urdma: A Remote Direct Memory Access verbs provider using DPDK

PATRICK MACARTHUR
UNIVERSITY OF NEW HAMPSHIRE

SEPTEMBER 6, 2018

Acknowledgements

- urdma was initially developed during an internship with the IBM Zurich Research Laboratory. The author would like to thank Dr. Bernard Metzler for the opportunity as well as Jonas Pfefferle, Patrick Stuedi, and Animesh Trivedi for their advice and critique on urdma.
- The author would like to thank Robert Russell and Timothy Carlin for their advice and critique on this report and the University of New Hampshire InterOperability Laboratory for the use of their RDMA cluster for the development, maintenance, and testing of urdma and UNH EXS.
- This material is based upon work supported by the National Science Foundation under Grant No. OCI-1127228 and under the National Science Foundation Graduate Research Fellowship Program under award number DGE-0913620.

Agenda

- Background
- Implementation
- Evaluation
- Summary

Background

Background: RDMA (Remote Direct Memory Access)

- Message-oriented
- “Zero-copy”: direct transfer between remote application virtual memory regions with no intermediate data copies (on the hosts)
  - Requires application to pre-register memory
- Kernel bypass: userspace application has direct access to network adapter
- Asynchronous: data transfers occur in parallel with application threads, using OpenFabrics Alliance (OFA) **verbs API**
- Transfer operations
  - SEND/RECV
  - RDMA WRITE: push data to remote memory region
  - RDMA READ: pull data from remote memory region
- Data structures: Queue Pair (QP), Completion Queue (CQ)
- Standards: InfiniBand, RoCE, **iWARP**

urdma: Userspace Software RDMA

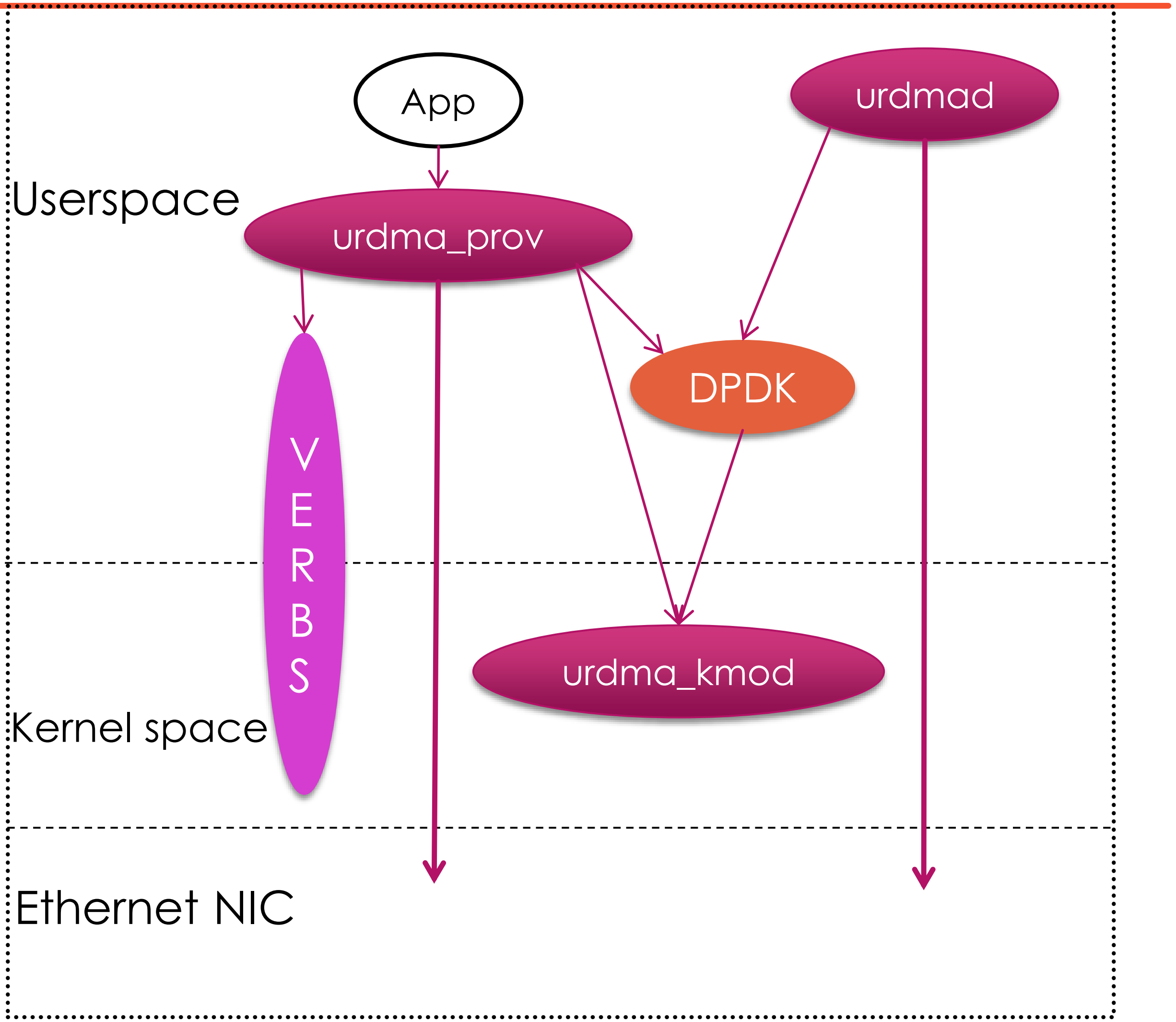
- Software emulation of RDMA using DPDK
- Goals
  - Low latency, high throughput
  - Run on commodity Ethernet NIC
  - Run unmodified verbs applications
  - Perform data transfers in userspace using DPDK
- Prior work: softiwarp/softroce
  - Perform data transfer in kernel space using kernel sockets
- Why urdma?
  - Ease of development, easy to use as a development vehicle for new RDMA features
  - Storage applications; integration with SPDK (Storage Performance Development Kit)

