AI Networking - RoCE v2 and netdev

David Ahern, Leon Romanovsky

March 2025

Introduction

- Goal of this workshop

  Normalize discussions within netdev community that involve any relevant networking transport with Linux based APIs

- AI networking is very relevant today

  Demands for high throughput and low latency drives a lot of networking research and discussions

- AI networking primarily driven by RDMA over ethernet - RoCEv2

  RoCEv2 is a standards based transport protocol over UDP

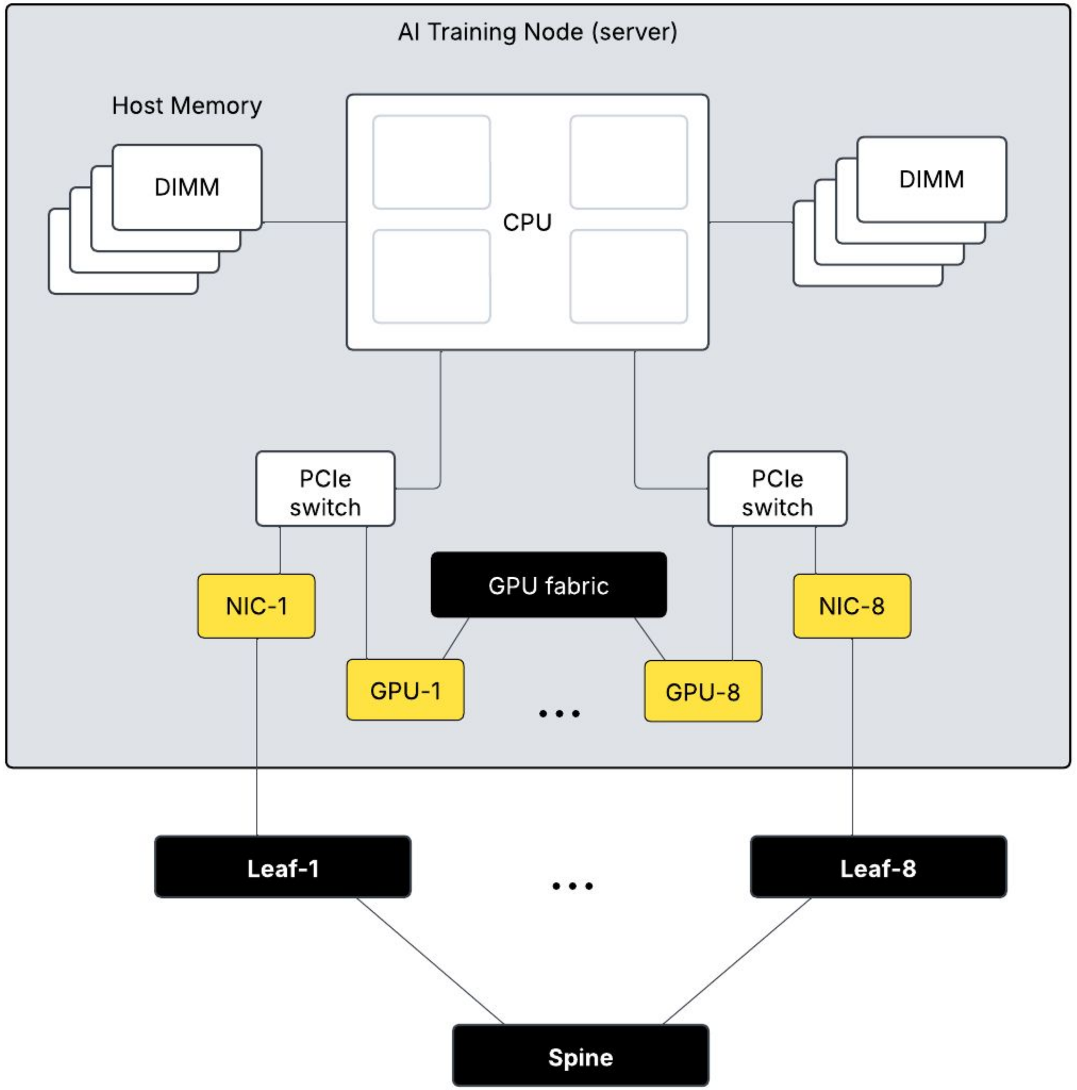
Agenda

- Set a foundation for AI networks
- Overview of socket networking vs verbs and RoCEv2
- Why RoCEv2 has become the dominant protocol for AI networks
- Latest advances for socket networking and expectations
- Revisit the proposal for using Linux TCP with QPs

Want this to be interactive. Ask questions as we go. Defer as needed if we hit a time crunch.

