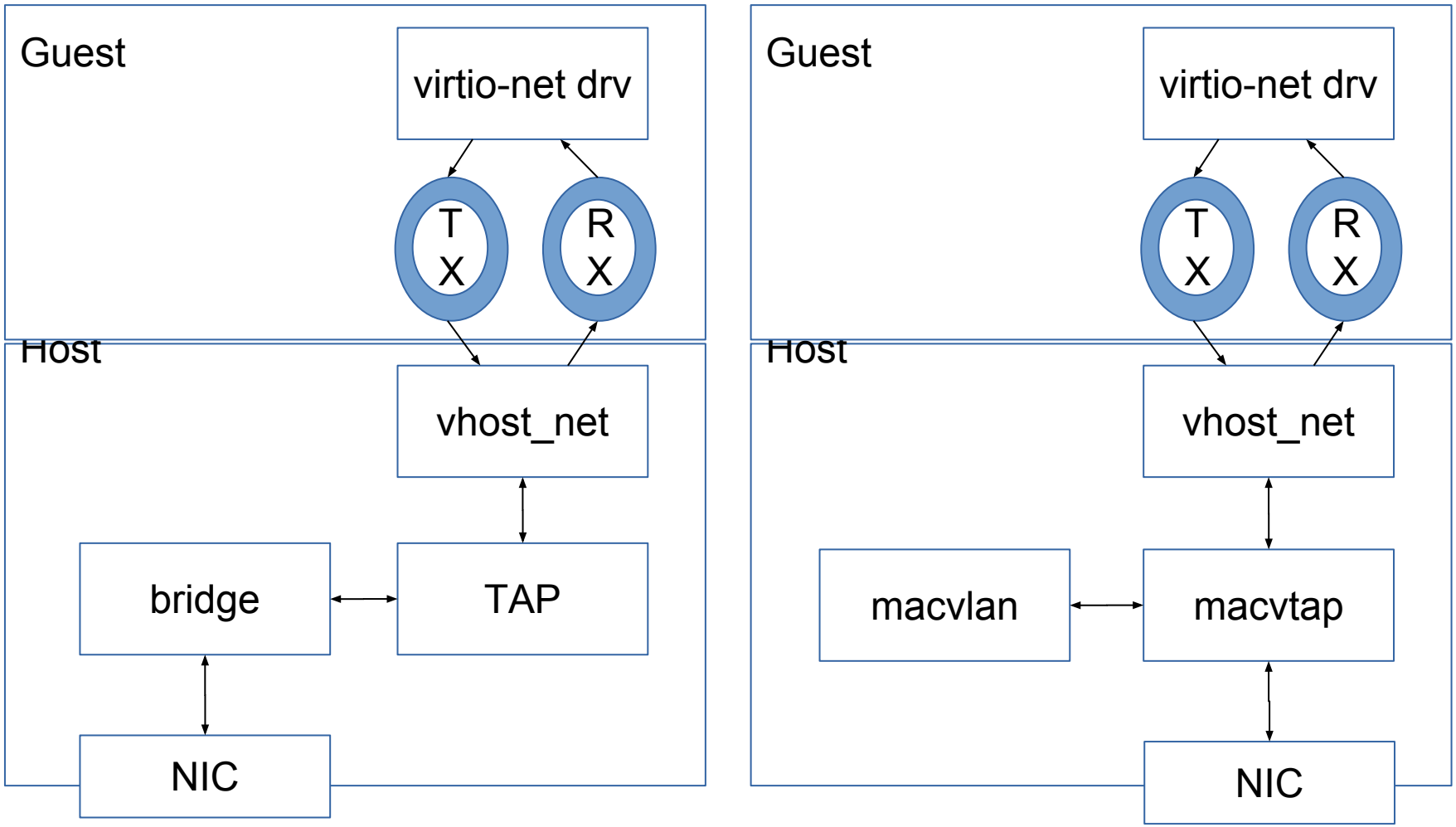
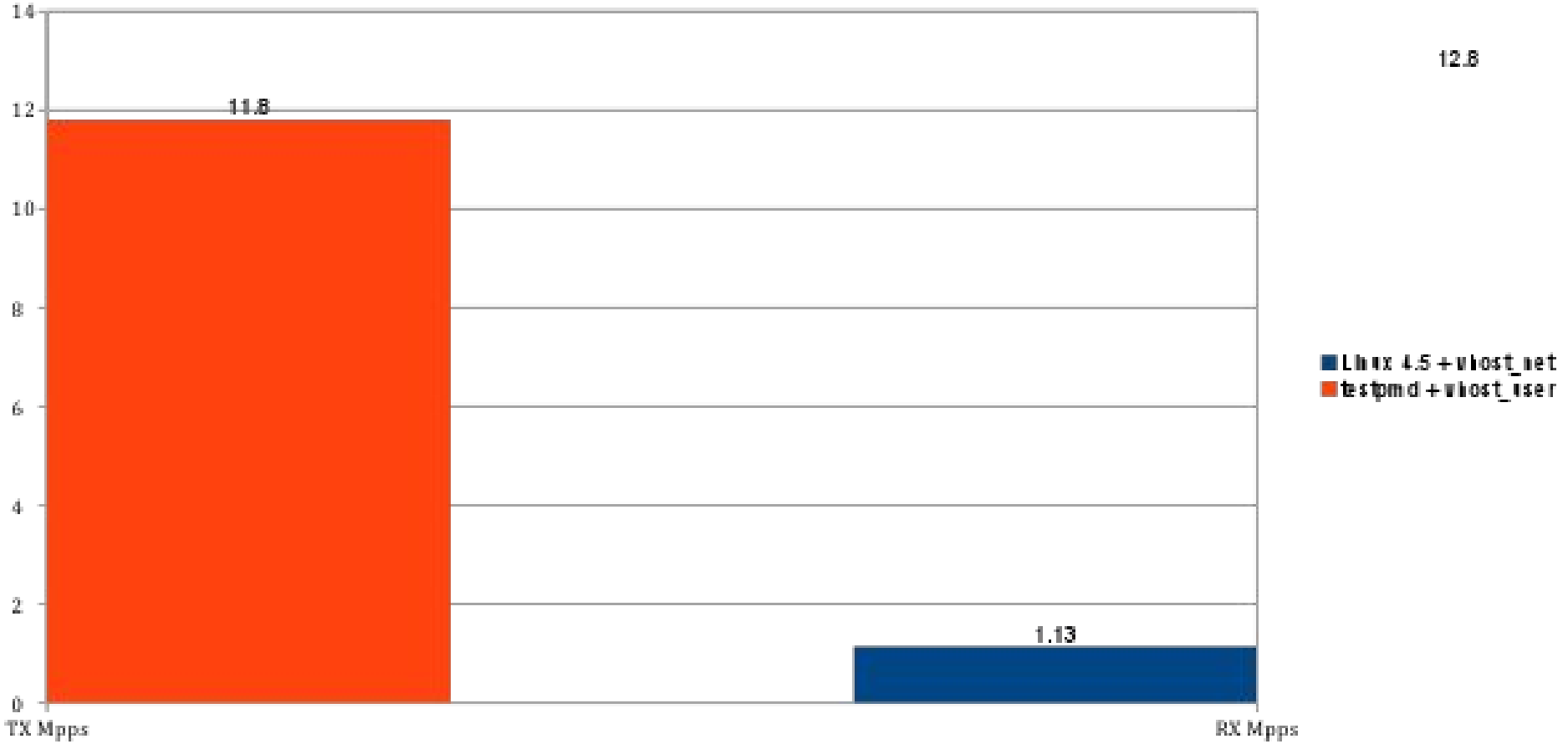
Performance Improvements of Virtual Machine Networking

Jason Wang
jasowang@redhat.com

Typical setup



How slow were we?

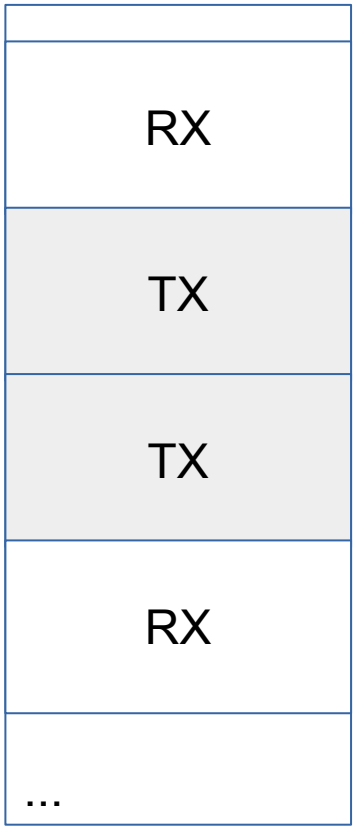


Agenda

- Vhost threading model
- Busy polling
- TAP improvements
- Batching virtio processing
- XDP
- Performance Evaluation
- TODO

Threading model

- one kthread worker for both RX and TX
- half duplex
- degradation on heavy bi-directional traffic
  - more devices since we are virt
  - Complexity for both management and application
- Scale?



