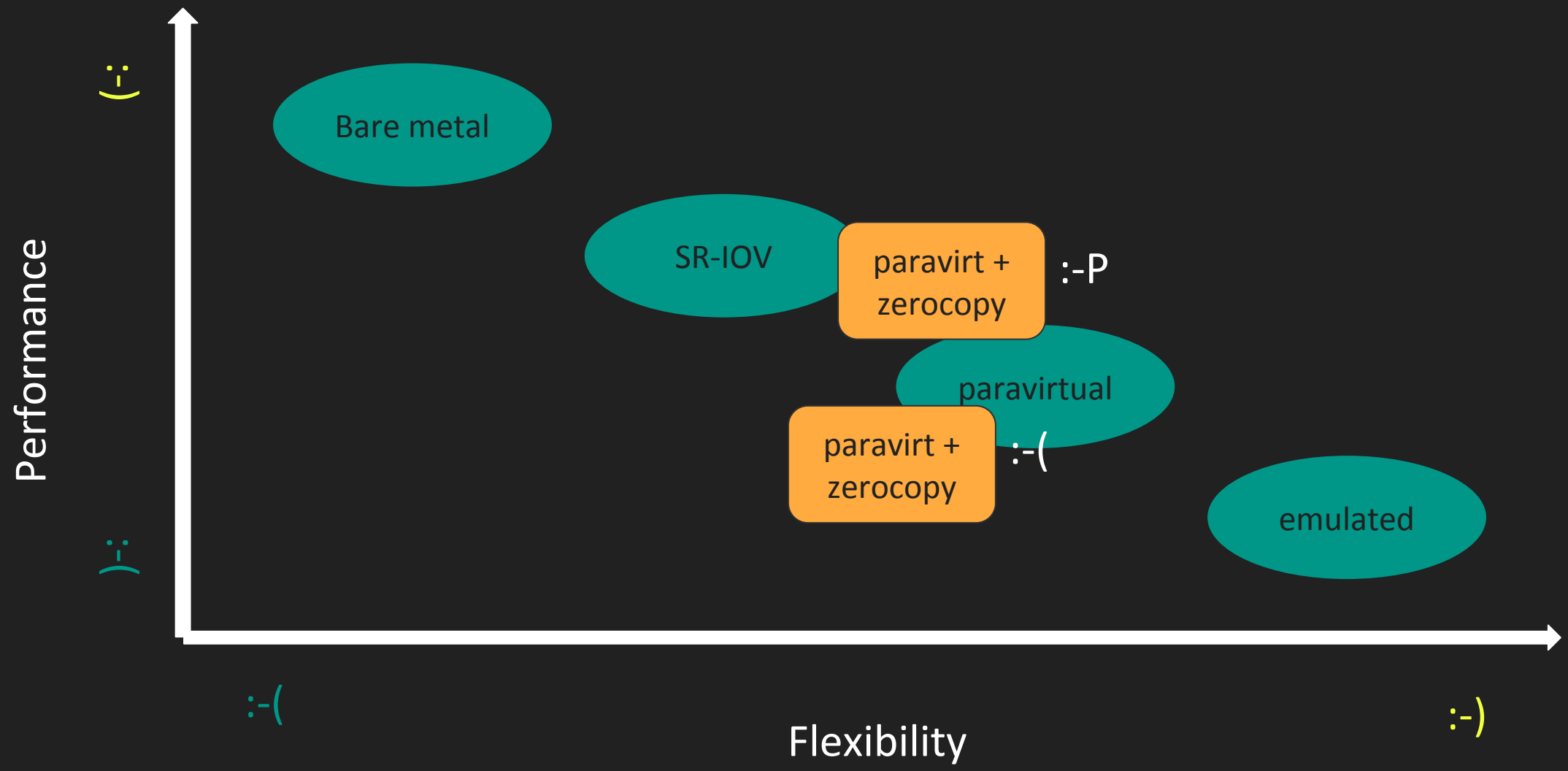
Kalman Meth, Mike Rapoport, Joel Nider

{meth,joeln}@il.ibm.com

rppt@linux.vnet.ibm.com

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreements No 645402 and No 688386.

