Latest Developments with NVMe/TCP

Sagi Grimberg
Lightbits Labs
NVMe-oF - Short Recap

- Early 2014: Initial NVMe/RDMA pre-standard prototypes
- Later 2014: NVMe-oF 1.0 standardization
- 2015: NVMe.org formed Fabrics Linux Driver Task Force
  - Converged on the starting point prototype
  - Developed from there on a main git repository
  - Heavy lifting of NVMe stack reorg contributed even before NVMe-oF support
- 2016 (Oct): NVMe over Fabrics support (host and target) merged into the Linux kernel (v4.8)
NVMe-oF - Since than...

- Hardening: various stability fixes
- Instrumentation: tracing support
- Additional transports: FC
- Tool chain enhancements:
  - queue_size, nr_io_queues, hostnqn, hostid, reconnect_delay, ctrl_loss_tmo, kato, host_traddr, duplicate_connect etc...
- Compliance: UUID support
- Enhancements: TCP Opal support, I/O Polling support, ANA support, etc..
NVMe-oF 1.1

- **Scope:**
  - NVMe/TCP
  - Dynamic Resource Enumeration
  - SQ flow control disabled mode
  - Traffic Based Keep Alive

- The Fabrics Linux Driver Task Force is developing the TPs to both drive the spec and provide early adoption to enhancements
Why NVMe/TCP?

- Ubiquitous - runs on everything everywhere…
- Well understood - TCP is probably the most common transport.
- High performance - TCP delivers excellent performance scalability.
- Well suited for large scale deployments and longer distances
- Actively developed - maintenance and enhancements are developed by major players.
Ratification Status

- Technical proposal Entered the final stage of 30-day member review
- After that will undergo integration and board ratification approval
- Multi-vendor joint effort to ensure interoperability
- Designed for simplicity and efficiency
Association Model

- Controller association maps 1x1 NVMe queue to a TCP connection
  - No controller-wide sequencing
  - No controller-wide reassembly constraints
- Connection binding is performed in NVMe-oF connect time (binding queue to controller)
Protocol Data Unit

- NVMe-oF Capsules and Data are encapsulated in PDUs
- PDU structure varies per PDU type
  - 8-byte Common Header
  - Variable length PDU specific header
- PDUs optionally contain Header and/or Data digest protection
- PDUs contain optional PAD used for alignment enhancements
PDU Types

- ICReq/ICResp: Connection Initialization PDUs
- H2CTermReq/C2HTermReq: Connection Termination PDUs (only for error flow)
- CapsuleCmd/CapsuleResp: NVMe-oF Command and Response Capsule PDUs
- H2CData/C2HData: Unidirectional Data PDUs
- R2T: Ready to Transfer PDU (solicit H2CData)
I/O flow

- Host to Controller data can come in-capsule (if the controller supports it) or in a solicited H2CData PDU (R2T PDU)
NVMe/TCP - Linux Support

- 2017: Fabrics Linux Driver Task Force took on developing the NVMe/TCP driver
  - Two prototypes existed (Lightbits, SolarFlare)
  - Converged on a single code-base moving forward

- As the spec evolved, code adjustments followed providing feedback to the standardization

- Code is in solid shape, expected to be submitted with close proximity to the spec ratification
Driver Design Guidelines

- Single reactor thread per-cpu (private bound workqueue)
  - Keep context switches to an absolute minimum
  - NVMe queues are spread among reactor threads
- NEVER block on I/O (unlike other TCP implementations)
- Aggressively avoid data copy (only copy RX data)
- Reuse common interfaces!
  - bio_vec, iov_iter, socket/datagram operations
- RX is either handled in soft-IRQ or in the same reactor context
- Keep atomic operations to an absolute minimum and keep uncontended (in the data path)
- Fairness and budgeting mechanisms multiplexing between NVMe queues
NVMe Host Stack

- TCP transport driver naturally plugs into the existing stack
- Almost no special additions for TCP (to this point)
- Control plane very similar to RDMA
  - Still have plenty of room for code reuse
NVMe Target Stack

