Chelsio Communications

Accelerate

Chelsio Unified Wire for Linux

Installation and User's Guide

Chelsio Communications (Headquarters)
735 N Pastoria Avenue,
Sunnyvale, CA 94085
U.S.A

www.chelsio.com

Tel: 408.962.3600
Fax: 408.962.3661

Chelsio (India) Private Limited
Subramanya Arcade, Floor 3, Tower B
No. 12, Bannerghatta Road,
Bangalore-560029
Karnataka,
India

Tel: +91-80-4039-6800

Sales
For all sales inquiries please send email to sales@chelsio.com

Support
For all support related questions please send email to support@chelsio.com

Document History

Version	Revision Date
1.4.5	03/29/2019
1.4.6	08/12/2019
1.4.7	11/11/2019
1.4.8	12/09/2019
1.4.9	01/29/2020
1.5.0	06/23/2020
1.5.1	01/11/2021
1.5.2	02/10/2021
1.5.3	03/31/2021
1.5.4	05/21/2021
1.5.5	07/08/2021
1.5.6	09/24/2021

TABLE OF CONTENTS

I. Chelsio Unified Wire

1. Introduction

Thank you for choosing Chelsio Unified Wire adapters. These high speed, single chip, single firmware cards provide enterprises and data centers with high performance solutions for various Network and Storage related requirements.

The **Terminator** series is Chelsio's next generation of highly integrated, hyper-virtualized 1/10/25/40/50/100GbE controllers. The adapters are built around a programmable protocol-processing engine, with full offload of a complete Unified Wire solution comprising NIC, TOE, iWARP RDMA, iSCSI, FCoE and NAT support. It scales to true 100Gb line rate operation from a single TCP connection to thousands of connections, and allows simultaneous low latency and high bandwidth operation thanks to multiple physical channels through the ASIC.

Ideal for all data, storage and high-performance clustering applications, the Unified Wire adapters enable a unified fabric over a single wire by simultaneously running all unmodified IP sockets, Fibre Channel and InfiniBand applications over Ethernet at line rate.

Designed for deployment in virtualized data centers, cloud service installations and high-performance computing environments, Chelsio adapters bring a new level of performance metrics and functional capabilities to the computer networking industry.

Chelsio Unified Wire software comes in two formats: Source code and RPM package forms. Installing from source requires compiling the package to generate the necessary binaries. You can choose this method when you are using a custom-built kernel. You can also install the package using the interactive GUI installer. In other cases, download the RPM package specific to your operating system and follow the steps mentioned to install the package.

This document describes the installation, use and maintenance of Unified Wire software and its various components.

1.1. Features

The Chelsio Unified Wire package uses a single command to install various drivers and utilities. It consists of the following software:

- **Network (NIC/TOE)**
- **Virtual Function Network (vNIC)**
- **iWARP RDMA Offload**
- **iSER (Target & Initiator)**
- **WD-UDP**
- **NVMe-oF iWARP (Target & Initiator)**
- **SPDK NVMe-oF iWARP (Target & Initiator)**
- **NVMe-oF TOE (Target & Initiator)**
- **SPDK NVMe-oF TOE Target**

- **SoftiWARP Initiator**
- **LIO iSCSI Target Offload**
- **iSCSI PDU Offload Target**
- **iSCSI PDU Offload Initiator**
- **Crypto Offload**
- **Data Center Bridging (DCB)**
- **FCoE full offload Initiator**
- **Offload Bonding**
- **Offload Multi-Adapter Failover (MAFO)**
- **UDP Segmentation Offload and Pacing**
- **Offload IPv6**
- **WD Sniffing & Tracing**
- **Classification and Filtering**
- **OVS Kernel Datapath Offload**
- **Mesh Topology**
- **Traffic Management feature (TM)**
- **Unified Boot Software**
- **Utility Tools (cop, cxgbtool, t4_perftune, benchmark tools)**

For detailed instructions on loading, unloading and configuring the drivers/tools please refer to their respective sections.

1.2. Hardware Requirements

The Chelsio Unified Wire software supports Chelsio Terminator series of Unified Wire adapters. To know more about the list of adapters supported by each driver, please refer to their respective sections.

1.3. Software Requirements

The Chelsio Unified Wire software has been developed to run on 64-bit Linux based platforms and therefore it is a base requirement for running the driver. To know more about the complete list of operating systems supported by each driver, please refer to their respective sections.

1.4. Package Contents

1.4.1. Source Package

The Chelsio Unified Wire source package consists of the following files/directories:

- **debrules**: This directory contains packaging specification files required for building Debian packages.
- **docs**: This directory contains support documents - README, Release Notes and User's Guide (this document) for the software.

- **kernels**: This directory contains kernel.org-5.4 installation files.
- **libs**: This directory is for iWARP and WD-UDP libraries. The libibverbs library has implementation of RDMA verbs which will be used by iWARP applications for data transfers. The librdmacm library works as an RDMA connection manager. The libcxgb4 library works as an interface between the above mentioned generic libraries and Chelsio iWARP driver. The libcxgb4_sock library is a LD_PRELOAD-able library that accelerates UDP Socket communications transparently and without recompilation of the user application.
- **RPM-Manager**: This directory contains support scripts used for cluster deployment.
- **scripts**: Support scripts used by the Unified Wire Installer.
- **specs**: The packaging specification files required for building RPM packages.
- **src**: Source code for different drivers.
- **support**: This directory contains source files for the dialog utility.
- **tools**:
    - **autoconf-x.xx**: This directory contains the source for Autoconf tool needed for iWARP and WD-UDP libraries.
    - **benchmarks**: This directory contains various benchmarking tools to measure throughput and latency of various networks.
    - **chelsio_adapter_config**: This directory contains scripts and binaries needed to configure Chelsio adapters.
    - **cop**: The cop tool compiles offload policies into a simple program form that can be loaded into the kernel and interpreted. These offload policies are used to determine the settings to be used for various connections. The connections to which the settings are applied are based on matching filter specifications. Please find more details on this tool in its manual page (run `man cop` command).
    - **cudbg**: Chelsio Unified Debug tool which facilitates collection and viewing of various debug entities like register dump, Devlog, CIM LA, etc.
    - **cxgbtool**: The cxgbtool queries or sets various aspects of Chelsio network interface cards. It complements standard tools used to configure network settings and provides functionality not available through such tools. Please find more details on this tool in its manual page (run man cxgbtool command). To use cxbtool for FCoE Initiator driver, use `[root@host~]# cxgbtool stor -h`
    - **nvme_utils**: This directory contains *nvmecli*, *nvmetcli* and *targetcli* installation files, and dependent components.
    - **rdma_tools**: This directory contains iWARP benchmarking tools.
    - **t4_sniffer**: This directory contains sniffer tracing and filtering libraries. See WD Sniffing and Tracing chapter for more information.
    - **90-rdma.rules**: This file contains udev rules needed for running RDMA applications as a non-root user.
    - **chdebug**: This script collects operating system environment details and debug information which can be sent to the support team, to troubleshoot Chelsio hardware/software related issues.

  - **chiscsi_set_affinity.sh**: This shell script is used for mapping iSCSI Worker threads to different CPUs.
  - **chsetup**: The chsetup tool loads NIC, TOE and iWARP drivers, and creates WD-UDP configuration file.
  - **chstatus**: This utility provides status information on any Chelsio NIC in the system.
  - **Makefile**: The Makefile for building and installing tools.
  - **t4_latencytune.sh**: Script used for latency tuning of Chelsio adapters.
  - **t4_perftune.sh**: This shell script is to tune the system for higher performance. It achieves it through modifying the IRQ-CPU binding. This script can also be used to change Tx coalescing settings.
  - **t4-forward.sh**: RFC2544 Forward test tuning script.
  - **uname_r**: This file is used by chstatus script to verify if the Linux platform is supported or not.
  - **wdload**: UDP acceleration tool.
  - **wdunload**: Used to unload all the loaded Chelsio drivers.
- **Uboot**: The directory contains Unified Boot Option ROM image (*cubt4.bin*), uEFI driver (*ChelsioUD.efi*), default boot configuration file (*boot.cfg*) and a legacy flash utility (*cfut4.exe*) to flash the option ROM onto Chelsio adapters.
- **chelsio-dkms.conf**: DKMS configuration files for Ubuntu.
- **install.py**, **dialog.py**: Python scripts needed for the GUI installer.
- **EULA**: Chelsio's End User License Agreement.
- **install-dkms.sh**: Installs necessary drivers to DKMS tree for Ubuntu.
- **install.log**: File containing installation summary.
- **Makefile**: The Makefile for building and installing from the source.
- **sample_machinefile**: Sample file used during iWARP installation on cluster nodes.

1.4.2. RPM Package

The Chelsio Unified Wire RPM package consists of the following:

- **config**: This directory contains firmware configuration files.
- **docs**: This directory contains support documents i.e. README, Release Notes and User's Guide (this document) for the software.
- **DRIVER-RPMS**: RPM packages of Chelsio drivers.
- **scripts**: Support scripts used by the Unified Wire Installer.
- **EULA**: Chelsio's End User License Agreement.
- **install.py**: Python script that installs the RPM package. See **Chelsio Unified Wire's** Software/Driver Installation section for more information.
- **uninstall.py**: Python script that uninstalls the RPM package. See **Chelsio Unified Wire's** Software/Driver Uninstallation section for more information.
- **Uboot**: The directory contains Unified Boot Option ROM image (*cubt4.bin*), uEFI driver (*ChelsioUD.efi*), default boot configuration file (*boot.cfg*) and a legacy flash utility (*cfut4.exe*) to flash the option ROM onto Chelsio adapters.

2. Hardware Installation

Follow these steps to install Chelsio adapter in your system:

i. Shutdown/power off your system.
ii. Power off all remaining peripherals attached to your system.
iii. Unpack the Chelsio adapter and place it on an anti-static surface.
iv. Remove the system case cover as per the system manufacturer's instructions.
v. Remove the PCI filler plate from the slot where you will install the Ethernet adapter.
vi. For maximum performance, it is highly recommended to install the adapter into a PCIe x8/x16 slot.

Note *All 4-ports of T6425-CR adapter will be functional only if PCIe x8 -> 2x PCIe x4 slot bifurcation is supported by the system and enabled in BIOS. Otherwise, only 2-ports will be functional.*

vii. Holding the Chelsio adapter by the edges, align the edge connector with the PCI connector on the motherboard. Apply even pressure on both edges until the card is firmly seated. It may be necessary to remove the SFP (transceiver) modules prior to inserting the adapter.
viii. Secure the Chelsio adapter with a screw, or other securing mechanism, as described by the system manufacturer's instructions. Replace the case cover.
ix. After securing the card, ensure that the card is still fully seated in the PCIE x8/x16 slot as sometimes the process of securing the card causes the card to become unseated.
x. Connect a fiber/twinax cable, multi-mode for short range (SR) optics or single-mode for long range (LR) optics, to the Ethernet adapter or regular Ethernet cable for the 1Gb Ethernet adapter.
xi. Power on your system.
xii. Run *update-pciids* command to download the current version of PCI ID list

```
[root@ ~]# update-pciids
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  227k  100  227k    0     0  68592      0  0:00:03  0:00:03 --:--:-- 68610
Done.
```

xiii. Verify if the adapter was installed successfully by using the *lspci* command

```
[root@ ~]# lspci | grep -i Chelsio
81:00.0 Ethernet controller: Chelsio Communications Inc T62100-LP-CR Unified Wire Ethernet Controller
81:00.1 Ethernet controller: Chelsio Communications Inc T62100-LP-CR Unified Wire Ethernet Controller
81:00.2 Ethernet controller: Chelsio Communications Inc T62100-LP-CR Unified Wire Ethernet Controller
81:00.3 Ethernet controller: Chelsio Communications Inc T62100-LP-CR Unified Wire Ethernet Controller
81:00.4 Ethernet controller: Chelsio Communications Inc T62100-LP-CR Unified Wire Ethernet Controller
81:00.5 SCSI storage controller: Chelsio Communications Inc T62100-LP-CR Unified Wire Storage Controller
81:00.6 Fibre Channel: Chelsio Communications Inc T62100-LP-CR Unified Wire Storage Controller
```

For Chelsio adapters, the physical functions are currently assigned as:

- Physical functions 0 - 3: for the SR-IOV functions of the adapter
- Physical function 4: for all NIC functions of the adapter
- Physical function 5: for iSCSI

- Physical function 6: for FCoE
- Physical function 7: Currently not assigned

Once Unified Wire package is installed and loaded, examine the output of `dmesg` to see if the card is discovered. You should see a similar output:

```
[ 1119.854346] cxgb4 0000:81:00.4: Chelsio T62100-LP-CR rev 0
[ 1119.854347] cxgb4 0000:81:00.4: S/N: RE41160042, P/N: 11012106003
[ 1119.854348] cxgb4 0000:81:00.4: Firmware version: [redacted]
[ 1119.854349] cxgb4 0000:81:00.4: Bootstrap version: 255.255.255.255
[ 1119.854350] cxgb4 0000:81:00.4: TP Microcode version: 0.1.23.2
[ 1119.854351] cxgb4 0000:81:00.4: No Expansion ROM loaded
[ 1119.854351] cxgb4 0000:81:00.4: Serial Configuration version: 0x7002000
[ 1119.854352] cxgb4 0000:81:00.4: VPD version: 0x52
[ 1119.854354] cxgb4 0000:81:00.4: Configuration: NIC MSI-X, non-Offload capable
[ 1119.854355] eth0: Chelsio T62100-LP-CR (eth0) 100GBASE-CR4_QSFP
```

The above outputs indicate the hardware configuration of the adapter as well as serial number.

Note

Network device names for Chelsio's physical ports are assigned using the following convention: the port farthest from the motherboard will appear as the first network interface. However, for T5 40G and T420-BT adapters, the association of physical Ethernet ports and their corresponding network device names is opposite. For these adapters, the port nearest to the motherboard will appear as the first network interface.

3. Software/Driver Installation

There are two main methods to install the Chelsio Unified Wire package: from source and RPM.
If you decide to use source, you can install the package using CLI or GUI mode.
If you decide to use RPM, you can install the package using Menu or CLI mode.

RPM packages support only distro base kernels. In case of updated/custom kernels, use source package.

The following table describes the various *configuration tuning options* available during installation and drivers/software installed with each option by default:

Configuration Tuning Option	Description	Driver/Software installed
Unified Wire (Default)	Default Configuration. Configures adapters to run all protocols simultaneously.	NIC/TOE,vNIC,iWARP,iSER,WD-UDP, NVMe-oF iWARP, SPDK NVMe-oF iWARP,NVMe-oF TOE,SPDK NVMe-oF TOE,SoftiWARP,LIO iSCSI Target,iSCSI Target,iSCSI Initiator,FCoE Initiator, Bonding,MAFO,UDP-SO,IPv6,Sniffer & Tracer,Filtering,Mesh,TM
Low latency Networking	Configures adapters to run TOE and iWARP traffic with low latency.	TOE, iWARP, WD-UDP, IPv6, Bonding, MAFO
High capacity RDMA	Configures adapters to establish a large number of iWARP connections.	iWARP
RDMA Performance	Improves iWARP performance.	iWARP, iSER, NVMe-oF
High capacity TOE	Configures adapters to establish a large number of TOE connections.	TOE, Bonding, MAFO, IPv6
iSCSI Performance+	Improves iSCSI performance.	LIO iSCSI Target, iSCSI Target, iSCSI Initiator, Bonding, DCB
UDP Seg.Offload & Pacing+	Configures adapters to establish a large number of UDP-SO connections.	UDP-SO, Bonding
Wire Direct Latency	Configures adapters to provide low Wire Direct latency.	TOE, iWARP, WD-UDP
High Capacity WD	Configures adapters to establish a large number of WD-UDP connections.	WD-UDP
NVMe Performance^	Improves NVMe-oF performance.	iWARP, NVMe-oF, SPDK NVMe-oF
High Capacity VF	Configures adapters to support more VFs.	NIC, vNIC
High Capacity Hash Filter	Configures largne number of filters	Filtering

+ Supported only on T5 *^ Supported only on T6*

Important *Crypto, DCB and OVS drivers will not be installed by default. Please refer to their respective sections for instructions on installing them.*

3.1. Pre-requisites

- RHEL 8.X distributions ship with Python v3.6 by default. Configure Python v2.7 using the below commands to run the installer.

```
[root@host~]# tar zxvf ChelsioUwire-x.x.x.x.tar.gz
[root@host~]# cd ChelsioUwire-x.x.x.x
[root@host~]# sh install-python.sh
```

- To install Unified Wire using GUI mode (with Dialog utility), *ncurses-devel* package must be installed.

3.2. Mounting debugfs

All driver debug data is stored in debugfs, which will be mounted in most cases. If not, mount it manually using:

```
[root@host~]# mount -t debugfs none /sys/kernel/debug
```

3.3. Installing Chelsio Unified Wire from source

3.3.1. GUI mode (with Dialog utility)

i. Download the Unified Wire driver package (tarball) from Chelsio Download Center.
ii. Untar the tarball using the following command:

```
[root@host~]# tar zxvf <driver_package>.tar.gz
```

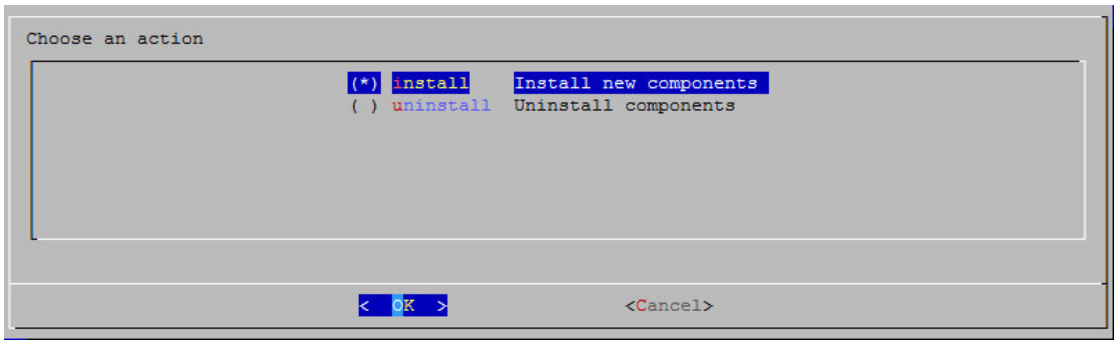
iii. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

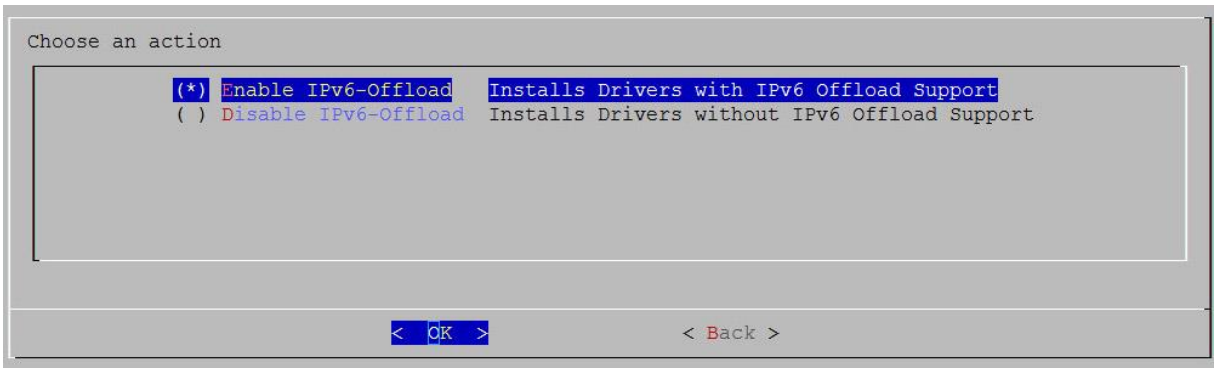
iv. Run the following script to start the GUI installer:

```
[root@host~]# ./install.py
```

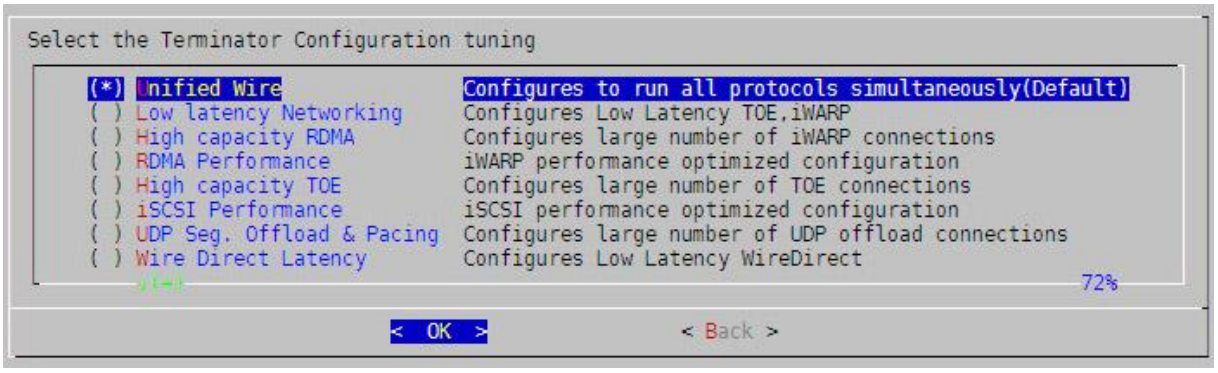
v. If **Dialog** utility is present, you can skip to step (vi). If not, press 'y' to install it when the installer prompts for input.
vi. Select "install" under "Choose an action".



vii. Select *Enable IPv6-Offload* to install drivers with IPv6 Offload support or *Disable IPv6-offload* to continue installation without IPv6 offload support.

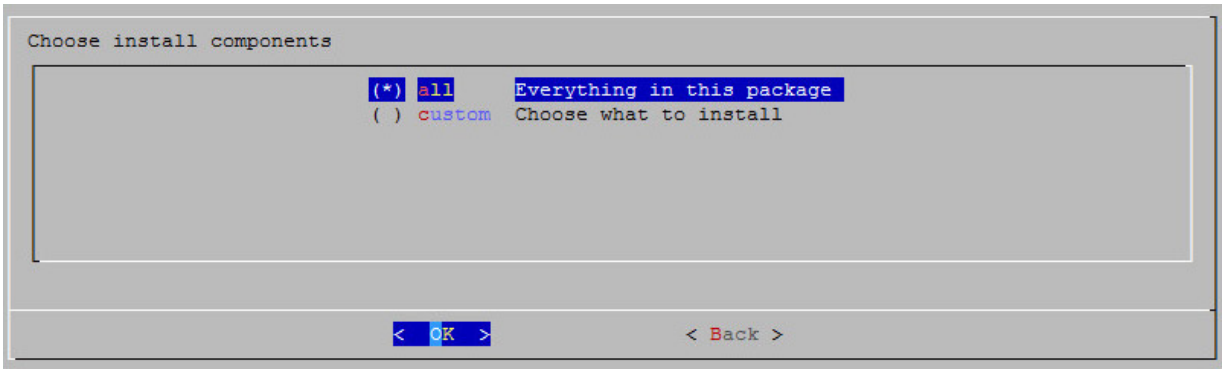


viii. Select the required configuration tuning option.



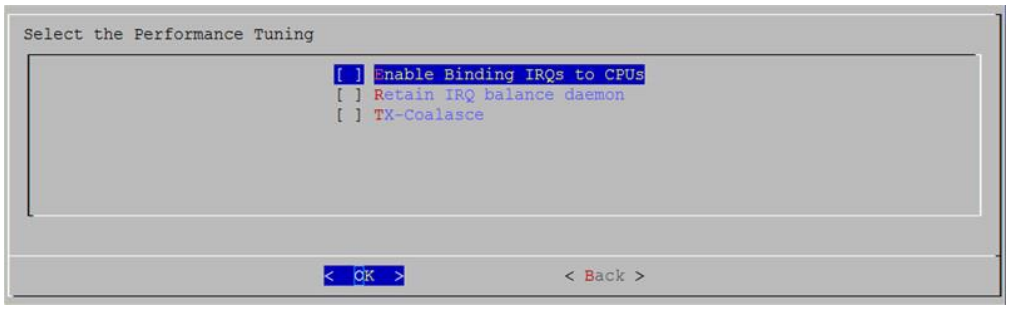
Note *The tuning options may vary depending on the Linux distribution.*

ix. Under “Choose install components”, select “all” to install all the related components for the option chosen in step (viii) or select “custom” to install specific components.



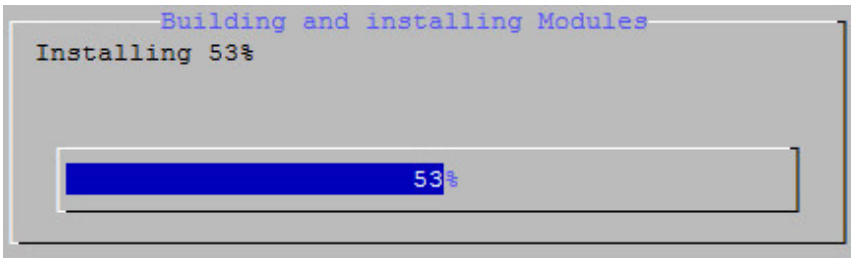
Important *To install Crypto Offload, OVS drivers and benchmark tools, please select “custom option”.*

x. Select the required performance tuning option.
   a. Enable Binding IRQs to CPUs: Bind MSI-X interrupts to different CPUs and disable IRQ balance daemon.
   b. *Retain IRQ balance daemon*: Do not disable IRQ balance daemon.
   c. *TX-Coalasce*: Write tx_coal=2 to modprobe.d/conf.

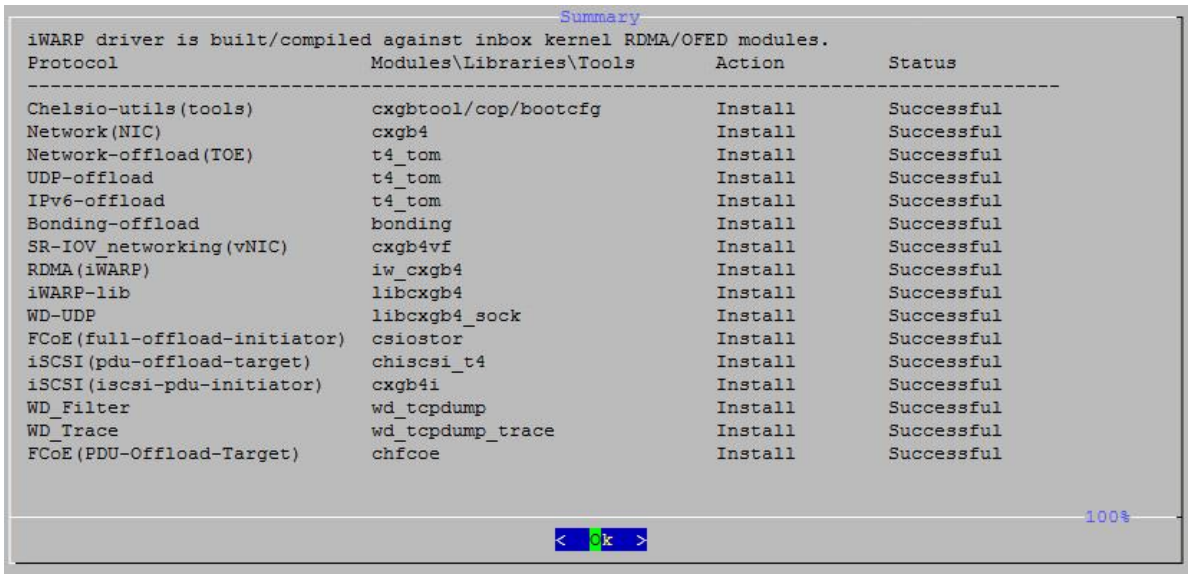


Note *For more information on the Performance tuning options, please refer to* Performance Tuning *section of the* ***Network (NIC/TOE)*** *chapter.*

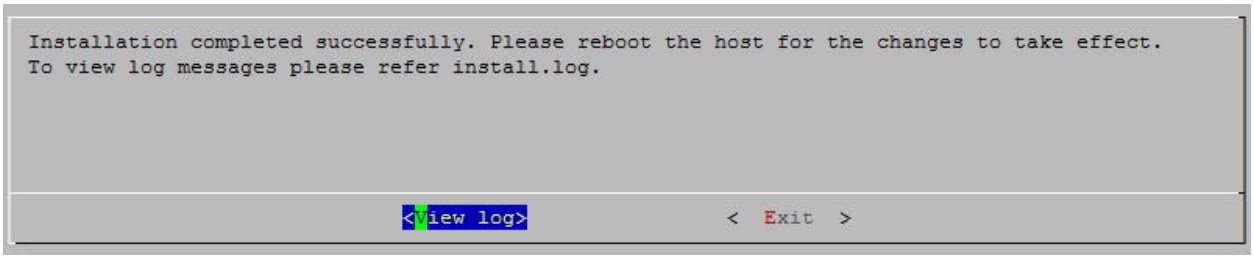
xi. The selected components will now be installed.



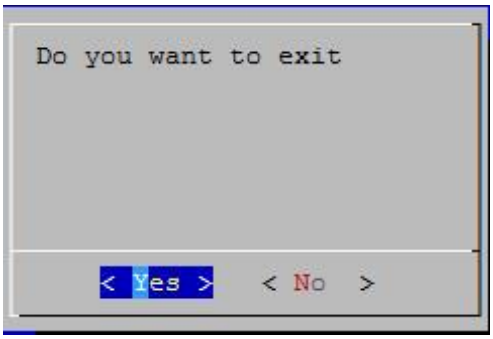
xii. After successful installation, summary of installed components will be displayed.



xiii. Select “View log” to view the installation log or “Exit” to continue.



xiv. Select “Yes” to exit the installer or “No” to go back.



xv. Reboot your machine for changes to take effect.

Note *Press Esc or Ctrl+C to exit the installer at any point of time.*

3.3.2. CLI mode (without Dialog utility)

If your system does not have **Dialog** or you choose not to install it, follow the steps mentioned below to install the Unified Wire package:

i. Download the Unified Wire driver package from Chelsio Download Center.
ii. Untar the tarball using the following command:

```
[root@host~]# tar zxvf <driver_package>.tar.gz
```

iii. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

iv. Run the following script to start the installer.

```
  [root@host~]# ./install.py -c <target>
```

v. Enter the number corresponding to the Configuration tuning option in the Input field and press Enter.

vi. The selected components will now be installed.
After successful installation you can press 1 to view the installation log. Press any other key to exit from the installer.

Important *To customize the installation, view the help by typing*
`[root@host~]#./install.py –h`

vii. Reboot your machine for changes to take effect.

iWARP driver installation on Cluster nodes

Important *Please make sure that you have enabled password less authentication with ssh on the peer nodes for this feature to work.*

Chelsio's Unified Wire package allows installing iWARP drivers on multiple Cluster nodes with a single command. Follow the procedure mentioned below:

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Create a file (*machinefilename*) containing the IP addresses or hostnames of the nodes in the cluster. You can view the sample file, *sample_machinefile,* provided in the package to view the format in which the nodes have to be listed.
iii. Now, execute the following command:

```
[root@host~]# ./install.py -C -m <machinefilename>
```

iv. Select the required configuration tuning option. The tuning options may vary depending on the Linux distribution.
v. Select the required Cluster Configuration.
vi. The selected components will now be installed.

The above commands will install iWARP (*iw_cxgb4*) and TOE (*t4_tom*) drivers on all the nodes listed in the *machinefilename* file.

3.3.3. CLI mode

i. Download the Unified Wire driver package from Chelsio Download Center.

ii. Untar the tarball using the following command:

```
[root@host~]# tar zxvf ChelsioUwire-x.x.x.x.tar.gz
```

iii. Change your current working directory to Chelsio Unified Wire package directory and build the source.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
[root@host~]# make
```

iv. Install the drivers, tools and libraries.

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

v. The default configuration tuning option is *Unified Wire.* The configuration tuning can be selected using the following commands:

```
[root@host~]# make CONF=<configuration_tuning>
[root@host~]# make CONF=<configuration_tuning> install
```

Important *Steps (iii) and (iv) mentioned above will NOT install Crypto, DCB, OVS drivers and benchmark tools. They will have to be installed manually. Please refer to their respective sections for instructions on installing them.*

Note *To view the different configuration tuning options, view help by typing* `[root@host~]# make help`

vi. Reboot your machine for changes to take effect.

3.3.4. CLI mode (additional flags)

Provided here are steps to build and install drivers using additional flags. For the complete list, view help by running `make help`.

Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

- To build and install all drivers without IPv6 support,

```
[root@host~]# make ipv6_disable=1
[root@host~]# make ipv6_disable=1 install
```

- The default configuration tuning option is *Unified Wire.* The configuration tuning can be selected using the following commands:

```
[root@host~]# make CONF=<configuration_tuning> <Build Target>
[root@host~]# make CONF=<configuration_tuning> <Install Target>
```

- To build and install drivers along with benchmarks,

```
[root@host~]# make BENCHMARKS=1
[root@host~]# make BENCHMARKS=1 install
```

- The drivers will be installed as RPMs or Debian packages (for ubuntu). To skip this and install drivers,

```
[root@host~]# make SKIP_RPM=1 install
```

- The installer will remove the Chelsio specific drivers (inbox/outbox) from *initramfs*. To skip this and install drivers,

```
[root@host~]# make SKIP_INIT=1 install
```

- The installer will check for the required dependency packages and will install them if they are missing from the machine. To skip this and install drivers,

```
[root@host~]# make SKIP_DEPS=1 install
```

Note

- *To view the different configuration tuning options, view the help by typing* `[root@host~]#make help`
- *If IPv6 is administratively disabled in the machine, the drivers will be built and installed without IPv6 Offload support by default.*

3.4. Installing Chelsio Unified Wire from RPM

Note

- *IPv6 should be enabled in the machine to use the RPM Packages.*
- *Drivers installed from RPM Packages do not have DCB support.*

3.4.1. Menu Mode

i. Download the tarball specific to your operating system and architecture from Chelsio Download Center.

ii. Untar the tarball:
E.g., for RHEL 6.10, untar using the following command:

```
[root@host~]# tar zxvf <driver_package>-RHEL6.10_x86_64.tar.gz
```

iii. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x-<OS>-<arch>
```

iv. Install Unified Wire.

```
[root@host~]# ./install.py
```

v. Select the Installation type as described below. Enter the corresponding number in the Input field and press Enter.
   a. *Unified Wire*: Install all the drivers in the Unified Wire software package.
   b. *Custom*: Customize the installation. Use this option to install drivers/software and related components as per the tuning option selected.
   c. *EXIT*: Exit the installer.

Note *The Installation options may vary depending on the Configuration tuning option selected.*

vi. The selected components will now be installed.
vii. Reboot your machine for changes to take effect.

Note *If the installation aborts with the message "Resolve the errors/dependencies manually and restart the installation", please go through the install.log to resolve errors/dependencies and then start the installation again.*

3.4.2. CLI mode

i. Download the tarball specific to your operating system and architecture from Chelsio Download Center.
ii. Extract the package. Example, for RHEL 6.10:

```
[root@host~]# tar zxvf ChelsioUwire-x.x.x.x-RHEL6.10_x86_64.tar.gz
```

iii. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x-<OS>-<arch>
```

iv. Install Unified Wire.

```
[root@host~]# ./install.py -i <nic_toe/all/udpso/wd/crypto/ovs>
```

Here,

nic_toe	: NIC and TOE drivers only
all	: All Chelsio drivers
udpso	: UDP segmentation offload capable NIC and TOE drivers only
wd	: Wire Direct drivers and libraries only
crypto	: Crypto drivers and OpenSSL modules.
ovs	: OVS modules and NIC driver.

Note *The Installation options may vary depending on the Linux Distribution.*

v. The default configuration tuning option is *Unified Wire*. The configuration tuning can be selected using the following command:

```
[root@host~]# ./install.py –i <Installation mode> -c <configuration_tuning>
```

Note *To view the different configuration tuning options, view the help by typing* `[root@host~]# ./install.py –h`

vi. Reboot your machine for changes to take effect.

iWARP driver installation on cluster nodes

Important *Please make sure that you have enabled password less authentication with ssh on the peer nodes for this feature to work.*

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x-<OS>-<arch>
```

ii. Create a file (*machinefilename*) containing the IP addresses or hostnames of the nodes in the cluster. You can view the sample file, *sample_machinefile,* provided in the package to view the format in which the nodes have to be listed.

iii. Now, execute the following command:

```
[root@host~]# ./install.py -C -m <machinefilename> -i
<nic_toe/all/udpso/wd> -c <configuration_tuning>
```

The above command will install iWARP (*iw_cxgb4*) and TOE (*t4_tom*) drivers on all the nodes listed in the <machinefilename> file.

iv. Reboot your machine for changes to take effect.

3.5. Firmware Update

The firmware is installed on the system, typically in `/lib/firmware/cxgb4`, and the driver will auto-load the firmware if an update is required. The kernel must be configured to enable userspace firmware loading support:

Device Drivers -> Generic Driver Options -> Userspace firmware loading support

The firmware version can be verified using *ethtool*.

```
[root@host~]# ethtool -i <iface>
```

3.5.1. Verifying Firmware Signature

The firmware signature can be optionally verified to ensure the integrity of

- Firmware binary file (or)
- Firmware loaded on the adapter flash

Please contact Chelsio support at support@chelsio.com for the public key and a sample program code to verify the firmware signature. You can use the sample program from Chelsio or your own developed program for the verification. Below are the steps provided to use the Chelsio program:

i. Compile the sample program code, *verifyfwsignature.c* received from Chelsio support.

```
[root@host~]# gcc -g verifyfwsignature.c -o verifyfwsignature -l crypto
```

- **Firmware binary file**

Use the sample program, *verifyfwsignature* and the public key, *public_key.pem* to verify the Firmware binary.

```
[root@host~]# ./verifyfwsignature -k public_key.pem -f
/lib/firmware/cxgb4/t6fw-x.x.x.x.bin
using public key file public_key.pem
using firmware binary file t6fw-x.x.x.x.bin
…
…
ECDSA signature verified successfully
```

- **Firmware loaded on adapter flash**

ii. Collect all adapter logs using *cudbg_app*.

```
[root@host~]# cudbg_app --collect all ethX logs
```

iii. Extract the flash dump from the collected logs.

```
[root@host~]# cudbg_app --extract flash --path flash_extract logs
```

iv. Use the sample program, *verifyfwsignature* and the public key, *public_key.pem* to verify the Firmware binary.

```
[root@host~]# ./verifyfwsignature -k public_key.pem -l
flash_extract/debug_1/flash
using public key file public_key.pem
using flash dump flash_extract/debug_1/flash
…
…
ECDSA signature verified successfully
```

4. Configuring Chelsio Network Interfaces

To test Chelsio adapters' features it is required to use two machines both with Chelsio's network adapters installed. These two machines can be connected directly without a switch (back-to-back), or both connected to a switch. The interfaces have to be declared and configured. The configuration files for network interfaces on Red Hat Enterprise Linux (RHEL) distributions are kept under `/etc/sysconfig/network-scripts`.

Note *Some operating systems may attempt to auto-configure the detected hardware and some may not detect all ports on a multi-port adapter. If this happens, please refer to the operating system documentation for manually configuring the network device.*

4.1. Configuring Adapters

T6 Unified Wire adapters support auto-negotiation (enabled by default) which allows link parameters like speed, duplex, FEC and Pause to be negotiated with the PEER.

4.1.1. Setting Speed

T6 100G ports support multiple speeds viz. 100G, 50G, 40G, 25G, 10G and 1G. T6 25G ports support 25G, 10G and 1G speeds. The supported speeds can be seen using ethtool.

Note *ethtool v4.8 or higher required.*

Below is a sample output for T6 100G port:

```
[root@host ~]# ethtool enp2s0f4d1
Settings for enp2s0f4d1:
        Supported ports: [ FIBRE ]
        Supported link modes:   1000baseT/Full
                                10000baseKR/Full
                                40000baseSR4/Full
                                25000baseCR/Full
                                50000baseCR2/Full
                                100000baseCR4/Full
        Supported pause frame use: Symmetric
        Supports auto-negotiation: Yes
        Supported FEC modes: BaseR RS
        Advertised link modes:  40000baseSR4/Full
                                25000baseCR/Full
                                50000baseCR2/Full
                                100000baseCR4/Full
        Advertised pause frame use: Symmetric
        Advertised auto-negotiation: Yes
        Advertised FEC modes: None
        Link partner advertised link modes:  40000baseSR4/Full
                                             25000baseCR/Full
                                             50000baseCR2/Full
                                             100000baseCR4/Full
        Link partner advertised pause frame use: Symmetric
        Link partner advertised auto-negotiation: Yes
        Link partner advertised FEC modes: RS
        Speed: 100000Mb/s
        Duplex: Full
        Port: Direct Attach Copper
        PHYAD: 255
        Transceiver: internal
        Auto-negotiation: on
        Current message level: 0x000000ff (255)
                               drv probe link timer ifdown ifup rx_err tx_err
        Link detected: yes
```

- **Optics**

Optics do not support auto-negotiation. Use the following command to change the speed:

```
[root@host~]# ethtool -s <ethX> speed <speed> autoneg off
```

The speed, duplex and FEC (if applicable) should be manually set to the same values on the PEER for the link to come up. For example, to set 25G speed on 100G port:

```
[root@host~]# ethtool -s <ethX> speed 25000 autoneg off
```

- **Twinax**

Twinax cables support auto-negotiation. The following speeds can be set in the *advertise* field.

- Advertise only 100G

```
[root@host~]# ethtool --change <ethX> advertise 0x4000000000
```

- Advertise only 40G

```
[root@host~]# ethtool --change <ethX> advertise 0x2000000
```

- Advertise only 50G

```
[root@host~]# ethtool --change <ethX> advertise 0x400000000
```

- Advertise only 25G

```
[root@host~]# ethtool --change <ethX> advertise 0x80000000
```

- Advertise 100/50/40/25G

```
[root@host~]# ethtool --change <ethX> advertise 0x4482000000
```

- Auto-negotiation OFF
  The *advertise* option is only supported with Auto-negotiation enabled. If it is disabled or to set 10G/1G speeds (which do not support Auto-negotiation), use the following command:

```
[root@host~]# ethtool -s <ethX> speed <speed> autoneg off
```

4.1.2. Setting FEC

100G, 50G and 25G speeds support changing Forward Error Correction (FEC). The existing FEC settings can be viewed using,

```
[root@host~]# cxgbtool <ethX> fec
```

Below is a sample output on T6 100G port:

```
[root@localhost ~]# cxgbtool ens1f4 fec
supported:         auto (IEEE 802.3) Base-R/Reed-Solomon Reed-Solomon
IEEE 802.3 'auto': auto (IEEE 802.3) Base-R/Reed-Solomon Reed-Solomon
requested:         auto (IEEE 802.3)
actual:            Reed-Solomon
```

RS FEC is set by default for the T6 port at 100G speed. FEC will be automatically determined with the PEER and the link will come up.

To configure a different FEC, use the below command:

```
[root@host~]# cxgbtool <ethX> fec <value>
```

Here *value* can be:

rs: Reed-Solomon FEC
baser: Base-R/Reed-Solomon FEC
auto: Use standard FEC settings as specified by IEEE 802.3 interpretations of Cable Transceiver Module parameters.
off: Turn off FEC

Important *RS FEC is not supported on 50G links.*

4.1.3. Setting Pause

Pause Autonegotiation is enabled by default. To override it and set Pause parameters, run:

```
[root@host ~]# ethtool -A <ethX> autoneg off tx on rx on
```

4.1.4. Spider and QSA Modes

- **T5 Adapters**

Chelsio T5 40G adapters can be configured in the following 3 modes:

i. 2X40Gbps: This is the default mode of operation where each port functions as 40Gbps link. The port nearest to the motherboard will appear as the first network interface (Port 0).

ii. 4X10Gbps (Spider): In this mode, port 0 functions as 4 10Gbps links and port 1 is disabled.
iii. QSA: This mode adds support for QSA (QSFP to SFP+) modules, enabling smooth, cost-effective, connections between 40 Gigabit Ethernet adapters and 1 or 10 Gigabit Ethernet networks using existing SFP+ based cabling. The port farthest from the motherboard will appear as the first network interface (Port 0).

- **T6 Adapters**

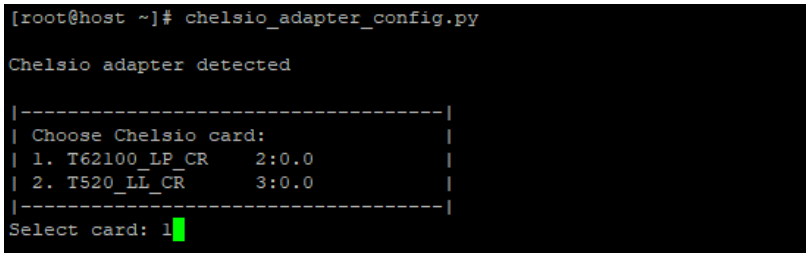
Chelsio T6 100G adapters can be configured in the following 2 modes:

i. 2X100Gbps: This is the default mode of operation where each port functions as 100Gbps link. The port farthest to the motherboard will appear as the first network interface (Port 0).
ii. 2X25Gbps (Spider): In this mode, port 0 functions as 2 25Gbps links and port 1 is disabled.

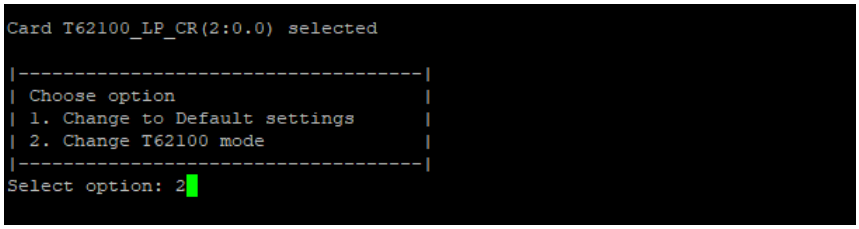
Note *QSA modules will work in the default mode.*

To configure/change the mode of operation, use the following procedure:

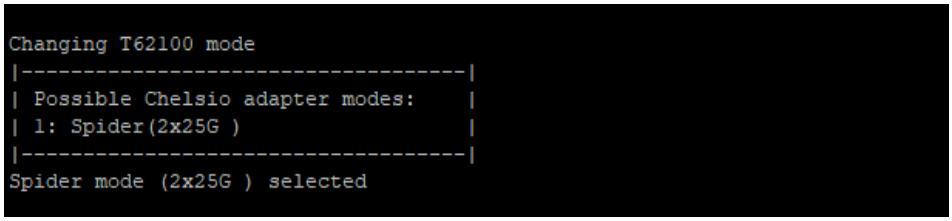
i. Run the *chelsio_adapter_config.py* command to detect all Chelsio adapter(s) present in the system. Select the adapter to configure by specifying the adapter index.



ii. Select *Change adapter mode.*



iii. Select the required mode.



iv. Reload the network driver for changes to take effect.

```
[root@host~]# rmmod cxgb4
[root@host~]# modprobe cxgb4
```

Note *If default option is selected in step ii, reboot the machine for changes to take effect.*

4.2. Configuring network-scripts

A typical interface network-script (e.g., eth0) on RHEL 6.X looks like the following:

```
# file: /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
HWADDR=00:30:48:32:6A:AA
ONBOOT="yes"
NM_CONTROLLED="no"
BOOTPROTO="static"
IPADDR=10.192.167.111
NETMASK=255.255.240.0
```

Note *On earlier versions of RHEL the NETMASK attribute is named IPMASK. Make sure you are using the right attribute name.*

In the case of DHCP addressing the last two lines should be removed and `BOOTPROTO="static"` should be changed to `BOOTPROTO="dhcp"`. The `ifcfg-ethX` files have to be created manually. They are required for bringing the interfaces up and down and attribute the desired IP addresses.

4.3. Creating network-scripts

To spot the new interfaces, make sure the driver is unloaded first. To that point `ifconfig -a | grep HWaddr` should display all non-chelsio interfaces whose drivers are loaded, whether the interfaces are up or not.

```
[root@host~]# ifconfig -a | grep HWaddr
eth0 Link encap:Ethernet HWaddr 00:30:48:32:6A:AA
```

Then load the driver using the modprobe cxgb4 command (for the moment it does not make any difference whether we are using NIC-only or the TOE-enabling driver). The output of ifconfig should display the adapter interfaces as:

```
[root@host~]# ifconfig -a | grep HWaddr
eth0 Link encap:Ethernet HWaddr 00:30:48:32:6A:AA
eth1 Link encap:Ethernet HWaddr 00:07:43:04:6B:E9e
eth2 Link encap:Ethernet HWaddr 00:07:43:04:6B:F1
```

For each interface you can write a configuration file in `/etc/sysconfig/network-scripts`. The `ifcfg-eth1` could look like:

```
# file: /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE="eth1"
HWADDR=00:07:43:04:6B:E9
ONBOOT="yes"
NM_CONTROLLED="no"
BOOTPROTO="static"
IPADDR=10.192.167.112
NETMASK=255.255.240.0
```

From now on, the `eth1` interface of the adapter can be brought up and down through the `ifup eth1` and `ifdown eth1` commands respectively. Note that it is of course not compulsory to create a configuration file for every interface if you are not planning to use them all.

