



# SPDK NVMe-oF (Target & Initiator) Performance Report Release 19.01.1

---

**Testing Date:** March 2019

**Performed by:** Karol Latecki ([karol.latecki@intel.com](mailto:karol.latecki@intel.com))

**Acknowledgments:**

Vishal Verma ([vishal4.verma@intel.com](mailto:vishal4.verma@intel.com))

John Kariuki ([john.k.kariuki@intel.com](mailto:john.k.kariuki@intel.com))

James R Harris ([james.r.harris@intel.com](mailto:james.r.harris@intel.com))



Revision History

Date	Revision	Comment
19/03/22	v. 1.0	Complete performance runs
19/03/22	v. 1.0	Review
19/03/26	v. 1.1	Reviews incorporated



# Contents

---

Contents .....	3
Audience and Purpose.....	4
Test setup .....	5
Target Configuration.....	5
Initiator 1 Configuration .....	6
Initiator 2 Configuration .....	6
BIOS settings .....	6
Kernel & BIOS spectre-meltdown information .....	7
Introduction to SPDK NVMe-oF (Target & Initiator) .....	8
Test Case 1: SPDK NVMe-oF Target I/O core scaling .....	10
4k Random Read results .....	13
4k Random Write results .....	13
4k Random Read-Write results.....	14
Large I/O Block Size sequential performance .....	16
Test Case 2: SPDK NVMe-oF Initiator I/O core scaling.....	17
4k Random Read results .....	19
4k Random Write results .....	19
4k Random Read-Write results.....	20
Test Case 3: Linux Kernel vs. SPDK NVMe-oF Latency.....	22
SPDK vs Kernel Target results.....	24
SPDK vs Kernel Initiator results .....	25
Test Case 4: NVMe-oF Performance with increasing # of connections .....	27
4k Random Read.....	29
4k Random Write results .....	30
4k Random Read-Write results.....	31
Summary .....	33
Appendix A.....	34



## ***Audience and Purpose***

---

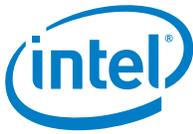
This report is intended for people who are interested in evaluating SPDK NVMe-oF (Target & Initiator) performance as compared to the Linux Kernel NVMe-oF (Target & Initiator). This report contains performance and efficiency information between SPDK NVMe-oF (Target & Initiator) vs. Linux Kernel NVMe-oF (Target & Initiator) on a set of underlying block devices under various test cases.

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	<p>SuperMicro SYS-2029U-TN24R4T</p>  
CPU	<p><a href="#">Intel® Xeon® Platinum 8180 Processor (38.5MB L3, 2.50 GHz)</a> Number of cores 28, number of threads 56</p>
Memory	Total 376 GBs
Operating System	Fedora 28
BIOS	2.1a 08/23/2018
Linux kernel version	4.20.6-100.fc28
SPDK version	SPDK 19.01 (e3e35a85ea)
Storage	<p><b>OS:</b> 1x 120GB Intel SSDSC2BB120G4 <b>Storage Target:</b> 16x <a href="#">Intel® P4600™ P4600x 2.0TB</a> (FW: QDV10150) (8 on each CPU socket)</p>
NIC	<p>2x 100GbE Mellanox ConnectX-5 NICs. Both ports connected. 1 NIC per CPU socket.</p>



## Initiator 1 Configuration

Item	Description
Server Platform	SuperMicro SYS-2028U TN24R4T+
CPU	<a href="#">Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz (55MB Cache, 2.20 GHz)</a> Number of cores 22, number of threads 44 per socket (Both sockets populated)
Memory	Total 64GBs
Operating System	Fedora 28
BIOS	3.1 06/08/2018
Linux kernel version	4.20.6-100.fc28
SPDK version	SPDK 19.01 (e3e35a85ea)
Storage	<b>OS:</b> 1x 240GB INTEL SSDSC2BB240G6
NIC	1x 100GbE Mellanox ConnectX-4 NIC. Both ports connected to Target server. (connected to CPU socket 0)

## Initiator 2 Configuration

Item	Description
Server Platform	SuperMicro SYS-2028U TN24R4T+
CPU	<a href="#">Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz (55MB Cache, 2.20 GHz)</a> Number of cores 22, number of threads 44 per socket (Only 1 CPU socket populated)
Memory	Total 64GBs
Operating System	Fedora 28
BIOS	3.1 06/08/2018
Linux kernel version	4.20.6-100.fc28
SPDK version	SPDK 19.01 (e3e35a85ea)
Storage	<b>OS:</b> 1x 240GB INTEL SSDSC2BB240G6
NIC	1x 100GbE Mellanox ConnectX-4 NIC. Both ports connected to Target server. (connected to CPU socket 0)

## BIOS settings

Item	Description
BIOS (Applied to all 3 systems)	Hyper threading Enabled CPU Power and Performance Policy <Performance> CPU C-state No Limit CPU P-state Enabled Enhanced Intel® Speedstep® Tech Enabled Turbo Boost Enabled



## **Kernel & BIOS spectre-meltdown information**

All three server systems use Fedora 4.20.6 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 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 RDMA connections to CPU cores to avoid synchronization and cache thrashing so that the data for those connections is kept as close to the CPU cache as possible.

The SPDK NVMe-oF target and initiator uses the Infiniband/RDMA verbs API to access an RDMA-capable NIC. These should work on all flavors of RDMA transports, but are 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 individual 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 P4600 SSDs which were used as block devices for NVMe-oF subsystems and two 100GbE Mellanox ConnectX-4 NICs connected to provide up to 200GbE of network bandwidth. Each Initiator system has one Mellanox ConnectX-4 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 are reported for 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 using fio and CPU utilization was collected using sar (systat).

