100G Kernel and User Space NVMe/TCP Using Chelsio TOE

Key Take-aways

- Chelsio T6 100G NVMe/TOE solution delivers line-rate 99 Gbps throughput for both READ and WRITE.
- Isolation of CPU Performance from Wire Jitter.
- Significantly lower CPU utilization leads to significantly cheaper CPU requirements and BOM savings.
- Very low latency for local vs. remote SSD access.
- Roadmap to enabling integrated TLS encryption with existing silicon.

The NVMe over Fabrics (NVMe-oF) specification extends the benefits of NVMe to large fabrics beyond the reach and scalability of PCIe. NVMe/TCP is a technology which enables the use of NVMe-oF over existing standard datacenter IP networks. It provides the following advantages over other Fabric transports like RDMA (RoCE) and Fibre Channel:

- Robust and stable protocol
  - TCP/IP has been an IETF standard (RFC 793, 791) for over 35 years.
  - Inherent accuracy, reliability and scalability.
- Easy to use
  - Plug-and-Play.
  - Lower setup time.
  - No application changes.
- Cheaper to deploy
  - No additional switches/hardware required. Can use existing legacy infrastructure.
  - Enables a decoupled server and switch upgrade cycle and a brownfield strategy for datacenter deployments.
  - End user can purchase more compute servers for the same investment amount.
- Available in latest Linux kernels and Storage Performance Development Kit (SPDK).

SPDK, designed to extract maximum performance by moving all the necessary drivers to user space and polling hardware for completions instead of relying on interrupts and avoiding all locks in the I/O path, provides the benefits of high and scalable performance and low latency for user space storage applications like NVMe/TCP target.

Chelsio's TOE (TCP/IP Offload Engine) is fully capable of offloading TCP/IP processing of Kernel and User space SPDK NVMe/TCP target to hardware at 100Gbps. It provides a low latency, high throughput Ethernet solution for connecting high performance NVMe SSDs over a scalable, congestion controlled and traffic managed fabric. The unique ability of a *TOE* to perform the full transport layer functionality in hardware is essential to obtaining tangible benefits. The vital aspect of the transport layer is process-to-process communication, i.e. the data passed to the *TOE* comes straight from the application process, and the data delivered by the *TOE* goes straight to the application process.

This paper presents the significant performance benefits of the Chelsio T6 100GbE NVMe/TOE solution in both Kernel and User modes. Chelsio's T6 adapter delivers line-rate throughput as well as more than 2.9 Million IOPs at the 4K I/O size. In addition, with only 5.5 µs delta latency between remote and local storage access, Chelsio's solution proves to be the best-in-breed in providing the next generation, scalable storage network over standard and cost-effective Ethernet infrastructure with an efficient processing path.

Test Results

The following graph presents IOPs and throughput results of SPDK NVMe/TOE Target with SPDK Kernel NVMe/TCP (regular NIC) hosts using Null Block devices. The results are collected using **fio** tool with I/O size varying from 4 to 256 KBytes with an access pattern of random READs and WRITEs.