Implementation

urdma: Components

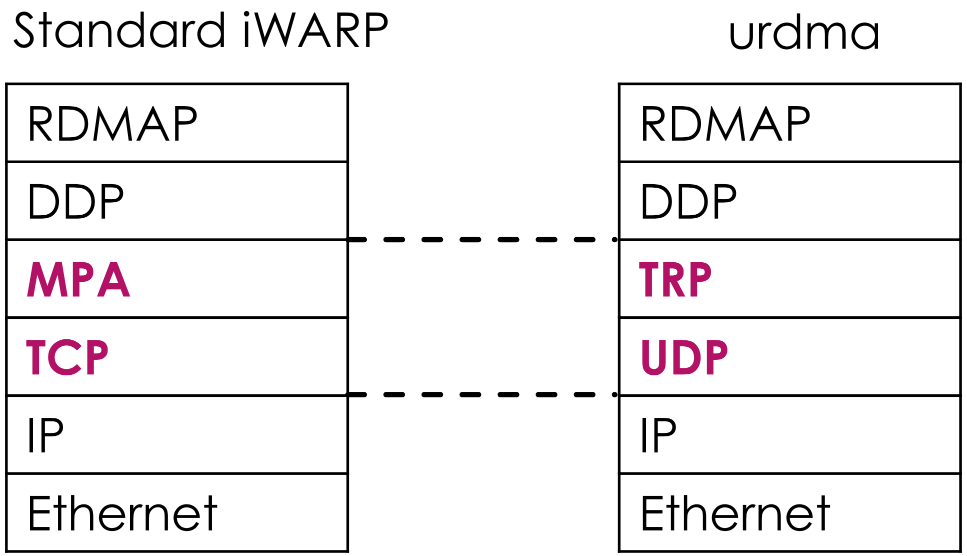
Multi-process application

- **urdma_kmod**: Loadable kernel module for RDMA CM support
- **urdmad**: DPDK primary process
- **urdma_prov**: User verbs provider library; applications run as DPDK secondary process



