SoftiWARP Performance with Chelsio 100GbE

NVMe-oF Bandwidth, IOPS and Latency Performance

Executive Summary

SoftiWARP is an open source software implementation of the iWARP protocol suite. It comprises of two main building blocks: a kernel module, which implements the iWARP protocols on top of TCP kernel sockets, and a user level library. SoftiWARP integrates with the industry standard RDMA host stack and thus exports the OpenFabrics RDMA API to both user space and kernel space applications. Due to close integration with the Linux kernel socket layer, SoftiWARP allows for efficient data transfer operations and since the implementation conforms to the iWARP protocol specification, it is wire compatible with any peer network adapter (RNIC) implementing iWARP in hardware. This software is available under a choice of GPL-2.0 or BSD-3 license.

NVMe over Fabrics specification extends the benefits of NVMe to large fabrics, beyond the reach and scalability of PCIe. NVMe enables deployments with hundreds or thousands of SSDs using a network interconnect, such as RDMA over Ethernet. iWARP with T6 provides a low latency, high throughput, plug-and-play Ethernet solution for connecting high performance NVMe SSDs over a scalable, congestion controlled and traffic managed fabric, with no special configuration needed. This paper presents the 100G NVMe-oF performance results with iWARP RDMA Offload target and SoftiWARP initiator. Chelsio's T6 with SoftiWARP delivers line-rate throughput and more than 1 Million IOPS at 4K I/O size. In addition, with only 12 µs delta latency between remote and local storage, 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.

Why SoftiWARP?

Chelsio adapters supporting iWARP RDMA hardware Offload completely bypass the host software stack processing thus eliminating any inefficiencies due to software processing. They now support software iWARP solution which offers the below advantages:

- Any L2 NIC can now run iWARP protocol and leverage the high performance.
- It provides a simple path for transition of RDMA applications to the cloud platform.
- It is useful for Client/Initiator side applications like iSER, NVMe-oF, NFSoRDMA, LustreoRDMA etc. to connect to hardware offloaded versions on the target side.
- Supports the ability to work with any legacy switch infrastructure, enabling a decoupled server and switch upgrade cycle.

Test Results

The following graph presents NVMe-oF READ, WRITE IOPS and throughput results of Chelsio T6 using SoftiWARP solution with Null Block devices. The results are collected using the **fio** tool with I/O size varying from 4 to 256 KBytes with an access pattern of random READs and WRITEs.

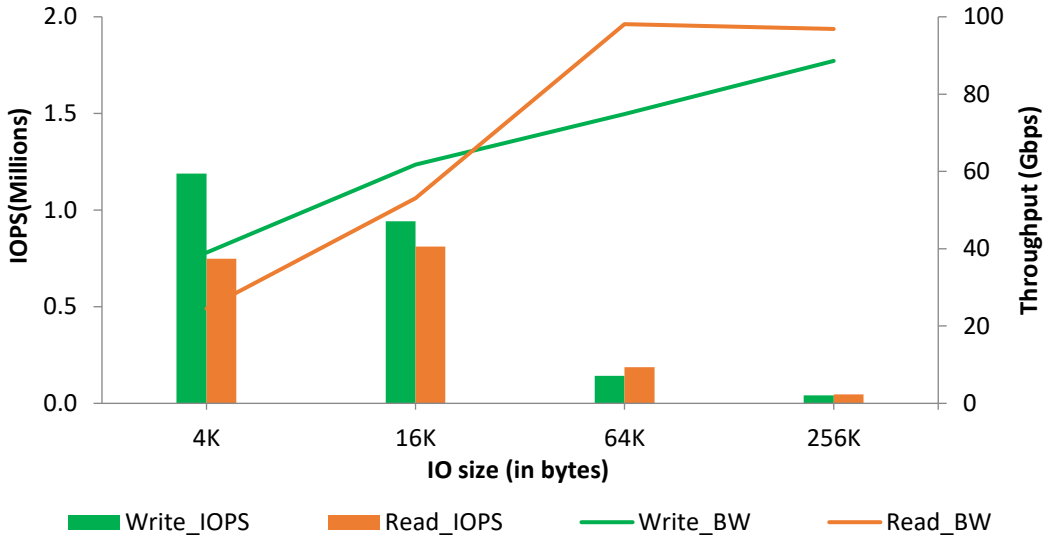


Figure 1 – NVMe-oF IOPS and Throughput vs. I/O size

As evident from the graph above, SoftiWARP with T6 solution delivers line-rate throughput for both READ and WRITE. WRITE IOPS exceeds 1 Million at 4K I/O size. *Performance tuning is in progress.*

The following graph presents the latency numbers at 4K I/O size varying the number of connections.