4.4. Configuring IPv6

The interfaces should come up with a link-local IPv6 address for complete and fully functional IPv6 configuration. Update the Interface network-script with `ONBOOT="yes"`.

4.5. Checking Link

Once the network-scripts are created for the interfaces you should check the link i.e. make sure it is actually connected to the network. First, bring up the interface you want to test using `ifup eth1`.

You should now be able to ping any other machine from your network provided it has ping response enabled.

5. Performance Tuning

The following section lists the steps to tune the system for optimal performance.

5.1. Generic

- Install the adapter into a PCIe Gen3 x8/x16 slot. Ensure that T6 100G adapters are placed in x16 slots and not in x8_in_x16 slots.
- Ensure that the system is populated with balanced memory configuration (Please refer the system manual for the recommended configurations). Atleast one DIMM per channel should be populated for maximum performance.

- BIOS settings:

i. Disable virtualization, c-state technology, VT-d, Intel I/O AT and SR-IOV.
ii. CPU Power setting to Performance.

- Turn off *irqbalance.*

```
[root@host~]# /etc/init.d/irqbalance stop
```

(or)

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

5.2. Throughput

In addition to the generic settings,

- Add *intel_pstate=disable processor.max_cstate=1 intel_idle.max_cstate=0* to the kernel command line to prevent the system from entering power-saving/idle states and avoid CPU frequency changes.

- Set the below tuned-adm profile:

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

5.3. Latency

In addition to the generic settings,

- Disable Hyperthreading in BIOS.
- Add *idle=poll* to the kernel command line.

- Disable SELinux.
- Set the below tuned-adm profile:

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

- Disable few services.

```
[root@host~]# t4_latencytune.sh <interface>
```

- Set sysctl param *net.ipv4.tcp_low_latency* to 1.

```
[root@host~]# sysctl -w net.ipv4.tcp_low_latency=1
```

To optimize your system for different protocols, please refer to their respective chapters.

6. Software/Driver Update

For any distribution-specific problems, please check README and Release Notes included in the release for possible workaround.

Please visit Chelsio Download Center for regular updates on various software/drivers. You can also subscribe to our newsletter for the latest software updates.

7. Software/Driver Uninstallation

Similar to installation, the Chelsio Unified Wire package can be uninstalled using two main methods: from the source and RPM, based on the method used for installation. If you decide to use source, you can uninstall the package using CLI or GUI mode.

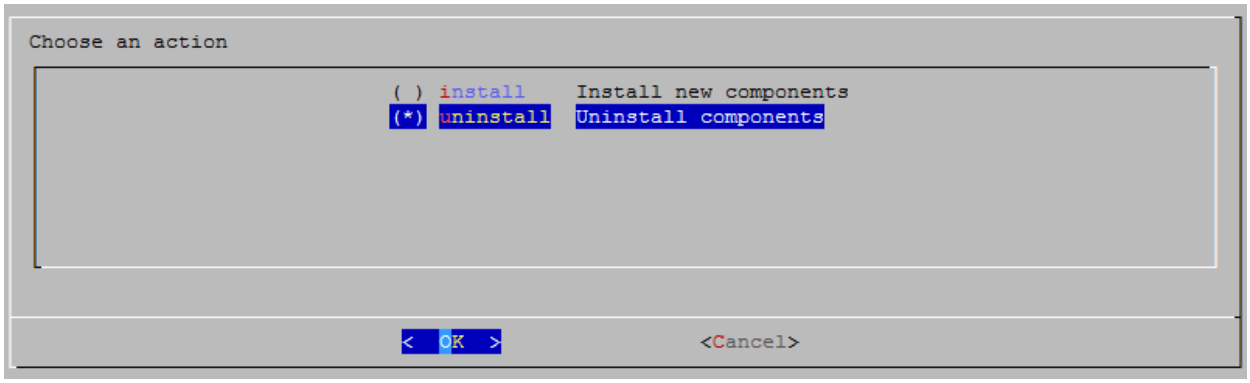
7.1. Uninstalling Chelsio Unified Wire from source

7.1.1. GUI mode (with Dialog utility)

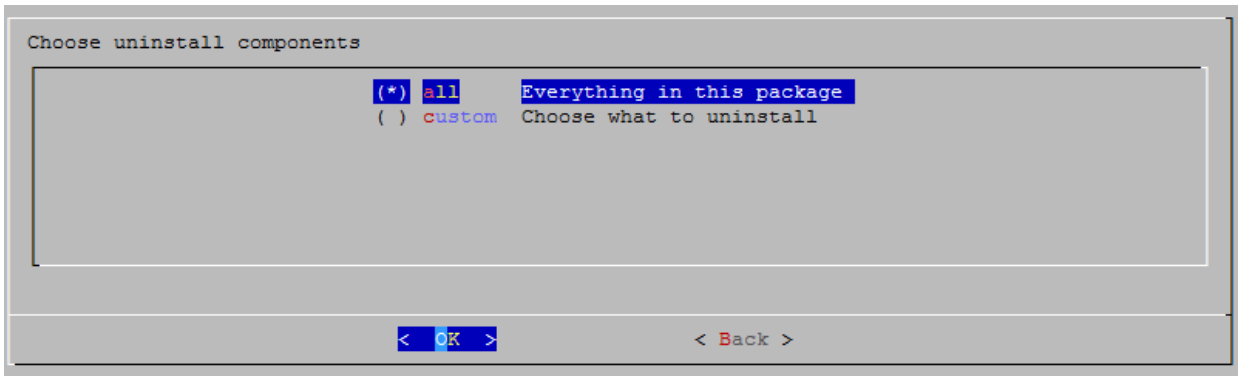
i. Change your current working directory to Chelsio Unified Wire package directory and run the following script to start the GUI installer:

```
[root@host~]# ./install.py
```

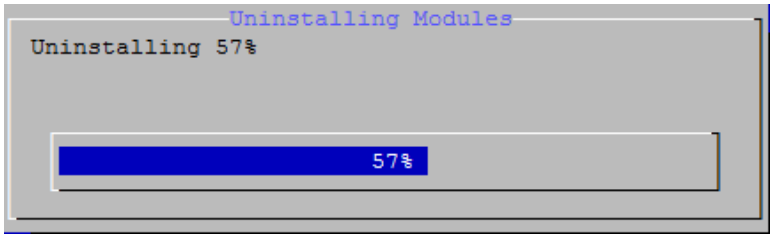
ii. Select “uninstall” , Under “Choose an action”.



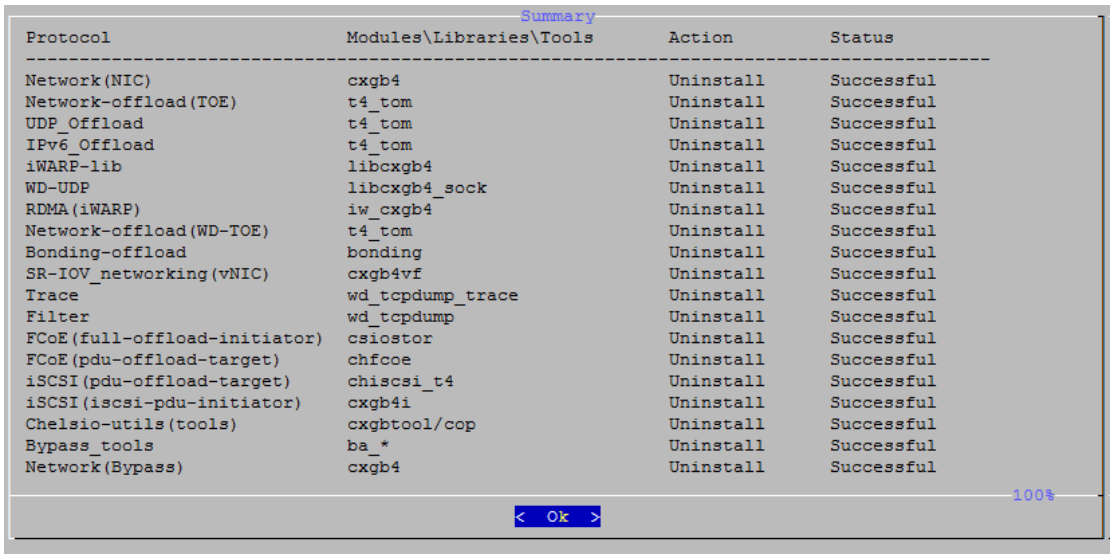
iii. Select “all” to uninstall all the installed drivers, libraries and tools or select “custom” to remove specific components.



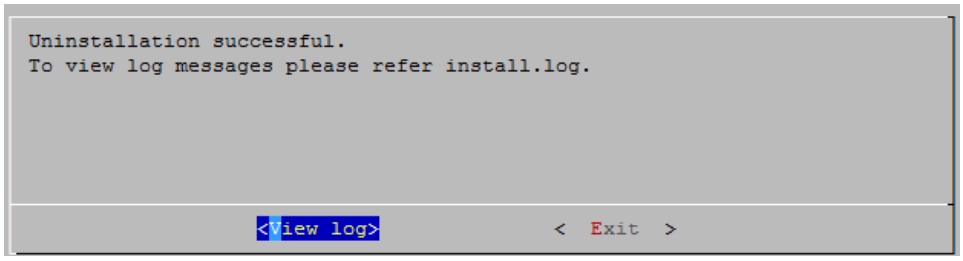
iv. The selected components will now be uninstalled.



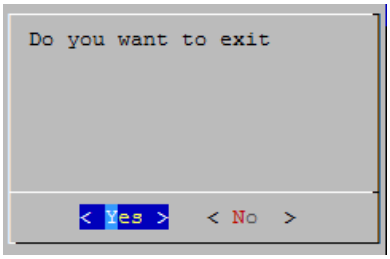
v. After successful uninstalltion, summary of the uninstalled components will be displayed.



vi. Select “View log” to view uninstallation log or “Exit” to continue.



vii. Select “Yes” to exit the installer or “No” to go back.



Note *Press Esc or Ctrl+C to exit the installer at any point of time.*

7.1.2. CLI mode (without Dialog utility)

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Run the following script with `-u` option to uninstall the Unified Wire Package:

```
[root@host~]# ./install.py -u <target>
```

Note *View help by typing* `[root@host~]# ./install.py -h` for more information

7.1.3. iWARP driver uninstallation on Cluster nodes

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Uninstall iWARP drivers on multiple Cluster nodes.

```
[root@host~]# ./install.py -C -m <machinefilename> -u all
```

The above command will remove Chelsio iWARP (*iw_cxgb4*) and TOE (*t4_tom*) drivers from all the nodes listed in the *machinefilename* file.

7.1.4. CLI mode

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Uninstall using the following command:

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

7.1.5. CLI mode (individual drivers/software)

You can also choose to uninstall drivers/software individually. Provided here are steps to uninstall few of them. For the complete list, view help by running `make help`

Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

- To uninstall NIC driver,

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

- To uninstall offload driver,

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

- To uninstall iWARP driver,

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

- To uninstall UDP Segmentation Offload driver,

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

7.2. Uninstalling Chelsio Unified Wire from RPM

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x-<OS>-<arch>
```

ii. Uninstall Unified Wire.

```
[root@host~]# ./uninstall.py
```

7.2.1. iWARP driver uninstallation on Cluster nodes

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x-<OS>-<arch>
```

ii. Uninstall iWARP drivers on multiple Cluster nodes.

```
[root@host~]# ./install.py -C -m <machinefilename> -u
```

The above command will remove Chelsio iWARP (*iw_cxgb4*) and TOE (*t4_tom*) drivers from all the nodes listed in the *machinefilename* file.

II. Network (NIC/TOE)

1. Introduction

Chelsio's Unified Wire adapters provide extensive support for NIC operation, including all stateless offload mechanisms for both IPv4 and IPv6 (IP, TCP and UDP checksum offload, LSO - Large Send Offload aka TSO - TCP Segmentation Offload, and assist mechanisms for accelerating LRO - Large Receive Offload).

A high performance fully offloaded and fully featured TCP/IP stack meets or exceeds software implementations in RFC compliance. Chelsio's Terminator engine provides unparalleled performance through a specialized data flow processor implementation and a host of features designed for high throughput and low latency in demanding conditions and networking environments.TCP offload is fully implemented in the hardware, thus freeing the CPU from TCP/IP overhead. The freed CPU can be used for any computing needs. The TCP offload in turn removes network bottlenecks and enables applications to take full advantage of the networking capabilities.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T62100-SO-CR*
- T61100-OCP*
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T6225-SO-CR (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T580-CR
- T580-LP-CR
- T580-SO-CR*
- T580-OCP-SO*
- T540-CR
- T540-LP-CR
- T540-SO-CR*
- T540-BT
- T520-CR
- T520-LL-CR
- T520-SO-CR*
- T520-OCP-SO*
- T520-BT

**Only NIC driver supported.*

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the Network driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 3.10.0-957.el7.ppc64le (POWER8 LE)
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)
- RHEL 7.5, 3.10.0-862.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Installation

Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

- To build and install NIC only driver (without offload support),

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

- To build and install drivers with offload support,

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

The driver must be loaded by the root user. Any attempt to load the driver as a regular user will fail.

3.1. Loading in NIC mode (without full offload support)

To load the Network driver without full offload support,

```
[root@host~]# modprobe cxgb4
```

3.2. Loading in TOE mode (with full offload support)

To enable full offload support,

```
[root@host~]# modprobe t4_tom
```

Note *Offload support needs to be enabled upon each reboot of the system. This can be done manually as shown above.*

In VMDirect Path environment, it is recommended to load the offload driver using the following command:

```
[root@host~]# modprobe t4_tom vmdirectio=1
```

4. Software/Driver Configuration

4.1. Enabling TCP Offload

Load the offload drivers and bring up the Chelsio interface.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig ethX <IP> up
```

All TCP traffic will be offloaded over the Chelsio interface now. To see the number of connections offloaded, run the following command:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<bus-id>/tids
```

```
[root@ ~]# cat /sys/kernel/debug/cxgb4/0000\:01\:00.4/tids
Connections in use: 8
TID range: 0..959/2048..18431, in use: 0/8
STID range: 960..1455, in use-IPv4/IPv6: 0/0
ATID range: 0..8191, in use: 0
FTID range: 1472..1967
UOTID range: 18432..19455, in use: 0
HW TID usage: 8 IP users, 0 IPv6 users
```

Where,

TID is the number of offload connections.
STID is the number of offload servers.

4.2. Enabling Busy waiting

Busy waiting/polling is a technique where a process repeatedly checks to see if an event has occurred, by spinning in a tight loop. By making use of similar technique, Linux kernel provides the ability for the socket layer code to poll directly on an Ethernet device's Rx queue. This eliminates the cost of interrupts and context switching, and with proper tuning allows to achieve latency performance similar to that of hardware.

Chelsio's NIC and TOE drivers support this feature and can be enabled on Chelsio supported devices to attain improved latency.

To make use of BUSY_POLL feature, follow the steps mentioned below:

i. Enable BUSY_POLL support in kernel config file by setting `CONFIG_NET_RX_BUSY_POLL=y`

ii. Enable BUSY_POLL globally in the system by setting the values of following sysctl parameters depending on the number of connections:

```
sysctl -w net.core.busy_read=<value>
sysctl -w net.core.busy_poll=<value>
```

Set the values of the above parameters to 50 for 100 or less connections; and 100 for more than 100 connections.

Note *BUSY_POLL can also be enabled on a per-connection basis by making use of SO_BUSY_POLL option in the socket application code. Refer socket man-page for more details.*

4.3. Precision Time Protocol (PTP)

Precision Time Protocol (PTP) standard defines a protocol for precise synchronization of clock between master and slave devices in a local area network. It can provide timing accuracies in nanosecond units. The protocol is based on time stamping and measuring the send and receive times. Most of the implementation relies on time stamping of the packets in the software which reduces the accuracy of the time measured. One possible solution to this problem is time stamping the packet in the NIC hardware itself.

Chelsio's Terminator hardware provides many features to support PTP implementations:

- High precision timers which can be read through PIO registers.
- Wall clock time based on the time of the day.
- Time stamping of selected PTP packets on both ingress and egress direction.

Important *This feature is not supported on RHEL6.X platforms.*

4.3.1. Synchronizing Clocks

ptp4l tool (installed during Unified Wire installation) is used to synchronise clocks.

i. Load the network driver on all master and slave nodes.

```
[root@host~]# modprobe cxgb4
```

ii. Assign IP addresses and ensure that master and slave nodes are connected.
iii. Start the *ptp4l* tool on master using the Chelsio interface.

```
[root@host~]# ptp4l -i <interface> -H -m
```

```
[root@ ~]# ptp4l -i enp1s0f4 -H -m
ptp4l[16681.046]: selected /dev/ptp4 as PTP clock
ptp4l[16681.054]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[16681.054]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[16681.055]: port 1: link up
ptp4l[16688.483]: port 1: LISTENING to MASTER on ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES
ptp4l[16688.483]: selected best master clock 000743.fffe.293be0
ptp4l[16688.483]: assuming the grand master role
```

iv. Start the *ptp4l* tool on slave nodes.

```
[root@host~]# ptp4l -i <interface> -H -m -s
```

Note *To view the complete list of available options, refer ptp4l help manual.*

```
[root@ ~]# ptp4l -i enp6s0f4 -m -H -s
ptp4l[13393.931]: selected /dev/ptp3 as PTP clock
ptp4l[13393.939]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[13393.940]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[13394.360]: port 1: new foreign master 000743.fffe.293be0-1
ptp4l[13398.360]: selected best master clock 000743.fffe.293be0
ptp4l[13398.360]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[13399.362]: master offset       5597 s0 freq  -26920 path delay       159
ptp4l[13400.362]: master offset       5643 s2 freq  -26874 path delay       159
ptp4l[13400.363]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[13401.362]: master offset       5516 s2 freq  -21358 path delay       159
ptp4l[13402.362]: master offset      -7045 s2 freq  -32264 path delay      3370
ptp4l[13403.362]: master offset       1897 s2 freq  -25436 path delay      3370
ptp4l[13404.362]: master offset        992 s2 freq  -25772 path delay      1801
ptp4l[13405.362]: master offset        398 s2 freq  -26068 path delay       427
ptp4l[13406.362]: master offset      -1038 s2 freq  -27385 path delay       395
ptp4l[13407.362]: master offset       -248 s2 freq  -26906 path delay       318
ptp4l[13408.362]: master offset       -209 s2 freq  -26941 path delay       318
ptp4l[13409.362]: master offset        -92 s2 freq  -26887 path delay       237
ptp4l[13410.362]: master offset       -233 s2 freq  -27056 path delay       237
ptp4l[13411.362]: master offset        -40 s2 freq  -26933 path delay       237
ptp4l[13412.363]: master offset        -10 s2 freq  -26915 path delay       237
ptp4l[13413.363]: master offset         93 s2 freq  -26815 path delay       175
ptp4l[13414.363]: master offset        -46 s2 freq  -26926 path delay       175
ptp4l[13415.363]: master offset         -3 s2 freq  -26897 path delay       169
ptp4l[13416.363]: master offset       -144 s2 freq  -27038 path delay       169
ptp4l[13417.363]: master offset         61 s2 freq  -26877 path delay       169
ptp4l[13418.363]: master offset        -68 s2 freq  -26987 path delay       171
```

v. Synchronize the system clock to a PTP hardware clock (PHC) on slave nodes.

```
[root@host~]# phc2sys -s <interface> -c CLOCK_REALTIME -w -m
```

```
[root@ ~]# phc2sys -s enp6s0f4 -c CLOCK_REALTIME -w -m
phc2sys[13406.672]: phc offset 36493387467 s0 freq  +29332 delay 1086689
phc2sys[13407.679]: phc offset 36493338184 s1 freq  -19723 delay 1084486
phc2sys[13408.684]: phc offset       1346 s2 freq  -18377 delay 1085776
phc2sys[13409.690]: phc offset      -2051 s2 freq  -21370 delay 1077105
phc2sys[13410.695]: phc offset       2070 s2 freq  -17864 delay 1078811
phc2sys[13411.701]: phc offset       5037 s2 freq  -14276 delay 1085488
phc2sys[13412.707]: phc offset      -2483 s2 freq  -20285 delay 1078173
phc2sys[13413.712]: phc offset        171 s2 freq  -18376 delay 1079112
phc2sys[13414.718]: phc offset        949 s2 freq  -17547 delay 1079990
phc2sys[13415.723]: phc offset      -1293 s2 freq  -19504 delay 1076567
phc2sys[13416.729]: phc offset     -15208 s2 freq  -33807 delay 1045916
phc2sys[13417.735]: phc offset      15711 s2 freq   -7450 delay 1076534
phc2sys[13418.740]: phc offset       5199 s2 freq  -13249 delay 1076439
phc2sys[13419.746]: phc offset       2017 s2 freq  -14871 delay 1080000
```

4.4. VXLAN Offload

Virtual Extensible LAN (VXLAN) is a network virtualization technique that uses overlay encapsulation protocol to provide Ethernet Layer 2 network services with extended scalability and flexibility. VXLAN extends the virtual LAN (VLAN) address space by adding a 24-bit segment ID and increasing the number of available logical networks from 4096 to 16 million, thereby addressing the scalability and network segmentation issues associated with large cloud computing deployments. Chelsio adapters are uniquely capable of offloading the processing of VXLAN encapsulated frames such that all stateless offloads (checksums and TSO) are preserved, resulting in significant performance benefits. This is enabled by default on loading the driver.

4.4.1. Host Configuration

i. Load the network driver and vxlan driver.

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe vxlan
```

ii. Configure larger MTU on the Chelsio interface to accommodate the larger frame size due to VXLAN encapsulation. Assign IP address and bring it up.

```
[root@host~]# ifconfig <interface> <IP address> mtu 1600 up
```

iii. Create the VXLAN interface, with the required VNI, multicast group, port number and flags.

IPv4

```
[root@host~]# ip link add <vxlan_interface> type vxlan id <vni> group
239.1.1.1 dev <interface> dstport 4789 noudpcsum
```

IPv6

```
[root@host~]# ip link add <vxlan_interface> type vxlan id <vni> group
ff08::114 dev <interface> dstport 4789 udp6zerocsumtx udp6zerocsumrx
```

iv. Bring up the VXLAN interface.

```
[root@host~]# ifconfig <vxlan_interface> up
```

v. Create the bridge interface and bring it up.

```
[root@host~]# brctl addbr <bridge_interface>
[root@host~]# ifconfig <bridge_interface> up
```

vi. Add the VXLAN interface to the bridge interface.

```
[root@host~]# brctl addif <bridge_interface> <vxlan_interface>
```

vii. Tx UDP Tunnel Segmentation Offload will be enabled by default on loading the network driver. To see the current settings,

```
[root@host~]# ethtool -k <interface>
...
tx-udp_tnl-segmentation: on
```

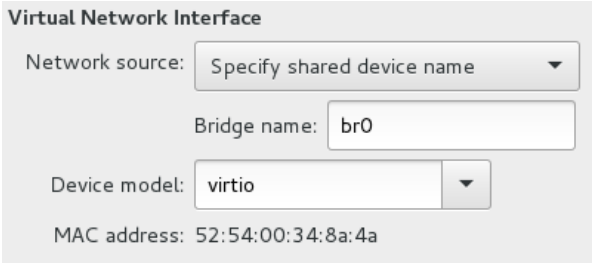
viii. For better performance, please configure the NIC settings of the Performance Tuning section.

4.4.2. Guest (VM) Configuration

i. Open the Virtual Machine Manager.

```
[root@host~]# virt-manager
```

ii. Add a Virtual Network Interface to the VM, by specifying the Bridge name configured in Step iv. of the Host Configuration section and Device Model as *virtio*.



iii. Bring up the Virtual Network interface with the required IP address.

```
[root@host~]# ifconfig <virtual-interface> <IP address> up
```

For better performance, the following settings are recommended:

i. Increase the number of queues for the Virtual network interface to 8.

```
[root@host~]# virsh edit <VM>
 </interface>
  <interface type='bridge'>
    <mac address='52:54:00:34:8a:4a'/>
    <source bridge='br0'/>
    <model type='virtio'/>
    <driver name='vhost' queues='8'/>
```

ii. Map the Virtual CPUs of the VM to physical CPUs which will be free.
Example: On a machine with 16 cores, VM Virtual CPUs were pinned to physical cores 8-15, leaving cores 0-7 to be utilized by the host.

```
[root@host~]# virsh edit <VM>
    <vcpu placement='static' cpuset='8-15'>8</vcpu>
```

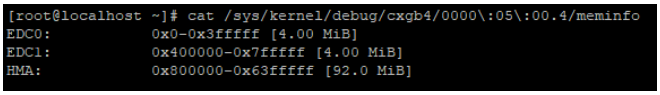
iii. Restart the libvirtd services and Virtual Machine Manager.

```
[root@host~]# systemctl restart libvirtd.service
[root@host~]# systemctl restart libvirt-guests.service
[root@host~]# virt-manager
```

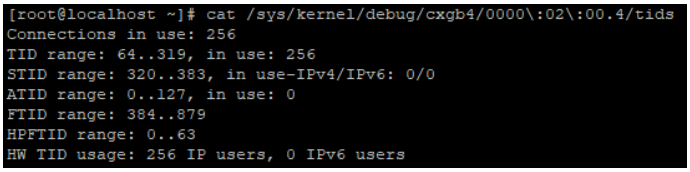
iv. Bind the Virtual Network Interface Queues to different CPUs.
v. Increase the TCP buffers by configuring the sysctl variables mentioned in NIC settings of Performance Tuning section.

4.5. HMA

To use HMA, please ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option. Currently 256 IPv4/128 IPv6 TOE connections are supported on T6 25G SO adapters. The following image shows the HMA reserved memory.



The following image shows the number of TOE offloaded connections.



4.6. Performance Tuning

Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.

- **TOE**

i. Run the performance tuning script to map TOE queues to different CPUs.

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

ii. Set the following *sysctl* parameters:

```
[root@host~]# sysctl -w net.ipv4.tcp_timestamps=0
[root@host~]# sysctl -w net.core.netdev_max_backlog=250000
[root@host~]# sysctl -w net.core.rmem_max=4194304
[root@host~]# sysctl -w net.core.wmem_max=4194304
[root@host~]# sysctl -w net.core.rmem_default=4194304
[root@host~]# sysctl -w net.core.wmem_default=4194304
[root@host~]# sysctl -w net.ipv4.tcp_rmem="4096 1048576 4194304"
[root@host~]# sysctl -w net.ipv4.tcp_wmem="4096 1048576 4194304"
```

iii. Disable Rx Coalesce and DDP using the following steps:

a. Create a COP policy.

```
[root@host~]# cat <policy_file>
all => offload !ddp !coalesce
```

b. Compile the policy.

```
[root@host~]# cop -d -o <policy_out> <policy_file>
```

c. Apply the policy.

```
[root@host~]# cxgbtool ethX policy <policy_out>
```

Note *The policy applied using cxgbtool is not persistent and should be applied every time drivers are reloaded or the machine is rebooted.*

The applied cop policies can be read using,

```
[root@host~]# cat /proc/net/offload/toeX/read-cop
```

- **NIC**

i. Run the performance tuning script to map NIC queues to different CPUs.

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

ii. Enable adaptive-rx.

```
[root@host~]# ethtool -C enp2s0f4 adaptive-rx on
```

iii. Set the following sysctl parameters.

```
[root@host~]# sysctl -w net.ipv4.tcp_timestamps=0
[root@host~]# sysctl -w net.core.netdev_max_backlog=250000
[root@host~]# sysctl -w net.core.rmem_max=4194304
[root@host~]# sysctl -w net.core.wmem_max=4194304
[root@host~]# sysctl -w net.core.rmem_default=4194304
[root@host~]# sysctl -w net.core.wmem_default=4194304
[root@host~]# sysctl -w net.ipv4.tcp_rmem="4096 1048576 4194304"
[root@host~]# sysctl -w net.ipv4.tcp_wmem="4096 1048576 4194304"
```

- **NIC Latency**

Enable BUSY_POLL feature.

```
[root@host~]# sysctl -w net.core.busy_poll=50
[root@host~]# sysctl -w net.core.busy_read=50
```

- **TOE Latency**

Set the below sysctl:

```
[root@host~]# sysctl -w toe.toeX_tom.recvmsg_spin_us=50
```

- **Receiver Side Scaling (RSS)**

Receiver Side Scaling enables the receiving network traffic to scale with the available number of processors on a modern networked computer. RSS enables parallel receive processing and dynamically balances the load among multiple processors. Chelsio's network controller fully supports Receiver Side Scaling for IPv4 and IPv6.

This script first determines the number of CPUs on the system and then each receiving queue is bound to an entry in the system interrupt table and assigned to a specific CPU. Thus, each receiving queue interrupts a specific CPU through a specific interrupt now. For example, on a 4-core system, `t4_perftune.sh` gives the following output:

```
[root@host~]# t4_perftune.sh
Discovering Chelsio T4/T5 devices ...
Configuring Chelsio T4/T5 devices ...
Tuning eth7
IRQ table length 4
Writing 1 in /proc/irq/62/smp_affinity
Writing 2 in /proc/irq/63/smp_affinity
Writing 4 in /proc/irq/64/smp_affinity
Writing 8 in /proc/irq/65/smp_affinity
eth7 now up and tuned
...
```

Because there are 4 CPUs on the system, 4 entries of interrupts are assigned. For other network interfaces, you should see similar output message.

Now the receiving traffic is dynamically assigned to one of the system's CPUs through a Terminator queue. This achieves a balanced usage among all the processors. This can be verified, for example, by using the **iperf** tool. First set up a server on the receiver host.

```
[root@receiver_host~]# iperf –s
```

Then on the sender host, send data to the server using the iperf client mode. To emulate a moderate traffic workload, use *-P* option to request 20 TCP streams from the server.

```
[root@sender_host~]# iperf -c receiver_host_name_or_IP -P 20
```

Then on the receiver host, look at interrupt rate at `/proc/interrupts`:

```
[root@receiver_host~]# cat /proc/interrupts | grep eth6

Id      CPU0      CPU1      CPU2      CPU3       type              interface
36:   115229         0         0         1   PCI-MSI-edge      eth6 (queue 0)
37:        0    121083         1         0   PCI-MSI-edge      eth6 (queue 1)
38:        0         0    105423         1   PCI-MSI-edge      eth6 (queue 2)
39:        0         0         0    115724   PCI-MSI-edge      eth6 (queue 3)
```

Now interrupts from eth6 are evenly distributed among the 4 CPUs.

Without Terminator's RSS support, the interrupts caused by network traffic may be distributed unevenly over CPUs. For your information, the traffic produced by the same iperf commands gives the following output in `/proc/interrupts`.

```
[root@receiver_host~]# cat /proc/interrupts | grep eth6

Id      CPU0      CPU1      CPU2      CPU3       type              interface
36:        0         9         0     17418   PCI-MSI-edge      eth6 (queue 0)
37:        0         0     21718      2063   PCI-MSI-edge      eth6 (queue 1)
38:        0         7    391519       222   PCI-MSI-edge      eth6 (queue 2)
39:        1         0        33     17798   PCI-MSI-edge      eth6 (queue 3)
```

Here there are 4 receiving queues from the eth6 interface, but they are not bound to a specific CPU or interrupt entry. Queue 2 has caused a very large number of interrupts on CPU2 while CPU0 and CPU1 are barely used by any of the four queues. Enabling RSS is thus essential for best performance.

Note

Linux's `irqbalance` *may take charge of distributing interrupts among CPUs on a multiprocessor platform. However,* `irqbalance` *distributes interrupt requests from all hardware devices across processors. For a server with Chelsio network card constantly receiving large volume of data at 40/10Gbps, the network interrupt demands are significantly high. Under such circumstances, it is necessary to enable RSS to balance the network load across multiple processors and achieve the best performance.*

- **Interrupt Coalescing**

The idea behind Interrupt Coalescing (IC) is to avoid flooding the host CPUs with too many interrupts. Instead of throwing one interrupt per incoming packet, IC waits for 'n' packets to be available in the Rx queues and placed into the host memory through DMA operations before an interrupt is thrown, reducing the CPU load and thus improving latency. It can be changed using the following command:

```
[root@host~]# ethtool -C ethX rx-frames n
```

Note

For more information, run the following command:

```
[root@host~]# ethtool -h
```

- **Large Receive Offload / Generic Receive Offload**

Large Receive Offload or Generic Receive Offload is a performance improvement feature at the receiving side. LRO/GRO aggregates the received packets that belong to same stream, and combines them to form a larger packet before pushing them to the receive host network stack. By doing this, rather than processing every small packet, the receiver CPU works on fewer packet headers but with same amount of data. This helps reduce the receive host CPU load and improve throughput in a 40/10Gb network environment where CPU can be the bottleneck.

LRO and GRO are different names to refer to the same receiver packets aggregating feature. LRO and GRO actually differ in their implementation of the feature in the Linux kernel. The feature was first added into the Linux kernel in version 2.6.24 and named Large Receive Offload (LRO). However, LRO only works for TCP and IPv4. As from kernel 2.6.29, a new protocol-independent implementation removing the limitation is added to Linux, and it is named Generic Receive Offload (GRO). The old LRO code is still available in the kernel sources but whenever both GRO and LRO are presented GRO is always the preferred one to use.

Please note that if your Linux system has IP forwarding enabled, i.e. acting as a bridge or router, the LRO needs to be disabled. This is due to a known kernel issue.

Chelsio's card supports both hardware assisted GRO/LRO and Linux-based GRO/LRO. `t4_tom` is the kernel module that enables the hardware assisted GRO/LRO. If it is not already in the kernel module list, use the following command to insert it:

```
[root@host~]# lsmod | grep t4_tom
[root@host~]# modprobe t4_tom
[root@host~]# lsmod | grep t4_tom
t4_tom 88378 0 [permanent]
toecore 21618 1 t4_tom
cxgb4 225342 1 t4_tom
```

Then Terminator's hardware GRO/LRO implementation is enabled.

If you would like to use the Linux GRO/LRO for any reason, first the `t4_tom` kernel module needs to be removed from kernel module list. Please note you might need to reboot your system. After removing the `t4_tom` module, you can use `ethtool` to check the status of current GRO/LRO settings, for example:

```
[root@host~]# ethtool -k eth6
Offload parameters for eth6:
rx-checksumming: on
tx-checksumming: on
scatter-gather: on
tcp-segmentation-offload: on
udp-fragmentation-offload: off
generic-segmentation-offload: on
generic-receive-offload: on
large-receive-offload: off
```

Now the `generic-receive-offload` option is on. This means GRO is enabled. Please note that there are two offload options here: `generic-receive-offload` and `large-receive-offload`. This is because on this Linux system (RHEL 6.0), the kernel supports both GRO and LRO. As mentioned earlier, GRO is always the preferred option when both are present. On other systems LRO might be the only available option. Then `ethtool` could be used to switch LRO on and off as well. When GRO is enabled, Chelsio's driver provides the following GRO-related statistics.

```
[root@host~]# ethtool -S eth6
...
GROPackets : 0
GROMerged : 897723
...
```

`GROPackets` is the number of held packets. Those are candidate packets held by the kernel to be processed individually or to be merged to larger packets. This number is usually zero. `GROMerged` is the number of packets that merged to larger packets. Usually this number increases if there is any continuous traffic stream present.

`ethtool` can also be used to switch off the GRO/LRO options when necessary.

```
[root@host~]# ethtool -K eth6 gro off
[root@host~]# ethtool -k eth6
Offload parameters for eth6:
rx-checksumming: on
tx-checksumming: on
scatter-gather: on
tcp-segmentation-offload: on
udp-fragmentation-offload: off
generic-segmentation-offload: on
generic-receive-offload: off
large-receive-offload: off
```

The output above shows a disabled GRO.

5. Software/Driver Unloading

5.1. Unloading the NIC Driver

To unload the NIC driver,

```
[root@host~]# rmmod cxgb4
```

5.2. Unloading the TOE Driver

A reboot is required to unload the TOE driver. To avoid rebooting, follow the below steps:

i. Load *t4_tom* driver with *unsupported_allow_unload* parameter.

```
[root@host~]# modprobe t4_tom unsupported_allow_unload=1
```

ii. Stop all the offloaded traffic, servers and connections. Check for the reference count.

```
[root@host~]# cat /sys/module/t4_tom/refcnt
```

If the reference count is 0, the driver can be directly unloaded. Skip to step (iii)

If the count is non-zero, load a COP policy which disables offload using the following procedure:

a. Create a policy file which will disable offload.

```
[root@host~]# cat policy_file
all => !offload
```

b. Compile and apply the output policy file.

```
[root@host~]# cop -o no-offload.cop policy_file
[root@host~]# cxgbtool ethX policy no-offload.cop
```

iii. Unload the driver.

```
[root@host~]# rmmod t4_tom
[root@host~]# rmmod toecore
[root@host~]# rmmod cxgb4
```

III. Virtual Function Network (vNIC)

1. Introduction

The ever-increasing network infrastructure of IT enterprises has lead to a phenomenal increase in maintenance and operational costs. IT managers are forced to acquire more physical servers and other data center resources to satisfy storage and network demands. To solve the Network and I/O overhead, users are opting for server virtualization which consolidates I/O workloads onto lesser physical servers thus resulting in efficient, dynamic and economical data center environments. Other benefits of Virtualization include improved disaster recovery, server portability, cloud computing, Virtual Desktop Infrastructure (VDI), etc.

Chelsio's Unified Wire family of adapters deliver increased bandwidth, lower latency and lower power with virtualization features to maximize cloud scaling and utilization. The adapters also provide full support for PCI-SIG SR-IOV to improve I/O performance on a virtualized system. User can configure up to 64 Virtual and 8 Physical functions (with 4 PFs as SR-IOV capable) along with 336 virtual MAC addresses.

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T62100-SO-CR
- T61100-OCP
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP
- T6225-SO-CR
- T580-CR
- T580-LP-CR
- T580-SO-CR
- T580-OCP-SO
- T540-CR
- T540-LP-CR
- T540-SO-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-SO-CR
- T520-OCP-SO
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the Virtual Function Network driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Installation

The Virtual Function implementation for Chelsio adapters comprises of two modules:

- Standard NIC driver module, *cxgb4*, which runs on base Hypervisor and is responsible for instantiation and management of the PCIe Virtual Functions (VFs) on the adapter.
- VF NIC driver module, *cxgb4vf*, which runs on Virtual Machine (VM) guest OS using VFs "attached" via Hypervisor VM initiation commands.

2.1. Pre-requisites

Please make sure that the following requirements are met before installation:

- PCI Express Slot should be ARI capable.
- SR-IOV should be enabled in the machine.
- Intel Virtualization Technology for Directed I/O (VT-d) should be enabled in the BIOS.
- Add *intel_iommu=on* to the kernel command line in grub/grub2 menu, to use VFs in VMs.

2.2. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. On the host, install network driver.

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

iii. On the guest (VM), install vNIC driver.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iv. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

3.1. Instantiate Virtual Functions (SR-IOV)

To instantiate Virtual Functions (VFs) on the host, run the following commands:

```
[root@host~]# modprobe cxgb4
[root@host~]# echo n >
/sys/class/net/ethX/device/driver/<bus_id>/sriov_numvfs
```

Here, *ethX* is the interface and *n* specifies the number of VFs to be instantiated per physical function (*bus_id*). VFs can be instantiated only from PFs 0 - 3 of the Chelsio adapter. A maximum of 64 virtual functions can be instantiated with 16 virtual functions per physical function.

Example: Instantiating 16 VFs on PF3 of Chelsio adapter.

Note *To get familiar with physical and virtual function terminologies, please refer the PCI Express specification.*

Unload the vNIC driver on the host (if loaded).

```
[root@host~]# rmmod cxgb4vf
```

The virtual functions can now be assigned to virtual machines (guests).

3.2. Loading the Driver

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

The vNIC driver must be loaded on the Guest OS by the root user. Any attempt to load the driver as a regular user will fail.

To load the driver, run the following command:

```
[root@host~]# modprobe cxgb4vf
```

4. Software/Driver Configuration and Fine-tuning

4.1. VF Communication

Once the VF driver (*cxgb4vf*) is loaded in the VM and the VF interface is up with an IP address, it will be able to communicate (send/receive network traffic).

```
[root@host~]# modprobe cxgb4vf
[root@host~]# ifconfig ethX <IP Address> up
```

2-port card

VFs of PF0 and PF2 can communicate with each other and with hosts connected to Port 0.
VFs of PF1 and PF3 can communicate with each other and with hosts connected to Port 1.

4-port card

VFs of PF0 can communicate with each other and with hosts connected to Port 0.
VFs of PF1 can communicate with each other and with hosts connected to Port 1.
VFs of PF2 can communicate with each other and with hosts connected to Port 2.
VFs of PF3 can communicate with each other and with hosts connected to Port 3.

By default, the VFs (in VM) can not communicate with PFs (on Host). To enable this communication, set ethtool private flag *port_tx_vm_wr* for PF interface (on Host).

```
[root@host~]# ethtool --set-priv-flags ethX port_tx_vm_wr on
```

Example:

i. 1 VF was instantiated on PF0.

```
[root@host~]# modprobe cxgb4
[root@host~]# echo 1 >
/sys/class/net/eth1/device/driver/0000\:01\:00.0/sriov_numvfs
[root@host~]# rmmod cxgb4vf
```

ii. ethtool private flag was set on the Host and PF interface was brought up on the Host.

```
[root@host~]# ethtool --set-priv-flags eth1 port_tx_vm_wr on
[root@host~]# ifconfig eth1 10.1.1.2/24 up
```

iii. VF was assigned to a VM. VF was brought up in the VM.

```
[root@VM ~]# modprobe cxgb4vf
[root@VM ~]# ifconfig eth2 10.1.1.3/24 up
```

VF will be able to commincate with PF interface on the host.

4.2. VF Link state

VF link state depends on the physical port link status to which the VF is mapped to. Please refer the above section for VF to physical port mappings. To override this and always enable the VF link, follow the below procedure. This will enable VF to VF communication irrespective of the physical port link status.

i. After instantiating the VFs, check the current VF link state using the below command on Host (hypervisor). By default, it will be *auto.*

```
[root@host~]# ip link show mgmtpfX,Y
```

```
[root@host ~]# ip link show mgmtpf1,0
18: mgmtpf1,0: <NOARP> mtu 0 qdisc noop state DOWN mode DEFAULT group default qlen 1
    link/none
    vf 0 MAC 06:44:04:b4:e0:00, link-state auto
    vf 1 MAC 06:44:04:b4:e0:01, link-state auto
```

ii. Enable the VF link state for the required VFs.

```
[root@host~]# ip link set dev mgmtpfX,Y vf Z state enable
```

```
[root@host ~]# ip link set dev mgmtpf1,0 vf 0 state enable
[root@host ~]# ip link show mgmtpf1,0
18: mgmtpf1,0: <NOARP> mtu 0 qdisc noop state DOWN mode DEFAULT group default qlen 1
    link/none
    vf 0 MAC 06:44:04:b4:e0:00, link-state enable
    vf 1 MAC 06:44:04:b4:e0:01, link-state auto
```

iii. The VFs can then be assigned to Virtual Machines. On loading cxgb4vf driver in the VM and bringing up the VF interface, the VF will be enabled. It can then communicate with other VFs (which are enabled) irrespective of physical link.

```
[root@VM ~]# ifconfig enp7s1
enp7s1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 06:44:04:b4:e0:00  txqueuelen 1000  (Ethernet)
```

To revert to default behaviour, set the VF link state to *auto.*

```
[root@host~]# ip link set dev mgmtpfX,Y vf Z state auto
```

4.3. VF Rate Limiting

This section describes the method to rate-limit traffic passing through virtual functions (VFs).

i. The VF rate limit needs to be set on the Host (hypervisor). Apply rate-limiting using:

```
[root@host~]# ip link set dev mgmtpfXX vf <vf_number> rate <rate_in_mbps>
```

Here,

- *mgmtpfXX* is the management interface to be used. For each PF on which VFs are instantiated, 1 management interface will be created (in "ifconfig -a").
- *vf_number* is the VF on which rate-limiting is applied. Value 0-15.

ii. Run traffic over the VF (using kernel mode cxgb4vf or DPDK PMD) and the throughput should be rate-limited as per the values set in the previous step.

Example:

i. 4 VFs are instantiated on PF0.

```
[root@host~]# modprobe cxgb4
[root@host~]# echo 4 >
/sys/class/net/ethX/device/driver/<bus_id>/sriov_numvfs
```

ii. 2 VMs are configured with 2 VFs each. 2 different networks are configured with the following IP configuration:

```
VM0: VF0 (102.1.1.2/24), VF1 (102.2.2.2/24)
VM1: VF2 (102.1.1.3/24), VF3 (102.2.2.3/24)
```

iii. VF Rate-limiting is configured on the host.

```
[root@host~]# ip link set dev mgmtpf10 vf 0 rate 2000
[root@host~]# ip link set dev mgmtpf10 vf 1 rate 3000
```

The traffic on 102.1.1.X network will be rate-limited to 2Gbps whereas traffic on 102.2.2.X network will be rate-limited to 3Gbps.

4.4. Bonding

The VF network interfaces (assigned to a VM) can be aggregated into a single logical bonded interface effectively combining the bandwidth into a single connection. It also provides redundancy in case one of the link fails. Execute the following steps in the VM (attached with more than 1 VF interface):

i. Load the Virtual Function network driver.

```
[root@host~]# modprobe cxgb4vf
```

ii. Create a bond interface.

```
[root@host~]# modprobe bonding mode=<bonding mode> <optional parameters>
```

iii. Bring up the bond interface and enslave the VF interfaces to the bond.

```
[root@host~]# ifconfig bond0 up
[root@host~]# ifenslave bond0 ethX ethY
```

Note *ethX and ethY are the VF interfaces attached to the same VM. It is recommended to use VFs of different Ports to achieve redundancy in case of link failures.*

iv. Assign IPv4/IPv6 address to the bond interface.

```
[root@host~]# ifconfig bond0 X.X.X.X/Y
[root@host~]# ifconfig bond0 inet6 add <128-bit IPv6 Address> up
```

Example:

i. 2 VFs are instantiated each on PF0 (Port 0) and PF1 (Port 1) on the host.

```
[root@host~]# modprobe cxgb4
[root@host~]# echo 2 >
/sys/class/net/eth4/device/driver/0000\:01\:00.0/sriov_numvfs
[root@host~]# echo 2 >
/sys/class/net/eth4/device/driver/0000\:01\:00.1/sriov_numvfs
```

ii. 1 VM was configured with VF0 of PF0 and VF1 of PF1.

```
[root@host~]# modprobe cxgb4vf force_link_up=0
[root@host~]# ifconfig enp8s1
enp8s1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
ether 06:44:3c:a8:40:00  txqueuelen 1000  (Ethernet)
[root@host~]# ifconfig enp8s1f5d1
enp8s1f5d1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
ether 06:44:3c:a8:40:11  txqueuelen 1000  (Ethernet)
```

iii. Bonding mode=1 was configured in the VM.

```
[root@host~]# modprobe bonding mode=1 miimon=100
[root@host~]# ifconfig bond0 up
[root@host~]# ifenslave bond0 enp8s1 enp8s1f5d1
[root@host~]# ifconfig bond0 10.1.1.223/24
```

The traffic will run over the bond interface in Active-Backup mode. If the link fails on enp8s1, the traffic will failver to enp8s1f5d1.

4.5. High Capacity VF Configuration

Chelsio adapters by default support 16 VFs per PF. In order to use more VFs per PF, please follow the below steps on the host:

Important *Currently supported on T6225-SO-CR and T6225-OCP adapters.*

i. Change your current working directory to Chelsio Unified Wire package directory and install the driver.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
[root@host~]# make CONF=HIGH_CAPACITY_VF install
```

Note *For more installation options, please run* `make help` *or* `install.py -h`

ii. Update adapter configuration and reboot the machine.

```
[root@host~]# chelsio_adapter_config.py
Chelsio adapter detected
|-------------------------------------|
| Choose Chelsio card:                |
| 1. T580_SO_CR      3:0.0            |
| 2. T6225_SO        4:0.0            |
|-------------------------------------|
Select card: 2
Card T6225_SO(4:0.0) selected
|-------------------------------------|
| Choose option                       |
| 1. Change to Default settings       |
| 2. Change Adapter Config settings   |
|-------------------------------------|
Select option: 2
Changing Adapter Config settings
|-------------------------------------|
| Possible Chelsio adapter settings:  |
| 1: 248 VFs mode                     |
|-------------------------------------|
248 VF setting selected
```

iii. Instantiate virtual functions.

```
[root@host~]# modprobe cxgb4
[root@host~]# echo n >
/sys/class/net/ethX/device/driver/<bus_id>/sriov_numvfs
```

- 124 virtual functions can be instantiated on T5 adapter, with 31 virtual functions per physical function{pf 0..3}.
- 248 virtual functions can be instantiated on T6 adapter, with 62 virtual functions per physical function{pf 0..3}.

iv. Unload the vNIC driver on the host (if loaded).

```
[root@host~]# rmmod cxgb4vf
```

v. The virtual functions can now be assigned to virtual machines (guests).
vi. For each PF on which VFs are instantiated, 1 management interface (mgmtpfX,Y) will be created. You can see them using *ip link show* command.

```
[root@host ~]# ip link show
14: mgmtpf1,0: <NOARP> mtu 0 qdisc noop state DOWN mode DEFAULT qlen 1
link/none
vf 0 MAC 06:44:3c:b1:00:00, link-state auto
15: mgmtpf1,1: <NOARP> mtu 0 qdisc noop state DOWN mode DEFAULT qlen 1
link/none
vf 0 MAC 06:44:3c:b1:80:10, link-state auto
16: mgmtpf1,2: <NOARP> mtu 0 qdisc noop state DOWN mode DEFAULT qlen 1
link/none
vf 0 MAC 06:44:3c:b1:80:20, link-state auto
17: mgmtpf1,3: <NOARP> mtu 0 qdisc noop state DOWN mode DEFAULT qlen 1
link/none
vf 0 MAC 06:44:3c:b1:80:30, link-state auto
```

vii. To set a VLAN ID on Virtual Function,

```
[root@host ~]# ip link set <mgmtpfX,Y> vf <vf_index> vlan <vlan_id>
```

Example: The below command will set VLAN ID 20 to VF0 device instantiated on PF0 function.

```
[root@host ~]# ip link set mgmtpf1,0 vf 0 vlan 20
```

viii. To set a MAC address on the Virtual Function,

```
[root@host ~]# ip link set <mgmtpfX,Y> vf <vf_index> mac <vnic_mac>
```

Example:

```
[root@host ~]# ip link set mgmtpf1,0 vf 0 mac 06:44:3c:11:22:33
```

Note *The VF driver (cxgb4vf) needs to be reloaded on the VM for the new settings (VLAN or MAC address) to take effect.*

5. Software/Driver Unloading

5.1. Unloading the Driver

The vNIC driver must be unloaded on the Guest OS by the root user. Any attempt to unload the driver as a regular user will fail.