urdma: Protocol

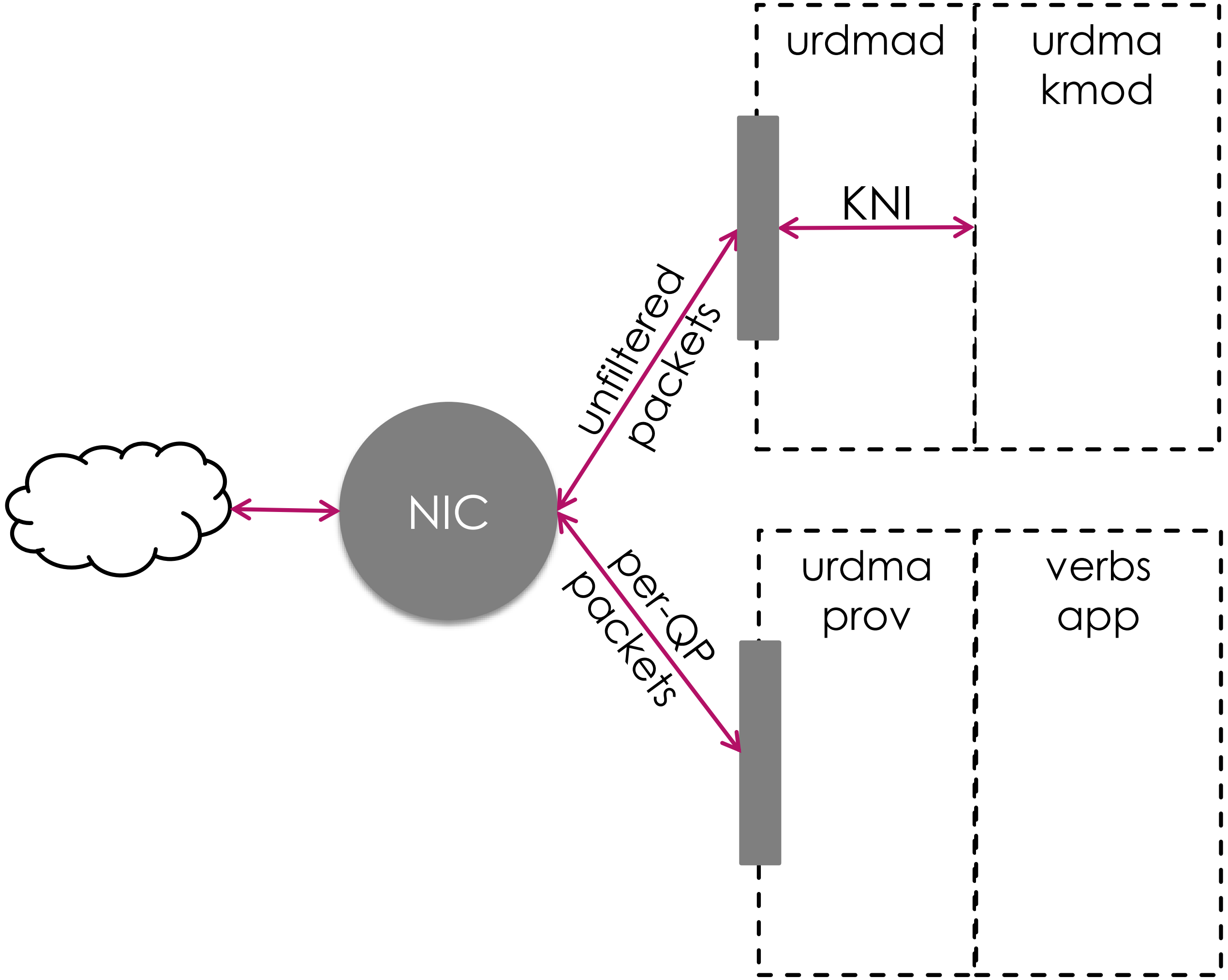
- Implements iWARP DDP and RDMAP protocols
- Runs over UDP transport protocol
  - TRP (Trivial Reliability Protocol) as thin reliability shim
  - Avoid byte-stream nature and state machine of TCP