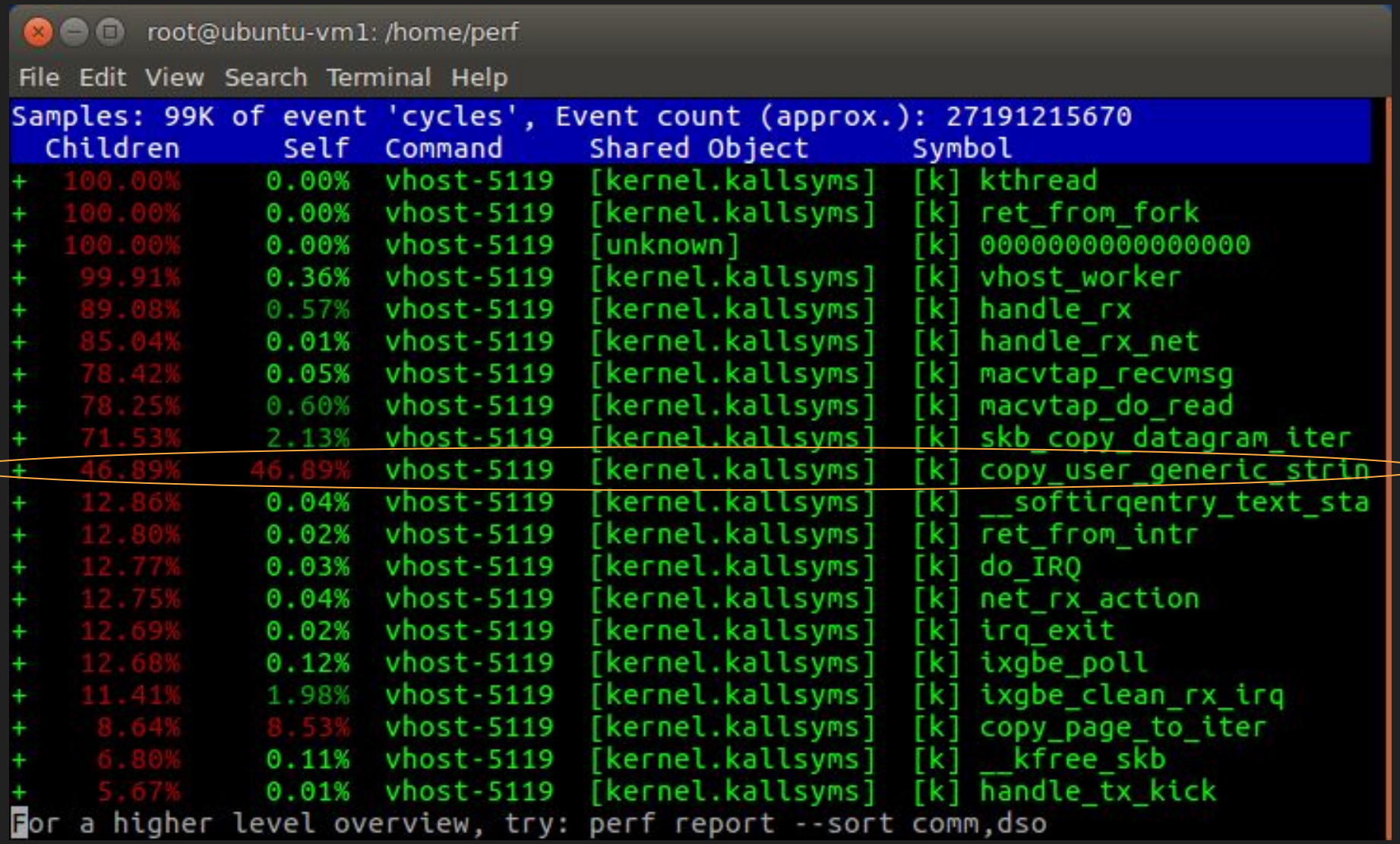
Virtualization and IO



Motivation

- No copy is better than copy
- Zerocopy TX without RX should feel lonely
- It was 8 years since the last attempt. Can we do better?