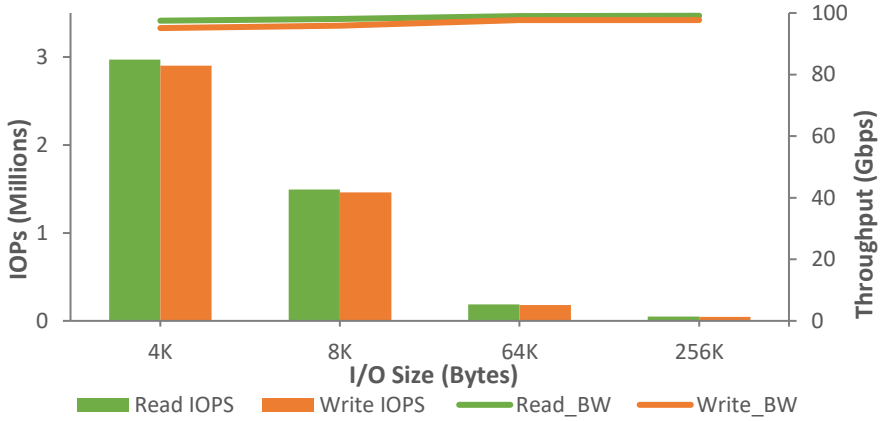


Figure 1 – SPDK NVMe/TOE Target IOPs and Throughput vs. I/O size

The following graph compares IOPs and throughput results of Kernel space NVMe/TOE Target & hosts with NVMe/TCP Target & hosts using Null Block devices. The results are collected using **fio** tool with I/O size varying from 4 to 512 KBytes with an access pattern of random READs and WRITEs.

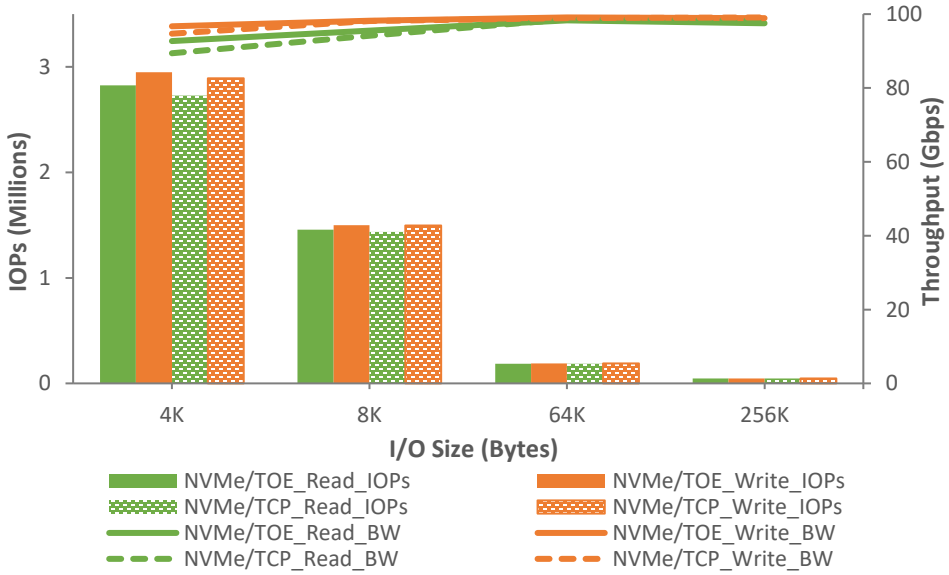


Figure 2 – NVMe/TOE, NVMe/TCP Target IOPs and Throughput vs. I/O size

As evident from the graphs above, T6 solution delivers line-rate READ and WRITE throughput for both Kernel and User space NVMe targets. With NVMe/TOE, READ and WRITE IOPs reach 2.9 Million at 4K I/O size.

The following graph compares the CPU consumption per Gbps of Kernel space NVMe/TOE and NVMe/TCP Targets for both READ and WRITE operations.

