AF_PACKET V4 AND PACKET_ZEROCOPY

Magnus Karlsson and Björn Töpel, Intel

John Fastabend, Covalent IO

Legal Disclaimer

- Intel technologies may require enabled hardware, specific software, or services activation. Check with your system manufacturer or retailer.
- No computer system can be absolutely secure.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
- Cost reduction scenarios described are intended as examples of how a given Intel- based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.
- All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
- Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.
- Intel, the Intel logo, and other Intel product and solution names in this presentation are trademarks of Intel.
- *Other names and brands may be claimed as the property of others.
-

Motivation & Problem Statement

- Lots of good features
- AF_PACKET performance does not meet application requirement

DPDK

Proprietary HW SDK

PF_RING

Netmap

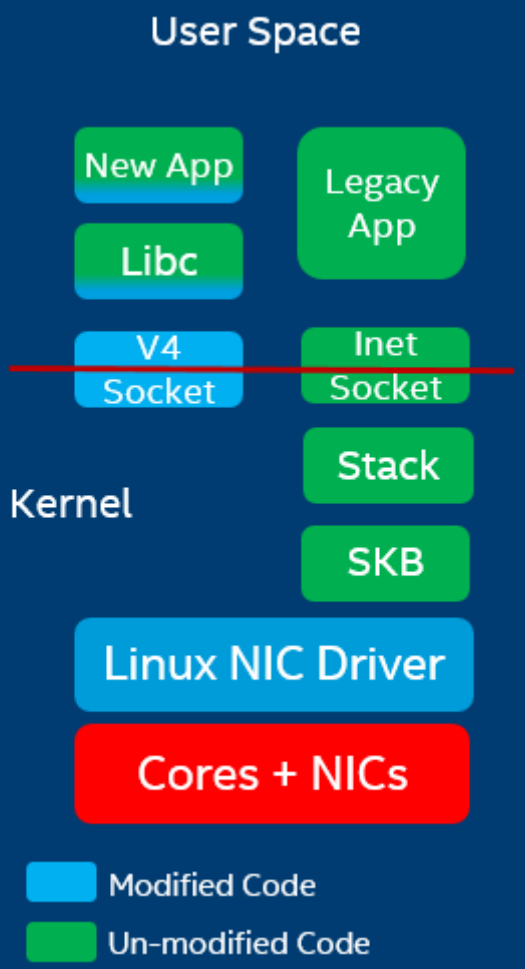
RDMA

- High networking performance
- Hard to use
- Might lack lots of features
- Might have little to no integration with Linux
- Not part of Linux net subsystem in kernel.org

How can we combine the functionality and ease-of-use of AF_PACKET sockets with the networking performance of these other solutions?

Proposed Solution

- New fast packet interfaces in Linux
  - AF_PACKET V4
  - No system calls in data path
  - Copy-mode by default
  - True zero-copy mode with PACKET_ZEROCOPY, DMA packet buffers mapped to user space
  - HW descriptors only mapped to kernel
- ZC mode requires HW steering support for untrusted applications
  - Copy required otherwise
- Goal is to hit 40 Gbit/s line rate on a single core for large packets and 30 Gbit/s for 64 byte packets



Results Summary

- Implemented V4 in af_packet.c
  - Two new NDOs need to be implemented for PACKET_ZEROCOPY
  - Introduced packet arrays to facilitate implementation
  - Also gives you XDP support with ZC mode for free
- V4 + PACKET_ZEROCOPY 6-40x the throughput of V2 and V3 on an I40E NIC
  - 40 Gbit/s line rate for RX on one core for large packets
  - TX and smaller packets not at line rate yet
  - Optimization work required
- Should lessen the need for SR-IOV