Standard iWARP	urdma
RDMAP	RDMAP
DDP	DDP
MPA	**TRP**
TCP	**UDP**
IP	IP
Ethernet	Ethernet

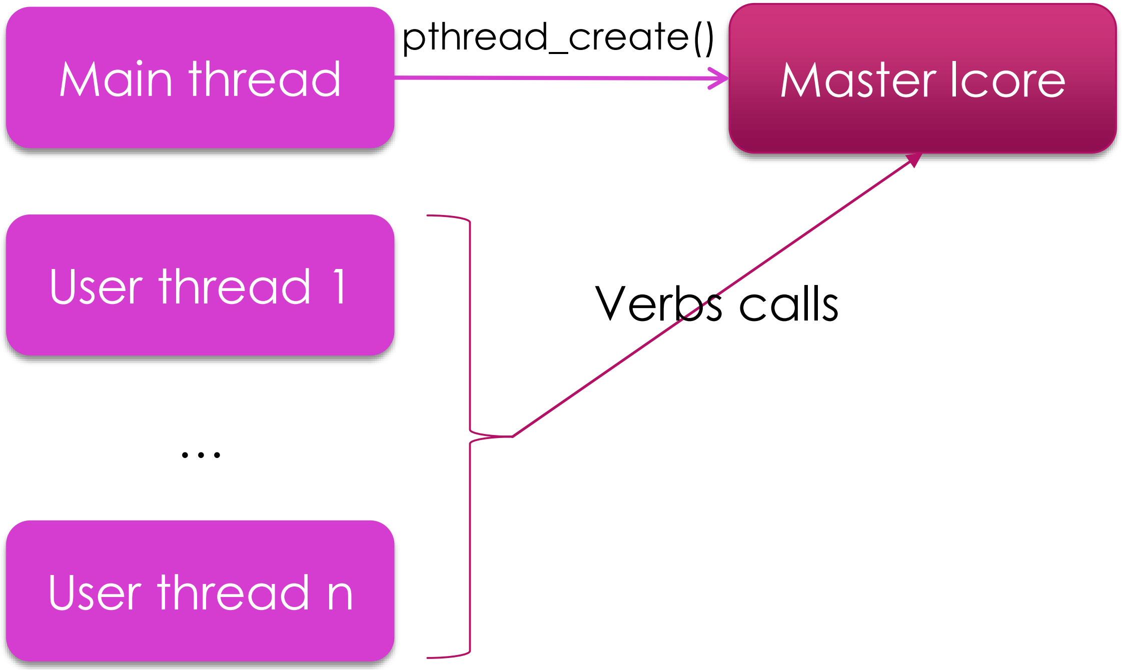
urdma: Packet Processing

- urdmad assigns each RDMA queue pair a hardware receive/transmit Ethernet queue
  - To allow verbs applications to access the NIC independently
- Ethernet NIC hardware filters used to separate packets into RX queues
  - Using **Flow Director** or **ntuple** filtering
  - urdmad forwards all unfiltered packets on each interface to kernel
  - For each established connection, packets filtered to specific receive queue—received directly by verbs application via urdma_prov



