



SPDK NVMe-oF TCP (Target & Initiator)

Performance Report

Release 20.01

Testing Date: March 2020

Performed by: Karol Latecki (karol.latecki@intel.com)

Maciej Wawryk (maciejx.wawryk@intel.com)

Acknowledgments:

John Kariuki (john.k.kariuki@intel.com)

James Harris (james.r.harris@intel.com)



Contents

Contents	2
Audience and Purpose	4
Test setup	5
Target Configuration	5
Initiator 1 Configuration	6
Initiator 2 Configuration	6
BIOS settings	6
TCP configuration	7
Kernel & BIOS spectre-meltdown information	7
Introduction to SPDK NVMe-oF (Target & Initiator)	8
Test Case 1: SPDK NVMe-oF TCP Target I/O core scaling	10
4KB Random Read Results	14
4KB Random Write Results	15
4KB Random Read-Write Results	16
Jumbo Frames Performance Impact	17
4KB Random Read with JF results	17
4KB Random Write with JF results	17
4KB Random Read-Write with JF results	18
Large Sequential I/O Performance	19
Conclusions	23
Test Case 2: SPDK NVMe-oF TCP Initiator I/O core scaling	24
4KB Random Read Results	27
4KB Random Write Results	28
4KB Random Read-Write Results	29
Conclusions	30
Test Case 3: Linux Kernel vs. SPDK NVMe-oF TCP Latency	31
SPDK vs Kernel NVMe-oF Target Latency Results	34
SPDK vs Kernel NVMe-oF TCP Initiator results	35
SPDK vs Kernel NVMe-oF Latency Results	36
Conclusions	37
Test Case 4: NVMe-oF Performance with increasing # of connections	38
4KB Random Read Results	40
4KB Random Write Results	41
4KB Random Read-Write Results	42
Low Connections Results	43
Conclusions	44
Summary	45
Appendix A	46





Audience and Purpose

This report is intended for people who are interested in evaluating SPDK NVMe-oF (Target & Initiator) performance as compared to the Linux Kernel NVMe-oF (Target & Initiator). This report compares the performance and efficiency of the SPDK NVMe-oF Target and Initiator vs. the Linux Kernel NVMe-oF Target and Initiator. This report covers the TCP transport only.

The purpose of reporting these tests is not to imply a single “correct” approach, but rather to provide a baseline of well-tested configurations and procedures that produce repeatable results. This report can also be viewed as information regarding best known method/practice when performance testing SPDK NVMe-oF (Target & Initiator).

Test setup

Target Configuration

Item	Description
Server Platform	SuperMicro SYS-2029U-TN24R4T  
CPU	Intel® Xeon® Gold 6230 Processor (27.5MB L3, 2.10 GHz) Number of cores 20, number of threads 40
Memory	10 x 32GB Hynix HMA84GR7AFR4N-VK, DDR4, 2666MHz Total of 320GB
Operating System	Fedora 30
BIOS	3.1a
Linux kernel version	5.4.14-100.fc30
SPDK version	SPDK 20.01
Storage	OS: 1x 120GB Intel SSDSC2BB120G4 Storage Target: 16x Intel® SSD DC P4610™ 1.6TB (FW: QDV10190) (8 on each CPU socket)
NIC	2x 100GbE Mellanox ConnectX-5 NICs. Both ports connected. 1 NIC per CPU socket.



Initiator 1 Configuration

Item	Description
Server Platform	Intel® Server System R2208WFTZSR
CPU	Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz (35.75MB Cache) Number of cores 24, number of threads 48 per socket (Both sockets populated)
Memory	6 x 32GB Micron M393A1G40EB1-CRC, DDR4, 2933MHz Total 192GBs
Operating System	Fedora 30
BIOS	02.01.0008 03/19/2019
Linux kernel version	5.4.14-100.fc30
SPDK version	SPDK 20.01
Storage	OS: 1x 240GB INTEL SSDSC2BB240G6
NIC	1x 100GbE Mellanox ConnectX-5 Ex NIC. Both ports connected to Target server. (connected to CPU socket 0)

Initiator 2 Configuration

Item	Description
Server Platform	Intel® Server System R2208WFTZSR
CPU	Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz (35.75MB Cache) Number of cores 24, number of threads 48 per socket (Both sockets populated)
Memory	6 x 32GB Micron M393A1G40EB1-CRC, DDR4, 2933MHz Total 192GBs
Operating System	02.01.0008 03/19/2019
BIOS	3.1 06/08/2018
Linux kernel version	5.4.14-100.fc30
SPDK version	SPDK 20.01
Storage	OS: 1x 240GB INTEL SSDSC2BB240G6
NIC	1x 100GbE Mellanox ConnectX-5 Ex NIC. Both ports connected to Target server. (connected to CPU socket 0)

BIOS settings

Item	Description
BIOS <i>(Applied to all 3 systems)</i>	Hyper threading Enabled CPU Power and Performance Policy: <ul style="list-style-type: none">• “Extreme Performance” for Target• “Performance” for Initiators CPU C-state No Limit CPU P-state Enabled Enhanced Intel® SpeedStep® Tech Enabled Turbo Boost Enabled



TCP configuration

Note that the SPDK NVMe-oF target and initiator use the Linux Kernel TCP stack. We tuned the Linux Kernel TCP stack for storage workloads over 100 Gbps NIC by settings the following parameters using sysctl:

```
# Set 256MB buffers
net.core.rmem_max = 268435456
net.core.wmem_max = 268435456
# Increase autotuning TCP buffer limits
# min, max and default settings
# auto-tuning allowed to 128MB
net.ipv4.tcp_rmem = 4096 87380 134217728
net.ipv4.tcp_wmem = 4096 65536 134217728
# MTU probing for Jumbo Frames
net.ipv4.tcp_mtu_probing = 1
```

The NIC ports were configured to use Jumbo Frames using network-scripts (/etc/sysconfig/network-scripts for RHEL-based distributions) and setting MTU=9000 for some of the test cases for comparison with standard MTU=1500.

Kernel & BIOS spectre-meltdown information

All three server systems use Fedora 5.4.14-100.fc30 kernel version available from DNF repository with default patches for spectre-meltdown issue enabled.

BIOS on all systems was updated to post spectre-meltdown versions as well.

Introduction to SPDK NVMe-oF (Target & Initiator)

The NVMe over Fabrics (NVMe-oF) protocol extends the parallelism and efficiencies of the NVMe Express* (NVMe) block protocol over network fabrics such as RDMA (iWARP, RoCE), InfiniBand™, Fibre Channel, TCP and Intel® Omni-Path. SPDK provides both a user space NVMe-oF target and initiator that extends the software efficiencies of the rest of the SPDK stack over the network. The SPDK NVMe-oF target uses the SPDK user-space, polled-mode NVMe driver to submit and complete I/O requests to NVMe devices which reduces the software processing overhead. Likewise, it pins connections to CPU cores to avoid synchronization and cache thrashing so that the data for those connections is kept close to the CPU.

The SPDK NVMe-oF target and initiator uses the underlying transport layer API which in case of TCP are POSIX sockets. In case of RDMA-capable NICs Infiniband/RDMA verbs API is used which should work on all flavors of RDMA transports, but is currently tested against RoCEv2, iWARP, and Omni-Path NICs. Similar to the SPDK NVMe driver, SPDK provides a user-space, lockless, polled-mode NVMe-oF initiator. The host system uses the initiator to establish a connection and submit I/O requests to an NVMe subsystem within an NVMe-oF target. NVMe subsystems contain namespaces, each of which maps to a single block device exposed via SPDK's bdev layer. SPDK's bdev layer is a block device abstraction layer and general purpose block storage stack akin to what is found in many operating systems. Using the bdev interface completely decouples the storage media from the front-end protocol used to access storage. Users can build their own virtual bdevs that provide complex storage services and integrate them with the SPDK NVMe-oF target with no additional code changes. There can be many subsystems within an NVMe-oF target and each subsystem may hold many namespaces. Subsystems and namespaces can be configured dynamically via a JSON-RPC interface.

Figure 1 shows a high level schematic of the systems used for testing in the rest of this report. The set up consists of three systems (two used as initiators and one used as the target). The NVMe-oF target is connected to both initiator systems point-to-point using QSFP28 cables without any switches. The target system has sixteen Intel® SSD DC P4600 SSDs which were used as block devices for NVMe-oF subsystems and two 100GbE Mellanox ConnectX®-5 NICs connected to provide up to 200GbE of network bandwidth. Each Initiator system has one Mellanox ConnectX®-5 100GbE NIC connected directly to the target without any switch.

One goal of this report was to make clear the advantages and disadvantages inherent to the design of the SPDK NVMe-oF components. These components are written using techniques such as run-to completion, polling, and asynchronous I/O. The report covers four real-world use cases.

For performance benchmarking the fio tool is used with two storage engines:

- 1) Linux Kernel libaio engine
- 2) SPDK bdev engine

Performance numbers reported are aggregate I/O per second, average latency, and CPU utilization as a percentage for various scenarios. Aggregate I/O per second and average latency data is reported from fio and CPU utilization was collected using sar (systat).

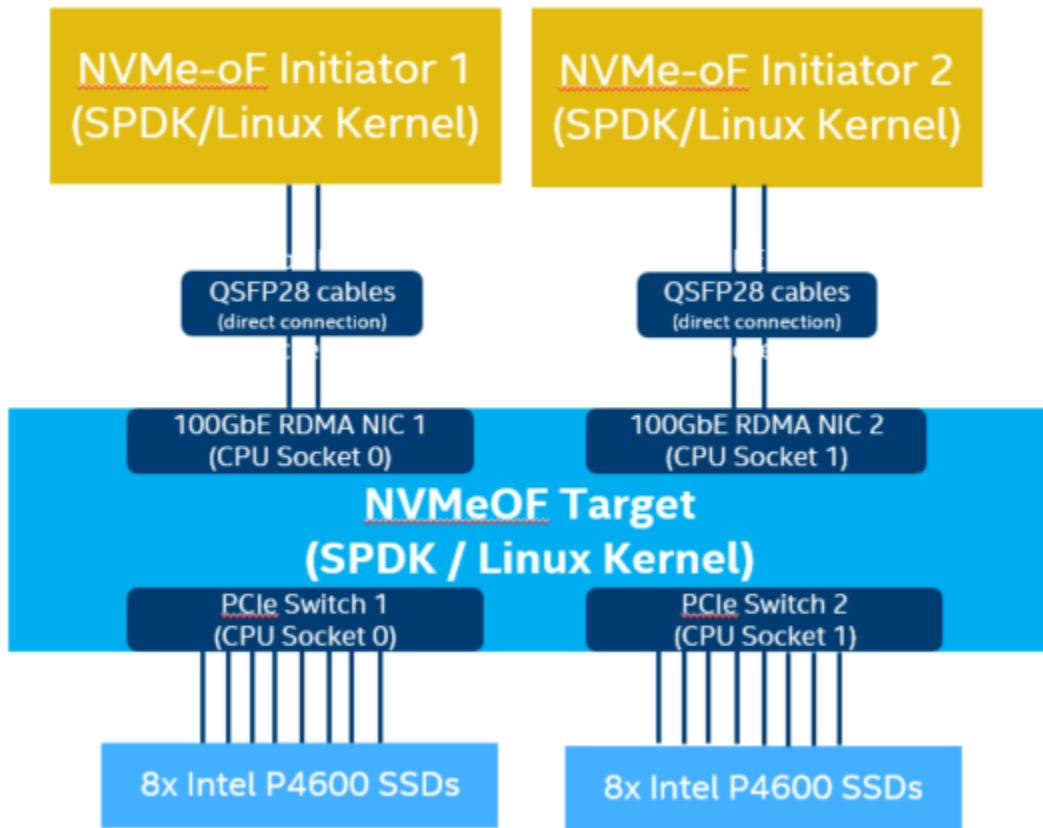


Figure 1: High-Level NVMe-oF TCP performance testing setup

Test Case 1: SPDK NVMe-oF TCP Target I/O core scaling

This test case was performed in order to understand the performance of SPDK TCP NVMe-oF target with I/O core scaling.

The SPDK NVMe-oF TCP target was configured to run with 16 NVMe-oF subsystems. Each NVMe-oF subsystem ran on top of an individual NVMe bdev backed by a single Intel P4600 device. Each of the 2 host systems was connected to 8 NVMe-oF subsystems which were exported by the SPDK NVMe-oF Target over 1x 100GbE NIC. The SPDK bdev FIO plugin was used to target 8 NVMe-oF bdevs on each of the host. The SPDK Target was configured to use 1, 4, 8, 12, 16, 20, 24, 32, 36 and 40 CPU cores. We ran the following workloads on each initiator:

- 4KB 100% Random Read
- 4KB 100% Random Write
- 4KB Random 70% Read 30% Write

We scaled the fio jobs using fio configuraton parameter numjob=2, in order to generate more I/O requests.

For detailed configuration please refer to the table below. The actual SPDK NVMe-oF configuration was done using JSON-RPC and the table contains the sequence of commands used by spdk/scripts/rpc.py script rather than a configuration file. The SPDK NVMe-oF Initiator (bdev fio_plugin) still uses plain configuration files.

Each workload was run three times at each CPU count and the reported results are the average of the 3 runs. For workloads which need preconditioning (4KB rand write and 4KB 70% read 30% write we ran preconditioning once before running all of the workload to ensure that NVMe devices reached higher IOPS so that we can saturate the network .