AI Training Networks

- Large number of hosts and GPUs
  - Meta: 24k GPUs across 3k nodes, 400 Gbps ports
    - targeting upwards of 128k GPUs across 16k hosts
  - xAI: 100k GPUs in a single RDMA fabric, 400Gbps port
    - Colossus - targeting upwards of a million GPUs
- 100s of billions to now trillions of parameters for models
  - Large data sets moved between nodes and GPUs
- Training time for each round dominated by tail latency
- Specialized, dedicated ethernet networks for training
  - Lossless, rail design
- Focus here is networking within each host in such clusters



Required Characteristics of Host Networking

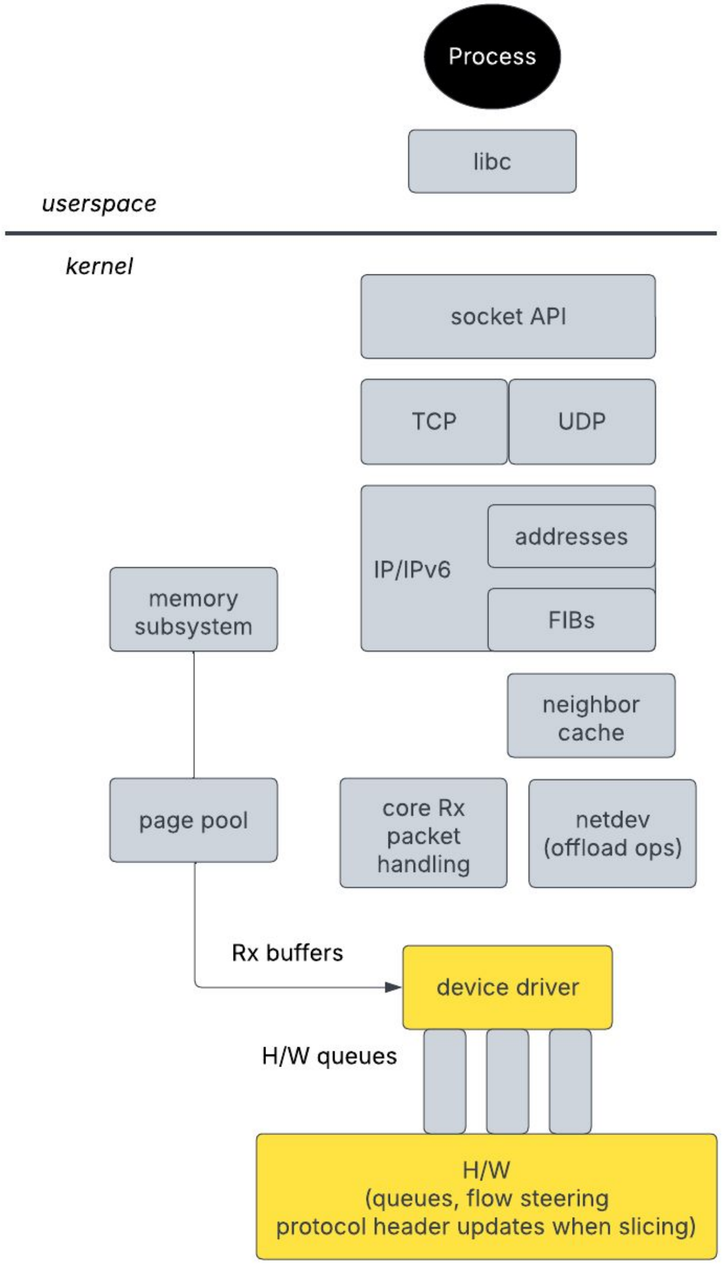
- Focus is a few large flows running at 100s of Gbps
  - vs millions of flows per host
  - 400Gbps port per GPU → 8 flows running at 400 Gbps
- Get the OS out of the way of the data path
  - At 100+Gbps every cycle matters
  - Every system call, generic feature and hook is just overhead
- Eliminate memory copies
- AI training leverages high performance GPUs and high speed NICs
  - NIC hardware needs to read / write payload directly from / to GPU memory
  - H/W needs to land packet payloads in proper order relative to memory / buffers posted for specific messages
    - Handling packets that arrive out-of-order is a hardware and transport problem
  - Completions for work requests are in order

Required Characteristics of Host Networking

- Leverage hardware
  - Ethernet protocol headers are simplistic and repetitive - generating wire packets is mechanical process
    - Hardware is really good at repetitive tasks
  - Similar to TSO/GSO - packet headers + large payload
    - Hardware updates protocol headers as needed (e.g., TCP Seq number, checksums, payload lengths)
- Avoid packet drops
  - Retransmits are expensive for flow rates and time to completion
  - Congestion control - slow down Tx side when Rx side gets overwhelmed