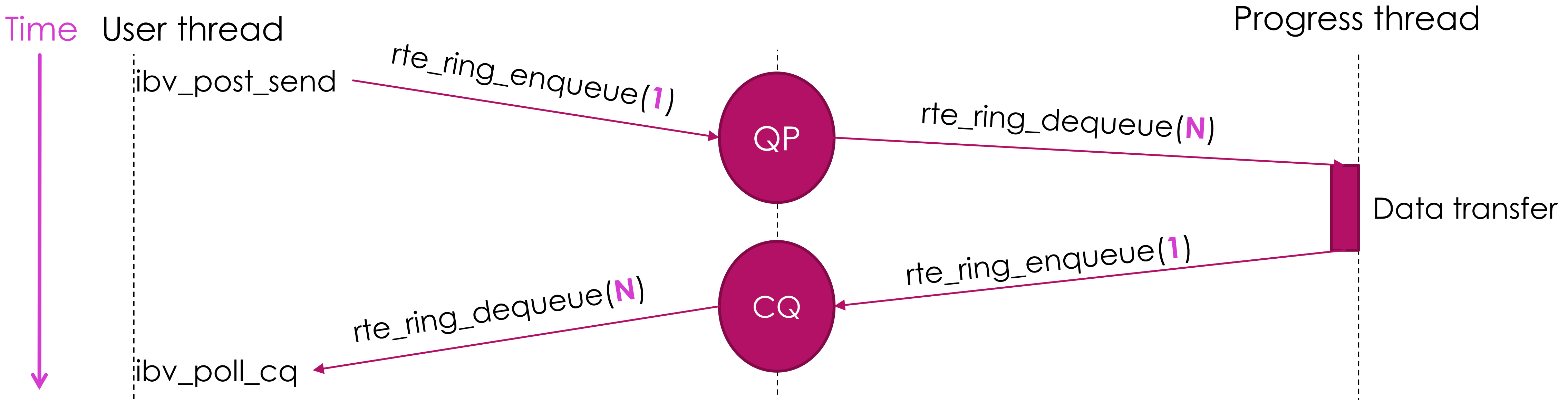
urdma initialization issues: rte_eal_init()

- As a verbs provider library, we want DPDK to be invisible to the user application
  - We call rte_eal_init() in our own implementation
- Our provider code is only run if urdma kernel module loaded and urdmad master process started
- Specific issues with rte_eal_init()
  - Takes command-line arguments
    - We construct our own fake argument list
  - Changes CPU affinity of calling thread
    - We create a new thread and call rte_eal_init() from that thread
    - Tell rte_eal_init() not to create other lcores
- All verbs applications must run as the same user (not necessarily root)

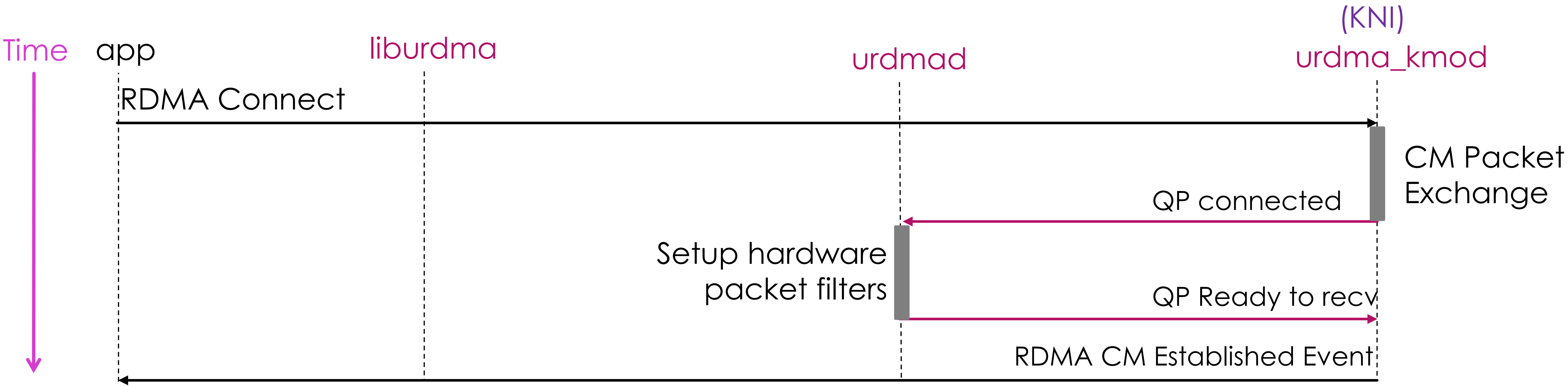


urdma_prov: Data Transfer

- Data transfer done in background progress thread
  - Separates DPDK operations from application threads
  - Allows progress for RDMA READ and RDMA WRITE outside of verbs calls
  - Inter-thread communication done via ring queues
    - ➢ Enqueue one entry at a time
    - ➢ Dequeue entries in bulk



urdma_kmod: Connection Establishment



urdmad must enable receive filter before first packet arrives

Evaluation

Performance Test Setup

2 pairs of systems with Ubuntu 16.10 with Linux 4.8.0-46-generic kernel, DPDK 16.07.2, PCIe generation 3