To unload the driver, execute the following command:

```
[root@host~]# rmmod cxgb4vf
```

IV. iWARP RDMA Offload

1. Introduction

Chelsio's Terminator engine implements a feature rich RDMA implementation which adheres to the IETF standards with optional markers and MPA CRC-32C.

The iWARP RDMA operation benefits from the virtualization, traffic management and QoS mechanisms provided by Terminator engine. It is possible to ACL process iWARP RDMA packets. It is also possible to rate control the iWARP traffic on a per-connection or per-class basis, and to give higher priority to QPs that implement distributed locking mechanisms. The iWARP operation also benefits from the high performance and low latency TCP implementation in the offload engine.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T6225-SO-CR (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the iWARP RDMA Offload driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64

- RHEL 7.6, 3.10.0-957.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 3.10.0-862.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work

2. Software/Driver Installation

2.1. Pre-requisites

- Uninstall any OFED present in the machine.
- *rdma-core-devel* package should be installed on RHEL 8.X/7.X systems.

2.2. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install iWARP drivers and libraries.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

3.1. Loading iWARP Driver

The driver must be loaded by the root user. Any attempt to load the driver as a regular user will fail.

To load the iWARP driver we need to load the NIC driver and core RDMA drivers first. Run the following commands:

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe iw_cxgb4
[root@host~]# modprobe rdma_ucm
```

Optionally, you can start the iWARP Port Mapper daemon to enable port mapping.

```
[root@host~]# iwpmd
```

4. Software/Driver Configuration and Fine-tuning

4.1. Testing connectivity with *ping* and *rping*

Load the NIC, iWARP & core RDMA modules as mentioned in Software/Driver Loading section. After which, you will see two or four ethernet interfaces for the Terminator device. Configure them with an appropriate ip address, netmask, etc. You can use the Linux *ping* command to test basic connectivity via the Terminator interface. To test RDMA, use the *rping* command that is included in the librdmacm-utils RPM.

Run the following command on the server machine:

```
[root@host~]# rping -s -a server_ip_addr -p 9999
```

Run the following command on the client machine:

```
[root@host~]# rping -c –Vv -C10 -a server_ip_addr -p 9999
```

You should see ping data like this on the client:

```
ping data: rdma-ping-0: ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqr
ping data: rdma-ping-1: BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrs
ping data: rdma-ping-2: CDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrst
ping data: rdma-ping-3: DEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstu
ping data: rdma-ping-4: EFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuv
ping data: rdma-ping-5: FGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
ping data: rdma-ping-6: GHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwx
ping data: rdma-ping-7: HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxy
ping data: rdma-ping-8: IJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
ping data: rdma-ping-9: JKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyzA
client DISCONNECT EVENT...
#
```

4.2. Enabling various MPIs

4.2.1. Setting shell for Remote Login

User needs to set up authentication on the user account on all systems in the cluster to allow user to remotely logon or executing commands without password.

Quick steps to set up user authentication:

i. Change to user home directory.

```
[root@host~]# cd
```

ii. Generate authentication key.

```
[root@host~]# ssh-keygen -t rsa
```

iii. Hit [Enter] upon prompting to accept default setup and empty password phrase.
iv. Create authorization file.

```
[root@host~]# cd .ssh
[root@host~]# cat *.pub > authorized_keys
[root@host~]# chmod 600 authorized_keys
```

v. Copy directory .ssh to all systems in the cluster.

```
[root@host~]# cd
[root@host~]# scp -r /root/.ssh remotehostname-or-ipaddress:
```

4.2.2. Configuration of various MPIs (Installation and Setup)

Intel-MPI

i. Download latest Intel MPI from the Intel website.
ii. Copy the license file (.lic file) into `l_mpi_p_x.y.z` directory.
iii. Create machines.LINUX (list of node names) in `l_mpi_p_x.y.z`
iv. Select advanced options during installation and register the MPI.
v. Install software on every node.

```
[root@host~]# ./install.py
```

vi. Set IntelMPI with mpi-selector (do this on all nodes).

```
[root@host~]# mpi-selector --register intelmpi --source-dir
/opt/intel/impi/3.1/bin/
[root@host~]# mpi-selector --set intelmpi
```

vii. Edit `.bashrc` and add these lines:

```
export RSH=ssh
export DAPL_MAX_INLINE=64
export I_MPI_DEVICE=rdssm:chelsio
export MPIEXEC_TIMEOUT=180
export MPI_BIT_MODE=64
```

viii. Logout & log back in.
ix. Populate `mpd.hosts` with node names.

Note

- *The hosts in this file should be Chelsio interface IP addresses.*
- `I_MPI_DEVICE=rdssm:chelsio` *assumes you have an entry in* `/etc/dat.conf` *named* `chelsio`*.*
- `MPIEXEC_TIMEOUT` *value might be required to increase if heavy traffic is going across the systems.*

x. Contact Intel for obtaining their MPI with DAPL support.
xi. To run Intel MPI over RDMA interface, DAPL 2.0 should be set up as follows:

Enable the Chelsio device by adding an entry at the beginning of the /`etc/dat.conf` file for the Chelsio interface. For instance, if your Chelsio interface name is eth2, then the following line adds a DAT version 2.0 device named “chelsio2" for that interface:

```
chelsio2 u2.0 nonthreadsafe default libdaplofa.so.2 dapl.2.0 "eth2 0" ""
```

Open MPI (Installation and Setup)

Open MPI iWARP support is only available in Open MPI version 1.3 or greater.

Open MPI will work without any specific configuration via the openib btl. Users wishing to performance tune the configurable options may wish to inspect the receive queue values. Those can be found in the "Chelsio T4" section of `mca-btl-openib-device-params.ini`. Follow the steps mentioned below to install and configure Open MPI.

i. If not alreay done, install *mpi-selector* tool.
ii. Download the latest stable/feature version of openMPI from OpenMPI website.

iii. Untar and change your current working directory to openMPI package directory.
iv. Configure and install as:

```
[root@host~]#./configure --with-openib=/usr  CC=gcc CXX=g++ F77=gfortran
FC=gfortran --enable-mpirun-prefix-by-default --prefix=/usr/mpi/gcc/openmpi-
x.y.z/ --with-openib-libdir=/usr/lib64/ --libdir=/usr/mpi/gcc/openmpi-
x.y.z/lib64/ --with-contrib-vt-flags=--disable-iotrace
[root@host~]# make
[root@host~]# make install
```

The above step will install openMPI in */usr/mpi/gcc/openmpi-x.y.z/*

Note *To enable multithreading, add "--enable-mpi-thread-multiple" and "--with-threads=posix" parameters to the above configure command.*

v. Next, create a shell script, *mpivars.csh,* with the following entry:

```
# path
if ("" == "`echo $path | grep /usr/mpi/gcc/openmpi-x.y.z/bin`") then
   set path=(/usr/mpi/gcc/openmpi-x.y.z/bin $path)
endif


# LD_LIBRARY_PATH
if ("1" == "$?LD_LIBRARY_PATH") then
    if ("$LD_LIBRARY_PATH" !~ */usr/mpi/gcc/openmpi-x.y.z/lib64*) then
    setenv LD_LIBRARY_PATH /usr/mpi/gcc/openmpi-
x.y.z/lib64:${LD_LIBRARY_PATH}
    endif
else
   setenv LD_LIBRARY_PATH /usr/mpi/gcc/openmpi-x.y.z/lib64
endif
# MPI_ROOT
setenv MPI_ROOT /usr/mpi/gcc/openmpi-x.y.z
```

vi. Simlarly, create another shell script, *mpivars.sh*, with the following entry:

```
# PATH
if test -z "`echo $PATH | grep /usr/mpi/gcc/openmpi-x.y.z/bin`"; then
    PATH=/usr/mpi/gcc/openmpi-x.y.z/bin:${PATH}
    export PATH
fi
# LD_LIBRARY_PATH
if test -z "`echo $LD_LIBRARY_PATH | grep
/usr/mpi/gcc/openmpi-          x.y.z/lib64`"; then
   LD_LIBRARY_PATH=/usr/mpi/gcc/openmpi-   x.y.z/lib64${LD_LIBRARY_PATH:+:}$
{LD_LIBRARY_PATH}
   export LD_LIBRARY_PATH
fi


# MPI_ROOT
MPI_ROOT=/usr/mpi/gcc/openmpi-x.y.z
export MPI_ROOT
```

vii. Next, copy the two files created in steps (v) and (vi) to */usr/mpi/gcc/openmpi-x.y.z/bin* and */usr/mpi/gcc/openmpi-x.y.z/etc*

viii. Register OpenMPI with MPI-selector.

```
[root@host~]# mpi-selector --register openmpi --source-dir
/usr/mpi/gcc/openmpi-x.y.z/bin
```

ix. Verify if it is listed in mpi-selector.

```
[root@host~]# mpi-selector --l
```

x. Set OpenMPI.

```
[root@host~]# mpi-selector --set openmpi –yes
```

xi. Logut and log back in.

MVAPICH2 (Installation and Setup)

i. Download the latest MVAPICH2 software package from http://mvapich.cse.ohio-state.edu/
ii. Untar and change your current working directory to MVAPICH2 package directory.
iii. Configure and install as:

```
[root@host~]# ./configure --prefix=/usr/mpi/gcc/mvapich2-x.y/ --with-
device=ch3:mrail --with-rdma=gen2 --enable-shared --with-ib-
libpath=/usr/lib64/ -enable-rdma-cm --libdir=/usr/mpi/gcc/mvapich2-x.y/lib64
[root@host~]# make
[root@host~]# make install
```

The above step will install MVAPICH2 in */usr/mpi/gcc/mvapich2-x.y/*

iv. Next, create a shell script , mpivars.csh, with the following entry:

```
# path
if ("" == "`echo $path | grep /usr/mpi/gcc/mvapich2-x.y/bin`") then
   set path=(/usr/mpi/gcc/mvapich2-x.y/bin $path)
endif


# LD_LIBRARY_PATH
if ("1" == "$?LD_LIBRARY_PATH") then
    if ("$LD_LIBRARY_PATH" !~ */usr/mpi/gcc/mvapich2-x.y/lib64*) then
    setenv LD_LIBRARY_PATH /usr/mpi/gcc/mvapich2-
x.y/lib64:${LD_LIBRARY_PATH}
    endif
else
   setenv LD_LIBRARY_PATH /usr/mpi/gcc/mvapich2-x.y/lib64
endif


# MPI_ROOT
setenv MPI_ROOT /usr/mpi/gcc/mvapich2-x.y
```

v. Simlarly, create another shell script, *mpivars.sh*, with the following entry:

```
# PATH
if test -z "`echo $PATH | grep /usr/mpi/gcc/ mvapich2-x.y/bin`"; then
    PATH=/usr/mpi/gcc/mvapich2-x.y/bin:${PATH}
    export PATH
fi
# LD_LIBRARY_PATH
if test -z "`echo $LD_LIBRARY_PATH | grep /usr/mpi/gcc/mvapich2-
x.y/lib64`"; then
   LD_LIBRARY_PATH=/usr/mpi/gcc/mvapich2-
x.y/lib64${LD_LIBRARY_PATH:+:}${LD_LIBRARY_PATH}
   export LD_LIBRARY_PATH
fi


# MPI_ROOT
MPI_ROOT=/usr/mpi/gcc/mvapich2-x.y
export MPI_ROOT
```

vi. Next, copy the two files created in steps (iv) and (v) *to /usr/mpi/gcc/mvapich2-x.y/bin* and */usr/mpi/gcc/mvapich2-x.y/etc*

vii. Add the following entries in *.bashrc* file:

```
export MVAPICH2_HOME=/usr/mpi/gcc/mvapich2-x.y/
export MV2_USE_IWARP_MODE=1
export MV2_USE_RDMA_CM=1
```

viii. Register MPI.

```
[root@host~]# mpi-selector --register mvapich2 --source-dir
/usr/mpi/gcc/mvapich2-x.y/bin/
```

ix. Verify if it is listed in mpi-selector.

```
[root@host~]# mpi-selector --l
```

x. Set MVAPICH2.

```
[root@host~]# mpi-selector --set mvapich2 –yes
```

xi. Logut and log back in.
xii. Populate `mpd.hosts` with node names.
xiii. On each node, create /etc/mv2.conf with a single line containing the IP address of the local adapter interface. This is how MVAPICH2 picks which interface to use for RDMA traffic.

4.2.3. Building MPI Tests

i. Download *Intel's MPI Benchmarks* from http://software.intel.com/en-us/articles/intel-mpi-benchmarks
ii. Untar and change your current working directory to *src* directory.
iii. Edit *make_mpich* file and set *MPI_HOME* variable to the MPI which you want to build the benchmarks tool against. For example, in case of openMPI-1.6.4 set the variable as:

```
MPI_HOME=/usr/mpi/gcc/openmpi-1.6.4/
```

iv. Next, build and install the benchmarks using:

```
[root@host~]# gmake -f make_mpich
```

The above step will install IMB-MPI1, IMB-IO and IMB-EXT benchmarks in the current working directory (i.e. *src*).

v. Change your working directory to the MPI installation directory. In case of OpenMPI, it will be */usr/mpi/gcc/openmpi-x.y.z/*
vi. Create a directory called *tests* and then another directory called *imb* under *tests.*
vii. Copy the benchmarks built and installed in step (iv) to the *imb* directory.
viii. Follow steps (v), (vi) and (vii) for all the nodes.

4.2.4. Running MPI Applications

- Run Intel MPI applications as:

```
mpdboot -n <no_of_nodes_in_cluster> -r ssh
mpdtrace
mpiexec -ppn -n 2 /opt/intel/impi/3.1/tests/IMB-3.1/IMB-MPI1
```

The performance is best with NIC MTU set to 9000 bytes.

- Run Open MPI application as:

```
mpirun --host  node1,node2 -mca btl openib,sm,self /usr/mpi/gcc/openmpi-
x.y.z/tests/imb/IMB-MPI1
```

Note *For OpenMPI/RDMA clusters with node counts greater than or equal to 8 nodes, and process counts greater than or equal to 64, you may experience the following RDMA address resolution error when running MPI jobs with the default OpenMPI settings:*

```
The RDMA CM returned an event error while attempting to make a connection.
This type of error usually indicates a network configuration error.
Local host:   core96n3.asicdesigners.com
Local device: Unknown
Error name:   RDMA_CM_EVENT_ADDR_ERROR
Peer:         core96n8
```

***Workaround**: Increase the OpenMPI rdma route resolution timeout. The default is 1000, or 1000ms. Increase it to 30000 with this parameter:*

```
--mca btl_openib_connect_rdmacm_resolve_timeout 30000
```

Important *openmpi-1.4.3 can cause IMB benchmark stalls due to a shared memory BTL issue. This issue is fixed in openmpi-1.4.5 and later releases. Hence, it is recommended that you download and install the latest stable release from Open MPI's official website, http://www.open-mpi.org*

- Run MVAPICH2 application as:

```
mpirun_rsh -ssh -np 8 -hostfile mpd.hosts $MVAPICH2_HOME/tests/imb/IMB-MPI1
```

4.3. Setting up NFS-RDMA

4.3.1. Starting NFS-RDMA

- **Server-side settings**

Follow the steps mentioned below to set up an NFS-RDMA server.

i. Make entry in `/etc/exports` file for the directories you need to export using NFS-RDMA on server as:

```
/share/rdma      *(fsid=0,rw,async,insecure,no_root_squash)
/share/rdma1     *(fsid=1,rw,async,insecure,no_root_squash)
```

Note that for each directory you export, you should have DIFFERENT fsid's.

ii. Load the iwarp modules and make sure peer2peer is set to 1.
iii. Load `xprtrdma` and `svcrdma` modules.

```
[root@host~]# modprobe xprtrdma
[root@host~]# modprobe svcrdma
```

iv. Start the nfs service.

```
[root@host~]# service nfs start
```

All services in NFS should start without errors.

v. Now we need to edit the file portlist in the path `/proc/fs/nfsd/`
Include the rdma port 20049 into this file.

```
[root@host~]# echo rdma 20049 > /proc/fs/nfsd/portlist
```

vi. Run `exportfs` to make local directories available for Network File System (NFS) clients to mount.

```
[root@host~]# exportfs
```

Now the NFS-RDMA server is ready.

- **Client-side settings**

Follow the steps mentioned below at the client side.

i. Load the iwarp modules and make sure peer2peer is set to 1. Make sure you are able to ping and ssh to the server Chelsio interface through which directories will be exported.

ii. Load the `xprtrdma` module.

```
[root@host~]# modprobe xprtrdma
```

iii. Run the `showmount` command to show all directories from server.

```
[root@host~]# showmount -e <server-chelsio-ip>
```

iv. Once the exported directories are listed, mount them.

```
[root@host~]# mount.nfs <serverip>:<directory> <mountpoint-on-client> -o
vers=3,rdma,port=20049,wsize=65536,rsize=65536
```

4.4. HMA

To use HMA, please ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option. Currently 256 IPv4/128 IPv6 iWARP connections are supported on T6 25G SO adapters. The following image shows the HMA reserved memory.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:05\:00.4/meminfo
EDC0:           0x0-0x3fffff [4.00 MiB]
EDC1:           0x400000-0x7fffff [4.00 MiB]
HMA:            0x800000-0x63fffff [92.0 MiB]
```

The following image shows the number of iWARP offloaded connections.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:02\:00.4/tids
Connections in use: 256
TID range: 64..319, in use: 256
STID range: 320..383, in use-IPv4/IPv6: 0/0
ATID range: 0..127, in use: 0
FTID range: 384..879
HPFTID range: 0..63
HW TID usage: 256 IP users, 0 IPv6 users
```

4.5. Performance Tuning

i. Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.
ii. Run the performance tuning script to map iWARP queues to different CPUs.

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

5. Software/Driver Unloading

To unload the iWARP driver, run the following command:

```
[root@host~]# rmmod iw_cxgb4
```

V. iSER

1. Introduction

The iSCSI Extensions for RDMA (iSER) protocol is a translation layer for operating iSCSI over RDMA transports, such as iWARP/Ethernet or InfiniBand.

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T6225-SO-CR (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the iSER driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

2. Kernel Configuration

Kernel.org linux-5.10.X/5.4.X

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. To install 5.4.143 kernel with iSER components enabled,

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

Note *If you wish to use custom 5.10.X/5.4.X kernel, enable the following iSER parameters in the kernel configuration file and then proceed with kernel installation：*

```
CONFIG_ISCSI_TARGET=m
CONFIG_INFINIBAND_ISER=m
CONFIG_INFINIBAND_ISERT=m
```

iii. Boot into the new kernel and install Chelsio Unified Wire.

RHEL 8.X/7.X, Ubuntu 20.04.X/18.04.X

No extra kernel configuration required.

3. Software/Driver Installation

3.1. Pre-requisites

- Python v2.7 or above is required for targetcli installation. If Python v2.7 is not already present in the system, or if an older version exists, v2.7.10 provided in the package will be installed.
- Uninstall any OFED present in the machine.
- *rdma-core-devel* package should be installed on RHEL 8.X/7.X systems.

3.2. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install Chelsio iSER driver, libraries and targetcli utilities.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

4. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

Follow the steps mentioned below on both target and initiator machines:

i. Unload Chelsio iWARP driver if previously loaded.

```
[root@host~]# rmmod iw_cxgb4
```

ii. Load the following modules.

```
[root@host~]# modprobe iw_cxgb4 mpa_rev=2
[root@host~]# modprobe rdma_ucm
```

iii. Start the iWARP Port Mapper Daemon.

```
[root@host~]# iwpmd
```

iv. Bring up the Chelsio interface(s).

```
[root@host~]# ifconfig ethX x.x.x.x up
```

v. On target, run the following command:

```
[root@host~]# modprobe ib_isert
```

On initiator, run the following command:

```
[root@host~]# modprobe ib_iser
```

5. Software/Driver Configuration and Fine-tuning

i. Configure LIO target with iSER support, using ramdisk as LUN.

```
[root@host~]# targetcli /backstores/ramdisk create name=ram0 size=1GB
[root@host~]# targetcli /iscsi create wwn=iqn.2003-01.org.lun0.target
[root@host~]# targetcli /iscsi/iqn.2003-01.org.lun0.target/tpg1/luns create
/backstores/ramdisk/ram0
[root@host~]# targetcli /iscsi/iqn.2003-01.org.lun0.target/tpg1 set
attribute authentication=0 demo_mode_write_protect=0 generate_node_acls=1
cache_dynamic_acls=1
[root@host~]# targetcli saveconfig
```

ii. Discover LIO target using OpeniSCSI initiator.

```
[root@host~]# iscsiadm -m discovery -t st -p  102.10.10.4
```

iii. Enable iSER support in LIO target.

```
[root@host~]# targetcli /iscsi/iqn.2003-
01.org.lun0.target/tpg1/portals/0.0.0.0:3260 enable_iser boolean=True
```

iv. Login from the initiator with iSER as transport.

```
[root@host~]# iscsiadm -m node -p 102.10.10.4 -T iqn.2003-01.org.lun0.target
--op update -n node.transport_name -v iser
[root@host~]# iscsiadm -m node -p 102.10.10.4 -T iqn.2003-01.org.lun0.target
--login
```

5.1. HMA

To use HMA, please ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option. Currently 256 IPv4/128 IPv6 iSER Offload connections are supported on T6 25G SO adapters. The following image shows the HMA reserved memory.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:05\:00.4/meminfo
EDC0:           0x0-0x3fffff [4.00 MiB]
EDC1:           0x400000-0x7fffff [4.00 MiB]
HMA:            0x800000-0x63fffff [92.0 MiB]
```

The following image shows the number of iSER offloaded connections.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:02\:00.4/tids
Connections in use: 256
TID range: 64..319, in use: 256
STID range: 320..383, in use-IPv4/IPv6: 0/0
ATID range: 0..127, in use: 0
FTID range: 384..879
HPFTID range: 0..63
HW TID usage: 256 IP users, 0 IPv6 users
```

5.2. Performance Tuning

i. Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.
ii. Run the performance tuning script to map iWARP queues to different CPUs.

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

6. Software/Driver Unloading

To unload iSER driver:

On target, run the following commands:

```
[root@host~]# rmmod ib_isert
[root@host~]# rmmod iw_cxgb4
```

On initiator, run the following commands:

```
[root@host~]# rmmod ib_iser
[root@host~]# rmmod iw_cxgb4
```

VI. WD-UDP

1. Introduction

Chelsio WD-UDP (Wire Direct-User Datagram Protocol) with Multicast is a user-space UDP stack with Multicast address reception and socket acceleration that enables users to run their existing UDP socket applications unmodified.

It features software modules that enable direct wire access from user space to the Chelsio network adapter with complete bypass of the kernel, which results in an ultra-low latency Ethernet solution for high frequency trading and other delay-sensitive applications.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the WD-UDP driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org 5.10.61
- Kernel.org 5.4.143

2. Software/Driver Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install iWARP driver and WD-UDP libraries.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers:*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

The driver must be loaded by the root user. Any attempt to load the driver as a regular user will fail.

To load the drivers, use the following commands:

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe iw_cxgb4
[root@host~]# modprobe rdma_ucm
```

4. Software/Driver Configuration and Fine-tuning

4.1. Accelerating UDP Socket Communications

The *libcxgb4_sock* library is a LD_PRELOAD-able library that accelerates UDP Socket communications transparently and without recompilation of the user application. This section describes how to use libcxgb4_sock.

By preloading *libcxgb4_sock*, all sockets created by the application are intercepted and possibly accelerated based on the user's configuration. Once accelerated, data for the UDP endpoint are transmitted or received via HW queues allocated specifically for the accelerated endpoint, bypassing the kernel, the host networking stack and sockets framework, and enabling ultra-low latency and high bandwidth utilization.

Due to HW resource limitations, only a small number of queues can be allocated for UDP acceleration. Therefore, only performance critical UDP applications should use *libcxgb4_sock*.

Only 64 IPv4 UDP / 28 IPv6 UDP sockets can be accelerated per Chelsio device, with *Unified Wire Configuration* tuning option. If you want more sockets to be accelerated, please use *Low Latency* or *High Capacity WD* tuning option.

4.1.1. Application Requirements

Certain application behavior is not supported by *libcxb4_sock* in this release. If your application does any of the following, it will not work with *libcxgb4_sock*:

- Calling *fork()* after creating UDP sockets and using the UDP socket in the child process.
- Using multiple threads on a single UDP socket without serialization. For instance, having one thread sending concurrently with another thread receiving. If your application does this, you need to serialize these paths with a spin or mutex lock.
- Only 1 UDP endpoint is allowed to bind to a given port per host. So, if you have multiple processes on the same host binding to the same UDP port number, you cannot use *libcxgb4_sock*.
- Applications must have root privileges to use *libcxgb4_sock*.
- Applications requiring bonded adapter interfaces are not currently supported.

The performance benefit observed with *libcxgb4_sock* will vary based on your application's behavior. While all UDP I/O is handled properly, only certain datagrams are accelerated. Non-accelerated I/O is handled by *libcxgb4_sock* via the host networking stack seamlessly. Both Unicast and Multicast datagrams can be accelerated, but the datagrams must meet the following criteria:

- Non-fragmented. In other words, they fit in a single IP datagram that is <= the adapter device MTU.

- Routed through the Terminator acceleration device. If the ingress datagram arrives via a device other than the Terminator acceleration device, then it will not utilize the acceleration path. On egress, if the destination IP address will not route out via the Terminator device, then it too will not be accelerated.

4.1.2. Using *libcxgb4_sock*

The *libcxgb4_sock* library utilizes the Linux RDMA Verbs subsystem, and thus requires the RDMA modules be loaded. Ensure that your systems load the *iw_cxgb4* and *rdma_ucm* modules.

```
[root@host~]# modprobe iw_cxgb4
[root@host~]# modprobe rdma_ucm
```

Now, preload *libcxgb4_sock*, using one of the methods mentioned below when starting your application:

- **Preloading using *wdload* script**

```
[root@host~]# PROT=UDP wdload <pathto>/your_application
```

The above command will generate an end point file, *libcxgb4_sock.conf* at /etc/. Parameters like interface name and port number can be changed in this file.

The following example shows how to run Netperf with WD-UDP:

server

```
[root@host~]# PROT=UDP wdload netserver -f -p <port_num> -D -L <server_ip>
```

client

```
[root@host~]# PROT=UDP wdload netperf -H <server_ip> -p <port_num> -t UDP_RR
```

- **Preloading manually**

Create a configuration file that defines which UDP endpoints should be accelerated, their vlan and priority if any, as well as which Terminator interface/port should be used. The file `/etc/libcxgb4_sock.conf` contains these endpoint entries. Create this file on all systems using *libcxgb4_sock*. Here is the syntax:

```
#
# Syntax:
#
# endpoint {attributes} ...
# where attributes include:
#                interface = interface-name
#                port = udp-port-number
#                vlan = vlan-id
#                priority = vlan-priority
#
# e.g.
# endpoint {
#                interface=eth2.5
#                port = 8000 vlan = 5 priority=1
#   }
# endpoint {interface=eth2 port=9999}
#
# endpoints that bind to port 0 (requesting the host allocate a port)
# can be accelerated with port=0:
#
# endpoint {interface=eth1 port=0}
```

Assume your Terminator interface is eth2. To accelerate all applications that preload *libcxgb4_sock* using eth2, you only need one entry in `/etc/libcxgb4_sock.conf`.

```
endpoint {interface=eth2 port=0}
```

For VLAN support, create your VLANs using the normal OS service (like vconfig, for example), then add entries to define the VLAN and priority for each endpoint to be accelerated.

```
endpoint {interface = eth2.5 port=10000}
endpoint {interface = eth2.7 priority=3 port=9000}
```

Now, preload *libcxgb4_sock*.

```
[root@host~]# CXGB4_SOCK_CFG=<path to config file>
LD_PRELOAD=libcxgb4_sock.so <pathto>/your_application
```

Note *In order to offload IPv6 UDP sockets, please select "low latency networking" as configuration tuning option during installation.*

- **Multiple interfaces**

To run on multiple interfaces, it is recommended to create a configuration file for each interface with the corresponding ports to offload. The applications can be started as below:

```
[root@host~]# CXGB4_SOCK_CFG=<config_file1> PROT=UDP wdload <application>
[root@host~]# CXGB4_SOCK_CFG=<config_file2> PROT=UDP wdload <application>
```

4.1.3. Running WD-UDP in debug mode

To use *libcxgb4_sock*'s debug capabilities, use the *libcxgb4_sock_debug* library provided in the package. Follow the steps mentioned below:

i. Make the following entry in the */etc/syslog.conf* file:

```
*.debug                                 /var/log/cxgb4.log
```

ii. Restart the service.

```
[root@host~]# /etc/init.d/syslog restart
```

iii. Finally, preload *libcxgb4_sock_debug* using the command mentioned below when starting your application:

```
[root@host~]# LD_PRELOAD=libcxgb4_sock_debug.so CXGB4_SOCK_DEBUG=-1
<pathto>/your_application
```

4.1.4. Running WD-UDP with larger I/O size

If the I/O size is > 3988, execute the commands mentioned below:

```
[root@host~]# echo 1024 > /proc/sys/vm/nr_hugepages
[root@host~]# CXGB4_SOCK_HUGE_PAGES=1 PROT=UDP wdload
<pathto>/your_application
```

4.1.5. Example with hpcbench/udp

The udp benchmark from the hpcbench suite can be used to show the benefits of libcxgb4_sock. The hpcbench suite can be found at:

Source: http://hpcbench.sourceforge.net/index.html

Sample: http://hpcbench.sourceforge.net/udp.html

The nodes in this example, r9 and r10, have Terminator eth1 configured and the ports are connected point-to-point.

```
[root@r9 ~]# ifconfig eth1|grep inet
          inet addr:192.168.2.111  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::7:4300:104:465a/64 Scope:Link

[root@r10 ~]# ifconfig eth1|grep inet
          inet addr:192.168.2.112  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::7:4300:104:456a/64 Scope:Link
```

For this benchmark, we need a simple “accelerate all” configuration on both nodes.

```
[root@r9 ~]# cat /etc/libcxgb4_sock.conf
endpoint {interface=eth1 port=0}

[root@r10 ~]# cat /etc/libcxgb4_sock.conf
endpoint {interface=eth1 port=0}
```

On R10, we run udpserver on port 9000 without *libcxgb4_sock* preloaded, and on port 90001 with preload.

```
[root@r10 ~]# /usr/local/src/hpcbench/udp/udpserver -p 9000 &
[1] 11453
[root@r10 ~]# TCP socket listening on port [9000]

[root@r10 ~]# LD_PRELOAD=libcxgb4_sock.so
/usr/local/src/hpcbench/udp/udpserver -p 9001 &
[2] 11454
[root@r10 ~]# TCP socket listening on port [9001]
```

Then on r9, we run udptest to port 9000 to see the host stack UDP latency.

```
[root@r9 ~]# /usr/local/src/hpcbench/udp/udptest -r 5 -a -h 192.168.1.112 -p
9000
```

Running the same test with *libcxgb4_sock*.

```
[root@r9 ~]# LD_PRELOAD=libcxgb4_sock.so /usr/local/src/hpcbench/udp/udptest
-r 5 -a -h 192.168.1.112 -p 9001
```

4.1.6. Determining if the application is being offloaded

To see if the application is being offloaded, open a window on one of the machines, and run `tcpdump` against the Chelsio interface. If you see minimal UDP output on the interface, then the UDP traffic is being properly offloaded.

5. Software/Driver Unloading

To unload the WD-UDP driver, run the following command:

```
[root@host~]# rmmod iw_cxgb4
```

VII. NVMe-oF iWARP

1. Introduction

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 iWARP RDMA over Ethernet. Thanks to an optimized protocol stack, an end-to-end NVMe solution is expected to reduce access latency and improve performance, particularly when paired with a low latency, high efficiency transport such as iWARP RDMA. This allows applications to achieve fast storage response times, irrespective of whether the NVMe SSDs are attached locally or accessed remotely across enterprise or datacenter networks.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T6225-SO-CR (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the NVMe-oF iWARP driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)

- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Kernel Configuration

Kernel.org linux-5.10.X/5.4.X

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. To install 5.4.143 kernel with NVMe-oF components enabled,

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

Note *If you wish to use custom 5.10.X/5.4.X kernel, enable the following parameters in the kernel configuration file and then proceed with kernel installation:*

```
CONFIG_BLK_DEV_NVME=m
CONFIG_NVME_RDMA=m
CONFIG_NVME_TARGET=m
CONFIG_NVME_TARGET_RDMA=m
CONFIG_BLK_DEV_NULL_BLK=m
CONFIG_CONFIGFS_FS=y
```

iii. Boot into the new kernel and install Chelsio Unified Wire.

RHEL 8.X/7.X, Ubuntu 20.04.X/18.04.X

No extra kernel configuration required.

3. Software/Driver Installation

3.1. Pre-requisites

- Uninstall any OFED present in the machine.
- *rdma-core-devel* package should be installed on RHEL 8.X/7.X systems.

3.2. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install iWARP RDMA Offload driver and NVMe utilities.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

4. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

Follow the steps mentioned below on both target and initiator machines:

i. Load the following drivers:

```
[root@host~]# modprobe iw_cxgb4
[root@host~]# modprobe rdma_ucm
```

ii. Bring up the Chelsio interface(s).

```
[root@host~]# ifconfig ethX x.x.x.x up
```

iii. Mount configfs.

```
[root@host~]# mount -t configfs none /sys/kernel/config
```

iv. On target, load the following drivers:

```
[root@host~]# modprobe null_blk
[root@host~]# modprobe nvmet
[root@host~]# modprobe nvmet-rdma
```

On initiator, load the following drivers:

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

5. Software/Driver Configuration and Fine-tuning

The following sections describe the method to configure target and initiator:

5.1. Target

i. The following commands will configure target using *nvmetcli* with a LUN:

```
[root@host~]# nvmetcli
/> cd subsystems
/subsystems> create nvme-ram0
/subsystems> cd nvme-ram0/namespaces
/subsystems/n...m0/namespaces> create nsid=1
/subsystems/n...m0/namespaces> cd 1
/subsystems/n.../namespaces/1> set device path=/dev/ram1
/subsystems/n.../namespaces/1> cd ../..
/subsystems/nvme-ram0> set attr allow_any_host=1
/subsystems/nvme-ram0> cd namespaces/1
/subsystems/n.../namespaces/1> enable
/subsystems/n.../namespaces/1> cd ../../../..
/> cd ports
/ports> create 1
/ports> cd 1/
/ports/1> set addr adrfam=ipv4
/ports/1> set addr trtype=rdma
/ports/1> set addr trsvcid=4420
/ports/1> set addr traddr=102.1.1.102
/ports/1> cd subsystems
/ports/1/subsystems> create nvme-ram0
```

ii. Save the target configuration to a file.

```
/ports/1/subsystems> saveconfig /root/nvme-target_setup
/ports/1/subsystems> exit
```

iii. To clear the targets,

```
[root@host~]# nvmetcli clear
```

5.2. Initiator

i. Discover the target.

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

ii. Connect to target.

- Connecting to a specific target.

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

- Connecting to all targets configured on a portal.

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

iii. List the connected targets.

```
[root@host~]# nvme list
```

iv. Format and mount the NVMe disks shown with the above command.

v. Disconnect from the target and unmount the disk.

```
[root@host~]# nvme disconnect -d <nvme_disk_name>
```

Note *nvme_disk_name is the name of the device (e.g., nvme0n1) and not the device path.*

5.3. HMA

To use HMA, please ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option. Currently 256 IPv4/128 IPv6 NVMe-oF iWARP connections are supported on T6 25G SO adapters.

The following image shows the HMA reserved memory.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:05\:00.4/meminfo
EDC0:           0x0-0x3fffff [4.00 MiB]
EDC1:           0x400000-0x7fffff [4.00 MiB]
HMA:            0x800000-0x63fffff [92.0 MiB]
```

The following image shows the number of NVMe-oF iWARP offloaded connections.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:02\:00.4/tids
Connections in use: 256
TID range: 64..319, in use: 256
STID range: 320..383, in use-IPv4/IPv6: 0/0
ATID range: 0..127, in use: 0
FTID range: 384..879
HPFTID range: 0..63
HW TID usage: 256 IP users, 0 IPv6 users
```

The total number of connections depends on the devices used and I/O queues. For example, if the Initiator connects to 2 target devices with 4 I/O queues per device (-i 4), a total of 10 NVMe-oF iWARP connections will be used.

5.4. Performance Tuning

Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.

i. Ensure that Unified Wire is installed with *NVMe Performance* configuration tuning.
ii. Run the performance tuning script to map iWARP queues to different CPUs.

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

iii. Set the *inline data size* to 8192 before enabling the NVMe port.

```
[root@host~]# mkdir /sys/kernel/config/nvmet/ports/1
[root@host~]# echo 8192 >
/sys/kernel/config/nvmet/ports/1/param_inline_data_size
```

The following log should be seen in dmesg on enabling the NVMe port.

```
[84779.386553] nvmet_rdma: enabling port 1 (10.1.1.149:4420) inline_data_size 8192
```

6. Software/Driver Unloading

Follow the steps mentioned below to unload the drivers:

On target, run the following commands:

```
[root@host~]# rmmod nvmet-rdma
[root@host~]# rmmod nvmet
[root@host~]# rmmod iw_cxgb4
```

On initiator, run the following commands:

```
[root@host~]# rmmod nvme-rdma
[root@host~]# rmmod nvme
[root@host~]# rmmod iw_cxgb4
```

VIII. SPDK NVMe-oF iWARP

1. Introduction

SPDK (storage performance development kit) designed to extract maximum performance by moving all the necessary drivers to user space, 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 storage applications. SPDK provides both user space NVMe-oF target (capable of serving disks) and initiator (host) which can run over iWARP RDMA transport.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the SPDK NVMe-oF iWARP driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Kernel.org linux-5.10.61
- Kernel.org linux-5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Kernel Configuration

Kernel.org linux-5.10.X/5.4.X

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. To install 5.4.143 kernel with NVMe-oF components enabled,

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

Note *If you wish to use custom 5.10.X/5.4.X kernel, enable the following parameters in the kernel configuration file and then proceed with kernel installation:*

```
CONFIG_BLK_DEV_NVME=m
CONFIG_NVME_RDMA=m
CONFIG_NVME_TARGET=m
CONFIG_NVME_TARGET_RDMA=m
CONFIG_BLK_DEV_NULL_BLK=m
CONFIG_CONFIGFS_FS=y
```

iii. Boot into the new kernel and install Chelsio Unified Wire.

RHEL 8.X/7.X, Ubuntu 20.04.X

No extra kernel configuration required.

3. Software/Driver Installation

3.1. Pre-requisites

- Uninstall any OFED present in the machine.
- *rdma-core-devel* package should be installed on RHEL 8.X/7.X systems.

3.2. Installation

i. rdma-core version > 23 is recommended for SPDK NVMe-oF iWARP. Below are the steps to install v27:

```
[root@host ~]# wget "https://github.com/linux-rdma/rdma-
core/releases/download/v27.0/rdma-core-27.0.tar.gz"
[root@host ~]# tar zxfv rdma-core-27.0.tar.gz
[root@host ~]# tar cjf /root/rpmbuild/SOURCES/rdma-core-27.0.tgz rdma-
core-27.0/
[root@host rdma-core-27.0]# rpmbuild -ba redhat/rdma-core.spec
[root@host ~]# cd /root/rpmbuild/RPMS/x86_64/
[root@host x86_64]# rpm -ivh *27*.rpm
```

Note *This can be skipped If the system already has the recommended version.*

ii. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

iii. Install iWARP RDMA Offload drivers, libraries and NVMe utilities.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iv. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

4. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

Follow the steps mentioned below on both target and initiator machines:

i. Load the iWARP RDMA Offload drivers.

```
[root@host~]# modprobe iw_cxgb4
[root@host~]# modprobe rdma_ucm
```

ii. Bring up the Chelsio interface(s).

```
[root@host~]# ifconfig ethX x.x.x.x up
```

5. Software/Driver Configuration and Fine-tuning

5.1. Target

i. Download SPDK v21.01.1.

```
[root@host~]# git clone https://github.com/spdk/spdk
[root@host~]# git checkout v21.01.1
[root@host~]# cd spdk
[root@host~]# git submodule update –init
 Change the below in CONFIG file.
            CONFIG_FIO_PLUGIN=y
            FIO_SOURCE_DIR=<path_to_FIO_source>
            CONFIG_RDMA=y
            CONFIG_RDMA_SEND_WITH_INVAL=y
```

ii. Run the below script to check that minimum SPDK dependencies are installed.

```
[root@host~]# cd spdk
[root@host~]# sh scripts/pkgdep.sh
```

iii. Compile SPDK with RDMA and install it.

```
[root@host~]# make clean ; ./configure --with-rdma; make; make install
```

iv. Configure Huge Pages.

```
[root@host~]# mkdir -p /mnt/huge
[root@host~]# echo 8192 > /proc/sys/vm/nr_hugepages
[root@host~]# echo 0 > /sys/kernel/mm/hugepages/hugepages-
1048576kB/nr_hugepages
[root@host~]# echo 8192 > /sys/kernel/mm/hugepages/hugepages-
2048kB/nr_hugepages
[root@host~]# vim /etc/fstab
nodev      /dev/hugepages          hugetlbfs pagesize=2MB   0 0
nodev      /mnt/huge               hugetlbfs pagesize=1GB   0 0
[root@host~]# mount -a
[root@host~]# cd spdk
[root@host~]# NRHUGE=8192 scripts/setup.sh
```

v. Start the SPDK NVMe-oF iWARP target.

```
[root@host~]# spdk/build/bin/nvmf_tgt -m 0xFFF &
```

vi. Below are the sample configuration steps to create a malloc LUN.

```
[root@host~]# spdk/scripts/rpc.py nvmf_create_transport -t RDMA -c 8192 -u
131072 -n 8192 -b 256
[root@host~]# spdk/scripts/rpc.py bdev_malloc_create -b Malloc$i 256 512
[root@host~]# spdk/scripts/rpc.py nvmf_create_subsystem nqn.2016-
06.io.spdk:cnode0 -a -s SPDK00000000000000 -d SPDK_Controller0
[root@host~]# spdk/scripts/rpc.py nvmf_subsystem_add_ns nqn.2016-
06.io.spdk:cnode0 Malloc0
[root@host~]# spdk/scripts/rpc.py nvmf_subsystem_add_listener nqn.2016-
06.io.spdk:cnode0 -t rdma -a 10.1.1.163 -s 4420
```

5.2. Initiator

SPDK NVMe-oF iWARP target works seamlessly with SPDK NVMe-oF iWARP initiator or any standard Linux kernel initiators. Please see NVMe-oF iWARP Initiator section for steps to use Linux kernel initiator. To use the SPDK NVMe-oF iWARP Initiator,

i. Follow steps i. to iv. of the SPDK Target section above to configure and install SPDK.
ii. Connect to the target using fio plugin.

```
[root@host~]# LD_PRELOAD=/root/spdk/build/fio/spdk_nvme fio  --
rw=randread/randwrite --name=random --norandommap=1 --
ioengine=/root/spdk/build/fio/spdk_nvme --thread=1 --size=400m --
group_reporting --exitall --invalidate=1 --direct=1 --filename='trtype=RDMA
adrfam=IPv4 traddr=10.1.1.163 trsvcid=4420 subnqn=nqn.2016-
06.io.spdk\:cnode0 ns=1' --time_based --runtime=20 --iodepth=64 --numjobs=4
--unit_base=1 --bs=<value> --kb_base=1000 --ramp_time=3
```

5.3. Performance Tuning

Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.

i. Ensure that Unified Wire is installed with *NVMe Performance* configuration tuning.
ii. Run the performance tuning script to map iWARP queues to different CPUs.

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

6. Software/Driver Unloading

Follow the steps mentioned below to unload the SPDK NVMe-oF iWARP driver:

```
[root@host~]# rmmod iw_cxgb4
```

IX. NVMe-oF TOE

1. Introduction

NVMe over Fabrics specification extends the benefits of NVMe to large fabrics, beyond the reach and scalability of PCIe. NVMe over Fabrics (NVMe-oF) based on TCP is a new technology which enables the use of NVMe-oF over existing Datacenter IP networks. Chelsio's TOE (TCP Offload Engine) is fully capable of offloading TCP/IP processing to hardware at 100Gbps and 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. 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.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-SO-CR (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T6225-OCP (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the NVMe-oF TOE driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Kernel.org linux-5.10.61
- Kernel.org linux-5.4.143

2. Kernel Configuration

Kernel.org linux-5.10.X/5.4.X

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. To install the 5.4.143 kernel with NVMe-TCP components enabled by default,

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

Note *If you wish to use a custom 5.10.X/5.4.X kernel, enable the following options in the kernel configuration file and then proceed with kernel installation:*

```
CONFIG_NVME_CORE=m
CONFIG_NVME_FABRICS=m
CONFIG_NVME_TCP=m
CONFIG_NVME_TARGET=m
CONFIG_NVME_TARGET_TCP=m
CONFIG_BLK_DEV_NVME=m
CONFIG_BLK_DEV_NULL_BLK=m
CONFIG_CONFIGFS_FS=y
```

iii. Boot into the new kernel and install Chelsio Unified Wire.

RHEL 8.X, Ubuntu 20.04.X

No extra kernel configuration required.

3. Software/Driver Installation

3.1. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install TOE driver and NVMe utilities.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

4. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

Follow the steps mentioned below on both target and initiator machines:

i. Load the TOE driver.

```
[root@host~]# modprobe t4_tom
```

ii. Bring up the Chelsio interface(s).

```
[root@host~]# ifconfig ethX x.x.x.x up
```

iii. Mount configfs.

```
[root@host~]# mount -t configfs none /sys/kernel/config
```

iv. Apply cop policy to disable DDP and Rx Coalesce.

```
[root@host~]# cat <policy_file>
all => offload !ddp !coalesce
[root@host~]# cop -d -o <policy_out> <policy_file>
[root@host~]# cxgbtool ethX policy <policy_out>
```

Note *The policy applied using cxgbtool is not persistent and should be applied every time drivers are reloaded or the machine is rebooted.*

The applied cop policies can be read using,

```
[root@host~]# cat /proc/net/offload/toeX/read-cop
```

v. Load the nvme drivers. On target, run the following commands:

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

On initiator, run the following commands:

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

5. Software/Driver Configuration and Fine-tuning

The following sections describe the method to configure target and initiator:

5.1. Target

i. The following commands will configure target using *nvmetcli* with a LUN.

```
[root@host~]# nvmetcli
/> cd subsystems
/subsystems> create nvme-ram0
/subsystems> cd nvme-ram0/namespaces
/subsystems/n...m0/namespaces> create nsid=1
/subsystems/n...m0/namespaces> cd 1
/subsystems/n.../namespaces/1> set device path=/dev/ram1
/subsystems/n.../namespaces/1> cd ../..
/subsystems/nvme-ram0> set attr allow_any_host=1
/subsystems/nvme-ram0> cd namespaces/1
/subsystems/n.../namespaces/1> enable
/subsystems/n.../namespaces/1> cd ../../../..
/> cd ports
/ports> create 1
/ports> cd 1/
/ports/1> set addr adrfam=ipv4
/ports/1> set addr trtype=tcp
/ports/1> set addr trsvcid=4420
/ports/1> set addr traddr=102.1.1.102
/ports/1> cd subsystems
/ports/1/subsystems> create nvme-ram0
```

ii. Save the target configuration to a file.

```
/ports/1/subsystems> saveconfig /root/nvme-target_setup
/ports/1/subsystems> exit
```

iii. To clear the targets,

```
[root@host~]# nvmetcli clear
```

5.2. Initiator

i. Discover the target.

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

ii. Connect to target.
- Connecting to a specific target.

```
[root@host~]# nvme connect -t tcp -a <target_ip> -s 4420 -n <target_name>
```

- Connecting to all targets configured on a portal.

```
[root@host~]# nvme connect-all -t tcp -a <target_ip> -s 4420
```

iii. List the connected targets.

```
[root@host~]# nvme list
```

iv. Format and mount the NVMe disks shown with the above command.
v. Disconnect from the target and unmount the disk.

```
[root@host~]# nvme disconnect -d <nvme_disk_name>
```

Note *nvme_disk_name is the name of the device (e.g., nvme0n1) and not the device path.*

5.3. HMA

To use HMA, please ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option. Currently 256 IPv4/128 IPv6 NVMe-oF TOE connections are supported on T6 25G SO adapters.