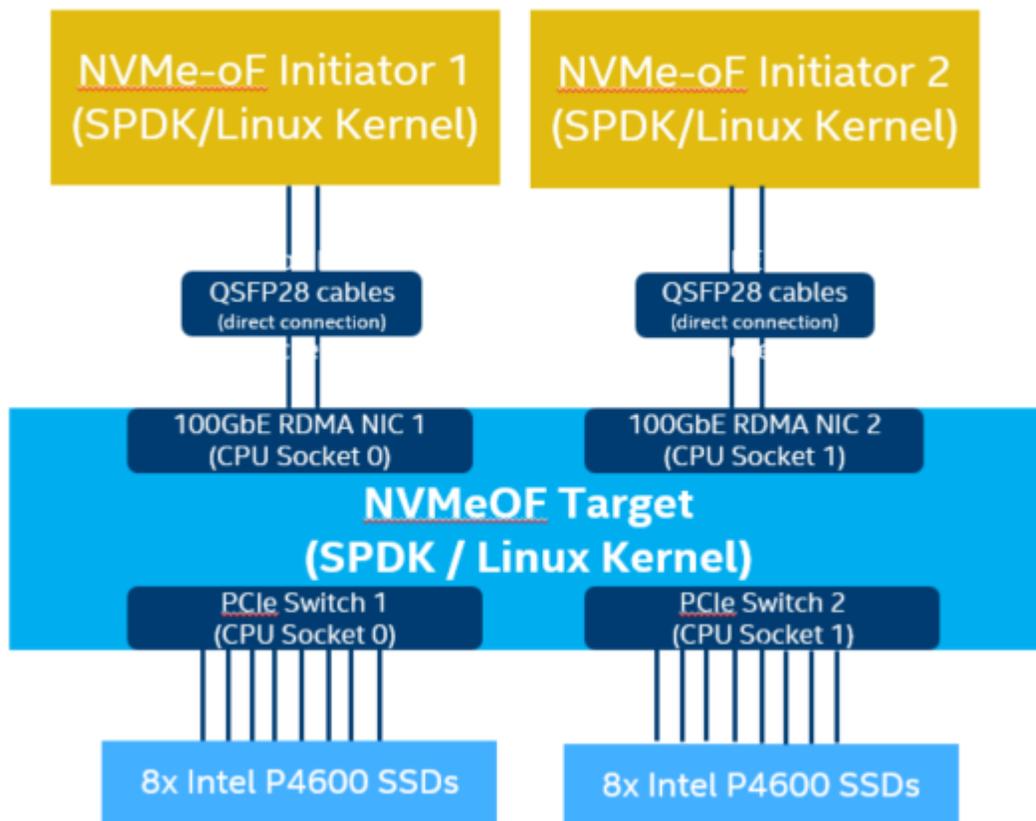


Figure 1: High level NVMe-oF performance testing setup

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

This test case was performed in order to understand the performance of SPDK NVMe-oF target with I/O core scaling. SPDK NVMe-oF target was configured to run with 16 NVMe-oF subsystems. Each NVMe-oF subsystem ran on top of an individual bdev backed by a single Intel P4600 device. Each of the 2 initiators were connected to 8 individual NVMe-oF subsystems which were exposed via SPDK NVMe-oF Target over 1x 100GbE NIC. SPDK bdev FIO plugin was used to target 8 individual NVMe-oF bdevs on each of the initiators. SPDK Target ReactorMask was configured to use 1, 2, 3 and 4 cores tests while running following workloads on each initiator:

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

For detailed configuration please refer to table below. Actual configuration of SPDK NVMeOF Target in test was done using JSON-RPC and the table contains a sequence of commands used by `spdk/scripts/rpc.py` script rather than a configuration file. SPDK NVMeOF Initiator (bdev fio\_plugin) still uses plain configuration files.

Each CPU – RW mode – Queue depth combination was run three times and reported end results are an average of these 3 runs. For write modes which need preconditioning (randwrite, randrw) the selected approach was to run preconditioning once before running all of the workload combinations for this RW mode. This allowed to artificially increase the number of IOPS and better saturate the network.

Item	Description
Test Case	Test SPDK NVMe-oF Target I/O core scaling
SPDK NVMeOF Target configuration	<pre> rpc.py construct_nvme_bdev -t PCIe -b Nvme0 -a 0000:60:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme1 -a 0000:61:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme2 -a 0000:62:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme3 -a 0000:63:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme4 -a 0000:64:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme5 -a 0000:65:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme6 -a 0000:66:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme7 -a 0000:67:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme8 -a 0000:b5:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme9 -a 0000:b6:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme10 -a 0000:b7:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme11 -a 0000:b8:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme12 -a 0000:b9:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme13 -a 0000:ba:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme14 -a 0000:bb:00.0 rpc.py construct_nvme_bdev -t PCIe -b Nvme15 -a 0000:bc:00.0  rpc.py nvme_create_transport -t RDMA (creates RDMA transport layer with default values: </pre>



<pre>trtype: "RDMA" 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)  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  nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode1 -t rdma -f ipv4 -s 4420 -a 20.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode2 -t rdma -f ipv4 -s 4420 -a 20.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode3 -t rdma -f ipv4 -s 4420 -a 20.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode4 -t rdma -f ipv4 -s 4420 -a 20.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode5 -t rdma -f ipv4 -s 4420 -a 20.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode6 -t rdma -f ipv4 -s 4420 -a 20.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode7 -t rdma -f ipv4 -s 4420 -a 20.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode8 -t rdma -f ipv4 -s 4420 -a 20.0.1.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode9 -t rdma -f ipv4 -s 4420 -a 10.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode10 -t rdma -f ipv4 -s 4420 -a 10.0.0.1 nvmf_subsystem_add_listener nqn.2018-09.io.spdk:cnode11 -t rdma -f ipv4 -s 4420 -a 10.0.0.1</pre>
---



	<pre>nvme_subsystem_add_listener nqn.2018-09.io.spdk:cnode12 -t rdma -f ipv4 -s 4420 -a 10.0.0.1 nvme_subsystem_add_listener nqn.2018-09.io.spdk:cnode13 -t rdma -f ipv4 -s 4420 -a 10.0.1.1 nvme_subsystem_add_listener nqn.2018-09.io.spdk:cnode14 -t rdma -f ipv4 -s 4420 -a 10.0.1.1 nvme_subsystem_add_listener nqn.2018-09.io.spdk:cnode15 -t rdma -f ipv4 -s 4420 -a 10.0.1.1 nvme_subsystem_add_listener nqn.2018-09.io.spdk:cnode16 -t rdma -f ipv4 -s 4420 -a 10.0.1.1</pre>
SPDK NVMeOF Initiator - FIO plugin configuration	<pre><b>BDEV.conf</b> [Nvme] TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode1" Nvme0 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode2" Nvme1 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode3" Nvme2 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode4" Nvme3 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode5" Nvme4 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode6" Nvme5 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode7" Nvme6 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode8" Nvme7  <b>FIO.conf</b> [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} time_based=1 ramp_time=60 runtime=300  [filename0] filename=Nvme0n1 [filename1] filename=Nvme1n1 [filename2] filename=Nvme2n1 [filename3] filename=Nvme3n1 [filename4] filename=Nvme4n1 [filename5] filename=Nvme5n1 [filename6] filename=Nvme6n1 [filename7] filename=Nvme7n1</pre>



## 4k Random Read results

Test Result: 4K 100% Random Read IOPS QD=32

# of Cores	Bandwidth (Mbps)	Throughput (IOPS k)	Avg. Latency (usec)
1 core	4851.77	1242.1	411.9
2 cores	10136.65	2595.0	197.5
3 cores	14306.15	3662.4	139.4
4 cores	16296.15	4171.8	122.5

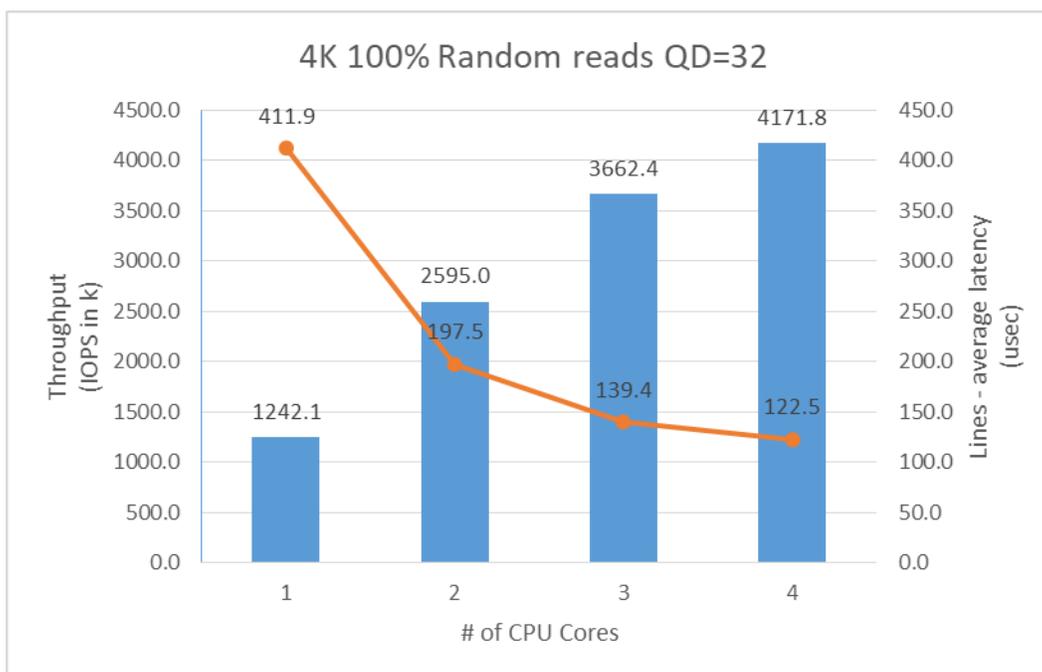


Figure 2: SPDK NVMe-oF Target I/O core scaling: IOPS vs. Latency while running 4KB 100% Random read workload

