Streaming with DPDK:

Raising the Throughput Ceiling with Drivers in User Space

Alex Williams

Principal Software Engineer, National Instruments

GNU Radio Conference 2019, Huntsville, AL

NATIONAL INSTRUMENTS

Ettus Research™
A National Instruments Brand

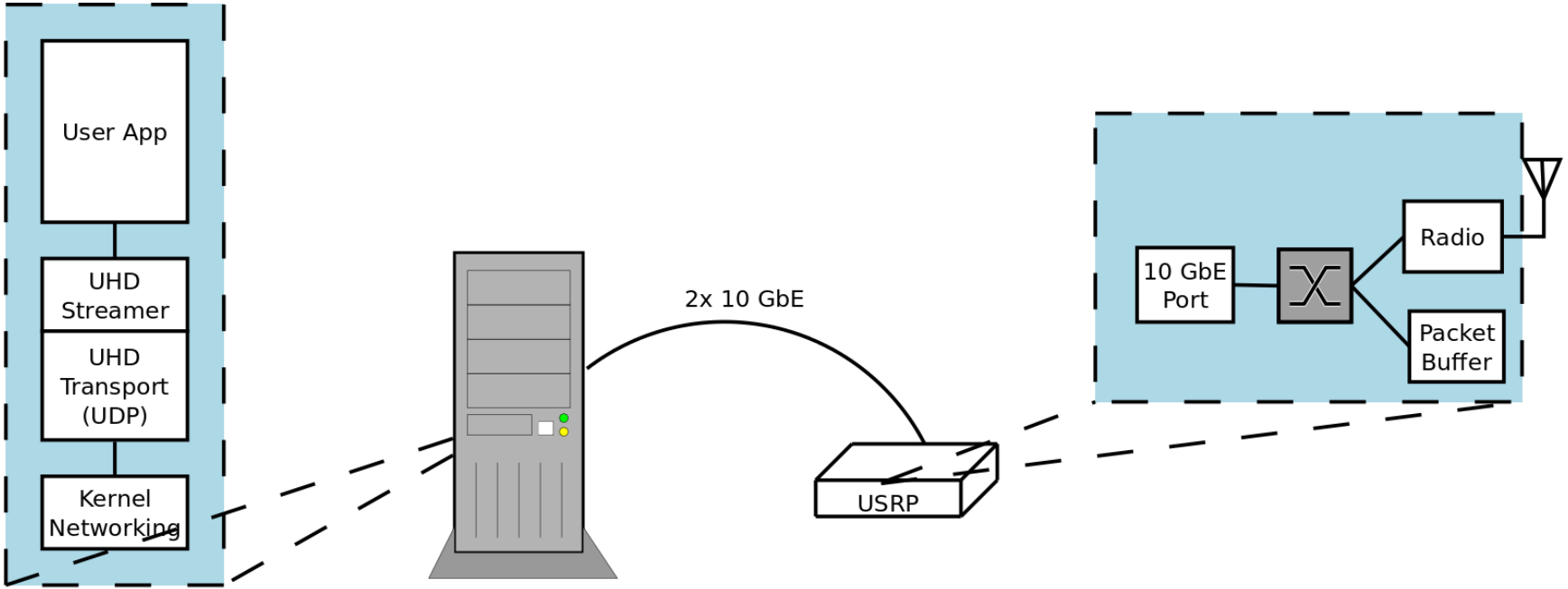
Contents

- Overview of USRP streaming in network mode
  - Architecture and limitations
- Overview of technologies that promote lower latency I/O
  - UIO, VFIO, vfio-mdev
  - CPU affinity and priority/real-time scheduling classes in Linux
  - Data Plane Development Kit (DPDK)
- Introduction to UHD-DPDK
- Benchmark results
- GNU Radio impact

