IT'S A BIRD... IT'S A PLANE... IT'S NVME OVER TCP AND MORE

Presenters: Seth Howell, Ziye Yang

Company: Intel

Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks .

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/benchmarks .

Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown." Implementation of these updates may make these results inapplicable to your device or system.

Intel® Advanced Vector Extensions (Intel® AVX)* provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at http://www.intel.com/go/turbo.

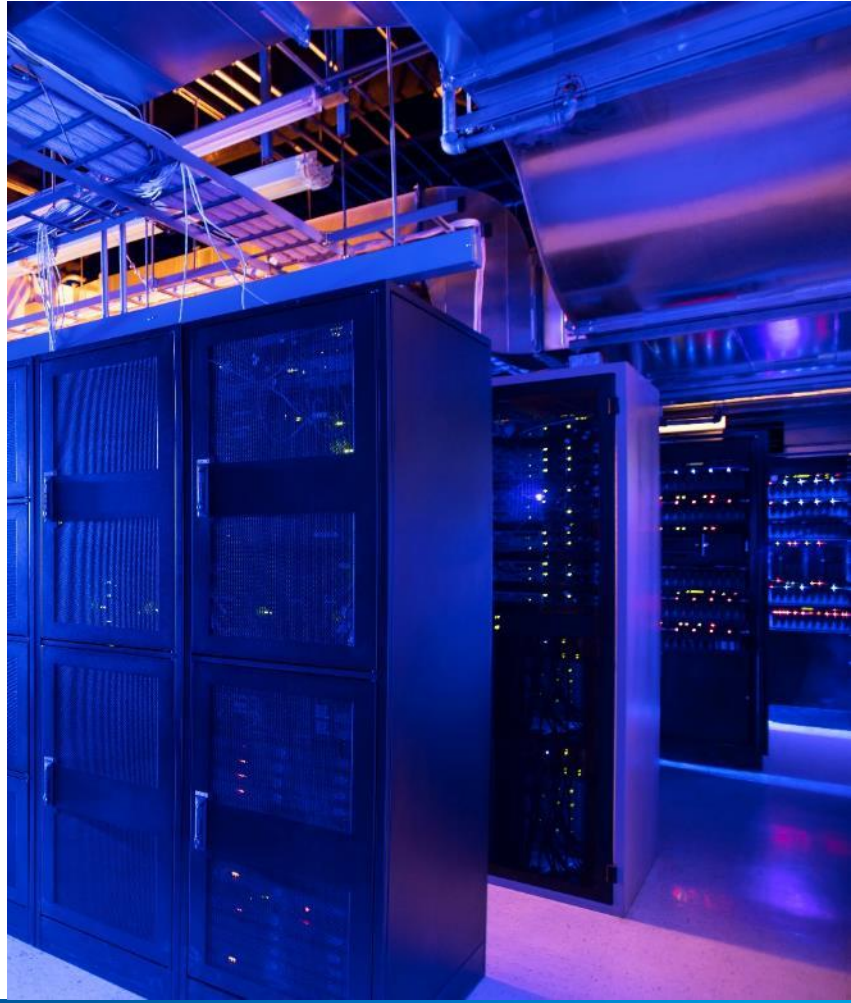
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

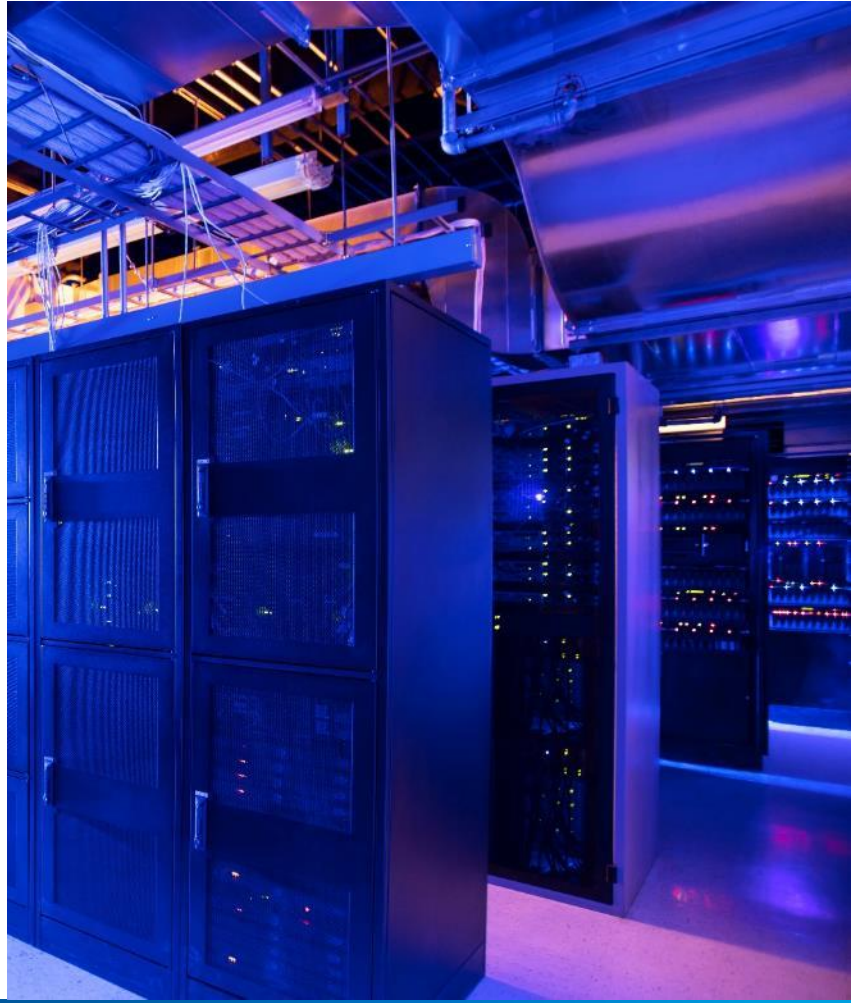
AGENDA

- SPDK NVMe-oF development history & status
- SPDK RDMA transport enhancement
- SPDK TCP transport introduction
- Conclusion