Item	Description
Test Case	Test SPDK NVMe-oF Target I/O core scaling
SPDK NVMe-oF Target configuration	All of the commands below were executed with spdk/scripts/rpc.py script. construct_nvme_bdev -t PCIe -b Nvme0 -a 0000:60:00.0 construct_nvme_bdev -t PCIe -b Nvme1 -a 0000:61:00.0 construct_nvme_bdev -t PCIe -b Nvme2 -a 0000:62:00.0 construct_nvme_bdev -t PCIe -b Nvme3 -a 0000:63:00.0 construct_nvme_bdev -t PCIe -b Nvme4 -a 0000:64:00.0 construct_nvme_bdev -t PCIe -b Nvme5 -a 0000:65:00.0 construct_nvme_bdev -t PCIe -b Nvme6 -a 0000:66:00.0 construct_nvme_bdev -t PCIe -b Nvme7 -a 0000:67:00.0 construct_nvme_bdev -t PCIe -b Nvme8 -a 0000:b5:00.0 construct_nvme_bdev -t PCIe -b Nvme9 -a 0000:b6:00.0 construct_nvme_bdev -t PCIe -b Nvme10 -a 0000:b7:00.0



```

construct_nvme_bdev -t PCIe -b Nvme11 -a 0000:b8:00.0
construct_nvme_bdev -t PCIe -b Nvme12 -a 0000:b9:00.0
construct_nvme_bdev -t PCIe -b Nvme13 -a 0000:ba:00.0
construct_nvme_bdev -t PCIe -b Nvme14 -a 0000:bb:00.0
construct_nvme_bdev -t PCIe -b Nvme15 -a 0000:bc:00.0

nvmf_create_transport -t TCP
(creates TCP transport layer with default values:
trtype: "TCP"
max_queue_depth: 128
max_qpairs_per_ctrlr: 64
in_capsule_data_size: 4096
max_io_size: 131072
"io_unit_size": 131072,
max_aq_depth: 128
num_shared_buffers: 4096
buf_cache_size: 32,
"c2h_success": true,
"dif_insert_or_strip": false,
"sock_priority": 0
)

nvmf_subsystem_create nqn.2018-09.io.spdk:cnode1 -s SPDK001 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode2 -s SPDK002 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode3 -s SPDK003 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode4 -s SPDK004 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode5 -s SPDK005 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode6 -s SPDK006 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode7 -s SPDK007 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode8 -s SPDK008 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode9 -s SPDK009 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode10 -s SPDK0010 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode11 -s SPDK0011 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode12 -s SPDK0012 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode13 -s SPDK0013 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode14 -s SPDK0014 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode15 -s SPDK0015 -a -m 8
nvmf_subsystem_create nqn.2018-09.io.spdk:cnode16 -s SPDK0016 -a -m 8

nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode1 Nvme0n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode2 Nvme1n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode3 Nvme2n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode4 Nvme3n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode5 Nvme4n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode6 Nvme5n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode7 Nvme6n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode8 Nvme7n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode9 Nvme8n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode10 Nvme9n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode11 Nvme10n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode12 Nvme11n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode13 Nvme12n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode14 Nvme13n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode15 Nvme14n1
nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode16 Nvme15n1

```

	<pre> nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode1 -t tcp -f ipv4 -s 4420 -a 20.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode2 -t tcp -f ipv4 -s 4420 -a 20.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode3 -t tcp -f ipv4 -s 4420 -a 20.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode4 -t tcp -f ipv4 -s 4420 -a 20.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode5 -t tcp -f ipv4 -s 4420 -a 20.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode6 -t tcp -f ipv4 -s 4420 -a 20.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode7 -t tcp -f ipv4 -s 4420 -a 20.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode8 -t tcp -f ipv4 -s 4420 -a 20.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode9 -t tcp -f ipv4 -s 4420 -a 10.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode10 -t tcp -f ipv4 -s 4420 -a 10.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode11 -t tcp -f ipv4 -s 4420 -a 10.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode12 -t tcp -f ipv4 -s 4420 -a 10.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode13 -t tcp -f ipv4 -s 4420 -a 10.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode14 -t tcp -f ipv4 -s 4420 -a 10.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode15 -t tcp -f ipv4 -s 4420 -a 10.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode16 -t tcp -f ipv4 -s 4420 -a 10.0.1.1 </pre>
SPDK NVMe-oF Initiator - FIO plugin configuration	<p>BDEV.conf</p> <pre> [Nvme] TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode1" Nvme0 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode2" Nvme1 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode3" Nvme2 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode4" Nvme3 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode5" Nvme4 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode6" Nvme5 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode7" Nvme6 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode8" Nvme7 </pre> <p>FIO.conf</p> <pre> [global] ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin spdk_conf=/tmp/spdk/bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth={1, 8, 16, 32, 64} time_based=1 numjobs=3 ramp_time=60 runtime=300 [filename0] filename=Nvme0n1 [filename1] filename=Nvme1n1 [filename2] filename=Nvme2n1 [filename3] filename=Nvme3n1 [filename4] </pre>



	filename=Nvme4n1 [filename5] filename=Nvme5n1 [filename6] filename=Nvme6n1 [filename7] filename=Nvme7n1
--	---

4KB Random Read Results

Test Result: 4KB 100% Random Read IOPS, QD=64

# of Cores	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	1786.93	457.4	6717.5
4 cores	6467.48	1655.7	1851.9
8 cores	13790.43	3530.3	867.5
12 cores	17310.46	4431.5	690.2
16 cores	19246.07	4927.0	620.0
20 cores	19840.68	5079.2	600.9
24 cores	19793.53	5067.1	602.4
28 cores	19736.83	5052.6	604.5
32 cores	19453.05	4980.0	613.6
36 cores	19832.67	5077.2	601.5
40 cores	19930.54	5102.2	597.8

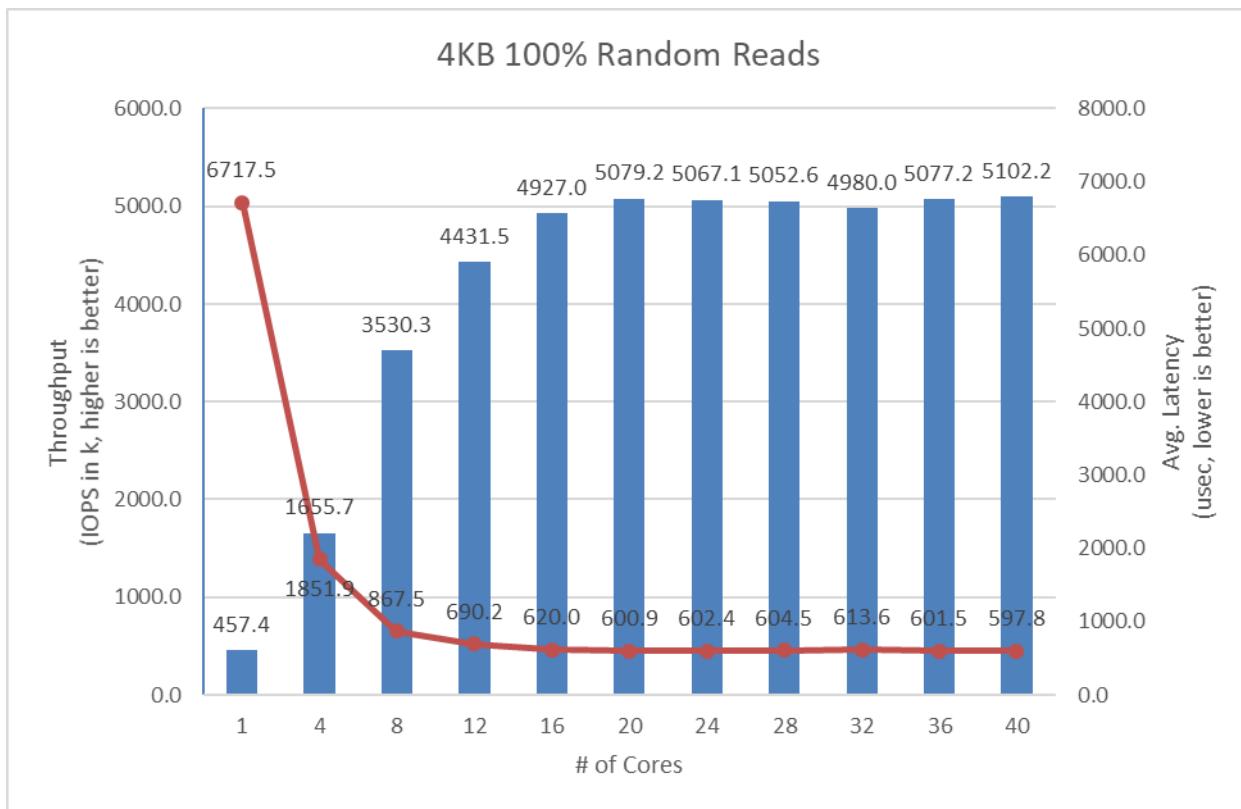


Figure 2: SPDK NVMe-of TCP Target I/O core scaling: IOPS vs. Latency while running 4KB 100% Random Read workload at QD = 64

4KB Random Write Results

Disks were not preconditioned for this test case, which allows for higher IOPS numbers.

Test Result: 4KB 100% Random Writes IOPS, QD=64

# of Cores	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	1077.8	275.9	11182.0
4 cores	4411.8	1129.4	2718.0
8 cores	8374.6	2143.9	1403.9
12 cores	11160.5	2857.1	1053.8
16 cores	13097.6	3353.0	888.6
20 cores	14630.2	3745.3	795.8
24 cores	14936.3	3823.7	780.3
28 cores	15283.9	3912.7	768.2
32 cores	15727.2	4026.1	746.4
36 cores	15727.2	4026.1	746.4
40 cores	16347.9	4185.0	719.1

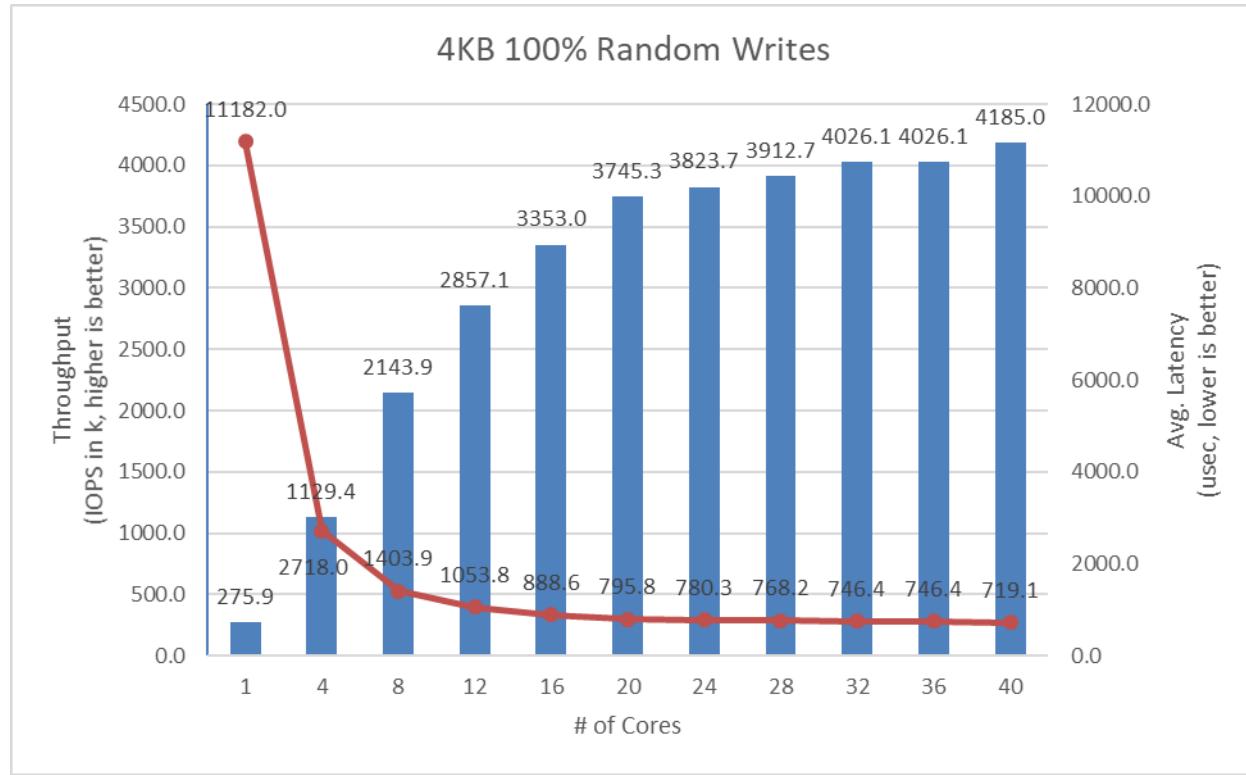


Figure 3: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 4KB 100% Random Write Workload at QD=64

4KB Random Read-Write Results

Test Result: 4KB Random Read/Write 70%/30% IOPS, QD=64

# of Cores	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	1392.7	356.5	8612.4
4 cores	5091.1	1303.3	2353.3
8 cores	10836.6	2774.2	1104.3
12 cores	14411.7	3689.4	827.3
16 cores	16953.5	4340.1	703.2
20 cores	18492.9	4734.2	644.0
24 cores	19527.6	4999.1	609.5
28 cores	19350.5	4953.7	616.3
32 cores	19310.5	4943.5	619.9
36 cores	19281.8	4936.1	618.6
40 cores	20285.5	5193.1	586.9

