nvm EXPRESS®

Bringing NVMe®/TCP Up to Speed

Sponsored by NVM Express organization, the owner of NVMe®, NVMe-oF™ and NVMe-MI™ standards

Sagi Grimberg, CTO, Lightbits

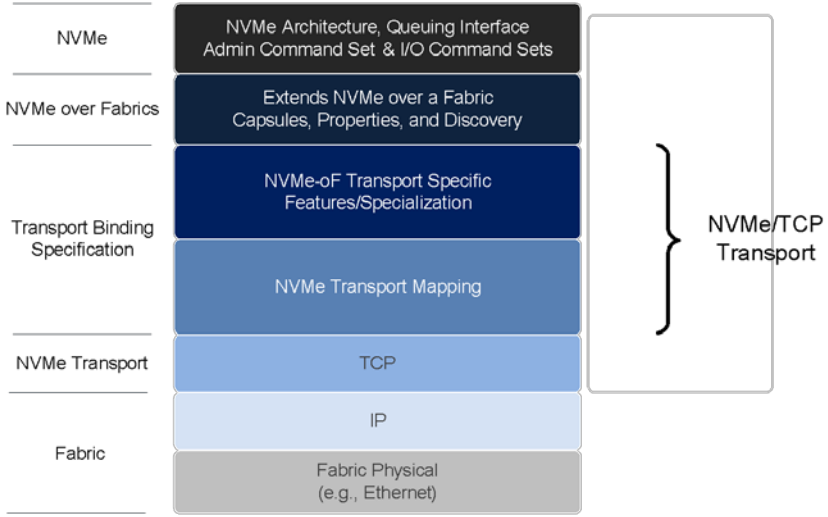
Speakers

Sagi Grimberg

lightbits

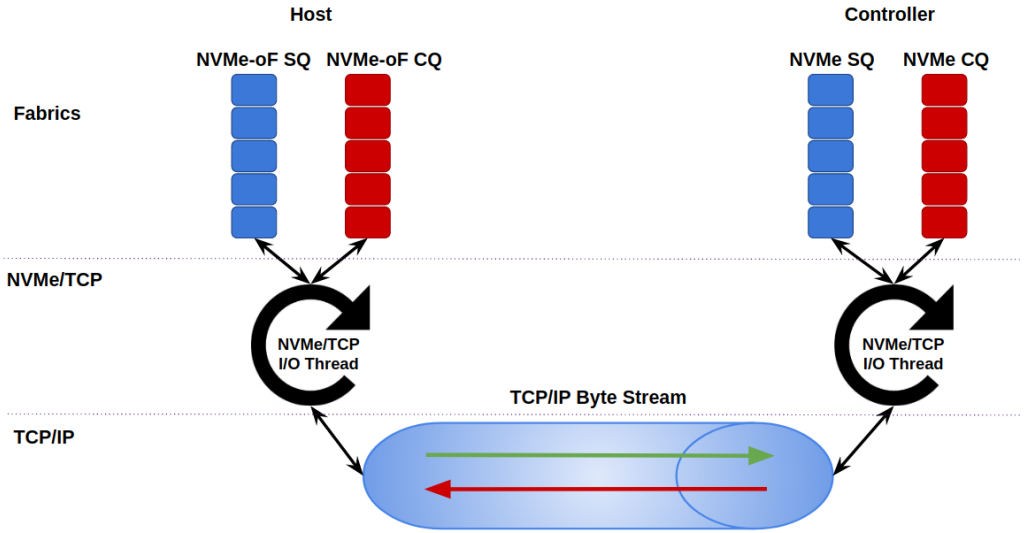
NVMe®/TCP Technology (Short) Intro

- NVMe/TCP technology is the standard transport binding to run NVMe architecture on top of standard TCP/IP networks
- Standard NVMe specification multi-queue interface runs on top of TCP sockets
- Same NVMe command set, encapsulated over NVMe/TCP PDUs