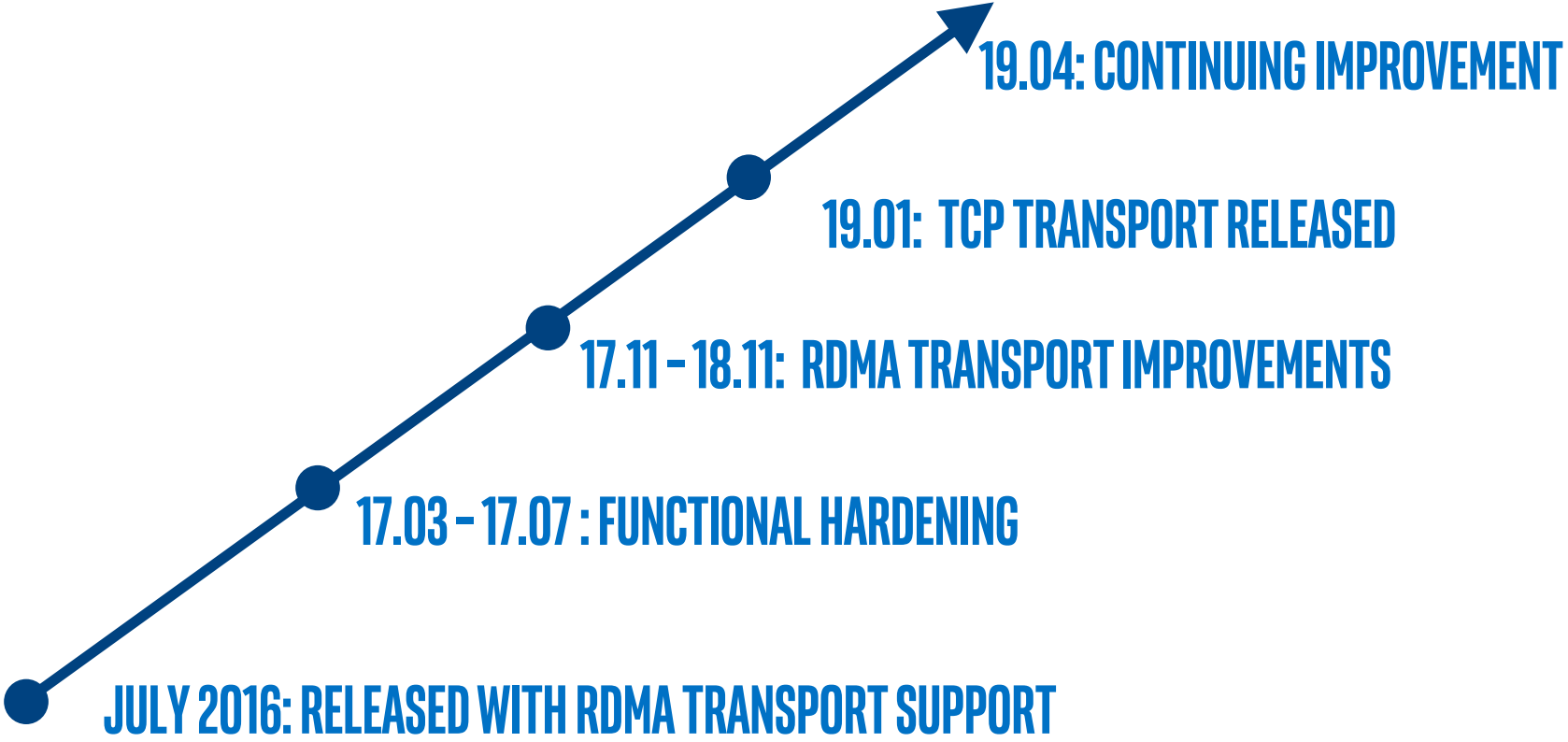


AGENDA

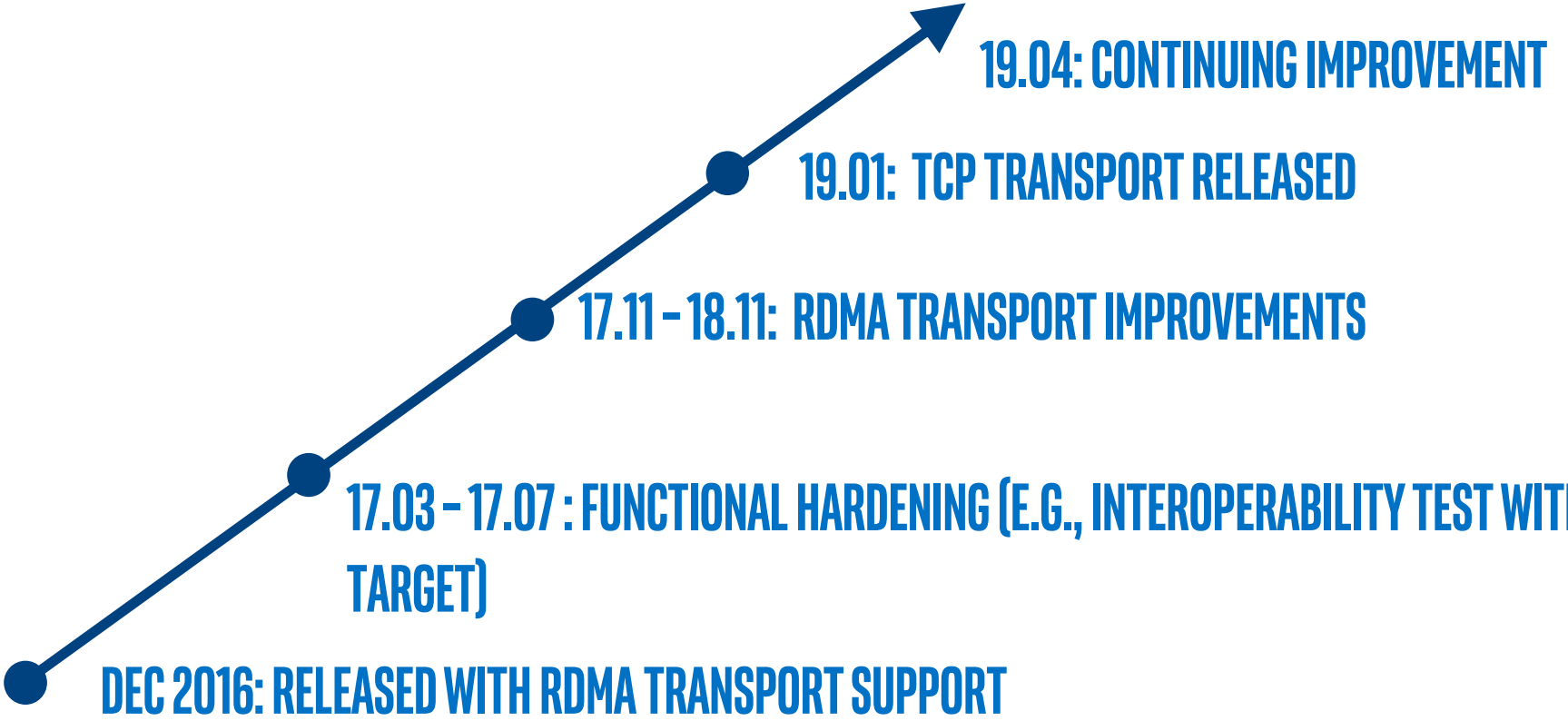
- **SPDK NVMe-oF development history & status**
- SPDK RDMA transport enhancement
- SPDK TCP transport introduction
- Conclusion



SDPK NVMe-oF Target Timeline



SDPK NVMe-oF host Timeline

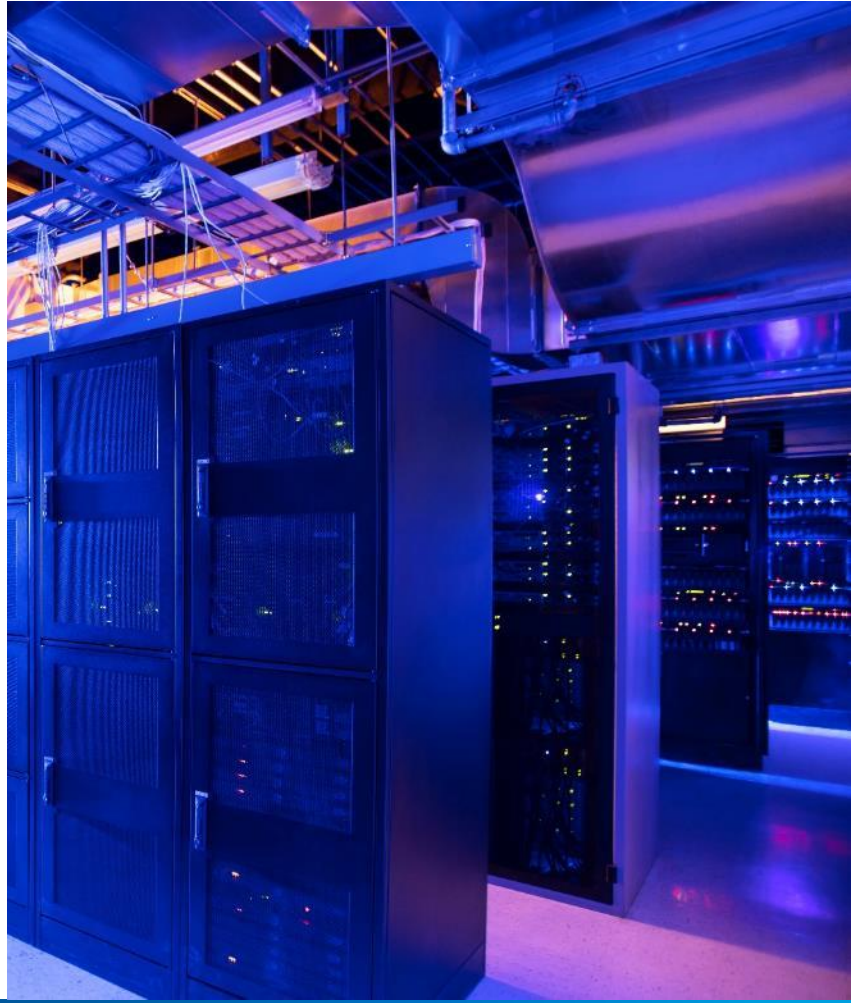


SPDK NVMe-oF target design highlights

NVMe* over Fabrics Target Features	Performance Benefit
Utilizes user space NVM Express* (NVMe) Polled Mode Driver	Reduced overhead per NVMe I/O
Group polling on each SPDK thread (binding on CPU core) for multiple transports	No interrupt overhead
Connections pinned to dedicated SPDK thread	No synchronization overhead
Asynchronous NVMe CMD handling in whole life cycle	No locks in NVMe CMD data handling path

AGENDA

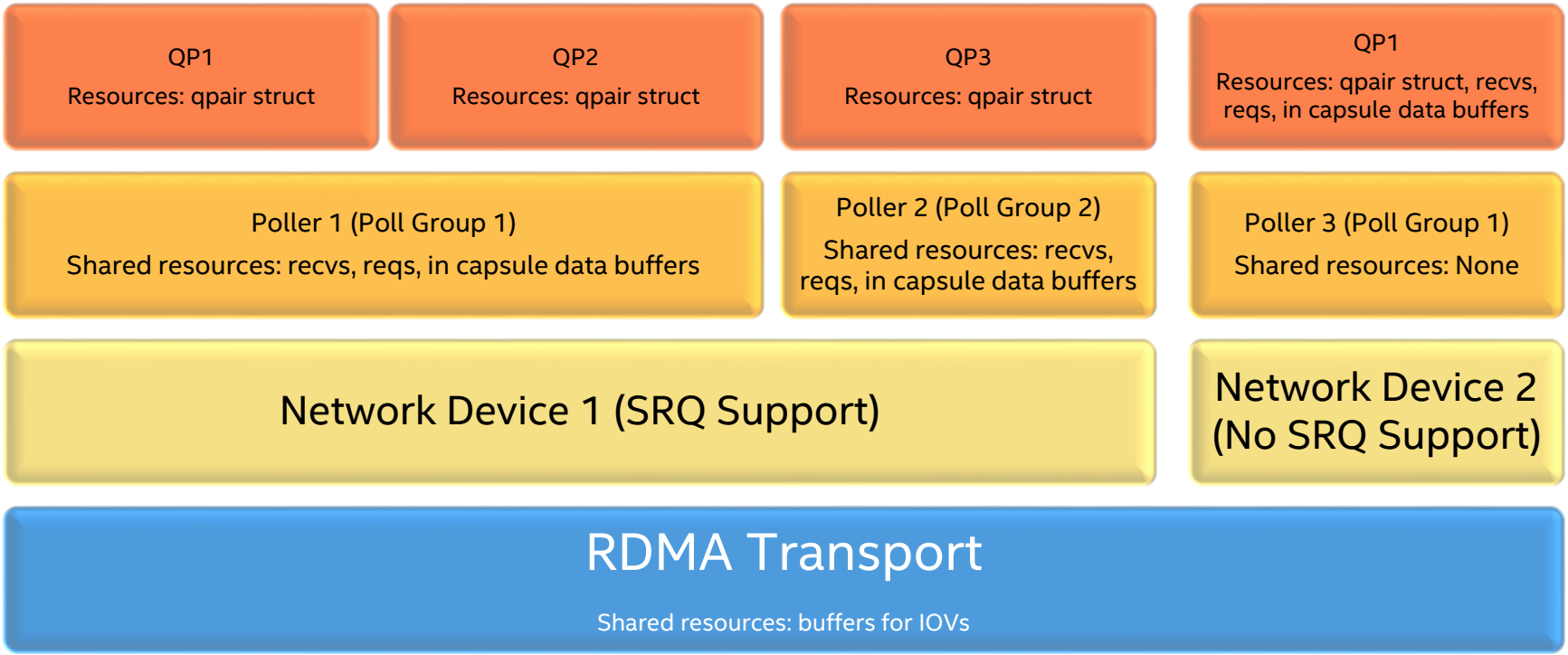
- SPDK NVMe-oF Development history & status
- **SPDK RDMA transport enhancement**
- SPDK TCP transport introduction
- Conclusion