More motivation



Zerocopy: TX vs RX

Transmit

- Downstream routing is easy
- Memory is always at hand

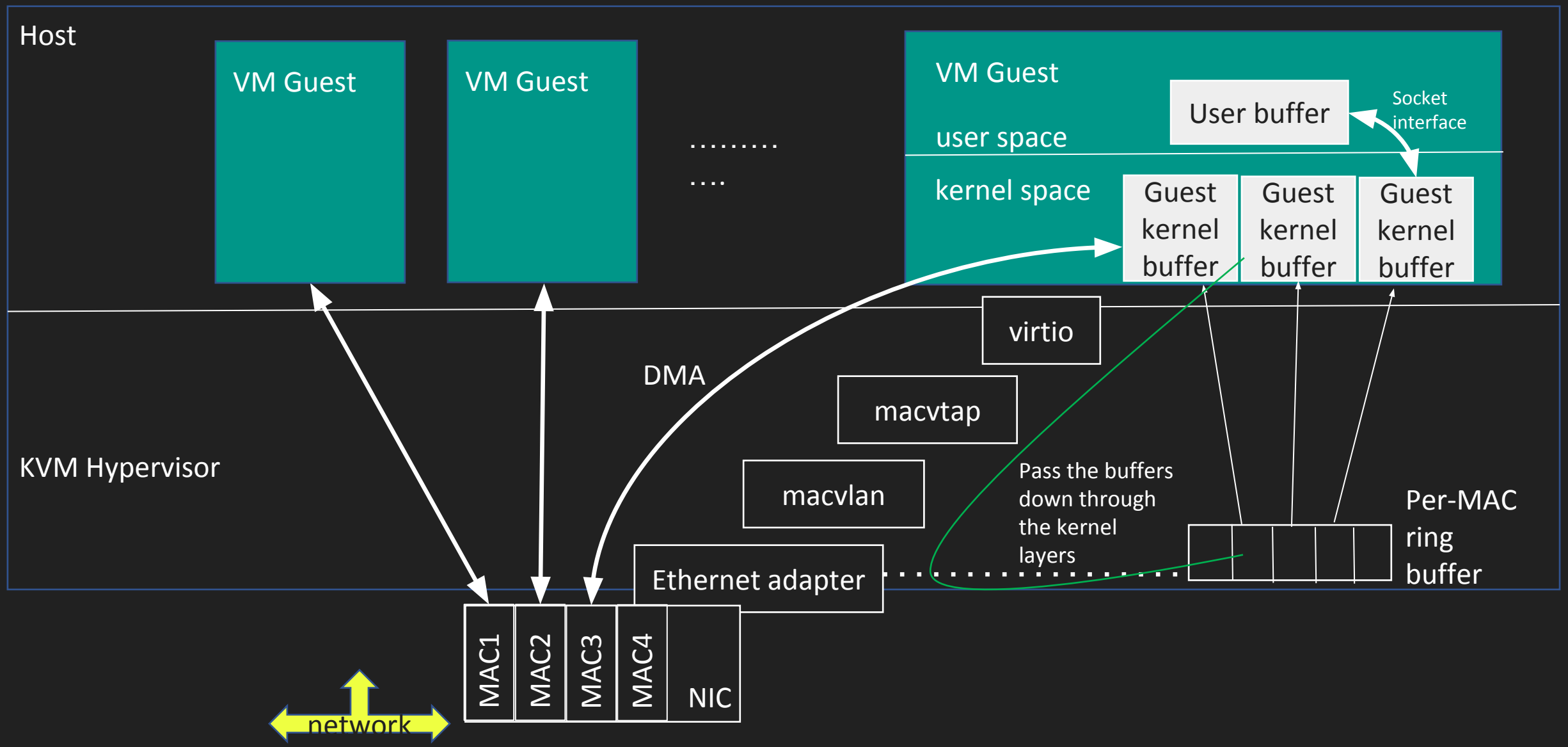
Receive

- Destination is not yet known
- Need memory for DMA
- Does not exist yet

Assumptions

- Modern NICs are multiqueue
  - Dedicate queues to virtual NIC
- Guest allocates the buffers
  - Remapping DMA region to guest is more complex
- Tight coupling between physical and virtual NICs
  - Restrict zerocopy-RX to macvtap
- Minimal changes to guest

Zero-Copy Rx Architecture



API changes

netdev

- `->ndo_set_zerocopy_rx(struct net_device *pdev, struct net_device *vdev)`
  - Pass vdev down the stack to the ethernet adapter to bind physical and virtual queues.
  - Similar to `->ndo_dfwd_add_station()`
- `->ndo_post_rx_buffer(struct net_device *dev, struct sk_buff *skb)`
  - Passes a single (page aligned) buffer to the ethernet adapter
  - `skb` contains pointer to the upper level device and `ubuf_info`

API changes (cont)

IBM

macvtap

- `MSG_ZCOPY_RX_POST`
  - Control message from vhost-net to macvtap to propagate the buffers from guest to the lower levels
- `MSG_ZCOPY_RX`
  - Flag indicating that message contains preallocated buffers that should not be copied to userspace

API changes (cont)

virtio-net

- `add_recvbuf_full_page()`
  - Ethernet adapter driver expects page size aligned buffers
  - Existing `add_recvbuf_*()` do not care since the data was always copied

Initialization

- Isolate set of queues in physical NIC
- Create 1:1 correspondence between physical and virtual queues
- Clear RX descriptor ring
- Drop pre-allocated RX buffers in physical NIC driver

Memory allocation

- virtio-net (guest)
  - Allocate buffers
    - DMA'able memory (`PAGE_SIZE` granularity and page aligned)
- vhost-net
  - Post buffers to macvtap
    - New control flag `MSG_ZCOPY_RX_POST` for `macvtap_recvmsg()`
- macvtap
  - Allocate `skb`
  - Map `iovec` to `skb` (similar to `zerocopy_sg_from_iter`)
  - Pass the buffers to physical NIC
    - New method `ndo_post_rx_buffer()`
- Physical NIC driver adds new buffers to RX descriptor ring

Packet receive

- Physical NIC driver
  - DMA directly to the guest buffers
  - Setup skb structure
  - `netif_rx()` and friends
- macvtap
  - Queue skb as ready for the userspace
  - Inform vhost-net about the virtio descriptor associated with the `skb`.

Packet receive (cont)