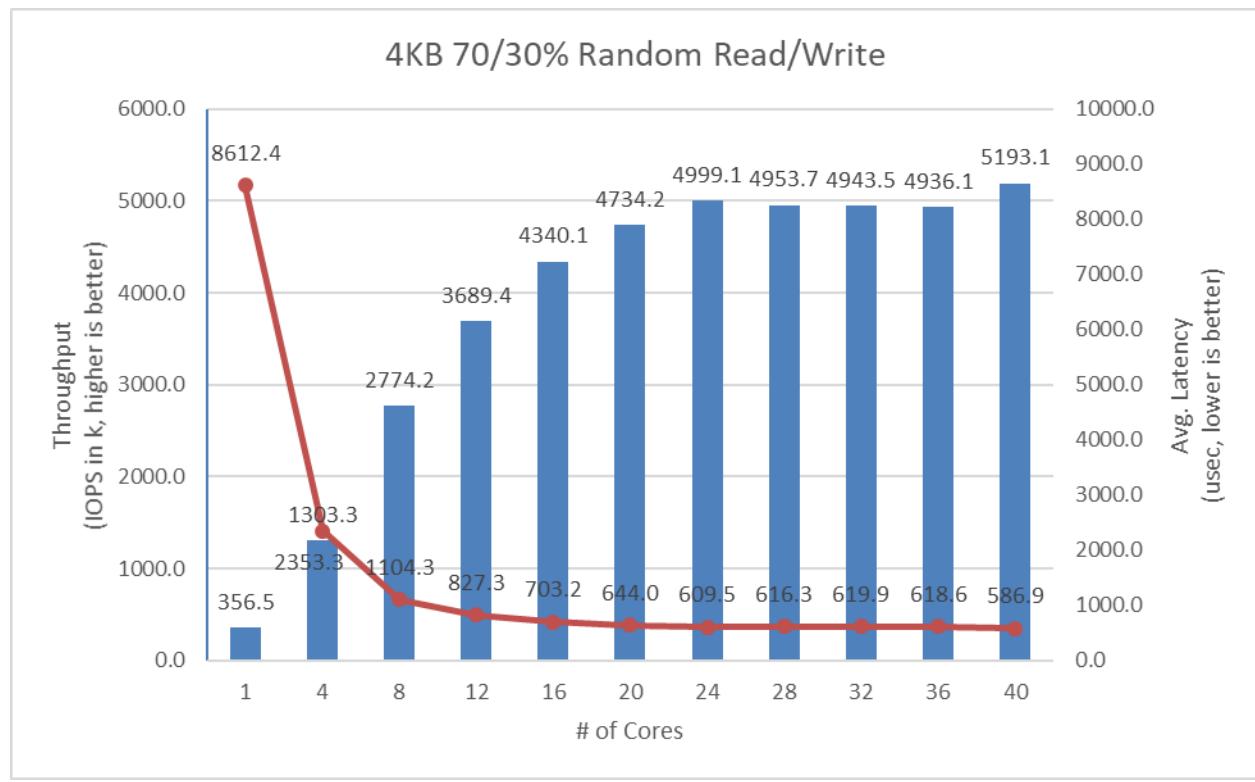


Figure 4: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 4KB Random 70/30 Read/Write workload at QD=64



Jumbo Frames Performance Impact

Selected test cases were re-run using 9000 Bytes MTU configured for RNIC interfaces to measure performance impact of using Jumbo Frames in the test setup.

4KB Random Read with JF results

# of Cores	Bandwidth (MBps)			Throughput (IOPS k)			Avg. Latency (usec)		
	1500MTU	9000MTU	Change [%]	1500MTU	9000MTU	Change [%]	1500MTU	9000MTU	Change [%]
1 core	1786.93	1816.65	1.66%	457.44	465.05	1.66%	6717.46	6610.08	-1.60%
4 cores	6467.48	6571.01	1.60%	1655.66	1682.17	1.60%	1851.91	1824.23	-1.49%
8 cores	13790.43	13120.96	-4.85%	3530.34	3358.95	-4.85%	867.54	912.20	5.15%
12 cores	17310.46	17300.42	-0.06%	4431.47	4428.90	-0.06%	690.20	690.29	0.01%
16 cores	19246.07	18890.84	-1.85%	4926.98	4836.04	-1.85%	620.00	631.49	1.85%
20 cores	19840.68	18957.95	-4.45%	5079.20	4853.22	-4.45%	600.92	629.52	4.76%
24 cores	19793.53	19675.24	-0.60%	5067.13	5036.85	-0.60%	602.44	606.56	0.68%
28 cores	19736.83	19579.94	-0.79%	5052.62	5012.45	-0.79%	604.49	609.32	0.80%
32 cores	19453.05	19695.63	1.25%	4979.97	5042.07	1.25%	613.64	606.04	-1.24%
36 cores	19832.67	19432.34	-2.02%	5077.15	4974.67	-2.02%	601.52	615.08	2.25%
40 cores	19930.54	19211.35	-3.61%	5102.21	4918.09	-3.61%	597.84	622.28	4.09%

4KB Random Write with JF results

# of Cores	Bandwidth (MBps)			Throughput (IOPS k)			Avg. Latency (usec)		
	1500MTU	9000MTU	Change [%]	1500MTU	9000MTU	Change [%]	1500MTU	9000MTU	Change [%]
1 core	1077.77	950.76	-11.78%	275.90	243.38	-11.79%	11182.00	12681.36	13.41%
4 cores	4411.80	3723.04	-15.61%	1129.41	953.09	-15.61%	2717.96	3214.37	18.26%
8 cores	8374.61	7110.05	-15.10%	2143.89	1820.16	-15.10%	1403.90	1658.05	18.10%
12 cores	11160.55	10003.98	-10.36%	2857.09	2561.01	-10.36%	1053.81	1170.25	11.05%
16 cores	13097.58	11593.60	-11.48%	3352.97	2967.95	-11.48%	888.56	1008.93	13.55%
20 cores	14630.23	12488.29	-14.64%	3745.33	3196.99	-14.64%	795.80	935.71	17.58%
24 cores	14936.33	13050.26	-12.63%	3823.69	3340.85	-12.63%	780.34	897.31	14.99%
28 cores	15283.91	13874.18	-9.22%	3912.67	3551.78	-9.22%	768.15	845.69	10.09%
32 cores	15727.19	14306.58	-9.03%	4026.15	3662.47	-9.03%	746.45	821.62	10.07%
36 cores	15727.19	14306.58	-9.03%	4026.15	3662.47	-9.03%	746.45	821.62	10.07%
40 cores	16347.86	13952.67	-14.65%	4185.04	3571.87	-14.65%	719.12	844.32	17.41%



4KB Random Read-Write with JF results

# of Cores	Bandwidth (MBps)			Throughput (IOPS k)			Avg. Latency (usec)		
	1500MTU	9000MTU	Change [%]	1500MTU	9000MTU	Change [%]	1500MTU	9000MTU	Change [%]
1 core	1392.74	1310.06	-5.94%	356.53	335.36	-5.94%	8612.44	9167.13	6.44%
4 cores	5091.10	4768.66	-6.33%	1303.31	1220.77	-6.33%	2353.32	2513.66	6.81%
8 cores	10836.58	9912.10	-8.53%	2774.15	2537.49	-8.53%	1104.30	1206.15	9.22%
12 cores	14411.73	13585.35	-5.73%	3689.39	3477.84	-5.73%	827.30	883.93	6.84%
16 cores	16953.45	15809.82	-6.75%	4340.07	4047.30	-6.75%	703.15	755.95	7.51%
20 cores	18492.86	17290.03	-6.50%	4734.16	4426.24	-6.50%	644.01	689.72	7.10%
24 cores	19527.63	18451.51	-5.51%	4999.06	4723.58	-5.51%	609.47	645.50	5.91%
28 cores	19350.45	18823.54	-2.72%	4953.70	4818.81	-2.72%	616.29	635.06	3.05%
32 cores	19310.48	18016.78	-6.70%	4943.47	4612.28	-6.70%	619.88	662.43	6.86%
36 cores	19281.84	19737.13	2.36%	4936.14	5052.69	2.36%	618.59	603.28	-2.48%
40 cores	20285.54	19102.79	-5.83%	5193.09	4890.30	-5.83%	586.89	625.52	6.58%



Large Sequential I/O Performance

We measured the performance of large block I/O workloads by performing sequential I/Os of size 128KBs at queue depth 4. We used iodepth=4 because higher queue depth resulted in negligible bandwidth gain and a significant increase in the latency. The rest of the FIO configuration is similar to the 4KB test case in the previous part of this document.

Test Result: 128KB 100% Sequential Reads, QD=4

# of Cores	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	6066.86	48.5	2667.8
4 cores	18914.87	151.3	845.8
8 cores	20298.58	162.4	788.0
12 cores	20012.00	160.1	799.9
16 cores	20208.98	161.7	791.7
20 cores	20122.50	161.0	795.0
24 cores	20082.41	160.7	796.7
28 cores	20035.81	160.3	798.4
32 cores	19923.88	159.4	802.9
36 cores	20269.61	162.2	789.3
40 cores	19535.86	156.3	819.8

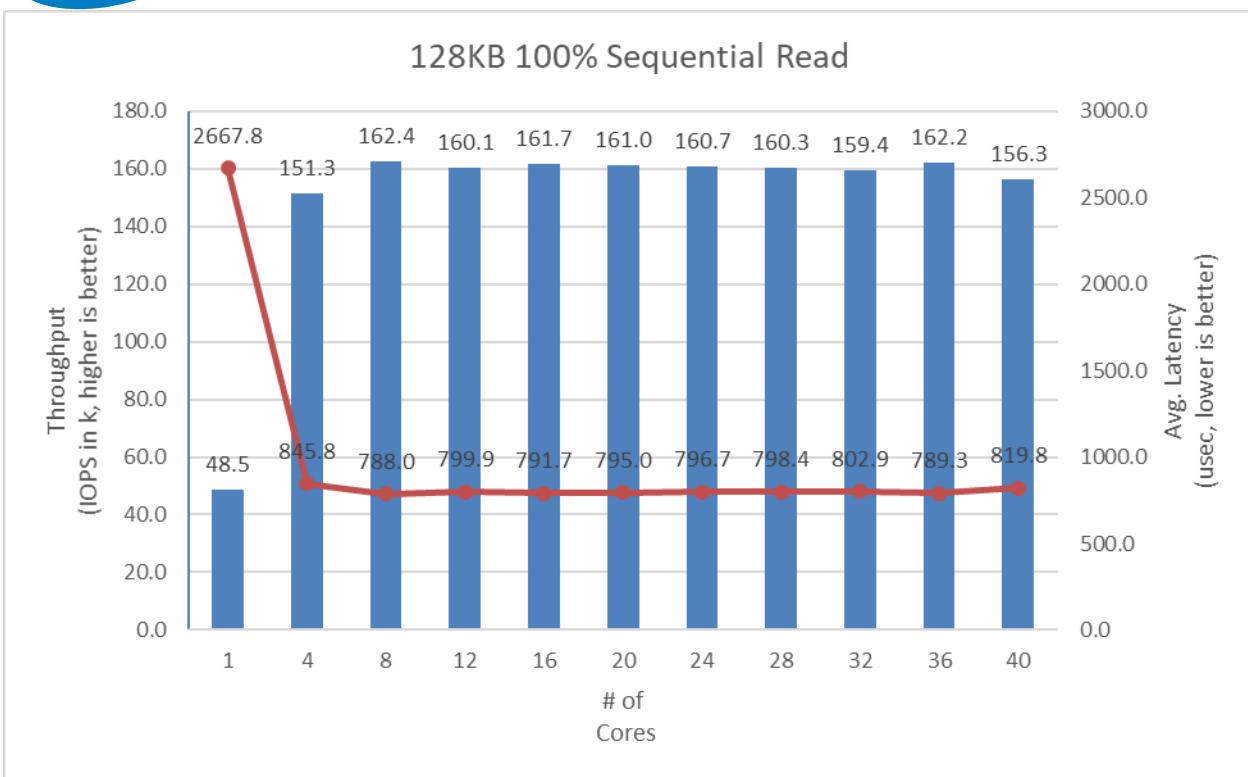


Figure 5: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 128KB 100% Sequential Read Workload at QD=4 and initiator FIO numjobs=2

Test Result: 128KB 100% Sequential Writes, QD=4

# of Cores	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	2168.49	17.3	7383.5
4 cores	8498.98	68.0	1885.2
8 cores	14896.24	119.2	1080.4
12 cores	18481.82	147.9	871.9
16 cores	20271.58	162.2	792.9
20 cores	21100.45	168.8	758.0
24 cores	21699.07	173.6	737.2
28 cores	19654.57	157.2	817.1
32 cores	22313.47	178.5	719.1
36 cores	21839.72	174.7	733.0
40 cores	20267.95	162.1	824.5