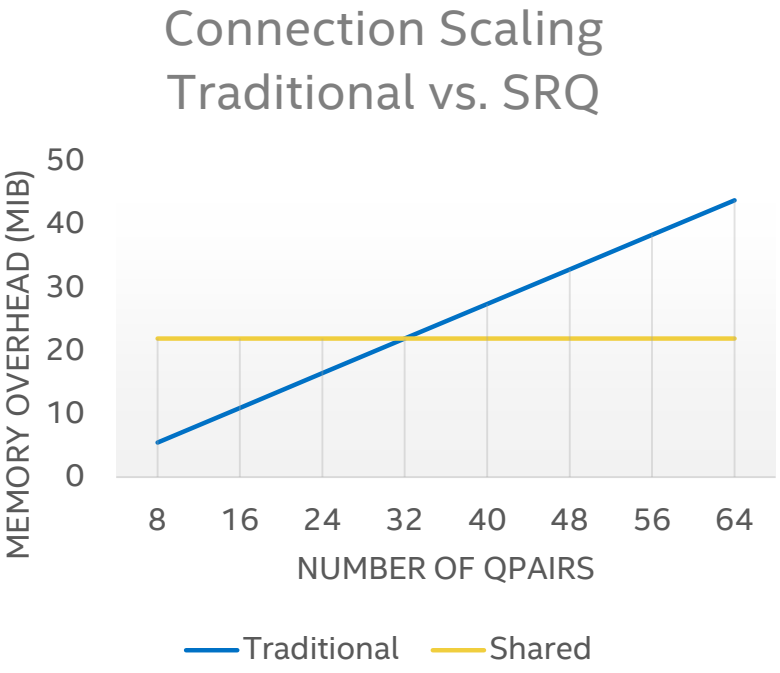
SHARED RECEIVE QUEUE SUPPORT

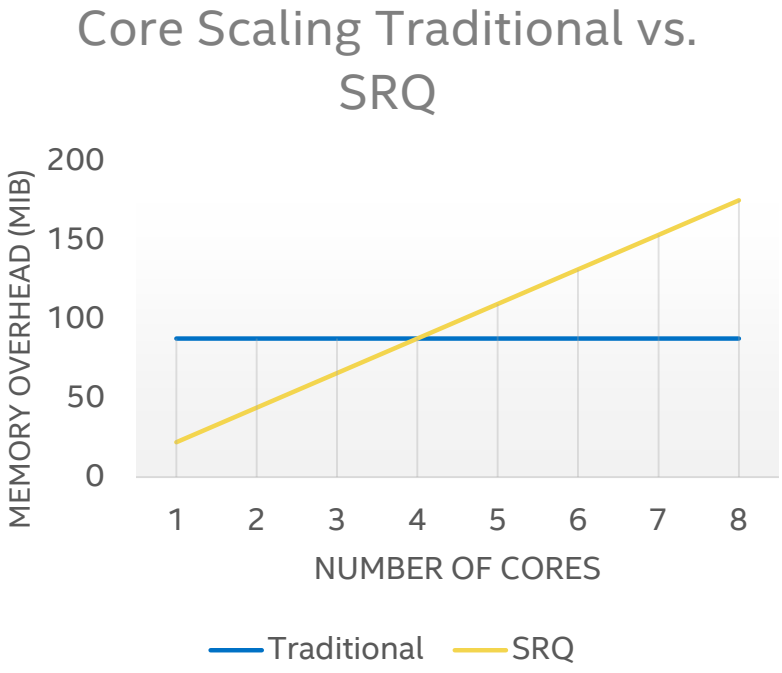
- Disaggregate requests and buffers from queue pairs.
- Fix the majority of allocations to startup.
- Scaling connections is much cheaper.
- Can increase the cost of core scaling.

NVMe-oF RDMA Transport Target Architecture



Scaling Connections and Cores



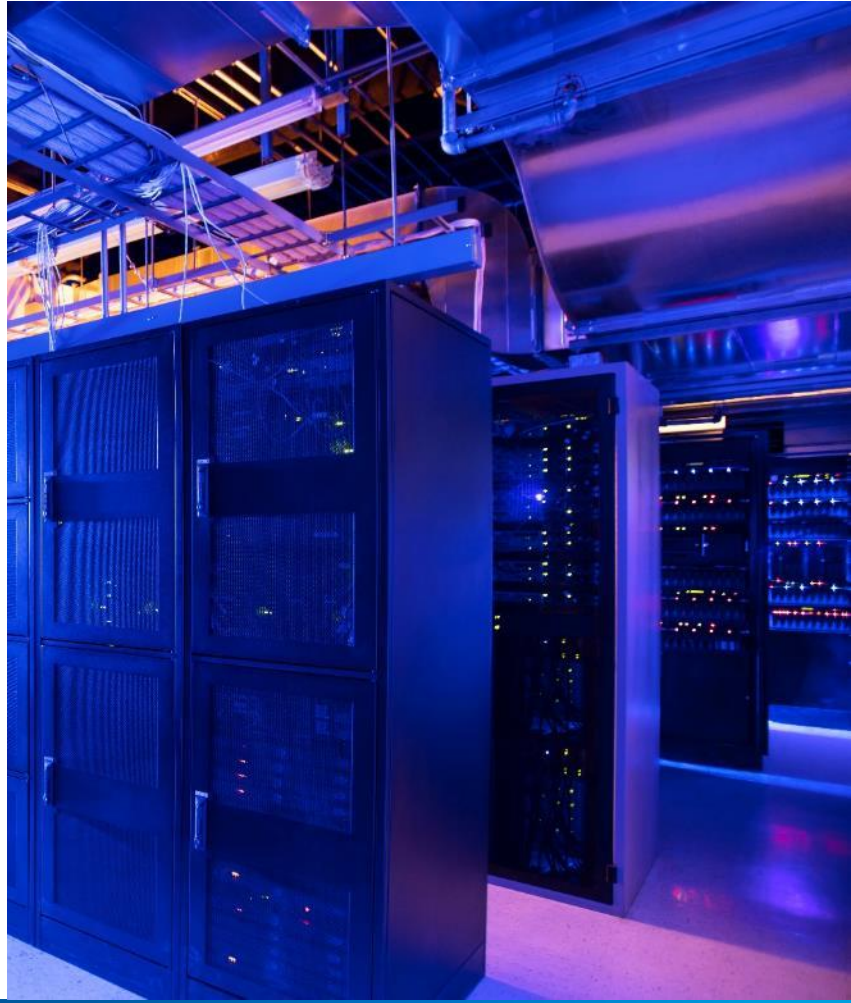


Further Enhancements

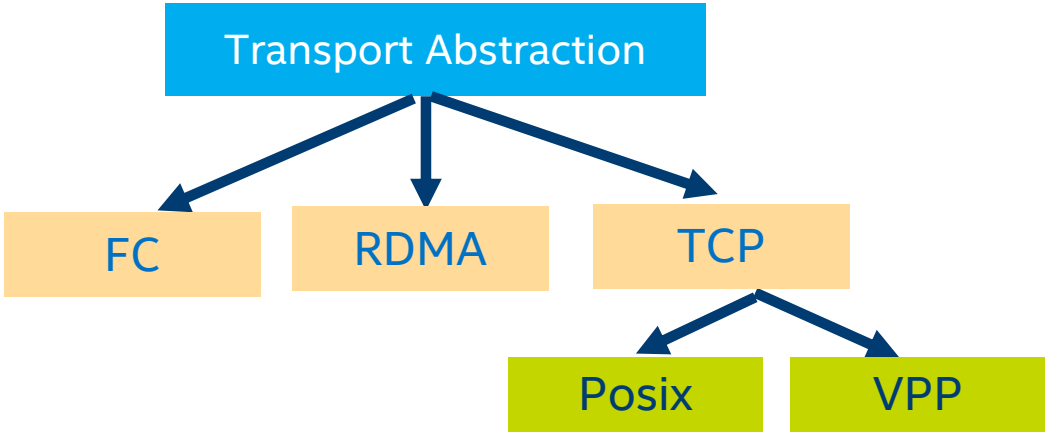
- Completed
  - Send With Invalidate Support
  - Multi-SGE in both Target and Initiator
- Ongoing and Future
  - Support for an increasing number of NICs from multiple vendors
  - SEND and RECV operation batching
  - Automated performance testing
  - Extended verbs interface
  - NVMe-oF offload support