The following image shows the HMA reserved memory.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:05\:00.4/meminfo
EDC0:           0x0-0x3fffff [4.00 MiB]
EDC1:           0x400000-0x7fffff [4.00 MiB]
HMA:            0x800000-0x63fffff [92.0 MiB]
```

The following image shows the number of NVMe-oF TOE offloaded connections.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:02\:00.4/tids
Connections in use: 256
TID range: 64..319, in use: 256
STID range: 320..383, in use-IPv4/IPv6: 0/0
ATID range: 0..127, in use: 0
FTID range: 384..879
HPFTID range: 0..63
HW TID usage: 256 IP users, 0 IPv6 users
```

The total number of connections depends on the devices used and I/O queues. For example, if the Initiator connects to 2 target devices with 4 I/O queues per device (-i 4), a total of 10 NVMe-oF TOE connections will be used.

5.4. Performance Tuning

Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.

i. Run the performance tuning script to map TOE queues to different CPUs.

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

ii. Set the following *sysctl* parameter.

```
[root@host~]# sysctl -w net.ipv4.tcp_timestamps=0
[root@host~]# sysctl -w net.core.netdev_max_backlog=250000
[root@host~]# sysctl -w net.core.rmem_max=4194304
[root@host~]# sysctl -w net.core.wmem_max=4194304
[root@host~]# sysctl -w net.core.rmem_default=4194304
[root@host~]# sysctl -w net.core.wmem_default=4194304
[root@host~]# sysctl -w net.ipv4.tcp_rmem="4096 1048576 4194304"
[root@host~]# sysctl -w net.ipv4.tcp_wmem="4096 1048576 4194304"
```

iii. Set the below TOE sysctl parameters.

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

6. Software/Driver Unloading

Follow the steps mentioned below to unload the nvme drivers:

On target, run the following commands:

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

On initiator, run the following commands:

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

To, unload TOE driver, see Software/Driver Unloading section in **Network (NIC/TOE)** chapter.

X. SPDK NVMe-oF TOE

1. Introduction

NVMe over Fabrics specification extends the benefits of NVMe to large fabrics, beyond the reach and scalability of PCIe. NVMe over Fabrics (NVMe-oF) based on TCP is a new technology which enables the use of NVMe-oF over existing Datacenter IP networks. SPDK (storage performance development kit) provides an accelerated user space NVMe-oF target (RDMA and TCP transports), which provides much better performance compared with kernel solution. Chelsio's TOE (TCP Offload Engine) is fully capable of offloading SPDK NVMe-oF TCP target processing to hardware at 100Gbps and 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. The unique ability of a TOE to perform the full transport layer functionality in hardware is essential to obtaining tangible benefits.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the SPDK NVMe-oF TOE driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic

- Kernel.org linux-5.10.61
- Kernel.org linux-5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Kernel Configuration

Kernel.org linux-5.10.X/5.4.X

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install the 5.4.143 kernel with NVMe-TCP components enabled by default,

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

Note *If you wish to use a custom 5.10.X/5.4.X kernel, enable the following options in the kernel configuration file and then proceed with kernel installation:*

```
CONFIG_NVME_CORE=m
CONFIG_NVME_FABRICS=m
CONFIG_NVME_TCP=m
CONFIG_NVME_TARGET=m
CONFIG_NVME_TARGET_TCP=m
CONFIG_BLK_DEV_NVME=m
CONFIG_BLK_DEV_NULL_BLK=m
CONFIG_CONFIGFS_FS=y
```

iii. Boot into the new kernel and install Chelsio Unified Wire.

RHEL 8.X/7.X, Ubuntu 20.04.X

No extra kernel configuration required.

3. Software/Driver Installation

3.1. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install SPDK NVMe-oF TOE driver and NVMe utilities.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

4. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

Follow the steps mentioned below on the target machine:

i. Load the SPDK NVMe-oF TOE driver.

```
[root@host~]# modprobe chtcp
```

ii. Bring up the Chelsio interface(s).

```
[root@host~]# ifconfig ethX x.x.x.x up
```

5. Software/Driver Configuration and Fine-tuning

5.1. Target

i. SPDK v21.01.1, customized to support TCP/IP offload and kernel bypass for SPDK NVMe-oF TCP Target is part of Chelsio Unified Wire package. Change your current working directory to Chelsio SPDK directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x/build/src/chspdk/user/spdk/
```

ii. Configure Huge Pages.

```
[root@host~]# mkdir -p /mnt/huge
[root@host~]# echo 8192 > /proc/sys/vm/nr_hugepages
[root@host~]# echo 0 > /sys/kernel/mm/hugepages/hugepages-
1048576kB/nr_hugepages
[root@host~]# echo 8192 > /sys/kernel/mm/hugepages/hugepages-
2048kB/nr_hugepages
[root@host~]# vim /etc/fstab
nodev      /dev/hugepages          hugetlbfs pagesize=2MB   0 0
nodev      /mnt/huge               hugetlbfs pagesize=1GB   0 0
[root@host~]# mount -a
[root@host~]# NRHUGE=8192 scripts/setup.sh
```

iii. Start the target.

```
[root@host spdk]# ./build/bin/nvmf_tgt -m <cpu_mask>
```

```
[root@host spdk]# ./build/bin/nvmf_tgt -m 0xFF
[2021-07-08 11:33:57.034825] Starting SPDK v21.01.1 / DPDK 20.11.0 initialization...
[2021-07-08 11:33:57.034913] [ DPDK EAL parameters: [2021-07-08 11:33:57.034939] nvmf [2021-07-08 11:33:57.034947] --no-shconf [2021-07-08 11:33:57.034953] -c
0xFF [2021-07-08 11:33:57.034960] --log-level=lib.eal:6 [2021-07-08 11:33:57.034976] --log-level=lib.cryptodev:5 [2021-07-08 11:33:57.034988] --log-level=user1
:6 [2021-07-08 11:33:57.034996] --iova-mode=pa [2021-07-08 11:33:57.035002] --base-virtaddr=0x200000000000 [2021-07-08 11:33:57.035013] --match-allocations [20
21-07-08 11:33:57.035025] --file-prefix=spdk_pid117859 [2021-07-08 11:33:57.035036] ]
EAL: No available hugepages reported in hugepages-1048576kB
EAL: No legacy callbacks, legacy socket not created
[2021-07-08 11:33:57.070210] app.c: 538:spdk_app_start: *NOTICE*: Total cores available: 8
[2021-07-08 11:33:57.239564] reactor.c: 915:reactor_run: *NOTICE*: Reactor started on core 1
[2021-07-08 11:33:57.240112] reactor.c: 915:reactor_run: *NOTICE*: Reactor started on core 2
[2021-07-08 11:33:57.240592] reactor.c: 915:reactor_run: *NOTICE*: Reactor started on core 3
[2021-07-08 11:33:57.241106] reactor.c: 915:reactor_run: *NOTICE*: Reactor started on core 4
[2021-07-08 11:33:57.241647] reactor.c: 915:reactor_run: *NOTICE*: Reactor started on core 5
[2021-07-08 11:33:57.242140] reactor.c: 915:reactor_run: *NOTICE*: Reactor started on core 6
[2021-07-08 11:33:57.242632] reactor.c: 915:reactor_run: *NOTICE*: Reactor started on core 7
[2021-07-08 11:33:57.243086] reactor.c: 915:reactor_run: *NOTICE*: Reactor started on core 0
[2021-07-08 11:33:57.243122] accel_engine.c: 692:spdk_accel_engine_initialize: *NOTICE*: Accel engine initialized to use software engine.
```

iv. Below are the sample configuration steps to create a LUN with null device.

```
SPDK_PATH=$'ChelsioUwire-x.x.x.x/build/src/chspdk/user/spdk/'
$SPDK_PATH/scripts/rpc.py nvmf_create_transport -t TCP
$SPDK_PATH/scripts/rpc.py bdev_null_create Null0 1024 4096
$SPDK_PATH/scripts/rpc.py nvmf_create_subsystem nqn.2016-06.io.spdk:cnode0 -
a -s SPDK00000000000000 -d SPDK_Controller0
$SPDK_PATH/scripts/rpc.py nvmf_subsystem_add_ns nqn.2016-06.io.spdk:cnode0
Null0
$SPDK_PATH/scripts/rpc.py nvmf_subsystem_add_listener nqn.2016-
06.io.spdk:cnode0 -t tcp -a 10.1.1.163 -s 4420
```

5.2. Initiator

SPDK NVMe-oF TOE target works seamlessly with SPDK NVMe-oF TCP initiator or any kernel mode initiators. Please see NVMe-oF TOE Initiator section for steps to connect to the target.

6. Software/Driver Unloading

Follow the steps mentioned below to unload the SPDK NVMe-oF TOE drivers:

On target, run the following commands:

```
[root@host~]# rmmod chtcp
[root@host~]# rmmod cxgb4
```

XI. SoftiWARP

1. Introduction

SoftiWARP (siw) is a software iWARP kernel driver and user library for Linux which implements the iWARP protocol suite completely in software, without requiring any dedicated RDMA hardware. 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. It offers the below advantages:

- Provides a simple path for transition of RDMA applications to the cloud platform.
- 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.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T62100-SO-CR
- T61100-OCP
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP
- T6225-SO-CR
- T580-CR
- T580-LP-CR
- T580-SO-CR
- T580-OCP-SO
- T540-CR
- T540-LP-CR
- T540-SO-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-SO-CR
- T520-OCP-SO
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the SoftiWARP driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Kernel.org linux-5.10.61
- Kernel.org linux-5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Kernel Configuration

Kernel.org linux-5.10.X/5.4.X

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. To install the 5.4.143 kernel with SoftiWARP (siw) enabled by default,

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

Note *If you wish to use a custom 5.10.X/5.4.X kernel, enable the following option in the kernel configuration file and then proceed with kernel installation:*

```
CONFIG_RDMA_SIW=m
```

iii. Boot into the new kernel and install Chelsio Unified Wire.

RHEL 8.X/Ubuntu 20.04.X

No extra kernel configuration required.

3. Software/Driver Installation

3.1. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install network driver and NVMe, iSER utilities.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

4. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

Follow the steps mentioned below on the Initiator/Client machine:

i. Load the network driver (cxgb4).

```
[root@host~]# modprobe cxgb4
```

ii. Load the SoftiWARP driver (siw).

```
[root@host~]# modprobe siw
```

iii. Unload the iWARP RDMA offload driver (iw_cxgb4).

```
[root@host~]# rmmod iw_cxgb4
```

5. Software/Driver Configuration and Fine-tuning

SoftiWARP (siw) can be used on initiators to connect to iWARP RDMA Hardware Offload iSER and NVMe-oF targets. It can also be used on NFSoRDMA, LustreoRDMA clients to connect to the Hardware Offload servers.

5.1. Initiator/Client

Important *Disable iWARP Port Mapper (iwpmd) service on Target and Initiator.*

```
[root@host~]# systemctl stop iwpmd
```

i. RDMA tool (*rdma*) is used to configure the siw device. It is installed by default in RHEL 8.3, 8.4 and Ubuntu 20.04 distributions. If not present in the machine, install it from latest iproute2 package.

ii. Configure the siw device.

```
[root@host~]# rdma link add <siw_device> type siw netdev <ethX>
[root@host~]# ifconfig ethX <IP address> up
```

iii. Verify the configuration using *ibv_devices.*

```
[root@host ~]# ibv_devices
    device                 node GUID
    ------              ----------------
    siw_enp6s0f4        0007433ddb400000
```

iv. The initiator/client can now connect to the target/server machines.
Please refer NVMe-oF iWARP initiator and iSER initiator sections for steps to connect to the respective targets.

6. Software/Driver Unloading

Follow the steps mentioned below to unload the SoftiWARP and network drivers:

```
[root@host~]# rmmod siw
[root@host~]# rmmod cxgb4
```

XII. LIO iSCSI Target Offload

1. Introduction

Linux-IO Target (LIO) is the in-kernel SCSI target implementation in Linux. This open-source standard supports common storage fabrics, including Fibre Channel, FCoE, iEEE 1394, iSCSI, NVMe-oF, iSER, SRP, USB, vHost, etc. The LIO iSCSI fabric module implements many advanced iSCSI features that increase performance and resiliency. The LIO iSCSI Target Offload driver provides the following high-level features:

- Offloads TCP/IP.
- Offloads iSCSI Header and Data Digest Calculations.
- Offload Speeds at 10/25/40/100Gb.
- Supports Direct Data Placement (DDP).
- Supports iSCSI Segmentation Offload and iSCSI PDU recovery.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

cxgb4, iscsi_target_mod, target_core_mod, ipv6 modules are required by LIO iSCSI Target Offload (`cxgbit.ko`) module to work.

1.2.1. Linux Requirements

Currently the LIO iSCSI Target Offload driver is available for the following version(s):

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64

- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 3.10.0-957.el7.ppc64le (POWER8 LE)
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)
- RHEL 7.5, 3.10.0-862.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other versions have not been tested and are not guaranteed to work.

2. Kernel Configuration

RHEL 8.X/7.X

i. Download the kernel source RPM *kernel-3.10.0-xxx.el7.src.rpm* for your distribution.
ii. Install the kernel source.

```
[root@host~]# rpm -ivh kernel-3.10.0-xxx.el7.src.rpm
```

iii. Prepare the kernel source.

```
[root@host~]# cd /root/rpmbuild/SPECS/
[root@host~]# rpmbuild -bp kernel.spec --nodeps
[root@host~]# cd /root/rpmbuild/BUILD/kernel-3.10.0-xxx.el7/linux-3.10.0-
xxx.el7.x86_64/
[root@host~]# make prepare
```

iv. Copy the source to */usr/src* directory.

```
[root@host~]# cp -r linux-3.10.0-xxx.el7 /usr/src
```

v. Proceed with driver installation as directed in the **Software/Driver Installation** section.

Kernel.org linux-5.10.X/5.4.X

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install 5.4.143 kernel with LIO iSCSI Target Offload enabled.

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

iii. Boot into the new kernel and proceed with driver installation as directed in the **Software/Driver Installation** section.

Alternatley, to use a different 5.10.X/5.4.X kernel version,

a. Download the required kernel version from kernel.org.
b. Untar the tar-ball.
c. Change your working directory to kernel directory and run the following command to invoke the installation menu.

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

d. Select **Device Drivers → Generic Target Core Mod (TCM) and ConfigFS Infrastructure**.

e. Enable **Linux-iSCSI.org iSCSI Target Mode Stack** as a Module (if not already enabled).
f. Select **Save**.
g. Exit from the installation menu.
h. Continue with kernel installation as usual.
i. Boot into the new kernel and proceed with driver installation as directed in the **Software/Driver Installation** section.

Kernel.org linux-4.9.X

i. Download the stable version of 4.9 from kernel.org.
ii. Untar the tar-ball.
iii. Change your working directory to kernel package directory and run the following command to invoke the installation menu.

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

iv. Select **Device Drivers** > **Generic Target Core Mod (TCM) and ConfigFS Infrastructure**.
v. Enable **Linux-iSCSI.org iSCSI Target Mode Stack**.
vi. Select **Save**.
vii. Exit from the installation menu.
viii. Apply the patch provided in the Unified Wire package.

```
[root@host~]# patch -p1 <
/root/<driver_package>/src/cxgbit/patch/iscsi_target.patch
```

ix. Continue with kernel installation as usual.
x. Boot into the new kernel and proceed with driver installation as directed in the **Software/Driver Installation** section.

Ubuntu 20.04.X/18.04.X

i. Clone Ubuntu Linux kernel source repository.

Ubuntu 18.04.X

```
[root@host~]# git clone git://kernel.ubuntu.com/ubuntu/ubuntu-bionic.git
```

Ubuntu 20.04.X

```
[root@host~]# git clone git://kernel.ubuntu.com/ubuntu/ubuntu-focal.git
```

ii. Check the booted kernel version using *uname -r*

iii. Find the git tag which matches the kernel version.

```
[root@host~]# cd ubuntu-bionic/
[root@host~]# git tag -l Ubuntu-* | grep -i 4.15.0-29
Ubuntu-4.15.0-29.31
```

iv. Check out to the changeset.

```
[root@host~]# git checkout Ubuntu-4.15.0-29.31
```

v. Proceed with driver installation as directed in the **Software/Driver Installation** section.

3.14.57

i. Download the kernel from kernel.org.
ii. Untar the tar-ball.
iii. Change your working directory to kernel directory and run the following command to invoke the installation menu.

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

iv. Select **Device Drivers → Generic Target Core Mod (TCM) and ConfigFS Infrastructure**.
v. Enable **Linux-iSCSI.org iSCSI Target Mode Stack** as a Module (if not already enabled).
vi. Select **Save**.
vii. Exit from the installation menu.
viii. Untar the patch file.

```
[root@host~]# cp /root/<driver_package>/src/cxgbit/patch/linux_3-14.a .
[root@host~]# ar xvf linux_3-14.a
```

ix. Apply all the patches to kernel source one by one.

```
[root@host~]# patch -p1 < <file_name>.patch
```

x. Continue with kernel installation as usual.
xi. Reboot to the newly installed kernel. Verify by running *uname -a* command.
xii. Install LIO iSCSI target offload driver as mentioned in the next section.

3. Software/Driver Installation

3.1. Pre-requisites

The LIO iSCSI Target Offload driver requires the following components to be installed to function:

- Python (v2.7.10 provided in the package)
- TargetCLI (v2.1 provided in the package)
- OpenSSL (Download from https://www.openssl.org/source/)

If not already present in the system, the component provided in the package will be installed along with the kernel.

3.2. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install LIO driver and targetcli utilities.

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

In case of RHEL 8.X/7.X and Ubuntu 20.04.X/18.04.X, you can use one of the following options to install the driver by specifying kernel source (KSRC) and kernel object (KOBJ):

- CLI mode

```
[root@host~]# make lio_install KSRC="<kernel_source_dir>"
KOBJ="<kernel_object_dir>"
```

Example: For Ubuntu 18.04.4,

```
[root@host~]# make lio_install KSRC=/root/ubuntu-bionic/
KOBJ=/lib/modules/4.15.0-76-generic/build
```

- CLI mode (without Dialog utility)

```
[root@host~]# ./install.py --ksrc=<kernel_source_dir> --
kobj=<kernel_object_dir>
```

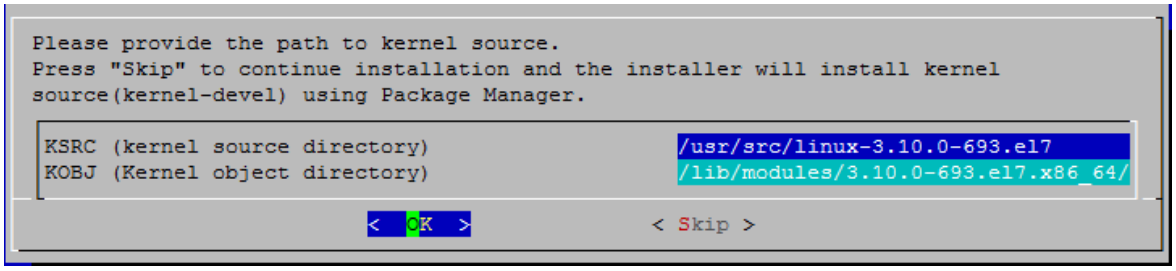
Example: For RHEL 7.8,

```
[root@host~]# ./install.py --ksrc=/usr/src/linux-3.10.0-1127.el7 -
kobj=/lib/modules/3.10.0-1127.el7.x86_64/build/
```

- GUI mode

```
[root@host~]# ./install.py --set-kpath
```

Provide the paths for kernel source and kernel object on the last screen of the installer. Select "OK".



Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

4. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

The driver must be loaded by the root user. Any attempt to load the driver as a regular user will fail.

i. Load network driver (*cxgb4*).

```
[root@host~]# modprobe cxgb4
```

ii. Bring up the interface.

```
[root@host~]# ifconfig ethX <IP address> up
```

iii. Load the LIO iSCSI Target Offload driver (*cxgbit)*.

```
 [root@host~]# modprobe cxgbit
```

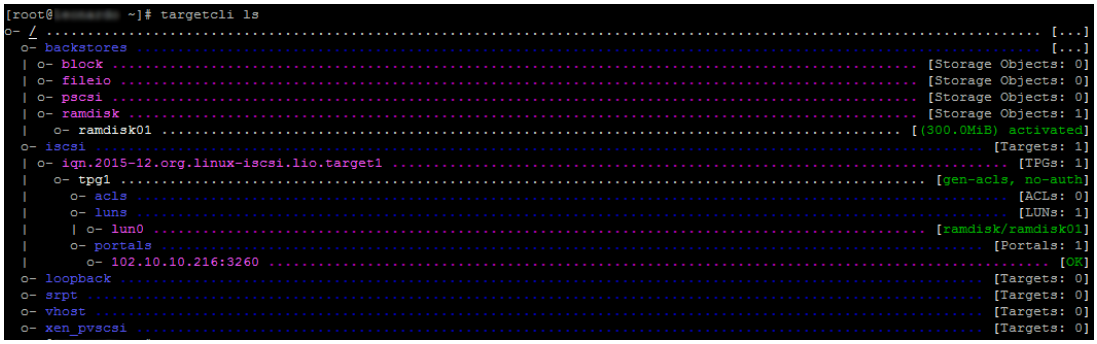
5. Software/Driver Configuration and Fine-tuning

5.1. Configuring LIO iSCSI Target

The LIO iSCSI Target needs to be configured before it can become useful. Please refer the user manual at http://www.linux-iscsi.org/Doc/LIO Admin Manual.pdf to do so.

5.1.1. Sample Configuration

Here is a sample iSCSI configuration listing a target configured with 1 RAM disk LUN and ACL not configured:



5.2. Offloading LIO iSCSI Connection

To offload the LIO iSCSI Target,

```
[root@host~]# targetcli /iscsi/<target_iqn>/tpg1/portals/<target_ip>\:3260
enable_offload boolean=True
```

Execute the above command for every portal address listening on Chelsio interface.

5.3. Running LIO iSCSI and Network Traffic Concurrently

If you wish to run network traffic with offload support (TOE) and LIO iSCSI traffic together,

i. If not done already, load network driver with offload support (TOE).

```
[root@host~]# modprobe t4_tom
```

ii. Create a new policy file.

```
[root@host~]# cat <new_policy_file>
```

iii. Add the following lines to offload all traffic except LIO iSCSI:

```
listen && src port <target_listening_port> && src host <target_listening_ip>
=> !offload
all => offload
```

iv. Compile the policy.

```
[root@host~]# cop -d -o <output_policy_file> <new_policy_file>
```

v. Apply the policy.

```
[root@host~]# cxgbtool ethX policy <output_policy_file>
```

Example:

```
[root@ ~]# modprobe t4_tom
[root@ ~]# cat policy
listen && src port 3260 && src host 102.11.11.216 => !offload
listen && src port 3260 && src host 102.22.22.216 => !offload
listen && src port 3260 && src host 0.0.0.0 => !offload
all => offload
[root@ ~]# cop -o /root/policy.o /root/policy
[root@ ~]# cxgbtool eth2 policy /root/policy.o
```

Note *The policy applied using cxgbtool is not persistent and should be applied every time drivers are reloaded or the machine is rebooted.*

The applied cop policies can be read using,

```
[root@host~]# cat /proc/net/offload/toeX/read-cop
```

5.4. Performance Tuning

i. Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.

ii. Run the performance tuning script to map LIO Target queues to different CPUs.

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

iii. For maximum performance, it is recommended to use iSCSI PDU offload initiator.

- For MTU 9000, no additional configuration needed.
- For MTU 1500, set *InitialR2T* to *No* using:

```
[root@host~]# targetcli iscsi/<target_iqn>/tpg1/ set parameter InitialR2T=No
```

6. Software/Driver Unloading

6.1. Unloading the LIO iSCSI Target Offload Driver

To unload the LIO iSCSI Target Offload kernel module, follow the steps mentioned below:

i. Log out from the initiator.
ii. Run the following command:

```
[root@host~]# targetcli /iscsi/<target_iqn>/tpg1/portals/<target_ip>\:3260
enable_offload boolean=False
```

Execute the above command for every portal address listening on Chelsio interface.

iii. Unload the driver.

```
[root@host~]# rmmod cxgbit
```

6.2. Unloading the NIC Driver

To unload the NIC driver, run the following command:

```
[root@host~]# rmmod cxgb4
```

XIII. iSCSI PDU Offload Target

1. Introduction

This section describes how to install and configure iSCSI PDU Offload Target software for use as a key element in your iSCSI SAN. The software runs on Linux-based systems that use Chelsio or non-Chelsio based Ethernet adapters. However, to guarantee highest performance, Chelsio recommends using Chelsio adapters. Chelsio's adapters include offerings that range from stateless offload adapters (regular NIC) to the full line of TCP/IP Offload Engine (TOE) adapters.

The software implements RFC 3720, the iSCSI standard of the IETF. The software has been fully tested for compliance to that RFC and others and it has been exhaustively tested for interoperability with the major iSCSI vendors.

The software implements most of the iSCSI protocol in software running in kernel mode on the host with the remaining portion, which consists of the entire fast data path, in hardware when used with Chelsio's TOE adapters. When standard NIC adapters are used the entire iSCSI protocol is executed in software.

The performance of this iSCSI stack is outstanding and when used with Chelsio's hardware it is enhanced further. Because of the tight integration with Chelsio's TOE adapters, this software has a distinct performance advantage over the regular NIC. The entire solution, which includes this software, Chelsio TOE hardware, an appropriate base computer system – including a high end disk subsystem, has industry leading performance. This can be seen when the entire solution is compared to others based on other technologies currently available on the market in terms of throughput and IOPS.

1.1. Features

Chelsio's iSCSI driver stack supports the iSCSI protocol in the Target mode. From henceforth "iSCSI Software Entity" term refers to the iSCSI target.

The Chelsio iSCSI PDU Offload Target software provides the following high level features:

- Expanded NIC Support
  - Chelsio TCP Offload Engine (TOE) Support
    - T6/T5/T4 Based HBAs (T6/T5/T4xx Series cards)
  - Non-Chelsio
    - Runs on regular NICs
- Chelsio Terminator ASIC Support
  - Offloads iSCSI Fast Data Path with Direct Data Placement (DDP)
  - Offloads iSCSI Header and Data Digest Calculations
  - Offload Speeds at 1Gb, 10Gb, 25Gb, 40Gb and 100Gb
  - Offloads TCP/IP for NAS simultaneously with iSCSI
- Target Specific features
  - Full compliance with RFC 3720

- Error Recovery Level 0 (ERL 0)
- CHAP support for both discovery and login including mutual authentication
- Internet Storage Name Service (iSNS) Client
- Target Access Control List (ACL)
- Multiple Connections per Session
- Multiple Targets
- Multiple LUNs per Target
- Multi Path I/O (MPIO)
- Greater than 2 TB Disk Support
- Reserve / Release for Microsoft Cluster© Support
- Persistent Reservation
- Dynamic LUN Resizing
- iSCSI Target Redirection
- Multiple Target device types
  - Block
  - Virtual Block (LVM, Software RAID, EVMS, etc.)
  - Built in RAM Disk
  - Built in zero copy RAM Disk
- Supports iSCSI Boot Initiators
- An Intuitive and Feature Rich Management CLI

This chapter will cover these features in detail.

1.2. Hardware Requirements

1.2.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2.2. Adapter Requirements

The Chelsio iSCSI PDU Offload Target software can be used with or without hardware protocol offload technology. There are two modes of operation using the iSCSI PDU Offload Target software on Ethernet-based adapters:

- Regular NIC – The software can be used in non-offloaded (regular NIC) mode. Please note however that this is the least optimal mode of operating the software in terms of performance.
- iSCSI HW Acceleration – In addition to offloading the TCP/IP protocols in hardware (TOE), this mode also takes advantage of Chelsio's ASIC capability of hardware assisted iSCSI data and header digest calculations as well as using the direct data placement (DDP) feature.

1.2.3. Storage Requirements

When using the Chelsio iSCSI target, a minimum of one hardware storage device is required. This device can be any of the device types that are supported (block, virtual block, RAM disk). Multiple storage devices are allowed by configuring the devices to one target or the devices to multiple targets. The software allows multiple targets to share the same device but use caution when doing this.

Chelsio's implementation of the target iSCSI stack has flexibility to accommodate a large range of configurations. For quick testing, using a RAM Disk as the block storage device works nicely. For deployment in a production environment a more sophisticated system would be needed. That typically consists of a system with one or more storage controllers with multiple disk drives attached running software or hardware based RAID.

1.3. Software Requirements

`chiscsi_base.ko` is iSCSI non-offload target mode driver and `chiscsi_t4.ko` is iSCSI PDU offload target mode driver.

cxgb4, toecore, t4_tom and chiscsi_base modules are required by `chiscsi_t4.ko` module to work in offloaded mode. Whereas in iscsi non-offloaded target (NIC) mode, only `cxgb4` is needed by `chiscsi_base.ko` module.

1.3.1. Linux Requirements

Currently the iSCSI PDU Offload Target driver is available for the following versions:

- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 16.04.6, 4.4.0-142-generic

Other kernel versions have not been tested and are not guaranteed to work.

1.3.2. Requirements for Installing the iSCSI Software

When installing the iSCSI software, it is required that the system have Linux kernel source or its headers installed in order to compile the iSCSI software as a kernel module. The source tree may be only header files, as for RHEL6 as an example, or a complete tree. The source tree needs to be configured and the header files need to be compiled. Additionally, the Linux kernel must be configured to use modules.

2. Software/Driver Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install iSCSI-target driver, firmware and utilities.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

There are two main steps to installing the Chelsio iSCSI PDU Offload Target software. They are:

1. **Installing the iSCSI software** – The majority of this section deals with how to install the iSCSI software.
2. **Configuring the iSCSI software** – Information on configuring the software can be found in a section further into this user's guide.

3.1. Latest iSCSI Software Stack Driver Software

The iSCSI software stack comes bundled in the Chelsio Unified Wire package which can be downloaded from the Chelsio Download Center.

The iSCSI software is available for use with most installations of the Linux kernel. The software is dependent on the underlying NIC adapter driver and thus the limitation on what version of the Linux kernel it can run on is mostly dependent on the NIC driver's limitations.

The iSCSI module will be installed in the `/lib/modules/<linux_kernel_version>/updates/kernel/drivers/scsi/chiscsi` directory. The modules database will be updated by the installer. This allows the iSCSI module to be located when using the `modprobe` utility. The actual module `chiscsi_t4.ko` can be found inside the package under /build/src/chiscsi/t4.

The `iscsictl` tool and the `chisns` tool will be installed in `/sbin`. The `chisns` tool starts the iSNS client. The `iscsictl` tool is provided for configuring and managing the iSCSI targets and iSNS client. It also provides control for iSCSI global settings.

1. Loading the Kernel module

- Run modprobe as follows:

```
[root@host~]# modprobe chiscsi_t4
```

Note

i. *While using rpm-tar-ball for installation*
   a. *Uninstallation will result into chiscsi.conf file renamed into chiscsi.conf.rpmsave, but if again uninstallation is done then it will lead to overwriting of the old chiscsi.rpmsave file.*
   b. *Its advised to take a backup of chiscsi.conf file before you do an uninstallation and installation of new/same unified wire package. As re-installing/upgrading unified-wire package may lead to loss of chiscsi.conf file.*

ii. *Installation/uninstallation using source-tar-ball will neither remove the conf file nor rename it. It will always be intact. However it's recommended to always take a backup of your configuration file for both methods of installation.*

A sample iSCSI configuration file will be installed in `/etc/chelsio-iscsi/chiscsi.conf`. This file should be edited using a standard text editor and customized to fit your environment.

2. Set iSCSI service to automatically start at bootup

The `chelsio-target` service scripts are installed to `/etc/init.d` and the parameters for the script are installed at `/etc/sysconfig/chiscsi`. The script is installed as a system service.

To auto-start the iSCSI target service at a certain runlevel, e.g., runlevel 3, `chkconfig` can be used on Red Hat and Novell / SuSE based systems as follows:

```
[root@host~]# chkconfig --level 3 chelsio-target on
```

The `chelsio-target` service scripts do basic checks before starting the iSCSI target service, loads the kernel module, and starts all the targets configured by default. It can also be used to stop the targets, and restart/reload configuration.

Note

For the script to execute properly, make sure the following flag is set on all kernel.org kernels.

 # CONFIG_MODULE_FORCE_LOAD=y

4. Software/Driver Configuration and Fine-tuning

The Chelsio iSCSI software needs configuration before it can become useful. The following sections describe how this is done.

There are two main components used in configuring the Chelsio iSCSI software: the **configuration file** and the **iSCSI control tool**. This section describes in some detail what they are and their relationship they have with one another.

4.1. Command Line Tools

There are two command line tools, one for control of the iSNS client and one for control of the iSCSI target nodes.

4.1.1. iscsictl

The Chelsio iSCSI control tool, `iscsictl`, is a Command Line Interface (CLI) user space program that allows administrators to:

- Start/Stop the iSCSI Target
- Start the iSNS client
- Get/Set the iSCSI driver global settings
- Get/Set/Remove the iSCSI Target configuration settings
- Retrieve active sessions' information of an iSCSI Target
- Manually flush data to the iSCSI Target disks
- Reload the iSCSI configuration file
- Update the iSCSI configuration file
- Save the current iSCSI configuration to a file

4.1.2. chisns

The Chelsio iSNS client, `chisns`, can be started independently of `iscsictl`.

4.2. iSCSI Configuration File

The iSCSI configuration file is the place where information about the Chelsio iSCSI software is stored. The information includes global data that pertains to all targets as well as information on each specific iSCSI target node. Most of the information that can be placed in the configuration file has default values that only get overwritten by the values set in the configuration file.There are only a few global configuration items that can be changed.

There are many specific parameters that can be configured, some of which are iSCSI specific and the rest being Chelsio specific. An example of an iSCSI specific item is "HeaderDigest" which is defaulted to "None" but can be overridden to "CRC32C". An example of a Chelsio specific

configurable item is “ACL” (for Access Control List). “ACL” is one of the few items that have no default.

Before starting any iSCSI target, an iSCSI configuration file must be created. An easy way to create this file is to use the provided sample configuration file and modify it. This file can be named anything and placed in any directory but it must be explicitly specified when using iscsictl by using the -f option. To avoid this, put configuration file in the default directory (`/etc/chelsio-iscsi`) and name it the default file name (`chiscsi.conf`).

4.2.1. “On the fly” Configuration Changes

Parameters for the most part can be changed while an iSCSI node is running. However, there are exceptions and restrictions to this rule that are explained in a later section that describes the details of the iSCSI control tool `iscsictl`.

4.3. A Quick Start Guide for Target

This section describes how to get started quickly with a Chelsio iSCSI target. It includes:

- Basic editing of the iSCSI configuration file.
- Basic commands of the iSCSI control tool including how to start and stop a target.

4.3.1. A Sample iSCSI Configuration File

The default Chelsio iSCSI configuration file is located at `/etc/chelsio-iscsi/chiscsi.conf`. If this file doesn’t already exist, then one needs to be created.

To configure an iSCSI target, there are three required parameters (in the form of key=value pairs) needed as follows:

- TargetName – A worldwide unique iSCSI target name.
- PortalGroup – The portal group tag associating with a list of target IP address (es) and port number(s) that service the login request. The format of this field is a Chelsio specific iSCSI driver parameter which is described in detail in the configuration file section.
- TargetDevice – A device served up by the associated target. A device can be:
    - A block device (e.g., /dev/sda)
    - A virtual block device (e.g., /dev/md0)
    - A RAM disk
    - A regular file

A target can serve multiple devices, each device will be assigned a Logical Unit Number (LUN) as per the order it is specified (i.e., the first device specified is assigned LUN 0, the second one LUN 1, …, and so on and so forth). Multiple TargetDevice key=value pairs are needed to indicate multiple devices.

Here is a sample of a minimum iSCSI target configuration located at `/etc/chelsio-iscsi/chiscsi.conf`:

```
target:
        TargetName=iqn.2006-02.com.chelsio.diskarray.san1
        TargetDevice=/dev/sda
        PortalGroup=1@192.0.2.178:3260
```

The `TargetDevice` value must match with the storage device in the system. The `PortalGroup` value must have a matching IP address of the Ethernet adapter card in the system.

For more information about TargetDevice configuration see Target Storage Device Configuration.

4.3.2. Basic iSCSI Control

Control of the Chelsio iSCSI software is done through `iscsictl`, the command line interface control tool. The following are the basic commands needed for effective control of the target.

Start Target: To start all of the iSCSI targets specified in the iSCSI configuration file, execute `iscsictl` with the "`-S`" option followed by "`target=ALL`".

```
[root@host~]# iscsictl -S target=ALL
```

To start a specific target execute `iscsictl` with "`-S`" followed by the target.

```
[root@host~]# iscsictl -S target=iqn.2006-02.com.chelsio.diskarray.san1
```

Stop Target: To stop the all the iSCSI target(s), execute `iscsictl` with "`-s`" option followed by "`target=ALL`".

```
[root@host~]# iscsictl -s target=ALL
```

To stop a specific target execute `iscsictl` with "`-s`" followed by the target name.

```
[root@host~]# iscsictl -s target=iqn.2006-02.com.chelsio.diskarray.san1
```

View Configuration: To see the configuration of all the active iSCSI targets, execute `iscsictl` with "`-c`" option.

```
[root@host~]# iscsictl -c
```

To see the more detailed configuration settings of a specific target, execute `iscsictl` with "`-c`" option followed by the target name.

```
[root@host~]# iscsictl -c target=iqn.2006-02.com.chelsio.diskarray.san1
```

View Global Settings: To see Chelsio global settings, execute `iscsictl` with "`-g`" option.

```
[root@host~]# iscsictl -g
```

Change Global Settings: To change Chelsio global settings, execute `iscsictl` with "`-G`" option.

```
[root@host~]# iscsictl -G iscsi_login_complete_time=300
```

View Help: To print help to stdout, execute `iscsictl` with "`-h`" option.

```
[root@host~]# iscsictl -h
```

4.4. The iSCSI Configuration File

The iSCSI configuration file consists of a series of blocks consisting of the following types of iSCSI entity blocks:

1. global
2. target

There can be only one global entity block whereas multiple target entity blocks are allowed. The global entity block is optional but there must be at least one target entity block.

An entity block begins with a block type (global or target). The content of each entity block is a list of parameters specified in a "key=value" format. An entity block ends at the beginning of the next entity block or at the end-of-file.

The parameter list in an entity block contains both:

- iSCSI parameters that override the default values
- Parameters that facilitate passing of control information to the iSCSI module

All lines in the configuration file that begin with "#" character are treated as comments and will be ignored. White space is not significant except in key=value pairs.

For the "key=value" parameters the <value> portion can be a single value or a list of multiple values. When <value> is a list of multiple values, they must be listed on one line with a comma "," to separate their values. Another way to list the values instead of commas is to list their values as key=value pairs repeatedly, each on a new line, until they are all listed.

There are three categories of "key=value" parameter, the first category belongs to the global entity block whereas the second and third categories belong to target entity block:

1. The Chelsio Global Entity Settings of key=value pairs
2. The iSCSI Entity Settings of key=value pairs
3. The Chelsio Entity Settings of key=value pairs

The following sub-sections describe these three categories and list in tables the details of their key=value parameters.

4.4.1. Chelsio System Wide Global Entity Settings

Description

Chelsio System Wide Global Entity Parameters pass system control information to the iSCSI software which affects all targets in the same way. More detail of these parameters below can be found in a later section entitled "System Wide Parameters".

Table of Chelsio Global Entity Settings

Key	Valid Values	Default Value	Multiple Values	Description
iscsi_auth_order	"ACL" "CHAP"	"CHAP"	No	Authorization order for login verification on the target. Valid only when a target's ACL_Enable=Yes **ACL**: ACL first then CHAP **CHAP**: CHAP first then ACL *Applies to Target(s) Only*
DISC_AuthMethod	"CHAP" "NONE"	None	No	To choose an authentication method for discovery phase.
DISC_Auth_CHAP_Policy	"Oneway" "Mutual"	"Oneway"	No	Oneway or Mutual (two-way) CHAP
DISC_Auth_CHAP_Target	"<user id>" :"<secret>"		Yes	CHAP user id and secret for the target. **<user id>** must be less than 256 characters. Commas "," are not allowed. **<secret>** must be between 6 and 255 characters. Commas "," are not allowed. The target user id and secret are used by the initiator to authenticate the target while doing mutual chap. *NOTE: The double quotes are required as part of the format.*
DISC_Auth_CHAP_Initiator	"<user id>" :"<secret>"		Yes	CHAP user id and secret for the initiator. **<user id>** must be less than 256 characters. Commas "," are not allowed. **<secret>** must be between 6 and 255 characters. Commas "," are not allowed. The initiator user id and secret are used by the target to authenticate the initiator. *NOTE: The double quotes are required as part of the format.*
iscsi_chelsio_ini_idstr	a string of maximum of 255 characters	"cxgb4i"	No	To enable additional optimization when Chelsio adapters and drivers are used at both ends (initiator and target) systems. Make sure the initiator name contain the substring set in iscsi_chelsio_ini_idstr when

				using Chelsio iscsi initiator driver.
iscsi_target_vendor_id	a string of maximum of 8 characters	"CHISCSI"	No	The target vendor ID part of the device identification sent by an iSCSI target in response of SCSI Inquiry command.
iscsi_login_complete_time	0 to 3600	300	No	Time allowed (in seconds) for the initiator to complete the login phase. Otherwise, the connection will be closed *NOTE: value zero means this check is NOT performed.*

4.4.2. iSCSI Entity Settings

Description

iSCSI Entity Parameters pass iSCSI protocol control information to the Chelsio iSCSI module. This information is unique for each entity block. The parameters follow the IETF iSCSI standard RFC 3720 in both definition and syntax. The descriptions below are mostly from this RFC.

Table of iSCSI Entity Settings

Key	Valid Values	Default Value	Multiple Values	Description
MaxConnections	1 to 65535	1	No	Initiator and target negotiate the maximum number of connections requested/acceptable.
InitialR2T	"Yes" "No"	"Yes"	No	To turn on or off the default use of R2T for unidirectional and the output part of bidirectional commands.
ImmediateData	"Yes" "No"	"Yes"	No	To turn on or off the immediate data.
FirstBurstLength	512 to 16777215 (2^{24} - 1)	65536	No	The maximum negotiated SCSI data in bytes of unsolicited data that an iSCSI initiator may send to a target during the execution of a single SCSI command.
MaxBurstLength	512 to 16777215 (2^{24} - 1)	262144	No	The maximum negotiated SCSI data in bytes, of a Data-In or a solicited Data-Out iSCSI sequence between the initiator and target.
DefaultTime2Wait	0 to 3600	2	No	The minimum time, in seconds, to wait before attempting an explicit / implicit logout or connection reset between initiator and target.
DefaultTime2Retain	0 to 3600	20	No	The maximum time, in seconds, after an initial wait.
MaxOutstandingR2T	1 to 65535	1	No	The maximum number of outstanding R2Ts per task.

DataPDUInOrder	"Yes" "No"	"Yes"	No	To indicate the data PDUs with sequence must be at continuously increasing order or can be in any order. *Chelsio only supports "Yes".*
DataSequenceInOrder	"Yes" "No"	"Yes"	No	To indicate the Data PDU sequences must be transferred in continuously non-decreasing sequence offsets or can be transferred in any order. *Chelsio only supports "Yes".*
ErrorRecoveryLevel	0 to 2	0	No	To negotiate the recovery level supported by the node. *Chelsio only supports 0.*
HeaderDigest	"None" "CRC32C"	"None"	Yes	To enable or disable iSCSI header Cyclic integrity checksums.
DataDigest	"None" "CRC32C"	"None"	Yes	To enable or disable iSCSI data Cyclic integrity checksums.
AuthMethod	"CHAP" and "None"	"None, CHAP"	Yes	To choose an authentication method during login phase.
TargetName	"<target name>"		No	A worldwide unique iSCSI target name. *Target only.*
TargetAlias	"<target alias>"		No	A human-readable name or description of a target. It is not used as an identifier, nor is it for authentication. *Target only.*
MaxRecvDataSegmentLength	512 to 16777215 (2^{24} - 1)	8192	No	To declare the maximum data segment length in bytes it can receive in an iSCSI PDU.
OFMarker	"Yes" "No"	"No"	No	To turn on or off the initiator to target markers on the connection. *Chelsio only supports "No".*
IFMarker	"Yes" "No"	"No"	No	To turn on or off the target to initiator markers on the connection. *Chelsio only supports "No".*
OFMarkInt	1 to 65535	2048	No	To set the interval for the initiator to target markers on a connection.
IFMarkInt	1 to 65535	2048	No	To set the interval for the target to initiator markers on a connection.

4.4.3. Chelsio Entity Settings

Description

Chelsio entity parameters pass control information to the Chelsio iSCSI module. The parameters are specific to Chelsio's implementation of the iSCSI node (target or initiator) and are unique for each entity block. The parameters consist of information that can be put into three categories:

1. Challenge Handshake Authentication Protocol (CHAP).
2. Target specific settings. All of the following parameters can have multiple instances in one target entity block (i.e., they can be declared multiple times for one particular target):

- Portal Group
- Storage Device

3. Access Control List (ACL)

Table of Chelsio Entity Settings

Key	Valid Values	Default Value	Multiple Values	Description
Chelsio CHAP Parameter (Target)				
Auth_CHAP_Target	"<user id>" :"<secret>"		No	CHAP user id and secret for the target. **<user id>** must be less than 256 characters. Commas "," are not allowed. **<secret>** must be between 6 and 255 characters. Commas "," are not allowed. The target user id and secret are used by the initiator to authenticate the target while doing mutual chap. *NOTE: The double quotes are required as part of the format.*
Auth_CHAP_Initiator	"<user id>" :"<secret>"		Yes	CHAP user id and secret for the initiator. **<user id>** must be less than 256 characters. Commas "," are not allowed. **<secret>** must be between 6 and 255 characters. Commas "," are not allowed. The initiator user id and secret are used by the target to authenticate the initiator. *NOTE: The double quotes are required as part of the format.*
Auth_CHAP_ChallengeLength	16 to 1024	16	No	CHAP challenge length
Auth_CHAP_Policy	"Oneway" or "Mutual"	"Oneway"	No	Oneway or Mutual (two-way) CHAP
Chelsio Target Specific Parameter				
PortalGroup	<portal group tag> @<target IP address> [:<port number>] . . . [,<target IP address> [:<port number>]] [,timeout=		Yes	The portal group name associates the given target with the given list of IP addresses (and optionally, port numbers) for servicing login requests. It"s required to have at least one per target. **<portal group tag>** is a unique tag identifying the portal group. It must be a positive integer. **<target IP address>** is the IP address associated with the portal

	<timeout value in milliseconds>] [,[portalgrouptag1, portalgrouptag2, … portalgrouptagn]			group tag. **<port number>** is the port number associated with the portal group tag. It is optional and if not specified the well-known iSCSI port number of 3260 is used. **<timeout>** is optional, it applies to all the portals in the group. The timeout value is in milliseconds and needs to be multiple of 100ms. It is used to detect loss of communications at the iSCSI level. *NOTE: There can be multiple target IP address/port numbers per portal group tag. This enables a target to operate on multiple interfaces for instance.* **<portalgrouptagX>**The portalgroup to which login requests should be redirected to. *NOTE: There can be multiple redirection target portalgroups specified for a particular target portal group and the redirection will happen to these in a round robin manner.*
ShadowMode	"Yes" "No"	"No"	No	To turn ShadowMode on or off for iSCSI Target Redirection
TargetSessionMaxCmd	1 to 2048	64	No	The maximum number of outstanding iSCSI commands per session.
TargetDevice*	<path/name> [,FILE|MEM|BLK] [,NULLRW] [,SYNC] [,RO] [,size=xMB] [,ID=xxxxxx] [,WWN=xxxxxxxxx] [,SN= xxxxxx]		No	A device served up by the associated target. The device mode can be a: • Block Device (e.g., /dev/sda) • Virtual Block Device (e.g., /dev/md0) • RamDisk • Regular File **<path/name>** is the path to the device - with the exception of when a RAM Disk is specified, where it is a unique name given to the device. If multiple RAM Disks are used for a target then each name must be unique within the target. **NULLRW** specifies that random data is returned for reads, and for writes data is dropped. Useful for testing network performance.

				SYNC specifies that the device will function in the write-through mode (i.e., the data will be flushed to the device before the response is returned to the initiator). *NOTE: SYNC is only applicable with FILE mode.* **RO** specifies the device as a read-only device. **FILE** specifies this device should be accessed via the kernel"s VFS layer. This mode is the most versatile, and it is the default mode in the cases where there is no mode specified. **BLK** specifies this device should be accessed via the kernel"s block layer. This mode is suitable for high-speed storage device such as RAID Controllers. **MEM** specifies this device should be created as a RAM Disk. **size**=xMB is used with “MEM”, to specify the RamDisk size. If not specified, the default RamDisk size is 16MB (16 Megabytes). The minimum value of x is 1 (1MB) and the maximum value is limited by system memory. **SN** is a 16 character unique value. **ID** is a 24 character unique value. **WWN** is a 16 character unique value. It is recommended when using a multipath aware initiator , the optional ID (short form for SCSI ID), SN and WWN values should be set manually for the TargetDevice. These values will be returned in Inquiry response (VPD 0x83). Multiple TargetDevice key=value pairs are needed to indicate multiple devices. There can be multiple devices for any particular target. Each device will be assigned a Logical Unit Number (LUN) as per the order it is specified (i.e., the first device specified is assigned LUN 0, the second one LUN 1, …, and so on and so forth).

				*NOTE: **FILE** mode is the most versatile mode, if in doubt use **FILE** mode.*
ACL_Enable	"Yes" "No"	"No"	No	Defines if Chelsio'sAccess Control List (ACL) method will be enforced on the target: **Yes:** ACL is enforced on the target **No:** ACL is not enforced on the target *NOTE*: ACL flag is not allowed to be updated on the fly. Target must be restarted for new ACL flag to take effect.
ACL	[iname=<name1>][;<sip=<sip1>][;dip=<dip1>][;lun=<lun_list:permissions>]		Yes	The ACL specifies which initiators and how they are allowed to access the LUNs on the target. **iname=<Initiator Name>** specifies one or more initiator names, the name must be a fully qualified iSCSI initiator name. **sip=<Source IP address>** specifies one or more IP addresses the initiators are connecting from. **Dip=<Destination IP address>** specifies one or more IP addresses that the iSCSI target is listening on (i.e., the target portal IP addresses). *NOTE: when configuring an ACL at least one of the above three must be provided:* • *iname, and/or* • *sip, and/or* • *dip.* **lun=<lun list>:<permission>** controls how the initiators access the luns. The supported value for **<lun list>** is **ALL**. **<permissions>** can be: **R**: Read Only **RW** or **WR**: Read and Write If permissions are specified then the associated LUN list is required. If no **lun=<lun list>:[R|RW]** is specified then it defaults to **ALL:RW**. *NOTE: For the Chelsio Target Software release with lun-masking included,* ***<lun list>** is in the format of **<0..N | 0~N | ALL>*** *Where:*

				0..N**: only one value from **0** through **N ***0~N**: a range of values between **0** through **N*** ***ALL**: all currently supported LUNs.* *Multiple lists of LUN numbers are allowed. When specifying the list separate the LUN ranges by a comma.*
RegisteriSNS	"Yes" "No"	"Yes"	No	To turn on or off exporting of target information via iSNS

4.4.4. Sample iSCSI Configuration File

Following is a sample configuration file. While using iSCSI node (target), irrelevant entity block can be removed or commented.