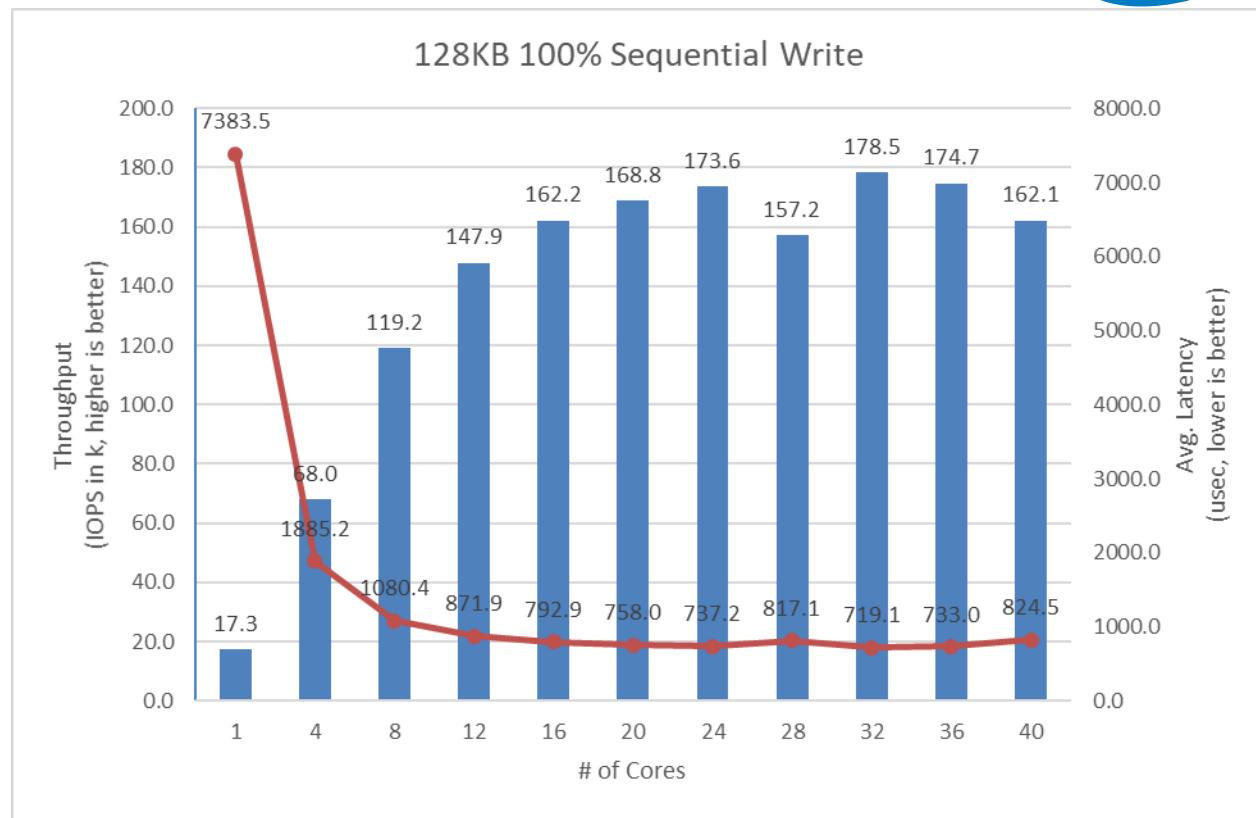


Figure 6: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 128KB 100% Sequential Write Workload at QD=4 and Initiator FIO numjobs=2

Test Result: 128KB Sequential 70% Reads 30% Writes, QD=4

# of Cores	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	3902.42	31.2	4109.0
4 cores	13740.46	109.9	1167.0
8 cores	18417.47	147.3	868.5
12 cores	19876.60	159.0	804.5
16 cores	20289.98	162.3	789.3
20 cores	19802.68	158.4	808.0
24 cores	20785.72	166.3	769.7
28 cores	20716.43	165.7	771.8
32 cores	21456.82	171.7	745.0
36 cores	20838.79	166.7	767.3
40 cores	20690.26	165.5	773.0

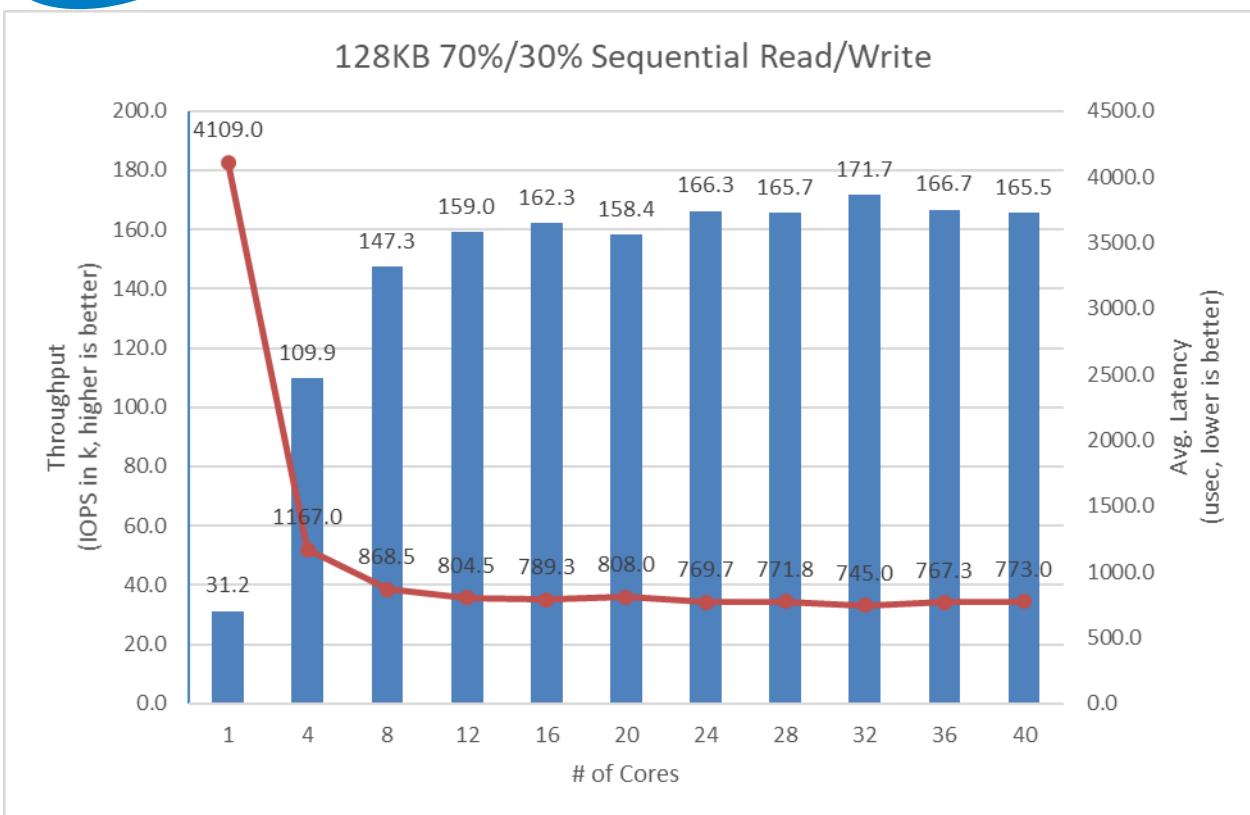


Figure 7: SPDK NVMe-oF TCP Target I/O core scaling: IOPS vs. Latency while running 128KB Sequential 70% Read 30% Write Workload at QD=4 and Initiator FIO numjobs=2



Conclusions

1. The SPDK NVMe-oF TCP Target IOPS throughput scales up almost linearly with addition of CPU cores for 4KB Random Read workload up to 12 CPU cores. Beyond 12 CPU cores, there are small IOPS gains that are non-linear. Peak performance of 5 million IOPS is reached at 20 CPU cores, which is close to saturating 200Gb network link.
2. For Random Write workload throughput scales almost linearly up to 16 CPU cores. Further improvements are non-linear. Peak measured performance is reached at 40 CPUs with 4.2 million IOPS.
3. Mixed Random Read-Write workload scales up linearly up to 16 CPU cores. Maximum measured performance is reached at 40CPU cores with 5.2 million IOPS.
4. The best trade-off between CPU efficiency and network saturation was observed when the Target was configured with 12 CPU cores. The performance we achieved fully saturated a 100Gbps NIC connection between Target and Initiator for all tested workloads. We added another 28CPU cores but could not fully saturate a 200Gbps network.
5. For the 4KB Random Write workload, we saturated the NVMe drives, which if preconditioned, would max out at about 3.2 million IOPS. Not preconditioning the drives allowed us to artificially increase their throughput and serve more IO requests than usual.
6. Previous SPDK NVMe-oF TCP reports were generated using 9000B MTU (Jumbo Frames) at the Transport Layer. Recent improvements in SPDK show that NVMe-oF TCP performance with Jumbo Frames is no longer ahead of standard 1500B MTU Frames.
7. For the Sequential 128k Read and Mixed Read/Write workloads, the IOPS throughput scaled up with addition of CPU cores up to 8 CPU cores and remained constant as we added more CPU cores. The network bandwidth reported by FIO was about 160Gbps, which is close to network saturation considering the network overhead.

Test Case 2: SPDK NVMe-oF TCP Initiator I/O core scaling

This test case was performed in order to understand the performance of SPDK NVMe-oF TCP Initiator as the number of CPU cores is scaled up.

The test setup for this test case is slightly different than the set up described in [introduction chapter](#), we used just a single SPDK NVMe-oF TCP Initiator to make it easier to understand of how the number of CPUs affects initiator IOPS throughput. The Initiator was connected to Target server with 100 Gbps network link.

The SPDK NVMe-oF TCP Target was configured similarly as in test case 1, using 20 cores. We used 20 CPU cores based on results of the previous test case which show that the target can easily serve over 3 million IOPS, that is enough IOPS to saturate 100 Gbps network connection

The SPDK bdev FIO plugin was used to target 16 individual NVMe-oF subsystems exported by the Target. The number of total CPU threads used by the FIO process was managed by setting the FIO job sections and numjobs parameter and ranged from 1 to 40 CPUs. For detailed FIO job configuration see table below. FIO was run with following workloads:

- 4KB 100% Random Read
- 4KB 100% Random Write
- 4KB Random 70% Read 30% Write

Item	Description
Test Case	Test SPDK NVMe-oF TCP Initiator I/O core scaling
SPDK NVMe-oF Target configuration	Same as in Test Case #1, using 20 CPU cores.
SPDK NVMe-oF Initiator 1 - FIO plugin configuration	BDEV.conf [Nvme] TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode1" Nvme0 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode2" Nvme1 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode3" Nvme2 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode4" Nvme3 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode5" Nvme4 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode6" Nvme5 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode7" Nvme6 TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode8" Nvme7 TransportId "trtype:TCP adrfam:IPv4 traddr:10.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode1" Nvme8 TransportId "trtype:TCP adrfam:IPv4 traddr:10.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode2" Nvme9 TransportId "trtype:TCP adrfam:IPv4 traddr:10.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode3" Nvme10 TransportId "trtype:TCP adrfam:IPv4 traddr:10.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode4" Nvme11 TransportId "trtype:TCP adrfam:IPv4 traddr:10.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode5" Nvme12 TransportId "trtype:TCP adrfam:IPv4 traddr:10.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode6" Nvme13 TransportId "trtype:TCP adrfam:IPv4 traddr:10.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode7" Nvme14 TransportId "trtype:TCP adrfam:IPv4 traddr:10.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode8" Nvme15



	<p>FIO.conf</p> <p>For 1 CPU initiator configuration:</p> <pre>[global] ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin spdk_conf=/tmp/spdk/bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth={32, 64, 128, 256} time_based=1 ramp_time=60 runtime=300 numjobs=1 [filename0] filename=Nvme0n1 filename=Nvme1n1 filename=Nvme2n1 filename=Nvme3n1 filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1 filename=Nvme8n1 filename=Nvme9n1 filename=Nvme10n1 filename=Nvme11n1 filename=Nvme12n1 filename=Nvme13n1 filename=Nvme14n1 filename=Nvme15n1</pre>
	<p>FIO.conf</p> <p>For X*4 CPU (up to 40) initiator configuration:</p> <pre>[global] ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin spdk_conf=/tmp/spdk/bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth={32, 64, 128, 256} time_based=1 ramp_time=60 runtime=300 numjobs=X</pre>

	[filename0] filename=Nvme0n1 filename=Nvme1n1 filename=Nvme2n1 filename=Nvme3n1 [filename1] filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1 [filename2] filename=Nvme8n1 filename=Nvme9n1 filename=Nvme10n1 filename=Nvme11n1 [filename3] filename=Nvme12n1 filename=Nvme13n1 filename=Nvme14n1 filename=Nvme15n1
--	--

4KB Random Read Results

Test Result: 4KB 100% Random Read, QD=64

# of Cores	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	651.67	166.8	375.5
4 cores	2651.28	678.7	371.6
8 cores	4739.42	1213.3	416.2
12 cores	6768.67	1732.8	437.7
16 cores	8420.07	2155.5	469.6
20 cores	9374.52	2399.9	527.9
24 cores	9180.83	2350.3	648.4
28 cores	9125.03	2336.0	763.0
32 cores	8721.03	2232.6	914.5
36 cores	7936.85	2031.8	1130.1
40 cores	7523.39	1926.0	1326.4