netdev and Linux Networking Stack

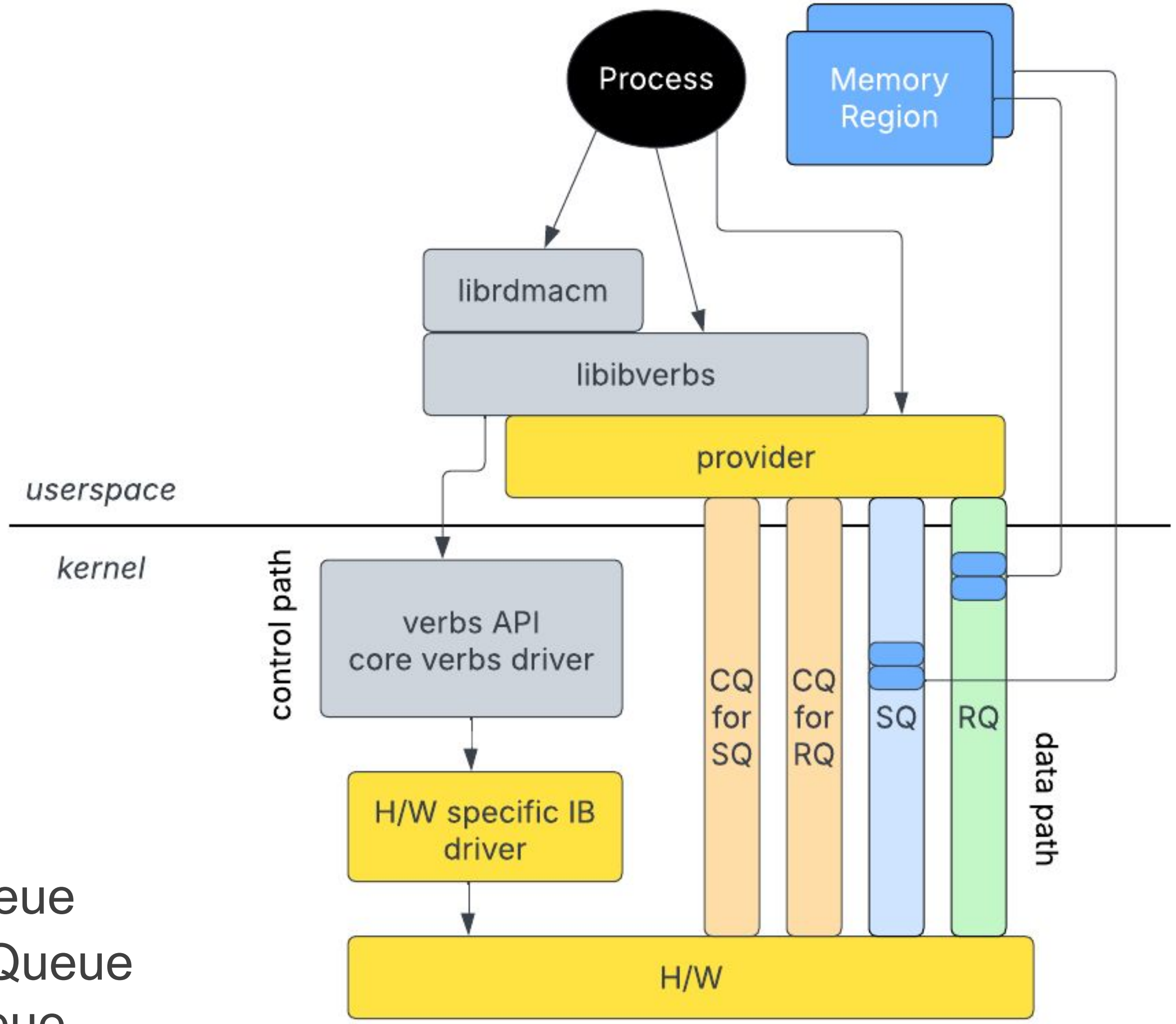
- net_device representer for ports
  - Hold port state - up / down, carrier, speed, mtu
  - Reference for network addresses, neighbor entries, FIB entries
- Page pool for Rx packet buffers
- Socket based networking
  - e.g., TCP/UDP L4, IP/IPv6 L3
- System calls in the datapath
- ZC APIs now exist for payloads
- Hooks for a lot of infrastructure
  - packet taps, netfilter, tc, ebpf



IB Verbs Software Vertical Stack

Open-source and community driven

- Kernel:
  - HW modules - configure devices
  - SW modules - simulate RDMA HW
  - IB/core - provides UAPI and internal verbs API
  - ULPs - implement extra functionality on top of verbs
- rdma-core:
  - libibverbs - exposes verbs to userspace
  - librdmacm - connection management API
  - providers - vendor support code for verbs
- Kernel modules and their rdma-core respective providers needs to be seen as one piece



Basic verbs terminology

- UCTX - user context
- PD - protection domain
- QP - queue pair
- MR - memory region