Vhost_net kthread

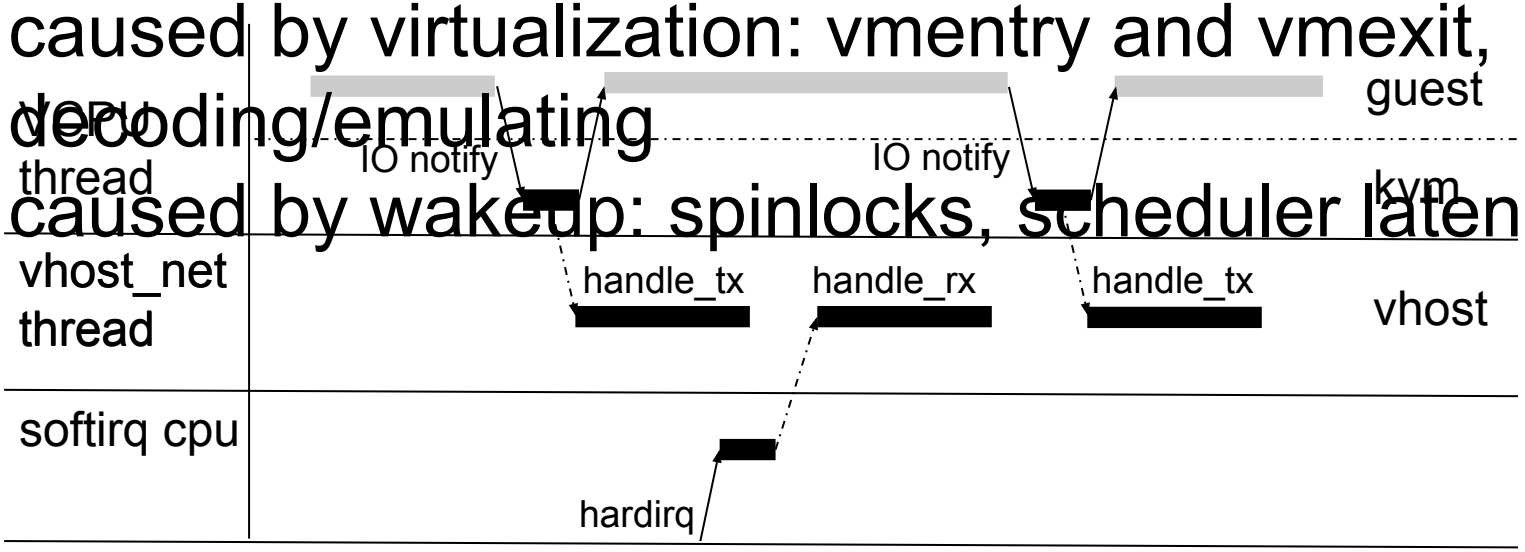
New models

-
- ELVIS by Abel Gordon
  - Dedicated cores for vhost
  - Several devices shares a single vhost worker thread
  - Polling and optimization on interrupt
  - Dedicated I/O scheduler
  - Lack of cgroup support
- CMWQ by Bandan Das
  - All benefits from CWMQ, e.g NUMA, dynamic workers
  - can be cgroup aware but expensive

Busy Polling

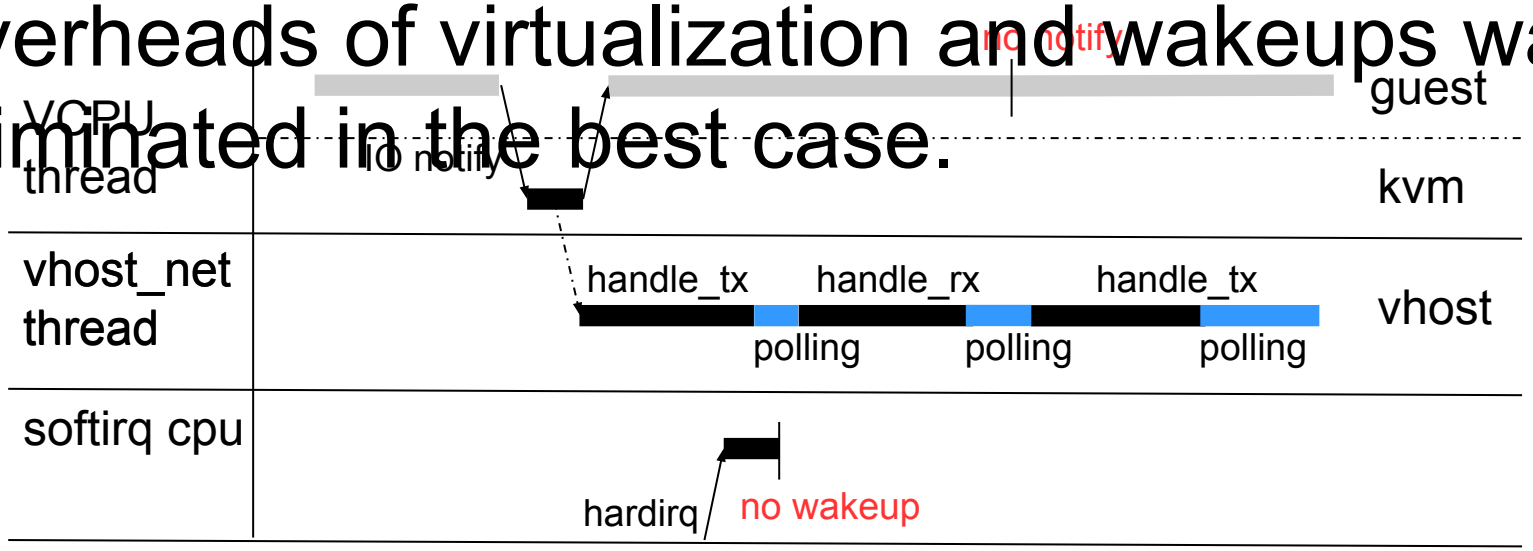
Event Driven Vhost

- vhost_net is driven by events:
  - virtqueue kicks: tx and rx
  - socket events: new packets arrived and sndbuf available
- overheads
  - caused by virtualization: vmentry and vmexit, decoding/emulating
  - caused by wakeup: spinlocks, scheduler latency



Limited busy polling (since 4.6)

- still driven by events but busy poll for a while if nothing to do
  - maximum us spent on busy polling is limited by userspace
  - disable events and poll the sources
- overheads of virtualization and wakeups was eliminated in the best case.



Limited busy polling (since 4.6)

- Exit the busy polling loop also when
  - signal is pending
  - TIF_NEED_RESCHED was set
- 1 byte TCP_RR shows 5%-20% improvements
- Issues
  - Not a 100% busy polling implementation
    - This could be done by specifying a very large poll-us
    - still some limitation caused by sharing kthread model
- Sometime user want a balance between latency and cpu consumption

TAP improvements

socket receive queue

- TAP use double linked list (sk_receive_queue) before 4.8
  - cache threshing
    - Every user has to write to lots of places
    - Every change has to be made multiple places
  - Spinlock is used for synchronization between producer and consumer

```
static inline void __skb_insert(struct sk_buff *newsk,
                                struct sk_buff *prev, struct sk_buff *next,
                                struct sk_buff_head *list)
{
        newsk->next = next;
        newsk->prev = prev;
        next->prev  = prev->next = newsk;
        list->qlen++;
}
```

ptr_ring (since 4.8)