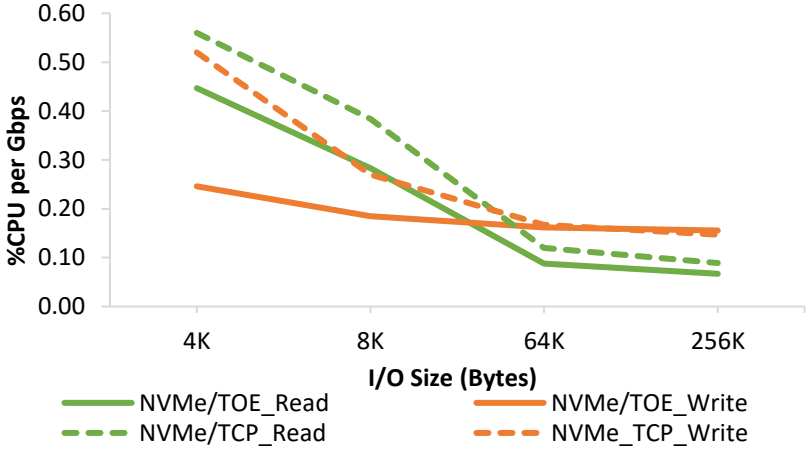


Figure 3 – NVMe/TOE, NVMe/TCP Target % CPU/Gbps vs. I/O size

The NVMe/TOE solution consumes significantly less CPU per Gbps (up to 50%) compared to NVMe/TCP. In fact, this is one of the most important benefits of a hardware offloaded TCP solution, resulting in a cheaper bill of materials. For example, in the testing in this paper, about 2 cores per socket can be saved for line rate performance using 4KB, when using TOE relative to software TCP. Using the pricing data in the link provided, this can map to several $100's of savings per socket (and a much lower power).

In addition, using a TOE isolates the host application's performance from the performance spikes caused by network traffic. No longer does the application have to get swapped out, so that the TCP stack can be brought back into the cache to retransmit or reorder a packet, thus making more efficient use of the expensive host CPU. A NVMe/TOE solution delivers a fully ordered, reliable data stream to the host. Chelsio's TOE solution is required to achieve 100 Gb/s and higher line-rate throughput with minimal CPU usage.

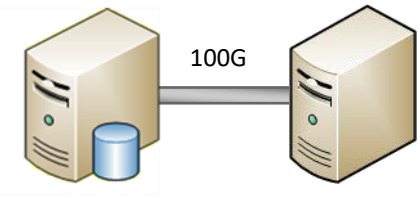
The following table presents the 4K I/O Random latency numbers of SPDK NVMe/TOE Target with SPDK Kernel NVMe/TCP (regular NIC) host and Kernel space NVMe/TOE Target & host using Micron NVMe SSDs.

	Read			Write		
	Local	**Remote**	**Delta**	**Local**	**Remote**	**Delta**
SPDK NVMe/TOE	109.16	114.88	5.72	24.69	30.2	5.51
Kernel NVMe/TOE	109.16	126.69	17.53	24.69	40.35	15.66

The remote versus local NVMe device access adds only 5.5 µs latency at 4K I/O with SPDK NVMe/TOE.

The Demonstration

- Supermicro X10DRG-Q Target with T62100-CR
- 2 Intel Xeon CPUs E5-2687W v4 12-core @ 3.00GHz (HT disabled)
- 128 GB RAM
- 1 Micron 9100 MAX 2.4TB PCIe NVMe SSD
- RHEL8.0 (5.4.45 kernel)



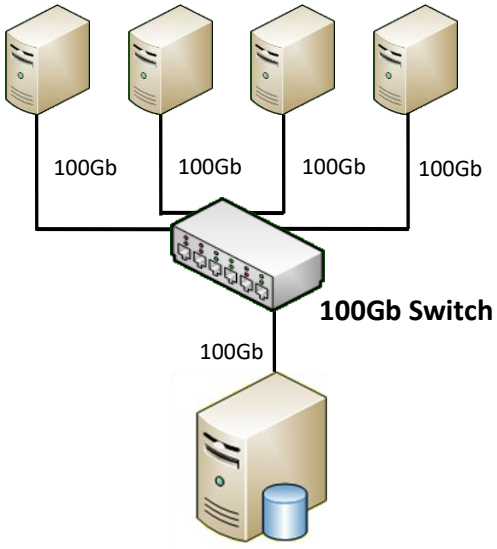
- Supermicro X10SRA-F host with T62100-CR
- 1 Intel Xeon CPU E5-1620 v4 4-core @ 3.50GHz (HT disabled)
- 32GB RAM
- RHEL 8.0 (5.4.45 kernel)

Figure 4 – Latency Test setup

The Latency test setup consists of a NVMe target machine connected to a single host back to back using a single port on each system.