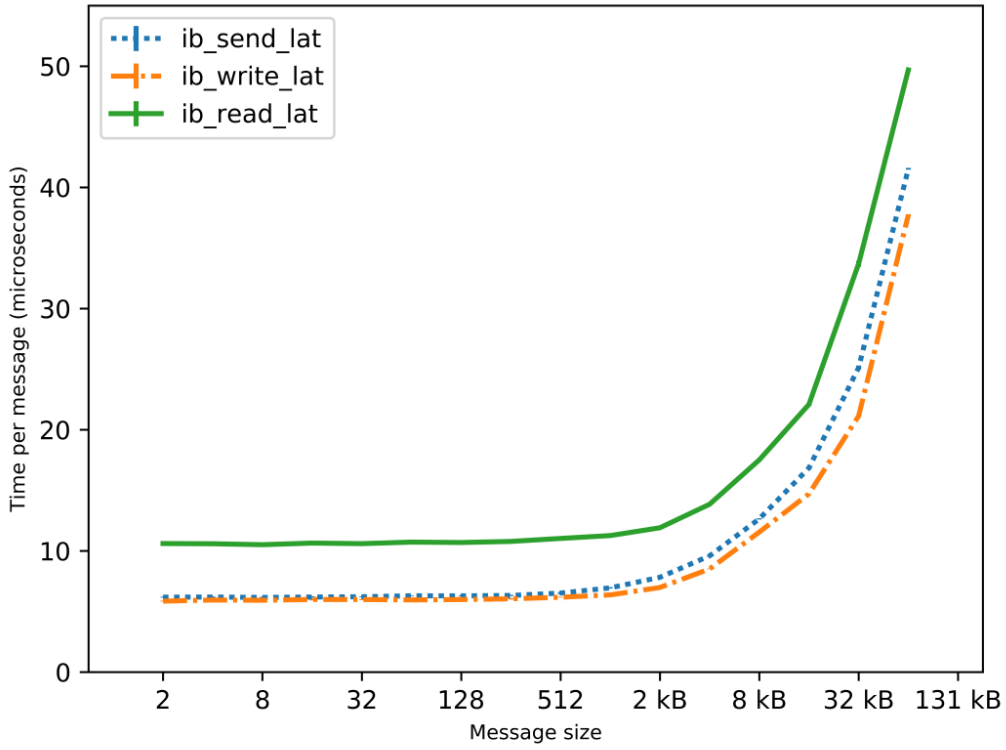
- urdma/softiwarp (Software Implementations)
  - Dual Intel Xeon ES-2630 v4 CPUs @ 2.20GHz
  - 64 GB DDR4 RAM
  - **Intel XL710 40GbE** NIC (firmware v5.05)
- Reference iWARP Hardware Implementation
  - Dual Intel Xeon E5 2609 CPUs
  - 64 GB DDR3 RAM
  - **Chelsio T580-LP-CR** Unified Wire Ethernet controller (firmware v0.271.9472)
- Applications used
  - perftest version 3.0+0.18.gb464d59-1