## 4k Random Write results

Test Result: 4K 100% Random Writes IOPS QD=32

# of Cores	Bandwidth (Mbps)	Throughput (IOPS in k)	Avg. Latency (usec)
1 core	6148.64	1574.0	324.1
2 cores	12674.51	3244.7	156.0
3 cores	18907.93	4840.4	104.1
4 cores	22560.17	5775.4	87.4

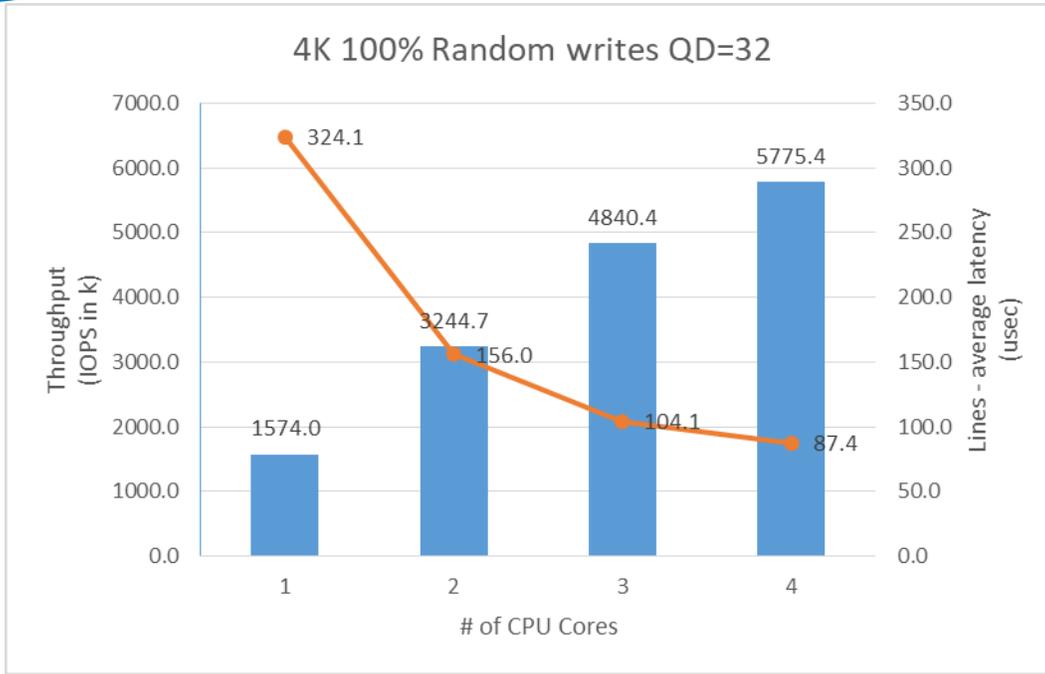


Figure 3: SPDK NVMe-oF Target I/O core scaling: IOPS vs. Latency while running 4KB 100% Random write workload

## 4k Random Read-Write results

Test Result: 4K 70% Read 30% Write IOPS, QD=32

# of Cores	Bandwidht (Mbps)	Throughput (IOPS in k)	Avg. Latency (usec)
1 core	5072.15	1298.5	393.8
2 cores	9866.31	2525.8	202.1
3 cores	10175.64	2605.0	196.2
4 cores	10400.38	2662.5	192.1

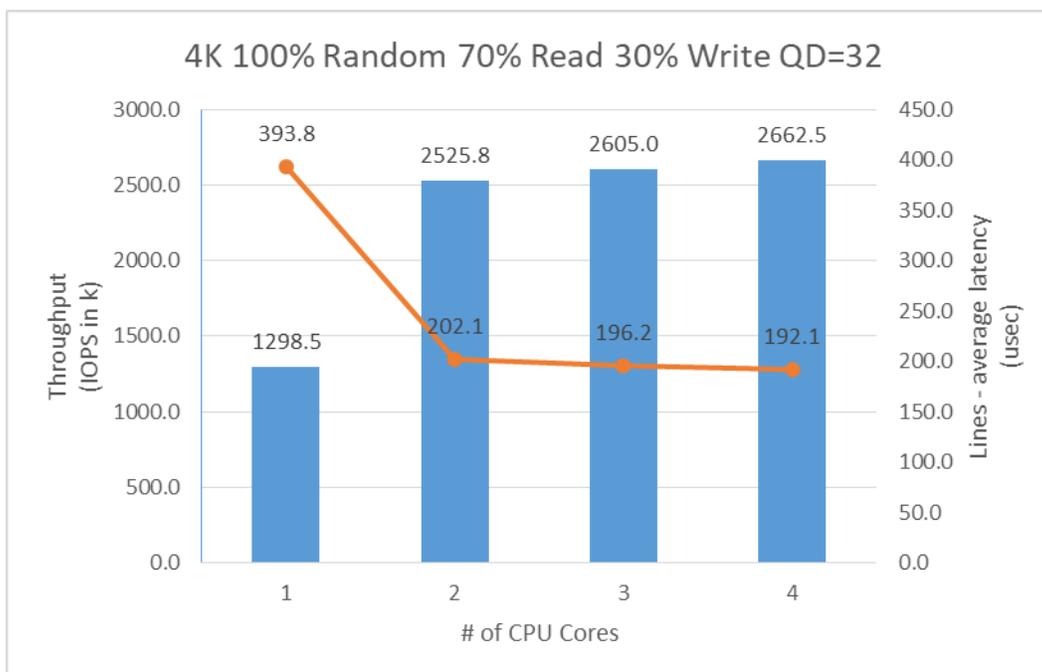


Figure 4: SPDK NVMe-of Target I/O core scaling: IOPS vs. Latency while running 4KB Random 70% read 30% write workload

**Conclusion:**

1. Scalability is best visible using a workload with 100% Random Writes where IOPS throughput and latency change almost linearly. 100% Random Reads workload is close to this, but there is a slight drop in IOPS scaling when increasing the number of CPUs from 3 to 4.
2. For workload consisting of 4K Random 70% Reads & 30% Writes performance doesn't scale from 2 to 3 and 4 cores due to some other bottleneck. It was observed that while running this test case locally without involving any network it can hit > 4M aggregate IOPS. But, while running this over the network, it could only achieve about 2.6M IOPS max. This points to some other platform or configuration bottleneck while running this workload.

## Large I/O Block Size sequential performance

128K block size I/O size tests were performed with 128K block size, 100% sequential, iodepth = 1, 8, 16 and 32 to each NVMe-oF subsystem. Rest of the FIO configuration is similar to 4K case mentioned earlier. Presented below are results using iodepth=8. Higher queue depth resulted in negligible bandwidth gain and big increase in latency.