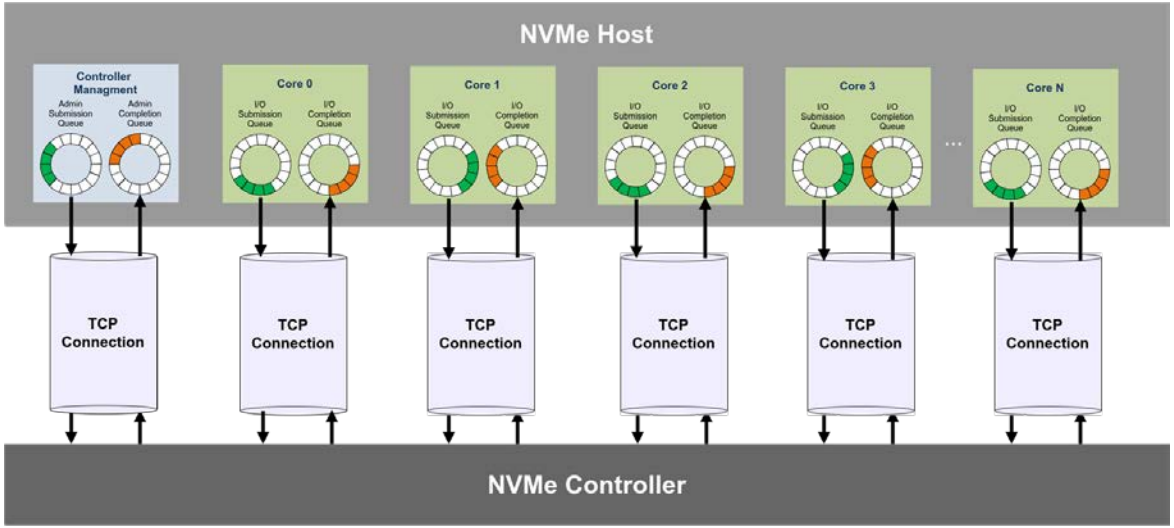
NVMe®/TCP Technology (Short) Intro

- Each NVMe queue-pair is mapped to a bidirectional TCP connection
- Commands and data-transfer are processed by a dedicated context



NVMe®/TCP Architecture Queue Mapping

- Each NVMe queue normally mapped to a dedicated CPU core
  - But not necessarily
- No controller-wide serialization



Latency Contributors

- **Serialization** - Lightweight, only on a per-queue basis (and hctx, sockets etc) - scales pretty well
- **Context Switching** - 2 at a minimum contributed by the driver
- **Memory copy** - Only on RX, not a huge contributor (sometimes is on high load)
- **Interrupts** - Definitely impactful, LRO/GRO/Adaptive-moderation can mitigate a bit, but latency is less consistent
- **socket overhead** - Exists, but not huge, mostly around small size RX/TX
- **Affinitization** - Definitely a contributor if not affinitized correctly
- **Cache pollution** - Has some, not excessive
- **Head-of-Line blocking** - Can be apparent in mixed workloads

Host Direct-IO Flow

- **User issues issues direct file/block I/O** (*ignoring the rest of the stack*)
- **nvme_tcp_queue_rq prepares NVMe®/TCP PDU and place it in a queue**
- **nvme_tcp_io_work context picks up I/O and process it**
- **I/O completes, controller sends back data/completion to the host**
- **NIC generates interrupt**
- **NAPI is triggered**
- **nvme_tcp_data_ready is triggered**
- **nvme_tcp_io_work context is triggered, processing and completing the I/O**
- **user context completes I/O**

Host Direct-IO Flow

- **User issues issues direct file/block I/O** (*ignoring the rest of the stack*)

Context-Switch

- **nvme_tcp_queue_rq prepares NVMe/TCP PDU and place it in a queue**
- **nvme_tcp_io_work context picks up I/O and process it**
- **I/O completes, controller sends back data/completion to the host**

Soft-IRQ

- **NIC generates interrupt**
- **NAPI is triggered**

Context-Switch

- **nvme_tcp_data_ready is triggered**
- **nvme_tcp_io_work context is triggered, processing and completing the I/O**
- **user context completes I/O**

Mixed Workload Optimization

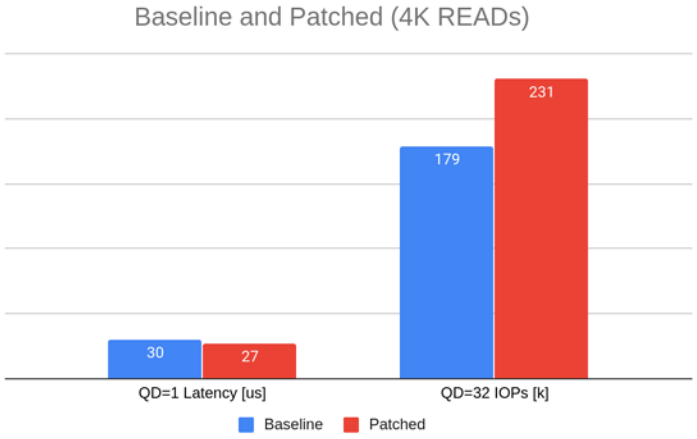
- **Linux block layer allows for multiple queue maps**
  - Default: normal set of HW queues
  - Read: Dedicated queues for Reads
  - Poll: Dedicated queues for polling application and RWF_HIPRI I/O
- **Eliminate Head-of-Line blocking of small reads vs. large writes**
  - Send Reads on dedicated read queues, and writes on default queues
- **Added support for multiple queue maps and plugging into the block layer**