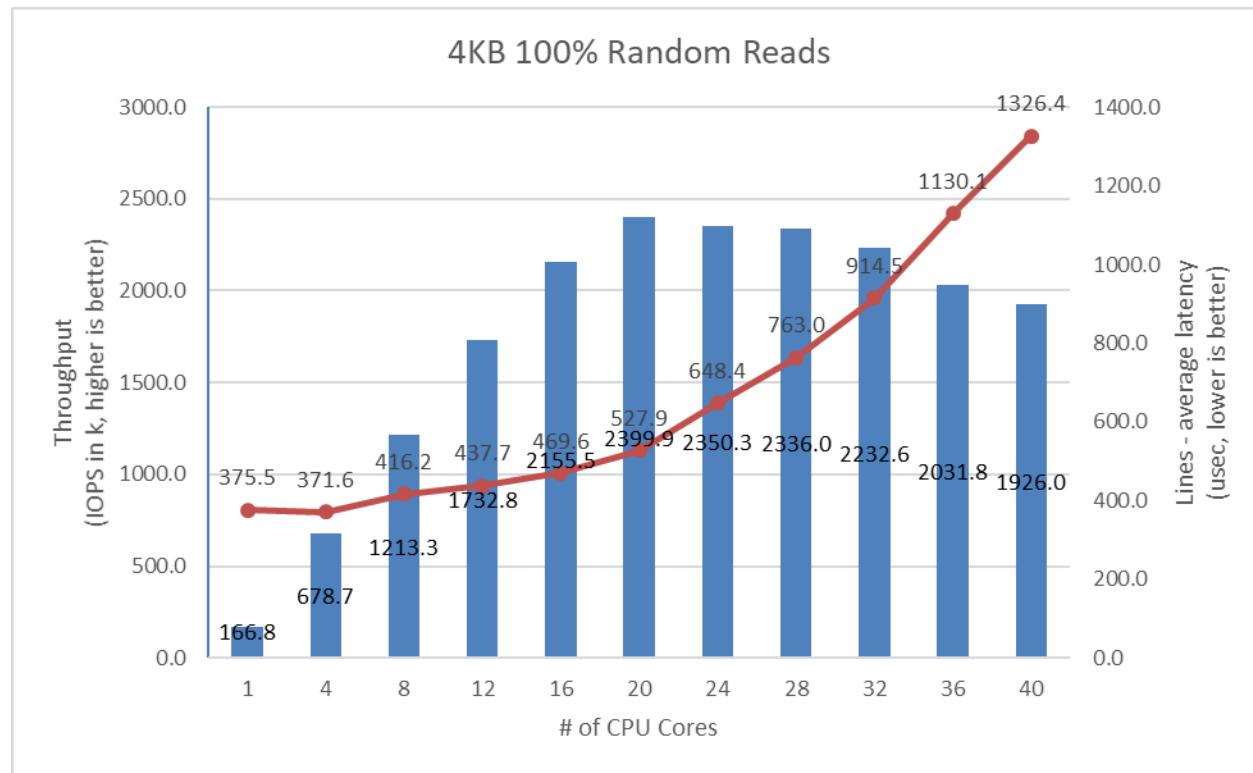


Figure 8: SPDK NVMe-oF TCP Initiator I/O core scaling: IOPS vs. Latency while running 4KB 100% Random Read QD=64 workload

4KB Random Write Results

Test Result: 4KB 100% Random Write, QD=64

# of Cores	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	855.95	219.1	284.7
4 cores	3430.86	878.3	285.1
8 cores	6083.57	1557.4	322.4
12 cores	8109.97	2076.1	364.1
16 cores	8198.13	2098.7	484.5
20 cores	9316.10	2384.9	532.5
24 cores	9355.94	2395.1	637.7
28 cores	9572.00	2450.4	726.9
32 cores	9311.23	2383.7	855.0
36 cores	9360.75	2396.3	957.6
40 cores	9477.75	2426.3	1050.4

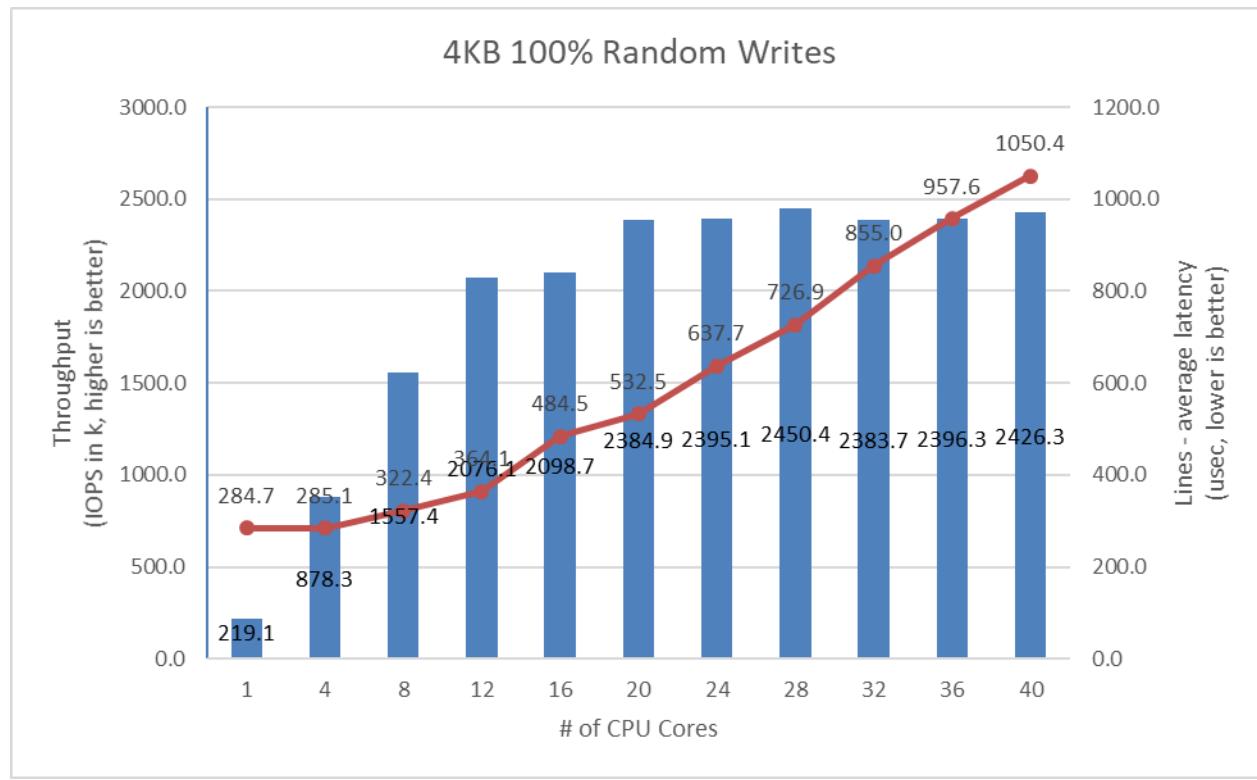


Figure 9: SPDK NVMe-oF TCP Initiator I/O core scaling: IOPS vs. Latency while running 4KB 100% Random Write Workload at QD=64

4KB Random Read-Write Results

Test Result: 4KB 70% Random Read 30% Random Write, QD=64

# of Cores	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	748.65	191.7	324.5
4 cores	2539.93	650.2	387.5
8 cores	4601.63	1178.0	428.5
12 cores	6535.39	1673.1	453.2
16 cores	8552.91	2189.5	461.8
20 cores	9949.31	2547.0	496.4
24 cores	10850.80	2777.8	547.4
28 cores	11316.92	2897.1	613.1
32 cores	11873.72	3039.7	668.2
36 cores	11319.54	2897.8	789.6
40 cores	11018.88	2820.8	901.9

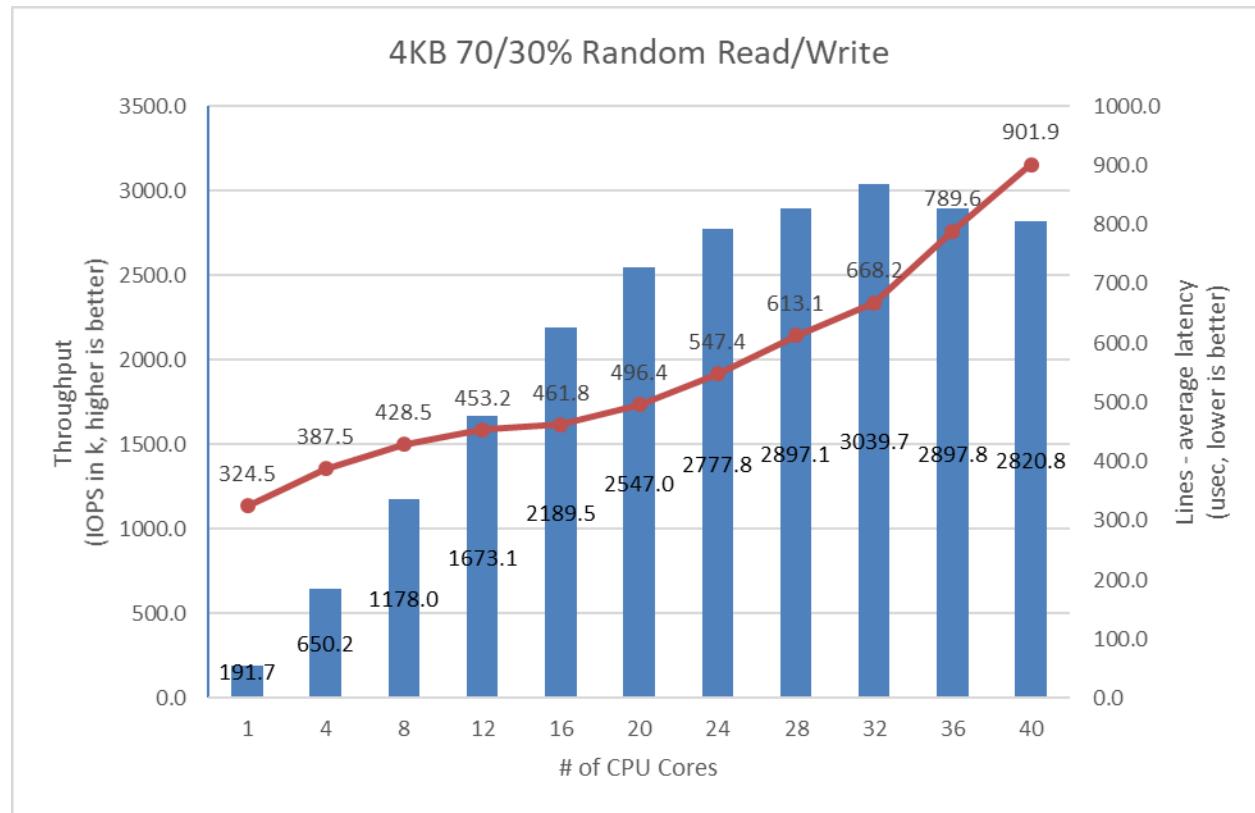


Figure 10: SPDK NVMe-oF TCP Initiator I/O core scaling: IOPS vs. Latency while running 4KB Random 70% Read 30% Write Workload at QD=64

Conclusions

1. For Random Read workload SPDK NVMe-oF TCP Initiator performance scales linearly up to 20 CPU cores. Transitioning beyond 20 CPU cores does not improve results.
2. In case of Random Write workload performance scales linearly up to 20 CPU cores. Peak performance of 2.4M IOPS is reached at 20 CPU cores. Further increasing number of Initiator cores does not improve results.
3. Mixed Random Read-Write workload performance scales linearly up to 32 CPU cores, reaching around 3.0M IOPS and saturating the network link.



Test Case 3: Linux Kernel vs. SPDK NVMe-oF TCP Latency

This test case was designed to understand latency characteristics of SPDK NVMe-oF TCP Target and Initiator vs. the Linux Kernel NVMe-oF TCP Target and Initiator implementations on a single NVMe-oF subsystem. The average I/O latency and p99 latency was compared between SPDK NVMe-oF (Target/Initiator) vs. Linux Kernel (Target/Initiator). Both SPDK and Kernel NVMe-oF Targets were configured to run on a single core, with a single NVMe-oF subsystem containing a *Null Block Device*. The null block device (bdev) was chosen as the backend block device to eliminate the media latency during these tests.