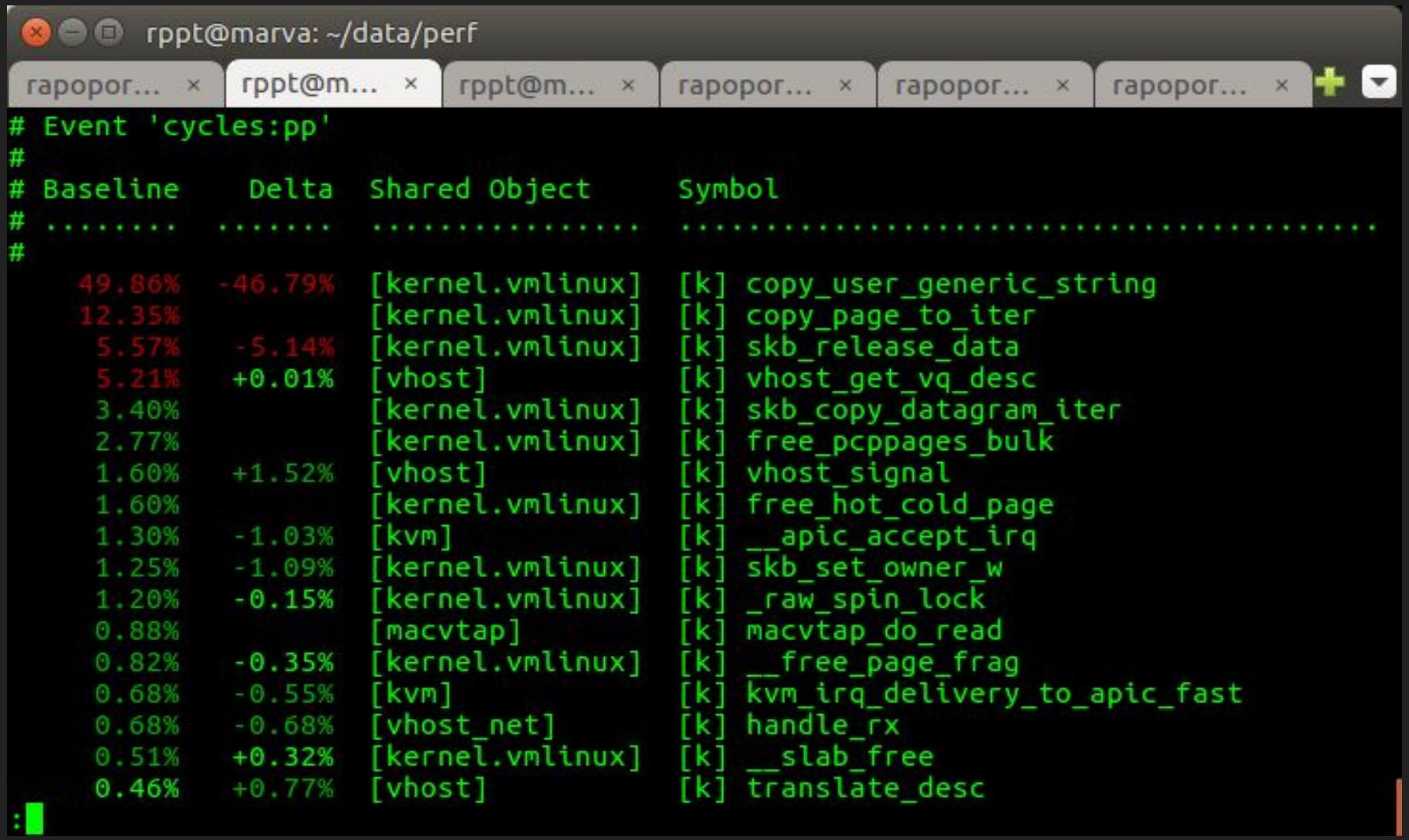
- vhost-net
  - `handle_rx_zero_copy()`:
    - Update virtqueue
    - Kick macvtap with `->recvmsg(MSG_ZCOPY_RX)`
- macvtap (again)
  - `macvtap_do_read_zero_copy()`
    - skb_array_consume
    - Cleanup

Implementation status

- Initial implementation
  - still sub-optimal
- Stable enough to benchmark
- Source:
  - https://github.com/mikelangelo-project/linux-zecorx
  - https://github.com/mikelangelo-project/qemu-zecorx

Test setup

- 2x IBM System x3550 M4 Server
  - Intel® Xeon® Processor E5-2660
    - 8 cores, 16 threads
  - 56G RAM
  - Intel 82599ES 10-Gigabit Network Connection (ixgbe)
- Back to back connection for host NICs
- VM with 4 vCPUs, 2G RAM
- Linux v4.8
- netperf 2.6.0