Test: 16 readers issuing synchronous 4K reads, 1 unbound writer issuing 1M writes @QD=32

	READ IOPs [k]	READ Ave Latency [us]	READ 99.99% latency [us]
Baseline	80.4	396	14222
Patched	171	181.5	1811

Affinity Optimizations

- **Linux grew capability to split different I/O types to different queue maps**
- **optimize queue io_cpu assignment for multiple queue maps**
  - Use separated alignment for different queue maps (read/default/polling)
  - Calculate each queue map alignment individually
  - Especially important for Read and Poll queue maps



Low QD Latency Optimizations - TX Path

- **Eliminate NVMe®/TCP context switch when queuing a request**
  - **Prepare NVMe/TCP technology and process directly from nvme_tcp_queue_rq**
  - **Network send might_sleep, so need to convert hctx locking to srcu**
  - **Serialize of two contexts of the same queue is required**
    - Introduce a mutex
    - Only if the queue is empty
    - Only if the queue mapped CPU matches the running cpu
- **Socket priority**
  - Steers egress traffic to the preferred NIC queue set

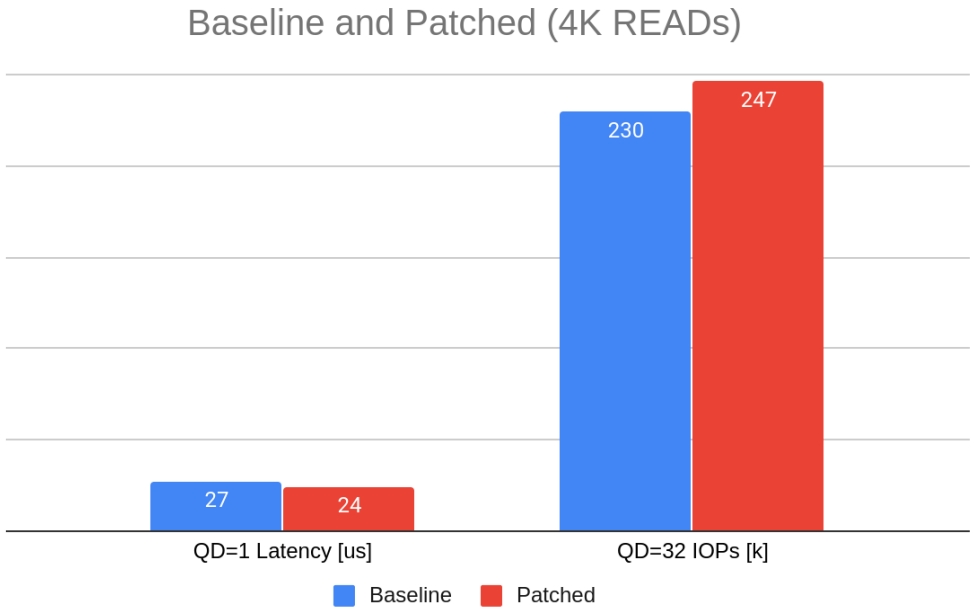
Low QD Latency Optimizations - RX Path

- **Linux grew a polling interface for latency sensitive I/O**
  - Submit with RWF_HIPRI
  - Poll for completion (also via io_uring IORING_SETUP_IOPOLL)

- **We add nvme_tcp_poll and plug it into blk_poll interface**
  - Add dedicated queues for polling (connect options)
  - nvme_tcp_poll calls sk_busy_loop

- **Skip RX data_ready context switch if application is polling at the same time**
  - Mostly true if NIC moderation is working well
  - If device can hold off interrupts more aggressively it works very well

Low QD Latency Optimizations - Results



ADQ improvements

Traffic Isolation - Direct NVMe® technology traffic to its dedicated queue set

- Inbound:
  - Dedicated queue-set configuration (tc-mqprio)
  - Traffic Filtering (tc-flower)
  - Queue selection (RSS/Flow Director)
- Outbound:
  - Set Socket priority
  - Extensions to Transmit Packet Steering (XPS)

Value

- No noisy traffic from neighbor workloads
- Opportunity to customize network parameters for a specific workload