AGENDA

- SPDK NVMe-oF Development history & status
- SPDK RDMA transport enhancement
- **SPDK TCP transport introduction**
- Conclusion



General design and implementation



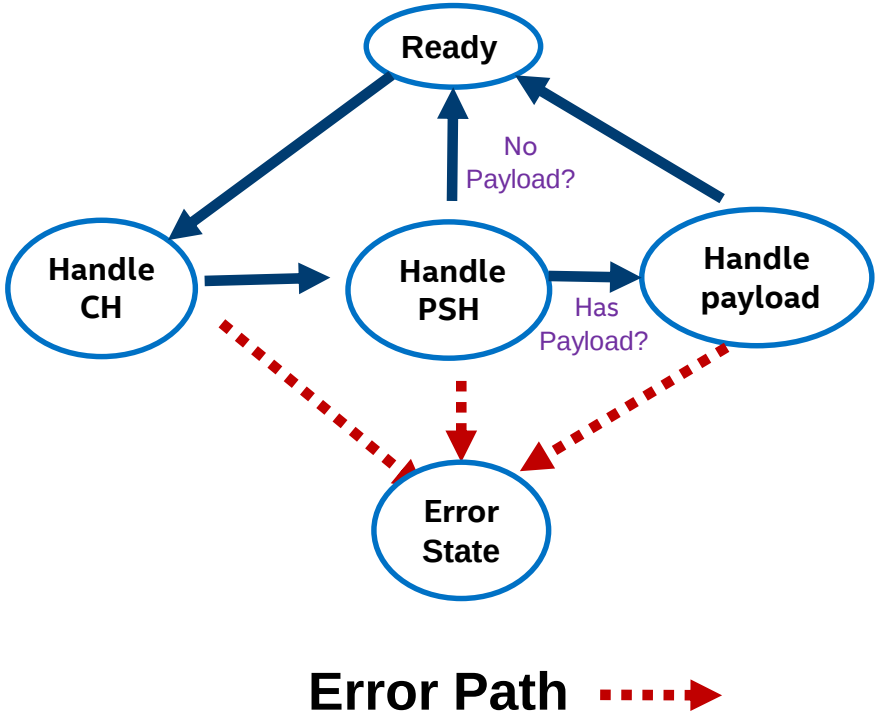
- Follow the SPDK transport abstraction:
  - Host side code: lib/nvme/nvme_tcp.c
  - Target side code: lib/nvmf/tcp.c

Performance design consideration for TCP transport in target side

Ingredients	Methodology
Design framework	Follow the general SPDK NVMe-oF framework (e.g., polling group)
TCP connection optimization	Use the SPDK encapsulated Socket API (preparing for integrating other stack, e.g., VPP)
NVMe/TCP PDU handling	Use state machine to track
NVMe/TCP request life time cycle	Use state machine to track (Purpose: Easy to debug and good for further performance improvement)

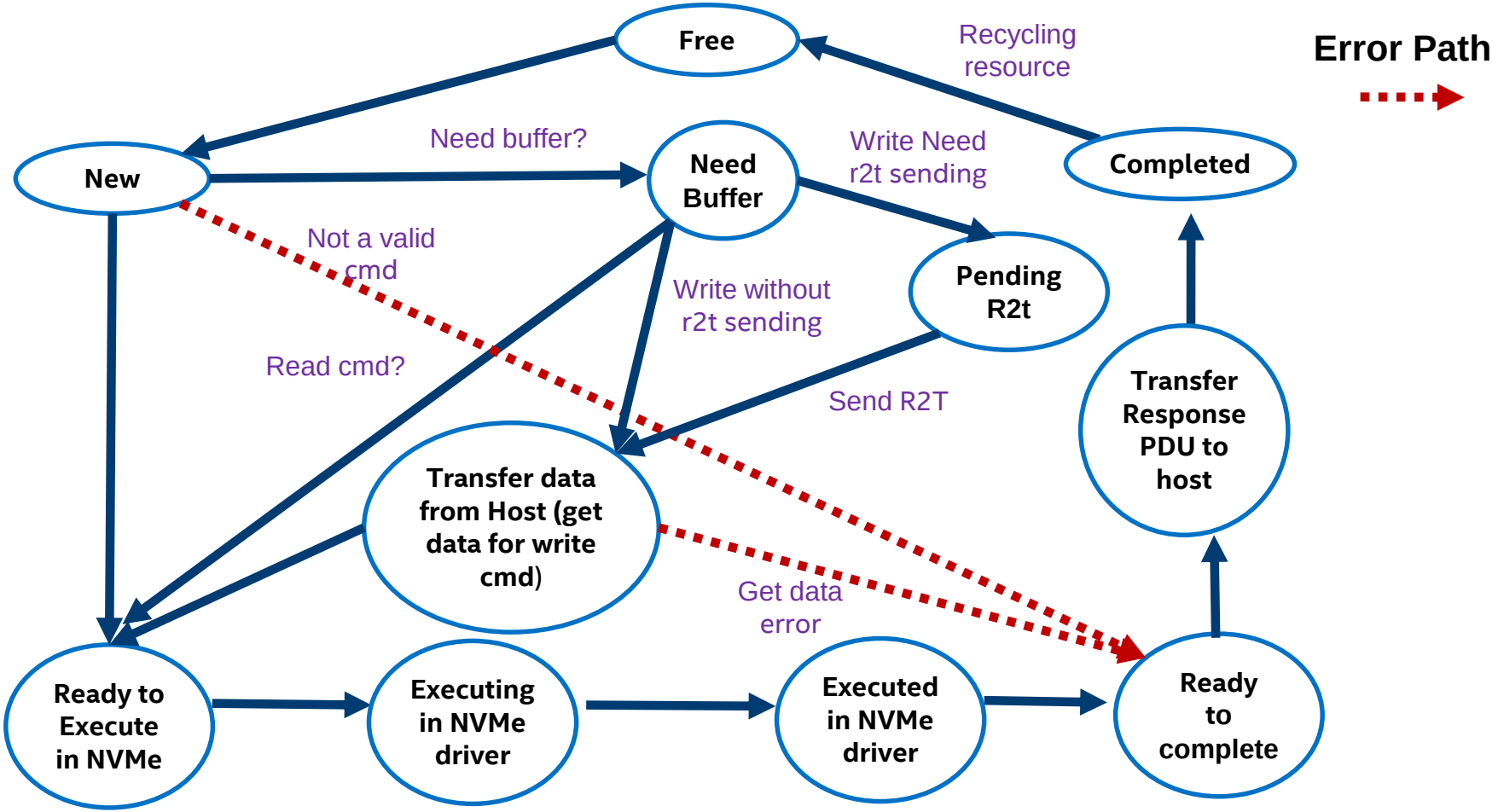
TCP PDU Receiving handling for each connection

```
enum nvme_tcp_pdu_recv_state {
        /* Ready to wait PDU */
        NVME_TCP_PDU_RECV_STATE_AWAIT_PDU_READY,

        /* Active tqpair waiting for any PDU common header */
        NVME_TCP_PDU_RECV_STATE_AWAIT_PDU_CH,

        /* Active tqpair waiting for any PDU specific header */
        NVME_TCP_PDU_RECV_STATE_AWAIT_PDU_PSH,

        /* Active tqpair waiting for payload */
        NVME_TCP_PDU_RECV_STATE_AWAIT_PDU_PAYLOAD,

        /* Active tqpair does not wait for payload */
        NVME_TCP_PDU_RECV_STATE_ERROR,
};
```