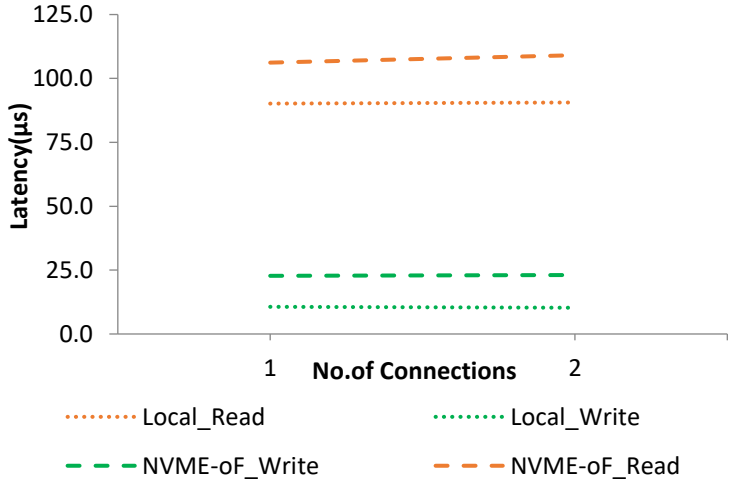
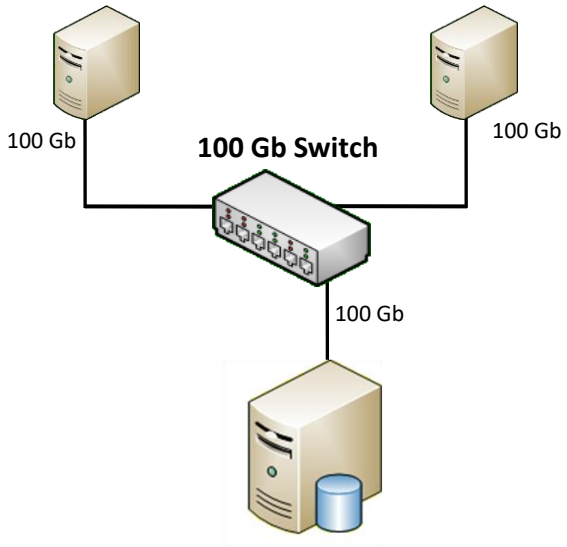


Figure 2 - Latency vs. #. connections

From the graph above, observe that remote versus local NVMe device adds only 12 and 16µs latency at 4K I/O for both WRITE and READ operations.

Test Setup



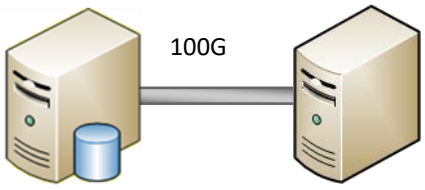
- Supermicro X10SRA-F Initiators with T62100-CR adapter
- 1 Intel Xeon CPU E5-1620 v4 4-core @ 3.50GHz (HT enabled)
- 32GB RAM
- RHEL 7.3 (5.0.0-rc5 kernel)

- Supermicro X10DRG-Q Target with T62100-CR adapter
- 2 Intel Xeon CPUs E5-2687W v4 12-core @ 3.00GHz (HT disabled)
- 128GB RAM
- RHEL 7.3 (5.0.0-rc5 kernel)

Figure 3 – Test setup

The BW/IOPS test setup consists of an NVMe target machine connected to two initiator machines through a 100GbE switch using single port on each system. An MTU of 9000B is used on the ports under test.