### Test Result: 128K 100% Sequential Reads QD=8

# of Cores	Bandwidth (MBPS)	IOPS in K	Avg. Latency (usec)
1 core	24962.03	199.7	640.8
2 cores	24947.14	199.6	641.1
3 cores	24928.21	199.4	641.6
4 cores	24910.79	199.3	642.1

### Test Result: 128K 100% Sequential Writes QD=8

# of Cores	Bandwidth (MBPS)	IOPS in K	Avg. Latency (usec)
1 core	23516.53	188.1	680.1
2 cores	23606.75	188.9	677.5
3 cores	23580.76	188.6	678.2
4 cores	23543.86	188.4	679.3

### Test Result: 128K 70% Reads 30% Writes QD=8

# of Cores	Bandwidth (MBPS)	IOPS in K	Avg. Latency (usec)
1 core	29842.07	238.7	623.8
2 cores	29857.59	238.9	620.3
3 cores	29087.92	232.7	625.0
4 cores	30261.42	242.1	623.6

### Conclusion:

1. A single CPU core saturated the network bandwidth. The SPDK NVMe-oF target running on 1 core does close to 24 GBps 100% Writes and to 25 GBps of 100% Reads. 70-30 Read-Write workload generated bandwidth is around 29-30GBps which is very close to 2x 100GbE NICs network bandwidth. Therefore, adding more CPU cores did not result in increased performance for these workloads because the network was the bottleneck.



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

This test case was performed in order to understand the performance of SPDK NVMe-oF Initiator with I/O core scaling. SPDK NVMe-oF Target was configured similar to test case 1, running using 4 cores. SPDK bdev FIO plugin was used to target 8 individual –NVMe-oF bdevs on each of the 2 initiators. FIO cpumask was varied in order to run 1, 2, 3 and 4 cores tests while running following workloads from both the initiators using fio client server method.

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

**1 core:** 1 initiator with single core for FIO was used.

**2 cores:** 2 separate initiators, both running single core each using FIO.

**3 cores:** 2 separate initiators, 1 with 1 core and other with 2 cores using FIO.

**4 cores:** 2 separate initiators, both running 2 core each using FIO.

Item	Description
Test Case	Test SPDK NVMe-oF Target I/O core scaling
SPDK NVMeOF Target configuration	Same as in Test Case #1, using 4 CPU cores.
SPDK NVMeOF Initiator 1 - FIO plugin configuration	<p><b>BDEV.conf</b> [Nvme] TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode1" Nvme0 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode2" Nvme1 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode3" Nvme2 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode4" Nvme3 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode5" Nvme4 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode6" Nvme5 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode7" Nvme6 TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.1.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode8" Nvme7</p> <p><b>FIO.conf</b> [global] ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin spdk_conf=/tmp/spdk/bdev.conf thread=1 group_reporting=1 direct=1</p> <p>norandommap=1 rw=randrw rwmixread={100, 70, 0}</p>

	<pre> bs=4k iodepth={32, 64, 128, 256} time_based=1 ramp_time=60 runtime=300  {filename_section}  <b>FIO.conf filename section for 1 &amp; 2 CPU test run</b> [filename0] filename=Nvme0n1 filename=Nvme1n1 filename=Nvme2n1 filename=Nvme3n1 filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1  <b>FIO.conf filename section for 3 &amp; 4 CPU test run</b> [filename0] filename=Nvme0n1 filename=Nvme1n1 filename=Nvme2n1 filename=Nvme3n1 [filename1] filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1 </pre>
<p>SPDK NVMeOF Initiator 2 - FIO plugin configuration</p>	<pre> <b>BDEV.conf</b> Similar as Initiator 1. Different IP / Port numbers.  <b>FIO.conf</b> Similar as Initiator 1. Different filename section settings (below).  <b>FIO.conf filename section for 1</b> N/A (only Initiator 1 is used)  <b>FIO.conf filename section for 2 &amp; 3 CPU test run</b> [filename0] filename=Nvme0n1 filename=Nvme1n1 filename=Nvme2n1 filename=Nvme3n1 filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1  <b>FIO.conf filename section for 4 CPU test run</b> [filename0] filename=Nvme0n1 filename=Nvme1n1 filename=Nvme2n1 </pre>



filename=Nvme3n1 [filename1] filename=Nvme4n1 filename=Nvme5n1 filename=Nvme6n1 filename=Nvme7n1
---

## 4k Random Read results

Results in the table represent aggregate performance (IOPS & Avg. latency) observed:

Test Result: 4K 100% Random Read, QD=256

# of Cores	Bandwidth (MBps)	Throughput (IOPS)	Avg. Latency (usec)
1 core	4968.21	1271.9	195.8
2 cores	10947.22	2802.5	175.1
3 cores	16797.94	4300.3	171.6
4 cores	19080.54	4884.6	210.0

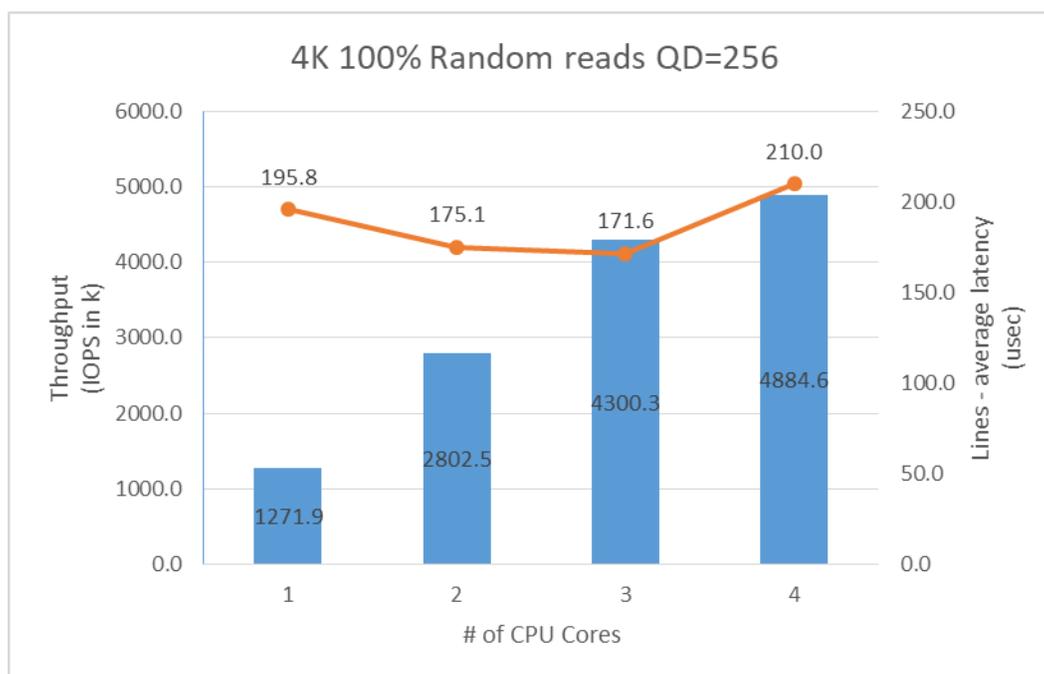


Figure 5: SPDK NVMe-oF Initiator I/O core scaling: IOPS vs. Latency while running 4KB 100% Random read workload

## 4k Random Write results

**Note:** Drives were pre-conditioned just once before running the 100% Random write I/O test combinations. This helped scale throughput and thereby avoiding storage bottleneck especially when testing 3 and 4 cores case until hitting 2x 100GbE bottleneck.

