# Specializing a Network Device Driver for Datacenter Applications

Han Dong, James Cadden, Tommy Unger, Yara Awad, Orran Krieger, Jonathan Appavoo

Boston University

## Introduction

- Data center applications often single purpose, I/O intensive, and reliant on system specific optimizations to get best performance; i.e. kernel bypass and offload compute to accelerators.
- Low latency is important for these applications and bare-metal provisioning is crucial in supporting these performance sensitive workloads.
- Ability to specialize all the way to the hardware is still limited by legacy device driver interfaces.
- **What kind of interfaces/data structures can enable developers to tailor performance sensitive code to the full abilities of a piece of hardware? (Example: Intel 82599 10GbE NIC [1])**

## System Software Trends

- Application specific LibOS: IX, Arrakis, EbbRT
- User-land packet processing: DPDK, Netmap, PF_RING
- Unikernels: MirageOS, OSv, IncludeOS

## Hardware Trends

- CPU Clock stagnating at around 3 GHz
- Faster NICs from 10 Gbps, 40 Gbps, to 100 Gbps, etc.
- RDMA in Datacenters
- FPGAs, GPUs in the Cloud as compute offload resources
- FPGA attached NIC for in-hardware TCP/IP stack processing

## Summary

System software becoming specialized, hardware getting more diverse and at faster rate compared to CPUs. Need new ways to use hardware from a system software perspective.

## NIC Device Driver Workflow

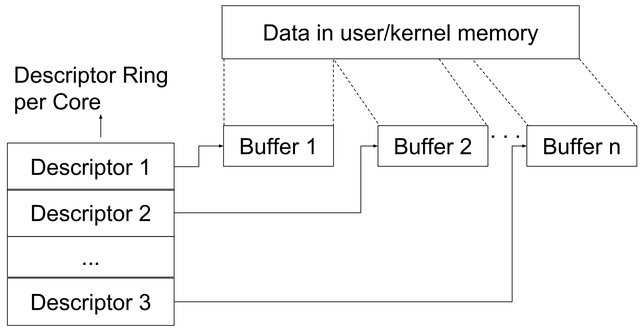


Figure 1: Descriptor ring and memory buffer layout

- **Transmit**: Each descriptor contains a pointer and length of buffer. Address for descriptor is written to NIC, software notifies NIC in order to transmit buffer data.
- **Receive**: A descriptor ring is allocated and pointer to ring and length is written to NIC. On a packet receive, NIC places copies data into a descriptor and triggers interrupt for software to process packet.

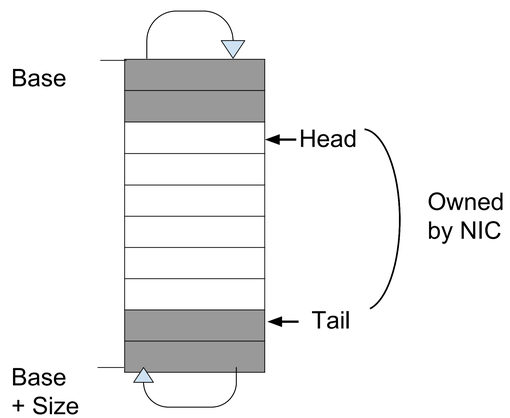


Figure 2: Overview of Head and Tail pointer mechanics in a descriptor ring.

- **Transmit**: Descriptors placed [Head, Tail) will be sent on wire by NIC, Head is incremented to notify software that packet has been sent. Increment Tail to add new descriptors to be sent.
- **Receive**: Descriptors placed [Head, Tail) allows NIC to copy new packets. On packet receive, Head is incremented and interrupt is generated. Increment Tail to add new descriptors for NIC.

## Receive Side Coalescing (RSC)

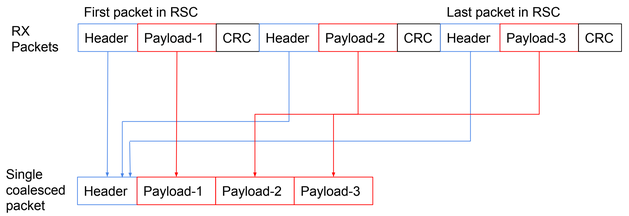


Figure 3: Hardware merges multiple receive frames from same connection into a single buffer. [1]

- Good for maximizing throughput, what about latency?
- Can enable/disable RSC per descriptor ring, do it dynamically?

## Direct Cache Access (DCA)

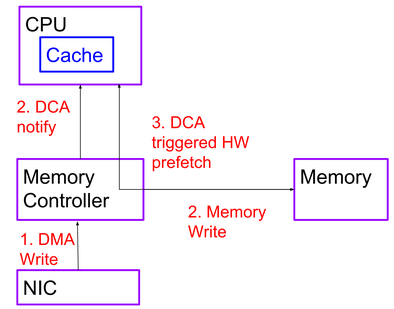


Figure 4: NIC triggers hardware prefetch to put data into cache, potentially improving performance. [1]

- Performance issues arise when CPU is swamped with packet header processing that end up causing preloaded cache to be evicted.
- Can DCA be customized per application or as application behavior change?

## Additional NIC features

- **Header Split**: Packet header and payload written to separate memory buffers for separate header processing
- **Head Pointer Writeback**: NIC automatically increments Head in descriptor ring without software poll and update
- **Jumbo Frames**: Max Ethernet frame size of 9 KB
- Four general purpose interrupts that can be triggered by software
- **IPSec** hardware offload
- **LinkSec** hardware offload

## Issues Enabling NIC features in Linux

- Defined but not implemented (e.g. Header Split, LinkSec)
- Require recompile of driver code and reboot of NIC (e.g. Number Descriptors)
- ethtool can enable some features but applies to entire NIC (cannot customize per descriptor ring)

## References

[1] Intel.
*Intel 82599 10 GbE Controller Datasheet.* 2016.

## Information

- Web: https://github.com/SESA/EbbRT