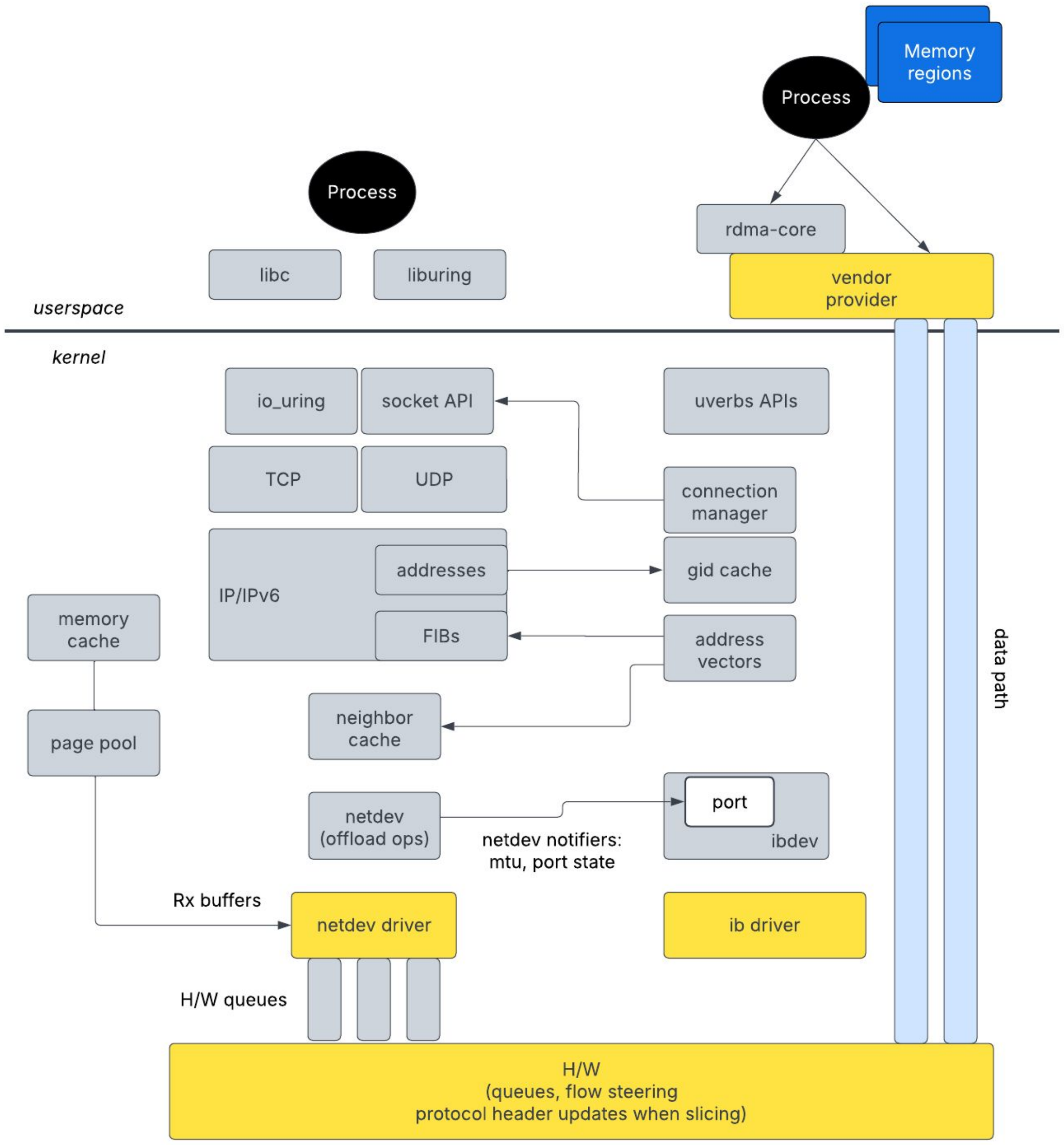
CQ - Completion Queue
RQ - Receive Work Queue
SQ - Send Work Queue

RDMA Data path

- Completion Queue, Receive Queue, Send Queue
  - H/W queues managed by userspace
  - Work requests submitted via SQ and RQ
  - CQ for notifications when WR is complete
- Message based transfers
  - Each WR in RQ, SQ equals 1 message
  - WR can reference an sg list with many buffers
- Opcodes
  - SEND
    - Rx side posts recv work request - buffers for expected incoming message (or max message size)
    - Tx side posts a send work request; hardware creates wire packets. Very similar to a large TSO packet
    - RC QP and SEND is the closest thing in RDMA world to netdev transaction
  - RDMA_WRITE - no Rx buffer needed; writing to specific address and length
  - RDMA_READ - lesser used opcode
    - Requester knows exact buffers needed for a response
    - Local end posts Rx buffers for a request
    - Sends READ request to pull data from a specific address and length

IB Stack and netdev

- Out-of-band communication messaging
  - Socket based or a second QP
- Connection Manager can be socket based
- gid cache - Addresses on netdev
- Address handles leverage FIB and neighbor lookups
- IB port state tracks netdev state
  - up / down state
  - RoCE MTU based on netdev MTU
- Data flow offloads are controlled from netdev ops
  - IPsec - transparent for IB stack, rely on netdev routing
  - MACsec - connected to IB stack through GID entries, IB stack manages their lifetime, everything else through netdev