- cache friendly ring for pointers (Michael S. Tsirkin)
  - an array of pointers
    - NULL means valid, !NULL means invalid
    - consumer and producer verify against NULL, no need to read the index of each other, no barrier needed
    - no lock contention between producer and consumer

```
struct ptr_ring {
    int producer ____cacheline_aligned_in_smp;
    spinlock_t producer_lock;
    int consumer ____cacheline_aligned_in_smp;
    spinlock_t consumer_lock;
    /* Shared consumer/producer data */
    /* Read-only by both the producer and the consumer */
    int size ____cacheline_aligned_in_smp; /* max entries in queue */
    void **queue;
};
```

producer only

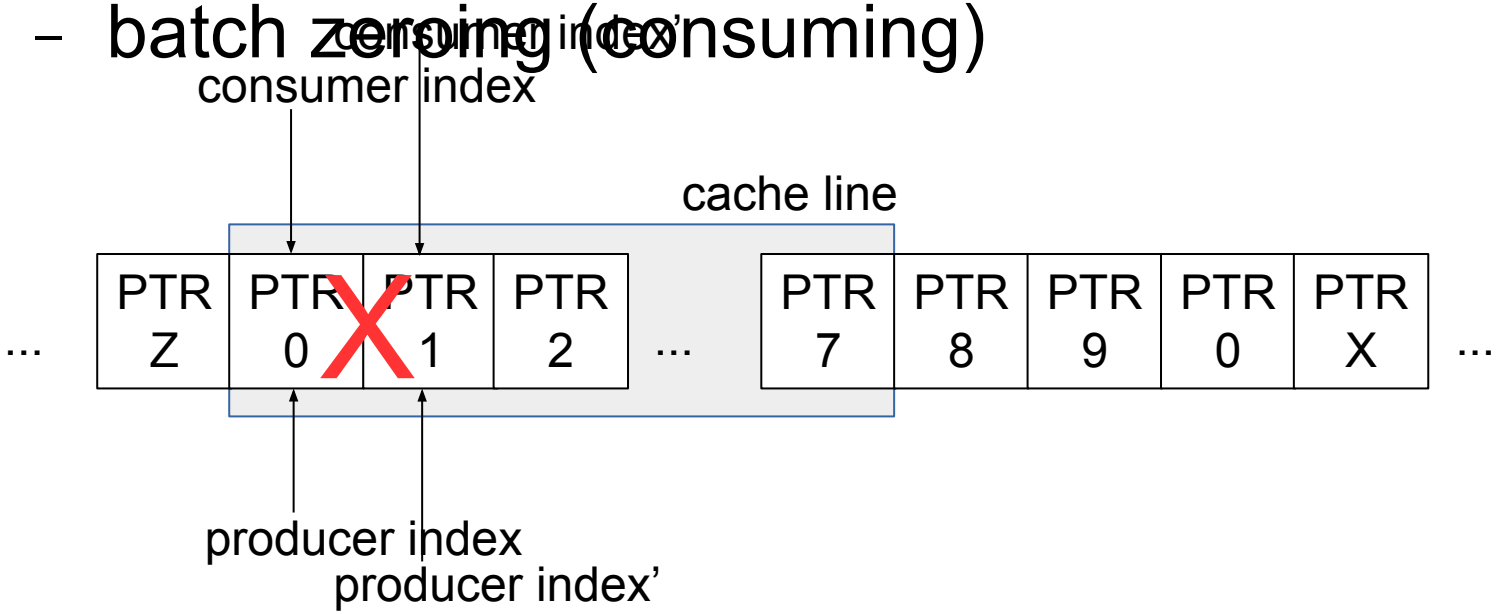
consumer only

skb_array (since 4.8)

- wrapper for storing pointers to skb
- sk_receive_queue was replaced by skb_array
- 15.3% RX pps was measured in guest during unit-test

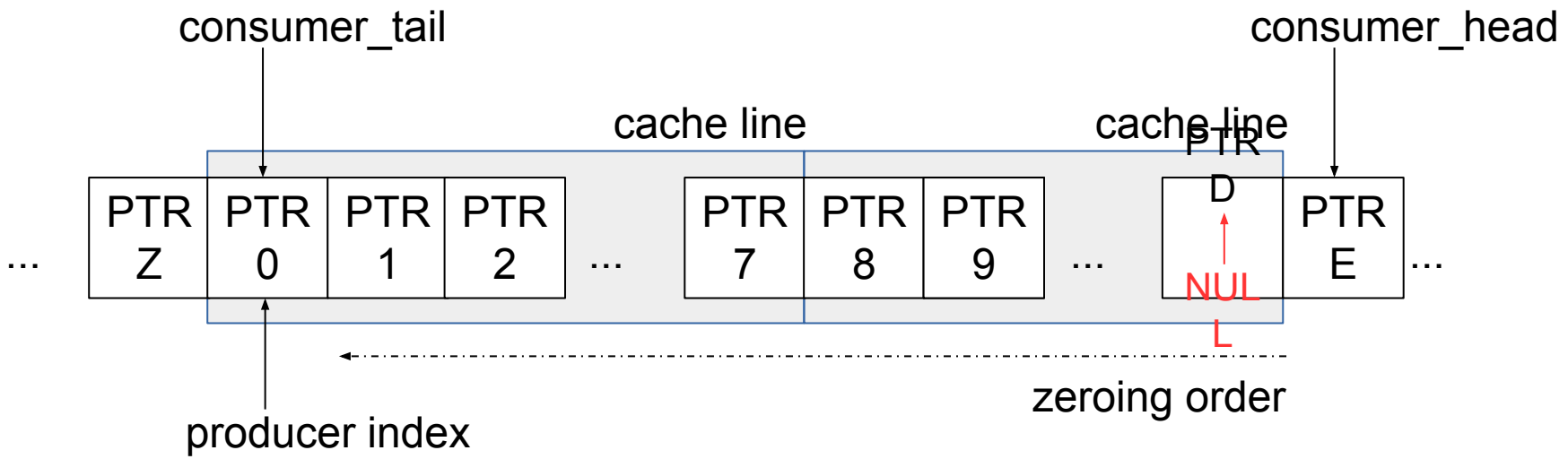
issue of slow consumer

- if consumer index advances one by one
  - producer and consumer are in the same cache line
  - cache line bouncing almost for each pointer
- Solution
  - batch zeroing (consuming)



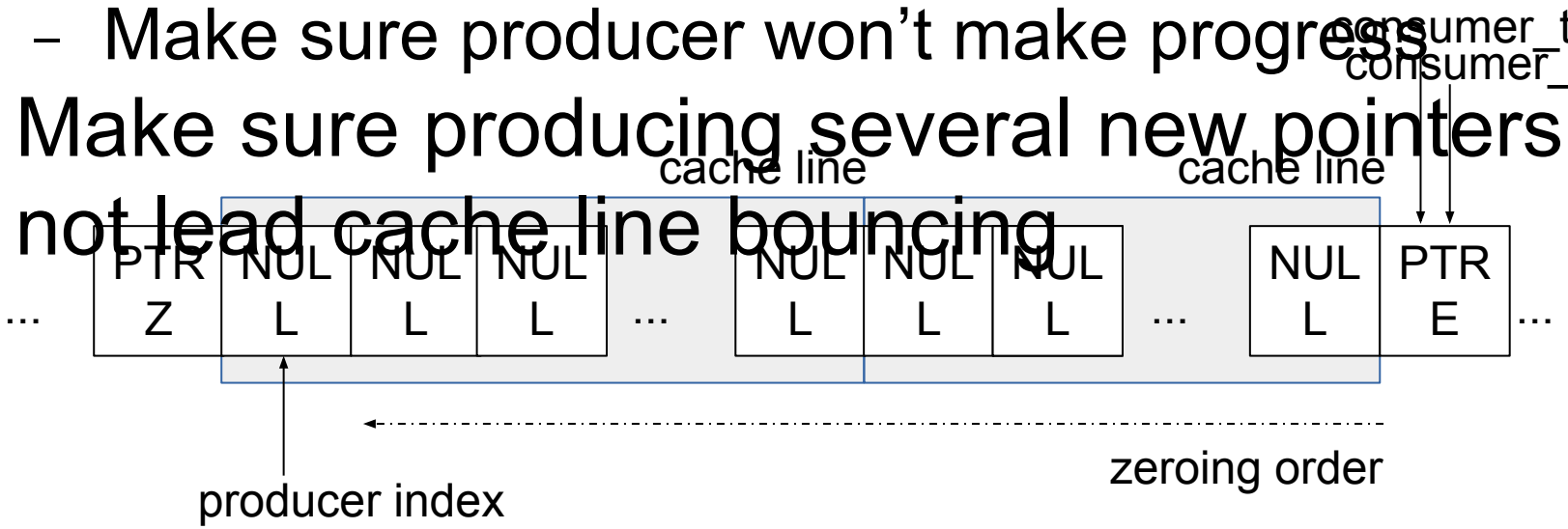
Batch zeroing (since 4.12)

```
struct ptr_ring {
        ...
        int consumer_head ____cacheline_aligned_in_smp; /* next valid entry */
        int consumer_tail; /* next entry to invalidate */
        ...
        int batch; /* number of entries to consume in a batch */
        void **queue;
};
```