The Bandwidth/IOPs test setup consists of a NVMe target machine connected to 4 host machines through a 100GbE switch using a single port on each system.

For both the tests, an MTU of 9000B is used on the ports under test. The latest Chelsio Unified Wire driver is installed on each machine.



- Supermicro X10SRA-F hosts with T62100-CR adapter
- 1 Intel Xeon CPU E5-1620 v4 4-core @ 3.50GHz (HT enabled)
- 32 GB RAM
- RHEL 8.0 (5.4.45 kernel)

- Supermicro X10DRG-Q Target with T62100-CR adapter
- 2 Intel Xeon CPUs E5-2687W v4 12-core @ 3.00GHz (HT enabled)
- 128 GB RAM
- RHEL 8.0 (5.4.45 kernel)

Figure 5 – Bandwidth/IOPs Test setup

Storage configuration

For Bandwidth/IOPs test, 8 NVMe targets are created on the target server using NULL BLOCK devices, each of 1GB size. The host connects to 2 targets using 4 connections each.

For latency test, the target is configured with 1 Micron 9100 MAX 2.4TB PCIe NVMe SSD. One host connects to the target using one connection.

Setup Configuration

General Configuration

i. Disable virtualization, c-state technology, VT-d, Intel I/O AT, SR-IOV in system BIOS.
ii. Enable Hyper-threading in system BIOS for Bandwidth/IOPs test.
iii. Compile and install the 5.4.45 kernel from Chelsio Unified Wire v3.13.0.1 package and reboot the machine into the newly installed kernel.

```
[root@host~]# cd ChelsioUwire-3.13.0.1
[root@host~]# make kernel_install
[root@host~]# reboot
```

iv. Install Chelsio drivers and tools.

```
[root@host~]# make install
```

v. Add the below parameters to grub kernel command line.
BW/IOPs test: intel_idle.max_cstate=0 processor.max_cstate=0 intel_pstate=disable
Latency test: idle=poll

vi. Set cpupower governor to performance

```
[root@host~]# cpupower frequency-set --governor performance
```

vii. Set the below tuned-adm profile for BW/IOPs test.

```
[root@host~]# tuned-adm profile network-throughput
```

Set the below tuned-adm profile for latency test.

```
[root@host~]# tuned-adm profile network-latency
```

viii. Set the below sysctl parameters.

```
sysctl -w net.ipv4.tcp_timestamps=0
sysctl -w net.core.netdev_max_backlog=250000
sysctl -w net.core.rmem_max=4194304
sysctl -w net.core.wmem_max=4194304
sysctl -w net.core.rmem_default=4194304
sysctl -w net.core.wmem_default=4194304
sysctl -w net.core.optmem_max=4194304
sysctl -w net.ipv4.tcp_rmem="4096 87380 4194304"
sysctl -w net.ipv4.tcp_wmem="4096 16384 4194304"
sysctl -w net.ipv4.tcp_low_latency=1
sysctl -w net.ipv4.tcp_adv_win_scale=1
```

ix. Copy the low latency firmware configuration file for latency test.

```
[root@host~]# cp ChelsioUwire-
3.13.0.1/src/network/firmware/low_latency_config/t6-config.txt
/lib/firmware/cxgb4/.
```

x. Precondition the NVMe SSD for latency test.

```
[root@host~]# msecli -N -f 1 -m 0 -g 512 -j 1 -n /dev/nvme0
[root@host~]#  for i in `seq 0 1`; do fio -name=SeqCond --readwrite=write --
bs=128k --ioengine=libaio --iodepth=64 --direct=1 --size=100% --thread --
filename=/dev/nvme0; done
[root@host~] # fio --rw=randwrite --name=random --ioengine=libaio --size=400m
--direct=1 --filename=/dev/nvme0 --time_based --runtime=4000 --iodepth=1 --
numjobs=1 --unit_base=1 --bs=4K --kb_base=1000
```