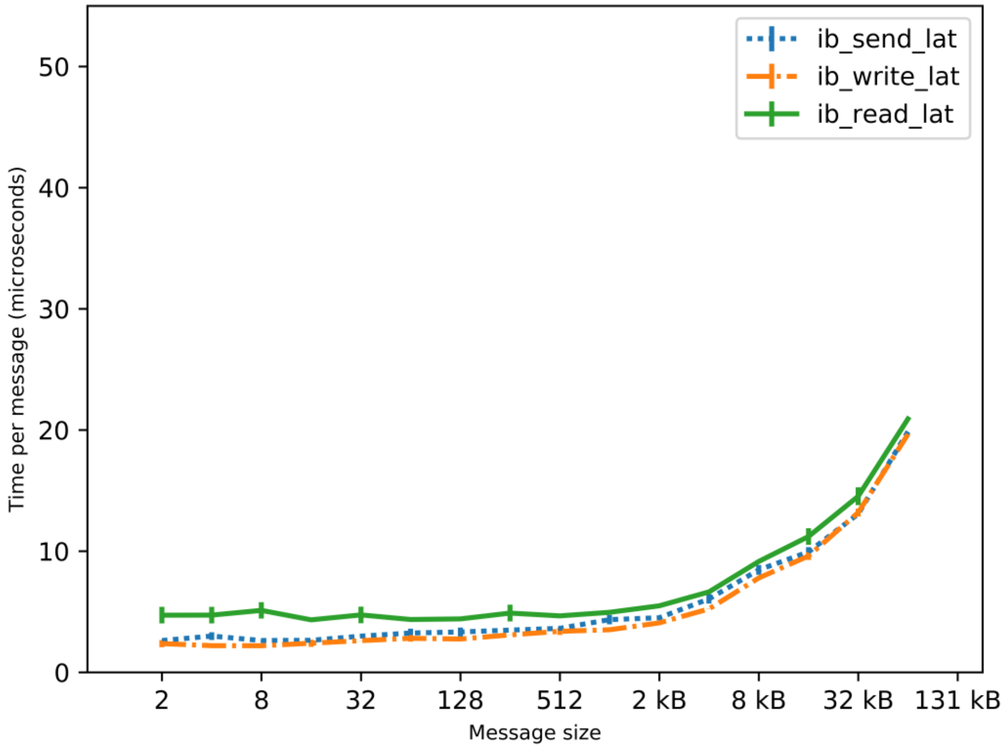
Perftest Latency: urdma vs. Chelsio iWARP NIC

urdma (Software)