Test Result: 4K 100% Random Write, QD=256

# of Cores	Bandwidth (MBps)	Throughput (IOPS)	Avg. Latency (usec)
1 core	4826.91	1235.7	197.4
2 cores	10407.71	2664.4	174.7
3 cores	15685.55	4015.5	168.5
4 cores	20242.84	5182.2	170.6

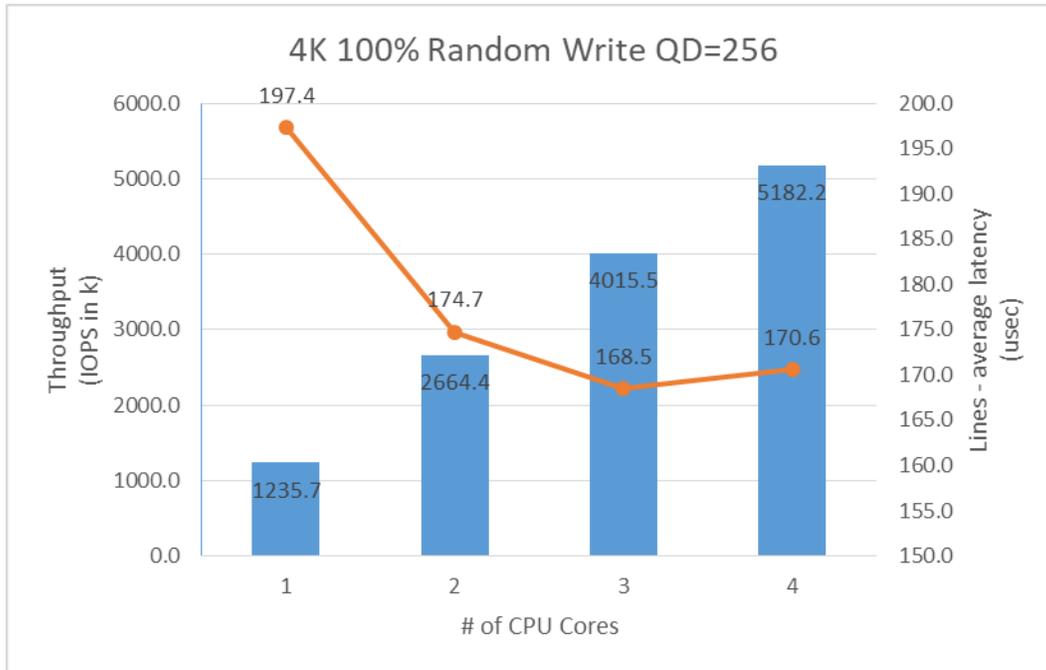


Figure 6: SPDK NVMe-of Initiator I/O core scaling: IOPS vs. Latency while running 4KB 100% Random write workload

## 4k Random Read-Write results

Test Result: 4K 70% Random Read 30% Random Write QD=256

# of Cores	Bandwidth (MBps)	Throughput (IOPS)	Avg. Latency (usec)
1 core	4570.25	1170.0	161.4
2 cores	9257.49	2369.9	114.2
3 cores	11989.00	3069.2	119.5
4 cores	14385.84	3682.8	134.4

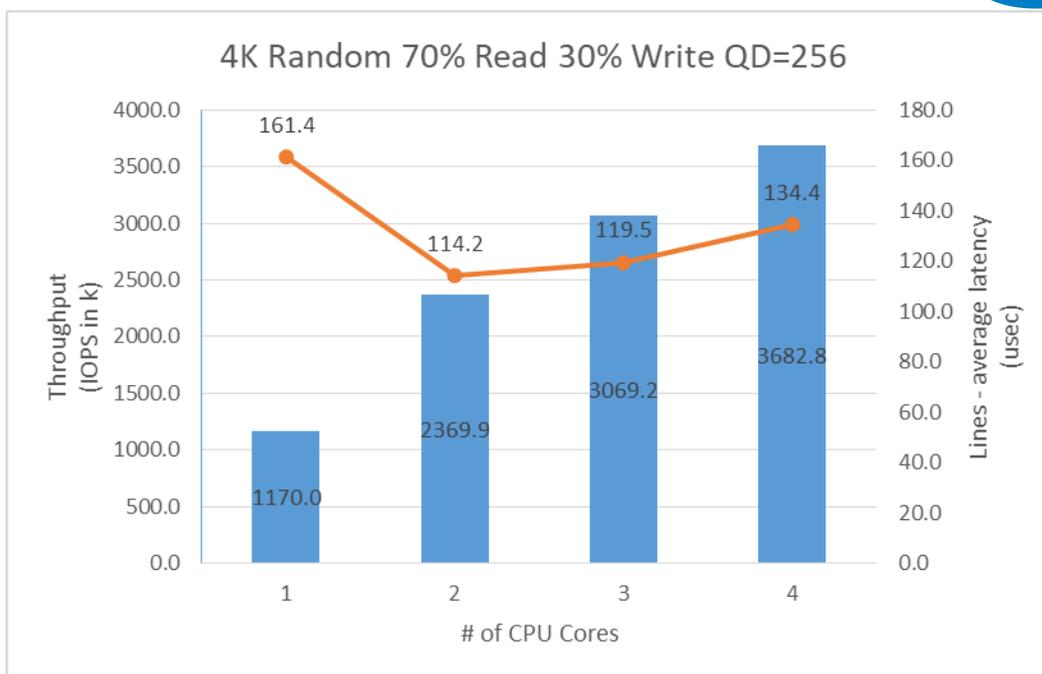


Figure 7: SPDK NVMe-oF Initiator I/O core scaling: IOPS vs. Latency while running 4KB Random 70% read 30% write workload

**Conclusion:**

1. In case of 100% Random Writes IOPS scale linearly with each CPU initiator added and at 4 CPUs is close to saturating the network bandwidth.
2. IOPS in 100% Random Read test scale linearly for 1, 2 and 3 CPUs. For 4 CPUs IOPS and bandwidth gain is halved, which might suggest some kind of bottleneck, as there is still some bandwidth available on NICs.
3. Average latencies for all workloads remain rather constant which suggests that SPDK Target on the other side of connection could handle more traffic generated from initiator side.

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

This test case was designed to understand latency capabilities of SPDK NVMe-oF Target and Initiator vs. corresponding Linux Kernel NVMe-oF Target and Initiator implementations on a single NVMe-oF subsystem. Average I/O latency and p99 was compared between SPDK NVMe-oF (Target/Initiator) vs. Linux Kernel (Target/Initiator). SPDK NVMe-oF Target was configured to run on a single NVMe-oF subsystem with a single NVMe bdev on top of **null block device**, running on a single core. Null block device (bdev) was chosen as the backend block device to eliminate media latency during these tests.

Item	Description
Test Case	NVMe-oF Target Latency
<b>Test configuration</b>	
SPDK NVMeOF Target configuration	<pre>rpc.py nvmf_create_transport -t RDMA (creates RDMA transport layer with default values: trtype: "RDMA" 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)  rpc.py construct_null_bdev Nvme0n1 10240 4096  nvme_subsystem_create nqn.2018-09.io.spdk:cnode1 -s SPDK001 -a -m 8  nvme_subsystem_add_ns nqn.2018-09.io.spdk:cnode1 Nvme0n1  nvme_subsystem_add_listener nqn.2018-09.io.spdk:cnode1 -t rdma -f ipv4 -s 4420 -a 20.0.0.1</pre>
Kernel NVMeOF Target configuration	<pre>Target configuration file loaded using nvmet-cli tool. {   "ports": [     {       "addr": {         "adrfam": "ipv4",         "traddr": "20.0.0.1",         "trsvcid": "4420",         "trtype": "rdma"       },       "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>
<b>FIO configuration</b>	
<p>SPDK NVMeOF Initiator FIO plugin configuration</p>	<p><b>BDEV.conf</b> [Nvme] TransportId "trtype:RDMA adrfam:IPv4 traddr:20.0.0.1 trsvcid:4420 subnqn:nqn.2018-09.io.spdk:cnode1" Nvme0</p> <p><b>FIO.conf</b> [global] ioengine=/tmp/spdk/examples/bdev/fio_plugin/fio_plugin spdk_conf=/tmp/spdk/bdev.conf thread=1 group_reporting=1 direct=1</p> <p>norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth=1 time_based=1 ramp_time=60 runtime=300</p> <p>[filename0] filename=Nvme0n1</p>
<p>Kernel initiator configuration</p>	<p><b>Device config</b> Done using nvme-cli tool. sudo modprobe nvme-fabrics sudo nvme connect -n nqn.2018-09.io.spdk:cnode1 -t rdma -a 20.0.0.1 -s 4420</p> <p><b>FIO.conf</b> [global]</p>

<pre> ioengine=libaio 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=/dev/nvme0n1 </pre>
---

## SPDK vs Kernel Target results

This following data was collected using Kernel initiator against both SPDK & Kernel Nvme-oF target.