SPDK NVMe/TOE Configuration

Target

i. Load the Chelsio SPDK NVMe/TOE Offload driver and bring up interface with IPv4 address.

```
[root@host~]# modprobe chtcp
[root@host~]# ifconfig ethX <IP address> mtu 9000 up
```

ii. Configure Hugepages.

```
[root@host~]# cd ChelsioUwire-3.13.0.1/build/src/chspdk/user/spdk/
[root@host spdk]# HUGEMEM=32768 scripts/setup.sh
```

iii. Update the target configuration file, *etc/spdk/nvmf.conf.in*

BW/IOPs test:

```
[Global]
[Null]
Dev Null0 1024 4096
Dev Null1 1024 4096
Dev Null2 1024 4096
Dev Null3 1024 4096
Dev Null4 1024 4096
Dev Null5 1024 4096
Dev Null6 1024 4096
Dev Null7 1024 4096

[Nvmf]
 AcceptorPollRate 10000
 ConnectionScheduler RoundRobin

[Transport]
 Type TCP

[Nvme]

[Subsystem0 ]
 NQN nqn.2016-06.io.spdk:cnode0
 Listen TCP 10.1.1.149:4420
 AllowAnyHost Yes
 Host nqn.2016-06.io.spdk:init
 SN SPDK00000000000000
 MN SPDK_Controller0
 Namespace Null0
.
.
[Subsystem7 ]
```

```
NQN nqn.2016-06.io.spdk:cnode7
Listen TCP 10.1.1.149:4420
AllowAnyHost Yes
Host nqn.2016-06.io.spdk:init
SN SPDK00000000000007
MN SPDK_Controller7
Namespace Null7
```

Latency test:

```
[Global]
[Malloc]
[AIO]
[Nvmf]
 AcceptorPollRate 10000
 ConnectionScheduler RoundRobin

[Transport]
 Type TCP

[Nvme]

 TransportID "trtype:PCIe traddr:0000:01:00.0" Nvme0

[Subsystem2]
  NQN nqn.2016-06.io.spdk:cnode2
  Listen TCP 10.1.1.144:4420
  AllowAnyHost Yes
  Host nqn.2016-06.io.spdk:init
  SN SPDK00000000000002
  MN SPDK_Controller2
  Namespace Nvme0n1 1
```

iv. Start the target.

```
[root@host spdk]# ./app/nvmf_tgt -m 0xFFF -c etc/spdk/nvmf.conf.in
```

Host

i. Load the Chelsio NIC driver and bring up interface with IPv4 address.

```
[root@host~]# modprobe cxgb4
[root@host~]# ifconfig ethX <IP address> mtu 9000 up
```

ii. CPU affinity was set for BW/IOPs test.

```
[root@host~]# t4_perftune.sh -n -Q nic
```

CPU affinity was set for latency test to use a single CPU (3 in this case).

```
[root@host~]# t4_perftune.sh -n -Q nic -c 3
```

iii. Clone the SPDK, configure with FIO and install it.

Note: Ensure CUnit and libuuid packages are installed.

```
[root@host~]# git clone https://github.com/spdk/spdk
[root@host~]# cd spdk
```

```
[root@host~]# git submodule update –init
[root@host~]# ./configure --with-fio=/root/fio-fio-3.20/ --disable-tests
[root@host~]# make && make install
[root@host~]# scripts/setup.sh
```

iv. Configure Hugepages.

```
[root@host~]# mkdir /root/huge_1GB
[root@host~]# echo 10 > /sys/kernel/mm/hugepages/hugepages-1048576kB/nr_hugepages
[root@host~]# echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
[root@host~]# vim /etc/fstab
nodev      /dev/hugepages          hugetlbfs pagesize=2MB  0 0
nodev      /root/huge_1GB          hugetlbfs pagesize=1GB  0 0
[root@host~]# mount -a
```

BW/IOPs test:

Run the fio test on all 4 hosts at the same time.

```
[root@host1~]# LD_PRELOAD=/root/spdk/build/fio/spdk_nvme fio --
rw=randread/randwrite --name=random --norandommap=1 --ioengine=spdk --thread=1
--size=400m --group_reporting --exitall --invalidate=1 --direct=1 --
filename='trtype=TCP adrfam=IPv4 traddr=10.1.1.149 trsvcid=4420
subnqn=nqn.2016-06.io.spdk\:cnode0 ns=1:trtype=TCP adrfam=IPv4
traddr=10.1.1.149 trsvcid=4420 subnqn=nqn.2016-06.io.spdk\:cnode1 ns=1' --
time_based --runtime=30 --iodepth=64 --numjobs=4 --unit_base=1 --bs=<value> --
kb_base=1000 --ramp_time=2

Host2 will use nqn.2016-06.io.spdk:cnode2 and nqn.2016-06.io.spdk:cnode3
Host3 will use nqn.2016-06.io.spdk:cnode4 and nqn.2016-06.io.spdk:cnode5
Host4 will use nqn.2016-06.io.spdk:cnode6 and nqn.2016-06.io.spdk:cnode7
```

Latency test:

Run the fio test on one host.

```
[root@host1~]# LD_PRELOAD=/root/spdk/build/fio/spdk_nvme fio --
rw=randread/randwrite --name=random --norandommap=1 --ioengine=spdk --thread=1
--size=400m --group_reporting --exitall --fsync_on_close=1 --invalidate=1 --
direct=1 --filename='trtype=TCP adrfam=IPv4 traddr=10.1.1.149 trsvcid=4420
ns=1' --time_based --runtime=30 --iodepth=1 --numjobs=1 --unit_base=1 --
bs=<value> --kb_base=1000 --ramp_time=2
```

Kernel NVMe/TOE Configuration

Execute the below commands on both Target and host machines.

i. Load the Chelsio Offload drivers and bring up interface with IPv4 address.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig ethX <IP address> mtu 9000 up
[root@host~]# mount -t configfs none /sys/kernel/config
```

ii. Apply the below TOE cop policy.

```
[root@host~]# cat /root/cop_policy
all => offload !nagle !ddp !coalesce
[root@host~]# cop -d -o file /root/cop_policy
[root@host~]# cxgbtool ethX policy file
```

iii. Set the below TOE sysctl parameters.

```
[root@host~]# sysctl -w toe.toe0_tom.max_host_sndbuf=49152
[root@host~]# sysctl -w toe.toe0_tom.txplen=0
```

iv. CPU affinity was set for BW/IOPs test.

```
[root@host~]# t4_perftune.sh -n -Q ofld
```

CPU affinity was set for Latency test to use a single CPU (3 in this case).

```
[root@host~]# t4_perftune.sh -n -Q ofld -c 3
```

Target

BW/IOPs test:

i. Load the NVMe drivers.

```
[root@host~]# modprobe nvmet
[root@host~]# modprobe nvmet-tcp
```

ii. Create 8 Null Block devices, each of 1GB size.

```
[root@host~]# modprobe null_blk nr_devices=8 gb=1 use_per_node_hctx=Y
```

iii. Configure the target using the below script.

```
#!/bin/bash
IPPORT="4420"
IPADDR="10.1.1.149"  # the ipaddress of your target TCP interface
NAME="nvme-nullb"    # Use "nvme-ssd" while configuring SSDs
DEV="/dev/nullb"     # Use "/dev/nvme0n1" while configuring SSDs

for i in `seq 0 7`; do   # For latency test, use `seq 0 1`
mkdir /sys/kernel/config/nvmet/subsystems/${NAME}${i}
mkdir -p /sys/kernel/config/nvmet/subsystems/${NAME}${i}/namespaces/1
echo -n ${DEV}${i}
>/sys/kernel/config/nvmet/subsystems/${NAME}${i}/namespaces/1/device_path
echo 1 > /sys/kernel/config/nvmet/subsystems/${NAME}${i}/attr_allow_any_host
echo 1 > /sys/kernel/config/nvmet/subsystems/${NAME}${i}/namespaces/1/enable
done