```
#
# Chelsio iSCSI Global Settings
#
global:
       iscsi_login_complete_time=300
       iscsi_auth_order=CHAP
       DISC_AuthMethod=None
       DISC_Auth_CHAP_Policy=Oneway
       DISC_Auth_CHAP_Target="target_id1":"target_secret1"
       DISC_Auth_CHAP_Initiator="initiator_id1":"initiator_sec1"
#
# an iSCSI Target "iqn.2006-02.com.chelsio.diskarray.san1"
# being served by the portal group "5". Setup as a RAM Disk.
#
target:
       TargetName=iqn.2006-02.com.chelsio.diskarray.san1
       # lun 0: a ramdisk with default size of 16MB
       TargetDevice=ramdisk,MEM
       PortalGroup=5@192.0.2.178:3260
#
# an iSCSI Target "iqn.2005-8.com.chelsio:diskarrays.san.328"
# being served by the portal group "1" and "2"
#
target:
       #
```

```
# iSCSI configuration
#
TargetName=iqn.2005-8.com.chelsio:diskarrays.san.328
TargetAlias=iTarget1
MaxOutstandingR2T=1
MaxRecvDataSegmentLength=8192
HeaderDigest=None,CRC32C
DataDigest=None,CRC32C
ImmediateData=Yes
InitialR2T=No
FirstBurstLength=65535
MaxBurstLength=262144

#
# Local block devices being served up
# lun 0 is pointed to /dev/sda
# lun 1 is pointed to /dev/sdb

TargetDevice=/dev/sda,ID=aabbccddeeffgghh,WWN=aaabbbcccdddeeef
TargetDevice=/dev/sdb

#
# Portal groups served this target
#

PortalGroup=1@102.50.50.25:3260
PortalGroup=2@102.60.60.25:3260

#
# CHAP configuration
#
Auth_CHAP_Policy=Mutual
Auth_CHAP_target="iTarget1ID":"iTarget1Secret"

Auth_CHAP_Initiator="iInitiator1":"InitSecret1"
Auth_CHAP_Initiator="iInitiator2":"InitSecret2"
Auth_CHAP_ChallengeLength=16

#
```

```
        # ACL configuration
        #
        # initiator "iqn.2006-02.com.chelsio.san1" is allowed full access
        # to this target
        ACL=iname=iqn.2006-02.com.chelsio.san1

        # any initiator from IP address 102.50.50.101 is allowed full
access                                  # of this target
        ACL=sip=102.50.50.101

        # any initiator connected via the target portal 102.60.60.25
is         # allowed full access to this target
        ACL=dip=102.60.60.25

        # initiator "iqn.2005-09.com.chelsio.san2" from 102.50.50.22 and
        # connected via the target portal 102.50.50.25 is allowed read
only         # access of this target
        ACL=iname=iqn.2006-
02.com.chelsio.san2;sip=102.50.50.22;dip=102.50.50.25;lun=ALL:R
```

4.5. Challenge-Handshake Authenticate Protocol (CHAP)

CHAP is a protocol that is used to authenticate the peer of a connection and uses the notion of a challenge and response, (i.e., the peer is challenged to prove its identity).

The Chelsio iSCSI software supports two CHAP methods: **one-way** and **mutual**. CHAP is supported for both login and discovery sessions.

4.5.1. Normal Session CHAP Authentication

For a normal Session, the CHAP authentication is configured on a per-target basis.

4.5.2. Oneway CHAP Authentication

With **one-way** CHAP (also called unidirectional CHAP) the target uses CHAP to authenticate the initiator. The initiator does not authenticate the target. This method is the default method.

For **one-way** CHAP, the initiator CHAP id and secret are configured and stored on a per-initiator with Chelsio Entity parameter "Auth_CHAP_Initiator".

4.5.3. Mutual CHAP Authentication

With **mutual** CHAP (also called bidirectional CHAP), the target and initiator use CHAP to authenticate each other.

For **mutual** CHAP, in addition to the initiator CHAP id and secret, the target CHAP id and secret are required. They are configured and stored on a per target basis with Chelsio Entity parameter “Auth_CHAP_Target”.

4.5.4. Adding CHAP User ID and Secret

A single `Auth_CHAP_Target` key and multiple `Auth_CHAP_Initiator` keys could be configured per target:

```
target:
      TargetName=iqn.2006-02.com.chelsio.diskarray.san1
      TargetDevice=/dev/sda
      PortalGroup=1@192.0.2.178:8000
      Auth_CHAP_Policy=Oneway
      Auth_CHAP_Initiator=”remoteuser1”:”remoteuser1_secret”
      Auth_CHAP_Initiator=”remoteuser2”:”remoteuser2_secret”
      Auth_CHAP_Target=“targetid1”:”target1_secret”
```

In the above example, target `iqn.2006-02.com.chelsio.diskarray.san1` has been configured to authenticate two initiators, and its own id and secret are configured for use in the case of mutual CHAP.

4.5.5. AuthMethod and Auth_CHAP_Policy Keys

By setting the iSCSI keys `AuthMethod` and `Auth_CHAP_Policy`, a user can choose whether to enforce CHAP and if mutual CHAP needs to be performed.

The `AuthMethod` key controls if an initiator needs to be authenticated or not. The default setting of `AuthMethod` is `None,CHAP`

The `Auth_CHAP_Policy` key controls which CHAP authentication (one-way or mutual) needs to be performed if CHAP is used. The default setting of `Auth_CHAP_Policy` is `Oneway`

On an iSCSI node, with `AuthMethod=None,CHAP`

- `Auth_CHAP_Policy=Oneway`, Chelsio iSCSI target will accept a relevant initiator if it does
    a) no CHAP
    b) CHAP Oneway or
    c) CHAP Mutual

- `Auth_CHAP_Policy=Mutual`, the Chelsio iSCSI target will accept a relevant initiator if it does
    a) no CHAP or
    b) CHAP Mutual

With `AuthMethod=None`, regardless the setting of the key Auth_CHAP_Policy, the Chelsio iSCSI target will only accept a relevant initiator if it does no CHAP.

With `AuthMethod=CHAP`, CHAP is enforced on the target:

i. `Auth_CHAP_Policy=Oneway`, the iSCSI target will accept a relevant initiator only if it does
   a) CHAP Oneway or
   b) CHAP Mutual

ii. `Auth_CHAP_Policy=Mutual`, the iSCSI node will accept a relevant initiator only if it does
   a) CHAP Mutual

4.5.6. Discovery Session CHAP

CHAP authentication is also supported for the discovery sessions where an initiator queries all of the available targets.

Discovery session CHAP is configured through the global section in the configuration file. List of keys to provision discovery CHAP are:

- DISC_AuthMethod: None or CHAP.
- DISC_Auth_CHAP_Policy: Oneway or Mutual (i.e., two-way) authentication.
- DISC_Auth_CHAP_Target: target CHAP user id and secret
- DISC_Auth_CHAP_Initiator: initiator CHAP user id and secret.

Sample:

```
#
# Chelsio iSCSI Global Settings
#
global:
        DISC_AuthMethod=CHAP
        DISC_Auth_CHAP_Policy=Mutual
        DISC_Auth_CHAP_Target="target_id1":"target_secret1"
        DISC_Auth_CHAP_Initiator="initiator_id1":"initiator_sec1"
```

4.6. Target Access Control List (ACL) Configuration

The Chelsio iSCSI target supports iSCSI initiator authorization via an Access Control List (ACL).

ACL configuration is supported on a per-target basis. The creation of an ACL for a target establishes:

- Which iSCSI initiators are allowed to access it
- The type of the access: read-write or read-only
- Possible SCSI layer associations of LUNs with the initiator

More than one initiator can be allowed to access a target and each initiator"s access rights can be independently configured.

The format for ACL rule is as follows:

```
ACL=[iname=<initiator name>][;<sip=<source ip addresses>]
    [;dip=<destination ip addresses>][;lun=<lun_list>:<permissions>]

 target:
      TargetName=iqn.2006-02.com.chelsio.diskarray.san1
      TargetDevice=/dev/sda

      PortalGroup=1@102.50.50.25:3260
      PortalGroup=2@102.60.60.25:3260

      # initiator "iqn.2006-02.com.chelsio.san1" is allowed
      # full read-write access to this target
      ACL=iname=iqn.2006-02.com.chelsio.san1

      # any initiator from IP address 102.50.50.101 is allowed full
      # read-write access of this target
      ACL=sip=102.50.50.101

      # any initiator connected via the target portal 102.60.60.25
      # is allowed full read-write access to this target
      ACL=dip=102.60.60.25

      # initiator "iqn.2005-09.com.chelsio.san2" from 102.50.50.22
      # and connected via the target portal 102.50.50.25 is allowed
      # read only access of this target
      ACL=iname=iqn.2006-      02.com.chelsio.san2;sip=102.50.50.22;dip=102.
50.50.25;lun=ALL:R
```

4.6.1. ACL Enforcement

To toggle ACL enforcement on a per-target base, a Chelsio keyword `ACL_Enable` is provided:

- Setting `ACL_Enable=Yes` enables the target to perform initiator authorization checking for all the initiators during login phase. And in addition, once the initiator has been authorized to access the target, the access rights will be checked for each individual LU the initiator trying to access.
- Setting `ACL_Enable=No` disable the target to perform initiator authorization checking.

When a target device is marked as read-only (RO), it takes precedence over ACL's write permission (i.e., all of ACL write permission of an initiator is ignored).

4.7. Target Storage Device Configuration

An iSCSI Target can support one or more storage devices. The storage device can either be the built-in RAM disk or actual backend storage.

Configuration of the storage is done through the Chelsio configuration file via the key-value pair `TargetDevice`.

When option `NULLRW` is specified, on writes the data is dropped without being copied to the storage device, and on reads the data is not actually read from the storage device but instead random data is used. This option is useful for measuring network performance.

The details of the parameters for the key `TargetDevice` are found in the table of Chelsio Entity Settings section earlier in this document.

4.7.1. RAM Disk Details

For the built-in RAM disk:

- The minimum size of the RAM disk is 1 Megabyte (MB) and the maximum is limited by system memory.
- To use a RAM disk with a Windows Initiator, it is recommended to set the size >= 16MB.

To configure an ramdisk specify `MEM` as the device mode:

```
TargetDevice=<name>,MEM,size=xMB
```

Where:	`<name>`	Is a unique name given to the RAM disk. This name identifies this particular ramdisk. If multiple RAM disks are configured for the same target, the name must be unique for each RAM Disk.
	`x`	Is the size of the RAM disk in MB. It's an integer between (1-max), where max is limited by system memory. If not specified, the default value is 16 MB.

```
target:
#<snip>
  # 16 Megabytes RAM Disk named ramdisk1
  TargetDevice=ramdisk1,MEM,size=16MB
#<snip>
```

4.7.2. FILE Mode Storage Device Details

The FILE mode storage device is the most common and versatile mode to access the actual storage attached to the target system:

- The FILE mode can accommodate both block devices and virtual block devices.
- The device is accessed in the exclusive mode. The device should not be accessed (or active) in any way on the target system.
- Each device should be used for one and only one iSCSI target.
- “SYNC” can be used with FILE mode to make sure the data is flushed to the storage device before the Target responds back to the Initiator.

To configure a FILE storage device specify `FILE` as the device mode:

```
TargetDevice=<path to the storage device>[,FILE][,SYNC]
```

Where:	`<path>`	Is the path to the actual storage device, such as /dev/sdb for a block device or /dev/md0 for a software RAID. The path must exist in the system.
	`SYNC`	When specified, the Target will flush all the data in the system cache to the storage driver before sending response back to the Initiator.

4.7.3. Example Configuration of FILE Mode Storage

Below is an example:

```
target:
#<snip>
 # software raid /dev/md0 is accessed in FILE mode
TargetDevice=/dev/md0,FILE
#<snip>
```

4.7.4. BLK Mode Storage Device Details

The BLK mode storage device is suitable for high-speed storage attached to the target system:

- The BLK mode can accommodate only block devices.
- The device is accessed in the exclusive mode. The device should not be accessed (or active) in any way on the target system.
- Each device should be used for one and only one iSCSI target.

To configure a block storage device specify BLK as the device mode:

```
TargetDevice=<path to the storage device>,BLK
```

where <path> Is the path to the actual storage device, such as /dev/sdb. The path must exist in the system.

```
target:
 #<snip>
  # /dev/sdb is accessed in BLK mode
  TargetDevice=/dev/sdb,BLK
 #<snip>
```

4.7.5. Multi-path Support

To enable multi-path support on the initiator, it is highly recommended that the following options are specified:

- *[,ID=xxxxxx]*: SCSI ID, a twenty-four (24) bytes alpha-numeric string
- *[,WWN=xxxxxxxxx]*: SCSI World Wide Name (WWN), a sixteen (16) bytes alpha-numeric string
- *[,SN= xxxxxx]*: SCSI SN, a sixteen (15) bytes alpha-numeric string.

The user should make sure the three values listed above are the same for the target LUNs involved in the multipath.

4.8. Target Redirection Support

An iSCSI Target can redirect an initiator to use a different IP address and port (often called a portal) instead of the current one to connect to the target. The redirected target portal can either be on the same machine, or a different one.

4.8.1. ShadowMode for Local vs. Remote Redirection

The *ShadowMode* setting specifies whether the Redirected portal groups should be present on the same machine or not. If *ShadowMode* is enabled, the redirected portal groups are on a different system. If it is disabled, then the redirected portal groups must be present on the same system otherwise the target would fail to start.

Below is an example with *ShadowMode* enabled.

```
target:
 #<snip>
     # any login requests received on 10.193.184.81:3260 will be
     # redirected to 10.193.184.85:3261.

     PortalGroup=1@10.193.184.81:3260,[2]
     PortalGroup=2@10.193.184.85:3261

     # the PortalGroup "2" is NOT presented on the same
system.          ShadowMode=Yes
 #<snip>
```

Below is an example with ShadowMode disabled:

```
target:
 #<snip>
     # any login requests received on 10.193.184.81:3260 will be
     # redirected to 10.193.184.85:3261

    PortalGroup=1@10.193.184.81:3260,[2]
    PortalGroup=2@10.193.184.85:3261
    # the PortalGroup "2" IS present on the same system
    ShadowMode=No
#<snip>
```

4.8.2. Redirecting to Multiple Portal Groups

The Chelsio iSCSI Target Redirection allows redirecting all login requests received on a particular portal group to multiple portal groups in a round robin manner.

Below is an example Redirection to Multiple Portal Groups:

```
target:
 #<snip>
     # any login requests received on 10.193.184.81:3260 will be
     # redirected to 10.193.184.85:3261 and 10.193.184.85:3262 in a
     # Round Robin Manner.

     PortalGroup=1@10.193.184.81:3260,[2,3]
     PortalGroup=2@10.193.184.85:3261
     PortalGroup=3@10.193.184.85:3262
     ShadowMode=No
#<snip>
```

4.9. The command line interface tools "iscsictl" & "chisns"

4.9.1. iscsictl

`iscsictl` is the tool Chelsio provides for controlling the iSCSI target. It is a Command Line Interface (CLI) that is invoked from the console. Its usage is as follows:

```
iscsictl <options> <mandatory parameters> [optional parameters]
```

The mandatory and optional parameters are the **key=value** pair(s) defined in RFC3720, or the **var=const** pair(s) defined for Chelsio iSCSI driver implementation. In this document, the key=value is referred to as "pair", and var=const is referred to as "parameter" to clarify between iSCSI protocol‟s pair value(s), and Chelsio iSCSI driver‟s parameter value(s). Note that all **value** and **const** are case sensitive.

4.9.2. chisns

`chisns` is the command line tool for controlling the iSNS client. This is a simple tool that starts the iSNS client with a client and server parameter.

4.9.3. iscsictl options

Options	Mandatory Parameters	Optional Parameters	Description
`-h`			Display the help messages.
`-v`			Display the version.
`-f`	`<[path/] filename>`		Specifies a pre-written iSCSI configuration text file, used to start, update, save, or reload the iSCSI node(s). This option must be specified with one of the following other options: "**-S**" or "**-W**". For the "**-S**" option "**-f**" must be specified first. All other options will ignore this "**-f**" option. If the "**-f**" option is not specified with the commands above the default configuration file will be used. It"s name and location is: `/etc/chelsio-iscsi/chiscsi.conf` The configuration file path and filename must conform to Linux standards. For the format of the iSCSI configuration file, please see "Format of The iSCSI Configuration File" section earlier in this document.
`-k`	`<key>[=<val>]`		Specifies an iSCSI Entity or Chelsio Entity parameter. This option can be specified after "**-c**" option to retrieve a parameter setting
`-c`	`target=<name> [,name2 . . . ,<nameN>]`		Display the Chelsio iSCSI target configuration. **target=<name>** parameter: Where **name** is the name of the node whose information will be returned. **name** can be one or more string of names, separated by a comma, **<name1[,name2,…,nameN] | ALL>** A **name** of **ALL** returns information on all targets. **ALL** is a reserved string that must be uppercase. Example: **iscsictl -c target=iqn.com.cc.it1** **Iscsictl -c target=iqn.com.cc.target1 -k TargetAlias** The **<name>** parameter can also be specified as one or more parameter on the same command line, separated by a comma,

			target=<name1>, <name2>, … ,<nameN> The **target=<name>** parameter(s) are optional and if not specified all active Chelsio iSCSI targets(s) configuration(s) will be displayed. If **target=ALL** is specified or no parameters are specified the output will be abbreviated. Specify specific targets to get detailed configuration data. If the **target=<name>** option is specified, the -k <key> option can optionally be specified along with this option to display only the selected entity parameter setting. Example: **iscsictl -c target=iqn.com.cc.target1 -k HeaderDigest**
`-F`		`target=<name>` `-k lun=<value>`	Flush the cached data to the target disk(s). **target=<name>** parameter: Where **name** is the name of the target to be flushed. **name** can be one or more string of names, separated by a comma, **<name1[,name2,…,nameN] | ALL>** A **name** of **ALL** will cause all the target data to be flushed. **ALL** is a reserved string that must be uppercase. The **target=name** parameter is optional. If no **target=name** parameter is specified, it is the same as specifying **target=ALL**. The -k lun=<value> option is optional. It can be used to further specify a particular lun to be flushed. Example: To flush all the targets in the system: **iscsictl -F** To flush a particular target: **iscsictl -F target=iqn.com.cc.it1** To flush only the lun 0 of a particular target: **iscsictl -F target=iqn.com.cc.it1 -k lun=0**
`-g`			Display the Chelsio iSCSI Global Entity Settings.
`-G`	`<var=const>`		Set the Chelsio iSCSI Global Entity Settings. **var=const** parameter: Where **var=const** can be any key listed under Chelsio Global Entity Settings. Example: **iscsictl -G iscsi_auth_order=ACL** The **var=const** parameter(s) are mandatory.

			If the **var=const** parameter is not specified, the command will be denied. If any of the specified **var=const** parameter is invalid, the command will reject only the invalid parameters, but will continue on and complete all other valid parameters if any others are specified.
`-s`	`target=<name>`		Stop the specified active iSCSI targets. **target=<name>** parameter: *See the description of option -c for the target=<name> parameter definition.* The **target=<name>** parameter is mandatory. If no **target=<name>** parameter is specified, the command will be denied. If the **target=<name>** parameter is specified, only the specified targets from the **target=<name>** parameters will be stopped. If **target=ALL** is specified, all active targets will stop.
`-S`	`target=<name>`		Start or reload the iSCSI targets. **target=<name>** parameter: Where **name** is the name of the target(s) that will be started or reloaded. The **target=<name>** parameter can be specified as one or more parameter on the same command line, separated by a space, **target=<name1> target=<name2> … target=<nameN>** The **target=<name>** parameter can also be, **target=ALL** A **name** of **ALL** starts or reloads all targets specified in the configuration file. **ALL** is a reserved string that must be uppercase. The **target=<name>** parameter is optional. If this command line option is specified without the -f option, the default configuration file `/etc/chelsio-iscsi/chiscsi.conf` will be used. Rules, 1. If the **target=<name>** parameter is specified, only the targets from the list will be started or reloaded. 2. If **target=ALL** is specified, all targets specified from the iSCSI configuration file will be started or reloaded. 3. If the **target=<name>** parameter is not specified, all active targets configurations will be reloaded from the configuration file while those targets are running. All non-active targets specified will not be loaded / started.

			For Rules 1-3, if the specified targets are currently active (running), they will get reloaded. For Rules 1 & 2, if the specified targets are not currently active, they will be started. For Rules 2 & 3, please note the differences – they are not the same! The global settings are also reloaded from the configuration file with this option.
-r	target=<name>	-k initiator=<name>	Retrieve active iSCSI sessions under a target. **target=<name>** parameter: Where **name** must be a single target name. If **target=<name>** parameter is specified as target=<name>, the sessions can be further filtered based on the remote node name with optional –k initiator=<name> option. Examples: **iscsictl -r target=iqn.com.cc.it1** **iscsictl -r target=iqn.com.cc.it1 -k initiator=iqn.com.cc.ii1** The first **target=<name>** parameter is mandatory. If it is not specified, the command will be denied.
-D	<Session handle in hex>		Drop initiator session. This option should be specified with the handle of the session (in hex) that needs to be dropped. The session handle can be retrieved using the previous mentioned iscsictl option (-r used to retrieve active iSCSI sessions under a target).
-W			Overwrite the specified iSCSI configuration file with ONLY the current iSCSI global settings and the active iSCSI targets" configuration to the specified iSCSI configuration file. *Will delete any non-active targets' configuration from the specified file.* The -f option MUST be specified along with this option.
-h			Display the help messages.
	server=<IP address> [:<port>]	id=<isns entity id> query=<query interval>	Start the Chelsio iSNS client. **server=<IP address>[:<port>]** where **server** is the iSNS server address. The port is optional and if it"s not specified it defaults to 3205. The server with the ip address is mandatory and if it"s not specified the, the command will be denied. **id=<isns entity id>** where **id** is the iSNS entity ID used to register with the server. It defaults to <hostname>. **query=<query interval>** where **query** is the initiator query interval (in seconds). It defaults to 60 seconds.

			Examples: chisns server=192.0.2.10 chisns server=192.0.2.10:3205 id=isnscln2 query=30 In the first example the minimum command set is given where the IP address of the iSNS server is specified. In the second example a fully qualified command is specified by also setting three optional parameters. Here, the mandatory IP address and the corresponding optional port number are specified. Also set is the iSNS entity ID to "isnscln2" as well as the query interval to 30 seconds.

4.9.4. chisns options

Options	Mandatory Parameters	Optional Parameters	Description
-h			Display the help messages.
	server=<IP address> [:<port>]	id=<isns entity id> query=<query interval>	Start the Chelsio iSNS client. **server=<IP address>[:<port>]** where **server** is the iSNS server address. The port is optional and if it's not specified it defaults to 3205. The server with the ip address is mandatory and if it's not specified the, the command will be denied. **id=<isns entity id>** where **id** is the iSNS entity ID used to register with the server. It defaults to <hostname>. **query=<query interval>** where **query** is the initiator query interval (in seconds). It defaults to 60 seconds. Examples: chisns server=192.0.2.10 chisns server=192.0.2.10:3205 id=isnscln2 query=30 In the first example the minimum command set is given where the IP address of the iSNS server is specified. In the second example a fully qualified command is specified by also setting three optional parameters. Here, the mandatory IP address and the corresponding optional port number are specified. Also set is the iSNS entity ID to 'isnscln2' as well as the query interval to 30 seconds.

4.10. Rules of Target Reload (i.e. “on the fly” changes)

After a target has been started its settings can be modified via reloading of the configuration file (i.e., `iscsictl -S`).

The following parameters cannot be changed once the target is up and running otherwise the target reload would fail:

- TargetName
- TargetSessionMaxCmd
- ACL_Enable
- ACL

The following parameters **CAN** be changed by reloading of the configuration file. The new value will become effective **IMMEDIATELY** for all connections and sessions:

- TargetDevice

PortalGroupThe following parameter **CAN** be changed by reloading of the configuration file. The new value will **NOT** affect any connections and sessions that already completed login phase:

- TargetAlias
- MaxConnections
- InitialR2T
- ImmediateData
- FirstBurstLength
- MaxBurstLength
- MaxOutstandingR2T
- HeaderDigest
- DataDigest
- MaxRecvDataSegmentLength
- AuthMethod
- Auth_CHAP_Initiator
- Auth_CHAP_Target
- Auth_CHAP_ChallengeLength
- Auth_CHAP_Policy

The following parameters **SHOULD NOT** be changed because only one valid value is supported:

- DataPDUInOrder (support only "Yes")
- DataSequenceInOrder (support only "Yes")
- ErrorRecoveryLevel (support only "0")
- OFMarker (support only "No")
- IFMarker (support only "No")

The following parameters can be changed but would not have any effect because they are either not supported or they are irrelevant:

- DefaultTime2Wait (not supported)
- DefaultTime2Retain (not supported)
- OFMarkInt (irrelevant because OFMarker=No)
- IFMarkInt (irrelevant because IFMarker=No)

4.11. System Wide Parameters

The Chelsio Global Entity Settings are system wide parameters that can be controlled through the configuration file or the use of the command line "`iscsictl -G`". The finer points of some of these parameters are described in detail here:

4.11.1. iscsi_login_complete_time

Options: An integer value between 0 and 3600 (seconds). Default value is 300 (seconds).

This is the login timeout check. The parameter controls the maximum time (in seconds) allowed to the initiator to complete the login phase. If a connection has been in the login phase longer than the set value, the target will drop the connection.

Value zero turns off this login timeout check.

4.11.2. iscsi_auth_order

Options: "ACL" or "CHAP", defaults to "CHAP"

On an iSCSI target when `ACL_Enable` is set to `Yes`, `iscsi_auth_order` decides whether to perform CHAP first then ACL or perform ACL then CHAP.

- **ACL:** When setting `iscsi_auth_order=ACL`, initiator authorization will be performed at the start of the login phase for an iSCSI normal session: upon receiving the first iscsi_login_request, the target will check its ACL. If this iSCSI connection does not match any ACL provisioned, the login attempt will be terminated.

- **CHAP**: When setting `iscsi_auth_order=CHAP`, initiator authorization will be performed at the end of the login phase for an iSCSI normal session: before going to the full feature phase, the target will check its ACL. If this iSCSI connection does not match any ACL provisioned, the login attempt will be terminated.

Note *iscsi_auth_order has no meaning when* `ACL_Enable` *is set to* `No` *on a target.*

4.11.3. iscsi_target_vendor_id

Options: A string of maximum of 8 characters, defaults to `CHISCSI`

The `iscsi_target_vendor_id` is part of the device identification sent by an iSCSI target in response of a SCSI Inquiry request.

4.11.4. iscsi_chelsio_ini_idstr

Options: A string of maximum of 255 characters, defaults to “cxgb4i”.

For an iscsi connection, more optimization can be done when both initiator and target are running Chelsio adapters and drivers.

This string is used to verify the initiator name received, and identify if the initiator is running Chelsio drivers: if the initiator name contains the same substring as `iscsi_chelsio_ini_idstr` it is assumed the initiator is running with the Chelsio iscsi initiator driver and additional offload optimization is performed.

4.12. Performance Tuning

i. Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.
ii. Ensure that Unified Wire is installed with *iSCSI Performance* configuration tuning.
iii. For T6 adapters, set *ImmediateData=No* in iSCSI target configuration file (*/etc/chelsio-iscsi/chiscsi.conf*).
iv. Next, load the iSCSI PDU offload target driver (*chiscsi_t4*) and run the *chiscsi_set_affinity.sh* script to map iSCSI worker threads to different CPUs.

```
[root@host~]# chiscsi_set_affinity.sh
```

v. Configure MTU 9000 on all interfaces.

For maximum performance, it is recommended to use iSCSI PDU offload initiator.

5. Software/Driver Unloading

Use the following command to unload the module:

```
[root@host~]# rmmod chiscsi_t4
```

XIV. iSCSI PDU Offload Initiator

1. Introduction

The Chelsio Unified Wire series of adapters support iSCSI acceleration and iSCSI Direct Data Placement (DDP) where the hardware handles the expensive byte touching operations, such as CRC computation and verification, and direct DMA to the final host memory destination:

- **iSCSI PDU digest generation and verification**
  On transmit -side, Chelsio hardware computes and inserts the Header and Data digest into the PDUs. On receive-side, Chelsio hardware computes and verifies the Header and Data digest of the PDUs.

- **Direct Data Placement (DDP)**
  Chelsio hardware can directly place the iSCSI Data-In or Data-Out PDU's payload into pre-posted destination host-memory buffers based on the Initiator Task Tag (ITT) in Data-In or Target Task Tag (TTT) in Data-Out PDUs.

- **PDU Transmit and Recovery**
  On transmit-side, Chelsio hardware accepts the complete PDU (header + data) from the host driver, computes and inserts the digests, decomposes the PDU into multiple TCP segments if necessary, and transmit all the TCP segments onto the wire. It handles TCP retransmission if needed.
  On receive-side, Chelsio hardware recovers the iSCSI PDU by reassembling TCP segments, separating the header and data, calculating and verifying the digests, then forwarding the header to the host. The payload data, if possible, will be directly placed into the pre-posted host DDP buffer. Otherwise, the data will be sent to the host too.

The *cxgb4i* driver interfaces with open-iSCSI initiator and provides the iSCSI acceleration through Chelsio hardware wherever applicable.

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T6225-SO-CR (*Memory-free; 256 IPv4/128 IPv6 offload connections supported*)
- T580-CR

- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the iSCSI PDU Offload Initiator driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 3.10.0-957.el7.ppc64le (POWER8 LE)
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)
- RHEL 7.5, 3.10.0-862.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Installation

2.1. Pre-requisites

Please make sure that the following requirements are met before installation:

- The iSCSI PDU Offload Initiator driver (cxgb4i) runs on top of NIC driver (cxgb4) and open-iscsi version greater than 2.0-872 on a Chelsio card.
- *openssl-devel* package should be installed.

2.2. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install open-iSCSI,iSCSI-initiator,firmware and utilities.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

The driver must be loaded by the root user. Any attempt to load the driver as a regular user will fail.

Run the following command to load the driver:

```
[root@host~]# modprobe cxgb4i
```

If loading of cxgb4i displays "unkown symbols found" error in dmesg, follow the steps mentioned below:

i. View all the loaded iSCSI modules.

```
[root@host~]# lsmod | grep iscsi
```

ii. Now, unload them using the following command:

```
[root@host~]# rmmod <modulename>
```

iii. Finally reload the *cxgb4i* driver.

4. Software/Driver Configuration and Fine-tuning

4.1. Accelerating open-iSCSI Initiator

The following steps need to be taken to accelerate the open-iSCSI initiator:

4.1.1. Configuring interface (*iface)* file

Create the file automatically by loading *cxgb4i* driver and then executing the following command:

```
[root@host~]# iscsiadm -m iface
```

Alternatively, you can create an interface file located under *iface* directory for the new transport class *cxgb4i* in the following format:

```
iface.iscsi_ifacename = <iface file name>
iface.hwaddress = <MAC address>
iface.transport_name = cxgb4i
iface.net_ifacename = <ethX>
iface.ipaddress = <iscsi ip address>
```

Here,

iface.iscsi_ifacename	: Interface file in `/etc/iscsi/ifaces/`
iface.hwaddress	: MAC address of the Chelsio interface via which iSCSI traffic will be running.
iface.transport_name	: Transport name, which is cxgb4i.
iface.net_ifacename	: Chelsio interface via which iSCSI traffic will be running.
iface.ipaddress	: IP address which is assigned to the interface.

Example:

```
iface.iscsi_ifacename = cxgb4i.00:07:43:04:5b:da
iface.hwaddress = 00:07:43:04:5b:da
iface.transport_name = cxgb4i
iface.net_ifacename = eth3
iface.ipaddress = 102.2.2.137
```

Note

i. *The interface file needs to be created in* `/etc/iscsi/ifaces/` *directory.*
ii. *If* `iface.ipaddress` *is specified, it needs to be either the same as the ethX's IP address or an address on the same subnet. Make sure the IP address is unique in the network.*

4.1.2. Discovery and Login

i. Starting iSCSI Daemon

Start Daemon from */sbin* by using the following command:

```
[root@host~]# iscsid
```

Note *If `iscsid` is already running, then kill the service and start it as shown above after installing the Chelsio Unified Wire package.*

ii. Discovering iSCSI Targets

To discover an iSCSI target, execute the command in the following format:

```
[root@host~]# iscsiadm -m discovery -t st -p <target ip address>:<target
port no> -I <cxgb4i iface file name>
```

Example:

```
[root@host~]# iscsiadm -m discovery -t st -p 102.2.2.155:3260 -I
cxgb4i.00:07:43:04:5b:da
```

iii. Logging into an iSCSI Target

Log into an iSCSI target using the following format:

```
[root@host~]# iscsiadm -m node -T <iqn name of target> -p <target ip
address>:<target port no> -I <cxgb4i iface file name> -l
```

Example:

```
[root@host~]# iscsiadm -m node -T iqn.2004-05.com.chelsio.target1 -p
102.2.2.155:3260,1 -I cxgb4i.00:07:43:04:5b:da -l
```

If the login fails with an error message in the format of `ERR! MaxRecvSegmentLength <X> too big. Need to be <= <Y>`. in dmesg, edit the `iscsi/iscsid.conf` file and change the setting for `MaxRecvDataSegmentLength`:

```
node.conn[0].iscsi.MaxRecvDataSegmentLength = 8192
```

Important *Always take a backup of `iscsid.conf` file before installing Chelsio Unified Wire Package. Although the file is saved to `iscsid.rpmsave` after uninstalling the package using RPM, you are still advised to take a backup.*

iv. Logging out from an iSCSI Target

Log out from an iSCSI Target by executing a command in the following format:

```
[root@host~]# iscsiadm -m node -T <iqn name of target> -p <target ip
address>:<target port no> -I <cxgb4i iface file name> -u
```

Example:

```
[root@host~]# iscsiadm -m node -T iqn.2004-05.com.chelsio.target1 -p
102.2.2.155:3260,1 -I cxgb4i.00:07:43:04:5b:da -u
```

Note *Other options can be found by typing* `iscsiadm --help`

4.2. HMA

To use HMA, please ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option.

i. Use LIO iSCSI Target in offload mode.
ii. Configure MTU 9000 for Chelsio Interfaces.
iii. Load the iSCSI PDU Offload Initiator driver using the following parameters.

```
[root@host~]# modprobe cxgb4i cxgb4i_snd_win=131072 cxgb4i_rcv_win=262144
```

Currently 256 IPv4/128 IPv6 iSCSI PDU Offload Initiator connections are supported on T6225-SO-CR adapter. The following image shows the HMA reserved memory.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:05\:00.4/meminfo
EDC0:           0x0-0x3fffff [4.00 MiB]
EDC1:           0x400000-0x7fffff [4.00 MiB]
HMA:            0x800000-0x63fffff [92.0 MiB]
```

The following image shows the number of offloaded connections.

```
[root@localhost ~]# cat /sys/kernel/debug/cxgb4/0000\:02\:00.4/tids
Connections in use: 256
TID range: 64..319, in use: 256
STID range: 320..383, in use-IPv4/IPv6: 0/0
ATID range: 0..127, in use: 0
FTID range: 384..879
HPFTID range: 0..63
HW TID usage: 256 IP users, 0 IPv6 users
```

4.3. Auto login from cxgb4i initiator at OS bootup

For iSCSI auto login (via *cxgb4i*) to work on OS startup, please add the following line to `start()` in `/etc/rc.d/init.d/iscsid` file on RHEL:

```
modprobe -q cxgb4i
```

Example:

```
force_start() {
    echo -n $"Starting $prog: "
    modprobe -q iscsi_tcpmodprobe -q ib_iser
    modprobe -q cxgb4i
    modprobe -q cxgb3i
    modprobe -q bnx2i
    modprobe -q be2iscsi
    daemon brcm_iscsiuio
    daemon $prog
    retval=$?
    echo
    [ $retval -eq 0 ] && touch $lockfile
    return $retval
}
```

4.4. Performance Tuning

Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.

In case iSCSI Initiator IRQs pose a bottleneck for multiple connections, you can improve IOPS performance using the steps mentioned below.

i. Enable iSCSI multi-queue. In 3.18+ kernels, add the below entry to the grub configuration file and reboot the machine:

```
scsi_mod.use_blk_mq=1
```

ii. Run the performance tuning script to map iSCSI Initiator queues to different CPUs.

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

iii. Load initiator driver.

```
[root@host~]# modprobe cxgb4i
```

iv. For MTU 9000, no additional configuration needed.

    For MTU 1500, set the following parameters in the iSCSI configuration file */etc/iscsi/iscsid.conf*.

```
node.session.iscsi.InitialR2T = No
node.session.iscsi.ImmediateData = Yes
node.session.iscsi.FirstBurstLength = 8192
node.conn[0].iscsi.MaxRecvDataSegmentLength = 1024
node.conn[0].iscsi.MaxXmitDataSegmentLength = 1024
```

v. Login to multiple targets.
vi. Run IOPS test.

5. Software/Driver Unloading

To unload the driver, execute the following commands:

```
[root@host~]# rmmod cxgb4i
[root@host~]# rmmod libcxgbi
```

XV. Crypto Offload

1. Introduction

Chelsio's Terminator 6 (T6) Unified Wire ASIC enables concurrent secure communication and secure storage with support for integrated TLS/SSL and inline cryptographic functions, leveraging the proprietary TCP/IP offload engine. Chelsio's full offload TLS/SSL is uniquely capable of 100Gb line-rate performance. In addition, the accelerator can be used in a traditional co-processor Lookaside mode to accelerate TLS/SSL, IPsec, SMB 3.X crypto, data at rest encryption/decryption and data-deduplication fingerprint computation.

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T62100-SO-CR*
- T61100-OCP*
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-SO-CR*
- T6225-OCP*

* *Only Co-processor driver supported.*

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the Crypto Offload driver is available for the following versions:

Linux Version	Crypto Components
RHEL 8.4, 4.18.0-305.el8.x86_64	Inline-TLS, Co-processor
RHEL 8.3, 4.18.0-240.el8.x86_64	
RHEL 7.9, 3.10.0-1160.el7.x86_64	Co-processor (IPsec)
RHEL 7.8, 3.10.0-1127.el7.x86_64	
RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)	
RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)	
Ubuntu 18.04.5, 4.15.0-135-generic	
Ubuntu 20.04.2, 5.4.0-65-generic	Inline-TLS, Co-processor
Kernel.org 5.10.61*	
Kernel.org 5.4.143*	

* Kernel compiled on RHEL 8.X.

2. Kernel Configuration

Kernel.org linux-5.10.X/5.4.X

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install the 5.4.143 kernel with Crypto components enabled by default.

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

iii. Boot into the new kernel and proceed with driver installation as directed in the **Software/Driver Installation** section.

Note *If you wish to use a custom 5.10.X/5.4.X kernel, enable the following options in the kernel configuration file and then proceed with kernel installation:*

```
CONFIG_KEYS=y
CONFIG_KEYS_DEBUG_PROC_KEYS=y
CONFIG_SECURITY=y
CONFIG_SECURITY_NETWORK=y
CONFIG_SECURITY_NETWORK_XFRM=y
CONFIG_LSM_MMAP_MIN_ADDR=65536
CONFIG_SECURITY_SELINUX=y
CONFIG_SECURITY_SELINUX_BOOTPARAM=y
CONFIG_SECURITY_SELINUX_BOOTPARAM_VALUE=1
CONFIG_SECURITY_SELINUX_DISABLE=y
CONFIG_SECURITY_SELINUX_DEVELOP=y
CONFIG_SECURITY_SELINUX_AVC_STATS=y
CONFIG_SECURITY_SELINUX_CHECKREQPROT_VALUE=1
CONFIG_DEFAULT_SECURITY_SELINUX=y
CONFIG_DEFAULT_SECURITY="selinux"
CONFIG_CRYPTO=y
CONFIG_CRYPTO_FIPS=y
CONFIG_CRYPTO_ALGAPI=y
CONFIG_CRYPTO_ALGAPI2=y
CONFIG_CRYPTO_AEAD=y
CONFIG_CRYPTO_AEAD2=y
CONFIG_CRYPTO_BLKCIPHER=y
CONFIG_CRYPTO_BLKCIPHER2=y
CONFIG_CRYPTO_HASH=y
CONFIG_CRYPTO_HASH2=y
CONFIG_CRYPTO_RNG=y
CONFIG_CRYPTO_RNG2=y
```

```
CONFIG_CRYPTO_PCOMP=y
CONFIG_CRYPTO_PCOMP2=y
CONFIG_CRYPTO_MANAGER=y
CONFIG_CRYPTO_MANAGER2=y
CONFIG_CRYPTO_NULL=y
CONFIG_CRYPTO_WORKQUEUE=y
CONFIG_CRYPTO_CRYPTD=y
CONFIG_CRYPTO_AUTHENC=y
CONFIG_CRYPTO_TEST=m
CONFIG_CRYPTO_CCM=y
CONFIG_CRYPTO_GCM=y
CONFIG_CRYPTO_SEQIV=y
CONFIG_CRYPTO_CBC=y
CONFIG_CRYPTO_CTR=y
CONFIG_CRYPTO_CTS=y
CONFIG_CRYPTO_ECB=y
CONFIG_CRYPTO_XTS=y
CONFIG_CRYPTO_HMAC=y
CONFIG_CRYPTO_GHASH=y
CONFIG_CRYPTO_MD4=m
CONFIG_CRYPTO_MD5=y
CONFIG_CRYPTO_SHA1=y
CONFIG_CRYPTO_SHA256=y
CONFIG_CRYPTO_SHA512=y
CONFIG_CRYPTO_AES=y
CONFIG_CRYPTO_AES_X86_64=y
CONFIG_CRYPTO_DEFLATE=y
CONFIG_CRYPTO_ZLIB=y
CONFIG_CRYPTO_LZO=y
CONFIG_CRYPTO_ANSI_CPRNG=y
CONFIG_CRYPTO_USER_API=y
CONFIG_CRYPTO_USER_API_HASH=y
CONFIG_CRYPTO_USER_API_SKCIPHER=y
CONFIG_CRYPTO_USER_API_RNG=y
CONFIG_CRYPTO_USER_API_AEAD=m
CONFIG_CRYPTO_HW=y
```

RHEL 8.X/7.X, Ubuntu 20.04.X/18.04.X, RHEL 7.5/7.6 ARM

No extra kernel configuration required.

3. Software/Driver Installation

3.1. Pre-requisites

Please make sure that SELinux and firewall are disabled.

3.2. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install Crypto driver.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

4. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

4.1. Inline

i. To load the Crypto Offload driver in Inline mode,

```
[root@host~]# modprobe t4_tom
```

ii. Bring up the Chelsio network interface.

```
[root@host~]# ifconfig ethX up
```

Where *ethX* is the Chelsio interface.

4.2. Co-processor

i. To load the Crypto Offload driver in Co-processor mode (*chcr*),

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe chcr
```

ii. Bring up the Chelsio network interface.

```
[root@host~]# ifconfig ethX up
```

Where *ethX* is the Chelsio interface.

5. Software/Driver Configuration and Fine-tuning

5.1. Configuring OpenSSL

OpenSSL v3.0.0 with kTLS support will be installed by Unified Wire installer at */usr/opensslv3/bin.* Additionally, all the necessary openssl configuration file changes will be made automatically.

Instead, if you wish to install the OpenSSL v3.0.0 with kTLS support and do the configuration, please follow the below mentioned steps:

i. Download the OpenSSL v3.0.0, compile with kTLS support and install it.

```
[root@host~]# wget https://www.openssl.org/source/openssl-3.0.0.tar.gz
[root@host~]# tar xf openssl-3.0.0.tar.gz
[root@host~]# cd openssl-3.0.0
[root@host~]# ./config enable-ktls shared --prefix=/root/sslv3-ktls --
openssldir=/root/sslv3-ktls
[root@host~]# make && make install
```

ii. Add the newly installed OpenSSL to ld library list.

```
[root@host~]# echo "/root/sslv3-ktls/lib64" > /etc/ld.so.conf.d/sslv3-
ktls.conf
[root@host~]# ldconfig
```

iii. Update *openssl.cnf* to enable kTLS during runtime.

```
[root@host~]# vim /root/sslv3-ktls/openssl.cnf
…
[openssl_init]
providers = provider_sect
ssl_conf = ssl_section
[ssl_section]
system_default = system_default_section
[system_default_section]
options = ktls
```

Please ensure that the following requirements are met for connections to be offloaded:

- TLS version should be v1.2
- Cipher should be AES128-GCM-SHA256

5.2. Inline TLS Offload

5.2.1. Configure TLS Offload and TOE Ports

To configure Inline TLS Offload, connection offload policy should be used with the required TCP port numbers. Follow the steps mentioned below:

i. Create a new policy file and add the following line for each TCP port (to be TLS offloaded).

```
src or dst port <tcp_port> => offload tls mss 32 bind random
.
.
all => offload
```

The *all => offload* is added to ensure that rest of the TCP ports will be regular TOE offloaded.

Example: To TLS offload TCP ports 443, 989, 1000, 1001 and 1002,

```
[root@host ~]# cat new_policy_file
src or dst port 443 => offload tls mss 32 bind random
src or dst port 989 => offload tls mss 32 bind random
src or dst port 1000 => offload tls mss 32 bind random
src or dst port 1001 => offload tls mss 32 bind random
src or dst port 1002 => offload tls mss 32 bind random
all => offload
```

Alternatively, portrange can be used to define a range of TCP ports (to be TLS offloaded).

```
src or dst portrange <M-N> => offload tls mss 32 bind random
all => offload
```

Example: To TLS offload TCP ports 443-900, create the below policy file.

```
[root@host ~]# cat new_policy_file
src or dst portrange 443-900 => offload tls mss 32 bind random
all => offload
```

ii. Compile the policy.