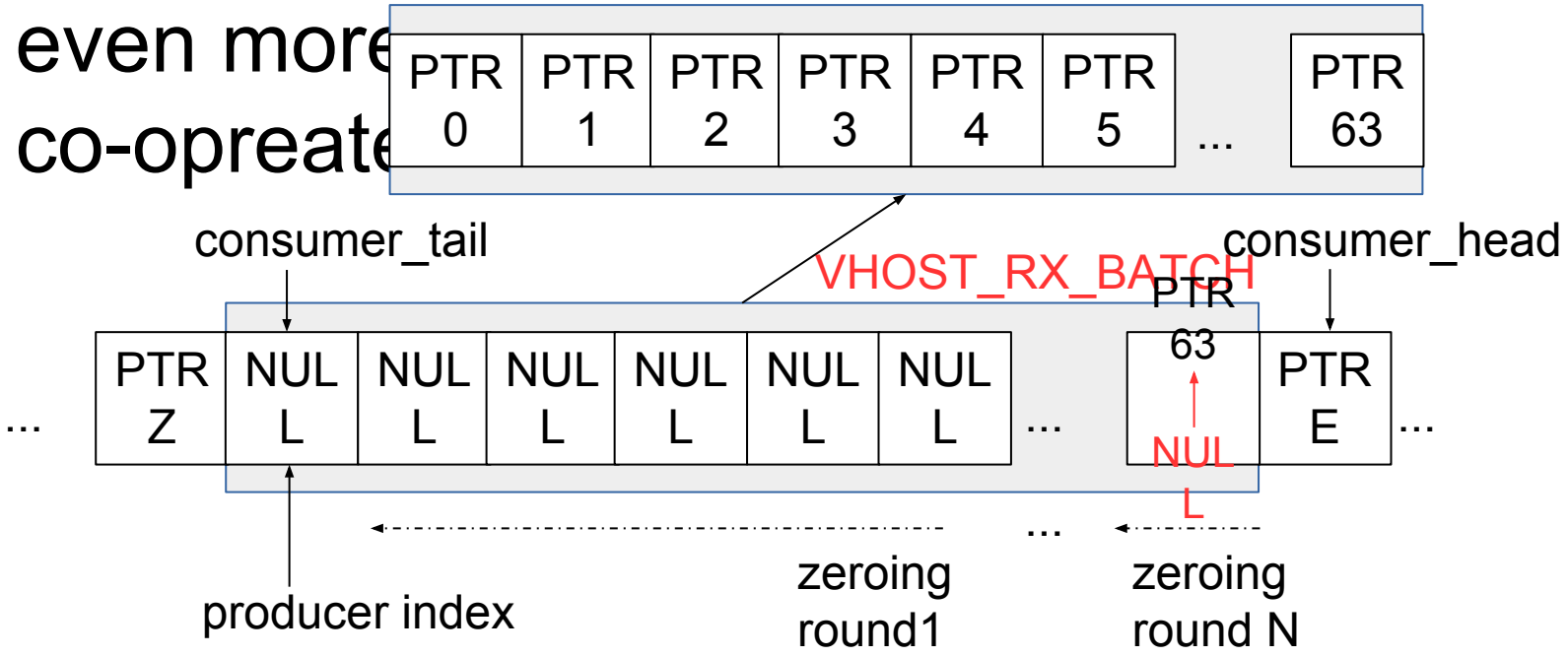
Batch zeroing (since 4.12)

- Start to invalidate consumed pointers only when consumer is 2x size of cache line far from producer
- Zeroing in the reverse order
  - Make sure producer won't make progress
- Make sure producing several new pointers does not lead cache line bouncing
-



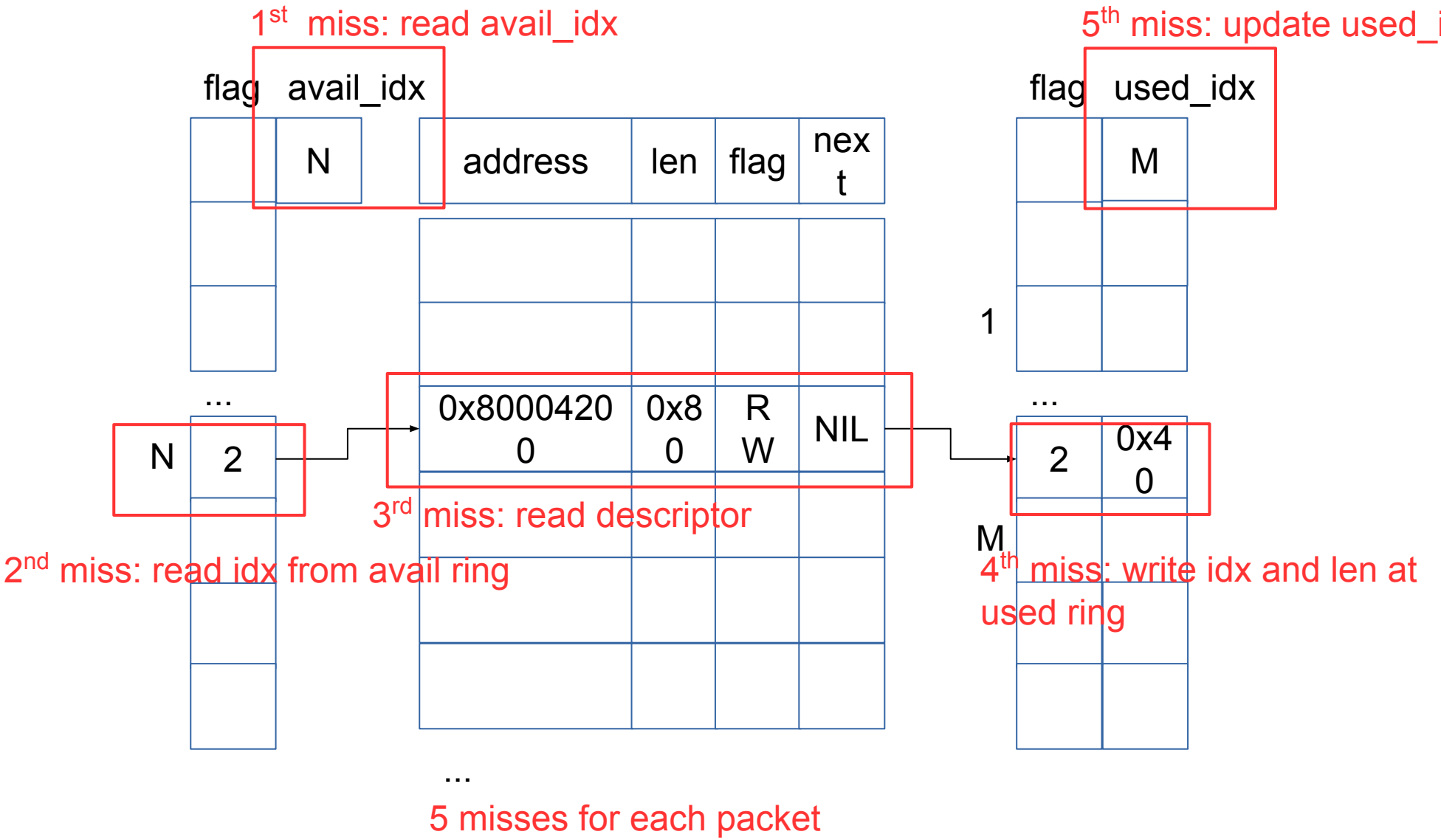
Batch dequeuing (since 4.13)

- consumer the pointers in a batch, pointer access is lock free afterwards
- reduce the cache misses and keep consumer even more
- co-opreate
-