Item	Description
Test Case	Linux Kernel vs. SPDK NVMe-oF Latency
Test configuration	
SPDK NVMe-oF Target configuration	<p>All of below commands are executed with spdk/scripts/rpc.py script.</p> <pre>nvmf_create_transport -t TCP (create TCP transport layer with default values: trtype: "TCP" max_queue_depth: 128 max_qpairs_per_ctrlr: 64 in_capsule_data_size: 4096 max_io_size: 131072 io_unit_size: 8192 max_aq_depth: 128 num_shared_buffers: 4096 buf_cache_size: 32) construct_null_bdev Nvme0n1 10240 4096 nvmf_subsystem_create nqn.2018-09.io.spdk:cnode1 -s SPDK001 -a -m 8 nvmf_subsystem_add_ns nqn.2018-09.io.spdk:cnode1 Nvme0n1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode1 -t tcp -f ipv4 -s 4420 -a 20.0.0.1</pre>
Kernel NVMe-oF Target configuration	<p>Target configuration file loaded using nvmet-cli tool.</p> <pre>{ "ports": [{ "addr": { "adrfam": "ipv4", "traddr": "20.0.0.1", "trsvid": "4420", "trtype": "tcp" }, "portid": 1, "referrals": [], "subsystems": ["nqn.2018-09.io.spdk:cnode1"] }] }</pre>

	<pre>], "hosts": [], "subsystems": [{ "allowed_hosts": [], "attr": { "allow_any_host": "1", "version": "1.3" }, "namespaces": [{ "device": { "path": "/dev/nullb0", "uuid": "621e25d2-8334-4c1a-8532-b6454390b8f9" }, "enable": 1, "nsid": 1 }], "nqn": "nqn.2018-09.io.spdk:cnode1" }] }</pre>
FIO configuration	
SPDK NVMe-oF Initiator FIO plugin configuration	<p>BDEV.conf</p> <pre>[Nvme] TransportId "trtype:TCP adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode1" Nvme0</pre> <p>FIO.conf</p> <pre>[global] ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin spdk_conf=/tmp/spdk/bdev.conf thread=1 group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth=1 time_based=1 ramp_time=60 runtime=300 [filename0] filename=NvmeOn1</pre>
Kernel initiator configuration	<p>Device config</p> <p>Done using nvme-cli tool. <code>modprobe nvme-fabrics</code> <code>nvme connect -n nqn.2018-09.io.spdk:cnode1 -t tcp -a 20.0.0.1 -s 4420</code></p> <p>FIO.conf</p> <pre>[global] ioengine=libaio thread=1</pre>



```
group_reporting=1
direct=1

norandommap=1
rw=randrw
rwmixread={100, 70, 0}
bs=4k
iodepth=1
time_based=1
numjobs=1
ramp_time=60
runtime=300

[filename0]
filename=/dev/nvme0n1
```

SPDK vs Kernel NVMe-oF Target Latency Results

This following data was collected using the Linux Kernel initiator against both SPDK & Linux Kernel NVMe-oF TCP target.

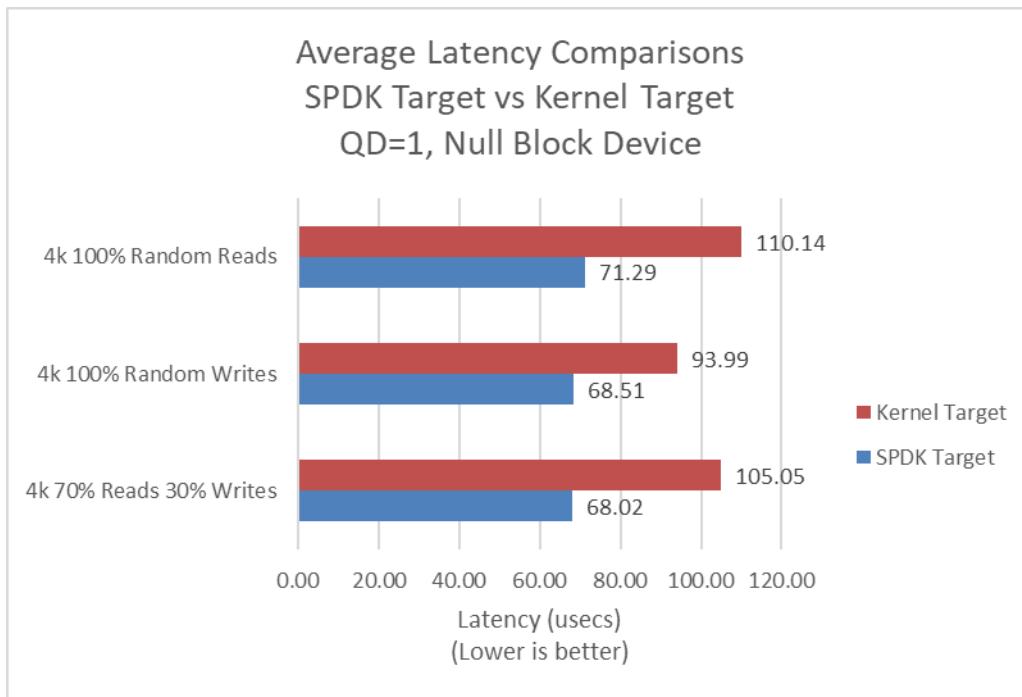


Figure 11: SPDK vs. Kernel NVMe-oF TCP Average I/O Latency for various workloads run using the Kernel Initiator

SPDK NVMe-oF Target Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4KB 100% Random Reads IOPS	71.29	13673	61.4
4KB 100% Random Writes IOPS	68.51	14203	57.9
4KB 100% Random 70% Reads 30% Writes IOPS	68.02	14301.27	62.34

Linux Kernel NVMe-oF Target Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4KB 100% Random Reads IOPS	110.14	8923	168.3
4KB 100% Random Writes IOPS	93.99	10433	89.3
4KB 100% Random 70% Reads 30% Writes IOPS	105.05	9339.20	157.29

SPDK vs Kernel NVMe-oF TCP Initiator results

The following data was collected using Kernel & SPDK initiator against an SPDK target.

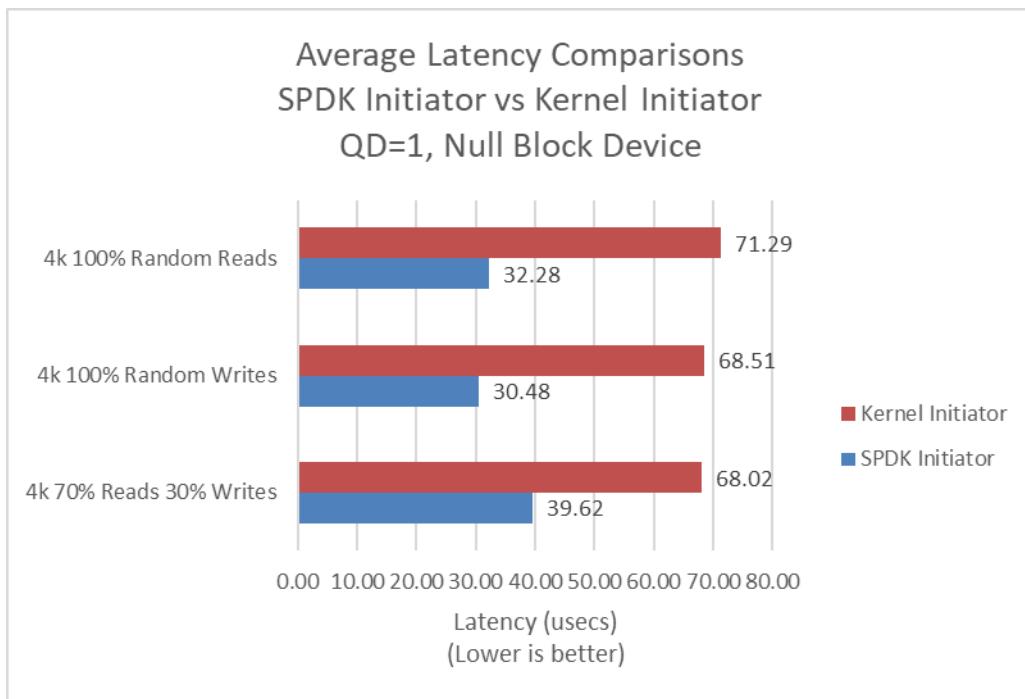


Figure 12: SPDK vs. Kernel NVMe-oF TCP Average I/O Latency for various workloads against SPDK Target

Linux Kernel NVMe-oF Initiator Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4KB 100% Random Reads IOPS	71.29	13673	61.4
4KB 100% Random Writes IOPS	68.51	14203	57.9
4KB 100% Random 70% Reads 30% Writes IOPS	68.02	14301	62.3

SPDK NVMe-oF Initiator Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4KB 100% Random Reads IOPS	32.28	30745	48.2
4KB 100% Random Writes IOPS	30.48	32578	38.8
4KB 100% Random 70% Reads 30% Writes IOPS	39.62	25163	80.1

SPDK vs Kernel NVMe-oF Latency Results

Following data was collected using SPDK Target with SPDK Initiator and Linux Target with Linux Initiator.

SPDK NVMe-oF Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4KB 100% Random Reads IOPS	32.28	30745	48.2
4KB 100% Random Writes IOPS	30.48	32578	38.8
4KB 100% Random 70% Reads 30% Writes IOPS	39.62	25163	80.1

Linux Kernel NVMe-oF Latency and IOPS at QD=1, Null Block Device

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4KB 100% Random Reads IOPS	110.14	8923	168.28
4KB 100% Random Writes IOPS	93.99	10433	89.26
4KB 100% Random 70% Reads 30% Writes IOPS	105.05	9339	157.29

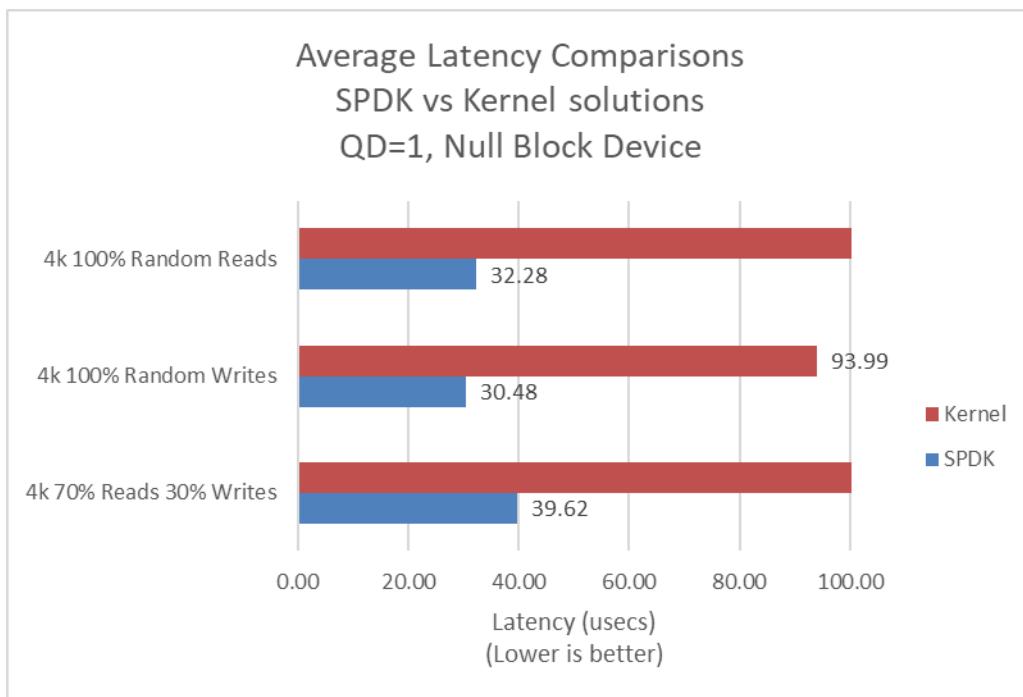


Figure 13: SPDK vs. Kernel NVMe-oF TCP solutions Average I/O Latency for various workloads



Conclusions

1. SPDK NVMe-oF Initiator reduces the average latency by up to 39 usec vs. the Linux Kernel NVMe-oF Initiator, which eliminates up to 45% NVMe-oF software overhead.
2. The SPDK NVMe-oF Target reduces the NVMe-oF average round trip I/O latency (reads/writes) by up to 39 usec vs. the Linux Kernel NVMe-oF target. This is entirely software overhead.
3. The SPDK NVMe-oF TCP target and initiator reduced the average latency by up to 70% vs. the Linux Kernel NVMe-oF target and initiator.
4. The SPDK NVMe-oF Initiator reduces the p99 latency by 20% and 35% for the 4KB random reads and write workloads respectively.

Test Case 4: NVMe-oF Performance with increasing # of connections

This test case was performed in order to understand throughput and latency capabilities of SPDK NVMe-oF Target vs. Linux Kernel NVMe-oF Target under increasing number of connections per subsystem. The number of connections (or I/O queue pairs) per NVMe-oF subsystem were varied and corresponding aggregated IOPS and number of CPU cores metrics were reported. The number of CPU cores metric was calculated from %CPU utilization measured using sar (systat package in Linux). The SPDK NVMe-oF Target was configured to run on 30 cores, 16 NVMe-oF subsystems (1 per Intel P4600) and 2 initiators were used both running I/Os to 8 separate subsystems using Kernel NVMe-oF initiator. We ran the following workloads on the host systems:

- 4KB 100% Random Read
- 4KB 100% Random Write
- 4KB Random 70% Read 30% Write

Item	Description
Test Case	NVMe-oF Target performance under varying # of connections
SPDK NVMe-oF Target configuration	Same as in Test Case #1, using 30 CPU cores.
Kernel NVMe-oF Target configuration	Target configuration file loaded using nvmet-cli tool. For detail configuration file contents please see Appendix A.
Kernel NVMe-oF Initiator #1	Device config Performed using nvme-cli tool. modprobe nvme-fabrics nvme connect -n nqn.2018-09.io.spdk:cnode1 -t tcp -a 20.0.0.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode2 -t tcp -a 20.0.0.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode3 -t tcp -a 20.0.0.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode4 -t tcp -a 20.0.0.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode5 -t tcp -a 20.0.1.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode6 -t tcp -a 20.0.1.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode7 -t tcp -a 20.0.1.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode8 -t tcp -a 20.0.1.1 -s 4420
Kernel NVMe-oF Initiator #2	Device config Performed using nvme-cli tool. modprobe nvme-fabrics nvme connect -n nqn.2018-09.io.spdk:cnode9 -t tcp -a 10.0.0.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode10 -t tcp -a 10.0.0.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode11 -t tcp -a 10.0.0.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode12 -t tcp -a 10.0.0.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode13 -t tcp -a 10.0.1.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode14 -t tcp -a 10.0.1.1 -s 4420 nvme connect -n nqn.2018-09.io.spdk:cnode15 -t tcp -a 10.0.1.1 -s 4420



	<pre>nvme connect -n nqn.2018-09.io.spdk:cnode16 -t tcp -a 10.0.1.1 -s 4420</pre>
FIO configuration (used on both initiators)	<pre>FIO.conf [global] ioengine=libaio thread=1 group_reporting=1 direct=1 norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth={8, 16, 32, 64, 128} time_based=1 ramp_time=60 runtime=300 numjobs={1, 4, 16, 32} [filename1] filename=/dev/nvme0n1 [filename2] filename=/dev/nvme1n1 [filename3] filename=/dev/nvme2n1 [filename4] filename=/dev/nvme3n1 [filename5] filename=/dev/nvme4n1 [filename6] filename=/dev/nvme5n1 [filename7] filename=/dev/nvme6n1 [filename8] filename=/dev/nvme7n1</pre>

The number of CPU cores used while running the SPDK NVMe-oF target was 30, whereas for the case of Linux Kernel NVMe-oF target there was no CPU core limitation applied.