Overview of UHD Streaming
And Current Performance

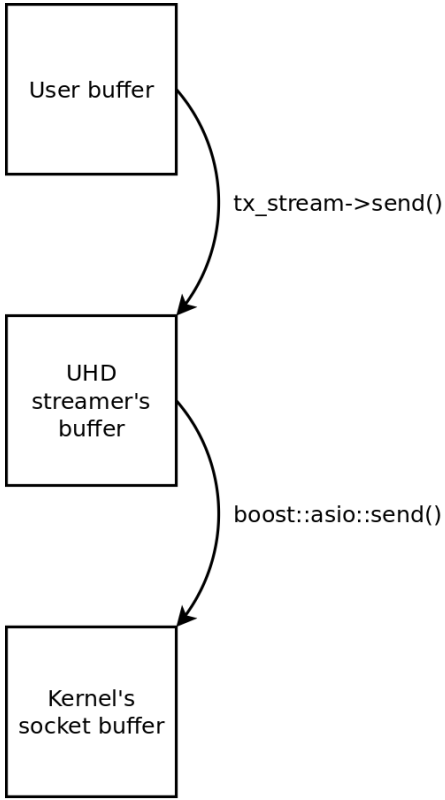
Background: UHD 3.14 Streaming Architecture

(Simplified)



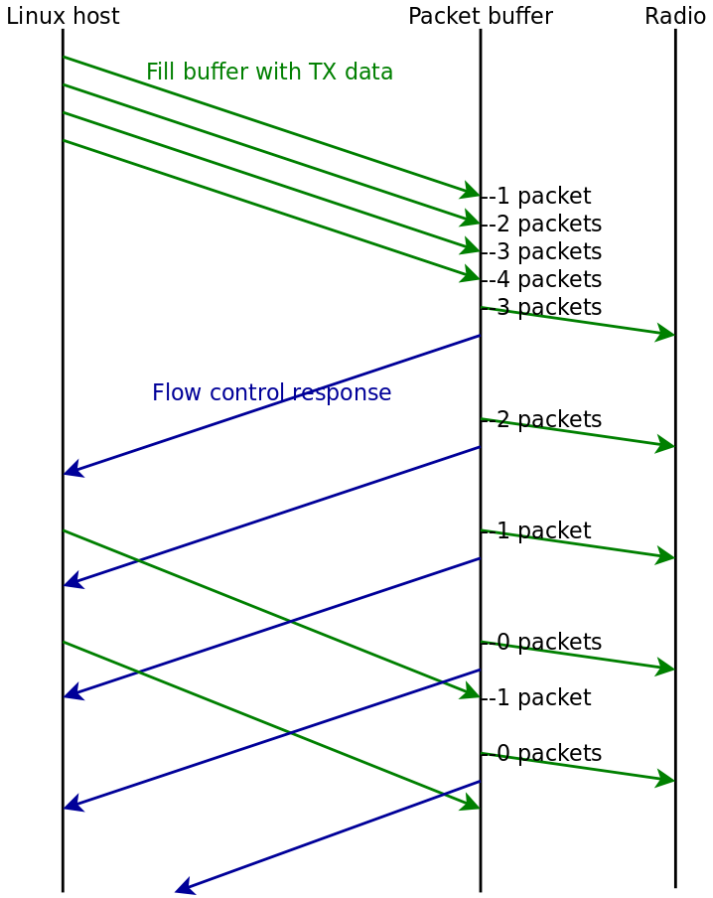
UHD Streaming

- Basic flow of data in software:
  - User creates samples to send in their own buffer
  - User passes buffer to UHD and waits
  - UHD converts samples to USRP's format and copies result to an available internal buffer
  - When USRP is ready, UHD sends a packet with some or all those samples
    - Copies to a socket buffer for kernel to pass to NIC and out to the USRP



UHD Streaming Flow Control

- UHD uses credit-based flow control
  - At init, learn buffer size downstream
  - Before streaming begins, fill buffer
  - Start streaming
  - Wait for free space before sending the next packet
  - Flow control response packet indicates when there is free space
- Need enough buffering to cover worst-case latency