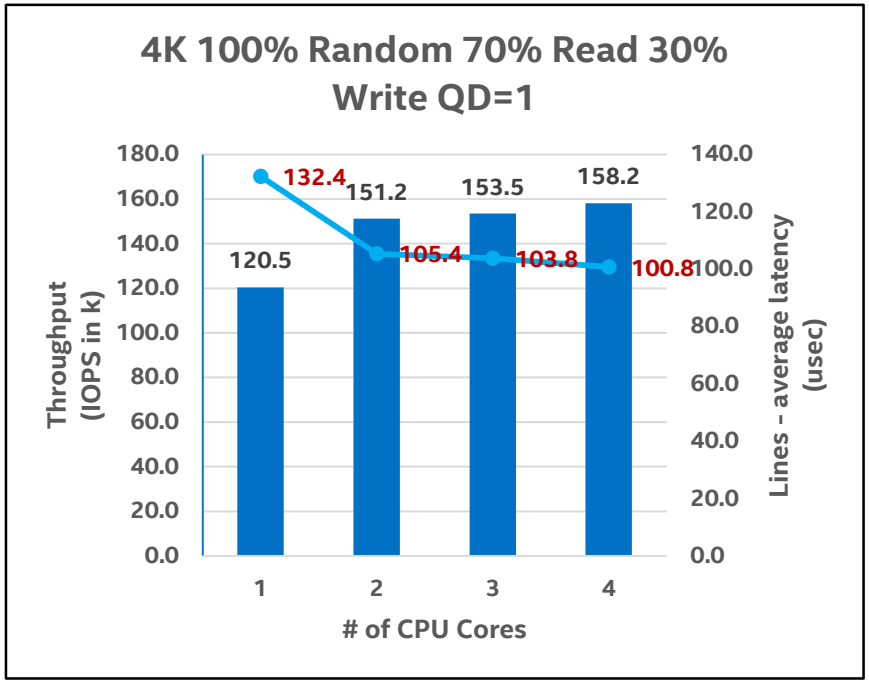


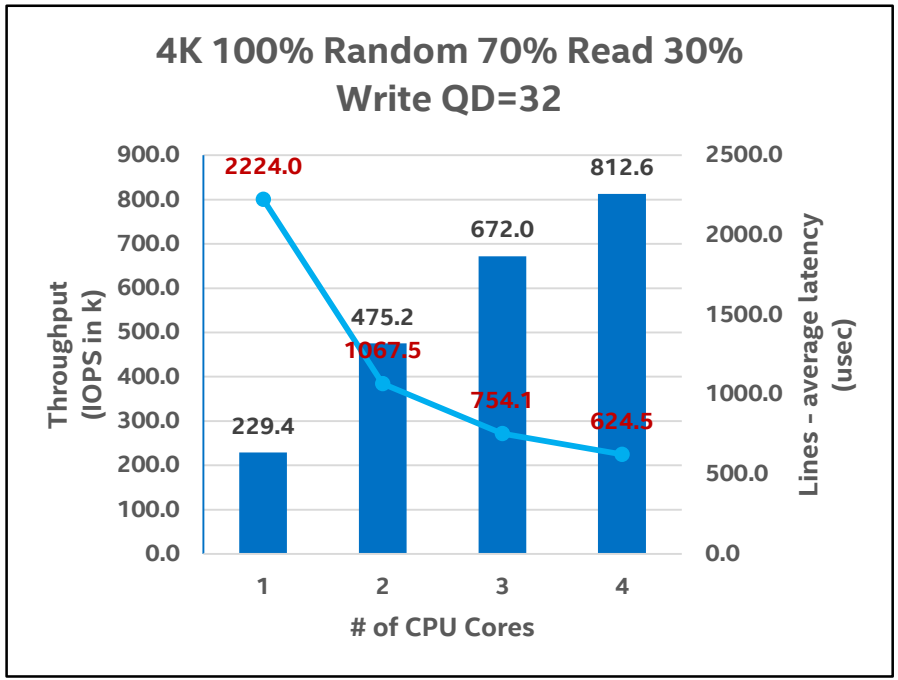
SPDK NVMe-oF TCP request life cycle of each connection in target side

```
enum spdk_nvmf_tcp_req_state {

        /* The request is not currently in use */
        TCP_REQUEST_STATE_FREE = 0,

        /* Initial state when request first received */
        TCP_REQUEST_STATE_NEW,

        /* The request is queued until a data buffer is available. */
        TCP_REQUEST_STATE_NEED_BUFFER,

        /* The request is pending on r2t slots */
        TCP_REQUEST_STATE_DATA_PENDING_FOR_R2T,

        /* The request is currently transferring data from the host to the controller. */
        TCP_REQUEST_STATE_TRANSFERRING_HOST_TO_CONTROLLER,

        /* The request is ready to execute at the block device */
        TCP_REQUEST_STATE_READY_TO_EXECUTE,

        /* The request is currently executing at the block device */
        TCP_REQUEST_STATE_EXECUTING,

        /* The request finished executing at the block device */
        TCP_REQUEST_STATE_EXECUTED,

        /* The request is ready to send a completion */
        TCP_REQUEST_STATE_READY_TO_COMPLETE,

        /* The request is currently transferring final pdus from the controller to the host. */
        TCP_REQUEST_STATE_TRANSFERRING_CONTROLLER_TO_HOST,

        /* The request completed and can be marked free. */
        TCP_REQUEST_STATE_COMPLETED,

        /* Terminator */
        TCP_REQUEST_NUM_STATES,
};
```


Free
Recycling resource
Error Path
New
Need buffer?
Need Buffer
Write Need r2t sending
Completed
Not a valid cmd
Write without r2t sending
Pending R2t
Read cmd?
Send R2T
Transfer Response PDU to host
Transfer data from Host (get data for write cmd)
Get data error
Ready to Execute in NVMe
Executing in NVMe driver
Executed in NVMe driver
Ready to complete

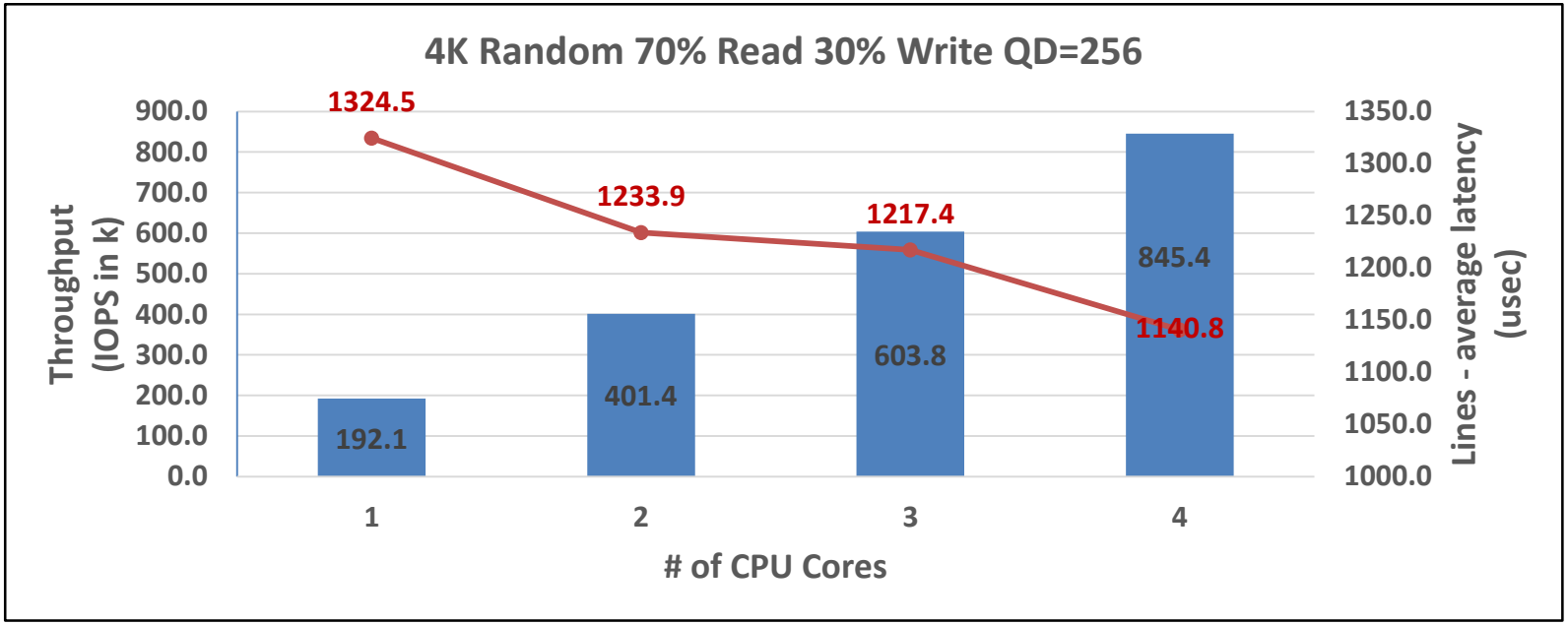
SPDK TARGET SIDE (TCP TRANSPORT): I/O SCALING





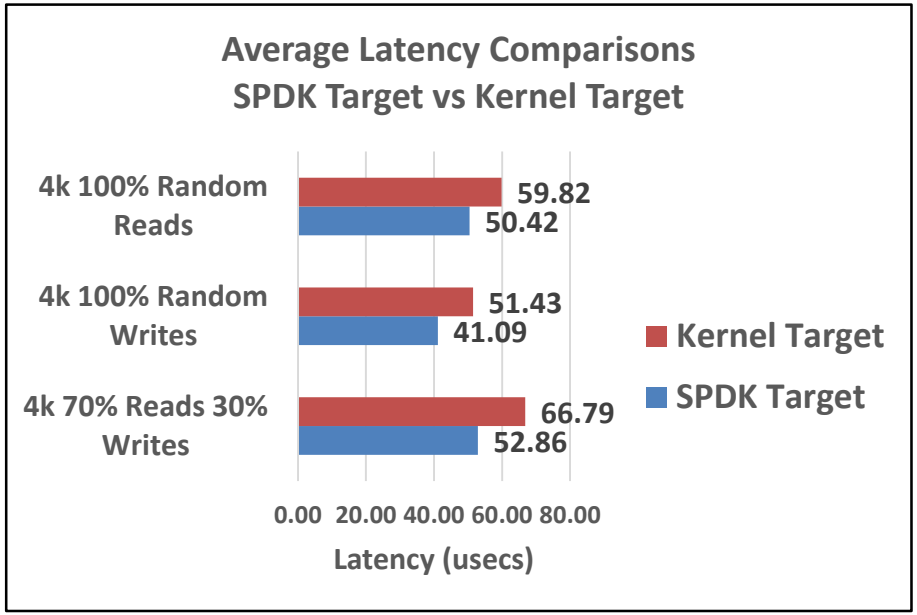
System configuration: (1) Target: server platform: SuperMicro SYS2029U-TN24R4T; 2x Intel® Xeon® Platinum 8180 CPU @ 2.50 GHz, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 4x 2GB DDR4 2666 MT/s, 1 DIMM per channel; 2x 100GbE Mellanox ConnectX-5 NICs; Fedora 28, Linux kernel 5.05, SPDK 19.01.1; 6x Intel® P4600TM P4600x 2.0TB; (2) initiator: Server platform: SuperMicro SYS-2028U TN24R4T+; 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); 1x 100GbE Mellanox ConnectX-4 NIC; Fedora 28, Linux kernel 5.05, SPDK 19.0.1. (3) : Fio ver: fio-3.3; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio or SPDK bdev engine, size=10G, ramp_time=0, run_time=300, group_reporting, thread, direct=1, rw=read/write/rw/randread/randwrite/randrw