rppt@marva: ~/data/perf
rapopor...
rppt@m...
rppt@m...
rapopor...
rapopor...
rapopor...
Event 'cycles:pp'
#
Baseline Delta Shared Object Symbol
....
#
49.86% -46.79% [kernel.vmlinux] [k] copy_user_generic_string
12.35% [kernel.vmlinux] [k] copy_page_to_iter
5.57% -5.14% [kernel.vmlinux] [k] skb_release_data
5.21% +0.01% [vhost] [k] vhost_get_vq_desc
3.40% [kernel.vmlinux] [k] skb_copy_datagram_iter
2.77% [kernel.vmlinux] [k] free_pcppages_bulk
1.60% +1.52% [vhost] [k] vhost_signal
1.60% [kernel.vmlinux] [k] free_hot_cold_page
1.30% -1.03% [kvm] [k] __apic_accept_irq
1.25% -1.09% [kernel.vmlinux] [k] skb_set_owner_w
1.20% -0.15% [kernel.vmlinux] [k] _raw_spin_lock
0.88% [macvtap] [k] macvtap_do_read
0.82% -0.35% [kernel.vmlinux] [k] __free_page_frag
0.68% -0.55% [kvm] [k] kvm_irq_delivery_to_apic_fast
0.68% -0.68% [vhost_net] [k] handle_rx
0.51% +0.32% [kernel.vmlinux] [k] __slab_free
0.46% +0.77% [vhost] [k] translate_desc
:

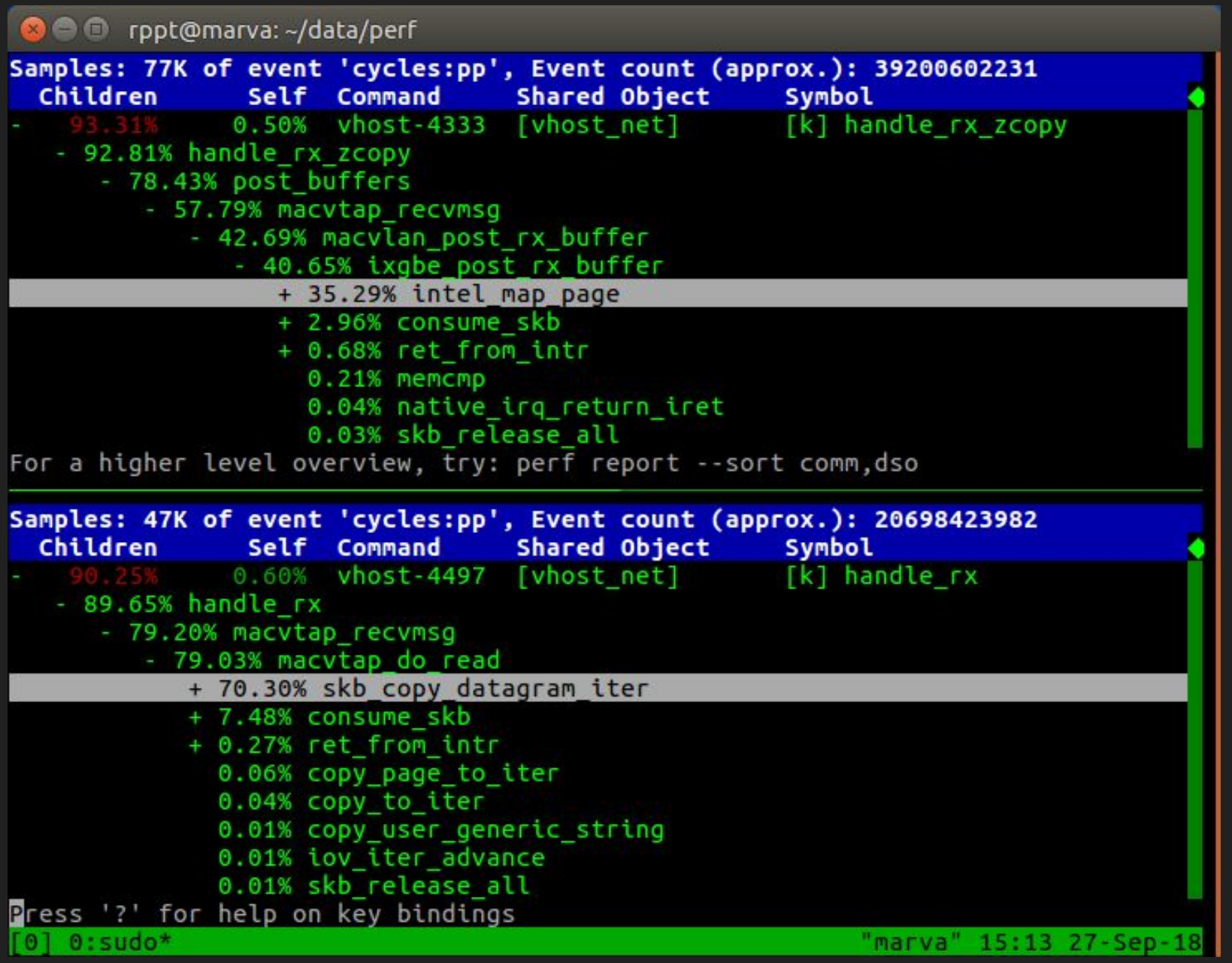
```
traffic-gen$ netperf -H guest
```

	v4.8	**v4.8 + ZCRX**
Throughput (Mbit/sec)	9255.22	4396.34
System Utilization (%)	10.9095	12.2941
CPU usage - vhost (%)	74.676	95.979
CPU usage - qemu (%)	99.6685	100.83

- RX interrupt, and vhost are on the same CPU
- Retry with forced CPU affinity

```
vm-host$ taskset -cp 4 $(pgrep vhost)
vm-host$ echo 5 > /proc/irq/111/smp_affinity_list
traffic-gen$ netperf -H guest
```

	v4.8	**v4.8 + ZCRX**
Throughput (Mbit/sec)	9255.22	8492.17
System Utilization (%)	10.9095	12.121
CPU usage - vhost (%)	74.676	86.1722
CPU usage - qemu (%)	99.6685	100.103


rppt@marva: ~/data/perf
Samples: 77K of event 'cycles:pp', Event count (approx.): 39200602231
Children Self Command Shared Object Symbol
- 93.31% 0.50% vhost-4333 [vhost_net] [k] handle_rx_zcopy
- 92.81% handle_rx_zcopy
- 78.43% post_buffers
- 57.79% macvtap_recvmsg
- 42.69% macvlan_post_rx_buffer
- 40.65% ixgbe_post_rx_buffer
+ 35.29% intel_map_page
+ 2.96% consume_skb
+ 0.68% ret_from_intr
0.21% memcmp
0.04% native_irq_return_iret
0.03% skb_release_all