- Supermicro X10DRG-Q Target with T62100-CR adapter.
- 2 Intel Xeon CPUs E5-2687W v4 12-core @ 3.00GHz (HT disabled)
- 128GB RAM, 2 INTEL 800GB PCIe NVME SSD



- Supermicro X10SRA-F Initiator with T62100-CR adapter.
- 1 Intel Xeon CPU E5-1620 v4 4-core @ 3.50GHz (HT disabled)
- 32GB RAM

Figure 4 – Latency Test setup

The Latency test setup consists of an NVMe target machine connected to a single initiator connected back to back using single port on each system. MTU of 9000B is used.

Storage configuration

Target server is enabled with T6 iWARP RDMA Offload and configured with 16 NVMe-oF targets, using NULL BLOCK devices, each 1GB in size. Each SoftiWARP Initiator connects to 8 targets using 2 connections per target.

For latency test, the target is configured with 2 Intel NVMe SSD's. One Initiator connects to the target using one connection.

Setup Configuration

Target/Initiator Configuration

i. Disable virtualization, c-state technology, VT-d, Intel I/O AT and SR-IOV in system BIOS.
ii. Compile and install custom 5.0.0-rc5 kernel from https://github.com/zrlio/softiwarp-for-linux-rdma.git, branch: origin/siw-for-rdma-next-v5.

    Make sure the below kernel config parameters are enabled:

```
CONFIG_RDMA_SIW=m
CONFIG_NVME_RDMA=m
CONFIG_NVME_CORE=m
CONFIG_NVME_FABRICS=m
CONFIG_NVME_TARGET=m
CONFIG_NVME_TARGET_RDMA=m
CONFIG_NVME_MULTIPATH=y
```

iii. Reboot the machine into the newly installed kernel.
iv. 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
```

v. For latency test, add *idle=poll* to the kernel command line.
vi. Install the rdma-core by following the below procedure to get the SoftiWARP user space library (libsiw).

    Note: 'pandoc' library is needed for generating rdma-core rpms. It can be downloaded from https://github.com/jgm/pandoc/releases and can be installed as shown below:

```
[root@host~]# tar xvzf pandoc-X.X-linux.tar.gz --strip-components 1 -C
/usr/local/

[root@host~]# cd rpmbuild/SOURCES/
[root@host~]# git clone https://github.com/zrlio/softiwarp-user-for-linux-
rdma rdma-core
[root@host~]# grep "Version:" rdma-core/redhat/rdma-core.spec
Version: 20.0
```

    Rename the rdma-core by appending the version number as shown below:

```
[root@host~]# mv rdma-core rdma-core-20.0
[root@host~]# tar -cf rdma-core-20.0.tgz ./rdma-core-20.0
[root@host~]# cd rdma-core-20.0
```

    Build rdma-core RPMs (All rdma-core rpms including libibverbs, librdmacm etc will be generated under 'rpmbuild/RPMS/' directory):

```
[root@host~]# rpmbuild -bb ./redhat/rdma-core.spec
[root@host~]# cd /root/rpmbuild/RPMS/x86_64
```

Uninstall all the previously installed rdma-core, librdmacm and libibverbs rpms on the system and install the newly built RPMs using 'rpm -ivh <rpm-name>' command.

vii. Load the NVMe drivers.

```
On target:
[root@host~]# modprobe nvmet
[root@host~]# modprobe nvme
[root@host~]# modprobe nvmet-rdma

On Initiator:
[root@host~]# modprobe nvme-rdma
```

viii. Stop the iwpmd service.