RoCE v2

- Standardized protocol
- RDMA over IP/ethernet networks
  - ethernet + IP/IPv6 + UDP + BTH + operation specific transport headers
  - Well known destination port
  - Source port is random for entropy like other UDP based tunneling protocols
  - QP flow steering based on QPN in BTH
- Basic RDMA ops and headers
  - SEND - only BTH (base transport header)
  - RDMA WRITE - BTH + reth extension header
  - RDMA READ - BTH + reth and aeth extension headers
- Requester - responder - completer model
- Message semantics
  - Multiple packets in a message use FIRST, MIDDLE, LAST modifications to opcode
  - H/W offload for large messages is similar to TSO
- Packets expected to arrive in-order

Networking Performance

- Tests with ConnectX-7 (400G ports)
  - easy, side-by-side comparison of RoCEv2 and full socket networking
- Server configuration - some are not realistic, just what is needed to push S/W to max
  - 9000 MTU, “Big TCP” (~512kB)
  - ethtool: gro, tso, gro-list, 8kB ring
  - large socket buffers (32M sendmsg)
  - hugepages
  - pinned flows
  - no netfilter rules
  - no qdisc
  - no packet sockets
  - no iommu (no VMs), spectre_v2 off
  - Zero copy Rx and Tx (for sockets, emulated via MSG_TRUNC)

Networking Performance - Socket Networking

- Max speed 215-220 Gbps for single flow; 397 Gbps for 2 flows
- Rx limited: softirq at 100%, Tx at 60%
    - `30.90%  [kernel]  [k] mlx5e_copy_skb_header`
    - `5.69%  [kernel]  [k] mlx5e_skb_from_cqe_mpwrq_nonlinear`
    - `4.56%  [kernel]  [k] mlx5e_add_skb_shared_info_frag`
    - `3.04%  [kernel]  [k] napi_pp_put_page`
    - `2.86%  [kernel]  [k] mlx5e_page_release_fragmented.isra.0`
    - `2.75%  [kernel]  [k] dma_sync_single_for_cpu`
    - `2.42%  [kernel]  [k] __napi_alloc_skb`
    - `2.21%  [kernel]  [k] mlx5e_build_rx_skb`
    - `2.08%  [kernel]  [k] __napi_build_skb`
    - `1.95%  [kernel]  [k] mlx5e_handle_rx_cqe_mpwrq`
    - `1.92%  [kernel]  [k] page_pool_put_unrefed_page`
    - `1.87%  [kernel]  [k] skb_release_data`
    - `1.83%  [kernel]  [k] skb_gro_receive`
    - `1.67%  [kernel]  [k] dev_gro_receive`
    - `1.54%  [kernel]  [k] tcp_gro_receive`
    - `1.52%  [kernel]  [k] inet_gro_receive`
    - `1.37%  [kernel]  [k] gro_list_prepare`
    - `1.15%  [kernel]  [k] dma_sync_single_for_device`
    - `1.02%  [kernel]  [k] mlx5e_post_rx_mpwqes`
    - `1.01%  [kernel]  [k] try_to_wake_up`
    - `1.01%  [kernel]  [k] page_pool_alloc_pages`
    - `1.00%  [kernel]  [k] page_pool_refill_alloc_cache`
- 400G for a single flow is a big stretch with full-stack socket networking