For a higher level overview, try: perf report --sort comm,dso
Samples: 47K of event 'cycles:pp', Event count (approx.): 20698423982
Children Self Command Shared Object Symbol
- 90.25% 0.60% vhost-4497 [vhost_net] [k] handle_rx
- 89.65% handle_rx
- 79.20% macvtap_recvmsg
- 79.03% macvtap_do_read
+ 70.30% skb_copy_datagram_iter
+ 7.48% consume_skb
+ 0.27% ret_from_intr
0.06% copy_page_to_iter
0.04% copy_to_iter
0.01% copy_user_generic_string
0.01% iov_iter_advance
0.01% skb_release_all
Press '?' for help on key bindings
[0] 0:sudo* "marva" 15:13 27-Sep-18

What happened

- New bottleneck is in DMA mapping
- Latency has grown
  - Measured with `rdtsc()` in `vhost-net::handle_rx()`
    - Copy: ≃ 2 tscs/byte
    - No copy: ≃ 3.5 tscs/byte
- Page recycling in ixgbe replaced with `ndo_post_buffers()`
  - Sequential instead of parallel
  - Move frequent `dma_map*()` / `dma_unmap()`

Can we do better?

- Maybe
- tl;dr
  - Try to re-parallel DMA mapping and RX processing
  - Better batching for memory allocation and dma_map/unmap
  - Major changes to virtio ring
  - `AF_XDP` based virtio backend in user space

Thank you!