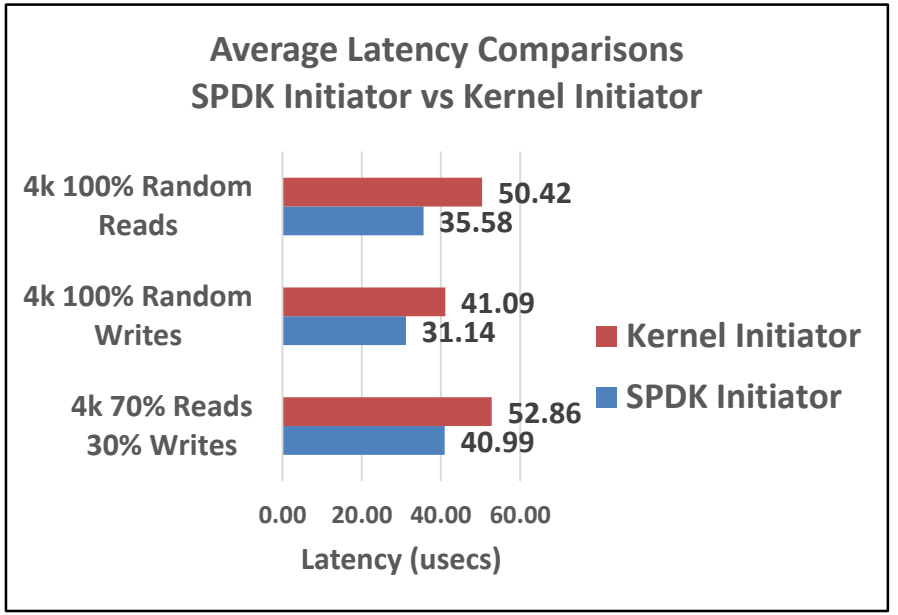
SPDK HOST SIDE (TCP TRANSPORT): I/O SCALING



System configuration: (1) Target: server platform: SuperMicro SYS2029U-TN24R4T; 2x Intel® Xeon® Platinum 8180 CPU @ 2.50 GHz, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 4x 2GB DDR4 2666 MT/s, 1 DIMM per channel; 2x 100GbE Mellanox ConnectX-5 NICs; Fedora 28, Linux kernel 5.05, SPDK 19.01.1; 6x Intel® P4600TM P4600x 2.0TB; (2) initiator: Server platform: SuperMicro SYS-2028U TN24R4T+; 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); 1x 100GbE Mellanox ConnectX-4 NIC; Fedora 28, Linux kernel 5.05, SPDK 19.0.1. (3) : Fio ver: fio-3.3; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio or SPDK bdev engine, size=10G, ramp_time=0, run_time=300, group_reporting, thread, direct=1, rw=read/write/rw/randread/randwrite/randrw

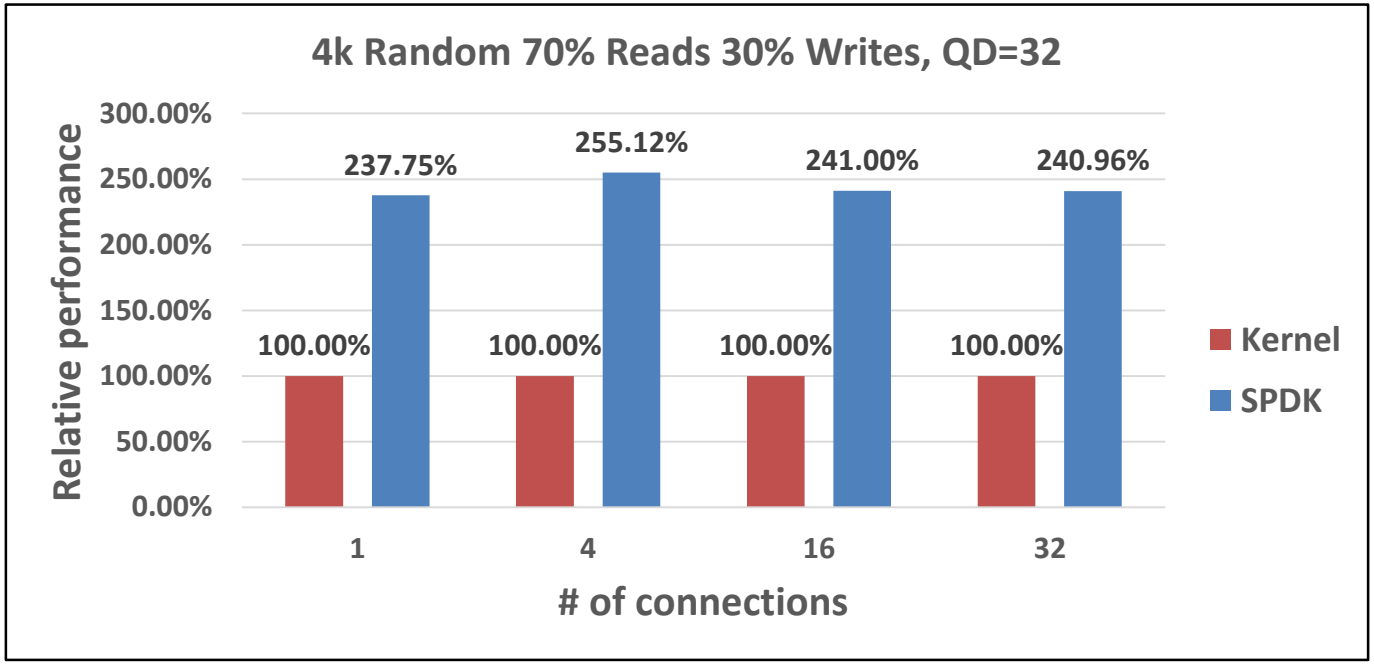
LATENCY COMPARISON BETWEEN SPDK AND KERNEL ()





System configuration: (1) Target: server platform: SuperMicro SYS2029U-TN24R4T; 2x Intel® Xeon® Platinum 8180 CPU @ 2.50 GHz, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 4x 2GB DDR4 2666 MT/s, 1 DIMM per channel; 2x 100GbE Mellanox ConnectX-5 NICs; Fedora 28, Linux kernel 5.05, SPDK 19.01.1; 6x Intel® P4600TM P4600x 2.0TB; (2) initiator: Server platform: SuperMicro SYS-2028U TN24R4T+; 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); 1x 100GbE Mellanox ConnectX-4 NIC; Fedora 28, Linux kernel 5.05, SPDK 19.0.1. (3) : Fio ver: fio-3.3; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio or SPDK bdev engine, size=10G, ramp_time=0, run_time=300, group_reporting, thread, direct=1, rw=read/write/rw/randread/randwrite/randrw