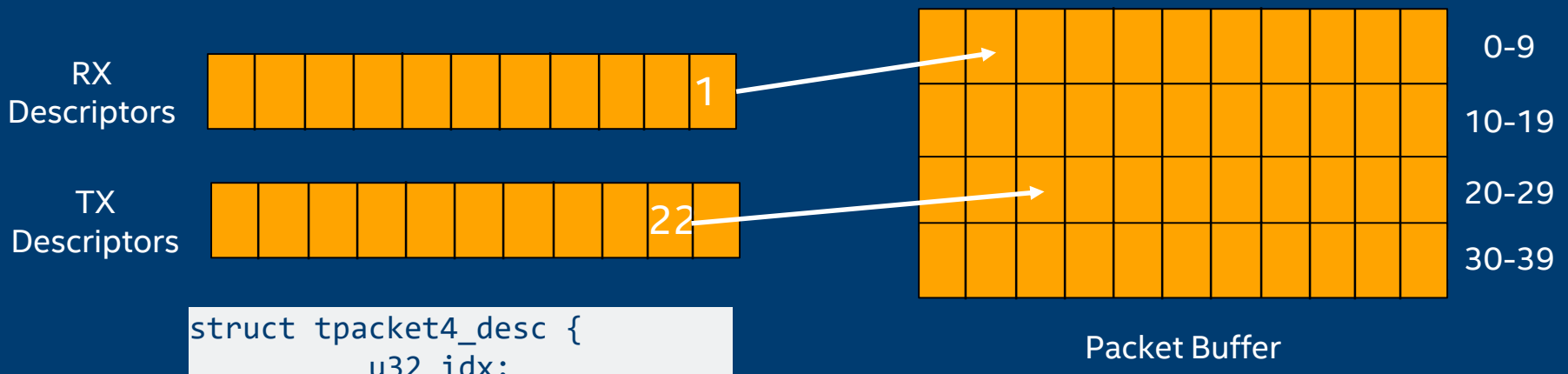
Outline

- AF_PACKET V4
- PACKET_ZEROCOPY
- Implementation with Packet Arrays
- XDP Integration
- Performance results
- Future work
- Conclusions

Motivation AF_PACKET V4

- Support true zero-copy
- Eliminate copies for TX and buffering
- Transparrent error reporting on every packet, if desired
- Faster than V2 and V3
- Integrated with XDP
- If you implement ZC in a driver you should get XDP ”for free”

AF_PACKET V4 Format




```
struct tpacket4_desc {
        __u32 idx;
        __u32 len;
        __u16 offset;
        __u8  error;
        __u8  flags;
        __u8  padding[4];
};
```

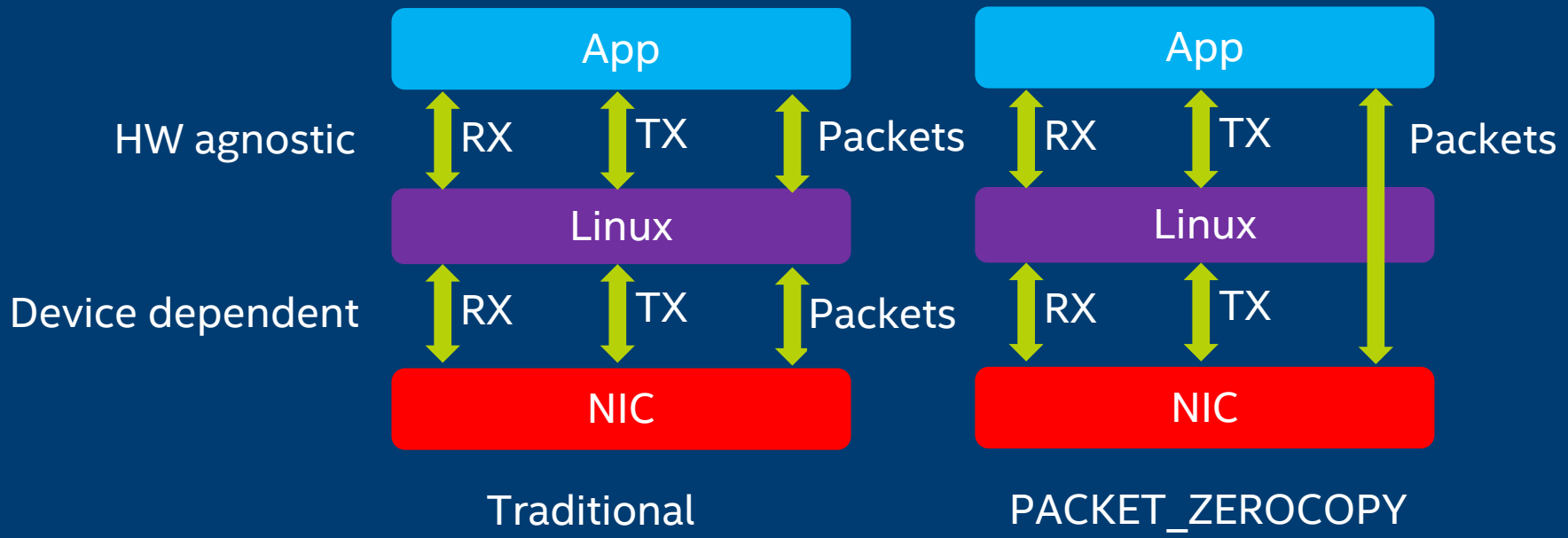
```
struct tpacket4_queue {
        struct tpacket4_desc *ring;

        unsigned int avail_idx;
        unsigned int last_used_idx;
        unsigned int num_free;
        unsigned int ring_mask;
};
```

