FAST PACKET PROCESSING IN LINUX WITH AF_XDP

Magnus Karlsson and Björn Töpel, Intel

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

Proprietary HW SDK

Netmap

PF_RING

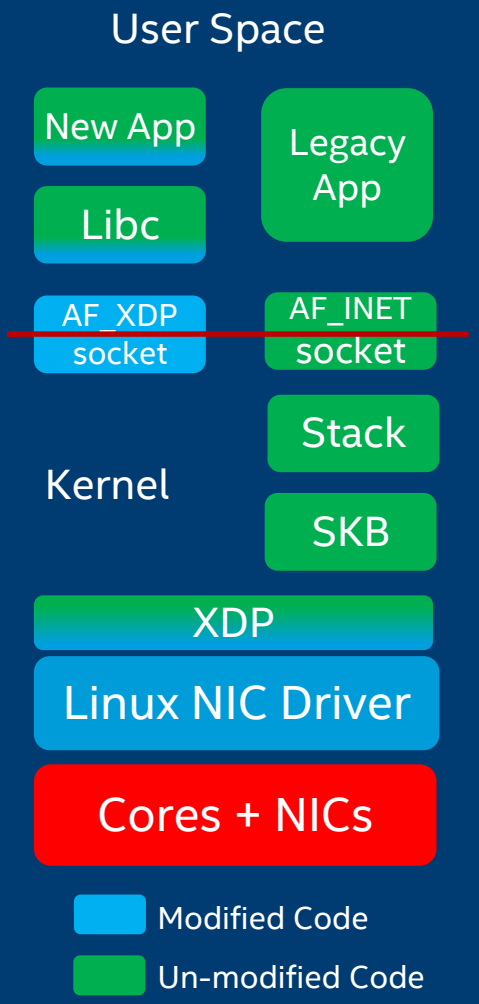
RDMA

- 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?

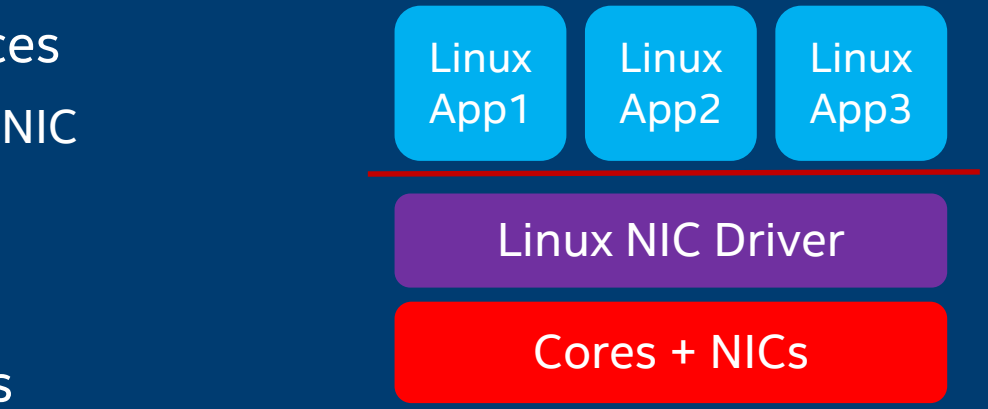
Proposed Solution

- New fast packet interfaces in Linux
  - AF_XDP: XDP's user-space interface
  - No system calls in data path
  - True zero-copy mode with new allocator, DMA packet buffers mapped to user space
  - Copy-mode for non-modified drivers
  - 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 25 Gbit/s for 64 byte packets



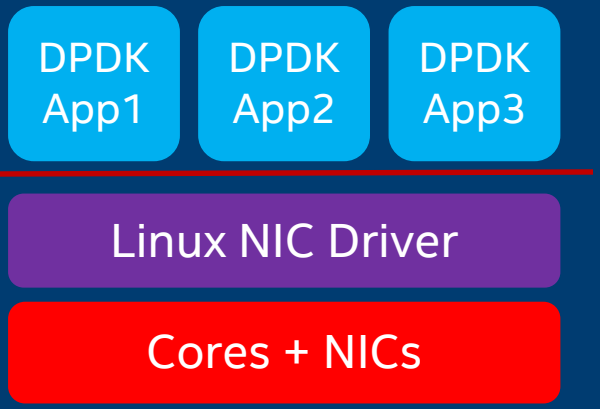
Benefits of Proposed Solution: Linux View

- Much faster standard libc based networking
  - Supports standard libc networking APIs
  - Goal: to be closer to DPDK performance
- Support all Linux network devices
  - E.g. virtio, veth, or your favorite NIC
  - Requires XDP support in driver
- Future work:
  - Speed up networking to VMs
    - Plug in virtio-net ring
    - No need for SR-IOV?
  - Extend it to other device types
    - Crypto and block devices?