mkdir /sys/kernel/config/nvmet/ports/1
echo "ipv4" > /sys/kernel/config/nvmet/ports/1/addr_adrfam
echo "tcp" > /sys/kernel/config/nvmet/ports/1/addr_trtype
echo $IPPORT > /sys/kernel/config/nvmet/ports/1/addr_trsvcid
echo $IPADDR > /sys/kernel/config/nvmet/ports/1/addr_traddr

for i in `seq 0 7`; do   # For latency test, use `seq 0 1`
                ln -s /sys/kernel/config/nvmet/subsystems/${NAME}${i}
/sys/kernel/config/nvmet/ports/1/subsystems/${NAME}${i}
done
```

Host

i. Load the NVMe drivers.

```
[root@host~]# modprobe nvme-tcp
```

BW/IOPs test:

i. The 4 hosts connect to the targets.

```
[root@host1~]# for i in `seq 0 1`; do nvme connect -t tcp -s 4420 -a
10.1.1.149 -n nvme-nullb${i} -i 4; done
[root@host2~]# for i in `seq 2 3`; do nvme connect -t tcp -s 4420 -a
10.1.1.149 -n nvme-nullb${i} -i 4; done
[root@host3~]# for i in `seq 4 5`; do nvme connect -t tcp -s 4420 -a
10.1.1.149 -n nvme-nullb${i} -i 4; done
[root@host4~]# for i in `seq 6 7`; do nvme connect -t tcp -s 4420 -a
10.1.1.149 -n nvme-nullb${i} -i 4; done
```

ii. *fio* tool was run on all 4 hosts at the same time.

```
[root@host~]# fio --rw=randwrite/randread --ioengine=libaio --name=random --
norandommap --size=400m --group_reporting --exitall --fsync_on_close=1 --
invalidate=1 --direct=1 --runtime=30 --time_based --filename= <device list> -
-iodepth=64 --numjobs=16 --bs=<value> --unit_base=1 -kb_base=1000 --
ramp_time=2
```

Latency test:

i. Single host connects to the target.

```
[root@host~]# nvme connect -t tcp -a 10.1.1.149 -n nvme-ssd0 -i 1
```

ii. *fio* tool was run on the host.

```
[root@host~]# fio --rw=randwrite/randread --ioengine=libaio --name=random  -
-size=400m --invalidate=1 --direct=1 --runtime=30 --time_based --
fsync_on_close=1 --group_reporting --filename= <device list> --iodepth=1 --
numjobs=1 --bs=4K
```

Kernel NVMe/TCP Configuration

Please refer the Kernel NVMe/TOE steps mentioned above. Chelsio NIC driver (cxgb4) needs to be loaded instead of Chelsio Offload driver (t4_tom) on the target and host machines.

Conclusion

This paper showcases the remote storage access performance capabilities of the Chelsio T6 100G NVMe/TOE solution. It enables the NVMe storage devices to be shared, pooled and managed more effectively across a low latency, high performance network. The results show that it:

- Delivers line-rate 99 Gbps throughput for both READ and WRITE.
- Reaches 2.9 Million IOPs at an I/O size of 4K.
- Adds only 5.5 µs latency for remote NVMe device access compared to local access.
- Provides significant CPU savings compared to NVMe/TCP.

TOE improves performance for all TCP applications while freeing up CPU resources to be used for the application processing. Using a Chelsio adapter along with the Unified Wire Software package available as part of the Chelsio solution, users can create and maintain a true Converged Fabric cluster where all storage and networking traffic runs over a single 25/100Gb network, rather than having to build and maintain multiple networks, resulting in significant acquisition and operational savings.

Related Links

The True Cost of Non-Offloaded NICs
We put the iWARP in NVMe-oF
100G SPDK NVMe over Fabrics
NVMe-oF with iWARP and NVMe/TCP