- 4 descriptors on a 64 byte cache line
- There is NO data header in V4 for performance reasons
- RX and TX can share the same packet buffer

How to Use It?

```
sfd = socket();
setsockopt(sfd, SOL_PACKET, PACKET_VERSION, PACKET_V4, ....);
setsockopt(sfd, SOL_PACKET, PACKET_MEMREG, &req, sizeof(req));
setsockopt(sfd, SOL_PACKET, PACKET_RX_RING, &req, sizeof(req));
setsockopt(sfd, SOL_PACKET, PACKET_TX_RING, &req, sizeof(req));
bind(sfd, ....”/dev/eth0”....);
setsockopt(sfd, SOL_PACKET, PACKET_ZEROCOPY, queue_pair, sizeof(int));
for (;;) {
        read_messages(sfd, msgs, ....);
        process_messages(msgs);
        send_messages(sfd, msgs, ....); }
```

PACKET_ZEROCOPY: Basic Principle



- Application still HW agnostic with PACKET_ZEROCOPY
- Each application gets its own packet buffer and tx/rx queue pair
  - Packet buffers can be shared if desired

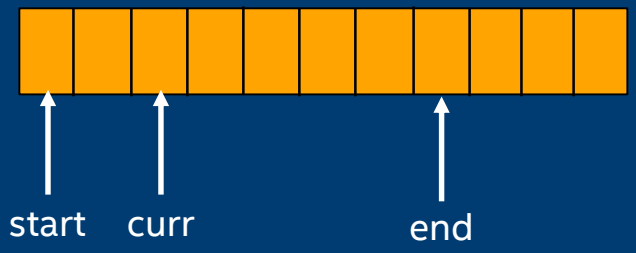
Security and Isolation Requirements for ZC

- Important properties:
  - User space cannot crash kernel or other processes
  - User space cannot read or write any kernel data
  - User-space cannot read or write any packets from other processes unless packet buffer is explicitly shared
- Requirement for untrusted applications:
  - HW packet steering, when there are packets with multiple destinations arriving on the same interface
  - If not available => kernel needs to own packet buffer and copy out data to correct destination. Not true zero-copy anymore

Implementation Goals

- Making the implementation of ZC in the driver simple
- To abstract away the V4 descriptor format
  - Same ZC driver code for SKBs, V2, V4, virtio-net, etc.
- To get XDP support for free when implementing ZC

Packet Arrays




```
In the control path:
rxa = tp4a_rx_new(rx_opaque, nb_elems, dev);
txa = tp4a_tx_new(tx_opaque, nb_elems, dev);

In the data path:
tp4a_populate(rxa);
while (tp4a_next_frame(rxa, p)) {
       tp4f_set_frame(p, len, offset, eop);
}
tp4a_flush(rxa);
```

- tp4a_* functions operate on packet arrays
- tp4f_* functions operate on frame sets
  - Frame set can be one or more frames representing zero or more packets
- Also used in V4 af_packet.c code

Implementation Example: I40E

RX

```
After RX IRQ:

while (tp4a_next_frame(rxa, p)) {
  if (out_of_buffers_next_itr)
     tp4a_populate(rxa);
  dma_sync(p);
  tp4f_set_frame(p, len, offset,
                 eop);
}
tp4a_flush(rxa);
```

TX

```
In send syscall path:

tp4a_populate(txa);
while (tp4a_next_frame(txa, p)) {
  if (no_space_on_tx_queue) {
     tp4a_return_packet(txa, p);
     break;
  }
  write_hw_tx_desc(p);
}

After TX IRQ:

tp4a_get_flushable_frame_set(txa, p);
while (tp4a_next_frame(txa, p)) {
  clean_up_tx_hw_desc(p);
}
tp4a_flush(txa);
```