Benefits of Proposed Solution: DPDK View

- DPDK AF_XDP based PMD
  - No change to DPDK apps
  - Cost goal: <10% performance decrease
  - Linux handles hardware
- Full isolation between processes/containers
- Linux features can now be used without having to reimplement some of them in DPDK
  - Power save, scheduling, etc.
- No need for bifurcated SR-IOV drivers
- Goal: Linux HW APIs can be used for setup
  - become as simple as "./my_app"
  - DPDK should just behave like a shared library

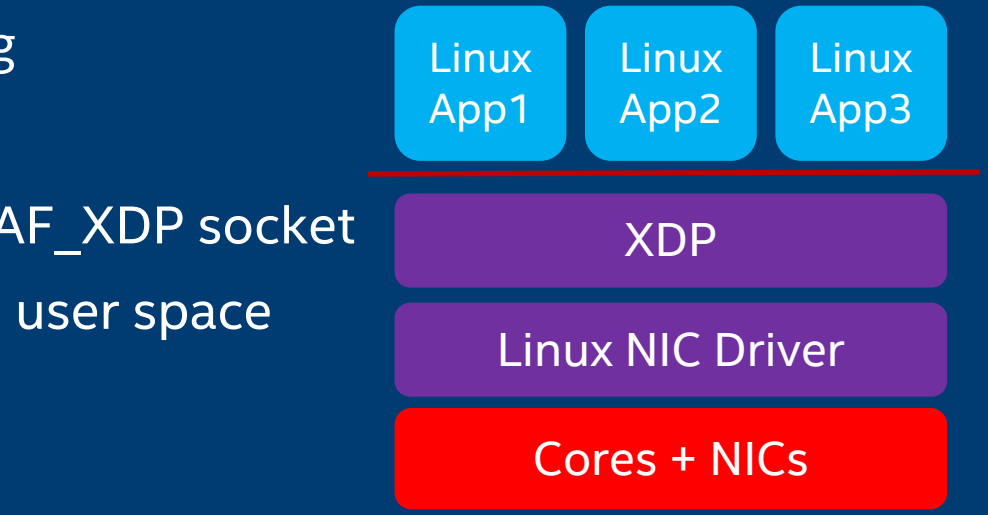


How to Use It?

```
sfd = socket(PF_XDP, SOCK_RAW, 0);
buffs = calloc(num_buffs, FRAME_SIZE);
setsockopt(sfd, SOL_XDP, XDP_MEM_REG, &mr_req, sizeof(mr_req));
setsockopt(sfd, SOL_XDP, XDP_RX_RING, &req, sizeof(req));
setsockopt(sfd, SOL_XDP, XDP_TX_RING, &req, sizeof(req));
mmap(..., sfd); /* map kernel Tx/Rx rings */
struct sockaddr_xdp addr = { PF_XDP, ifindex, queue_id };
bind(sfd, addr, sizeof(addr));
for (;;) {
        read_messages(sfd, msgs, ....);
        process_messages(msgs);
        send_messages(sfd, msgs, ....);
};
```

XDP with AF_XDP

- XDP = Small program injected into driver
- Actions: DROP, PASS, TX, and REDIRECT
  - Forwarding
  - Introspection and debugging
  - ACLs and DDos mitigation
- REDIRECT can now be done to AF_XDP socket
  - E.g., send specific packets to user space
- Future: descriptor rewriting
  - virtio-net support
  - Or any other format
- Future: load balancing, copy to socket + PASS (fast tcpdump)