Hardware

Worse

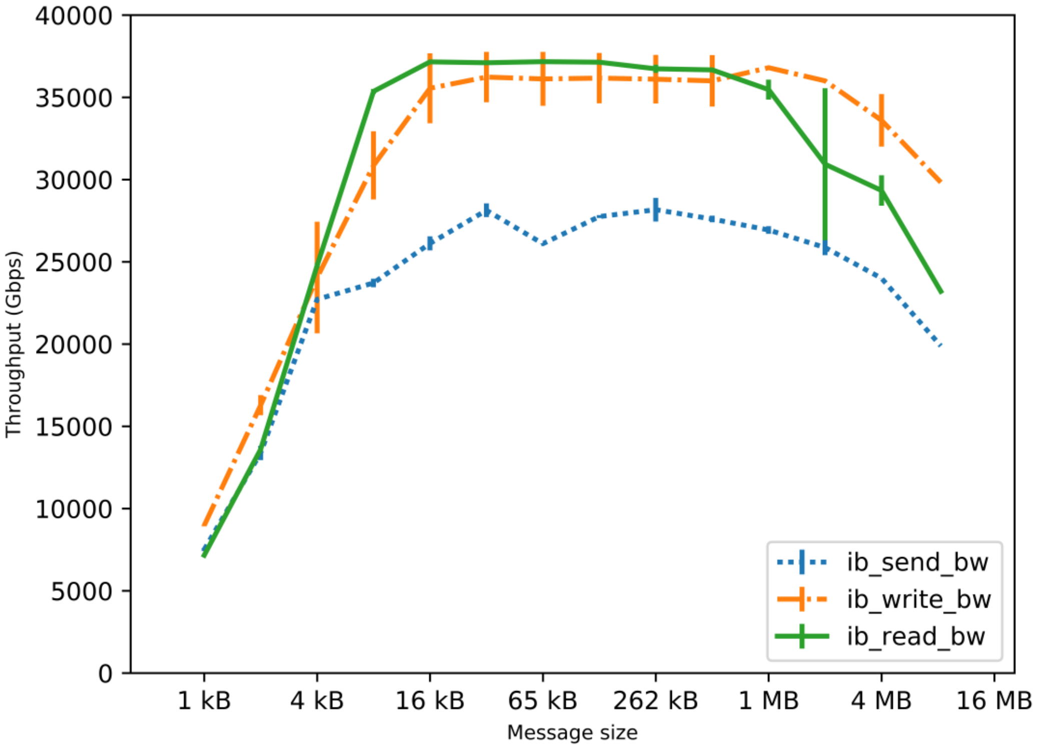




Better

Perftest Throughput: urdma vs. Chelsio iWARP NIC

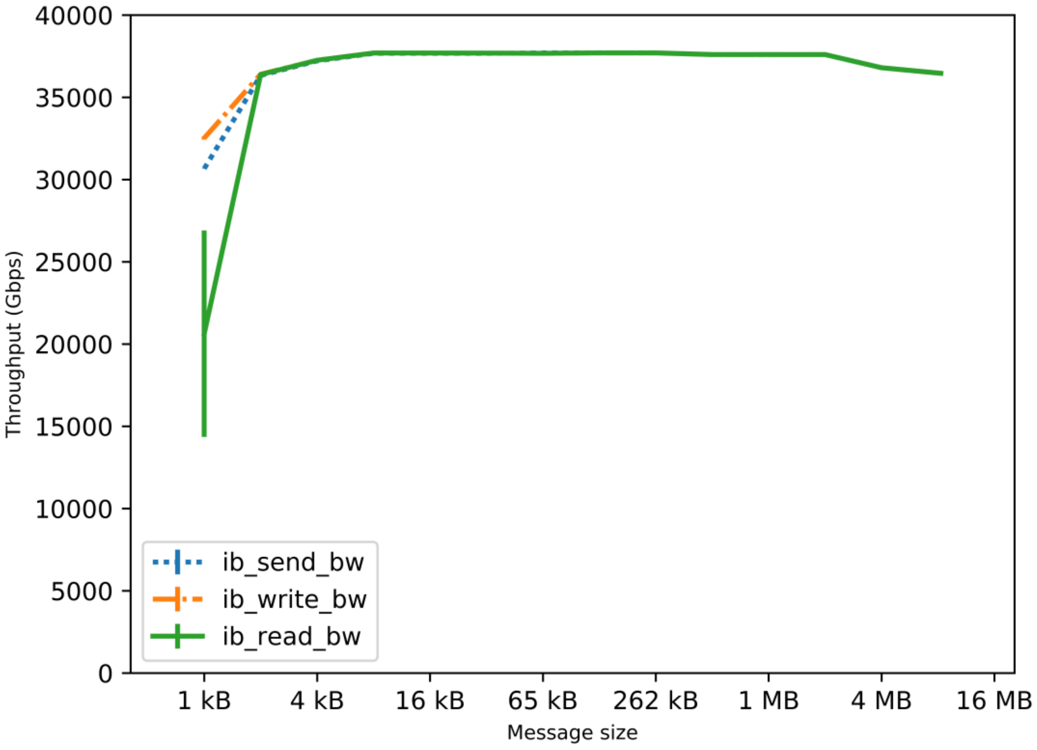
urdma (Software)



Better

Worse

Hardware



Summary

Summary

- urdma
  - Software emulation of RDMA
  - Runs unmodified RDMA verbs applications
  - Performs all data transfer in userspace
  - No dependency on specific hardware
  - Achieves reasonable performance
- Future work
  - Zero copy sends?
  - Using urdma for NVMf traffic
  - Integration with emerging storage class memory technologies

“

Thanks!

”

Questions?

Patrick MacArthur <patrick@patrickmacarthur.net>

urdma download: https://github.com/zrlio/urdma