```
[root@host~]# cop -d -o <policy_out> <new_policy_file>
```

```
[root@host ~]# cop -d -o policy_out new_policy_file
policy rules read:
  rule 0: src or dst port 443 => offload tls mss 32 bind random
  rule 1: src or dst port 989 => offload tls mss 32 bind random
  rule 2: src or dst port 1000 => offload tls mss 32 bind random
  rule 3: src or dst port 1001 => offload tls mss 32 bind random
  rule 4: src or dst port 1002 => offload tls mss 32 bind random
  rule 5: all => offload

classifier program:
   0    8/01bb0000%ffff0000  yes->[0]     no->step 1
   1    8/000001bb%0000ffff  yes->[0]     no->step 2
   2    8/03dd0000%ffff0000  yes->[1]     no->step 3
   3    8/000003dd%0000ffff  yes->[1]     no->step 4
   4    8/03e80000%ffff0000  yes->[2]     no->step 5
   5    8/000003e8%0000ffff  yes->[2]     no->step 6
   6    8/03e90000%ffff0000  yes->[3]     no->step 7
   7    8/000003e9%0000ffff  yes->[3]     no->step 8
   8    8/03ea0000%ffff0000  yes->[4]     no->step 9
   9    8/000003ea%0000ffff  yes->[4]     no->[5]
optimized classifier program:
   0  8 #1  ffff0000  yes->[0]     no->step 5
   4    01bb0000
   5  8 #1  0000ffff  yes->[0]     no->step 10
   9    000001bb
  10  8 #1  ffff0000  yes->[1]     no->step 15
  14    03dd0000
  15  8 #1  0000ffff  yes->[1]     no->step 20
  19    000003dd
  20  8 #1  ffff0000  yes->[2]     no->step 25
  24    03e80000
  25  8 #1  0000ffff  yes->[2]     no->step 30
  29    000003e8
  30  8 #1  ffff0000  yes->[3]     no->step 35
  34    03e90000
  35  8 #1  0000ffff  yes->[3]     no->step 40
  39    000003e9
  40  8 #1  ffff0000  yes->[4]     no->step 45
  44    03ea0000
  45  8 #1  0000ffff  yes->[4]     no->[5]
  49    000003ea
offload settings:
   0: offload 1, ddp -1, coalesce -1, cong_algo -1, queue -2, class -1, tstamp -1, sack -1, tls 1, nagle -1, mss 32
   1: offload 1, ddp -1, coalesce -1, cong_algo -1, queue -2, class -1, tstamp -1, sack -1, tls 1, nagle -1, mss 32
   2: offload 1, ddp -1, coalesce -1, cong_algo -1, queue -2, class -1, tstamp -1, sack -1, tls 1, nagle -1, mss 32
   3: offload 1, ddp -1, coalesce -1, cong_algo -1, queue -2, class -1, tstamp -1, sack -1, tls 1, nagle -1, mss 32
   4: offload 1, ddp -1, coalesce -1, cong_algo -1, queue -2, class -1, tstamp -1, sack -1, tls 1, nagle -1, mss 32
   5: offload 1, ddp -1, coalesce -1, cong_algo -1, queue -2, class -1, tstamp -1, sack -1, tls 0, nagle -1, mss 0
   6: offload 0, ddp -1, coalesce -1, cong_algo -1, queue -2, class -1, tstamp -1, sack -1, tls 0, nagle -1, mss 0
```

iii. Apply the policy.

```
[root@host~]# cxgbtool <iface> policy <policy_out>
```

```
[root@      ~]# cxgbtool enp129s0f4 policy policy_out
```

Upon applying the above policy, traffic on all the mentioned TCP ports are TLS offloaded, while traffic on other TCP ports are TOE offloaded.

Note *The policy applied using cxgbtool is not persistent and should be applied every time drivers are reloaded or the machine is rebooted.*

The applied cop policies can be read using,

```
[root@host~]# cat /proc/net/offload/toeX/read-cop
```

5.2.2. Configuring and running Applications

- **OpenSSL tool**

i. Start TLS offload Server.

```
[root@host~]# cd /usr/opensslv3/bin
[root@host~]# ./openssl s_server -key <key_file> -cert <cert_file> -accept
<server_ip>:<port> -cipher AES128-GCM-SHA256 -WWW -4 -tls1_2
```

ii. Start TLS offload Client.

```
[root@host~]# cd /usr/opensslv3/bin
[root@host~]# ./openssl s_time -connect <server_ip>:<port> -www /<file>
```

```
[root@ bin]# ./openssl s_time -connect 102.1.1.154:443  -www /1G
No CIPHER specified
```

In case of IPv6, the address should be specified within [].

```
[root@host bin]# ./openssl s_time -connect '[1000::157]:443' -www /1K
No CIPHER specified
```

- **Custom Applications**

To compile custom applications using OpenSSL library,

```
[root@host~]# gcc -g -o <server/client output file> <server/client file> -
lcrypto -lssl -L/usr/opensslv3/lib64/
```

Client:

```
[root@host ~]# gcc -g -o client client.c -lcrypto -lssl -L/usr/opensslv3/lib64/
```

Server:

```
[root@host ~]# gcc -g -o server server.c -lcrypto -lssl -L/usr/opensslv3/lib64/
```

- **nginx server**

i. Download the latest stable version from nginx website.
ii. Compile nginx with the OpenSSL library and install it.

```
[root@host~]# cd nginx-x.xx.x
[root@host~]# ./configure --prefix=/usr/local/nginx --with-http_ssl_module -
-with-http_v2_module --with-http_dav_module --with-cc-opt="-
DNGX_SSL_SENDFILE -DOPENSSL_API_COMPAT=10101 -I /usr/opensslv3/include" --
with-ld-opt="-L/usr/opensslv3/lib64" && make && make install
```

iii. Configure the nginx server by updating required settings in */usr/local/nginx/nginx.conf* file.

iv. Update the below in */usr/local/nginx/nginx.conf* file.

```
http {
   ...
   ssl_protocols           TLSv1.2;
   ssl_ciphers  AES128-GCM-SHA256;
   ssl_prefer_server_ciphers  on;
   ...
      }
```

v. Load Chelsio Inline drivers and configure nginx server port as a TLS Offload port as described in Configure TLS Offload and TOE Ports section.

vi. Start nginx server.

```
[root@host~]# ./usr/local/nginx/nginx
```

The nginx server will be Inline TLS offloaded now.

vii. The Client can now connect to the Server and download the files.

5.2.3. Inline TLS Counters

To verify if Chelsio Inline Crypto is used, run the following command:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<PF4_id>/tls
Chelsio Inline TLS Stats
TLS PDU Tx: 32661534
TLS PDU Rx: 231039210
TLS Keys (DDR) Count: 48
```

5.3. Co-processor

To view the complete list of supported cryptographic algorithms, use the following command:

```
[root@host~]# cat /proc/crypto|grep -i chcr
```

The following applications can be offloaded by Chelsio Co-processor:

- Data at Rest
  - Dmcrypt
  - VeraCrypt
- TLS/SSL
  - Nginx
- IPsec
  - Strongswan

5.3.1. Configuring and running Applications

- **nginx server**

i. Download the latest stable version from nginx website.
ii. Compile and install nginx.

```
[root@host~]# cd nginx-x.xx.x
[root@host~]# ./configure --prefix=/usr/local/nginx --with-http_ssl_module --
with-http_v2_module --with-http_dav_module --with-cc-opt="-DNGX_SSL_SENDFILE
-DOPENSSL_API_COMPAT=10101 -I /usr/openssl3/include" --with-ld-opt="-
L/usr/openssl3/lib64" && make && make install
```

iii. Configure the nginx server by updating required settings in */usr/local/nginx/nginx.conf* file.
iv. Load the Chelsio Co-processor, Kernel TLS drivers and bring up the interface.

```
[root@host~]# modprobe tls
[root@host~]# modprobe chcr
[root@host~]# ifconfig ethX <IPv4/IPv6 address> up
```

v. Start nginx server.

```
[root@host~]# ./usr/local/nginx/nginx
```

The nginx server will be offloaded by Chelsio Co-processor now.

vi. The Client can now connect to the Server and download the files.

5.3.2. Coprocessor counters

To verify if Chelsio Co-processor is used by the applications, run the following command:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<PF4_id>/crypto
Chelsio Crypto Co-processor Stats
aes_ops: 1016
digest_ops: 323
aead_ops: 2739611
comp: 2740950
error: 0
Fallback: 9
```

5.4. Performance Tuning

Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.

Inline-TLS

i. Run the performance tuning script to map TOE queues to different CPUs.

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

ii. Ensure that the application sends 8k PDU for best performance.

Co-processor

i. Run the performance tuning script to map crypto queues to different CPUs.

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

6. Software/Driver Unloading

To unload Crypto Offload driver in Co-processor mode, run the following command:

```
[root@host~]# rmmod chcr
```

To unload Crypto Offload driver in Inline mode, unload the network driver in TOE mode. See Software/Driver Unloading section in **Network (NIC/TOE)** chapter for more information.

XVI. Data Center Bridging (DCB)

1. Introduction

Data Center Bridging (DCB) refers to a set of bridge specification standards, aimed to create a converged Ethernet network infrastructure shared by all storage, data networking and traffic management services. An improvement to the existing specification, DCB uses priority-based flow control to provide hardware-based bandwidth allocation and enhances transport reliability.

One of DCB's many benefits includes low operational cost, due to consolidated storage, server and networking resources, reduced heat and noise, and less power consumption.
Administration is simplified since the specifications enable transport of storage and networking traffic over a single unified Ethernet network.

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T62100-SO-CR
- T61100-OCP
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP
- T6225-SO-CR
- T580-CR
- T580-LP-CR
- T580-SO-CR
- T580-OCP-SO
- T540-CR
- T540-LP-CR
- T540-SO-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-SO-CR
- T520-OCP-SO
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the DCB feature is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org 5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Build and install all drivers with DCB support.

```
[root@host~]# make dcbx=1 install
```

Note *For more installation options, please run* `make help`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers:*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

Before proceeding, please ensure that Unified Wire is installed with DCB support as mentioned in previous section. The switch ports need to be enabled with DCBX configuration (Class mapping, ETS and PFC).

Upon loading the network/storage driver and interface bringup, firmware completes DCBX negotiation with the switch.

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig ethX up
[root@host~]# modprobe csiostor
```

The negotiated DCBX parameters can be reviewed at */sys/kernel/debug/cxgb4/<PF4_id>/dcb_info*

Example:

```
[root@[redacted] ~]# cat /sys/kernel/debug/cxgb4/0000\:04\:00.4/dcb_info
Data Center Bridging Information

Port: 0 (DCB negotiated: yes)
[ DCBx Version DCBx-CEE 1.01 ]

  Index                  :        0   1   2   3   4   5   6   7

  Priority Group IDs     :        6   6   0   6   6   6   1   6
  Priority Group BW(%)   :       60  40   0   0   0   0   0   0
  Max PG Traffic Classes [  8  ]

  Priority Flow Control  :        0   0   1   0   0   0   1   0
  Max PFC Traffic Classes [  8  ]

  Application Information:
  App    Priority    Selection         Protocol
  Index  Map         Field             ID
    0    0x40        Socket TCP (1)    0x0cbc (3260)
    1    0x04        Socket TCP (1)    0xc350 (50000)

Port: 1 (DCB negotiated: no)
```

The storage driver (FCoE Full Offload Initiator) uses the DCBX negotiated parameters (ETS, PFC etc.) without any further configuration. The network drivers (*cxgb4*, *t4_tom*) and iSCSI drivers (*cxgb4i*, *chiscsi*) need further VLAN configuration to be setup, which is explained in the Running NIC & iSCSI Traffic together with DCBx section.

4. Software/Driver Configuration and Fine-tuning

4.1. Configuring Cisco Nexus 5010 switch

4.1.1. Configuring the DCB parameters

Note *By default, the Cisco Nexus switch enables DCB functionality and configures PFC for FCoE traffic making it no drop with bandwidth of 50% assigned to FCoE class of traffic and another 50% for the rest (like NIC). If you wish to configure custom bandwidth, then follow the procedure below.*

In this procedure, you may need to adjust some of the parameters to suit your environment, such as VLAN IDs, Ethernet interfaces, and virtual Fibre Channel interfaces.

To enable PFC, ETS, and DCB functions on a Cisco Nexus 5010 series switch:

i. Open a terminal configuration setting.

```
switch# config terminal
switch(config)#
```

ii. Configure qos class-maps and set the traffic priorities: NIC uses priority 0 and FcoE uses priority 3.

```
switch(config)#class-map type qos class-nic
switch(config-cmap-qos)# match cos 0
switch(config-cmap-qos)# class-map type qos class-fcoe
switch(config-cmap-qos)# match cos 3
```

iii. Configure queuing class-maps.

```
switch(config)#class-map type queuing class-nic
switch(config-cmap-que)#match qos-group 2
```

iv. Configure network-qos class-maps.

```
switch(config)#class-map type network-qos class-nic
switch(config-cmap-nq)#match qos-group 2
```

v. Configure qos policy-maps.

```
switch(config)#policy-map type qos policy-test
switch(config-pmap-qos)#class type qos class-nic
switch(config-pmap-c-qos)#set qos-group 2
```

vi. Configure queuing policy-maps and assign network bandwidth. Divide the network bandwidth between FcoE and NIC traffic.

```
switch(config)#policy-map type queuing policy-test
switch(config-pmap-que)#class type queuing class-nic
switch(config-pmap-c-que)#bandwidth percent 50
switch(config-pmap-c-que)#class type queuing class-fcoe
switch(config-pmap-c-que)#bandwidth percent 50
switch(config-pmap-c-que)#class type queuing class-default
switch(config-pmap-c-que)#bandwidth percent 0
```

vii. Configure network-qos policy maps and set up the PFC for no-drop traffic class.

```
switch(config)#policy-map type network-qos policy-test
switch (config-pmap-nq)#class type network-qos class-nic
switch(config-pmap-nq-c)#pause no-drop
```

Note *By default, FCoE is set to pause no drop. In such a trade off, one may want to set NIC to drop instead.*

viii. Apply the new policy (PFC on NIC and FcoE traffic) to the entire system.

```
switch(config)#system qos
switch(config-sys-qos)#service-policy type qos input policy-test
switch(config-sys-qos)#service-policy type queuing output policy-test
switch(config-sys-qos)#service-policy type queuing input policy-test
switch(config-sys-qos)#service-policy type network-qos policy-test
```

4.1.2. Configuring the FCoE/FC Ports

In this procedure, you may need to adjust some of the parameters to suit your environment, such as VLAN IDs, Ethernet interfaces, and virtual Fibre Channel interfaces.

i. Following steps will enable FCoE services on a particular VLAN and does a VSAN-VLAN mapping. Need not do these steps every time, unless a new mapping has to be created.

```
switch(config)# vlan 2
switch(config-vlan)# fcoe vsan 2
switch(config-vlan)#exit
```

ii. Following steps help in creating a virtual fibre channel (VFC) and binds that VFC to a Ethernet interface so that the Ethernet port begins functioning as a FCoE port.

```
switch(config)# interface vfc 13
switch(config-if)# bind interface ethernet 1/13
switch(config-if)# no shutdown
switch(config-if)# exit
switch(config)#vsan database
switch(config-vsan-db)# vsan 2
switch(config-vsan-db)# vsan 2 interface vfc 13
switch(config-vsan-db)# exit
```

Note *If you are binding the VFC to a MAC address instead of an ethernet port then make sure the ethernet port is part of both default VLAN and FCoE VLAN.*

iii. Assign VLAN ID to the Ethernet port on which FCoE service was enabled in step1.

```
switch(config)# interface ethernet 1/13
switch(config-if)# switchport mode trunk
switch(config-if)# switchport trunk allowed vlan 2
switch(config-if)# no shutdown
switch(config)#exit
```

iv. Enabling DCB.

```
switch(config)# interface ethernet 1/13
switch(config-if)# priority-flow-control mode auto
switch(config-if)# flowcontrol send off
switch(config-if)# flowcontrol receive off
switch(config-if)# lldp transmit
switch(config-if)# lldp receive
switch(config-if)# no shutdown
```

v. On the FC Ports, if a FC target is connected then perform the following steps:

```
switch(config)#vsan database
switch(config-vsan-db)#vsan 2
switch(config-vsan-db)# vsan 2 interface fc 2/2
switch(config-vsan-db)#exit
switch(config)interface fc 2/2

switch(config-if)# switchport mode auto
switch(config-if)# switchport speed auto
switch(config-if)# no shutdown.
```

vi. If you have not created a zone then make sure the default-zone permits the VSAN created, otherwise the initiator and the target on that particular VSAN although FLOGI'd into the switch will not talk to each other. To enable it, execute the below command:

```
switch(config)# zone default-zone permit vsan 2
```

4.2. Configuring the Brocade 8000 switch

i. Configure LLDP for FCoE. Example of configuring LLDP for 10-Gigabit Ethernet interface.

```
switch(config)#protocol lldp
switch(conf-lldp)#advertise dcbx-fcoe-app-tlv
switch(conf-lldp)#advertise dcbx-fcoe-logical-link-tlv
```

ii. Create a CEE Map to carry LAN and SAN traffic if it does not exist. Example of creating a CEE map.

```
switch(config)# cee-map default
switch(conf-cee-map)#priority-group-table 1 weight 40 pfc
switch(conf-cee-map)#priority-group-table 2 weight 60
switch(conf-cee-map)#priority-table 2 2 2 1 2 2 2 2
```

iii. Configure the CEE interface as a Layer 2 switch port. Example of configuring the switch port as a 10-Gigabit Ethernet interface.

```
switch(config)#interface tengigabitethernet 0/16
switch(config-if-te-0/16)#switchport
switch(config-if-te-0/16)#no shutdown
switch(config-if)#exit
```

iv. Create an FCoE VLAN and add an interface to it. Example of creating a FCoE VLAN and adding a single interface.

```
switch(config)#vlan classifier rule 1 proto fcoe encap ethv2
switch(config)#vlan classifier rule 2 proto fip encap ethv2
switch(config)#vlan classifier group 1 add rule 1
switch(config)#vlan classifier group 1 add rule 2
switch(config)#interface vlan 1002
switch(conf-if-vl-1002 )#fcf forward
switch(conf-if-vl-1002 )#interface tengigabitethernet 0/16
switch(config-if-te-0/16)#switchport
switch(config-if-te-0/16)#switchport mode converged
switch(config-if-te-0/16)#switchport converged allowed vlan add 1002
switch(config-if-te-0/16)#vlan classifier activate group 1 vlan 1002
switch(config-if-te-0/16)#cee default
switch(config-if-te-0/16)#no shutdown
switch(config-if-te-0/16)#exit
```

Note *Unlike cisco, only one VLAN ID can carry FCoE traffic for now on Brocade 8000. It is their limitation.*

v. Save the Configuration.

```
switch#copy running-config startup-config
```

5. Running NIC & iSCSI Traffic together with DCBx

Note *Please refer iSCSI PDU Offload Initiator chapter to configure iSCSI Initiator.*

Use the following procedure to run NIC and iSCSI traffic together with DCBx enabled.

i. Identify the VLAN priority configured for NIC and iSCSI class of traffic on the switch.
ii. Create VLAN interfaces for running NIC and iSCSI traffic, and configure corresponding VLAN priority.

Example:

Switch is configured with a VLAN priority of 2 and 5 for NIC and iSCSI class of traffic respectively. NIC traffic is run on VLAN10 and iSCSI traffic is run on VLAN20.

Assign proper VLAN priorities on the interface (here eth5), using the following commands on the host machine:

```
[root@host~]# vconfig set_egress_map eth5.10 0 2
[root@host~]# vconfig set_egress_map eth5.20 5 5
```

XVII. FCoE Full Offload Initiator

1. Introduction

Fibre Channel over Ethernet (FCoE) is a mapping of Fibre Channel over selected full duplex IEEE 802.3 networks. The goal is to provide I/O consolidation over Ethernet, reducing network complexity in the Datacenter. Chelsio FCoE initiator maps Fibre Channel directly over Ethernet while being independent of the Ethernet forwarding scheme. The FCoE protocol specification replaces the FC0 and FC1 layers of the Fibre Channel stack with Ethernet. By retaining the native Fibre Channel constructs, FCoE will integrate with existing Fibre Channel networks and management software.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the FCoE full offload Initiator driver is available for the following version(s):

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org linux-5.4.143

2. Software/Driver Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install FCoE full offload initiator driver.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

The driver must be loaded by the root user. Any attempt to load the driver as a regular user will fail.

To load the driver, execute the following:

```
[root@host~]# modprobe csiostor
```

4. Software/Driver Configuration and Fine-tuning

4.1. Configuring Cisco Nexus 5010 and Brocade switch

To configure various Cisco and Brocade switch settings, please refer Software/Driver Configuration and Fine-tuning section of **Data Center Bridging (DCB)** chapter.

4.2. FCoE fabric discovery verification

4.2.1. Verifying Local Ports

Once connected to the switch, use the following command to see if the FIP has gone through and a VN_Port MAC address has been assigned.

Verify if all the FCoE ports are online/ready and a successful FIP has taken place using the following command. The **wwpn** and **state** of the initiator local port can be found under sysfs.

```
[root@host~]# cat /sys/class/fc_host/hostX/port_name
```

```
[root@      ~]# cat /sys/class/fc_host/host0/port_name
0x500074304639f080
[root@      ~]# cat /sys/class/fc_host/host0/port_state
Online
```

Note

- *The hosts under fc_host depends on the number of ports on the adapter used.*
- *Inorder to identify chelsio fc_host from other vendor fc_host, the WWPN always begins with **0x5000743***

Alternatively, the local port information can also be found using:

```
[root@host~]# cat /sys/kernel/debug/csiostor/<pci_id>/lnodes
```

```
[root@ ~]# cat /sys/kernel/debug/csiostor/0000\:81\:00.6/lnodes
****************[Index:  0]***************
device id: 1613956
vnpi: 41092
fcfi: 41120
mac: 0efcfa340020
nport id: 340020
wwnn: 50007433cadb6000
wwpn: 50007433cadb6080
num rnodes:   4
npiv: SUPPORTED
common service params:
        hi ver:00
        low ver:00
        bb credit:10
        word1(31:16) flags:8000
        rcv size:2068
        maxsq_reloff:16711711
        ratov:16711711
        edtov:2000
class service params:
class 1:NOT SUPPORTED
class 2:NOT SUPPORTED
class 3:SUPPORTED
initiator ctl:0
recipient ctl:0
rcv size:2068
Total concurrent seq:0
ee credit:0
open sequence per exch:0

class 4:NOT SUPPORTED
```

4.2.2. Verifying the target discovery

To view the list of targets discovered on a particular FCoE port, follow the below mentioned steps:

i. Determine the WWPN of the initiator local port under *sysfs*. The hosts under *fc_host* depends on the number of ports on the adapter used.

```
[root@host~]# cat /sys/class/fc_host/hostX/port_name
```

ii. After finding the localport, go to the corresponding remote port under sysfs # **cat /sys/class/fc_remote_ports/rport-X:B:R** where X is the Host ID, B is the bus ID and R is the remote port.

```
[root@ ~]# cat /sys/class/fc_remote_ports/rport-0\:0-0/roles
Fabric Port
[root@ ~]# cat /sys/class/fc_remote_ports/rport-0\:0-1/roles
Directory Server
[root@ ~]# cat /sys/class/fc_remote_ports/rport-0\:0-2/roles
Management Server
[root@ ~]# cat /sys/class/fc_remote_ports/rport-0\:0-3/roles
FCP Initiator
[root@ ~]# cat /sys/class/fc_remote_ports/rport-0\:0-4/roles
FCP Initiator
[root@ ~]# cat /sys/class/fc_remote_ports/rport-0\:0-9/roles
FCP Target
```

Note *R can correspond to NameServer, Management Server and other initiator ports logged in to the switch and targets.*

Alternatively, the local ports can also be found using*:*

```
[root@host~]# cat /sys/kernel/debug/csiostor/<pci_id>/lnodes
```

After finding out the WWPN of the local node, to verify the list of discovered targets, use the following command.

```
[root@host~]# cat /sys/kernel/debug/csiostor/<pci_id>/rnodes
```

```
[root@ ~]# cat /sys/kernel/debug/csiostor/0000\:81\:00.6/rnodes
**********Index :  0**********
ssni: 40064
vnpi: 41092
fcfi: 41120
wwnn: 2058000decb1bd41
wwpn: 2003000decb1bd7f
nport id: fffffe
fcp flags: 0
role: fabric
class service params:
class 1:NOT SUPPORTED
class 2:NOT SUPPORTED
class 3:SUPPORTED
class 4:NOT SUPPORTED
**********Index :  1**********
ssni: 40065
vnpi: 41092
fcfi: 41120
wwnn: 2058000decb1bd41
wwpn: 250d000decb1bd40
nport id: fffffc
fcp flags: 0
role: nameserver
class service params:
class 1:NOT SUPPORTED
class 2:NOT SUPPORTED
class 3:SUPPORTED
class 4:NOT SUPPORTED
**********Index :  2**********
ssni: 40067
vnpi: 41092
fcfi: 41120
wwnn: 2058000decb1bd41
wwpn: 250b000decb1bd40
nport id: fffffa
fcp flags: 0
role: nport
class service params:
class 1:NOT SUPPORTED
class 2:NOT SUPPORTED
class 3:SUPPORTED
class 4:NOT SUPPORTED
**********Index :  3**********
ssni: 40068
vnpi: 41092
fcfi: 41120
wwnn: 500a0980892bb831
wwpn: 500a0981992bb831
nport id: 340000
fcp flags: 0
role: target
class service params:
class 1:NOT SUPPORTED
class 2:NOT SUPPORTED
class 3:SUPPORTED
class 4:NOT SUPPORTED
```

4.3. Formatting the LUNs and Mounting the Filesystem

Use *lsscsi -g* to list the LUNs discovered by the initiator.

```
[root@host~]# lsscsi –g
```

```
[root@      ~]# lsscsi -g
[0:0:0:0]    disk    NETAPP   LUN             8010  /dev/sda   /dev/sg0
[0:0:0:1]    disk    NETAPP   LUN             8010  /dev/sdb   /dev/sg1
[0:0:0:2]    disk    NETAPP   LUN             8010  /dev/sdc   /dev/sg2
[0:0:0:3]    disk    NETAPP   LUN             8010  /dev/sdd   /dev/sg3
[0:0:0:4]    disk    NETAPP   LUN             8010  /dev/sde   /dev/sg4
[0:0:0:5]    disk    NETAPP   LUN             8010  /dev/sdf   /dev/sg5
[0:0:0:6]    disk    NETAPP   LUN             8010  /dev/sdg   /dev/sg6
[0:0:0:7]    disk    NETAPP   LUN             8010  /dev/sdh   /dev/sg7
[0:0:0:8]    disk    NETAPP   LUN             8010  /dev/sdi   /dev/sg8
[0:0:0:9]    disk    NETAPP   LUN             8010  /dev/sdj   /dev/sg9
[0:0:0:10]   disk    NETAPP   LUN             8010  /dev/sdk   /dev/sg10
[0:0:0:11]   disk    NETAPP   LUN             8010  /dev/sdl   /dev/sg11
[0:0:0:12]   disk    NETAPP   LUN             8010  /dev/sdm   /dev/sg12
[0:0:0:13]   disk    NETAPP   LUN             8010  /dev/sdn   /dev/sg13
[0:0:0:14]   disk    NETAPP   LUN             8010  /dev/sdo   /dev/sg14
[0:0:0:15]   disk    NETAPP   LUN             8010  /dev/sdp   /dev/sg15
[0:0:0:16]   disk    NETAPP   LUN             8010  /dev/sdq   /dev/sg16
[0:0:0:17]   disk    NETAPP   LUN             8010  /dev/sdr   /dev/sg17
[0:0:0:18]   disk    NETAPP   LUN             8010  /dev/sds   /dev/sg18
[0:0:0:19]   disk    NETAPP   LUN             8010  /dev/sdt   /dev/sg19
[1:0:0:0]    disk    NETAPP   LUN             8010  /dev/sdu   /dev/sg20
[1:0:0:1]    disk    NETAPP   LUN             8010  /dev/sdv   /dev/sg21
[1:0:0:2]    disk    NETAPP   LUN             8010  /dev/sdw   /dev/sg22
[1:0:0:3]    disk    NETAPP   LUN             8010  /dev/sdx   /dev/sg23
[1:0:0:4]    disk    NETAPP   LUN             8010  /dev/sdy   /dev/sg24
[1:0:0:5]    disk    NETAPP   LUN             8010  /dev/sdz   /dev/sg25
[1:0:0:6]    disk    NETAPP   LUN             8010  /dev/sdaa  /dev/sg26
[1:0:0:7]    disk    NETAPP   LUN             8010  /dev/sdab  /dev/sg27
[1:0:0:9]    disk    NETAPP   LUN             8010  /dev/sdac  /dev/sg28
[1:0:0:10]   disk    NETAPP   LUN             8010  /dev/sdad  /dev/sg29
[1:0:0:11]   disk    NETAPP   LUN             8010  /dev/sdae  /dev/sg30
[1:0:0:12]   disk    NETAPP   LUN             8010  /dev/sdaf  /dev/sg31
[1:0:0:15]   disk    NETAPP   LUN             8010  /dev/sdag  /dev/sg32
[3:0:0:0]    disk    NETAPP   LUN             8010  /dev/sdah  /dev/sg33
[3:0:0:1]    disk    NETAPP   LUN             8010  /dev/sdai  /dev/sg34
[3:0:0:2]    disk    NETAPP   LUN             8010  /dev/sdaj  /dev/sg35
[3:0:0:3]    disk    NETAPP   LUN             8010  /dev/sdak  /dev/sg36
[3:0:0:4]    disk    NETAPP   LUN             8010  /dev/sdal  /dev/sg37
[3:0:0:5]    disk    NETAPP   LUN             8010  /dev/sdam  /dev/sg38
[3:0:0:6]    disk    NETAPP   LUN             8010  /dev/sdan  /dev/sg39
[3:0:0:7]    disk    NETAPP   LUN             8010  /dev/sdao  /dev/sg40
[3:0:0:8]    disk    NETAPP   LUN             8010  /dev/sdap  /dev/sg41
[3:0:0:9]    disk    NETAPP   LUN             8010  /dev/sdaq  /dev/sg42
[3:0:0:10]   disk    NETAPP   LUN             8010  /dev/sdar  /dev/sg43
[3:0:0:11]   disk    NETAPP   LUN             8010  /dev/sdas  /dev/sg44
[3:0:0:12]   disk    NETAPP   LUN             8010  /dev/sdat  /dev/sg45
[3:0:0:13]   disk    NETAPP   LUN             8010  /dev/sdau  /dev/sg46
[3:0:0:14]   disk    NETAPP   LUN             8010  /dev/sdav  /dev/sg47
[3:0:0:15]   disk    NETAPP   LUN             8010  /dev/sdaw  /dev/sg48
[3:0:0:16]   disk    NETAPP   LUN             8010  /dev/sdax  /dev/sg49
```

Alternatively, the LUNs discovered by the Chelsio FCoE initiators can be accessed via easily-identifiable ‘udev’ path device files like:

```
[root@host~]# ls /dev/disk/by-path/pci-0000:04:00.0-csio-fcoe
<local_wwpn>:<remote_wwpn>:<lun_wwn>
```

```
[root@      ~]# mount /dev/disk/by-path/pci-0000:03:00.6-csio-fcoe-0x50007430463b7380:0x500a0981892bb831:0x0000000000000000 /mnt/
[root@      ~]# mount
/dev/sdbe5 on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw)
/dev/sdbe1 on /boot type ext3 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
/tmp on /tmp type none (rw,bind)
/var/tmp on /var/tmp type none (rw,bind)
/home on /home type none (rw,bind)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
none on /sys/kernel/debug type debugfs (rw)
gvfs-fuse-daemon on /root/.gvfs type fuse.gvfs-fuse-daemon (rw,nosuid,nodev)
/dev/sdah on /mnt type ext3 (rw)
```

4.4. Creating Filesystem

Create an ext3 filesystem using the following command:

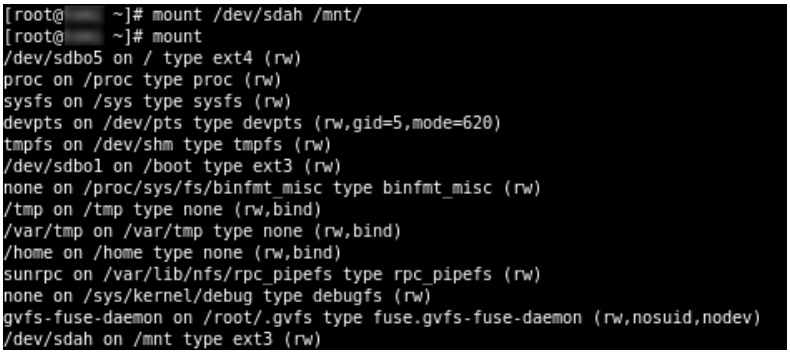
```
[root@host~]# mkfs.ext3 /dev/sdx
```

```
[root@      ~]# mkfs.ext3 /dev/sdah
mke2fs 1.41.12 (17-May-2010)
/dev/sdah is entire device, not just one partition!
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=1 blocks, Stripe width=16 blocks
327680 inodes, 1310720 blocks
65536 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1342177280
40 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 25 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

4.5. Mounting the formatted LUN

The formatted LUN can be mounted on the specified mountpoint using the following command:

```
[root@host~]# mount /dev/sdx /mnt
```



5. Software/Driver Unloading

To unload the driver, run the following command:

```
[root@host~]# modprobe -r csiostor
```

Note *If multipath services are running, unload of FCoE driver is not possible. Stop the multipath service and then unload the driver.*

XVIII. Offload Bonding

1. Introduction

The Chelsio Offload bonding driver provides a method to aggregate multiple network interfaces into a single logical bonded interface effectively combining the bandwidth into a single connection. It also provides redundancy in case one of link fails.

The traffic running over the bonded interface can be fully offloaded to the adapter, thus freeing the CPU from TCP/IP overhead.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the Offload Bonding driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 3.10.0-957.el7.ppc64le (POWER8 LE)
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)
- RHEL 7.5, 3.10.0-862.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- RHEL 6.10, 2.6.32-754.el6.x86_64

- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install Chelsio Offload bonding driver.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

The driver must be loaded by the root user. Any attempt to load the driver as a regular user will fail.

To load the driver (with offload support), run the following command:

```
[root@host~]# modprobe bonding
```

4. Software/Driver Configuration and Fine-tuning

4.1. Offloading TCP traffic over a bonded interface

The Chelsio Offload Bonding driver supports all the bonding modes in NIC Mode. In offload mode (t4_tom loaded) however, only the **balance-rr (mode=0)**, **active-backup (mode=1)**, **balance-xor (mode=2)** and **802.3ad (mode=4)** modes are supported.

To offload TCP traffic over a bond interface, use the following method:

i. Load the network driver with TOE support.

```
[root@host~]# modprobe t4_tom
```

ii. Create a bond interface.

```
[root@host~]# modprobe bonding mode=1 miimon=100
```

Note *On RHEL8.X distributions, max_bonds=1 should be provided additionally.*

iii. Bring up the bond interface and enslave the interfaces to the bond.

```
[root@host~]# ifconfig bond0 up
[root@host~]# ifenslave bond0 ethX ethY
```

Note ***ethX*** *and* ***ethY*** *are interfaces of the same adapter.*

iv. Assign IPv4/IPv6 address to the bond interface.

```
[root@host~]# ifconfig bond0 X.X.X.X/Y
[root@host~]# ifconfig bond0 inet6 add <128-bit IPv6 Address> up
```

v. Disable FRTO on the PEER.

```
[root@host~]# sysctl -w net.ipv4.tcp_frto=0
```

vi. Ping the PEER interface and verify the successful connectivity over the bond interface.

All TCP traffic will be offloaded over the bond interface now.

5. Software/Driver Unloading

To unload the driver, run the following command:

```
[root@host~]# rmmod bonding
```

XIX. Offload Multi-Adapter Failover (MAFO)

1. Introduction

Chelsio's adapters offer a complete suite of high reliability features, including adapter-to-adapter failover. The patented offload Multi-Adapter Failover (MAFO) feature ensures all offloaded traffic continue operating seamless in the face of port failure.

MAFO allows aggregating network interfaces across multiple adapters into a single logical bonded interface, providing effective fault tolerance.

The traffic running over the bonded interface can be fully offloaded to the adapter, thus freeing the CPU from TCP/IP overhead.

Important

- *Portions of this software are covered under US Patent, Failover and migration for full-offload network interface devices: US 8346919 B1*
- *Use of the covered technology is strictly limited to Chelsio ASIC-based soutions.*

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the Offload Multi-Adapter Failover driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64

- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 3.10.0-957.el7.ppc64le (POWER8 LE)
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)
- RHEL 7.5, 3.10.0-862.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

1.2.2. Driver Requirements

Multi-adapter failover feature will work for *Link Down* events caused by:

- Cable unplug on bonded interface.
- Bringing corresponding switch port down.

Note *The feature will not work if the bonded interfaces are administratively taken down.*

2. Software/Driver Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install MAFO feature.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

The driver must be loaded by the root user. Any attempt to load the driver as a regular user will fail.

To load the driver (with offload support), run the following command:

```
[root@host~]# modprobe bonding
```

4. Software/Driver Configuration and Fine-tuning

4.1. Offloading TCP traffic over a bonded interface

The Chelsio MAFO driver supports only the **active-backup (mode=1)** mode. To offload TCP traffic over a bond interface, use the following method:

i. Load the network driver with TOE support.

```
[root@host~]# modprobe t4_tom
```

ii. Create a bond interface.

```
[root@host~]# modprobe bonding mode=1 miimon=100
```

Note *On RHEL8.X distributions, max_bonds=1 should be provided additionally.*

iii. Bring up the bond interface and enslave the interfaces to the bond.

```
[root@host~]# ifconfig bond0 up
[root@host~]# ifenslave bond0 ethX ethY
```

Note ***ethX*** *and* ***ethY*** *are interfaces of different adapters.*

iv. Assign IPv4/IPv6 address to the bond interface.

```
[root@host~]# ifconfig bond0 X.X.X.X/Y
[root@host~]# ifconfig bond0 inet6 add <128-bit IPv6 Address> up
```

v. Disable TCP timestamps.

```
[root@host~]# sysctl -w net.ipv4.tcp_timestamps=0
```

vi. Disable FRTO on the PEER.

```
[root@host~]# sysctl -w net.ipv4.tcp_frto=0
```

vii. Ping the PEER interface and verify the successful connectivity over the bond interface.

All TCP traffic will be offloaded over the bond interface now and fail-over will happen in case of link-down event.

5. Software/Driver Unloading

To unload the driver, run the following command:

```
[root@host~]# rmmod bonding
```

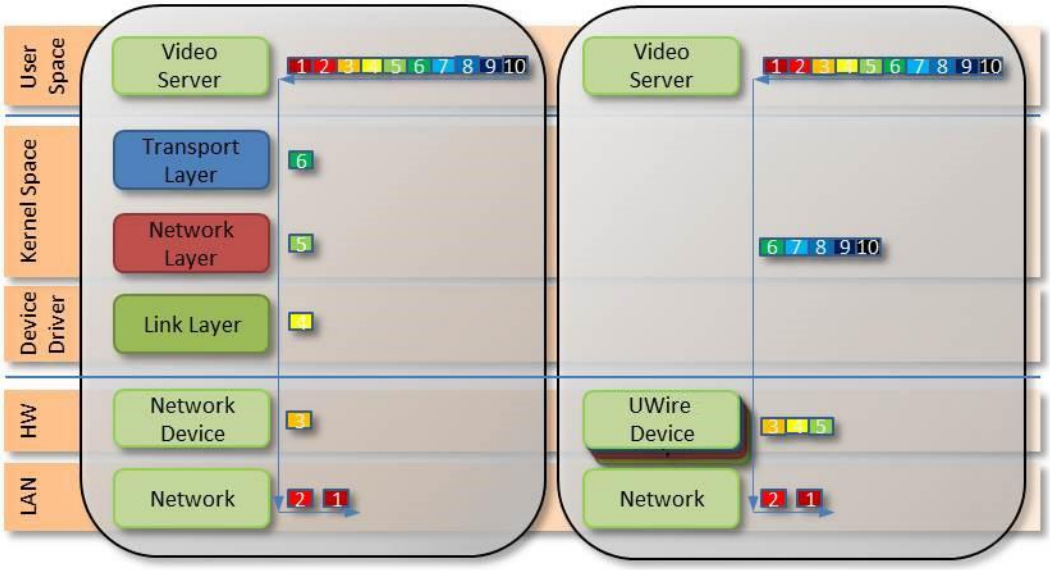
XX. UDP Segmentation Offload and Pacing

1. Introduction

Chelsio's Terminator series of adapters provide UDP segmentation offload and per-stream rate shaping to drastically lower server CPU utilization, increase content delivery capacity, and improve service quality.

Tailored for UDP content, UDP Segmentation Offload (USO) technology moves the processing required to packetize UDP data and rate control its transmission from software running on the host to the network adapter. USO increases performance and dramatically reduces CPU overhead, allowing significantly higher capacity using the same server hardware. Without USO support, UDP server software running on the host needs to packetize payload into frames, process each frame individually through the network stack and schedule individual frame transmission, resulting in millions of system calls, and packet traversals through all protocol layers in the operating system to the network device. In contrast, USO implements the network protocol stack in the adapter, and the host server software simply hands off unprocessed UDP payload in large I/O buffers to the adapter.

The following figure compares the traditional datapath on the left to the USO datapath on the right, showing how per-frame processing is eliminated. In this example, the video server pushes 5 frames at a time. In an actual implementation, a video server pushes 50 frames or more in each I/O, drastically lowering the CPU cycles required to deliver the content.



Pacing is beneficial for several reasons, one example is for Content Delivery Networks (CDNs)/Video On Demand (VOD) providers to avoid receive buffer overflows, smooth out network traffic, or to enforce Service Level Agreements (SLAs).

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the UDP Segmentation Offload and Pacing driver is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Installation

The offload drivers support UDP Segmentation Offload with limited number of connections (1024 connections).To build and install UDP Offload drivers which support large number of offload connections (approx 10K):

Note *10K UDP Segmentation offload connections currently not supported on T6.*

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Run the following command:

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

The driver must be loaded by the root user. Any attempt to load the driver as a regular user will fail.

Run the following commands to load the drivers.

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe t4_tom
```

Though normally associated with the Chelsio TCP Offload engine, the *t4_tom* module is required in order to allow for the proper redirection of UDP socket calls.

4. Software/Driver Configuration and Fine-tuning

4.1. Modifying the Application

To use the UDP offload functionality, the application needs to be modified. Follow the steps mentioned below:

i. Determine the UDP socket file descriptor in the application through which data is sent

ii. Declare and initialize two variables in the application.

```
int fs=1316;
int cl=1;
```

Here,

- *fs* is the UDP packet payload size in bytes that is transmitted on the wire. The minimum value of fs is 256 bytes.
- *cl* is the UDP traffic class (scheduler-class-index) that the user wishes to assign the data stream to. This value needs to be in the range of 0 to 14 for T4/T5 adapters and 0 to 30 for T6 adapters.

The application will function as per the parameters set for that traffic class.

iii. Add socket option definitions.

In order to use *setsockopt()* to set the options to the UDP socket, the following three definitions need to be made:

- SO_FRAMESIZE used for setting frame size, which has the value 291.
- SOL_SCHEDCLASS used for setting UDP traffic class, which has the value 290.
- IPPROTO_UDP used for setting the type of IP Protocol.

```
# define SO_FRAMESIZE 291
# define SOL_SCHEDCLASS 290
# define IPPROTO_UDP 17
```

iv. Use the *setsockopt()* function to set socket options.

```
//Get the UDP socket descriptor variable
setsockopt (sockfd , IPPROTO_UDP, SO_FRAMESIZE, &fs, sizeof(fs));
setsockopt (sockfd , IPPROTO_UDP, SOL_SCHEDCLASS, &cl, sizeof(cl));
```

Here:

- *sockfd* : The file descriptor of the UDP socket
- *&fs* / *&cl* : Pointer to the framesize and class variables
- *sizeof(fs)* / *sizeof(cl)* : The size of the variables

v. Now, compile the application.

4.1.1. UDP offload functionality for RTP data

In case of RTP data, the video server application sends the initial sequence number and the RTP payload. The USO engine segments the payload data, increments the sequence number and sends out the data.

In order to use the UDP offload functionality for RTP data, make the following additions to the steps mentioned above:

i. In step (ii), declare and initialize a new variable in the application.

```
int rtp_header_size=16;
```

Here, *rtp_header_size* is the RTP header size in bytes that the application sends.

ii. In step (iii), define a new macro, *UDP_RTPHEADERLEN* used for setting RTP header length with the value 292.

```
# define UDP_RTPHEADERLEN 292
```

iii. In step (iv), define a new socket option.

```
setsockopt (sockfd,17,UDP_RTPHEADERLEN,&rtp_header_size,
sizeof(rtp_header_size));
```

Here,

- *&rtp_header_size* : pointer to the RTP header length variable
- *sizeof(rtp_header_size)* : the size of the RTP header length variable

4.2. Configuring UDP Pacing

Now that the application has been modified to associate the application's UDP socket to a particular UDP traffic class, the pacing of that socket's traffic can be set using the *cxgbtool* utility.

i. Bring up the network interface.

```
[root@host~]# ifconfig <ethX> up
```

ii. Run the following command.

```
[root@host~]# cxgbtool <ethX> sched-class params type packet level cl-rl
mode flow rate-unit bits rate-mode absolute channel <Channel No.> class
<scheduler-class-index> max-rate <maximum-rate> pkt-size <Packet size>
```

Here,

- *ethX* is the Chelsio interface
- *Channel No.* is the port on which data is flowing (0-3)
- *scheduler-class-index* is the UDP traffic class (0-14 for T4/T5 adapters and 0-30 for T6 adapters) set in the SOL_SCHEDCLASS socket option in the application in section 4.1.
- *maximum-rate* is the bit rate (Kbps) for this UDP stream. This value should be in the range of 50 Kbps to 50 Mbps for T4 adapters. For T5/T6 adapters, the value should be in the range of 100 kbps to 1 Gbps.
- *Packet size* is the UDP packet payload size in bytes; it should be equal to the value set in the *SO_FRAMESIZE* socket option in the application in section 4.1.

Example:

The user wants to transfer UDP data on port 0 of the adapter using the USO engine. The application has been modified as shown in section 4.1. In order to set a bit rate of 10Mbps for traffic class 1 with payload size of 1316 on port 0, the following invocation of *cxgbtool* is used:

```
[root@host~]# cxgbtool ethX sched-class params type packet level cl-rl mode
flow  rate-unit bits rate-mode absolute channel 0 class 1 max-rate 10000
pkt-size 1316
```

Note *To get an accurate bit rate per class, data sent by the application to the sockets should be a multiple of the value set for the "pkt-size" parameter. In above example, IO size sent by application should be a multiple of 1316.*

Note *Linux Unified Wire currently supports 10240 offload UDP connections. If the application needs to establish more than 10240 UDP connections, it can check the return code of ENOSPC from a send() or sendto() call and close this socket and open a new one that uses the kernel UDP stack.*

4.3. Enabling Offload

Load the offload drivers and bring up the Chelsio interface.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig ethX <IP> up
```

The traffic will be offloaded over the Chelsio interface now. To see the number of connections offloaded, run the following command:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<bus-id>/tids
```

```
[root@host ~]# cat /sys/kernel/debug/cxgb4/0000\:81\:00.4/tids
Connections in use: 0
TID range: 64..2047/3072..19455, in use: 0/0
STID range: 2048..2543, in use-IPv4/IPv6: 0/0
ATID range: 0..8191, in use: 0
FTID range: 2560..3055
HPFTID range: 0..63
UOTID range: 19456..20479, in use: 5
HW TID usage: 0 IP users, 0 IPv6 users
```

Where,

UOTID is the number of UDP offload connections.

5. Software/Driver Unloading

Reboot the system to unload the driver. To unload without rebooting, refer Unloading the TOE driver section of **Network (NIC/TOE)** chapter.

XXI. Offload IPv6

1. Introduction

The growth of the Internet has created a need for more addresses than are possible with IPv4. Internet Protocol version 6 (IPv6) is a version of the Internet Protocol (IP) designed to succeed the Internet Protocol version 4 (IPv4).

Chelsio's Offload IPv6 feature provides support to fully offload IPv6 traffic to the Unified Wire adapter.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the Offload IPv6 feature is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 3.10.0-957.el7.ppc64le (POWER8 LE)
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)
- RHEL 7.5, 3.10.0-862.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- RHEL 6.10, 2.6.32-754.el6.x86_64

- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Installation

2.1. Pre-requisites

Please make sure that the following requirements are met before installation:

- IPv6 must be enabled in your system (enabled by default).
- Unified Wire must be installed with IPv6 support as explained in the Unified Wire chapter.

2.2. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install Unified Wire with IPv6 support.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

After installing Unified Wire package and rebooting the host, load the NIC (*cxgb4*) and TOE (*t4_tom*) drivers. The drivers must be loaded by the root user. Any attempt to load the drivers as a regular user will fail.

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe t4_tom
```

4. Software/Driver Configuration and Fine-tuning

i. Load the Offload capable drivers.

```
[root@host~]# modprobe t4_tom
```

ii. Bring up the interface and ensure that IPv6 Link Local address is present.

```
[root@host~]# ifconfig ethX up
```

```
enp6s0f4d1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::207:43ff:fe4a:8ab8  prefixlen 64  scopeid 0x20<link>
        ether 00:07:43:4a:8a:b8  txqueuelen 1000  (Ethernet)
        RX packets 1532  bytes 128688 (125.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2289  bytes 215930 (210.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 232
```

On some distributions, `ONBOOT="yes"` should be added to interface network script for the interface to come up automatically with IPv6 Link Local address.

iii. Configure the the required IPv6 address.

```
[root@host~]# ifconfig ethX inet6 add <IPv6 address>
```

iv. All the IPv6 traffic over the Chelsio interface will be offloaded now. To see the number of connections offloaded, run the following command:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<bus-id>/tids
```

5. Software/Driver Unloading

5.1. Unloading the NIC Driver

To unload the NIC driver, run the following command:

```
[root@host~]# rmmod cxgb4
```

5.2. Unloading the TOE Driver

Please reboot the system to unload the TOE driver. To unload without rebooting, refer Unloading the TOE driver section of **Network (NIC/TOE)** chapter.