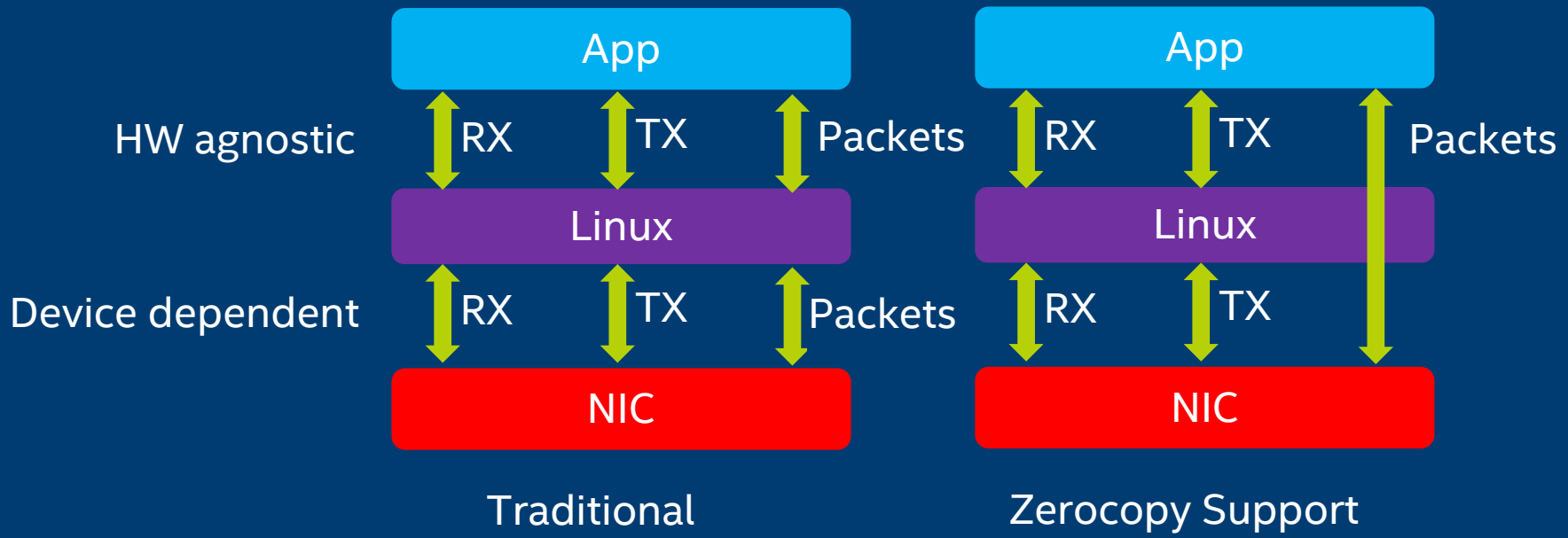


Operation Modes

From slower -> faster

- XDP_SKB:
  - Works on any netdevice using sockets and generic XDP path
- XDP_DRV:
  - Works on any device with XDP support (all three NDOs)
- XDP_DRV + ZC:
  - Need buffer allocator support in driver + a new NDO for TX

Zerocopy Support: Basic Principle



- Application still HW agnostic with ZC
- Each application gets its own packet buffer and TX/RX descriptor rings
  - Packet buffers can be shared if desired
  - TX/RX descriptor rings always private to process

Security and Isolation Requirements for XDP_DRV + 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 => XDP_SKB or XDP_DRV mode need to be used

Experimental Setup

- RFC V1 of AF_XDP published on Januray 31, 2018
- Broadwell E5-2699 v4 @ 2.10GHz
- 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 I40E. 40Gbit/s interface
- Ixia load generator blasting at full 40 Gbit/s

Performance I40E 64-Byte Packets

	AF_PACKET V3	XDP_SKB	XDP_DRV	XDP_DRV + ZC
rxdrop	0.73 Mpps	3.3 Mpps	11.6 Mpps	16.9 Mpps
txpush	0.98 Mpps	2.2 Mpps	-	21.8 Mpps
l2fwd	0.71 Mpps	1.7 Mpps	-	10.3 Mpps

- XDP_SKB mode up to 5x faster than previous best on Linux
- XDP_DRV ~16x faster
- XDP_DRV + ZC up to ~22x faster
  - Not optimized at all at this point!
  - Rxdrop for AF_PACKET V4 in zero-copy mode was at 33.7 Mpps after some optimizations. We have more work to do.

Future Work

- More performance optimization work
- 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
- Queue pairs w/o HW support gets emulated
- XDP redirect to other netdevices RX path
- 1 XDP program per queue pair
- XDP support on TX
- Multi produce single consumer queues for AF_XDP
- Clone pkt configuration

Conclusions

- Introduced AF_XDP
- Integrated with XDP
- AF_XDP with zero-copy provides up to 20x performance improvements compared to AF_PACKET V2 and V3 in our experiments on I40E NIC
- RFC on the netdev mailing list
- Still lots of performance optimization and design work to be performed
- Lots of exciting XDP extensions possibile in conjunction with AF_XDP

Check out the RFC:

https://patchwork.ozlabs.org/cover/867937/

Acknowledgements

- Alexei Starovoitov, Alexander Duyck, John Fastabend, Willem de Bruijn, 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, DPDK, Netmap and PF_RING for the data path inspiration

Check out the RFC:

https://patchwork.ozlabs.org/cover/867937/

Packets from Kernel to Process