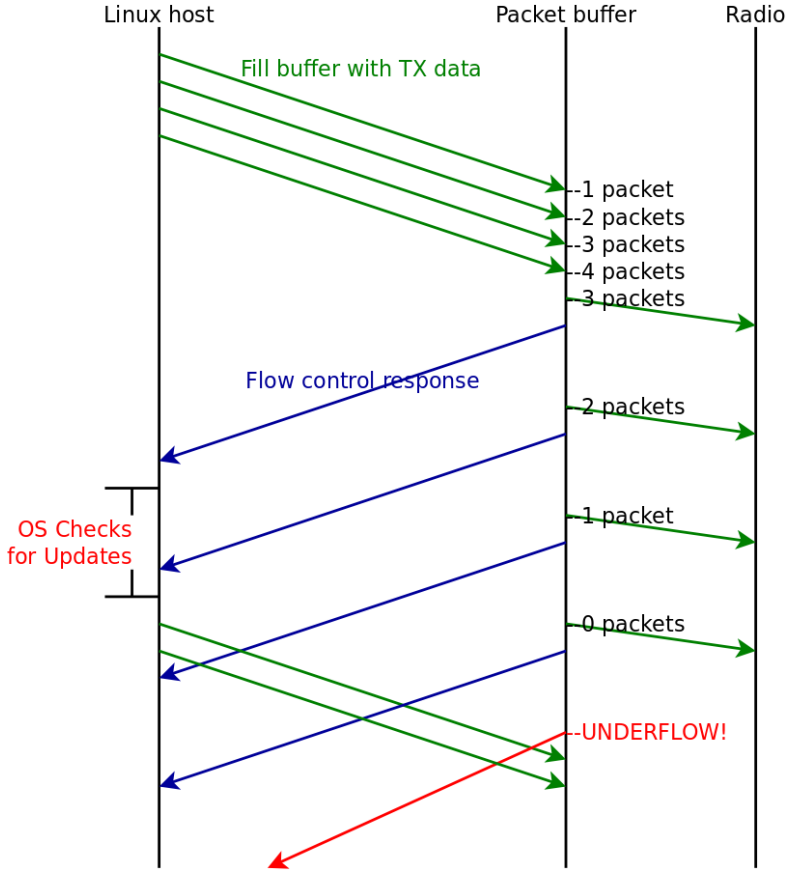


NATIONAL INSTRUMENTS

Ettus Research™ A National Instruments Brand

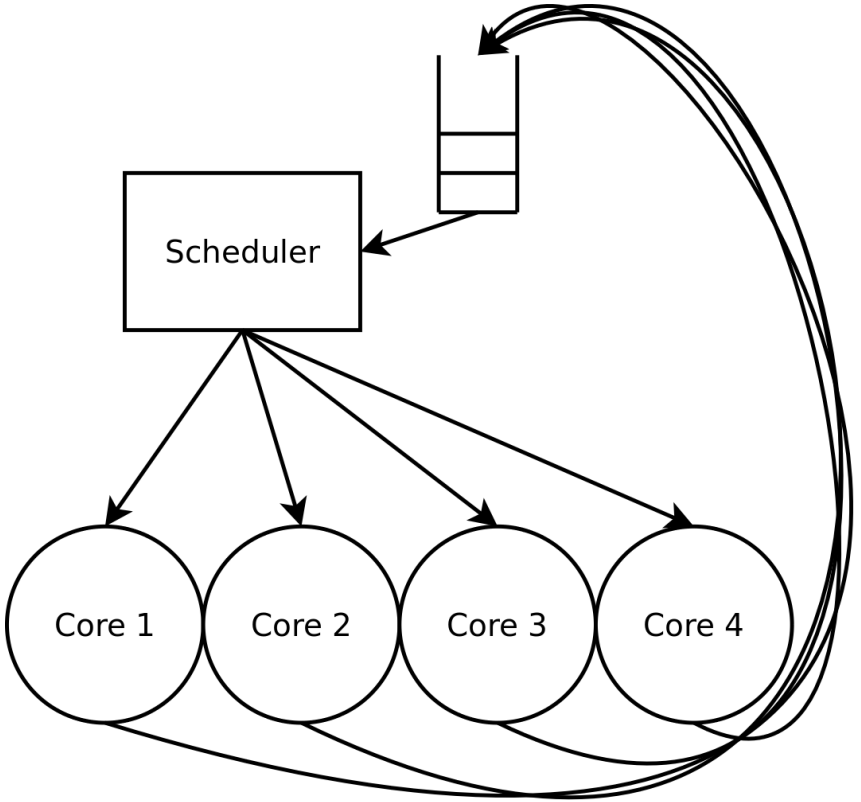
UHD Streaming Flow Control

- Reality: PC's transmissions are bursty
- Significant jitter induced by scheduler
  - Can maintain stream with larger buffers
  - Cost: Higher delays in response, more hardware required