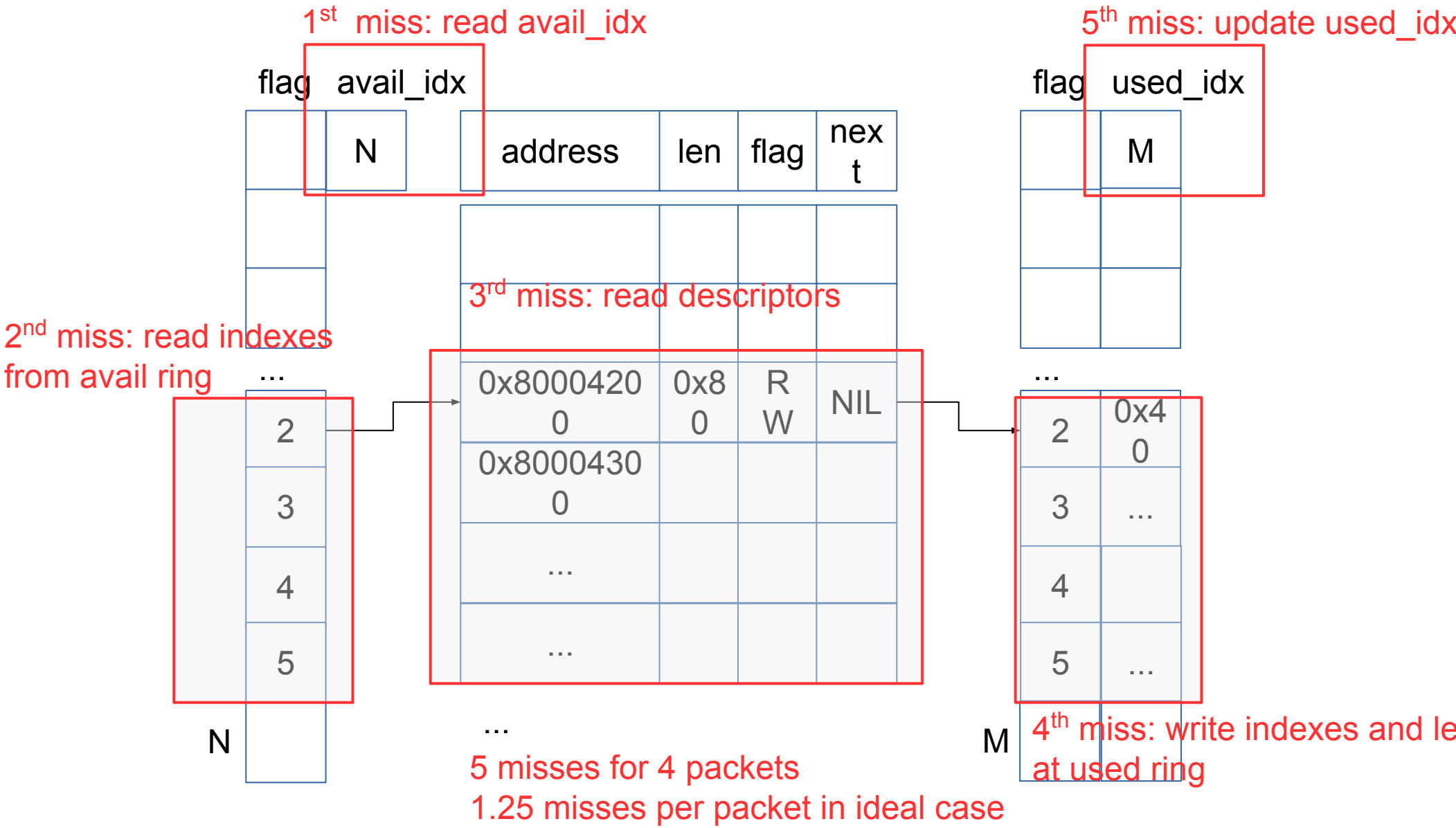


Batching for Virtio

Virtqueue and cache misses



How batching helps



Batching (WIP)

- Reduce cache misses
- Reduce cache threshing
  - When ring in almost empty or full
  - Device or driver won't make progress when avail idx or used idx changes
    - Cache line contention on avail, used and descriptor ring was mitigated
- Fast string copy function
  - Benefit from modern CPU

Batching in vhost_net (WIP)

- Prototype:
  - Batch reading avail indexes
  - Batch update them in used ring
  - Update used idx once for a batch
- TX get ~22% improvements
- RX get ~60% improvements
- TODO:
  - Batch descriptor table reading

XDP

Introduction to XDP

- short for eXpress Data Path
- work at early stage on driver rx
  - before skb is created
- Fast
  - page level
  - driver specific optimizations (page recycling ...)
- Programmable
  - eBPF
- Actions
  - DROP, TX, PASS, **REDIRECT**

Typical XDP implementation

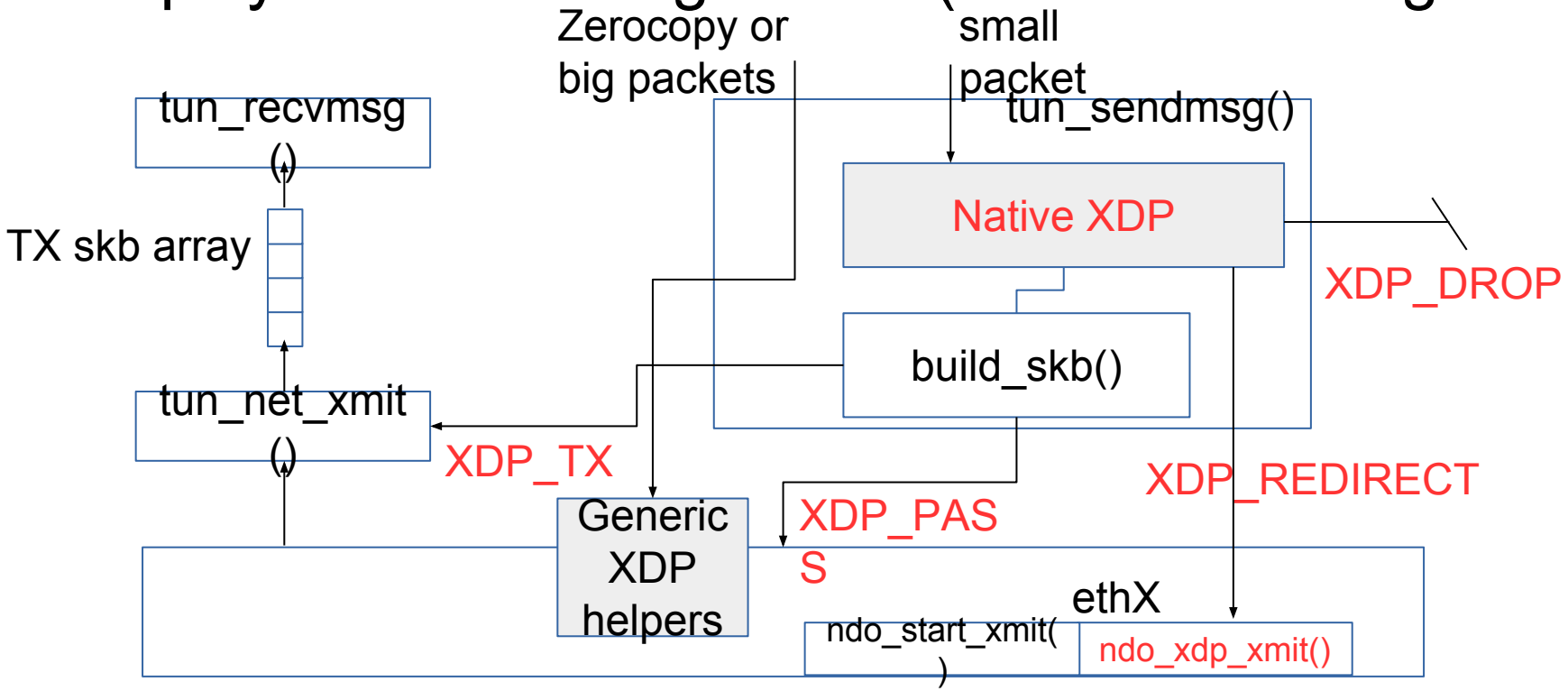
- Typical Ethernet XDP support
  - Dedicated TX queue for lockless XDP_TX
    - per CPU or paired with RX queue
    - Multiqueue support is needed
      - Adding/removing queues when XDP is set/unset
  - Run under NAPI poll routine
    - after DMA is done
  - Don’t support large packets
    - JUMBO/LRO/RSC needs to be disabled during XDP set
- But TAP is a little bit different

XDP for TAP (since 4.13)

- Challenge for TAP
  - Multiqueue is controlled by userspace:
    - solution: No dedicated TX queue, sharing TX queue
    - work even for single queue TAP
  - Changing LRO/RSC/Jumbo configuration:
    - solution: Hybird mode XDP implementation
  - Datacopy was done with skb allocation:
    - solution: Decouple data copy out of skb allocation, build_skb()
  - No NAPI by default:
    - run inside tun_sendmsg()
  - Zerocopy:
    - done through Generic XDP, adjust_head