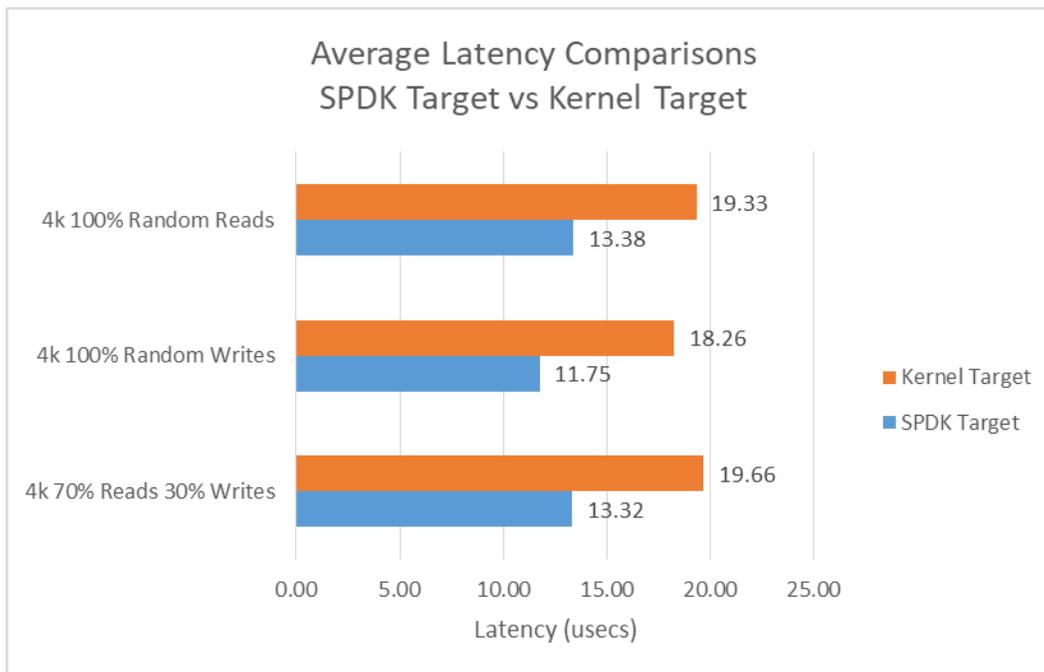


Figure 8: Average I/O Latency Comparisons b/w SPDK vs. Kernel NVMe-oF Target for various workloads (Kernel Initiator)

### SPDK NVMe-oF Target

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4K 100% Random Reads IOPS	13.32	73122	24.0
4K 100% Random Writes IOPS	11.75	82469	21.0
4K 100% Random 70% Reads 30% Writes IOPS	13.38	73051	24.4



### Linux Kernel NVMe-oF Target

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4K 100% Random Reads IOPS	19.66	49485	29.0
4K 100% Random Writes IOPS	18.26	53176	24.9
4K 100% Random 70% Reads 30% Writes IOPS	19.33	50355	24.8

#### Conclusion:

1. SPDK NVMe-oF Target average round trip I/O latency (reads/writes) is up to 7 usec faster than the Linux kernel NVMe-oF target. This is entirely software overhead.
2. Kernel NVMe-oF Target performance has changed since the last report 18.04: IOPS for all workloads have improved and latency has dropped significantly, up to 9 usec.

## SPDK vs Kernel Initiator results

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

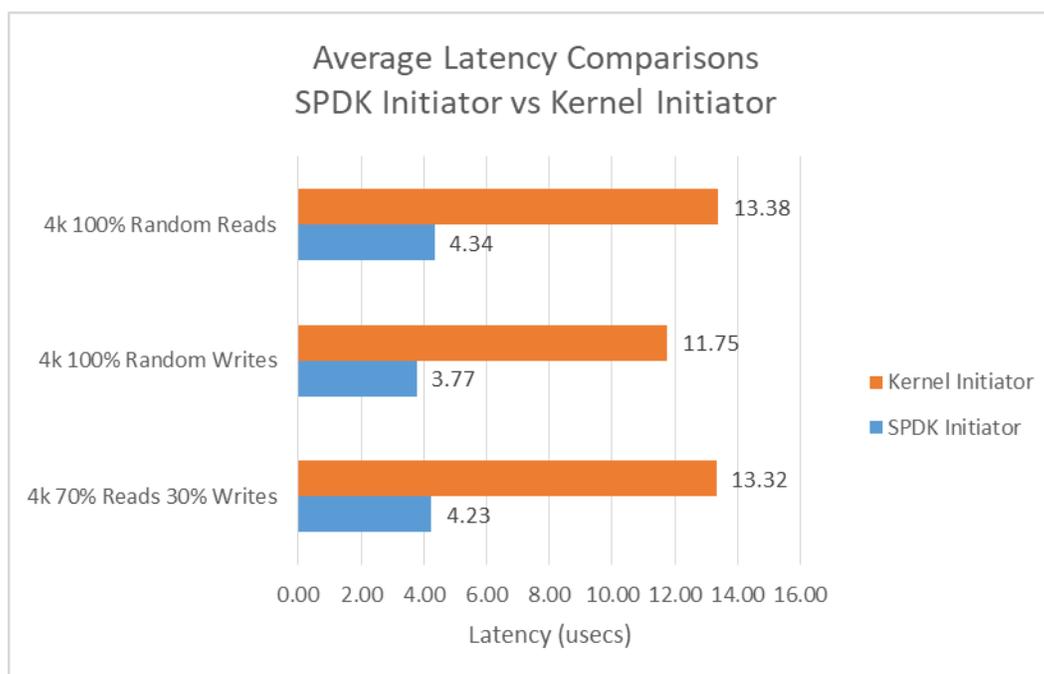
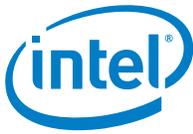


Figure 9: Average I/O Latency Comparisons b/w SPDK vs. Kernel NVMe-oF Initiator for various workloads (SPDK Target)



### Linux Kernel NVMe-oF Initiator

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4K 100% Random Reads IOPS	13.32	73122	24.0
4K 100% Random Writes IOPS	11.75	82469	22.7
4K 100% Random 70% Reads 30% Writes IOPS	13.38	73051	24.4

### SPDK NVMe-oF Initiator

Access Pattern	Average Latency (usec)	IOPS	p99 (usec)
4K 100% Random Reads IOPS	4.23	226639	5.0
4K 100% Random Writes IOPS	3.77	253694	5.6
4K 100% Random 70% Reads 30% Writes IOPS	4.34	221712	4.8

### Conclusion:

1. SPDK NVMe-oF Initiator average round trip I/O latency (reads/writes) is up to 4x as fast as Kernel NVMe-oF Initiator using null bdev backend.



## 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. 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. Number of CPU cores metric was calculated from %CPU utilization measured using sar (systat package in linux). SPDK NVMe-oF Target was configured to run on 4 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.