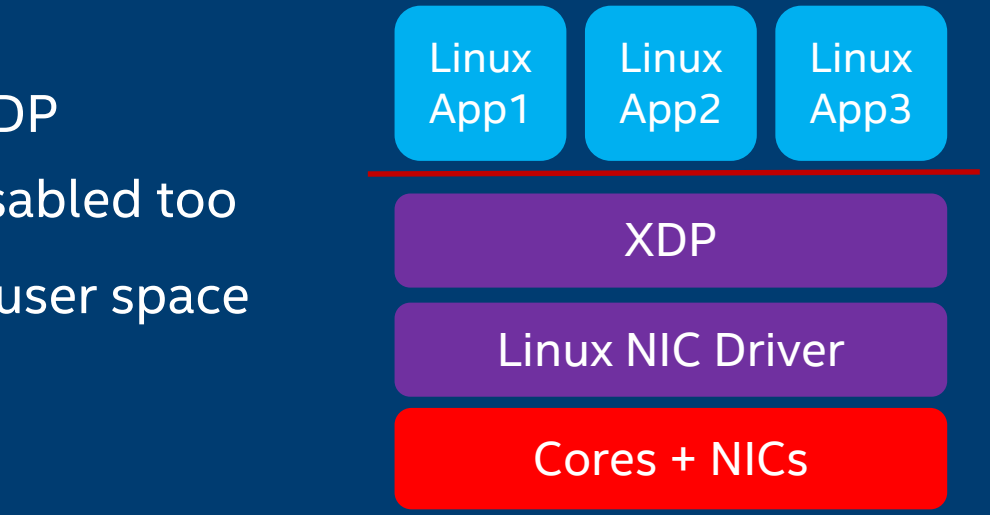
Implementation Example: veth

```
tp4a_populate(my_tp4a_tx);
tp4a_populate(other_process_tp4a_rx);

tp4a_copy_packets(other_process_tp4a_rx, my_tp4a_tx);

tp4a_flush(other_process_tp4a_rx);
tp4a_flush(my_tp4a_tx);
```

- This code handles SKB -> V4, V4 -> SKB as well as V4 -> V4
  - But this code is not in the current RFC ☹
  - Can also handle SKB -> SKB, but not efficiently. Better use existing path for that

XDP Support with Packet Arrays

- XDP is executed on tp4a_flush
  - Goal to get XDP support under ZC for free with packet arrays
  - RFC: still one extra call for XDP
  - Need support when ZC is disabled too
- XDP_PASS sends packet to V4 user space
  - Still zero copy



Experimental Setup

- Broadwell E5-2699 v4 @ 2.20GHz
- 2 cores used for benchmarks
- Rx is a softirq (thread)
- Tx is driven from application via syscall
  - TX and RX is currently in same NAPI context
  - Item in backlog to make this a thread on third core
- One VSI / queue pair used on FVL. 40Gbit/s interface
- Ixia load generator blasting at full 40 Gbit/s

Performance I40E 64-Byte Packets

	V2	V3	V4	V4 + ZC
rxdrop	0.67 Mpps	0.73 Mpps	0.74 Mpps	33.7 Mpps
txpush	0.98 Mpps	0.98 Mpps	0.91 Mpps	19.6 Mpps
l2fwd	0.66 Mpps	0.71 Mpps	0.67 Mpps	15.5 Mpps
tcpdump	-	0.74 Mpps	0.74 Mpps	14.1 Mpps

- Zero-copy 20x – 40x faster than previous best on Linux
- Copy mode a mixed bag
- Not optimized yet though
  - Still a syscall on TX
  - TX colocated with RX

"Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance/datacenter.

Performance I40E 1500-Byte Packets

	V2	V3	V4	V4 + ZC
rxdrop	0.56 Mpps	0.58 Mpps	0.66 Mpps	3.3 Mpps
txpush	0.81 Mpps	0.81 Mpps	0.88 Mpps	3.1 Mpps
l2fwd	0.55 Mpps	0.56 Mpps	0.62 Mpps	2.9 Mpps
tcpdump	-	0.62 Mpps	0.64 Mpps	3.3 Mpps

- Zero-copy 40 Gbits/s line rate for RX workloads
  - Not there yet for TX workloads
  - Goal is 40 Gbit/s line rate for all these workloads
- V4 copy mode around 10% faster than V2 and V3
  - Avoids copy on TX

"Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance/datacenter.