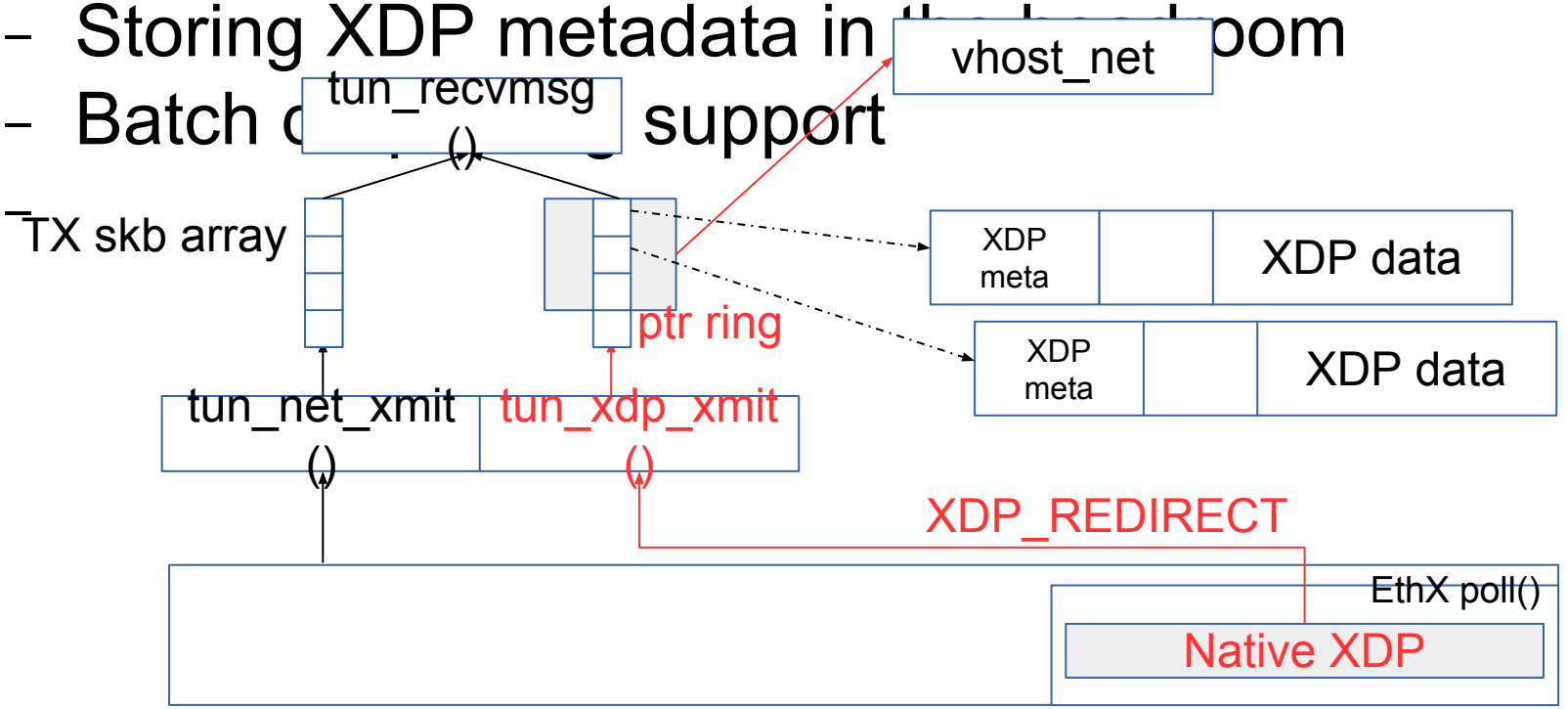
Hybrid XDP in TAP (since 4.13)

- Merged in 4.13
  - mix using native XDP and skb XDP
  - simplify the VM configuration (no notice from guest)



XDP transmission for TAP (WIP)

- For accelerating guest RX
  - An XDP queue (ptr_ring) is introduced for each tap socket
  - Storing XDP metadata in the headroom
  - Batch dequeuing support



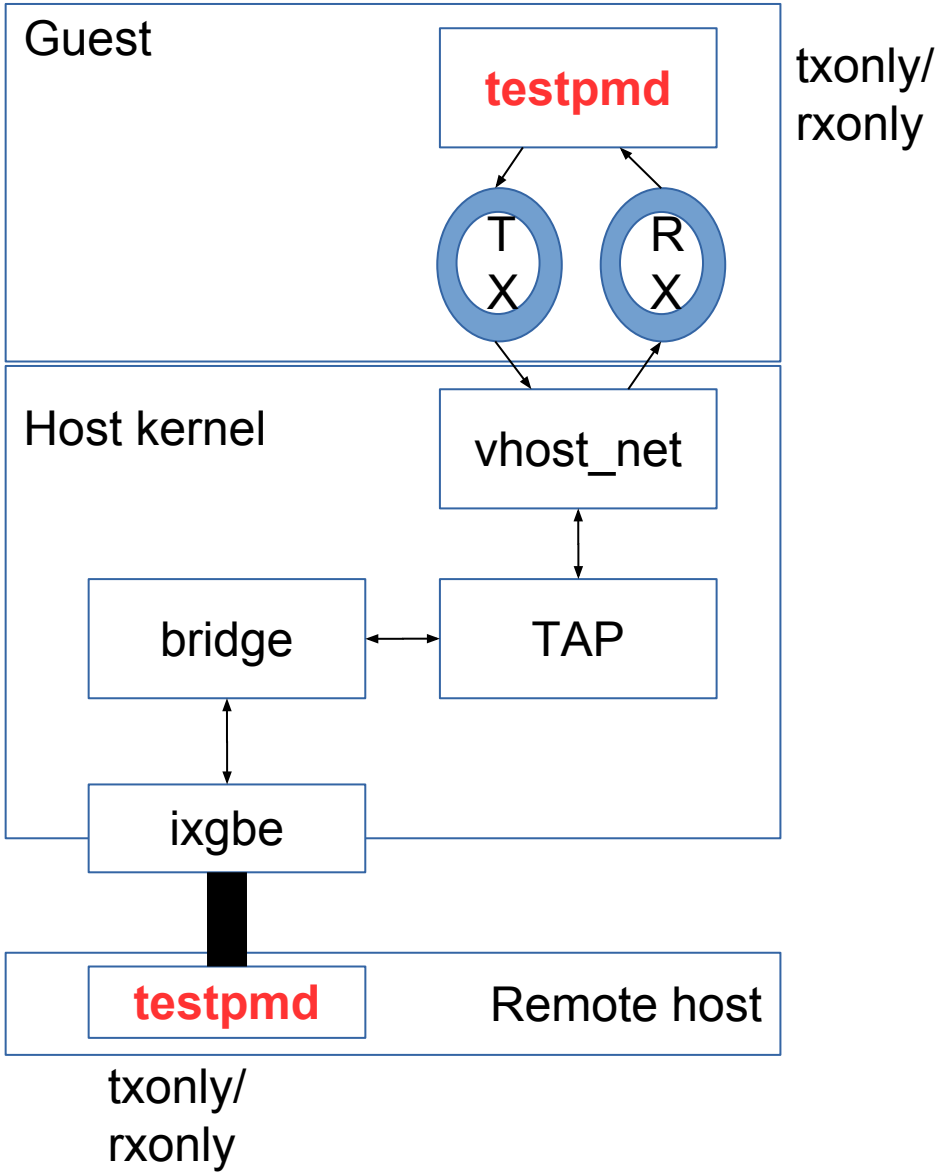
XDP for virtio-net (since 4.10)

- Multiqueue based
  - Per CPU TX XDP queue
  - Need reserve enough queue pairs during VM launching
- OFFLOADS were disabled on set on demand
- No reset
  - Copy the packet if headroom is not enough
    - A little bit slow but should be rare
- Support XDP redirecting/transmission
  - Since 4.13
- No page recycling yet