Networking Performance - RoCEv2

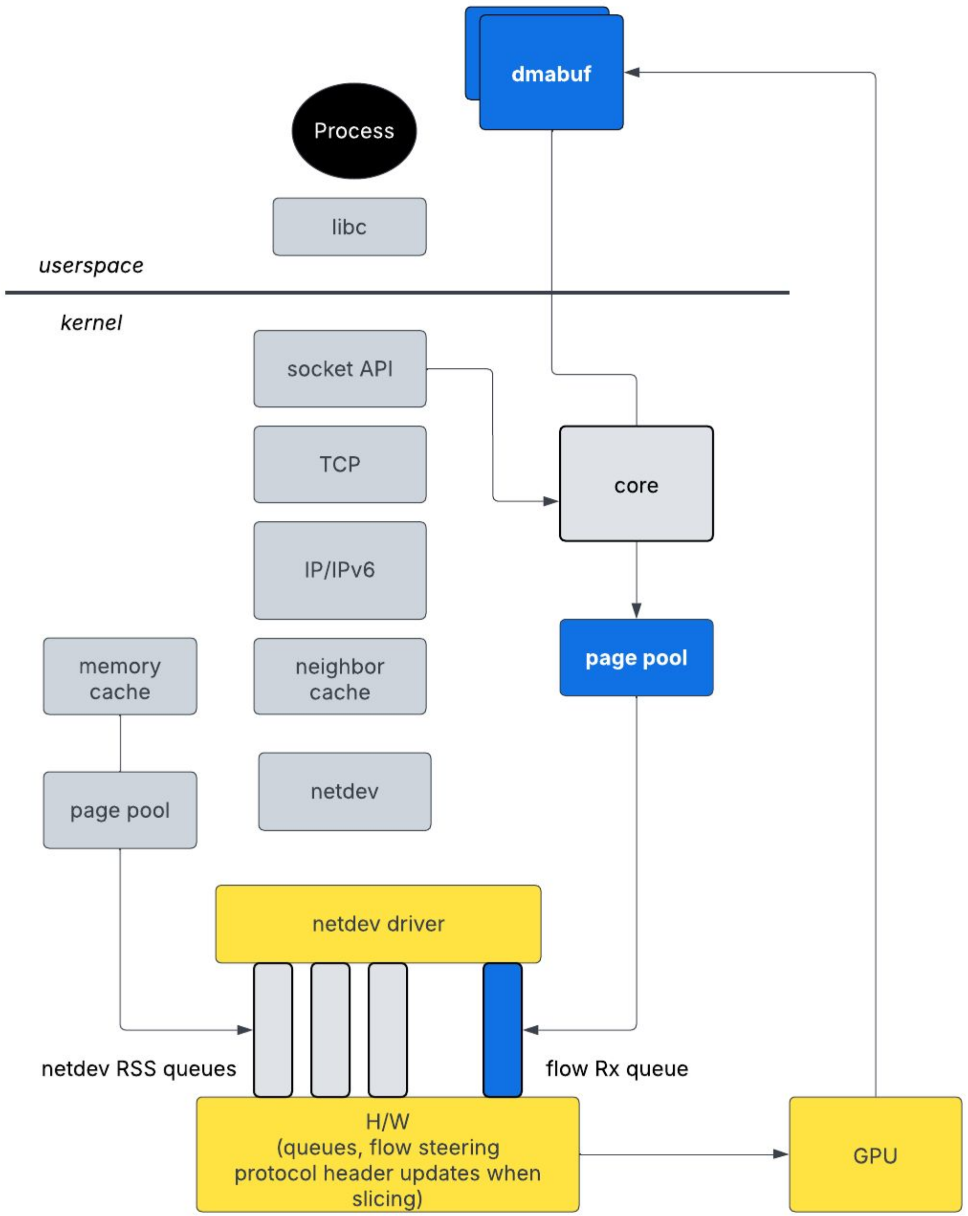
- Same hardware setup
  - ib_send_bw (IBV_WR_SEND)
    - Relies on buffers to be posted on Rx side (vs RDMA_WRITE)
  - Closest comparison to netdev model as possible
  - 2MB message size
  - 4096 RoCE MTU
  - 392 Gbps single flow

How RDMA and RoCEv2 avoids the overhead

- Clear separation between control and data paths
- Pedantic control path configuration, no fallback, everything must be supported by HW
- Data path is fully offloaded
  - H/W manages the mechanical task of generating wire packets
- User owns data, no syscalls
- Memory regions vs anonymous buffers
  - No kernel side page pool to manage - page pool is overhead
- Zerocopy and in-order payloads
  - Payloads are landed in message and byte order directly to user managed MRs
  - No need for linearizing and returning pages to a pool
- Not building skb’s per packet (or GRO packet)
- Very strict object lifetimes (verbs objects)
- No reconfiguration of data path