ADQ Improvements

Minimizing Context switching and Interrupts overhead

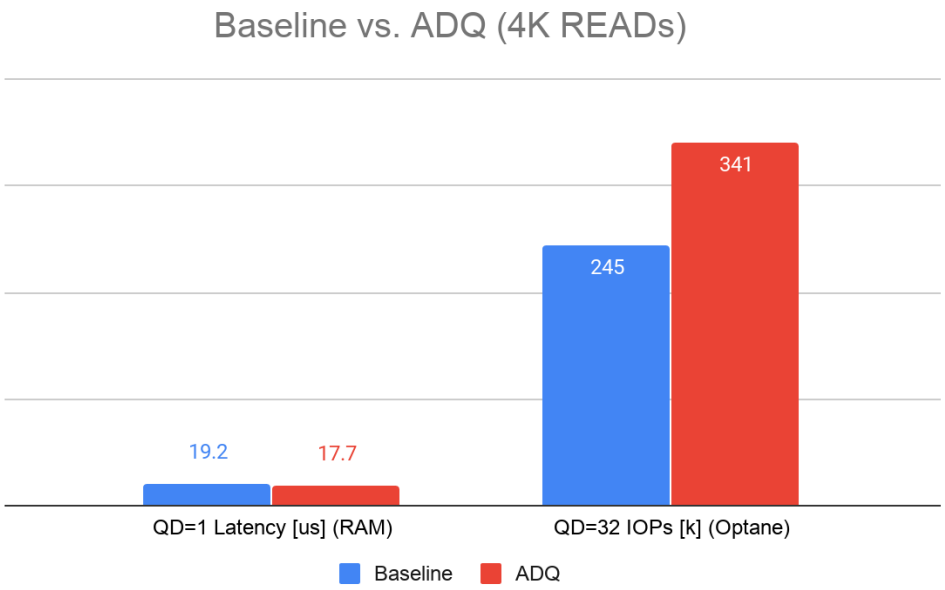
- Busy polling on dedicated queue set
  - Drain network completions in application context
  - Process NVMe® technology completions directly in application context
- Handle Request/Response in application context
  - Keeps the application thread active - no redundant context switch
- Grouping multiple NVMe®/TCP queues to a single NIC HW queue
  - Streamlines sharing of a NIC HW queue - no redundant context switch

Value

- Reducing CPU utilization
- Lowering Latency

ADQ Measurements

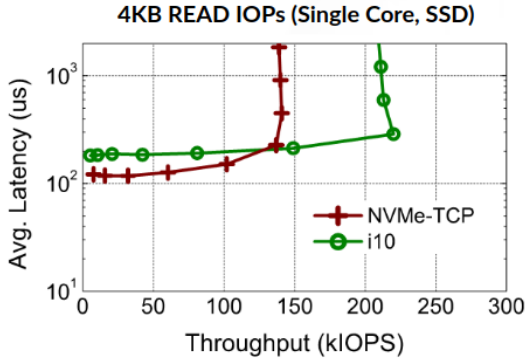
- Comparing NVMe/TCP with ADQ enabled vs. ADQ Disabled
- Platform is Cascade-Lake

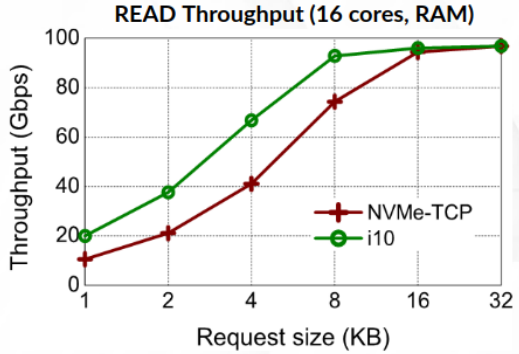


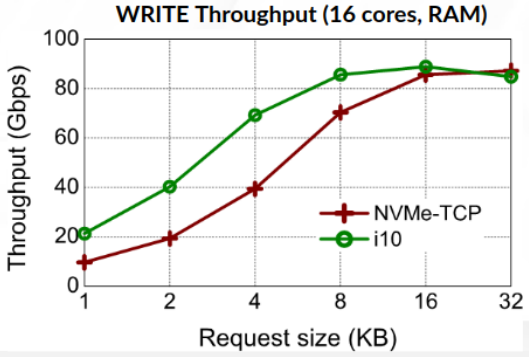
High QD Latency Optimizations - Batching

- We want to leverage information about build up of a queue (opportunity to batch)
  - The block layer indicates the driver if request is the last one or more is coming (bd->last indicator)
- Modified the driver send queue from list (protected by a spinlock) to a lockless list
  - I/O thread pulls from list in batches, has a better view of what is coming
  - Schedule I/O thread only when the “last in batch” arrives…
- Optimized network MSG flags based on this information: MSG_MORE, MSG_SENDPAGE_NOTLAST (and MSG_OER if last in batch)
- Improve batching support in blk-mq in case of I/O schedulers [Ming Lei]
- Implemented an optimized batching scheduler for TCP stream based storage devices
  - i10 paper *[Jaehyun Hwang, Qizhe Cai Ao Tang, Rachit Agarwal Cornell University]*

High QD Latency Optimizations - Results







Questions?


nvm
EXPRESS®
Architected for Performance