A Little Deeper: Scheduling

- Kernel assigns task threads to cores
  - Metrics include fairness and latency
- Threads may be pulled off a core
  - Voluntarily (e.g. sleeping/waiting on I/O)
  - Involuntarily (this thread's turn is up)
- Linux's default scheduler is the Completely Fair Scheduler
  - Made to share a CPU core fairly
    - Doesn’t know network activity should be very high priority!
    - Thread waiting for the flow control response may wait to handle it



Threads and Contention

- UHD spawns multiple threads (N320 example):
  - 2 for the logger
  - 1 RPC client thread for each radio
  - 1 RPC client thread for MPM's claimer loop
  - 1 for RFNoC's async message handling
  - 1 for each TX channel's asynchronous messages
  - (These spend most of time sleeping)
- User's application threads can compete for core time
- Drivers responding to interrupts compete for core time
- Scheduler will treat the threads the same
  - Kernel defaults are not optimized for real-time systems

USRP Streaming Rates

USRP Type	Sample Rate	# Channels	Minimum Needed Link Utilization	Max Rate Achieved w/ Kernel Stack
X310	200 MSPS	2	> 65%	2x 100 MSPS -or- 1x 200 MSPS
N310	125 MSPS	4	> 80%	3x 125 MSPS
N320	250 MSPS	2	> 80%	2x 125 MSPS -or- 1x 250 MSPS

Overview of Kernel Bypass Technologies

A Strategy for Lower Latency

- Take some scheduling control from kernel
  - Kernel's defaults are optimized for general-purpose
- Minimize context switching for real-time tasks
  - Poll NIC for packets instead of using interrupts
  - Take control of NIC and remove overhead caused by code for sharing
- Some technologies that help:
  - CPU affinity control
  - High-priority/real-time scheduling classes
  - APIs for drivers in user space (uio, vfio, vfio-mdev)
  - Hugepages

Controlling scheduling

- CPU affinity control APIs
  - Can restrict set of CPUs to which a thread may be assigned
  - Migrating threads to other cores is expensive
- High-priority/real-time scheduling classes
  - Linux supports multiple scheduling classes
  - Completely Fair Scheduler is the default
  - SCHED_RR and SCHED_FIFO are part of higher-priority class
    - Threads here always run before threads at level using CFS
- Can effectively take control of CPU cores with both
  - Or use the *isolcpus* boot argument to isolate cores directly

Drivers in User Space

- Userspace I/O (UIO)
  - Make available portion of address space for direct access to device memory
  - Can read/write registers and receive IRQ signals
  - No memory protection for device DMA
- Virtual Function I/O (VFIO)
  - Superset of UIO functionality
  - Uses IOMMU for memory protection, to facilitate safe DMA to user space
- VFIO for Mediated Devices (vfio-mdev)
  - Special subtype of VFIO, where only certain functions are exported to user space
  - Example: A driver that wants kernel to control device configuration but gives user space access to the DMA engines

Data Plane Development Kit (DPDK)

Introduction

- Networking framework for fast packet processing
- Takes advantage of these technologies
  - UIO/VFIO for user space net device drivers
  - CPU affinity + real-time scheduling classes to map tasks directly to cores
  - Polling mode instead of interrupt processing
- Provides raw interface to net devices
  - Operates at layer 2
  - No ARP, IP, TCP, or UDP protocols built-in

Data Plane Development Kit (DPDK)

Other niceties

- Environment Abstraction Layer (EAL)
  - "Logical cores" representing each CPU
  - APIs to launch tasks on cores
  - Currently supports Linux and FreeBSD
    - Windows port is a WIP
- Lockless ring buffers
  - Useful data structure to avoid blocking/sleeping
- Intelligent memory manager
  - Spreads memory used across DRAM channels
  - Manages caching of objects close to the CPU