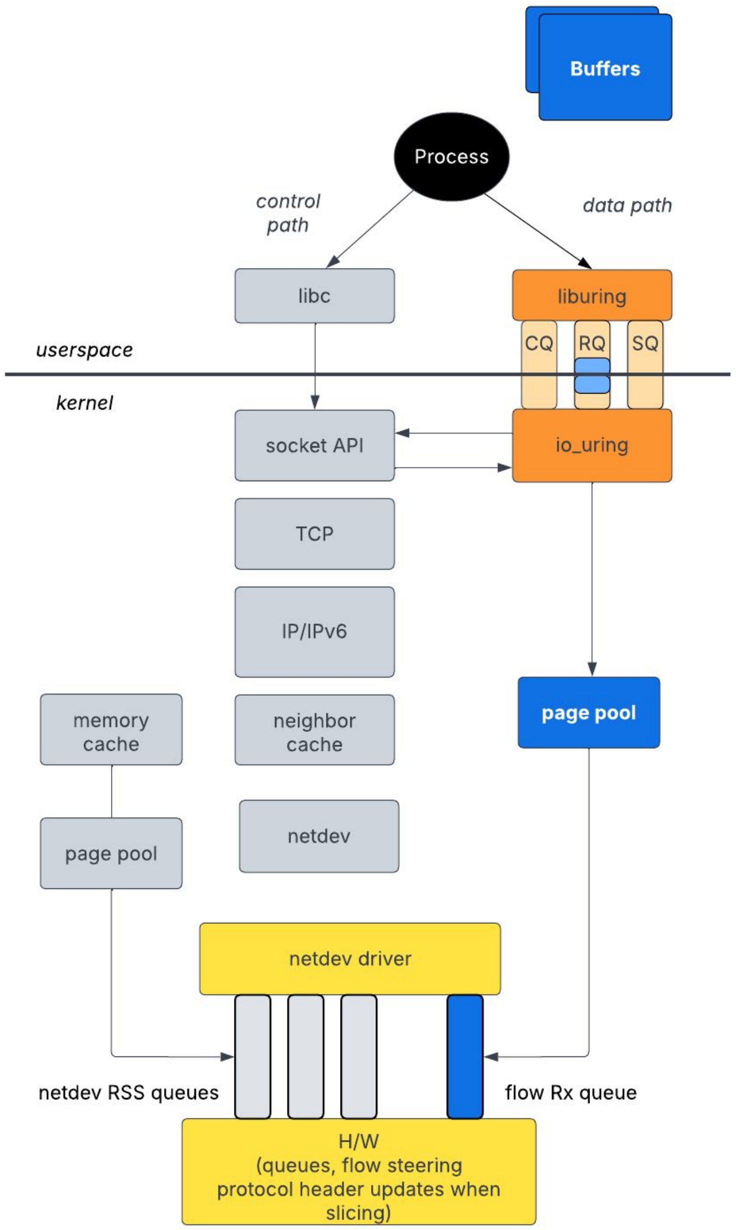
Linux TCP and “devmem”

- Recent feature (v6.12 for Rx) to enable use of GPU memory with TCP
  - Tx ZC with GPU memory is a WIP
- Uses a page pool with GPU memory via dmabuf and dedicated Rx queue for flow
  - syscalls are needed to recv indication of filled buffers (recvmsg + cmag)
  - syscalls to return buffers to page pool (setsockopt)
- Zerocopy into GPU memory, but in an anonymous memory pool style
  - Not direct data placement with in order payloads
  - Buffers with packet data need to be linearized (copied into buffer to be consumed by GPU thread)
  - yes, HBM - but still overhead
- Good for low 100Gbps range