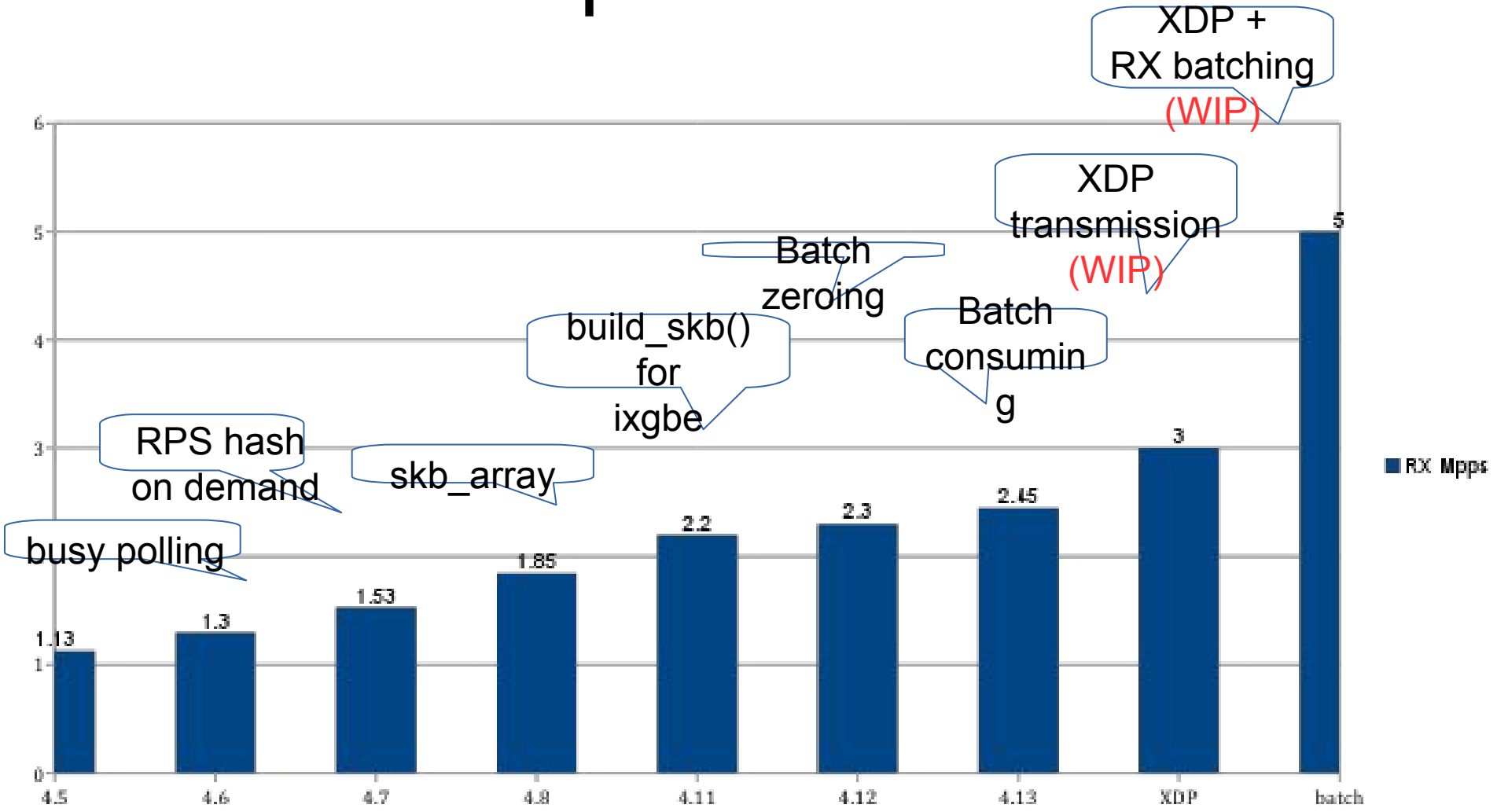
Performance Evaluation

Test setup bridge

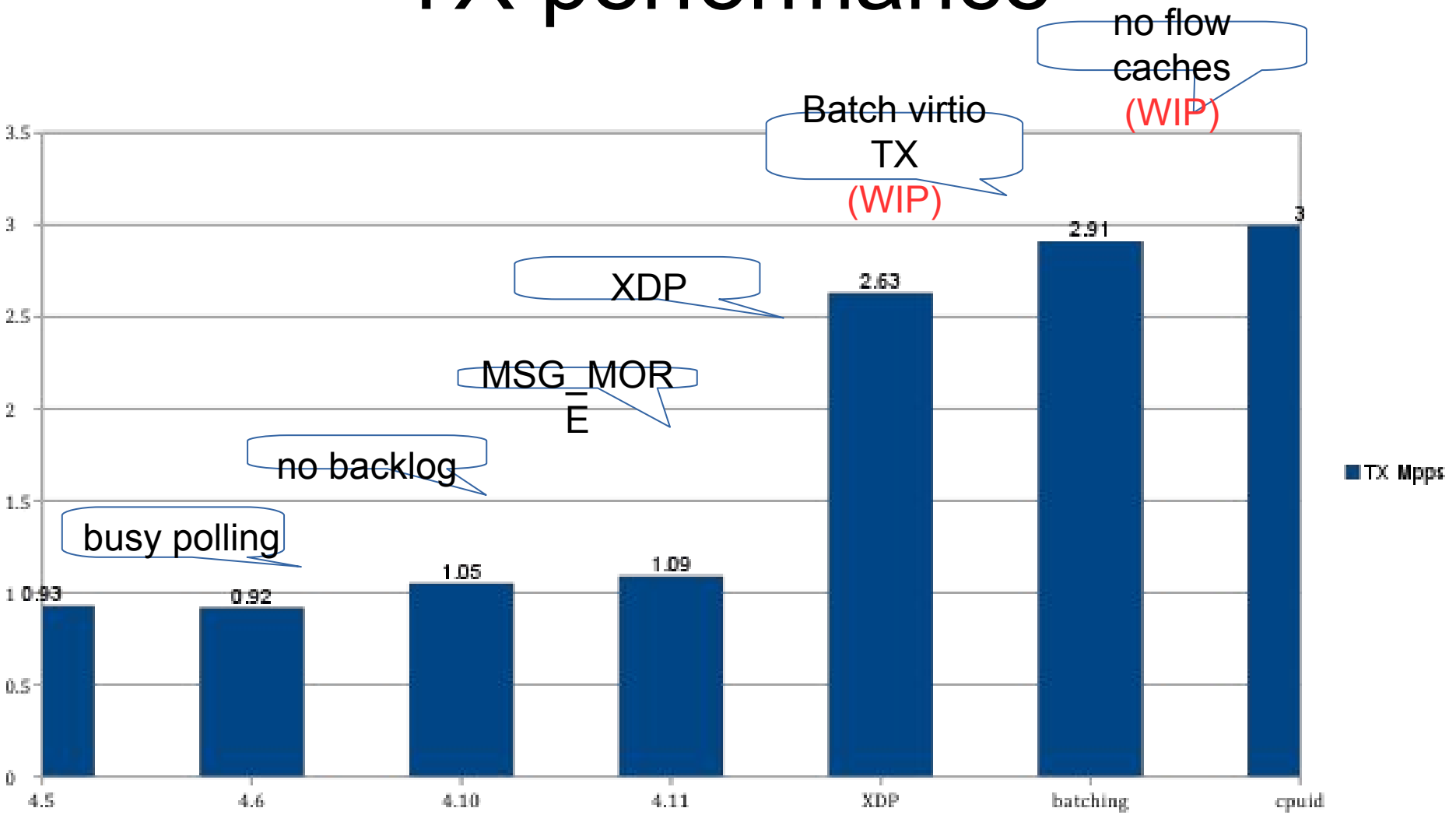
- Two Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
- Back to back ixgbes
- Testpmd is used:
  - traffic generator and receiver
    - 30% faster than pktgen
  - No interrupt
  - Busy polling
- Tx and rx was measured separately