DPDK

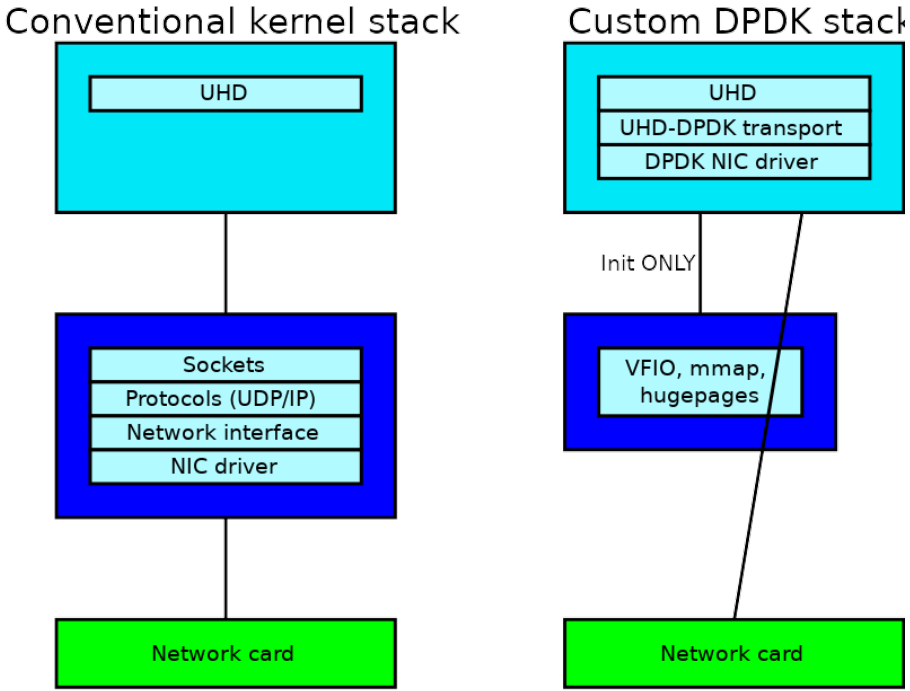
Vendor Support

- Most prominent supporters are Intel and Mellanox
  - Intel approach uses vfio-pci
    - Chip bound to special user space driver
  - Mellanox approach akin to vfio-mdev
    - Kernel driver exports access to DMA engines for packet transmission / reception
    - No need to load a separate driver
- Other vendors with support include Cavium, Chelsio, and Solarflare

UHD-DPDK

UHD-DPDK

- DPDK-based transport for UHD
  - UHD network traffic completely bypasses the kernel
  - Much-reduced latency for transmission/reception
- Minimal network stack
  - UDP / IPv4, ARP
  - No IPv4 fragmentation
- Zero-copy operation
  - Socket-like registration
    - Maps RX queue to UDP port
  - Queues contain pointers to DMA-able packet buffers