XXII. WD Sniffing and Tracing

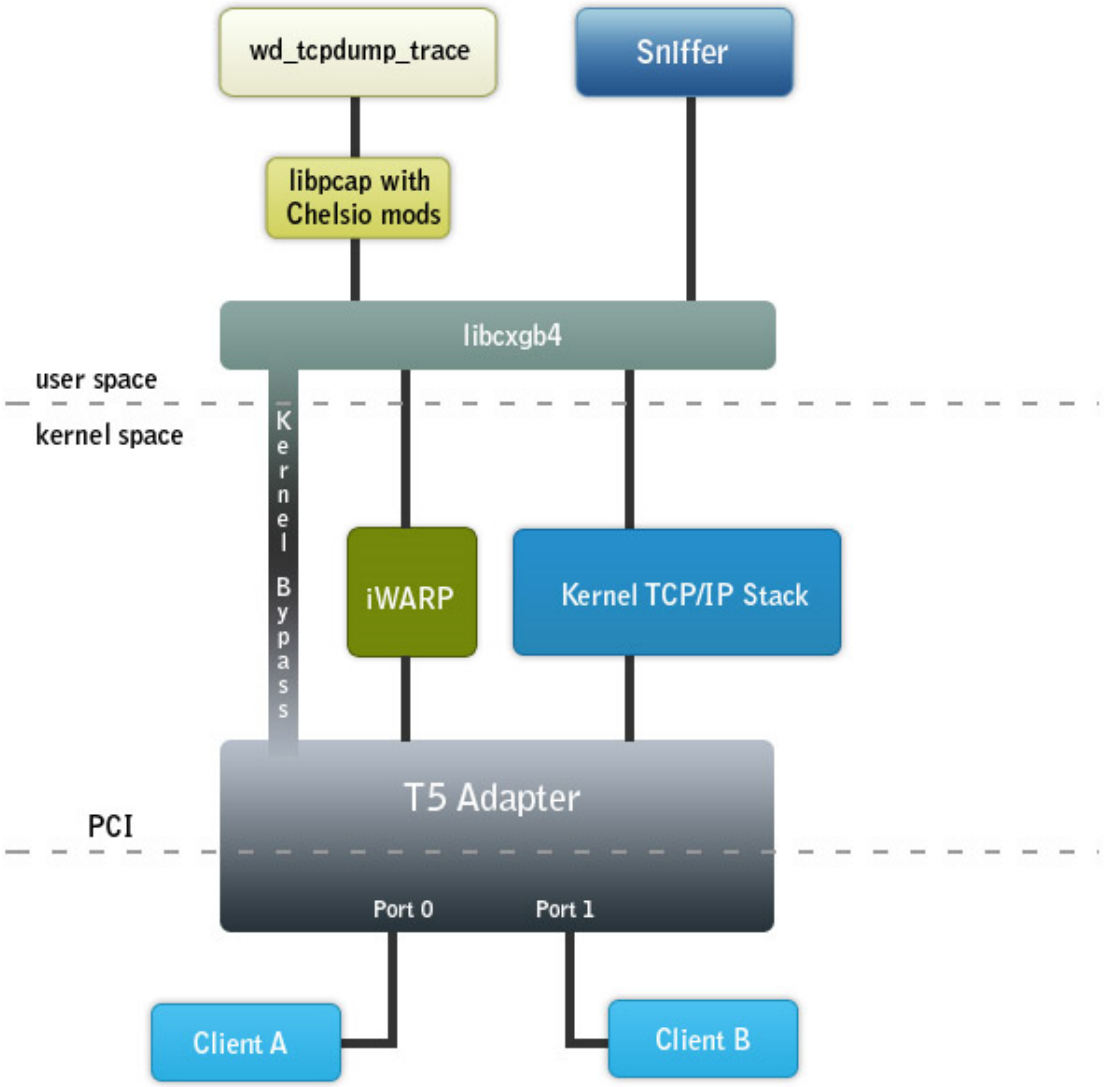
1. Theory of Operation

The objective of these utilities (*wd_sniffer* and *wd_tcpdump_trace*) is to provide sniffing and tracing capabilities by making use of Chelsio adapter's hardware features.

- Sniffer is a tool to measure bandwidth and involves targeting specific multicast traffic and sending it directly to user space.
    a) Get a Queue (raw QP) idx.
    b) Program a filter to redirect specific traffic to the raw QP queue.

- Tracer - All tapped traffic is forwarded to user space and also pushed back on the wire via the internal loop back mechanism.
    a) Get a Queue (raw QP) idx.
    b) Set the adapter in loop back.
    c) Connect Client A and B to ports 0 and 1 or ports 2 and 3.
    d) Enable tracing.

In either mode, the targeted traffic bypasses the kernel TCP/IP stack and is delivered directly to user space by means of an RX queue.



Schematic diagram of sniffer and tracer

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T580-CR
- T580-LP-CR

- T540-CR
- T540-LP-CR
- T540-BT
- T520-CR
- T520-LL-CR
- T520-BT

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the WD Sniffing and Tracing utility is available for the following version:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install Sniffer & Tracer utilities and iWARP driver.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Usage

3.1. Installing Basic Support

iw_cxgb4 (Chelsio iWARP driver) and *cxgb4* (Chelsio NIC driver) drivers should be compiled and loaded before running the utilities. Refer to the **Software/Driver Loading** section for each driver and follow the instructions mentioned before proceeding.

3.2. Using Sniffer (wd_*sniffer)*

1. Setup

Wire filter sniffing requires 2 systems with one machine having a Chelsio card.

The machines should be setup in the following manner:

Machine A <---------> Machine B

192.168.1.100 192.168.1.200

2. Procedure

On the Device Under Test (DUT), start sniffer.

```
[root@host~]# wd_sniffer -T 20 -s 1000 -I <MAC address of interface to
sniff>
```

Start traffic on the PEER and watch the sniffer.

The sniffer will receive all packets as fast as possible, update the packet count, and then discard the data. Performance is a full 10Gbps for packet size 1000.

3.3. Using Tracer (wd_tcpdump_trace)

1. Setup

Wire tapping requires 3 systems with one machine having a Chelsio card with two or more ports. The machines should be setup in the following manner:

DUT: Machine B

PEER: Machine A <-----> (port 0) (port 1) <-----> PEER: Machine C

192.168.1.100 IP-dont-care IP-dont-care 192.168.1.200

2. Procedure

Run `wd_tcpdump_trace -i iface` on the command prompt where *iface* is one of the interfaces whose traffic you want to trace. In the above diagram its port 0 or port 1.

```
[root@host~]# wd_tcpdump_trace -i <iface>
```

Use any tool (like ping or ssh) to run traffic between machines A and B. The traffic should successfully make it from end to end and wd_tcpdump_trace on the DUT should show the tapped traffic. The below options can be provided additionally to capture more packets.

```
[root@host~]# wd_tcpdump_trace -i <iface> -s 64 -B 1024000 -w capture.pcap
```

Note *Please refer* `wd_tcpdump_trace -h` *for more information on the above options.*

XXIII. Classification and Filtering

1. Introduction

Classification and Filtering feature enhances network security by controlling incoming traffic as they pass through network interface based on source and destination addresses, protocol, source and receiving ports, or the value of some status bits in the packet. This feature can be used in the ingress path to:

- Steer ingress packets that meet ACL (Access Control List) accept criteria to a particular receive queue.
- Switch (proxy) ingress packets that meet ACL accept criteria to an output port, with optional header rewrite.
- Drop ingress packets that meet ACL accept criteria.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T62100-SO-CR*
- T61100-OCP*
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP*
- T6225-SO-CR*
- T580-CR
- T580-LP-CR
- T580-SO-CR*
- T580-OCP-SO*
- T540-CR
- T540-LP-CR
- T540-SO-CR*
- T540-BT
- T520-CR
- T520-LL-CR
- T520-SO-CR*
- T520-OCP-SO*
- T520-BT

* *Hash filter not supported.*

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the Classification and Filtering feature is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 3.10.0-957.el7.ppc64le (POWER8 LE)
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)
- RHEL 7.5, 3.10.0-862.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other kernel versions have not been tested and are not guaranteed to work.

2. LE-TCAM Filters

The default (*Unified Wire)* configuration tuning option allows you to create LE-TCAM filters, which has a limit of 496 for T5, and 560 for T6 adapters. For T5 adapters, all available filter indices can be optionally configured as high priority. In case of T6 adapters, the following filter indices are available:

- High priority indices (PRIO): 0 to X
- Normal indices: X+1 to X+1+495

Where *X* is the upper limit in the *HPFTID range*, mentioned in the tids file (*/sys/kernel/debug/cxgb4/<bus-id>/tids*).

```
[root@    ~]# cat /sys/kernel/debug/cxgb4/0000\:02\:00.4/tids
Connections in use: 0
TID range: 64..2047/3072..19455, in use: 0/0
STID range: 2048..2543, in use-IPv4/IPv6: 0/0
ATID range: 0..8191, in use: 0
FTID range: 2560..3055
HPFTID range: 0..63
UOTID range: 19456..20479, in use: 0
HW TID usage: 0 IP users, 0 IPv6 users
```

For example, if the upper hpftid limit is 63, then high priority indices will be from 0 to 63 and normal indices will be from 64 to 559. It is mandatory to add **prio 1** when creating high priority filter rules.

Note *T6 SO adapters currently do not support high priority indices. Therefore only 496 LE-TCAM filters can be created.*

2.1. Configuration

2.1.1. Filter Modes

The Classification and Filtering feature is configured by specifying the filter modes in the firmware configuration file (*t6-config.txt* for T6 adapters; *t5-config.txt* for T5 adapters) located in */lib/firmware/cxgb4/* . The following filter tuples are present in filter modes:

fcoe	: Fibre Channel over Ethernet frames
port	: Packet ingress physical port number
vnic_id	: VF ID in MPS TCAM (*Currently not supported*) and outer VLAN ID, Encapsulation
vlan	: Inner VLAN ID
Tos	: Type of Service
protocol	: IP protocol number (ICMP=1, TCP=6, UDP=17, etc)
Ethertype	: Layer 2 EtherType
Macmatch	: MAC index in MPS TCAM
mpshittype	: MAC address "match type" (none, unicast, multicast, promiscuous, broadcast)
fragmentation	: Fragmented IP packets

Note Adapter initialization will fail if *filterMask* contains a tuple which is not present in *filterMode*.

2.1.2. Supported Filter Combinations

The following combination is set by default and packets will be matched accordingly:

- For T5/T6:

```
filterMode = fcoemask, srvrsram, fragmentation, mpshittype, protocol, vlan,
port, fcoe
```

- For T4:

```
filterMode = fragmentation, mpshittype, protocol, vlan, port, fcoe
```

Serial #	Filter Combination
1	*fragmentation, mpshittype, macmatch, ethertype, protocol, port*
2	*fragmentation, mpshittype, macmatch, ethertype, protocol, fcoe*
3	*fragmentation, mpshittype, macmatch, ethertype, tos, port*
4	*fragmentation, mpshittype, macmatch, ethertype, tos, fcoe*
5	*fragmentation, mpshittype, macmatch, ethertype, port, fcoe*
6	*fragmentation, mpshittype, macmatch, protocol, tos, port, fcoe*
7	*fragmentation, mpshittype, macmatch, protocol, vlan, fcoe*
8	*fragmentation, mpshittype, macmatch, protocol, vnic_id, fcoe*
9	*fragmentation, mpshittype, macmatch, tos, vlan, fcoe*
10	*fragmentation, mpshittype, macmatch, tos, vnic_id, fcoe*
11	*fragmentation, mpshittype, macmatch, vlan, port, fcoe*
12	*fragmentation, mpshittype, macmatch, vnic_id, port, fcoe*
13	*fragmentation, mpshittype, ethertype, protocol, tos, port, fcoe*
14	*fragmentation, mpshittype, ethertype, vlan, port*
15	*fragmentation, mpshittype, ethertype, vlan, fcoe*
16	*fragmentation, mpshittype, ethertype, vnic_id, port*
17	*fragmentation, mpshittype, ethertype, vnic_id, fcoe*
18	*fragmentation, mpshittype, protocol, tos, vlan, port*
19	*fragmentation, mpshittype, protocol, tos, vlan, fcoe*
20	*fragmentation, mpshittype, protocol, tos, vnic_id, port*
21	*fragmentation, mpshittype, protocol, tos, vnic_id, fcoe*
22	*fragmentation, mpshittype, protocol, vlan, port, fcoe*
23	*fragmentation, mpshittype, protocol, vnic_id, port, fcoe*
24	*fragmentation, mpshittype, tos, vlan, port, fcoe*
25	*fragmentation, mpshittype, tos, vnic_id, port, fcoe*
26	*fragmentation, mpshittype, vlan, vnic_id, fcoe*
27	*fragmentation, macmatch, ethertype, protocol, port, fcoe*
28	*fragmentation, macmatch, ethertype, tos, port, fcoe*
29	*fragmentation, macmatch, protocol, vlan, port, fcoe*
30	*fragmentation, macmatch, protocol, vnic_id, port, fcoe*
31	*fragmentation, macmatch, tos, vlan, port, fcoe*
32	*fragmentation, macmatch, tos, vnic_id, port, fcoe*
33	*fragmentation, ethertype, vlan, port, fcoe*

34	*fragmentation, ethertype, vnic_id, port, fcoe*
35	*fragmentation, protocol, tos, vlan, port, fcoe*
36	*fragmentation, protocol, tos, vnic_id, port, fcoe*
37	*fragmentation, vlan, vnic_id, port, fcoe*
38	*mpshittype, macmatch, ethertype, protocol, port, fcoe*
39	*mpshittype, macmatch, ethertype, tos, port, fcoe*
40	*mpshittype, macmatch, protocol, vlan, port*
41	*mpshittype, macmatch, protocol, vnic_id, port*
42	*mpshittype, macmatch, tos, vlan, port*
43	*mpshittype, macmatch, tos, vnic_id, port*
44	*mpshittype, ethertype, vlan, port, fcoe*
45	*mpshittype, ethertype, vnic_id, port, fcoe*
46	*mpshittype, protocol, tos, vlan, port, fcoe*
47	*mpshittype, protocol, tos, vnic_id, port, fcoe*
48	*mpshittype, vlan, vnic_id, port*

Important *Using any other filter mode combination is strictly not supported.*

2.1.3. Changing default filter mode

Based on your requirement, you can change the default filter mode to any one of the combinations mentioned in the table above. To do so, replace the default mode with the chosen mode in firmware configuration file (*t6-config.txt* for T6 adapters; *t5-config.txt* for T5 adapters) located in */lib/firmware/cxgb4/*.

For example, if you want to filter traffic based on *ethtype* value in the packets for T6 adapters, replace the default filterMode,

```
filterMode = fcoemask, srvrsram, fragmentation, mpshittype, protocol, vlan,
port, fcoe
```

with

```
filterMode = fragmentation, mpshittype, macmatch, ethertype, protocol, port
```

The network driver needs to be reloaded next using the following command:

```
[root@host~]# rmmod cxgb4
[root@host~]# modprobe cxgb4
```

2.2. Creating Filter Rules

Network driver (*cxgb4*) must be installed and loaded before setting the filter rule.

i. If you haven't done already, run the Unified Wire Installer with the appropriate configuration tuning option to install the network driver.
ii. Load the network driver and bring up the Chelsio interface.

```
[root@host~]# modprobe cxgb4
[root@host~]# ifconfig ethX up
```

iii. Now, create filter rules using *cxgbtool*.

```
[root@host~]#cxgbtool ethx filter <index> action [pass/drop/switch] <prio 1>
<hitcnts 1>
```

Where,

ethX	: Chelsio interface
index	: positive number set as filter id. 0-495 for T5 adapters; 0-559 for T6 adapters. Should be an even number while creating IPv6 filter.
action	: Ingress packet disposition
pass	: Ingress packets will be passed through set ingress queues
switch	: Ingress packets will be routed to an output port with optional header rewrite.
drop	: Ingress packets will be dropped.
prio 1	: Optional for T5. Mandatory for T6 indices 0-63; Should not be added for T6 indices 64-559
hitcnts 1	: To enable hit counts in cxgbtool filter show output.

Note *In case of multiple filter rules, the rule with the lowest filter index takes higher priority.*

2.2.1. Examples

- ***drop* action**

```
[root@host~]# cxgbtool ethX filter 100 action drop fip 192.168.1.5
```

The above filter rule will drop all ingress packets from source IP 192.168.1.5. Remaining packets will be sent to the host.

- ***pass* action**

```
[root@host~]# cxgbtool ethX filter 100 action pass lport 10001 fport 355
queue 2
```

The above filter rule will pass all ingress packets that match destination port 10001 and source port 355 to ingress queue 2 for load balancing. Remaining packets will be sent to the host.

- ***switch* action**

```
[root@host~]# cxgbtool ethX filter 100 action switch iport 0 eport 1 ivlan 3
```

The above filter rule will route all ingress packets that match VLAN id 3 from port 0 of Chelsio adapter to port 1. Remaining packets will be sent to the host.

- ***prio* option**

To filter offloaded ingress packets, use the `prio` argument with the above command:

```
[root@host~]# cxgbtool ethx filter <index> action <pass/drop/switch> prio 1
```

Where index is a positive integer set as filter id. 0-495 for T5 adapters and 0-63 for T6 adapters.

Note *For more information on additional parameters, refer cxgbtool manual by running the `man cxgbtool` command.*

2.3. Listing Filter Rules

To list the filters set, run the following command:

```
[root@host~]# cxgbtool ethX filter show
```

OR

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<bus-id>/filters
```

2.4. Removing Filter Rules

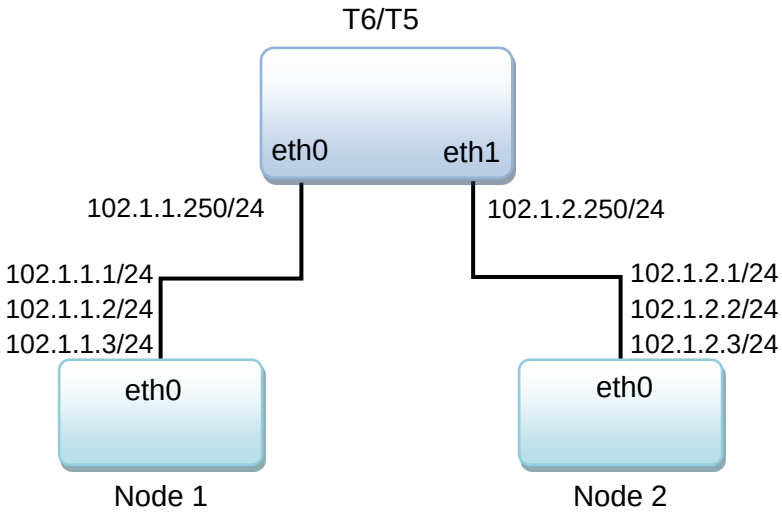
To remove a filter, run the following command with the corresponding filter rule index:

```
[root@host~]# cxgbtool ethX filter <index> <delete|clear>
```

Note *For more information on additional parameters, refer cxgbtool manual by running the `man cxgbtool` command*

2.5. Layer 3 Example

Here's an example on how to achieve L3 routing functionality:



- **Follow these steps on Node 1**

i. Configure IP address and enable the 3 interfaces.

```
[root@host~]# ifconfig eth0 102.1.1.1/24 up
[root@host~]# ifconfig eth0:2 102.1.1.2/24 up
[root@host~]# ifconfig eth0:3 102.1.1.3/24 up
```

ii. Setup a static OR default route towards T6/T5 router to reach 102.1.2.0/24 network.

```
[root@host~]# route add -net 102.1.2.0/24 gw 102.1.1.250
```

- **Follow these steps on Node 2**

i. Configure IP address and enable the 3 interfaces.

```
[root@host~]# ifconfig eth0 102.1.2.1/24 up
[root@host~]# ifconfig eth0:2 102.1.2.2/24 up
[root@host~]# ifconfig eth0:3 102.1.2.3/24 up
```

ii. Setup a static OR default route towards T6/T5 router to reach 102.1.1.0/24 network.

```
[root@host~]# route add -net 102.1.1.0/24 gw 102.1.2.250
```

- **Follow these steps on machine with T6/T5 adapter**

i. Configure IP address and enable the 2 interfaces.

```
[root@host~]# ifconfig eth0 102.1.1.250/24 up
[root@host~]# ifconfig eth1 102.1.2.250/24 up
```

ii. Create filter rule to send packets for 102.1.2.0/24 network out via eth1 interface.

```
[root@host~]# cxgbtool eth0 filter 100 lip 102.1.2.0/24 hitcnts 1 action
switch  eport 1 smac 00:07:43:04:96:48 dmac 00:07:43:12:D4:88
```

Where, `smac` is the MAC address of eth1 interface on T6/T5 adapter machine and `dmac` is the MAC address of eth0 interface on Node 2.

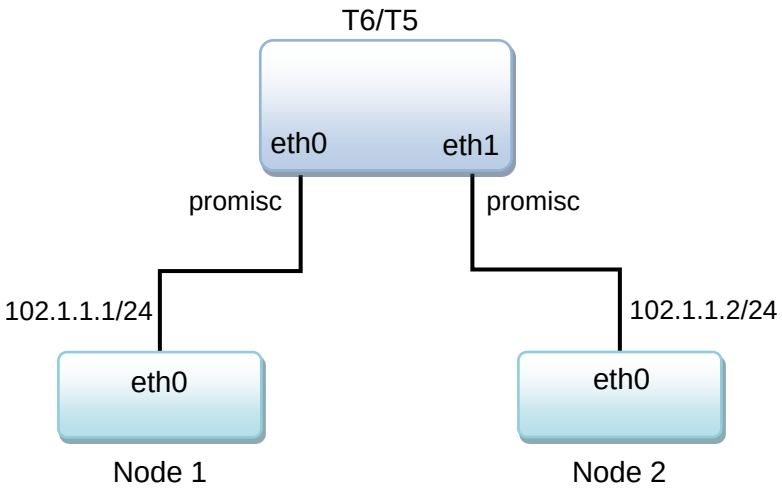
iii. Create filter rule to send packets for 102.1.1.0/24 network out via eth0 interface.

```
[root@host~]# cxgbtool eth0 filter 101 lip 102.1.1.0/24 hitcnts 1 action
switch eport 0 smac 00:07:43:04:96:40 dmac 00:07:43:04:7D:50
```

Where, `smac` is the MAC address of eth0 interface on T6/T5 adapter machine and `dmac` is the MAC address of eth0 interface on Node 1.

2.6. Layer 2 Example

Here's an example on how to achieve L2 switching functionality. The following will only work on kernel 3.10 and above.



- **Follow these steps on Node 1**

i. Configure IP address and enable the interface.

```
[root@host~]# ifconfig eth0 102.1.1.1/24 up
```

ii. Setup ARP entry to reach 102.1.1.2

```
[root@host~]# arp -s 102.1.1.2 00:07:43:12:D4:88
```

- **Follow these steps on Node 2**

i. Configure IP address and enable the interface.

```
[root@host~]# ifconfig eth0 102.1.1.2/24 up
```

ii. Setup ARP entry to reach 102.1.1.1

```
[root@host~]# arp -s 102.1.1.1 00:07:43:04:7D:50
```

- **Follow these steps on machine with T6/T5 adapter**

i. Update filterMode value with below combination in */lib/firmware/cxgb4/t6-config.txt* to enable matching based on macidx (use *t5-config.txt* for T5 adapters).

```
filterMode = fragmentation, macmatch, mpshittype, protocol, tos, port, fcoe
```

ii. Unload and re-load the *cxgb4* driver.

iii. Enable promiscuous mode on both the interfaces on T6/T5 adapter machine.

```
[root@host~]# ifconfig eth0 up promisc
[root@host~]# ifconfig eth1 up promisc
```

iv. Build and install latest iproute2 package.

v. Add fdb entry corresponding to Node-2 on T6/T5's eth0 interface.

```
[root@host~]# bridge fdb add 00:07:43:12:D4:88 dev eth0 self
```

vi. Add fdb entry corresponding to Node-1 on T6/T5's eth1 interface.

```
[root@host~]# bridge fdb add 00:07:43:04:7D:50 dev eth1 self
```

vii. Both MAC entries should show up in MPS table. Run the following command to view the table and note the index (*idx* field) of the entries:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/0000\:01\:00.4/mps_tcam | more
```

viii. Create a filter to match incoming packet's *dst-mac 00:07:43:12:d4:88* with particular mac-idx and switch it out via eport 1.

```
[root@host~]# cxgbtool eth0 filter 100 macidx 5 action switch eport 1
hitcnts 1
```

ix. Create a filter to match incoming packet's *dst-mac 00:07:43:04:7d:50* with particular mac-idx and switch it out via eport 0.

```
[root@host~]# cxgbtool eth0 filter 101 macidx 7 action switch eport 0
hitcnts 1
```

2.7. Filtering VF traffic

To filter VF traffic, replace the default filterMode in the firmware configuration file (*t6-config.txt* for T6 adapters; *t5-config.txt* for T5 adapters) located in */lib/firmware/cxgb4/* with any combination containing *vnic_id.*

```
 filterMode = fragmentation, mpshittype, protocol, tos, vnic_id, port
```

The network driver needs to be reloaded next using the following command:

```
[root@host~]# rmmod cxgb4
[root@host~]# modprobe cxgb4
```

Instantiate the required VFs on the host and assign them to the Virtual Machines (VMs). Bring up the VFs in the VMs and note the corresponding MAC addresses.

Note *For information on VFs and instaniating them, please refer to* Instantiate Virtual Functions (SR-IOV) *section of the* ***Virtual Function Network (vNIC)*** *chapter.*

On the host, check the MPS TCAM entry for the VF MAC and note the corresponding VF ID.

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<pci_bus_id>/mps_tcam | less
```

Idx	Ethernet address	Mask	VNI	Mask	IVLAN	Vld	DIP_Hit	Lookup	Port	Vld	Ports	PF	VF
0	01:80:c2:00:00:0e	fffffffffff	-	-	-	N	-	O	0	Y	0x3	7	104
1	00:00:00:00:00:00	fffffffffff	-	-	-	N	-	O	0	Y	0x3	4	68
2	00:07:43:3c:a8:40	fffffffffff	-	-	-	N	-	O	0	Y	0x1	4	68
3	01:00:5e:00:00:01	fffffffffff	-	-	-	N	-	O	0	Y	0x3	4	68
4	00:07:43:3c:a8:48	fffffffffff	-	-	-	N	-	O	0	Y	0x2	4	69
5	33:33:00:00:00:01	fffffffffff	-	-	-	N	-	O	0	Y	0x3	4	68
6	33:33:ff:3c:a8:40	fffffffffff	-	-	-	N	-	O	0	Y	0x1	4	68
7	33:33:ff:3c:a8:48	fffffffffff	-	-	-	N	-	O	0	Y	0x2	4	69
8	06:44:3c:a8:40:00	fffffffffff	-	-	-	N	-	O	0	Y	0x1	0	0
9	06:44:3c:a8:40:01	fffffffffff	-	-	-	N	-	O	0	Y	0x1	0	1
10	06:44:3c:a8:40:02	fffffffffff	-	-	-	N	-	O	0	Y	0x1	0	2
11	06:44:3c:a8:40:03	fffffffffff	-	-	-	N	-	O	0	Y	0x1	0	3
12	06:44:3c:a8:40:04	fffffffffff	-	-	-	N	-	O	0	Y	0x1	0	4
13	06:44:3c:a8:40:05	fffffffffff	-	-	-	N	-	O	0	Y	0x1	0	5
14	06:44:3c:a8:40:06	fffffffffff	-	-	-	N	-	O	0	Y	0x1	0	6
15	06:44:3c:a8:40:07	fffffffffff	-	-	-	N	-	O	0	Y	0x1	0	7
16	-												

Apply filter rules on the Host using *cxgbtool*.

```
[root@host~]# cxgbtool ethX filter <index> vf <vf_id> action
[pass/drop/switch]
```

Example:

i. 4 VFs (VF0, VF1, VF2, Vf3) are instantiated on PF0.

```
[root@host~]# modprobe cxgb4
[root@host~]# echo 4 >
/sys/class/net/ethX/device/driver/<bus_id>/sriov_numvfs
```

ii. 1 VM was brought up with VF2. cxgb4vf was loaded on the VM and the VF was brought up.

```
[root@host~]# ifconfig enp8s2
enp8s2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
ether 06:44:3c:a8:40:02  txqueuelen 1000  (Ethernet)
```

iii. Observe the VF id for the MAC address *06:44:3c:a8:40:02* in mps_tcam.
iv. Apply the following filter rule on the host to drop all ingress packets to VF2 originating from source IP 192.168.1.5. Remaining packets will be sent to the VF.

```
[root@host~]# cxgbtool ethX filter 101 vf 2 fip 192.168.1.5 action drop
```

3. Hash/DDR Filters

If you wish to create large number of filters, select one of the below configuration tuning options during Unified Wire installation:

- High Capacity Hash Filter: Allows you to create ~0.5 million filter rules. Can run non-offload NIC traffic.
- Unified Wire (Default): Allows you to create ~18k filter rules. Can run all offload traffic.

You can create both LE-TCAM and Hash/DDR filters in the above configurations.

Note *T5/T6 SO adapters do not support Hash Filters as they are memory free. Up to 496 LE-TCAM filters are supported with Hash Filter configurations.*

Hash filters are created based on *filterMask* tuples in firmware configuration file (*t6-config.txt* for T6 adapters; *t5-config.txt* for t5 adapters) located in */lib/firmware/cxgb4/*. *filterMask* tuples should be either subset of or equal to *filterMode* tuples.

Hash filters are exact match filters. Hence, when you enable more fields (tuples) in *filterMask*, you must create a filter rule with exactly same tuples as mentioned in *filterMask*.

3.1. Configuration

3.1.1. Filter Modes

The Classification and Filtering feature is configured by specifying the filter modes in the firmware configuration file located in */lib/firmware/cxgb4/*

Adapter initialization will fail if *filterMask* contains a tuple which is not present in *filterMode*.

The following are the filter tuples supported with Hash Filters:

fcoe	: Fibre Channel over Ethernet frames
port	: Packet ingress physical port number
vnic_id	: VF ID in MPS TCAM (*Currently not supported*) and outer VLAN ID
vlan	: Inner VLAN ID
tos	: Type of Service
protocol	: IP protocol number (ICMP=1, TCP=6, UDP=17, etc)
ethertype	: Layer 2 EtherType
macmatch	: MAC index in MPS TCAM
mpshittype	: MAC address "match type" (none,unicast,multicast,promiscuous,broadcast)

3.1.2. Supported Filter Combinations

The following table lists the supported FilterMode combinations.

Serial #	Filter Combination
1	*fragmentation, mpshittype, macmatch, ethertype, protocol, port*
2	*fragmentation, mpshittype, macmatch, ethertype, protocol, fcoe*
3	*fragmentation, mpshittype, macmatch, protocol, tos, port, fcoe*
4	*fragmentation, mpshittype, macmatch, protocol, vlan, fcoe*
5	*fragmentation, mpshittype, macmatch, protocol, vnic_id, fcoe*
6	*fragmentation, mpshittype, ethertype, protocol, tos, port, fcoe*
7	*fragmentation, mpshittype, protocol, tos, vlan, port*
8	*fragmentation, mpshittype, protocol, tos, vlan, fcoe*
9	*fragmentation, mpshittype, protocol, tos, vnic_id, port*
10	*fragmentation, mpshittype, protocol, tos, vnic_id, fcoe*
11	*fragmentation, mpshittype, protocol, vlan, port, fcoe*
12	*fragmentation, mpshittype, protocol, vnic_id, port, fcoe*
13	*fragmentation, macmatch, ethertype, protocol, port, fcoe*
14	*fragmentation, macmatch, protocol, vlan, port, fcoe*
15	*fragmentation, macmatch, protocol, vnic_id, port, fcoe*
16	*fragmentation, protocol, tos, vlan, port, fcoe*
17	*fragmentation, protocol, tos, vnic_id, port, fcoe*
18	*mpshittype, macmatch, ethertype, protocol, port, fcoe*
19	*mpshittype, macmatch, protocol, vlan, port*
20	*mpshittype, macmatch, protocol, vnic_id, port*
21	*mpshittype, protocol, tos, vlan, port, fcoe*
22	*mpshittype, protocol, tos, vnic_id, port, fcoe*

Important *Using any other filter mode combination is strictly not supported.*

3.1.3. Changing default filter mode

Based on your requirement, you can change the default filter mode to any one of the combinations mentioned above. Replace the default filterMode with the chosen mode in firmware configuration file (*t6-config.txt* for T6 adapters; *t5-config.txt* for T5 adapters) located in */lib/firmware/cxgb4/*.

For example, if you want to filter traffic based on *ethtype* value in the packets for T6 adapters, replace the default filterMode with,

```
filterMode = fragmentation, mpshittype, macmatch, ethertype, protocol, port
```

The network driver needs to be reloaded next using the following command:

```
[root@host~]# rmmod cxgb4
[root@host~]# modprobe cxgb4 use_ddr_filters=1
```

3.2. Creating Filter Rules

Network driver (*cxgb4*) must be installed and loaded before setting the filter rule.

i. If you haven't done already, run the Unified Wire Installer with the *High Capacity Hash Filter* or *Unified Wire (Default)* configuration tuning option to install the drivers.
ii. Load the network driver with DDR filters support and bring up the Chelsio interface.

```
[root@host~]# modprobe cxgb4 use_ddr_filters=1
[root@host~]# ifconfig ethX up
```

iii. Now, create filter rules using cxgbtool.

```
[root@host~]# cxgbtool ethX filter <index> action [pass/drop/switch] fip
<source_ip> lip <destination_ip> fport <source_port> lport
<destination_port> proto <protocol> <hitcnts 1> <cap maskless>
```

Where,

ethX	:	Chelsio interface.
index	:	Filter index. For LE-TCAM filters, filter index should be 0-495 for T5 adapters and 0-559 for T6 adapters. In case of Hash/DDR filter, the index will be ignored and replaced by an automatically computed value, based on the hash (4-tuple). The index will be displayed after the filter rule is created successfully.
action	:	Ingress packet disposition.
pass	:	Ingress packets will be passed through set ingress queues.
switch	:	Ingress packets will be routed to an output port with optional header rewrite.
drop	:	Ingress packets will be dropped.
source_ip/port	:	Source IP/port of incoming packet.
destination_ip/port	:	Destination IP/port of incoming packet.
protocol	:	TCP by default. To change, specify the corresponding internet protocol number, e.g., use 17 for UDP.
hitcnts 1	:	To enable hit counts in cxgbtool filter show output.
cap maskless	:	This is mandatory for hash filter. If not provided, LE-TCAM filter will be created at the specified index.

Note

In case of Hash/DDR filters, ***source_ip****,* ***destination_ip****,* ***source_port*** *and* ***destination_port*** *are mandatory, since the filters don't support masks and hence, 4-tuple must always be supplied.* ***Proto*** *is also a mandatory parameter.*

3.2.1. Choosing filterMode and filterMask

As mentioned earlier filterMask tuples can be subset of or equal to filterMode tuples. Following are examples of how you can select filterMode and filterMask and create a Hash filter based on those values:

- **When all tuples from filterMode are enabled in filterMask**

i. Select a filterMode from supported filterMode table based on your requirement. E.g.,

```
filterMode = fragmentation, mpshittype, protocol, vlan, port, fcoe
```

ii. Select a filterMask so that it is a subset of or equal to filterMode based on application. E.g.;

```
filterMask = fragmentation, mpshittype, protocol, vlan, port, fcoe
```

It is mandatory to create a filter based on all the above tuples mentioned in filterMask. Otherwise, filter rule will not honour.

iii. Now, to create a hash filter based on the filterMode and filterMask values selected above:

```
[root@host~]# cxgbtool eth18 filter 100 action drop lip 120.10.10.100 fip
120.10.10.200 lport 5001 fport 51549 proto 6 frag 0 matchtype 0 ivlan 10
iport 0 fcoe 0 hitcnts 1 cap maskless
Hash-Filter Index = 303760
```

- **When all tuples from filterMode are not enabled in filterMask**

In case if you don't want to create filter rule based on any particular tuple from filterMode, don't select it in filterMask. For example, if you don't want to create a filter based on VLAN value, then don't select it from filterMode to filterMask.

i. Select a filterMode from supported filterMode table based on your requirement. E.g.,

```
filterMode = fragmentation, mpshittype, protocol, vlan, port, fcoe
```

ii. Select a filterMask so that it is a subset of or equal to filterMode based on application without VLAN tuple. E.g.;

```
filterMask = fragmentation, mpshittype, protocol, port, fcoe
```

Here, we have selected *fragmentation, mpshittype, protocol, port, fcoe* in filterMask so it is mandatory to create a filter based on only those tuples mentioned in filterMask. Otherwise, filter rule will not honour.

iii. Now, to create a hash filter based on the filterMode and filterMask values selected above:

```
[root@indus sw]# cxgbtool eth18 filter 100 action drop lip 120.10.10.100
fip 120.10.10.200 lport 5001 fport 51549 proto 6 frag 0 matchtype 0 iport
0 fcoe 0 hitcnts 1 cap maskless
Hash-Filter Index = 196568
```

3.2.2. Examples

- ***drop* action**

```
[root@host~]# cxgbtool ethX filter 496 action drop lip 102.1.1.1 fip
102.1.1.2 lport 12865 fport 20000 hitcnts 1 cap maskless iport 1 proto 17
Hash-Filter Index = 61722
```

The above filter rule will drop all UDP packets matching above 4 tuple coming on Chelsio port 1. Remaining packets will be sent to the host.

- ***pass* action**

```
[root@host~]# cxgbtool ethX filter 496 action pass lip 102.2.2.1 fip
102.2.2.2 lport 12865 fport 12000 hitcnts 1 cap maskless proto 6
Hash-Filter Index = 308184
```

The above filter rule will pass all TCP packets matching above 4 tuple. Remaining packets will be sent to the host.

- ***switch* action**

```
[root@host~]# cxgbtool ethX filter 496 action switch lip 102.3.3.1 fip
102.3.3.2 lport 5001 fport 16000 proto 6 iport 0 eport 1 hitcnts 1 cap
maskless
Hash-Filter Index = 489090
```

The above filter rule will switch all TCP packets matching above 4 tuple from Chelsio port 0 to Chelsio port 1. Remaining packets will be sent to the host.

Note *For more information on additional parameters, refer cxgbtool manual by running the `man cxgbtool` command.*

3.3. Listing Filter Rules

- To list the Hash/DDR filters set, run the following command:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<bus-id>/hash_filters
```

- To list the both LE-TCAM and Hash/DDR filters set, run the following command:

```
[root@host~]# cxgbtool ethX filter show
```

3.4. Removing Filter Rules

To remove a filter, run the following command with *cap maskless* parameter and corresponding filter rule index:

```
[root@host~]# cxgbtool ethX filter <index> <delete|clear> cap maskless
```

Note

- *Filter rule index can be determined by referring the "hash_filters" file located in /sys/kernel/debug/cxgb4/<bus-id>/.*
- *For more information on additional parameters, refer cxgbtool manual by runing the `man cxgbtool` command.*

3.5. Filter Priority

By default, Hash/DDR filter has priority over LE-TCAM filter. To override this, the LE-TCAM filter should be created with *prio* option. For example:

```
[root@host~]# cxgbtool ethx filter <index> action <pass/drop/switch> prio 1
```

Where *index* is a positive integer set as filter id. 0-495 for T5 adapters and 0-63 for T6 adapters.

3.6. Swap MAC Feature

Chelsio's T6/T5 Swap MAC feature swaps packet source MAC and destination MAC addresses. This is applicable only for switch filter rules. Here's an example:

```
[root@host~]# cxgbtool eth2 filter 100 action switch lip 102.2.2.1 fip
102.2.2.2 lport 5001 fport 14000 hitcnts 1 iport 1 eport 0 swapmac 1 proto
17 cap maskless
Hash-Filter Index = 21936
```

The above example will swap source and destination MAC addresses of UDP packets (matching above 4 tuple) received on adapter port 1 and then switch them to port 0.

3.7. Traffic Mirroring

On T5/T6 adapters, when using *Hash Filter* configuration tuning options, Network driver (cxgb4) parameter *enable_mirror* can be used to enable mirroring of traffic running on physical ports. The mirrored traffic will be received via Mirror PF/VF on Mirror Receive queues, which will then inject this traffic into network stack of Linux kernel.

3.7.1. Enabling Mirroring

To enable traffic mirroring, follow the steps mentioned below:

i. If not done already, install Unified Wire with *High Capacity Hash Filter* or *Unified Wire (Default)* configuration tuning option as mentioned in the Unified Wire chapter.
ii. Enable *vnic_id* match for filterMode in Hash filter config file, *t5-config.txt*, located in */lib/firmware/cxgb4/*

```
filterMode = fragmentation, mpshittype, protocol, vnic_id, port, fcoe
filterMask = port, protocol, vnic_id
```

iii. Unload network driver (*cxgb4*) and reload it with mirroring enabled.

```
[root@host~]# rmmod cxgb4
[root@host~]# modprobe cxgb4 enable_mirror=1 use_ddr_filters=1
```

iv. The traffic will now be mirrored and received via mirror PF/VF corresponding to each port.

3.7.2. Switch Filter with Mirroring

The following example explains the method to switch and mirror traffic simultaneously:

i. Obtain the PF and VF values of the incoming port from */sys/kernel/debug/cxgb4/<bus-id>/mps_tcam*
ii. Create the desired switch filter rule.

```
[root@host~]# cxgbtool ethX filter 100 fip 102.8.8.2 lip 102.8.8.1 fport
20000 lport 12865 proto 6 pf 4 vf 64 action switch iport 0 eport 1 cap
maskless
```

The hash filter rule switches TCP traffic matching the above 4-tuple received on port 0 to port 1. The traffic will be switched and simultaneously received on mirror queues and network stack of host as mirroring is enabled.

3.7.3. Filtered Traffic Mirroring

Once mirroring is enabled, all the traffic received on a physical port will be duplicated. The following example explains the method to filter out the redundant traffic and receive only specific traffic on mirror queues:

i. Obtain the mirror PF and VF values from *dmesg*. You should see a similar output:

```
[165299.356887] cxgb4 0000:02:00.4: Port 0 Traffic Mirror PF = 4; VF = 66
[165299.358004] cxgb4 0000:02:00.4: Port 1 Traffic Mirror PF = 4; VF = 67
```

ii. Create a DROP-ALL rule as below:

```
[root@host~]# cxgbtool ethX filter 255 pf 4 vf 66 action drop
```

Where, 255 is the last index of available TCAM filters. This will create a catch-all DROP filter for Mirror PF/VF of port 0. Similarly, create DROP filters for rest of Mirror PF/VF.

iii. Create specific filter rules to allow specific traffic to be received on mirror queues as below:

```
[root@host~]# cxgbtool ethX filter 101 lip 102.8.8.1 fip 102.8.8.2 lport
12865 fport 20000 pf 4 vf 66 action pass
```

Now, the above specific traffic (from *102.8.8.2,20000* to *102.8.8.1,12865*) will be received in mirror receive queues and network stack of host.

3.8. Packet Tracing and Hit Counters

For T5/T6 LE-TCAM and T6 Hash/DDR filters, *hit counters* will work simply by adding *hitcnts 1* parameter to the filter rule. However, for T5 Hash/DDR filters, you will have to make use of tracing feature and RSS queues. Here's a step-by-step guide to enable packet tracing and *hit counters* for T5 Hash/DDR filter rules:

i. Load network driver with the following parameters:

```
[root@host~]# modprobe cxgb4 use_ddr_filters=1 enable_traceq=1
```

ii. Configure the required filter rules.
iii. Enable tracing on T5 adapter.

```
[root@host~]# cxgbtool ethX reg 0x09800=0x13
```

iv. Setup a trace filter.

```
[root@host~]# echo tx1 snaplen=40 > /sys/kernel/debug/cxgb4/<bus_id>/trace0
```

Here, *snaplen* is the length in bytes to be captured.

Note *Use "snaplen=60" in case of IPV6.*

The above step will trace all the packets transmitting from port1(tx1) to trace filter 0.

v. Configure RSS Queue to send trace packets. Determine the RspQ ID of the queues by looking at *Trace* QType in */sys/kernel/debug/cxgb4/<bus-id>/sge_qinfo* file.

```
[root@host~]# cxgbtool ethX reg 0x0a00c=<Trace Queue0-RspQ ID>
```

Now the traced packets can be seen in tcpdump and the hit counters will also increment.

- **Multi-tracing**

To enable packet capture or *hit counters* for multiple chelsio ports in Tx/Rx direction enable Multi-tracing. Using this we can configure 4 different RSS Queues separately corresponding to 4 trace-filters.

i. Enable Tracing as well as MultiRSSFilter.

```
[root@host~]# cxgbtool ethX reg 0x09800=0x33
```

ii. Setup a trace filter.

```
[root@host~]# echo tx0 snaplen=40 > /sys/kernel/debug/cxgb4/<bus_id>/trace0
```

iii. Configure the RSS Queue corresponding to trace0 filter configured above. Determine the *RspQ ID* of the queues by looking at *Trace* QType in */sys/kernel/debug/cxgb4/<bus-id>/sge_qinfo* file.

```
[root@host~]# cxgbtool ethX reg 0x09808=<Trace-Queue0-RspQ ID>
```

iv. Similarly for other direction and for multiple ports run the follow commands:

```
[root@host~]# echo rx0 snaplen=40 > /sys/kernel/debug/cxgb4/<bus_id>/trace1
[root@host~]# echo tx1 snaplen=40 > /sys/kernel/debug/cxgb4/<bus_id>/trace2
[root@host~]# echo rx1 snaplen=40 > /sys/kernel/debug/cxgb4/<bus_id>/trace3
[root@host~]# cxgbtool ethX reg 0x09ff4=<Trace-Queue1-RspQ ID>
[root@host~]# cxgbtool ethX reg 0x09ffc=<Trace-Queue2-RspQ ID>
[root@host~]# cxgbtool ethX reg 0x0a004=<Trace-Queue3-RspQ ID>
```

Note *Use “snaplen=60” in case of IPV6.*

4. NAT Filtering

T5/T6 adapters support offloading of stateless/static NAT functionality i.e. translating source/ destination L3 IP addresses, and source/destination L4 port numbers. This feature is supported with both LE-TCAM and Hash filters.

Note *This feature is only supported with filter action switch.*

Syntax:

```
[root@host~]# cxgbtool ethX filter <index> action switch fip <source_ip> lip
<destination_ip> fport <source_port> lport <destination_port> nat <mode>
nat_fip <new_source_ip> nat_lip <new_destination_ip> nat_fport
<new_source_port> nat_lport <new_destination_port>
```

Where,

ethX	: Chelsio interface.
source_ip/port	: Source IP/port of incoming packet.
destination_ip/port	: Destination IP/port of incoming packet.
new_source_ip/port	: Source IP/port to be translated to.
new_destination_ip/port	: Destination IP/port to be translated to.
Mode	: Combination of IP/port to be translated. *all* will translate all 4-tuple fields. To see other modes, refer cxgbtool manual page.

Examples:

- Hash filter to translate all four tuples, viz. source IP, destination IP, source port and destination port to new values.

```
[root@ ~]# cxgbtool eth0 filter 101 action switch iport 0 eport 1 fip 102.10.10.100 lip 102.10.10.200 fpo
rt 50000 lport 60000 proto 17 nat all nat_fip 192.168.10.100 nat_lip 192.168.10.200 nat_fport 60000 nat_lport
61000 hitcnts 1 cap maskless
Hash-Filter Index = 232776
```

- Hash filter to translate source IP and source port to new values.

```
[root@ ~]# cxgbtool eth0 filter 101 action switch iport 0 eport 1 fip 102.10.10.100 lip 102.10.10.200 fpo
rt 50000 lport 60000 proto 17 nat sip-sp nat_fip 192.168.10.100  nat_fport 61000 hitcnts 1 cap maskless
Hash-Filter Index = 232776
```

- LE-TCAM filter to translate destination IP and destination port to new values.

```
[root@ ~]# cxgbtool eth0 filter 101 action switch iport 0 eport 1 lip 102.10.10.200 lport 60000 nat dip-d
p nat_lip 192.168.10.200 nat_lport 61000 hitcnts 1
```

XXIV. OVS Kernel Datapath Offload

1. Introduction

Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols.

Chelsio's T6/T5 Unified Wire solution can offload OVS datapath flow match entries and action processing onto Chelsio adapter for hardware acceleration of OVS datapath flow processing.

Chelsio 1/10/25/40/50/100Gb Ethernet controllers and adapters are capable of offloading OpenFlow and non-OpenFlow network traffic simultaneously, including tunnel handling (e.g. VXLAN / IPsec), NAT, IP stack (ARP, route lookup, frag tracking, fragment / defragment) and other kernel functionalities. A high performance, scalable network I/O is delivered by leveraging built in eSwitch and traffic manager capabilities. In addition, features like traffic classifier, load balancer and firewall are supported at port level by all Chelsio adapters.