- 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 NVMeOF Target configuration	Same as in Test Case #1, using 4 CPU cores.
Kernel NVMeOF Target configuration	Target configuration file loaded using nvmet-cli tool. For detail configuration file contents please see Appendix A.
Kernel NVMeOF Initiator #1	<p><b>Device config</b> Done using nvme-cli tool.</p> <pre>sudo modprobe nvme-fabrics sudo nvme connect -n nqn.2018-09.io.spdk:cnode1 -t rdma -a 20.0.0.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode2 -t rdma -a 20.0.0.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode3 -t rdma -a 20.0.0.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode4 -t rdma -a 20.0.0.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode5 -t rdma -a 20.0.1.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode6 -t rdma -a 20.0.1.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode7 -t rdma -a 20.0.1.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode8 -t rdma -a 20.0.1.1 -s 4420</pre>
Kernel NVMeOF Initiator #2	<p><b>Device config</b> Done using nvme-cli tool.</p> <pre>sudo modprobe nvme-fabrics sudo nvme connect -n nqn.2018-09.io.spdk:cnode9 -t rdma -a 10.0.0.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode10 -t rdma -a 10.0.0.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode11 -t rdma -a 10.0.0.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode12 -t rdma -a 10.0.0.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode13 -t rdma -a 10.0.1.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode14 -t rdma -a 10.0.1.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode15 -t rdma -a 10.0.1.1 -s 4420 sudo nvme connect -n nqn.2018-09.io.spdk:cnode16 -t rdma -a 10.0.1.1 -s 4420</pre>

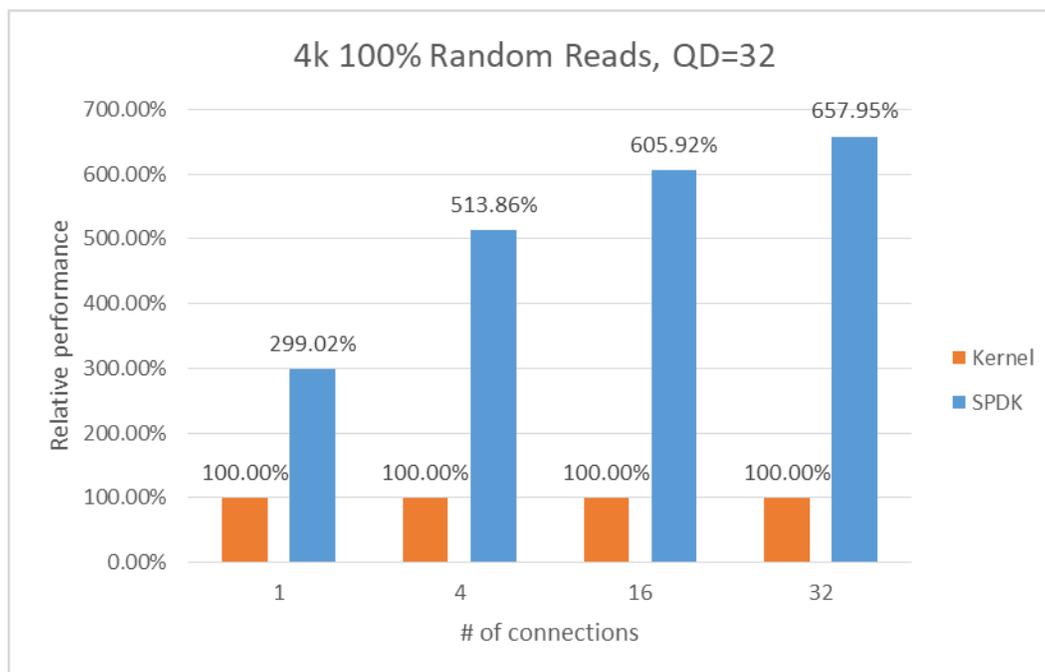
<p>FIO configuration (used on both initiators)</p>	<pre> <b>FIO.conf</b> [global] ioengine=libaio thread=1 group_reporting=1 direct=1  norandommap=1 rw=randrw rwmixread={100, 70, 0} bs=4k iodepth={8, 16, 32, 64, 128} time_based=1 ramp_time=60 runtime=300 numjobs={1, 4, 16, 32}  [filename1] filename=/dev/nvme0n1  [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>
--	---

Number of CPU cores used while running SPDK Nvme-oF target were 4, whereas for the case of linux Kernel Nvme-oF target there was no cpu core limitation applied.

Numbers in the graph represent relative performance which are in terms of 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.



## 4k Random Read



**Figure 10:** Relative Performance Comparison b/w Kernel vs. SPDK NVMe-oF Target for 4K 100% Random Reads (Kernel Initiator)

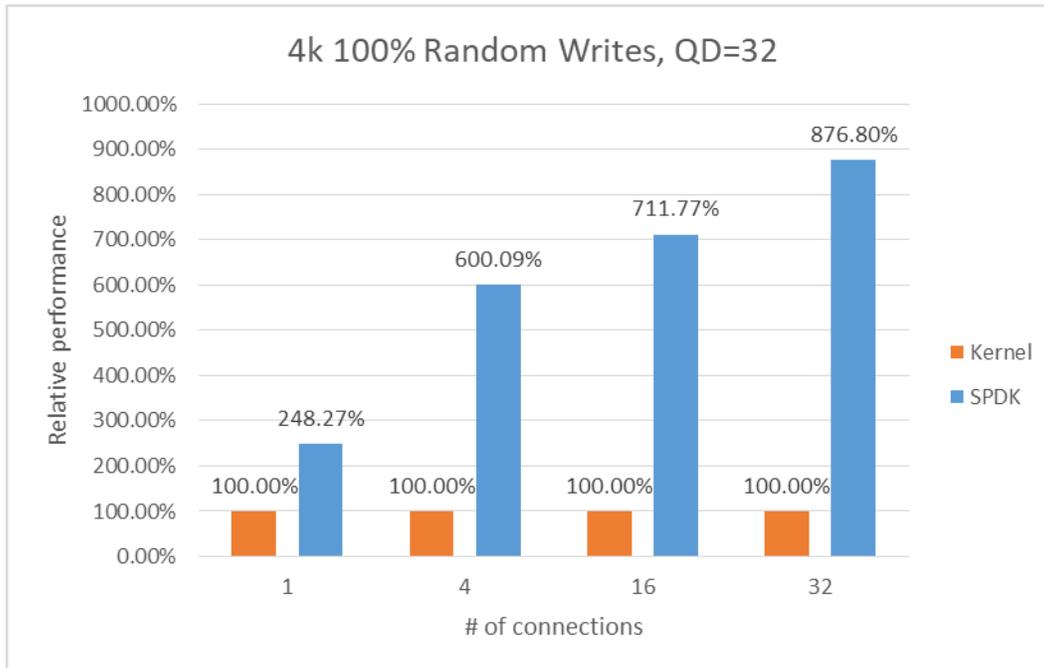
### Linux Kernel NVMe-oF Target: 4K 100% Random Reads, QD=32

# of Connections	Bandwidth (MBps)	IOPS (in thousands)	Average Latency (usec)	# of CPU cores utilized
1	13153.28	3367.2	151.9	12.0
4	14866.64	3805.9	537.7	19.8
16	13823.73	3538.8	2314.5	26.1
32	13287.02	3401.4	4816.2	30.4

### SPDK NVMe-oF Target: 4K 100% Random Reads, QD=32

# of Connections	Bandwidth (MBps)	IOPS (in thousands)	Average Latency (usec)	# of CPU cores utilized
1	13062.58	3344.0	153.0	4
4	15445.60	3954.1	517.8	4
16	12859.08	3291.9	2489.4	4
32	11512.27	2947.1	5579.9	4