Discussion: Unifying XDP and ZEROCOPY

XDP

```
int (*ndo_xdp)(struct net_device *dev,
               struct netdev_xdp *xdp);

int (*ndo_xdp_xmit)
       (struct net_device *dev,
        struct xdp_buff *xdp);

void (*ndo_xdp_flush)
       (struct net_device *dev);
```

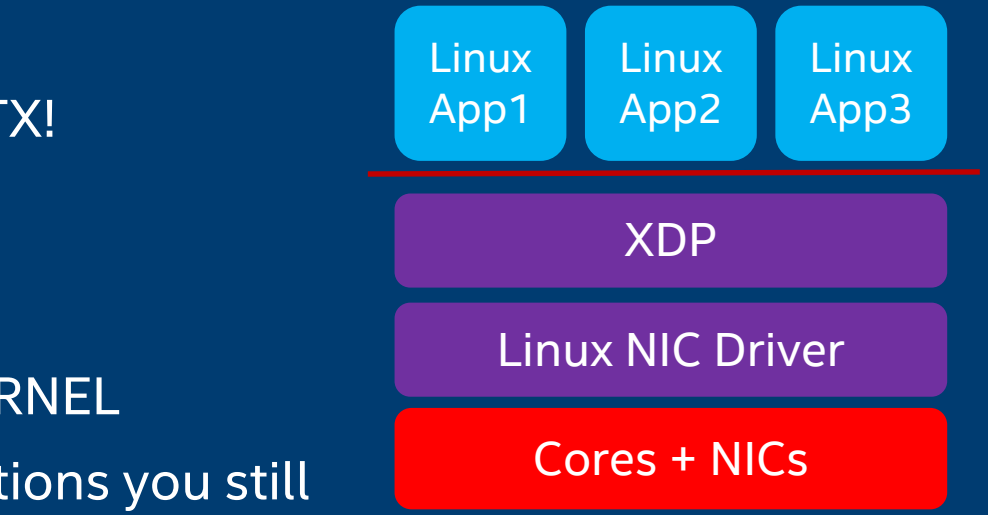
V4

```
int (*ndo_tp4_zerocopy)
       (struct net_device *dev,
        struct tp4_netdev_parms *parms);

int (*ndo_tp4_xmit)
       (struct net_device *dev,
        int queue_pair);
```

- Cumbersome to implement support for two techniques
- TX sides similar: V4 xmit = XDP xmit + XDP flush
- XDP support with packet arrays even when zero-copy is not enabled
  - Buffers only allocated from the normal allocator in driver
- XDP_REDIRECT needs a destructor for V4 to work in zero-copy mode
  - Currently a copy is needed ☹

Possible XDP Extensions with AF_PACKET V4

- Descriptor rewriting in zero-copy path
  - virtio-net support
  - V2 support?
  - Other formats?
  - Needs an XDP program for TX!
- Load balancing
  - More flexible than HW
- New action: XDP_PASS_TO_KERNEL
  - NOTE: for untrusted applications you still need HW packet steering
  - Per ring XDP program might help



RFC ToDo

- Investigate the user-space ring structure's performance problems
- Continue the XDP integration into packet arrays
- Optimize performance
- SKB <-> V4 conversions in tp4a_populate & tp4a_flush
- Packet buffer is unnecessarily pinned for virtual devices
- Support shared packet buffers
- Unify V4 and SKB receive path in I40E driver
- Support for packets spanning multiple frames
- Disassociate the packet array implementation from the V4 queue structure
- ...and all things you will detect!

Future Work

- Get ready for a proper patch set
- More performance optimization work
- Implement zero-copy support for other devices
  - Which ones?
- Try it out on real workloads
- Make send syscall optional and get TX off RX core
- Packet steering using XDP
- Metadata support, using XDP data_meta?

Acknowledgements

- Alexei Starovoitov, Alexander Duyck, and Jepser Dangaard Brouer for all your feedback on the early RFCs
- Rami Rosen, Jeff Shaw, Ferruh Yigit, and Qi Zhang for your help with the code, performance results and the paper
- The developers of RDMA, Netmap and PF_RING for the data path inspiration

Conclusions

- Introduced AF_PACKET V4 and PACKET_ZEROCOPY
- Packet arrays used to facilitate implementation
- Integrated with XDP
- V4 + zero-copy provides 6x to 40 x performance improvements compared to V2 and V3 in our experiments on I40E NIC
- Still lots of performance optimization work to be performed
- Lots of exciting XDP extensions possibile in conjunction with V4