The numbers in the graph represent relative performance in IOPS/core which was calculated based on total aggregate IOPS divided by total CPU cores used while running that specific workload. For the case of Kernel NVMe-oF target, total CPU cores was calculated from % CPU utilization which was measured using sar utility in Linux.

4KB Random Read Results

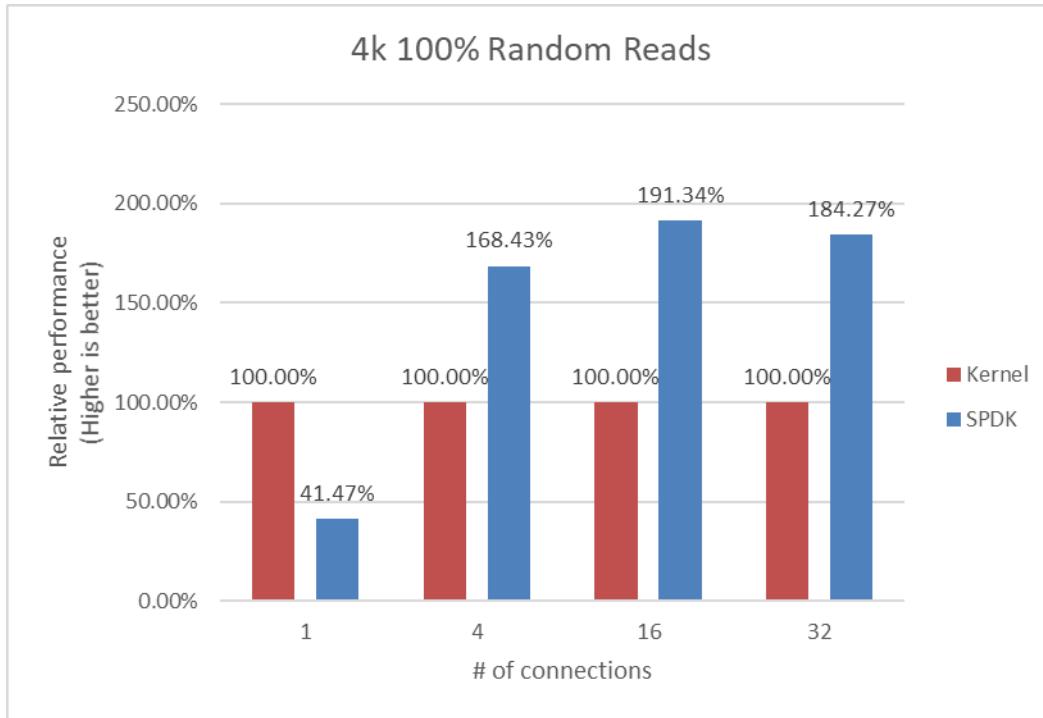


Figure 14: Relative Performance Comparison of Linux Kernel vs. SPDK NVMe-oF Target for 4KB 100% Random Reads using the Kernel Initiator

Linux Kernel NVMe-oF TCP Target: 4KB 100% Random Reads, QD=64

Connections per subsystem	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	2727.02	698.1	1466.2	11.3
4	12425.17	3180.8	1286.8	44.1
16	14365.18	3677.4	4457.3	59.7
32	14154.61	3623.4	9045.8	60.1

SPDK NVMe-oF TCP Target: 4KB 100% Random Reads, QD=64

Connections per subsystem	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	2990.40	765.5	1337.2	30.0
4	14226.37	3641.9	1123.9	30.0
16	13806.41	3534.4	4634.9	30.0
32	13022.34	3333.6	9844.8	30.0

4KB Random Write Results

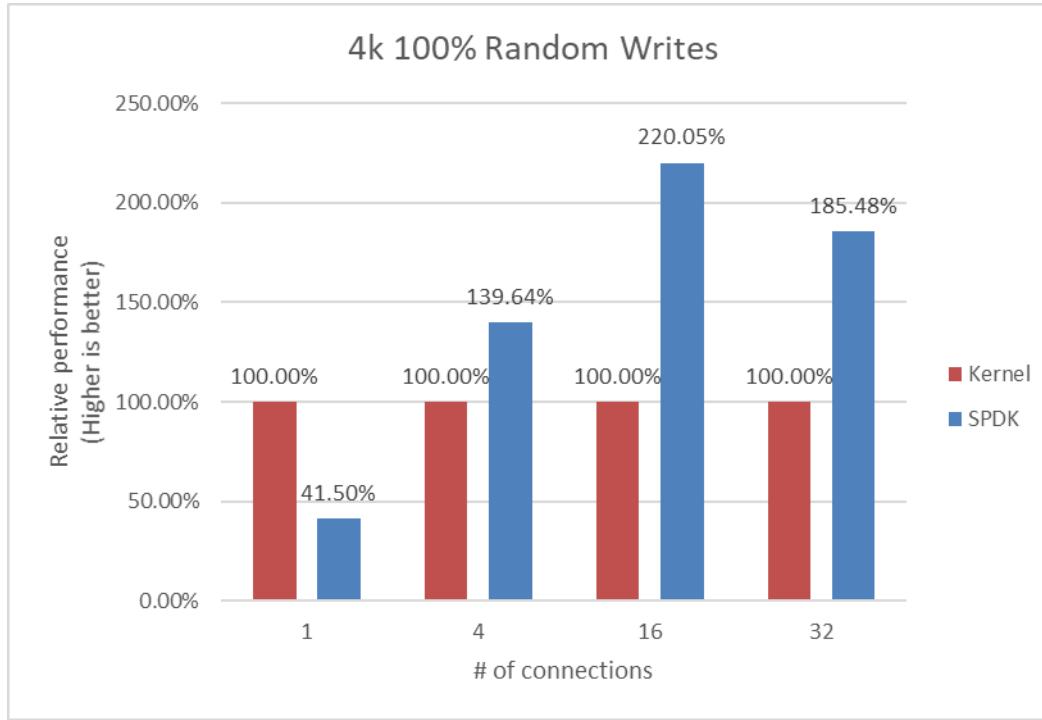


Figure 15: Relative Performance Comparison of Linux Kernel vs. SPDK NVMe-oF Target for 4KB 100% Random Writes

Note: Drives were not pre-conditioned while running 100% Random write I/O Test

Linux Kernel NVMe-oF TCP Target: 4KB 100% Random Writes, QD=64

Connections per subsystem	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	3309.61	847.3	1208.8	12.4
4	11144.17	2852.9	1461.7	40.9
16	14549.16	3724.5	4398.1	72.2
32	14305.72	3662.1	8944.2	74.9

SPDK NVMe-oF TCP Target: 4KB 100% Random Writes, QD=64

Connections per subsystem	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	3320.38	850.0	1204.4	30.0
4	11411.03	2921.2	1407.5	30.0
16	13302.34	3405.3	4810.5	30.0
32	10623.46	2719.5	12059.7	30.0

4KB Random Read-Write Results

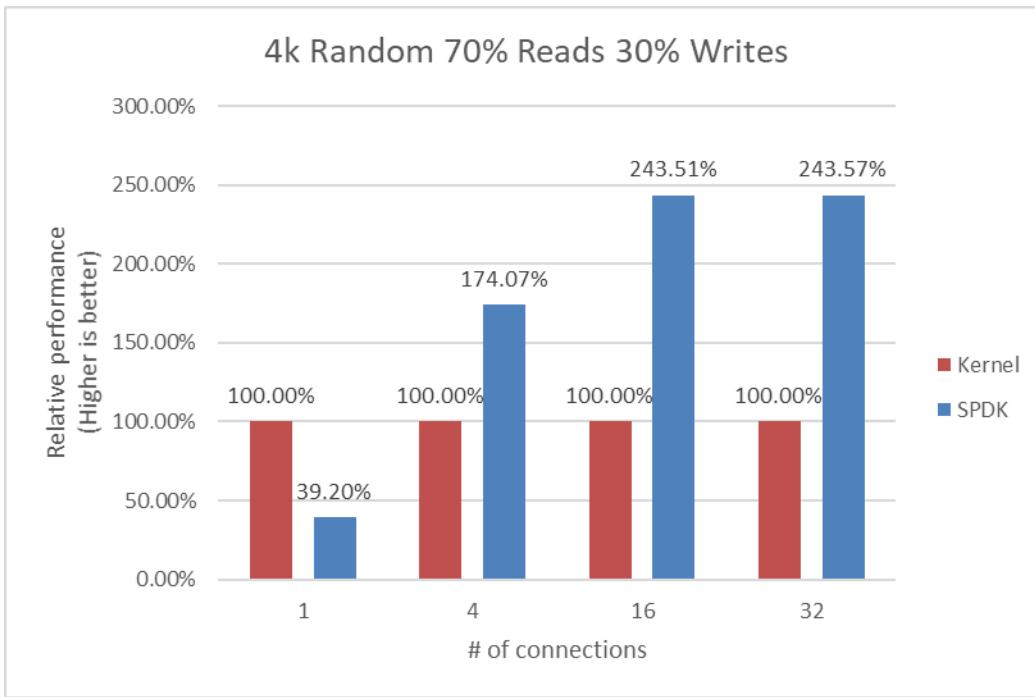


Figure 16: Relative Performance Comparison of Linux Kernel vs. SPDK NVMe-oF Target for 4KB Random 70% Reads 30% Writes

Linux Kernel NVMe-oF TCP Target: 4KB 70% Random Read 30% Random Write, QD=64

Connections per subsystem	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	2857.85	731.6	1290.4	11.4
4	12298.91	3148.5	1181.8	47.0
16	15522.78	3973.8	3711.0	72.0
32	15383.60	3938.1	7617.8	73.2

SPDK NVMe-oF TCP Target: 4KB 70% Random Read 30% Random Write, QD=64

Connections per subsystem	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
1	2941.85	753.1	1205.6	30.0
4	13657.05	3496.2	1032.2	30.0
16	15744.97	4030.6	1822.5	30.0
32	15361.70	3932.5	3552.9	30.0



Low Connections Results

During testing it was observed that relative performance of SPDK Target is about 60-70% of Kernel Target performance. This is because SPDK uses a fixed number of CPU cores and does not have a mechanism to decrease the number of cores on the fly if it does not use all of the CPU resources.

The test cases with 1 connection per subsystems were re-run with SPDK using only 4 CPU cores.

[SPDK & Kernel NVMe-oF TCP Target relative performance comparison for various workloads, QD=64, 1 connection per subsystem.](#)

Workload	Target	Bandwidth (MBps)	Throughput (IOPS k)	Avg. Latency (usec)	# CPU Cores
Random Read	Linux	2727.02	698.1	1466.2	11.3
	SPDK	2913.75	745.9	1372.2	4.0
Random Write	Linux	3309.61	847.3	1208.8	12.4
	SPDK	2669.92	683.5	1497.3	4.0
Random Read/Write	Linux	2857.85	731.6	2736.4	11.4
	SPDK	2849.14	729.4	2723.9	4.0

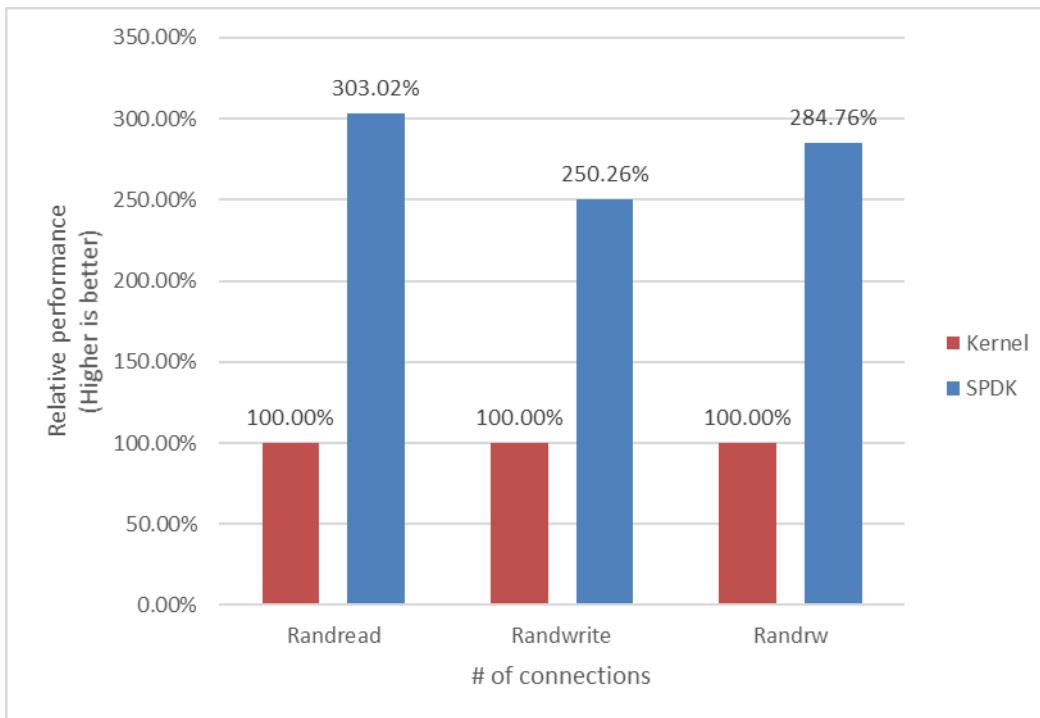


Figure 17: Relative Performance Comparison of Linux Kernel vs. SPDK NVMe-oF Target for various workloads, 1 connection per subsystem and reduced number of SPDK Target CPU Cores (4)

Conclusions

1. The performance of SPDK NVMe-oF TCP Target for all Random Read workload peaked when the number of connections per subsystem was 4, and 16 for Random Write and Random Read/Write workloads.
2. Optimum configuration of number of connections per subsystem seems to be 4 and allows for best performance in used setup. IOPS numbers peaked for Random Write and Random Read/Write at 16 connections, but at a cost of high increase in latency. Increasing the number of connections beyond that value results in minimal or no performance improvement.
3. Relative performance of Kernel NVMe-oF TCP Target was better than SPDK in cases where there was just one connection per subsystem because SPDK uses a fixed number of CPU cores and there is no mechanism which would allow us to dynamically decrease the number of cores if needed.