```
[root@host~]# systemctl disable iwpmd.service
[root@host~]# systemctl stop iwpmd.service
```

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

```
On target:
[root@host~]# t4_perftune.sh -n -Q rdma

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

Target Configuration

i. Load the Chelsio iWARP RDMA Offload driver with T6 NVMe-oF performance config file and bring up the interface with an MTU 9000.

```
[root@host~]# modprobe iw_cxgb4
[root@host~]# ifconfig ens2f4 10.1.1.149/24 mtu 9000 up
```

BW/IOPs test:

i. Clone the below repo to get the 'nvmetcli' utility. Follow the README inside for installing it.

```
[root@host~]# git clone git://git.infradead.org/users/hch/nvmetcli.git
```

ii. Create 16 Null Block devices, each of 1GB size.

```
[root@host~]# modprobe null_blk nr_devices=16 gb=1 use_per_node_hctx=Y
```

iii. Configure 16 targets using the below script:

```
IPPORT="4420"        # 4420 is the reserved NVME/Fabrics RDMA port
IPADDR="10.1.1.149"  # the ipaddress of your target rdma interface
NAME="nvme-nullb"    # Use "nvme-ssd" while configuring SSDs
DEV="/dev/nullb"     # Use " /dev/nvme" while configuring SSDs

for i in `seq 0 15`; 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
# Use "echo -n ${DEV}${i}n1" while configuring SSDs
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 8192 > /sys/kernel/config/nvmet/ports/1/param_inline_data_size
echo "ipv4" > /sys/kernel/config/nvmet/ports/1/addr_adrfam
echo "rdma" > /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 15`; do
 ln -s /sys/kernel/config/nvmet/subsystems/${NAME}${i}
/sys/kernel/config/nvmet/ports/1/subsystems/${NAME}${i}
done
```

iv. Run the below to check the target configuration.

```
[root@host~]# nvmetcli/nvmetcli ls
```

Initiator Configuration

i. Clone and compile the 'nvme-cli' repo to get the 'nvme' command.

```
[root@host~]# git clone https://github.com/linux-nvme/nvme-cli
[root@host~]# cd nvme-cli
[root@host~]# make
```

ii. Clone the iproute2 package and install the 'rdma' tool.

a. Git link: https://github.com/larrystevenwise/iproute2.git , branch: origin/wip/newlink-2019-01-07
b. Pre-requisites: Install libmnl and libmnl-devel rpms. To find out more dependencies, run './configure' and install the missing packages.

iii. Load the necessary drivers along with 'siw' and bring up the interface.

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe siw
[root@host~]# rmmod iw_cxgb4
[root@host~]# iproute2/rdma/rdma link add siw-ens1f4 type siw netdev ens1f4
[root@host~]# ifconfig ens1f4 10.1.1.150/24 mtu 9000 up
```

Discover the target:

```
[root@host~]# nvme discover -t rdma -a <target_ip> -s 4420
```

BW/IOPs test:

i. Initiator1 connects to 8 Targets.

```
[root@host1~]# for i in `seq 0 7`; do nvme connect -i 2 -t rdma -a <target_ip>
-s 4420 -n nvme-nullb${i}; done
```

ii. Initiator2 connects to 8 Targets.

```
[root@host2~]# for i in `seq 8 15`; do nvme connect -i 2 -t rdma -a
<target_ip> -s 4420 -n nvme-nullb${i}; done
```

iii. *fio* tool is run on both initiators at the same time.

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

iv. To disconnect the targets.

```
[root@host~]# for i in /dev/nvme*n1 ; do nvme disconnect -d $(basename $i);
done
```

Latency test:

i. Single Initiator connects to the target:

```
[root@host~]# for i in `seq 0 $((N))`; do /root/nvme-cli/nvme connect -t rdma
-a 10.1.1.149 -n nvme-ssd${i} -i 1; done
```

where N specifies the number of target devices.

ii. *fio* tool is run on the initiator.

```
[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
```

iii. After collecting latency for single target device, initiator was disconnected and logs into two target devices for latency collection.

Conclusion

This paper showcases the remote storage access performance capabilities of Chelsio T6 100G NVMe-oF with SoftiWARP initiator and hardware iWARP offload on target. It also proves the SoftiWARP solution is compatible with Chelsio RNICs which already has iWARP implemented in the hardware. Combining both can results in significant acquisition with operational savings and can achieve adequate performance for applications (like iSER initiator, NVMe-oF initiator, NFSRDMA client and others). Using iWARP RDMA enables the NVMe storage devices to be shared, pooled and managed more effectively across a low latency, high performance network.

References

https://arxiv.org/pdf/1703.07626.pdf