## 4k Random Write results



**Figure 11:** Relative Performance Comparison b/w Kernel vs. SPDK NVMe-oF Target for 4K 100% Random Writes

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

### Linux Kernel NVMe-oF Target: 4K 100% Random Writes, QD=32

# of Connections	Bandwidth (MBps)	IOPS (in thousands)	Average Latency (usec)	# of CPU cores utilized
1	16219.94	4152.3	123.0	13.5
4	15008.11	3842.1	532.8	17.2
16	14271.00	3653.3	2241.9	23.4
32	14070.91	3602.1	4560.0	29.1

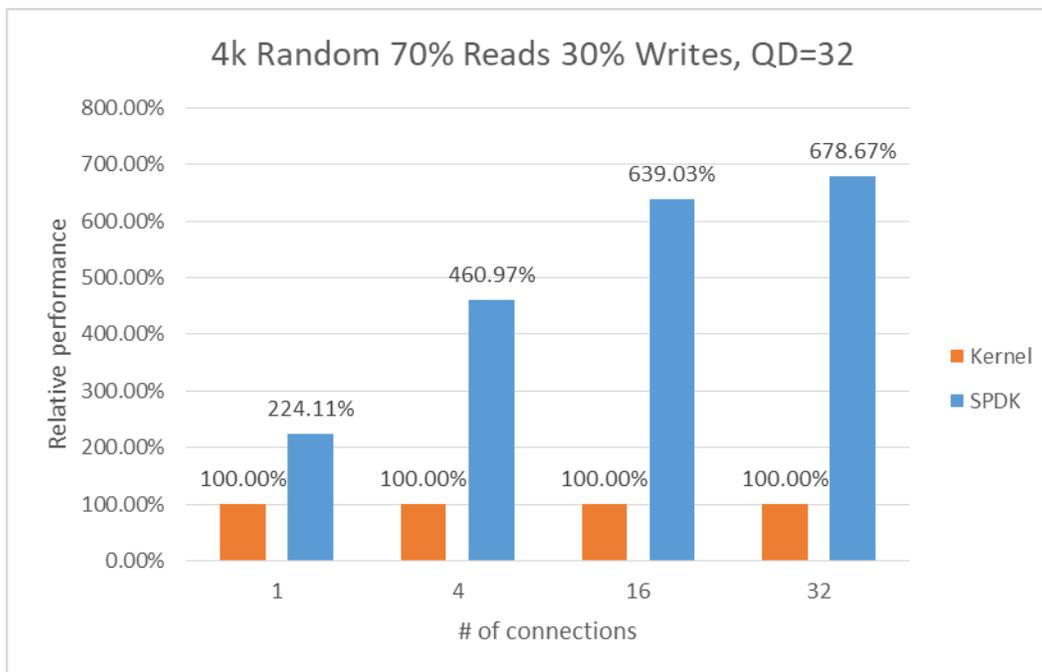
### SPDK NVMe-oF Target: 4K 100% Random Writes, QD=32

# of Connections	Bandwidth (MBps)	IOPS (in thousands)	Average Latency (usec)	# of CPU cores utilized
1	11934.90	3055.3	172.0	4
4	20922.56	5356.2	382.3	4
16	17382.37	4449.9	1843.0	4
32	16974.95	4345.5	3769.5	4



## 4k Random Read-Write results

For 4K Random 70% reads, 30% writes workload it was noticed that performance didn't increase when going from 4 to 16 connections per subsystem. This was due to storage/platform bottleneck noticed as described in test case 1 and 2.



**Figure 12:** Relative Performance Comparison b/w Kernel vs. SPDK NVMe-oF Target for 4K Random 70% Reads 30% Writes

### Linux Kernel NVMe-oF Target: 4K 70% Random Read 30% Random Write, QD=32

# of Connections	Bandwidth (MBps)	IOPS (in thousands)	Average Latency (usec)	# of CPU cores utilized
1	9333.25	2389.3	46.2	8.4
4	19229.42	4922.7	121.0	21.1
16	16913.31	4329.8	1474.2	32.2
32	16648.65	4262.0	3289.4	39.0

### SPDK NVMe-oF Target: 4K 70% Random Read 30% Random Write, QD=32

# of Connections	Bandwidth (MBps)	IOPS (in thousands)	Average Latency (usec)	# of CPU cores utilized
1	9909.67	2536.9	46.9	4
4	16768.54	4292.7	193.3	4
16	13405.55	3431.8	2132.7	4
32	11601.18	2969.8	5241.8	4



4K 70% Random Reads 30% Random Write performance was best when number of connections was 4 and iodepth was 4 per subsystem. This seem to be the optimum configuration while achieving max performance as was seen for 4K 100% Random Reads case as well.

**Conclusion:**

1. SPDK NVMe-oF target performs up to 6.5-7.5 times better w.r.t IOPS/core than Linux kernel NVMe-oF target while running 4K 100% Random Read workload with increasing number of connections (16-32) per NVMe-oF subsystem.
2. SPDK NVMe-oF performance is also better than Linux in case of 4k Random Write workload: up to 6-7.25 times better.
3. For 70-30 Random Read-Write workload general performance of SPDK in terms of throughput per CPU core is still much better. However there is a visible performance degradation when increasing the number of connections which may suggest similar bottleneck for this workload as in test cases 1 and 2. This degradation is not visible in case of Kernel Target and this may imply that bottleneck is located somewhere in SPDK.



## Summary

---

This report showcased performance results with SPDK NVMe-oF 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. Like in the last report, throughput scales up and latency decreases almost linearly with the scaling of SPDK NVMe-oF target cores. In case of SDPK NVMe-OF Initiator core scaling it was observed that throughput scales almost linearly, but the latency remains rather stable which might suggest that the Target side might be able to process slightly more IO.

In case of 4k Random 70% Read 30% Write workload there was an unidentified bottleneck, which prevents from confirming proper throughput and latency scaling for this workload.

It was also observed that in case of Kernel vs SPDK comparison, the SPDK NVMe-oF Target is up to 7 usec faster (~40%) than Kernel when testing against null bdev based backend. The advantage of SPDK is even greater when comparing NVMe-oF Initiators – SPDK is up to 4 times fast as Kernel Initiator.

SPDK NVMe-oF Target performed up to 6.5 times better w.r.t IOPS/core than Linux Kernel NVMe-oF target while running 4K 100% random read workload with increasing number of connections (16) 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 different hardware and software configurations.

## Appendix A

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

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



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



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



```
    "referrals": [],
    "subsystems": [
      "nqn.2018-09.io.spdk:cnode13"
    ]
  },
  {
    "addr": {
      "adrfam": "ipv4",
      "traddr": "10.0.1.1",
      "trsvcid": "4433",
      "trtype": "rdma"
    },
    "portid": 14,
    "referrals": [],
    "subsystems": [
      "nqn.2018-09.io.spdk:cnode14"
    ]
  },
  {
    "addr": {
      "adrfam": "ipv4",
      "traddr": "10.0.1.1",
      "trsvcid": "4434",
      "trtype": "rdma"
    },
    "portid": 15,
    "referrals": [],
    "subsystems": [
      "nqn.2018-09.io.spdk:cnode15"
    ]
  },
  {
    "addr": {
      "adrfam": "ipv4",
      "traddr": "10.0.1.1",
      "trsvcid": "4435",
      "trtype": "rdma"
    },
    "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"
},
]
```



```
"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® 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/>**

The benchmark results may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

Intel and the Intel logo are trademarks of Intel Corporation in the US and other countries

Copyright © 2019 Intel Corporation. All rights reserved.