For this reason, we've re-run the test cases where number of connections per subsystem was 1 but lowered the number of SPDK Target CPU cores down to 4 and added the results to the tables.

4. SPDK relative performance was between 1.4 - 2.4 times better in all other test cases where each subsystem had multiple connections.



Summary

This report showcased performance results with SPDK NVMe-oF TCP target and initiator under various test cases, including:

- I/O core scaling
- Average I/O latency
- Performance with increasing number of connections per subsystems

It compared performance results while running Linux Kernel NVMe-oF (Target/Initiator) against the accelerated polled-mode driven SPDK NVMe-oF (Target/Initiator) implementation.

Throughput scales up and latency decreases almost linearly with the scaling of SPDK NVMe-oF target cores when serving 4KB random workloads until the network traffic reaches around 100 Gb saturation around 8-12 CPU cores. Beyond that the trend becomes non-linear, it took almost at 40 CPU cores for the target to almost saturate 200Gbps network link.

For the SDPK NVMe-oF TCP Initiator, the IOPS throughput scales almost linearly with addition of CPU cores until the network was almost saturated, however, as we got closer to network saturation it was observed that the throughput scaling becomes non-linear. A single initiator was able to almost saturate 100Gb link.

For the NVMe-oF TCP latency comparison, the SPDK NVMe-oF Target and Initiator average latency is up to 50% lower than their Kernel counterparts when testing against null bdev based backend.

The SPDK NVMe-oF TCP Target performed up to 2.4 times better w.r.t IOPS/core than Linux Kernel NVMe-oF target while running 4KB 100% random read workload with increasing number of connections per NVMe-oF subsystem.

This report provides information regarding methodologies and practices while benchmarking NVMe-oF using SPDK, as well as the Linux Kernel. It should be noted that the performance data showcased in this report is based on specific hardware and software configurations and that performance results may vary depending on the hardware and software configurations.

Appendix A

Example Kernel NVMe-oF TCP Target configuration for Test Case 4.

```
{  
    "ports": [  
        {  
            "addr": {  
                "adrfam": "ipv4",  
                "traddr": "20.0.0.1",  
                "trsvcid": "4420",  
                "trtype": "tcp"  
            },  
            "portid": 1,  
            "referrals": [],  
            "subsystems": [  
                "nqn.2018-09.io.spdk:cnode1"  
            ]  
        },  
        {  
            "addr": {  
                "adrfam": "ipv4",  
                "traddr": "20.0.0.1",  
                "trsvcid": "4421",  
                "trtype": "tcp"  
            },  
            "portid": 2,  
            "referrals": [],  
            "subsystems": [  
                "nqn.2018-09.io.spdk:cnode2"  
            ]  
        },  
        {  
            "addr": {  
                "adrfam": "ipv4",  
                "traddr": "20.0.0.1",  
                "trsvcid": "4422",  
                "trtype": "tcp"  
            },  
            "portid": 3,  
            "referrals": [],  
            "subsystems": [  
                "nqn.2018-09.io.spdk:cnode3"  
            ]  
        },  
        {  
            "addr": {  
                "adrfam": "ipv4",  
                "traddr": "20.0.0.1",  
                "trsvcid": "4423",  
                "trtype": "tcp"  
            },  
            "portid": 4,  
            "referrals": [],  
            "subsystems": [  
                "nqn.2018-09.io.spdk:cnode4"  
            ]  
        }  
    ]  
}
```



```
        ],
    },
    {
        "addr": {
            "adrfam": "ipv4",
            "traddr": "20.0.1.1",
            "trsvcid": "4424",
            "trtype": "tcp"
        },
        "portid": 5,
        "referrals": [],
        "subsystems": [
            "nqn.2018-09.io.spdk:cnode5"
        ]
    },
    {
        "addr": {
            "adrfam": "ipv4",
            "traddr": "20.0.1.1",
            "trsvcid": "4425",
            "trtype": "tcp"
        },
        "portid": 6,
        "referrals": [],
        "subsystems": [
            "nqn.2018-09.io.spdk:cnode6"
        ]
    },
    {
        "addr": {
            "adrfam": "ipv4",
            "traddr": "20.0.1.1",
            "trsvcid": "4426",
            "trtype": "tcp"
        },
        "portid": 7,
        "referrals": [],
        "subsystems": [
            "nqn.2018-09.io.spdk:cnode7"
        ]
    },
    {
        "addr": {
            "adrfam": "ipv4",
            "traddr": "20.0.1.1",
            "trsvcid": "4427",
            "trtype": "tcp"
        },
        "portid": 8,
        "referrals": [],
        "subsystems": [
            "nqn.2018-09.io.spdk:cnode8"
        ]
    },
    {
        "addr": {
            "adrfam": "ipv4",
```

```
        "traddr": "10.0.0.1",
        "trsvid": "4428",
        "trtype": "tcp"
    },
    "portid": 9,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode9"
    ]
},
{
    "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.0.1",
        "trsvid": "4429",
        "trtype": "tcp"
    },
    "portid": 10,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode10"
    ]
},
{
    "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.0.1",
        "trsvid": "4430",
        "trtype": "tcp"
    },
    "portid": 11,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode11"
    ]
},
{
    "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.0.1",
        "trsvid": "4431",
        "trtype": "tcp"
    },
    "portid": 12,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode12"
    ]
},
{
    "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvid": "4432",
        "trtype": "tcp"
    },
    "portid": 13,
```



```
"referrals": [],
"subsystems": [
    "nqn.2018-09.io.spdk:cnode13"
]
},
{
    "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4433",
        "trtype": "tcp"
    },
    "portid": 14,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode14"
    ]
},
{
    "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4434",
        "trtype": "tcp"
    },
    "portid": 15,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode15"
    ]
},
{
    "addr": {
        "adrfam": "ipv4",
        "traddr": "10.0.1.1",
        "trsvcid": "4435",
        "trtype": "tcp"
    },
    "portid": 16,
    "referrals": [],
    "subsystems": [
        "nqn.2018-09.io.spdk:cnode16"
    ]
},
],
"hosts": [],
"subsystems": [
    {
        "allowed_hosts": [],
        "attr": {
            "allow_any_host": "1",
            "version": "1.3"
        },
        "namespaces": [
            {
                "device": {
                    "path": "/dev/nvme0n1",

```

```
        "uuid": "b53be81d-6f5c-4768-b3bd-203614d8cf20"
    },
    "enable": 1,
    "nsid": 1
}
],
"nqn": "nqn.2018-09.io.spdk:cnode1"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
        {
            "device": {
                "path": "/dev/nvme1n1",
                "uuid": "12fcf584-9c45-4b6b-abc9-63a763455cf7"
            },
            "enable": 1,
            "nsid": 2
        }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode2"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
        {
            "device": {
                "path": "/dev/nvme2n1",
                "uuid": "ceae8569-69e9-4831-8661-90725bdf768d"
            },
            "enable": 1,
            "nsid": 3
        }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode3"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
        {
            "device": {
                "path": "/dev/nvme3n1",
                "uuid": "39f36db4-2cd5-4f69-b37d-1192111d52a6"
            },
            "enable": 1,
```



```

        "nsid": 4
    }
],
"nqn": "nqn.2018-09.io.spdk:cnode4"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
        {
            "device": {
                "path": "/dev/nvme4n1",
                "uuid": "984aed55-90ed-4517-ae36-d3afb92dd41f"
            },
            "enable": 1,
            "nsid": 5
        }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode5"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
        {
            "device": {
                "path": "/dev/nvme5n1",
                "uuid": "d6d16e74-378d-40ad-83e7-b8d8af3d06a6"
            },
            "enable": 1,
            "nsid": 6
        }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode6"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
        {
            "device": {
                "path": "/dev/nvme6n1",
                "uuid": "a65dc00e-d35c-4647-9db6-c2a8d90db5e8"
            },
            "enable": 1,
            "nsid": 7
        }
    ],

```

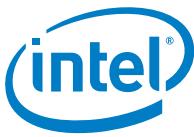


```
        "nqn": "nqn.2018-09.io.spdk:cnode7"
    },
    {
        "allowed_hosts": [],
        "attr": {
            "allow_any_host": "1",
            "version": "1.3"
        },
        "namespaces": [
            {
                "device": {
                    "path": "/dev/nvme7n1",
                    "uuid": "1b242cb7-8e47-4079-a233-83e2cd47c86c"
                },
                "enable": 1,
                "nsid": 8
            }
        ],
        "nqn": "nqn.2018-09.io.spdk:cnode8"
    },
    {
        "allowed_hosts": [],
        "attr": {
            "allow_any_host": "1",
            "version": "1.3"
        },
        "namespaces": [
            {
                "device": {
                    "path": "/dev/nvme8n1",
                    "uuid": "f12bb0c9-a2c6-4eef-a94f-5c4887bbf77f"
                },
                "enable": 1,
                "nsid": 9
            }
        ],
        "nqn": "nqn.2018-09.io.spdk:cnode9"
    },
    {
        "allowed_hosts": [],
        "attr": {
            "allow_any_host": "1",
            "version": "1.3"
        },
        "namespaces": [
            {
                "device": {
                    "path": "/dev/nvme9n1",
                    "uuid": "40fae536-227b-47d2-bd74-8ab76ec7603b"
                },
                "enable": 1,
                "nsid": 10
            }
        ],
        "nqn": "nqn.2018-09.io.spdk:cnode10"
    },
    {

```



```
"allowed_hosts": [],
"attr": {
    "allow_any_host": "1",
    "version": "1.3"
},
"namespaces": [
{
    "device": {
        "path": "/dev/nvme10n1",
        "uuid": "b9756b86-263a-41cf-a68c-5cfb23c7a6eb"
    },
    "enable": 1,
    "nsid": 11
}
],
"nqn": "nqn.2018-09.io.spdk:cnode11"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
{
    "device": {
        "path": "/dev/nvme11n1",
        "uuid": "9d7e74cc-97f3-40fb-8e90-f2d02b5fff4c"
    },
    "enable": 1,
    "nsid": 12
}
],
"nqn": "nqn.2018-09.io.spdk:cnode12"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
{
    "device": {
        "path": "/dev/nvme12n1",
        "uuid": "d3f4017b-4f7d-454d-94a9-ea75ffc7436d"
    },
    "enable": 1,
    "nsid": 13
}
],
"nqn": "nqn.2018-09.io.spdk:cnode13"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
    }
}
```



```
        "version": "1.3"
    },
    "namespaces": [
        {
            "device": {
                "path": "/dev/nvme13n1",
                "uuid": "6b9a65a3-6557-4713-8bad-57d9c5cb17a9"
            },
            "enable": 1,
            "nsid": 14
        }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode14"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
        {
            "device": {
                "path": "/dev/nvme14n1",
                "uuid": "ed69ba4d-8727-43bd-894a-7b08ade4f1b1"
            },
            "enable": 1,
            "nsid": 15
        }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode15"
},
{
    "allowed_hosts": [],
    "attr": {
        "allow_any_host": "1",
        "version": "1.3"
    },
    "namespaces": [
        {
            "device": {
                "path": "/dev/nvme15n1",
                "uuid": "5b8e9af4-0ab4-47fb-968f-b13e4b607f4e"
            },
            "enable": 1,
            "nsid": 16
        }
    ],
    "nqn": "nqn.2018-09.io.spdk:cnode16"
}
]
```

**DISCLAIMERS**

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein.

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 information go to <http://www.intel.com/performance>

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.

Intel® AES-NI requires a computer system with an AES-NI enabled processor, as well as non-Intel software to execute the instructions in the correct sequence. AES-NI is available on select Intel® processors. For availability, consult your reseller or system manufacturer. **For more information, see <http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni/>**

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

§