IOPS/CORE COMPARISON BETWEEN SPDK AND KERNEL IN TARGET SIDE



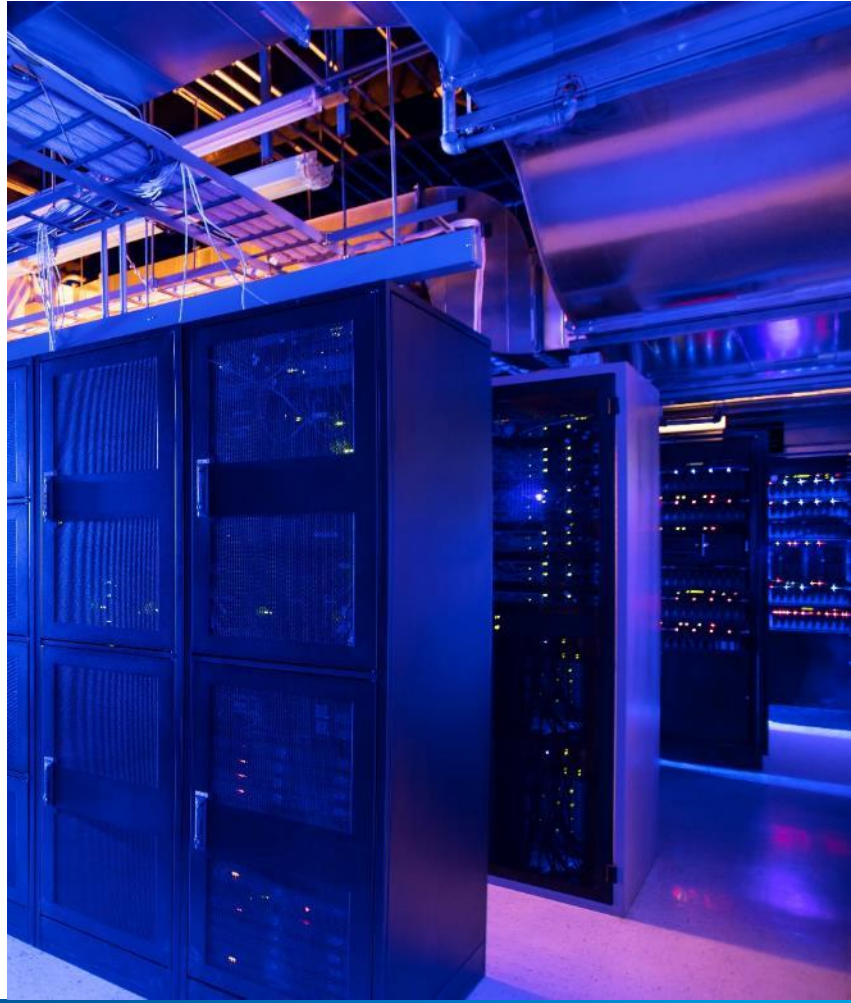
System configuration: (1) Target: server platform: SuperMicro SYS2029U-TN24R4T; 2x Intel® Xeon® Platinum 8180 CPU @ 2.50 GHz, Intel® Speed Step enabled, Intel® Turbo Boost Technology enabled, 4x 2GB DDR4 2666 MT/s, 1 DIMM per channel; 2x 100GbE Mellanox ConnectX-5 NICs; Fedora 28, Linux kernel 5.05, SPDK 19.01.1; 6x Intel® P4600TM P4600x 2.0TB; (2) initiator: Server platform: SuperMicro SYS-2028U TN24R4T+; 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); 1x 100GbE Mellanox ConnectX-4 NIC; Fedora 28, Linux kernel 5.05, SPDK 19.0.1. (3) : Fio ver: fio-3.3; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio or SPDK bdev engine, size=10G, ramp_time=0, run_time=300, group_reporting, thread, direct=1, rw=read/write/rw/randread/randwrite/randrw

Further development plan

- Continue enhancing the function feature
  - Including the compatible test with Linux kernel solution.
- Performance tuning
- Deep integration with user space stack: VPP + DPDK
  - Use the hardware features of NICs for performance improvement (e.g., VMA from Mellanox's NIC, load balance features from Intel's 100Gbit NIC)
  - Figuring out offloading methods with hardware, e.g., FPGA, Smart NIC, and etc.

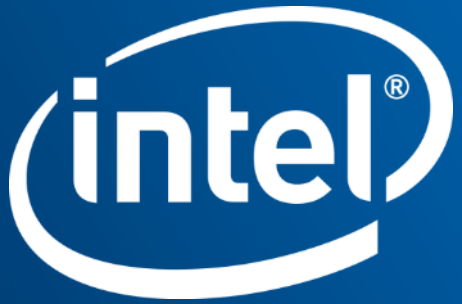
AGENDA

- SPDK NVMe-oF Development history & status
- SPDK RDMA transport enhancement
- SPDK TCP transport introduction
- **Conclusion**



Conclusion

- SPDK NVMe-oF solution is well adopted by the industry. In this presentation, followings are introduced, i.e.,
  - RDMA transport enhancement
  - Newly TCP transport development.
- Further development
  - Continue following the NVMe-oF spec and adding more features.
  - Continue performance enhancements and integration with other solutions.
- Call for activity in community
  - Welcome to bug submission, idea discussion and patch submission for NVMe-oF


intel®

BACKUP

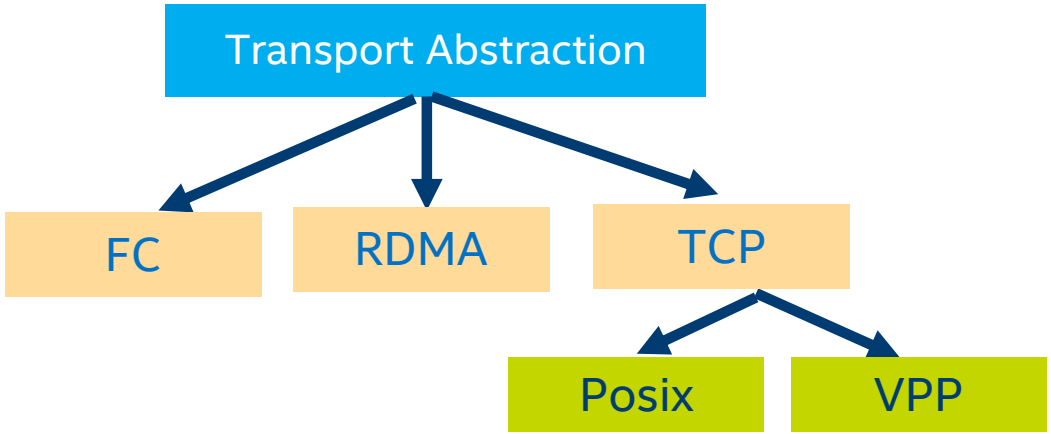
SDPK NVMe-oF Timeline

- NVMe over Fabrics Target
  - July 2016: Released with RDMA transport support
  - July 2016 – Oct 2018:
    - Hardening: e.g., Intel test infrastructure construction, Discovery simplification
      Correctness & kernel interop)
    - Performance improvements: e.g., Read latency improvement, Scalability validation, event framework enhancements, multiple connection performance improvement with group polling
- Jan 2019: Release with TCP transport support
  - Compatibility: Support both kernel & SPDK host on TCP transport
  - Optimization: Support integrating different TCP stack (e.g., Kernel, VPP)

- NVMe over Fabrics Host (Initiator)
  - Dec 2016: Released with RDMA transport support
  - Dec 2016 – Oct 2018:
    - Hardening: Interoperability test with Kernel/SPDK NVMe-oF target
    - Performance improvements: e.g., zero copy, SGL support
  - Jan 2019: Release with TCP transport support
    - Compatibility: Support both kernel & SPDK target on TCP transport

General design and implementation

- It is very easy to add a new transport in SPDK NVMe-oF solution, we need to follow the SPDK transport abstraction:
  - Host side: Implement transport functions defined in lib/nvme/nvme_transport.c and code is in lib/nvme/nvme_tcp.c
  - Target side: Implement the transport related functions defined in lib/nvmf/transport.h and code is in lib/nvmf/tcp.c