- TCP transport driver naturally plugs into the existing stack
- Very few changes to the existing core/fabrics stack
The Linux NVMe subsystem is in pretty good shape (most of the code is common)
Still has plenty of room for improvement in error handling code reuse
Online Data Digest Hash Updates

- We use existing `skb_copy_datagram_iter` interface for incoming data placement
  - Both skb and bio_vec walks are abstracted away

- **Problem**: Calculate data digest while data is still hot in the cache
- **Solution**: Provide new interface that will perform online digest updates
  - `skb_copy_and_hash_datagram_iter` which receives a pre-initialized ahash_request and updates it on the fly
  - Provides the same level of abstraction and allows consumer to not open code iterator walks when online digest operations are needed
slab, sendpage, and kernel hardening

- NVMe/TCP PDU headers are pre-allocated from the memory allocator
- Like any other buffer, PDU headers are never copied when sent to the network
- When the queue depth is high and network is congested, PDU headers might get coalesced together
- Kernel Hardening will panic the kernel when usercopy attempts to read slab originated buffer if they cross slab objects
  - Heuristic attempt to catch an exploit
slab, *sendpage*, and kernel hardening

- Userspace programs is allowed to use packet filters and read
  - bpf, tap (nit), etc…
  - dhclient was the culprit in this case
- Basically every userspace program can panic the kernel if a slab page will be passed to *sendpage*
- **Solution**: Page fragments API
  - NVMe/TCP PDUs will keep a per-queue `page_frag_cache` that will use normal page allocation not originating from the memory allocator
  - Also very cacheline friendly for network stack page refcounts (avoid cpu cores sharing atomic areas)
Features

- Zero Copy Transmission
- Header/Data Digest
- CPU/NUMA affinity assignments for I/O threads
- TLS Support - Future
  - Will probably be trampolined to userspace for TLS handshake
- Polling Mode I/O - Future
  - Need to continue polling with an inherent context switch
- Automatic aRFS support - Future
  - Need to figure out atomicity of NIC steering table updates
- Out-of-Order Data transfers - Future
  - Probably fabrics 1.2 material
In-Transit Encryption - TLS

- A NVMe/TCP enabled subsystem port can optionally support TLS
  - TLS indication appears in the TSAS field of the discovery log entry
  - One mandatory cipher-suite, three other recommended

- TLS handshake in Linux is supported in userspace

- Two possible implementations:
  - Trampoline to userspace that implements TLS handshake
  - Port TLS handshake to the kernel

- Support is expected post initial submission
Canonical Latency

- Latency is still higher than RDMA but still pretty good

- @ the tail percentiles the difference is not even noticeable

* 100% 4KB Random Read @ QD=1 (using RAM device)
Scaling with threads

- Multithreaded application using synchronous I/O

- IOPs scale linearly and Latency is almost not affected

* 12 cpu cores @ 2GHz (Intel® Xeon® Processor E5-2620)
Host side CPU utilization

- CPU utilization decreases with higher block size
  - NIC offloads come into play (LRO, TSO)

- READs are naturally more intensive than WRITE
  - Involves data copy

* 12 cpu cores @ 2GHz (Intel® Xeon® Processor E5-2620)
* Mellanox ConnectX-4/LX
* QD=32/64
Q&A
Backup
Dynamic Resource Enumeration

- A Discovery Subsystem provide information on available fabric subsystems to hosts
  - Theory of operation is to simply connect, retrieve discovery log page and teardown

- Dynamic Resource Enumeration enables discovery subsystems to notify interested hosts about new fabric subsystems
  - Define persistent discovery controllers (support Keep Alive)
  - Define discovery log change AEN
Dynamic Resource Enumeration
Dynamic Resource Enumeration

- Host1
- Host2
- Host3
- Host4

Discovery Service

- Subsystem1
- Subsystem3
- Subsystem2
Dynamic Resource Enumeration