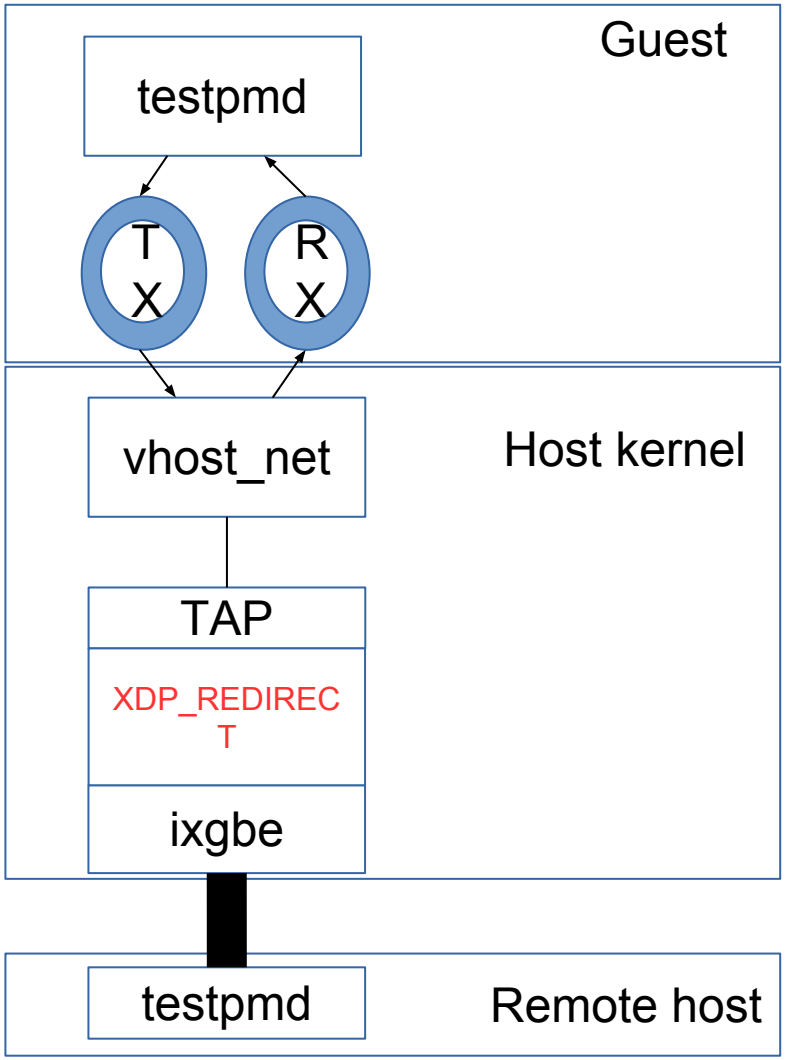
RX performance

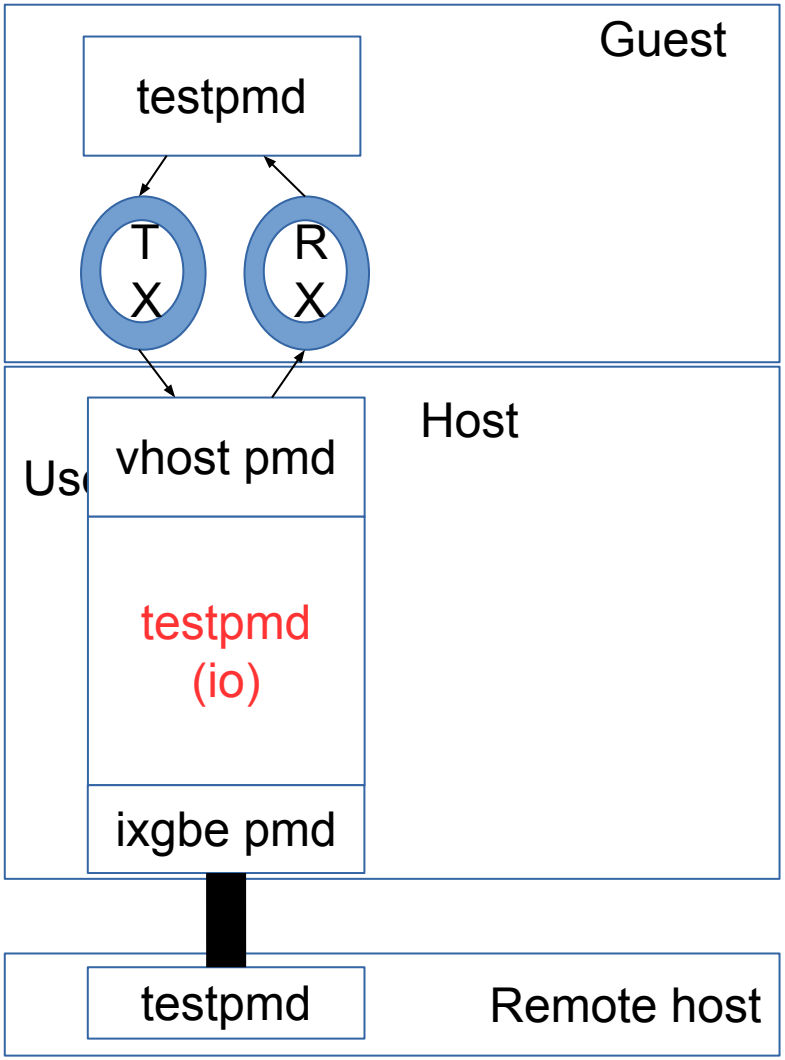


TX performance

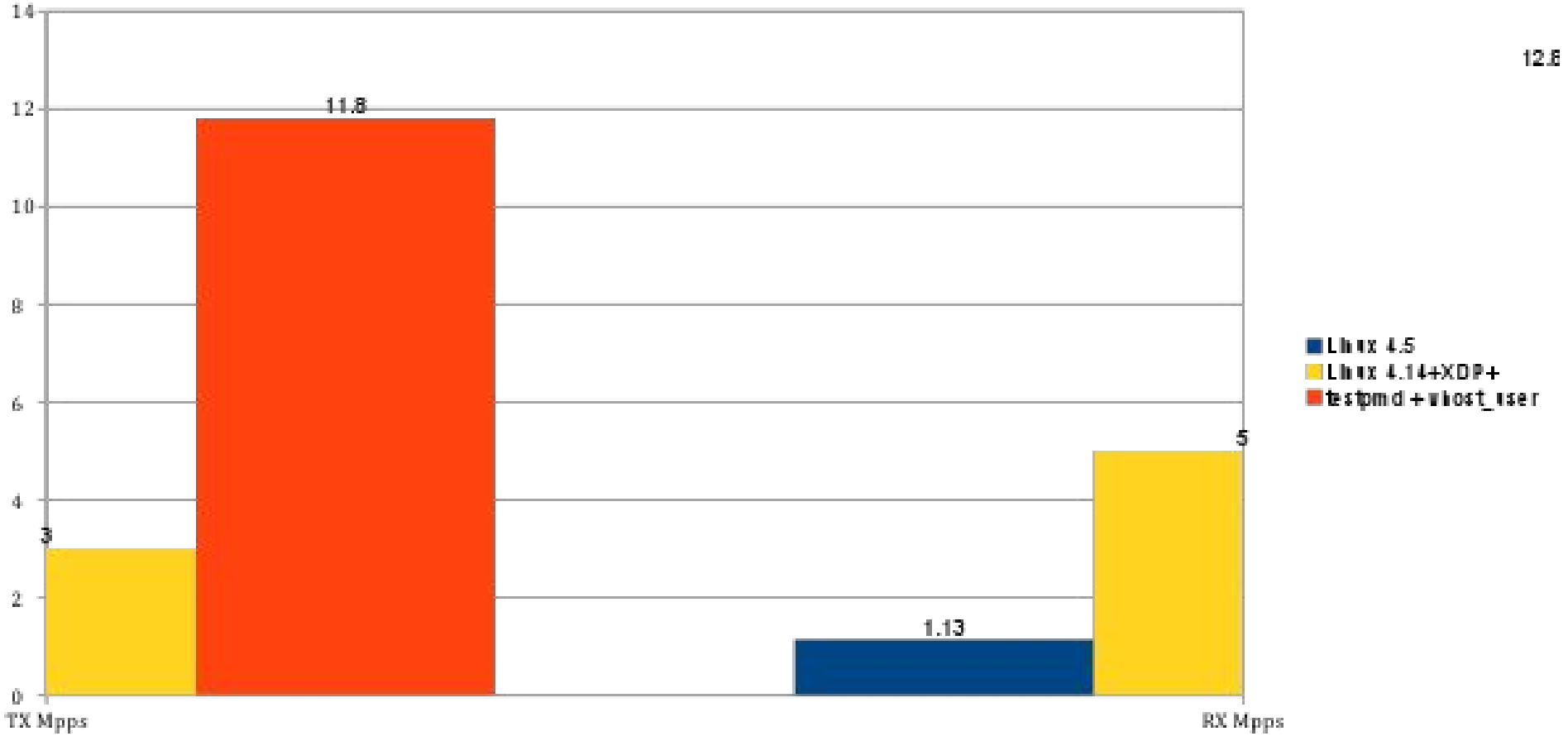


XDP vs testpmd





Here we are


14
12
10
8
6
4
2
0
11.8
3
1.13
5
12.8
TX Mpps
RX Mpps
Linux 4.5
Linux 4.14+XDP+
testpmd+vhost_user

perf – ksoftirqd RX

- 26.49% [kernel] [k] _raw_spin_lock
- 16.00% [ixgbe] [k] ixgbe_clean_rx_irq
- 15.99% [kernel] [k] sock_def_readable
- 5.63% [kernel] [k] dev_get_by_index_rcu
- 5.48% [kernel] [k] __bpf_tx_xdp
- 4.42% [tun] [k] tun_xdp_xmit
- 4.29% [kernel] [k] xdp_do_redirect
- 3.70% [ixgbe] [k] ixgbe_alloc_rx_buffers
- 2.53% [kernel] [k] swiotlb_sync_single
- 2.08% [kernel] [k]

perf – vhost_net RX

- 43.38% [vhost_net] [k] handle_rx
- 9.86% [kernel] [k] copy_page_to_iter
- 8.87% [kernel] [k] _copy_to_iter
- 7.41% [vhost_net] [k] vhost_net_buf_peek
- 6.38% [vhost] [k] __vhost_get_vq_desc
- 6.22% [kernel] [k] iov_iter_advance
- 6.16% [kernel] [k] copy_user_generic_unrolled
- 3.80% [vhost] [k] vhost_get_vq_desc
- 3.64% [vhost] [k] translate_desc
- 2.40% [kernel] [k] copyout

perf – vhost_net TX

- 21.49% [vhost] [k] translate_desc
- 13.41% [tun] [k] tun_get_user
- 10.12% [vhost] [k] __vhost_get_vq_desc
- 6.54% [kernel] [k] iov_iter_advance
- 4.32% [kernel] [k] copy_page_from_iter
- 4.15% [kernel] [k] copy_user_enhanced_fast_string
- 3.92% [ixgbe] [k] ixgbe_xmit_xdp_ring.isra.88
- 3.56% [vhost_net] [k] handle_tx
- 3.46% [tun] [k] tun_sendmsg

TODO/Raw ideas

- Raw ideas
  - better integration with NAPI busy polling in vhost_net?
  - pure busy polling vhost_net?
  - Better XDP co-operation on page recycling for hardware NIC drivers?
  - Build and receive skb/XDP in vhost_net?
  - Rx zerocopy
    - ndo_post_rx_buffer()?
- Please comment on virtio 1.1

Thanks