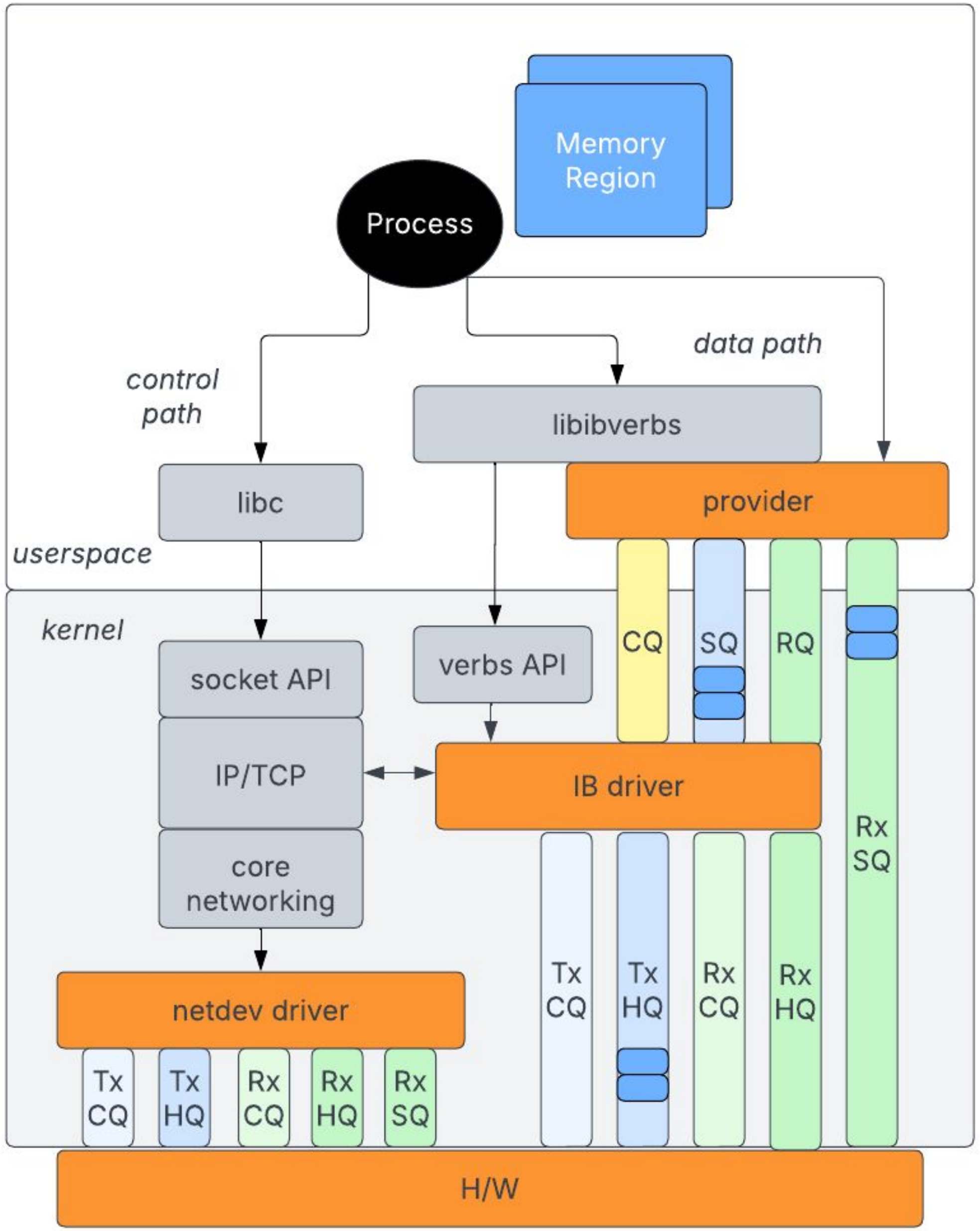
Linux TCP and io_uring

- Control path - normal socket APIs
- Data path - socket read, write, etc managed through io_uring APIs
  - User-kernel queues avoid networking syscalls
  - io_uring kicks are needed
- Rx ZC with cpu memory merged for v6.15
  - page_pool for supplying buffers to H/W for flow
  - fallback to copy mode
- Tx ZC merged for v6.1
- Neither ZC support handles GPU memory
- Single flow performance in patch set listed as 116G



Linux TCP with QPs

- Proposal for using Linux TCP with QPs
- Control path - normal socket APIs
  - Includes configuring Big TCP, hardware offloads (H/W GRO, TSO)
- QP for a TCP based flow
  - Flow specific hardware queues
- Data path - socket management handed off to IB driver
  - Bypass unnecessary overhead top to bottom
- Linux TCP
  - Congestion control decides when to send payload and how much
  - Manages ACKs, SACKs, TSN, retries



Linux TCP with QPs - Message Framing

- Need a message framing solution for TCP
  - TCP option does not pollute payload bytes
- Message framing requirements:
  - Message sequence number
  - Needs to include opcode (e.g., SEND with FIRST, MIDDLE, LAST variants)
  - Robust solution has each packet self describing its place within the message (i.e., offset in message)
  - Need to handle 4B immediate data and icrc too
- Constraint: TCP options limited to 40B
  - Always having timestamp option is best practice (12B)

Linux TCP with QPs - TCP option for Messages

- Essential BTH data + timestamp exceeds TCP option length
    - Fold timestamps into BTH
- Hardware can correlate any packet to a message and offset within the message
    - Knows where to place payload even if packet arrives out-of-order

```
struct tcp_bth {
        __u8    ver:2,
                timestamp_present:1,
                more_segments:1,
                rtr:1,
                rtr_ack:1,
                rsvd:2;
        __u8    opcode;       /* RDMA opcode */
        __be16  msn;          /* message seq number */
        __be32  mbo_seg;      /* 24b message byte offset;
                                 8b segment number */
        __be32  icrc;

        __be32  timestamp;
        __be32  timestamp_echo;

        union {
                __be32 imm_data;
                /* other extensions */
        };
};
```

Final Thoughts

- Socket networking
  - well established paradigm
  - useful for general services and use cases that need scaling to 100k+ sockets per server
  - has well known limitations on its performance
  - one small part of the “netdev” code base which includes protocols, device models, etc
- RDMA and IB stack
  - specifically designed for demanding, performance sensitive use cases
- RoCEv2 is one form of bringing the 2 together
  - UDP/IP over ethernet
  - Reuses netdev model for addresses, routes, neighbor cache, port state
- Possible for more - e.g., Linux TCP with RDMA QPs
- About building blocks and letting users decide what trade-offs work for them

Final Thoughts

- Future netdevconf workshops
  - NCCL, for example, supports sockets and RDMA - side by side examples of performance
  - congestion control, spraying (message and packet)
  - top-of-mind ideas and discussion points

References

https://engineering.fb.com/2024/08/05/data-center-engineering/roce-network-distributed-ai-training-at-scale/

https://www.tomshardware.com/tech-industry/artificial-intelligence/xai-colossus-supercomputer-with-100k-h100-gpus-comes-online-musk-lays-out-plans-to-double-gpu-count-to-200k-with-50k-h100-and-50k-h200

https://www.nextplatform.com/2024/03/13/inside-the-massive-gpu-buildout-at-meta-platforms/

https://www.nextplatform.com/2024/08/23/this-ai-network-has-no-spine-and-thats-a-good-thing/

Thank You