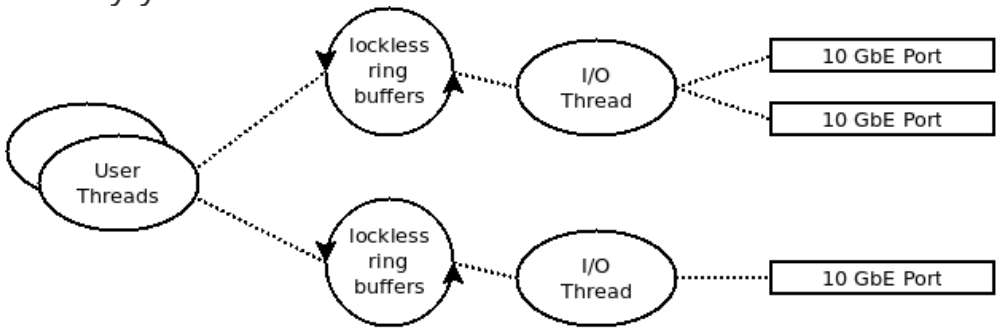
NATIONAL INSTRUMENTS

Ettus Research™ A National Instruments Brand

UHD-DPDK

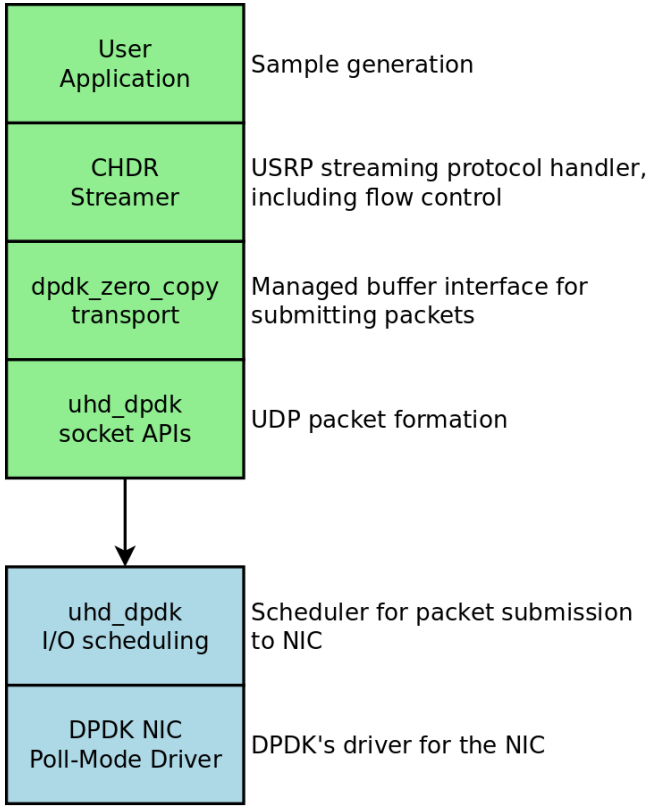
Threading model

- Pool of I/O threads services requests and schedules I/O
  - An I/O thread maps to a configurable set of NIC ports and has exclusive access
  - User threads interact with I/O thread via lockless ring buffers and pthread condition variables (for blocking calls)
  - Each I/O thread consumes a CPU core 100%
    - Never voluntarily yields the CPU



UHD-DPDK

- Replaces UDP transport
  - CHDR streaming layered on top
- No changes to streamer APIs
  - Flow control handling still in user core
  - I/O loop split across two threads



UHD-DPDK

Performance

- With DPDK, full-rate streaming achieved on N320!
  - Test app = simple signal generator + received samples written to /dev/null
  - 1 I/O thread serving 2 NIC ports
  - BRAM buffers of 64k samples (256 KiB) per TX channel
  - RX buffers are roughly the same size (512 packets deep for my tests)
- Significant latency improvements: (*latency_test* UHD example)

Transport	Typical round-trip latency
udp_zero_copy	300 us
dpdk_zero_copy	80 us

GNU Radio + UHD-DPDK

Performance

- Performance improves, but… GNU Radio has bottlenecks
- Max rates without underflow for similar work:

Test	Kernel Driver	DPDK
tx_waveforms	(2x) 100 MSPS	(2x) 250 MSPS
uhd_siggen	(2x) 83.3 MSPS	(2x) 125 MSPS

GNU Radio Bottleneck

- Thread-per-block scheduler a likely bottleneck for higher streaming rates
  - Rapidly creates more threads than available CPU cores
    - Excessive overhead due to high level of thread switching and migration
    - Some blocks have tiny workloads but still get own threads
    - Frequent voluntary switching invites latency spikes
- What can we do about it?
  - Partition the graph at a coarser granularity than individual blocks
  - Assign subgraphs to threads (with heuristic for load balancing?)
  - Possibly auto-adapt and move partitions
    - But have a way for users to create partitions manually

GNU Radio

Questions for Community

- Do we want GNU Radio support for streaming higher data rates?
  - Can do this for UHD, but is it important to GNU Radio users?
- If so, how do we want to handle processing?
  - Limits to what can be done on a general-purpose processor
  - Recall GRCon 2018 discussion surrounding heterogeneous computing
    - Are there GNU Radio applications that require support for heterogeneous computing?

Discussion