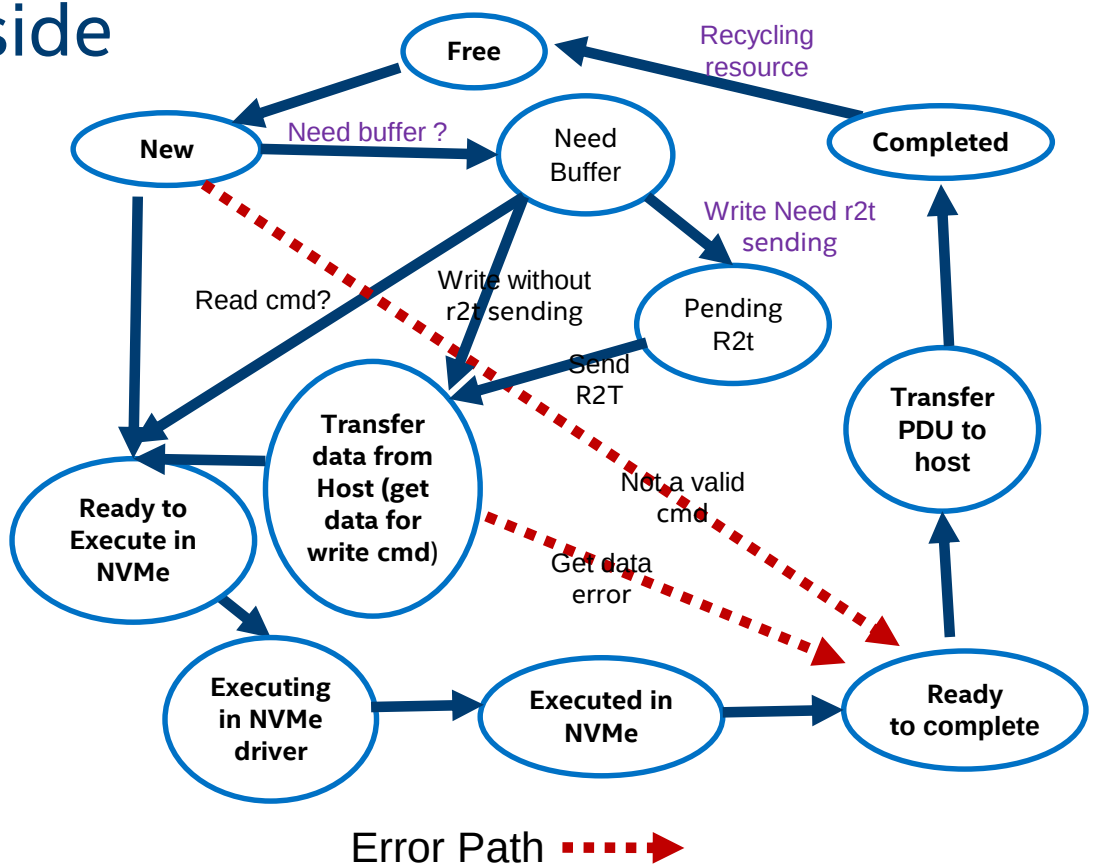


General design and implementation

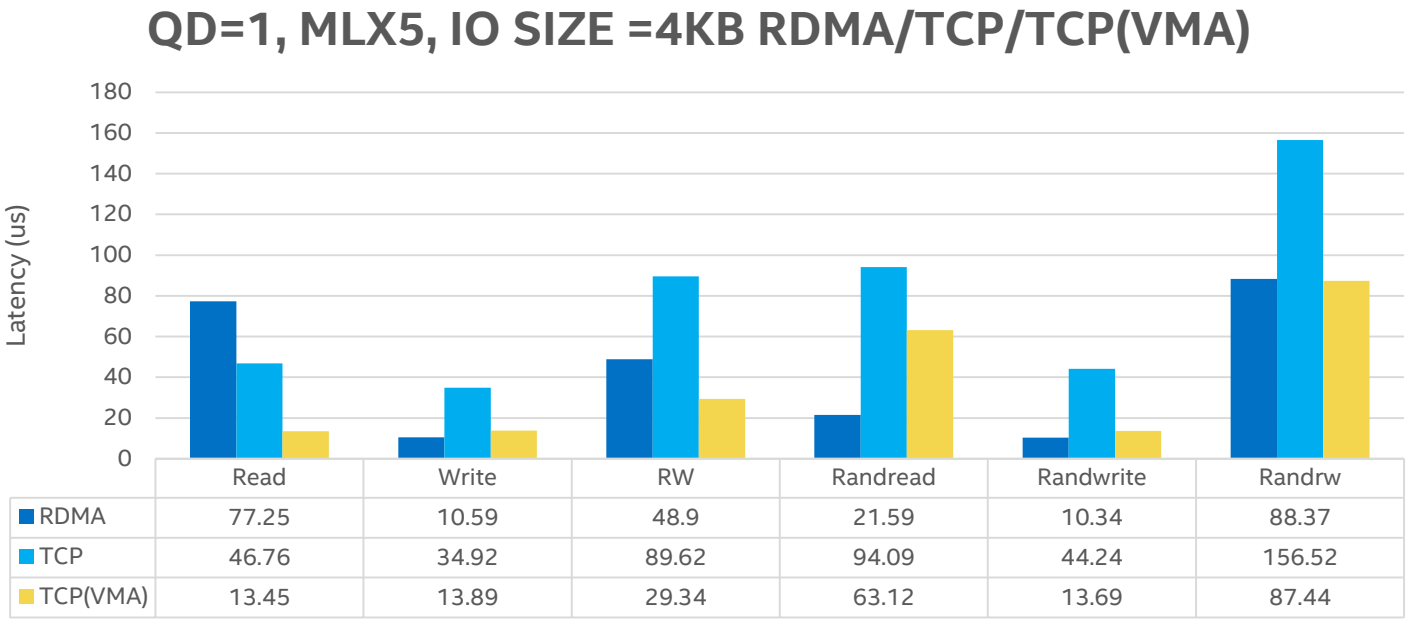
- Performance design consideration for TCP transport in target side
  - Follow the general SPDK NVMe-oF framework, e.g., via independent polling group on each thread.
  - TCP connection optimization (e.g., read/write) for each qpair.
    - Use the SPDK encapsulated Socket API.
    - Purpose: To reserve the interface that users can use different TCP stack (e.g., VPP) for further optimization.
    - Use state machine to track NVMe/TCP Receiving PDU status.
  - NVMe command handling on each qpair.
    - Use state machine to track the Life cycle of NVM request
    - Benefit: Easy for debugging and further performance improvement.

SPDK NVMe-oF TCP request life cycle of each connection in target side

```
enum spdk_nvmf_tcp_req_state {

        /* The request is not currently in use */
        TCP_REQUEST_STATE_FREE = 0,

        /* Initial state when request first received */
        TCP_REQUEST_STATE_NEW,

        /* The request is queued until a data buffer is available. */
        TCP_REQUEST_STATE_NEED_BUFFER,

        /* The request is pending on r2t slots */
        TCP_REQUEST_STATE_DATA_PENDING_FOR_R2T,

        /* The request is currently transferring data from the host to the controller. */
        TCP_REQUEST_STATE_TRANSFERRING_HOST_TO_CONTROLLER,

        /* The request is ready to execute at the block device */
        TCP_REQUEST_STATE_READY_TO_EXECUTE,

        /* The request is currently executing at the block device */
        TCP_REQUEST_STATE_EXECUTING,

        /* The request finished executing at the block device */
        TCP_REQUEST_STATE_EXECUTED,

        /* The request is ready to send a completion */
        TCP_REQUEST_STATE_READY_TO_COMPLETE,

        /* The request is currently transferring final pdus from the controller to the host. */
        TCP_REQUEST_STATE_TRANSFERRING_CONTROLLER_TO_HOST,

        /* The request completed and can be marked free. */
        TCP_REQUEST_STATE_COMPLETED,

        /* Terminator */
        TCP_REQUEST_NUM_STATES,
};
```



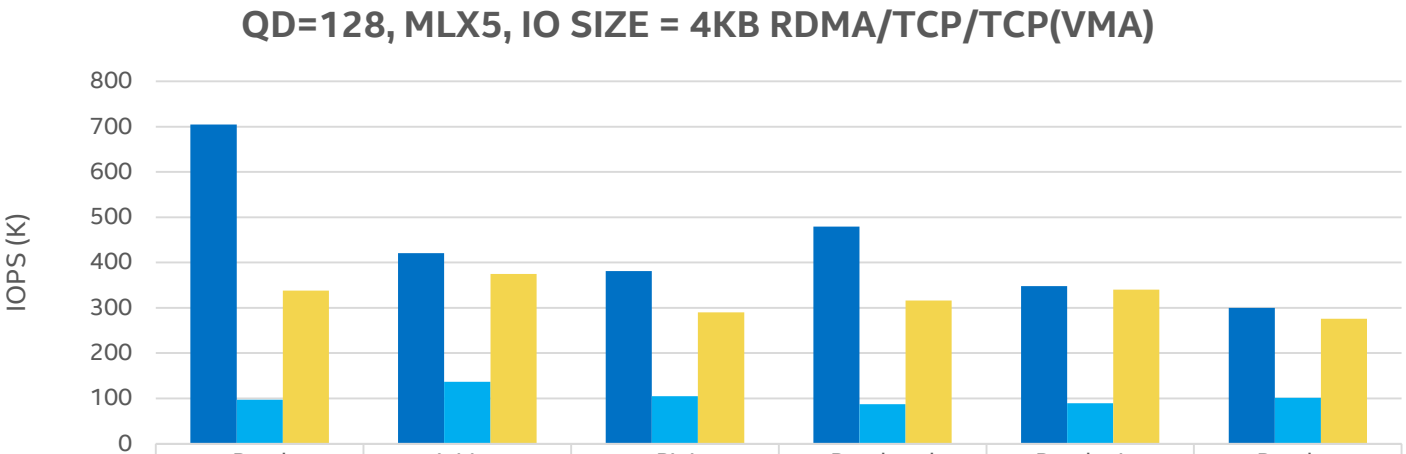
BENCHMARKS - LATENCY (NEED BE UPDATED)



	Read	Write	RW	Randread	Randwrite	Randrw
RDMA	77.25	10.59	48.9	21.59	10.34	88.37
TCP	46.76	34.92	89.62	94.09	44.24	156.52
TCP(VMA)	13.45	13.89	29.34	63.12	13.69	87.44

System configuration: 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); Cores per socket: 22; 16x Samsung 8GB DDR4 @2400 1x Intel SSD DC P3700 Series 375GB @ FW E2010320 SPDK:18.10; Host Dist/Kernel: Fedora 28/Kernel 4.18.16-200; Guest Dist/Kernel: Fio ver: fio-3.6-34; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio, size=10G, ramp_time=0, run_time=300, group_reporting, thread, numjobs=1, direct=1, rw=read/write/rw/randread/randwrite/randrw

BENCHMARK – IOPS (NEED TO BE UPDATED)



	Read	Write	RW	Randread	Randwrite	Randrw
RDMA	705	421	381	479	348	300.2
TCP	97	137	104.9	87	89.7	101.6
TCP(VMA)	338	375	290	316	340	276

System configuration: 44x Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (HT off); Cores per socket: 22; 16x Samsung 8GB DDR4 @2400 1x Intel SSD DC P3700 Series 375GB @ FW E2010320 SPDK:18.10; Host Dist/Kernel: Fedora 28/Kernel 4.18.16-200; Guest Dist/Kernel: Fio ver: fio-3.6-34; Fio workload: blocksize=4k, iodepth=1, iodepth_batch=128, iodepth_low=256, ioengine=libaio, size=10G, ramp_time=0, run_time=300, group_reporting, thread, numjobs=1, direct=1, rw=randread/randwrite/randrw(50% read, 50% write)


intel