1.1. Hardware Requirements

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T62100-SO-CR*
- T61100-OCP*
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP*
- T6225-SO-CR*
- T580-CR
- T580-LP-CR
- T580-SO-CR*
- T580-OCP-SO*
- T540-CR
- T540-LP-CR
- T540-SO-CR*
- T540-BT
- T520-CR
- T520-LL-CR
- T520-SO-CR*
- T520-OCP-SO*
- T520-BT

** Hash Filter (exact-match) flows not supported*

1.2. Software Requirements

Currently the OVS Kernel Datapath Offload driver is available for the following versions:

- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64

Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Installation

2.1. Pre-requisites

GCC 4.6+, Python 2.7+, Python-six, Autoconf 2.63+, Automake 1.10+, libtool 2.4+ packages should be installed. For the complete list of software required visit *http://docs.openvswitch.org/en/latest/intro/install/general/*

2.2. Installation

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install OVS driver.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Configuration and Fine Tuning

Supported Fields

The following match fields are supported for offload:

- Input Port
- L2 Ethernet Type
- L3 IP Protocol Type
- L3 IPv4 address
- L3 IPv6 address
- L3 IPv4 TOS
- L3 IP Fragmentation
- L4 Ports (tcp/udp src-port, dst-port)
- Tunnel/Encapsulation VNI (only on T6)

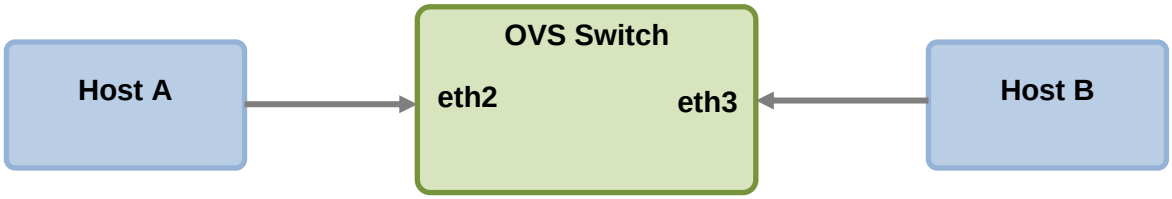
Supported Actions

The following actions are supported for offload:

- Drop
- Switch (output to a port)
- L2 Rewrite: src-mac, dst-mac
- VLAN Rewrite: push, pop, modify
- L3 Rewrite: ip-src, ip-dst (IPv4 and IPv6)
- L4 Rewrite: src-port, dst-port (TCP/UDP)

3.1. Configuring OVS Machine

The following example explains the method to configure an OVS machine:



**eth2* and *eth3* are Chelsio interfaces.

i. Ensure that Unified Wire is installed with ***High Capacity Hash Filter*** configuration tuning option.
ii. Update the *filterMode* and *filterMask* in the config file in */lib/firmware/cxgb4/*. Select a Filter Mode combination with fragmentation, ethertype, protocol and port from the supported list. Use *t6-config.txt* for T6 adapters and *t5-config.txt* for T5 adapters.

```
filterMode = fragmentation, mpshittype, ethertype, protocol, tos, port, fcoe
filterMask = fragmentation, ethertype, protocol, port
```

Note *FilterMask tuples can be subset of or equal to filterMode tuples.*

iii. Load NIC (cxgb4) driver with hash-filter support.

```
[root@host~]# modprobe cxgb4 use_ddr_filters=1
```

iv. Bring up the Chelsio interfaces in promiscuous mode.

```
[root@host~]# ifconfig eth2 promisc up
[root@host~]# ifconfig eth3 promisc up
```

v. Load Open vSwitch module.

```
[root@host~]# modprobe openvswitch
```

vi. Configure OVS.

```
[root@host~]# ovs-appctl exit
[root@host~]# pkill -9 ovs
[root@host~]# rm -rf /usr/local/etc/ovs-vswitchd.conf
[root@host~]# rm -rf /usr/local/var/run/openvswitch/db.sock
[root@host~]# rm -rf /usr/local/etc/openvswitch/conf.db
[root@host~]# touch /usr/local/etc/ovs-vswitchd.conf
[root@host~]# ovsdb-tool create /usr/local/etc/openvswitch/conf.db
<uwire_package>/src/openvswitch-x.x.x/vswitchd/vswitch.ovsschema
[root@host~]# ovsdb-server /usr/local/etc/openvswitch/conf.db --
remote=punix:/usr/local/var/run/openvswitch/db.sock --
remote=db:Open_vSwitch,Open_vSwitch,manager_options --bootstrap-ca-
cert=db:Open_vSwitch,SSL,ca_cert --pidfile --detach --log-file
[root@host~]# ovs-vsctl --no-wait init
[root@host~]# export DB_SOCK=/usr/local/var/run/openvswitch/db.sock
[root@host~]# ovs-vswitchd --pidfile --detach
```

vii. Create an OVS bridge and add Chelsio interfaces to it.

```
[root@host~]# ovs-vsctl add-br br0
[root@host~]# sleep 2
[root@host~]# ifconfig br0 up
[root@host~]# ovs-vsctl add-port br0 eth2
[root@host~]# sleep 5
[root@host~]# ovs-vsctl add-port br0 eth3
[root@host~]# sleep 5
[root@host~]# ovs-vsctl show
```

Note *Ports on OVS bridge must be added in the same order as the adapter, since there's no mapping between OVS and physical ports.*

viii. Now ping from Host A to Host B to verify that OVS is configured successfully.

ix. Stop the ping traffic and delete all the flows on switch.

```
[root@host~]# ovs-ofctl del-flows br0
```

3.2. Creating OVS flows

It is mandatory to specify L2 Ethernet Type (dl_type) to offload OVS flows. There are two types of flows:

- **exact-match**: Protocol and 4-tuple are mandatory to create an exact-match flow. ~0.5 million exact-match flows can be offloaded.
- **wild-card**: If any of 4-tuple and protocol are absent, wild-card flow is created. 496 wild-card flows can be offloaded.

Note *T5/T6 SO adapters do not support exact-match flows. Up to 494 wild-card flows are supported.*

3.2.1. Examples

3.2.1.1. Generic Flows

Below are few example OVS Flows with the following *filterMode* and *filterMask* combination:

```
filterMode = fragmentation, mpshittype, ethertype, protocol, tos, port, fcoe
filterMask = fragmentation, ethertype, protocol, port
```

- Wild-card flow to drop incoming packets on first port.

```
[root@host~]# ovs-ofctl add-flow br0 in_port=1,dl_type=0x800,action=drop
```

- Wild-card flow to switch ARP packets (L2 EtherType=0x0806) on 1st port to 2nd port.

```
[root@host~]# ovs-ofctl add-flow br0
in_port=1,dl_type=0x0806,action=output:2
```

- Wild-card flow to switch TCP packets (L3 proto=6) on 1st port to 2nd port by rewriting source and destination MAC addresses.

```
[root@host~]# ovs-ofctl add-flow br0 in_port=1,dl_type=0x800,
nw_proto=6,action=mod_dl_src:00:07:43:28:E4:50,
mod_dl_dst:00:07:43:44:64:50,output:2
```

- Exact-match flow to drop matching 4-tuple traffic.

```
[root@host~]# ovs-ofctl add-flow br0
in_port=1,dl_type=0x800,nw_proto=6,nw_src=10.1.1.66,tp_src=11000,nw_dst=10.1
.1.58,tp_dst=11000,action=drop
```

- Exact-match flow to match 4-tuple IPv4 traffic and do NAT rewrite.

```
[root@host~]# ovs-ofctl add-flow br0
dl_type=0x800,nw_proto=6,nw_src=10.1.1.66,tp_src=11000,nw_dst=10.1.1.58,tp_d
st=21000,action=mod_nw_src=10.2.2.66,mod_tp_src=11005,mod_nw_dst:10.2.2.62,m
od_tp_dst:12345,output:2
```

- Exact-match flow to match 4-tuple IPv6 traffic and do NAT rewrite.

```
[root@host~]# ovs-ofctl add-flow br0
in_port=1,dl_type=0x86dd,nw_proto=6,ipv6_src=2000::66,tp_src=11000,ipv6_dst=
2000::58,tp_dst=11000,action=set_field:2001::66-
\>ipv6_src,mod_tp_src=15000,output:2
```

- Wild-card flow to drop fragmented packets.

```
[root@host~]# ovs-ofctl add-flow br0 dl_type=0x800,ip_frag=yes,action=drop
```

- If a wild-card and exact match flow both exist for the same traffic pattern, the flow that is created first will take priority. In the below example, the wild-card flow will take priority as it was created first.

```
[root@host~]# ovs-ofctl add-flow br0
dl_type=0x800,nw_src=10.1.1.58,nw_dst=10.1.1.66,tp_src=15000,tp_dst=15000,ac
tion=output:1
[root@host~]# ovs-ofctl add-flow br0
dl_type=0x800,nw_proto=6,nw_src=10.1.1.58,nw_dst=10.1.1.66,tp_src=15000,tp_d
st=15000,action=output:1
```

3.2.1.2. VLAN Flows

Note *Only wild-card flows are currently supported with VLAN matches.*

Below are few example VLAN flows with the following *FilterMode* and *FiletrMask* combination.

```
filterMode = fragmentation,mpshittype,ethertype,vlan,port
filterMask = ethertype,vlan,port
```

Reload *cxgb4* driver after updating *filterMode* and *filterMask*.

- Strip VLAN tag and switch traffic.

```
[root@host~]# ovs-ofctl add-flow br0
in_port=1,dl_type=0x800,action=strip_vlan,output:2
```

- Insert VLAN tag 100 and switch traffic.

```
[root@host~]# ovs-ofctl -O OpenFlow11 add-flow br0
in_port=1,dl_type=0x800,action=push_vlan:0x8100,set_field:100-
\>vlan_vid,output:2
```

- Modify VLAN tag 30 to 50 and switch traffic.

```
[root@host~]# ovs-ofctl -O OpenFlow11 add-flow br0
in_port=1,dl_type=0x800,vlan_vid=30,action=mod_vlan_vid=50,output:2
```

If *vlan_vid* is not specieifed, the *mod_vlan_vid* tag will be added with a priority of 0.

- Modify VLAN priority and switch traffic.

```
[root@host~]# ovs-ofctl -O OpenFlow11 add-flow br0
in_port=1,dl_type=0x800,vlan_pcp=4,action=mod_vlan_pcp=3,output:2
```

3.2.1.3. VXLAN Flows

The following steps describe the method to configure VXLAN using OVS with single port connected on Server and Client machines.

Note

- *Only wild-card flows are currently supported with VXLAN matches.*
- *VxLAN VNI rewrite is not supported.*
- *Supported only on kernels above 4.9*

Server

i. Update the firmware configuration file, *t6-config.txt*, located at */lib/firmware/cxgb4/*

```
filterMode = fragmentation, mpshittype, ethertype, vnic_id, port
filterMask = ethertype, vnic_id, port
vnicMode = encapsulation
```

ii. Load NIC (cxgb4) driver with hash-filter support.

```
[root@host~]# modprobe cxgb4 use_ddr_filters=1
```

iii. Bring up Chelsio interfaces in promiscous mode.

```
[root@host~]# ifconfig ethX <chelsio_port0_ip_address> promisc up
[root@host~]# ifconfig ethY promisc up
```

iv. Load Open vSwitch module.

```
[root@host~]# modprobe openvswitch
```

v. Configure OVS.

```
[root@host~]# ovs-appctl exit
[root@host~]# pkill -9 ovs
[root@host~]# rm -rf /usr/local/etc/ovs-vswitchd.conf
[root@host~]# rm -rf /usr/local/var/run/openvswitch/db.sock
[root@host~]# rm -rf /usr/local/etc/openvswitch/conf.db
[root@host~]# touch /usr/local/etc/ovs-vswitchd.conf
[root@host~]# ovsdb-tool create /usr/local/etc/openvswitch/conf.db
<uwire_package>/src/openvswitch-x.x.x/vswitchd/vswitch.ovsschema
[root@host~]# ovsdb-server /usr/local/etc/openvswitch/conf.db --
remote=punix:/usr/local/var/run/openvswitch/db.sock --
remote=db:Open_vSwitch,Open_vSwitch,manager_options --bootstrap-ca-
cert=db:Open_vSwitch,SSL,ca_cert --pidfile --detach --log-file
[root@host~]# ovs-vsctl --no-wait init
[root@host~]# export DB_SOCK=/usr/local/var/run/openvswitch/db.sock
[root@host~]# ovs-vswitchd --pidfile -detach
[root@host~]# ovs-vsctl add-br br0
[root@host~]# ifconfig br0 <server_ip>/24 up
[root@host~]# ovs-vsctl add-port br0 <chelsio_port0_name> -- set interface
<chelsio_port0_name> type=vxlan options:remote_ip=<peer_chelsio_port0_ip>
options:local_ip=<local_chelsio_port0_ip> options:key=flow
```

vi. Disable GRO on the chelsio adapter, bridge and VXLAN interfaces.

```
[root@host~]# ethtool -K <chelsio_port0_name> gro off
[root@host~]# ethtool -K <chelsio_port1_name> gro off
[root@host~]# ethtool -K br0 gro off
[root@host~]# ethtool -K vxlan_sys_* gro off
```

vii. Set a rule to match packets with VNI=42 and drop.

```
[root@host~]# ovs-ofctl add-flow br0 in_port=1,tun_id=0x2a,action=drop
```

Client

i. Follow steps (i)-(vi) described in the previous section **Server**.
ii. Set a rule to set VNI to 42 and send traffic.

```
[root@host~]# ovs-ofctl add-flow br0
in_port=LOCAL,action=set_tunnel:0x2a,output:1
```

3.3. Verifying OVS Flow Dump

OVS flow dump can be verified using:

```
[root@host~]# ovs-ofctl dump-flows br0
```

Run traffic between hosts which matches the flow and verify if the *n_packet* counter is incrementing.

To check if the OVF Flows were offloaded, run the below command:

```
[root@host~]# cxgbtool ethX filter show
```

Wild-card flows will be shown as *LE-TCAM Filters* and Exact-match flows will be shown as *Hash Filters. Hits* and *Hit-Bytes* will increment for the corresponding filters.

3.4. Setting up ODL with OVS

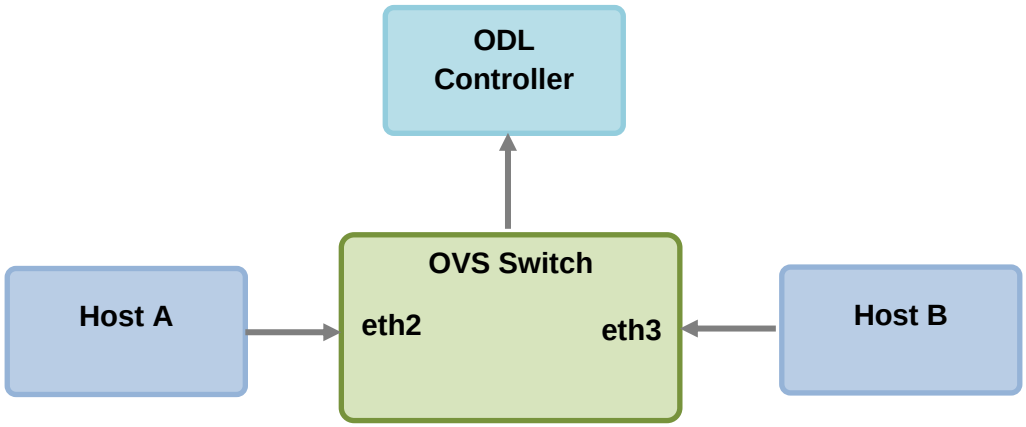
The following example explains the method to set up OpenDaylight (ODL) using OVS:

On the ODL controller setup,

i. Download latest Java Development Kit.
ii. Untar the tar file.

iii. Create an entry in *.bashrc* which points to the extracted folder.

```
export JAVA_HOME=<path>/jdk1.8.0_92
export PATH=$PATH:$JAVA_HOME
```



**eth2* and *eth3* are Chelsio interfaces.

iv. Logout & log back in.
v. Download ODL controller pre-built zip package.
vi. Unzip the package and change your working directory to *opendaylight.*
vii. Run the script *run.sh* and wait for ~3 minutes for the controller to be setup.
viii. Open a web browser and enter the address *http://localhost:8080*
ix. Login with *admin* keyword for both username and password.
x. On the OVS machine, add the bridge to the controller and disable in-band.

```
[root@host ~]# ovs-vsctl set-controller br0 tcp:<ODL Controller IP>:6633
[root@host ~]# ovs-vsctl set bridge br0 other-config:disable-in-band=true
```

xi. Refresh the webpage on the ODL controller and you should see the OVS details.
xii. Goto **Flows** tab, add and install a flow.
xiii. Verify the flow dump on the OVS machine.

```
[root@host ~]# ovs-ofctl dump-flows br0
```

Run traffic between hosts which matches the flow and verify if the *n_packet* counter is incrementing.

4. Software/Driver Uninstallation

i. Change your working directory to Chelsio Unified Wire directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Uninstall OVS driver.

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

XXV. Mesh Topology

1. Introduction

Chelsio's fifth/sixth generation (T5/T6), high performance 10/25/40/50/100GbE adapters enable incremental, non-disruptive server installs, and support the ability to work without requiring any discrete external network switch, delivering a brownfield strategy to enable high performance, low cost, scalable deployments. Major benefits include cost savings on switches at higher speeds with each deployment. Mesh topology involves connecting each node to every other node.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T61100-OCP*
- T62100-CR
- T62100-LP-CR
- T62100-SO-CR*
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP ^
- T6225-SO-CR ^
- T580-OCP-SO*
- T580-CR
- T580-LP-CR
- T580-SO-CR*
- T540-CR
- T540-LP-CR
- T540-SO-CR
- T520-CR
- T520-LL-CR
- T520-SO-CR*
- T520-OCP-SO*
- T520-BT
- T540-BT

* *Only NIC driver supported.*
^ *Memory-free; 256 IPv4/128 IPv6 offload connections supported.*

1.2. Software Requirements

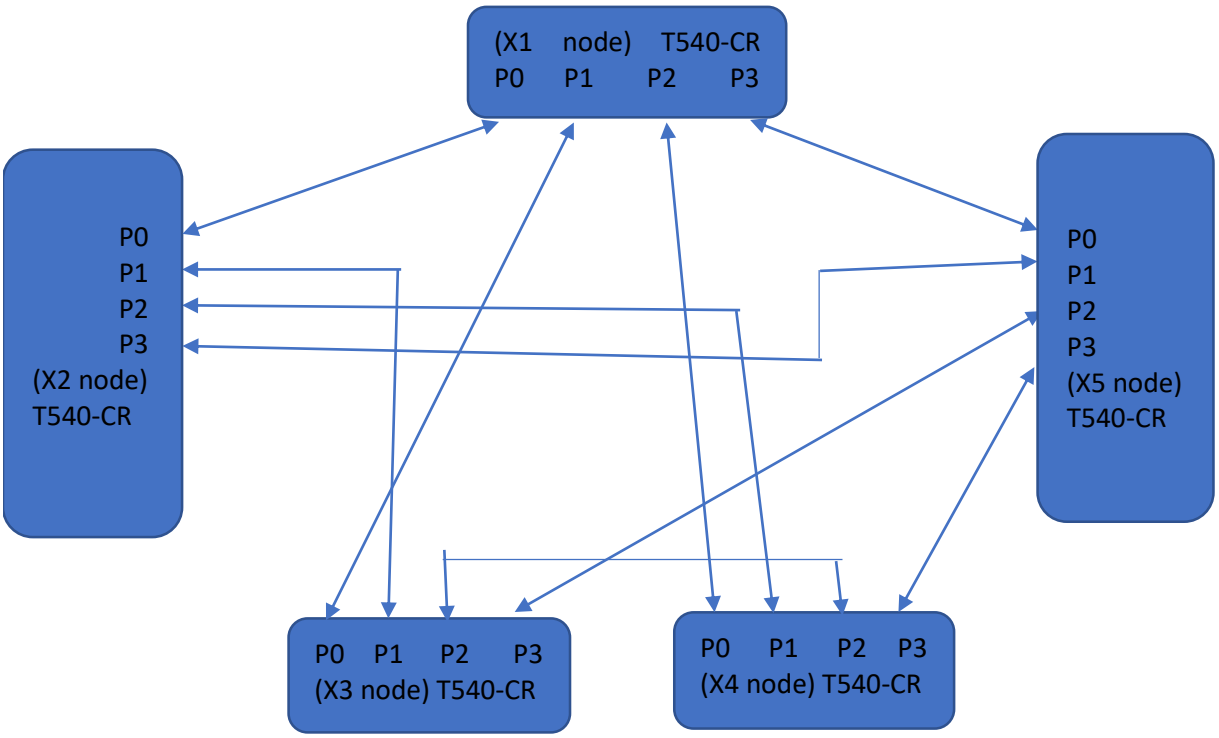
Currently the Mesh topology is available for the following Linux version(s):

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

Other versions have not been tested and are not guaranteed to work.

1.3. Mesh topology

Each node should be connected to other node. Supported configs using this approach: N ports per node, N+1 node cluster. The below is a 5-node Mesh using 4-port Chelsio adapters. NIC ports on each server connected to each other (1<->2, 1<->3, 1<->4, 1<->5, 2<->3, 2<->4, 2<->5, 3<->4, 3<->5, 4<->5).



2. Software/Driver Installation

Install Unified Wire on all the machines in the mesh topology.

i. Change your current working directory to Chelsio Unified Wire package directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
```

ii. Install the drivers, tools and libraries.

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

Note *For more installation options, please run* `make help` *or* `install.py -h`

iii. Reboot your machine for changes to take effect.

```
[root@host~]# reboot
```

3. Software/Driver Configuration and Fine-tuning

Configure all the machines in the mesh topology using the below steps:

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

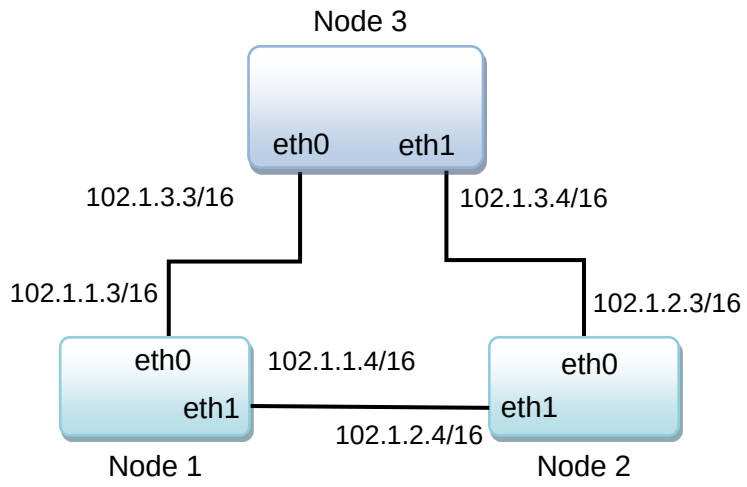
i. Load network driver (*cxgb4*).

```
[root@host~]# modprobe cxgb4
```

ii. Configure interfaces with required IPs and networking as mentioned in https://access.redhat.com/solutions/30564 article.

You should be able to run traffic between the nodes. To run different protocol traffic, please refer their respective sections for protocol configuration.

Example:
3 nodes are connected to each other in mesh topology with the following IP addresses.



If node3 wants to communicate to node1,

```
[root@host~]# ping -I 102.1.3.3 102.1.1.3
```

If node3 wants to communicate to node2,

```
[root@host~]# ping -I 102.1.3.4 102.1.2.3
```

XXVI. Traffic Management

1. Introduction

Traffic Management capabilities built-in to Chelsio adapters can shape transmit data traffic through the use of sophisticated queuing and scheduling algorithms built-in to the ASIC hardware which provides fine-grained software control over latency and bandwidth parameters such as packet rate and byte rate. These features can be used in a variety of data center application environments to solve traffic management problems.

Traffic Management features in Chelsio‘s adapters allow the user to control three main things:

- Guarantee low latency in the presence of other traffic
- Control max bandwidth that a connection or a flow (a group of connections) can use
- Allocate available bandwidth to several connection or flows based on desired levels of performance

Once the offload transmit traffic shaping classes have been configured, individual offloaded connections (flows) may be assigned to a traffic shaping class to manage the flows as per the class configuration. The mechanism to accomplish this "flow to class" mapping assignment is the Connection Offload Policy (COP) configuration system.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the Chelsio adapters that are supported:

- T62100-CR
- T62100-LP-CR
- T62100-SO-CR*
- T61100-OCP*
- T6425-CR
- T6225-CR
- T6225-LL-CR
- T6225-OCP*
- T6225-SO-CR*
- T580-CR
- T580-LP-CR
- T580-SO-CR*
- T580-OCP-SO*
- T540-CR
- T540-LP-CR
- T540-SO-CR*
- T540-BT
- T520-CR
- T520-LL-CR

- T520-SO-CR*
- T520-OCP-SO*
- T520-BT

* *Only NIC driver supported*.

1.2. Software Requirements

1.2.1. Linux Requirements

Currently the Traffic Management feature is available for the following versions:

- RHEL 8.4, 4.18.0-305.el8.x86_64
- RHEL 8.3, 4.18.0-240.el8.x86_64
- RHEL 7.9, 3.10.0-1160.el7.x86_64
- RHEL 7.8, 3.10.0-1127.el7.x86_64
- RHEL 7.6, 3.10.0-957.el7.ppc64le (POWER8 LE)
- RHEL 7.6, 4.14.0-115.el7a.aarch64 (ARM64)
- RHEL 7.5, 3.10.0-862.el7.ppc64le (POWER8 LE)
- RHEL 7.5, 4.14.0-49.el7a.aarch64 (ARM64)
- RHEL 6.10, 2.6.32-754.el6.x86_64
- Ubuntu 20.04.2, 5.4.0-65-generic
- Ubuntu 18.04.5, 4.15.0-135-generic
- Kernel.org linux-5.10.61
- Kernel.org 5.4.143

  Other kernel versions have not been tested and are not guaranteed to work.

2. Software/Driver Loading

Important *Please ensure that all inbox drivers are unloaded before proceeding with unified wire drivers.*

```
[root@host~]# rmmod csiostor cxgb4i cxgbit iw_cxgb4 chcr cxgb4vf cxgb4
libcxgbi libcxgb
```

Traffic Management can be performed on non-offloaded connections as well as on offloaded connections.

The drivers must be loaded by the root user. Any attempt to load the drivers as a regular user will fail.

Run the following commands to load the TOE driver:

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe t4_tom
```

3. Software/Driver Configuration and Fine-tuning

3.1. Traffic Management Rules

Traffic Management supports the following types of scheduler hierarchy levels which can be configured using the *cxgbtool* utility:

i. Class Rate Limiting
ii. Class Weighted Round Robin
iii. Channel Rate Limiting

3.1.1. Class Rate Limiting

This scheduler hierarchy level can be used to rate limit individual traffic classes or individual connections (flow) in a traffic class. Configure it using the below command.

```
[root@host~]# cxgbtool <ethX> sched-class params type packet level cl-rl
mode <scheduler-mode> rate-unit <scheduler-rate-unit> rate-mode <scheduler-
rate-mode> channel <Channel No.> class <scheduler-class-index> max-rate
<maximum-rate> pkt-size <Packet size>
```

Here,

ethX	:	Chelsio interface
scheduler-mode	:	specifies whether the rule is configured for individual traffic classes or individual connections (flow) in a traffic class. Possible values include *flow* or *class*.
scheduler-rate-unit	:	Specifies whether the rule is configured for bit-rate or packet rate. Possible values include *bits* or *pkts.*
scheduler-rate-mode	:	Specifies whether the rule is configured to support a percent of the channel rate or an effective rate. Possible values include *relative* or *absolute.*
Channel No.	:	Port on which data is flowing (0-3).
scheduler-class-index	:	TCP traffic class index; 0-14 for T4/T5 and 0-30 for T6 adapters.
maximum-rate	:	Bit rate (Kbps) for this TCP stream. The lower limit is 10 Kbps.
Packet size	:	TCP mss size in bytes; for example - for an MTU of 1500, use a packet size of 1460.

3.1.2. Class Weighted Round Robin

Incoming traffic flows from various applications can be prioritized and provisioned using a weighted round-robin scheduling algorithm. Configure it using the below command.

```
[root@host~]# cxgbtool <ethX> sched-class params type packet level cl-wrr
channel <Channel No.> class <scheduler-class-index> weight <Y>
```

Here,

ethX	:	Chelsio interface.
Channel No.	:	Port on which data is flowing (0-3).
scheduler-class-index	:	TCP traffic class index; 0-14 for T4/T5 and 0-30 for T6 adapters.
weight	:	Weight to be used for a weighted-round-robin scheduling hierarchy. Possible values include 1 to 99.

3.1.3. Channel Rate Limiting

This scheduler hierarchy level can be used to rate limit individual channels. Atleast one class should be specified while configuring Channel Rate Limiting using the below command.

```
[root@host~]# cxgbtool <ethX> sched-class params type packet level ch-rl
rate-unit <scheduler-rate-unit> rate-mode <scheduler-rate-mode> channel
<Channel No.> class <scheduler-class-index> max-rate <maximum-rate>
```

Here,

ethX	:	Chelsio interface.
scheduler-rate-unit	:	Specifies whether the traffic management rule is configured for bit-rate or packet-rate. Possible values include *bits* or *pkts.*
scheduler-rate-mode	:	Specifies whether the traffic management rule is configured to support a percent of the channel rate or an effective rate. Possible values include *relative* or *absolute.*
Channel No.	:	Port on which data is flowing (0-3).
scheduler-class-index	:	TCP traffic class index; 0-14 for T4/T5 and 0-30 for T6 adapters.
maximum-rate	:	Bit rate (Kbps) for this TCP stream. The lower limit is 1 Gbps.

3.1.4. Listing TM parameters

To view the paramters of a class or a channel,

```
[root@host~]# cxgbtool <ethX> sched-class show channel <Channel No.> class
<scheduler-class-index>
```

Channel No.	:	Port on which data is flowing (0-3).
scheduler-class-index	:	TCP traffic class index; 0-14 for T4/T5 adapters and 0-30 for T6 adapters.

3.2. Configuring Traffic Management

3.2.1. For Non-offloaded connections

Traffic Management of non-offloaded connections is a 2-step process. In the first step bind connections to indicated NIC TX queue using *tc* utility from *iproute2-3.9.0* package. In the second step bind the indicated NIC TX queue to the specified TCP Scheduler class using the *cxgbtool* utility.

i. Load the network driver and bring up the interface.

```
[root@host~]# modprobe cxgb4
[root@host~]# ifconfig ethX up
```

ii. Bind connections to queues.

```
[root@host~]# tc qdisc add dev ethX root handle 1: multiq
[root@host~]# tc filter add dev ethX parent 1: protocol all prio 1 u32 match
ip dst <IP address of destination> action skbedit queue_mapping <queue>
```

Note *For additional binding options, run [root@host~]# man tc*

iii. Now, bind the NIC TX queue with traffic class.

```
[root@host~]# cxgbtool ethX sched-queue <queue> <class>
```

Here,

ethX : Chelsio interface
queue : NIC TX queue
class : Class index; 0-14 for T4/T5 adapters and 0-30 for T6 adapters.

Note *If the TX queue is all, * or any negative value, the binding will apply to all of the TX queues associated with the interface. If the class is unbind, clear or any negative value, the TX queue(s) will be unbound from any current TX Scheduler Class binding.*

Important *Flow mode is not supported for Non-offloaded connections.*

3.2.2. For Offloaded connections

Traffic Management of offloaded connections can be configured either by applying *COP* policies that associate offloaded connections to classes or by modifying the application.

Both the methods have been described below:

- **Applying *COP* policy**

i. Load the TOE driver and bring up the interface.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig ethX up
```

ii. Create a new policy file (say *new_policy_file*) and add the following line to associate connections with the given scheduling class. Class can have values ranging from 0-14 for T4/T5 adapters and 0-30 for T6 adapters.

Example:
src host 102.1.1.1 => offload class 0

The above example will associate all connections originating from IP address 102.1.1.1 with scheduling class 0.

Note *If no specified rule matches a connection, a default setting will be used which disables offload for that connection. That is, there will always be a final implicit rule following all the rules in the input rule set of:*

all => !offload

iii. Compile the policy file using *COP.*

```
[root@host~]# cop -d –o <output_policy_file> <new_policy_file>
```

iv. Apply the *COP* policy.

```
[root@host~]# cxgbtool ethX policy <output_policy_file>
```

Note *The policy applied using cxgbtool is not persistent and should be applied every time drivers are reloaded or the machine is rebooted.*

The applied cop policies can be read using,

```
[root@host~]# cat /proc/net/offload/toeX/read-cop
```

- **Modifying the application**

The application can also be modified to associate connections to scheduling classes. Follow the steps mentioned below:

i. Determine the TCP socket file descriptor in the application through which data is sent.
ii. Declare and initialize a variable in the application.

```
int cl=1;
```

Here,

cl is the TCP traffic class (scheduler-class-index) that the user wishes to assign the data stream to. This value needs to be in the range of 0-14 for T4/T5 adapters and 0-30 for T6 adapters.

The application will function as per the parameters set for that traffic class.

iii. Add socket option definitions.

In order to use *setsockopt()* to set the options to the TCP socket, the following two definitions need to be made:

- SOL_SCHEDCLASS used for setting TCP traffic class, which has the value 290.
- IPPROTO_TCP used for setting the type of IP Protocol.

```
# define SOL_SCHEDCLASS 290
# define IPPROTO_TCP 6
```

iv. Use the *setsockopt()* function to set socket options.

The *setsockopt()* call must be mentioned after the connect() call.

```
//Get the TCP socket descriptor variable
setsockopt (sockfd , IPPROTO_TCP, SOL_SCHEDCLASS, &cl, sizeof(cl));
```

Here,

sockfd : The file descriptor of the TCP socket.
&cl : Pointer to the class variables.
sizeof(cl) : The size of the variable.

v. Now, compile the application.

4. Usage

4.1. Non-Offloaded Connections

The following example demonstrates the method to rate limit all TCP connections on class 0 to a rate of 300 Mbps for Non-offload connections:

i. Load the network driver and bring up the interface.

```
[root@host~]# modprobe cxgb4
[root@host~]# ifconfig eth0 up
```

ii. Bind connections with destination IP address 192.168.5.3 to NIC TX queue 3.

```
[root@host~]# tc qdisc add dev eth0 root handle 1: multiq
[root@host~]# tc filter add dev eth0 parent 1: protocol all prio 1 u32 match
ip dst 192.168.5.3 action skbedit queue_mapping 3
```

iii. Bind the NIC TX queue to class 0.

```
[root@host~]# cxgbtool eth0 sched-queue 3 0
```

iv. Set the appropriate rule for class 0.

```
[root@host~]# cxgbtool eth0 sched-class params type packet level cl-rl
mode class rate-unit bits rate-mode absolute channel 0 class 0 max-rate
300000 pkt-size 1460
```

Important *Flow mode is not supported for Non-offloaded connections.*

4.2. Offloaded Connections

The following example demonstrates the method to rate limit all TCP connections on class 0 to a rate of 300 Mbps for offloaded connections:

i. Load the TOE driver and bring up the interface.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig eth0 up
```

ii. Create a new policy file (say *new_policy_file*) and add the following line to associate connections with the given scheduling class.

```
src host 102.1.1.1 => offload class 0
```

iii. Compile the policy file using *COP.*

```
[root@host~]# cop -d –o <output_policy_file> <new_policy_file>
```

iv. Apply the *COP* policy.

```
[root@host~]# cxgbtool eth0 policy <output_policy_file>
```

Note *The policy applied using cxgbtool is not persistent and should be applied every time drivers are reloaded or the machine is rebooted.*

The applied cop policies can be read using,

```
[root@host~]# cat /proc/net/offload/toeX/read-cop
```

v. Set the appropriate rule for class 0.

```
[root@host~]# cxgbtool ethX sched-class params type packet level cl-rl mode
class rate-unit bits rate-mode absolute channel 0 class 0 max-rate 300000
pkt-size 1460
```

4.3. Offloaded Connections with Modified Application

The following example demonstrates the method to rate limit all TCP connections on class 0 to a rate of 300 Mbps for offloaded connections with modified application.

i. Load the TOE driver and bring up the interface.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig eth0 up
```

ii. Modify the application as mentioned in the Configuring Traffic Management section.

iii. Set the appropriate rule for class 0

```
[root@host~]# cxgbtool ethX sched-class params type packet level cl-rl mode
class rate-unit bits rate-mode absolute channel 0 class 0 max-rate 300000
pkt-size 1460
```

4.4. Inline TLS Offload Connections

Please refer Inline TLS Offload chapter to go through configuration steps. To rate limit Inline TLS Offload connections, follow the steps mentioned below:

i. Load the TOE driver and bring up the interface.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig eth0 up
```

ii. Create a new policy file and add the following line for TCP port (to be TLS offloaded), 443 in this case. Bind the connections to class 0.

```
src or dst port 443 => offload tls mss 32 bind random class 0
all => offload
```

The *all => offload* is added to ensure that rest of the TCP ports will be regular TOE offloaded.

iii. Compile the policy file using *COP.*

```
[root@host~]# cop -d –o <output_policy_file> <new_policy_file>
```

iv. Apply the *COP* policy.

```
[root@host~]# cxgbtool ethX policy <output_policy_file>
```

Note *The policy applied using cxgbtool is not persistent and should be applied every time drivers are reloaded or the machine is rebooted.*

The applied cop policies can be read using,

```
[root@host~]# cat /proc/net/offload/toeX/read-cop
```

v. Set the appropriate rule for class 0 with the required rate and burst size 16384.

```
[root@host~]# cxgbtool ethX sched-class params type packet level cl-rl mode
flow rate-unit bits rate-mode absolute channel 0 class 0 max-rate 5000 pkt-
size 1460 burst-size 16384
```

This rule will rate limit all Inline TLS connections on class 0 to 5 Mbps per connection.

5. Software/Driver Unloading

Reboot the system to unload the driver. To unload without rebooting, refer Unloading the TOE driver section of **Network (NIC/TOE)** chapter.

XXVII. Unified Boot

1. Introduction

PXE is short for Preboot eXecution Environment and is used for booting computers over an Ethernet network using a Network Interface Card (NIC). FCoE SAN boot process involves installation of an operating system to an FC/FCoE disk and then booting from it. iSCSI SAN boot process involves installation of an operating system to an iSCSI disk and then booting from it.

This section of the guide explains how to configure and use Chelsio Unified Boot Option ROM which flashes PXE, iSCSI and FCoE Option ROM onto Chelsio's adapters. It adds functionalities like PXE, FCoE and iSCSI SAN boot.

1.1. Hardware Requirements

1.1.1. Supported platforms

Following is the list of hardware platforms supported by Chelsio Unified Boot software:

- Dell T5600
- DELL PowerEdge 2950
- DELL PowerEdge T110
- DELL PowerEdge T710
- DELL PowerEdge R220
- DELL PowerEdge R720
- IBM X3650 M2
- IBM X3650 M4*
- HP Proliant DL180 gen9
- HP ProLiant DL385G2
- Supermicro X7DWE
- Supermicro X8DTE-F
- Supermicro X8STE
- Supermicro X8DT6
- Supermicro X9SRL-F
- Supermicro X9SRE-3F
- Supermicro-X10DRi
- ASUS P5KPL
- ASUS P8Z68
- Lenovo X3650 M5
- Intel DQ57TM

* If system BIOS version is lower than 1.5 and both Legacy and uEFI are enabled, system will hang during POST. Please upgrade the BIOS version to 1.5 or higher to avoid this issue.

1.1.2. Supported Switches

Following is the list of network switches supported by Chelsio Unified Boot software:

- Cisco Nexus 5010 with 5.1(3) N1 (1a) firmware.
- Arista DCS-7124S-F
- Mellanox SX_PPC_M460EX

Other platforms/switches have not been tested and are not guaranteed to work.

1.1.3. Supported Adapters

Following are the currently shipping Chelsio adapters that are compatible with Chelsio Unified Boot software:

- T62100-CR
- T62100-LP-CR
- T62100-SO-CR*
- T6425-CR
- T6225-CR
- T6225-SO-CR*
- T6225-LL-CR
- T580-CR
- T580-LP-CR
- T580-SO-CR*
- T580-OCP-SO*
- T540-CR
- T540-LP-CR
- T520-CR
- T520-LL-CR
- T520-SO-CR*
- T520-OCP-SO*
- T520-BT
- T540-BT

** Only PXE supported*

1.2. Software Requirements

Chelsio Unified Boot Option ROM software requires Disk Operating System to flash PXE ROM onto Chelsio adapters.

The installation of the following Linux distributions is supported using Chelsio inbox drivers. No separate DUDs are required.

Linux Distribution	Drivers
RHEL 8.4, 4.18.0-305.el8	PXE, FCoE, iSCSI
RHEL 8.3, 4.18.0-240.el8	
RHEL 7.9, 3.10.0-1160.el7	
RHEL 7.8, 3.10.0-1127.el7	

Note *Other kernel versions have not been tested and are not guaranteed to work.*

1.3. Pre-requisites

A DOS bootable USB flash drive or Floppy Disk is required for updating firmware, option ROM etc.

2. Secure Boot

Secure Boot, a high-performance computing software solution is a method to restrict which binaries can be executed to boot the system. With Secure Boot, the system BIOS will only allow the execution of boot loaders that carry the cryptographic signature of trusted entities. In other words, anything run in the BIOS must be “signed” with a key that the system knows is trustworthy. With each reboot of the server, every executed component is verified.

The following example describes the method to enable Secure Boot on HP ProLiant servers. Steps may differ slightly on other platforms:

i. During system boot, press F9 to run the **System Utilities**.

ii. Select **System Configuration**.

iii. Select **BIOS/Platform Configuration (RBSU)**.

iv. Select **Server Security**.

v. Select **Secure Boot Settings**.

vi. Select **Advanced Secure Boot Options**.

vii. Provide the Platform Key (PK), Key Exchange Key (KEK) and Allowed Signature Database (DB) to the respective uEFI NVRAM variables.

- **Windows**:
  - PK: Will be generated at the discretion of the platform owner (OEM). Click here for more information.
  - KEK: http://www.microsoft.com/pkiops/certs/MicCorKEKCA2011_2011-06-24.crt
  - Windows DB: http://www.microsoft.com/pkiops/certs/MicWinProPCA2011_2011-10-19.crt
  - uEFI DB: http://www.microsoft.com/pkiops/certs/MicCorUEFCA2011_2011-06-27.crt
  - Signature GUID for all the above keys: 77fa9abd-0359-4d32-bd60-28f4e78f784b
- **Linux**:
  - Use the same values for PK, KEK, Windows DB, uEFI DB and Signature ID as mentioned above.
  - In addition, provide the following values:
    - chcert.cer: *Provided in ChelsioUwire-x.x.x.x/Uboot/chelsio_key/*
    - Signature GUID for chcert.cer: 0b74ace7-6136-a493-19a9-6104d6d1e432

viii. Reboot the system, run **System Utilities** and go to **Secure Boot Settings**.

ix. Select and enable **Secure Boot Enforcement** and reboot the system.

3. Flashing Firmware and Option ROM

Depending on the boot mode selected, Chelsio Unified Boot provides the following methods to flash Firmware and Option ROM onto Chelsio adapters:

- Legacy mode: cfut4
- uEFI mode:
  - HII
  - drvcfg
  - Firmware Manager Protocol (FMP)
- OS Level:
  - cxgbtool

These methods also provide the functionality to update/erase Hardware configuration and Phy Firmware files.

Important *It is highly recommended to use the same Option ROM (type and version) on all the Chelsio adapters present in the system.*

3.1. Preparing USB flash drive

This document assumes that you are using a USB flash drive as a storage media for the necessary files. Follow the steps below to prepare the drive:

i. Create a DOS bootable USB flash drive. (Click here for instructions)

ii. Create a directory *CHELSIO* on the USB flash drive.

iii. If you haven't done already, download the driver package from Chelsio Download Center.

iv. Untar the downloaded package and change your working directory to *OptionROM* directory.

```
[root@host~]# cd <driver_package>/Uboot/OptionROM
```

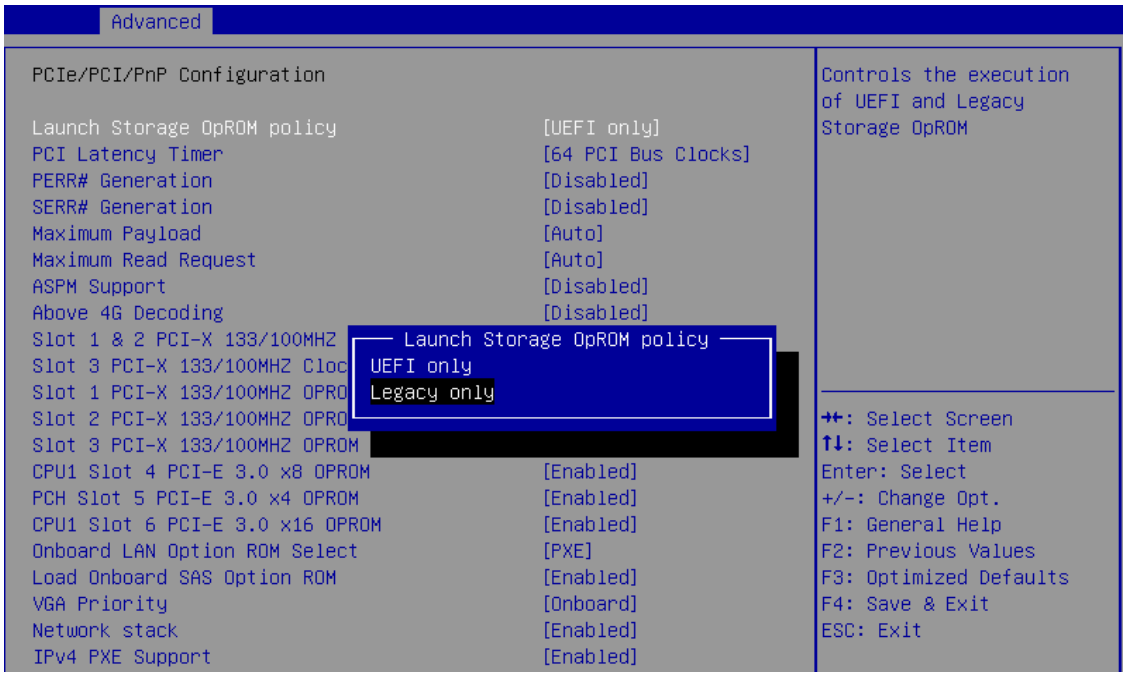
v. Copy all the files and place them in the *CHELSIO* directory created on the USB flash drive.

vi. Plug-in the USB flash drive in the system on which the Chelsio CNA is installed.

vii. Reboot the system.

3.2. Legacy

i. In BIOS, configure the system having Chelsio adapter to boot in Legacy mode.



ii. Boot the system from the plugged in USB flash drive and change your working directory to CHELSIO directory.

```
C:\>cd CHELSIO
```

iii. Run the following command to list all Chelsio adapters present on the system. The list displays a unique index for each adapter found.

```
C:\CHELSIO>cfut4 -l
```

iv. Delete any previous version of Option ROM flashed onto the adapter.

```
C:\CHELSIO>cfut4 -d <idx> -xb
```

Here, `idx` is the adapter index found in step iii (0 in this case)

```
C:\CHELSIO>cfut4 -d 0 -xb

Chelsio T5/T6 Flash Utility v1.5

Erasing serial flash sector(s) ... Done
Reboot machine for changes to take effect
```

v. Delete any previous firmware using the following command.

```
C:\CHELSIO>cfut4 -d <idx> -xh -xf
```

```
C:\CHELSIO>cfut4 -d 0 -xh -xf

Chelsio T5/T6 Flash Utility v1.5

Erasing serial flash sector(s) ... Done
Erasing serial flash sector(s) ... Done
Reboot machine for changes to take effect
```

vi. Delete any previous Option ROM settings.

```
C:\CHELSIO>cfut4 -d <idx> -xc
```

```
C:\CHELSIO>cfut4 -d 0 -xc

Chelsio T5/T6 Flash Utility v1.5

Erasing serial flash sector(s) ... Done
Reboot machine for changes to take effect
```

vii. Run the following command to flash the appropriate firmware.

```
C:\CHELSIO>cfut4 -d <idx> -uf <firmware_file>.bin
```

Here, `firmware_file` is the firmware image file present in the *CHELSIO* directory.

```
C:\CHELSIO>cfut4 -d 0 -uf T6FW-1~1.BIN

Chelsio T5/T6 Flash Utility v1.5

Erasing serial flash sector(s) ... Done
Writing Image at Base 00080000 ... Done
Writing Image at Base 00088000 ... Done
Writing Image at Base 00090000 ... Done
Writing Image at Base 00098000 ... Done
Writing Image at Base 000a0000 ... Done
Writing Image at Base 000a8000 ... Done
Writing Image at Base 000b0000 ... Done
Writing Image at Base 000b8000 ... Done
Writing Image at Base 000c0000 ... Done
Writing Image at Base 000c8000 ... Done
Writing Image at Base 000d0000 ... Done
Writing Image at Base 000d8000 ... Done
Writing Image at Base 000e0000 ... Done
Writing Image at Base 000e8000 ... Done
Writing Image at Base 000f0000 ... Done
Writing Image at Base 000f8000 ... Done
Reboot machine for changes to take effect
```

viii. Flash the Unified Option ROM onto the Chelsio adapter using the following command.

```
C:\CHELSIO>cfut4 -d <idx> -ub cubt4.bin
```

Here, `cubt4.bin` is the unified Option ROM image file present in the *CHELSIO* directory.

```
C:\CHELSIO>cfut4 -d 0 -ub cubt4.bin

Chelsio T5/T6 Flash Utility v1.5

Erasing serial flash sector(s) ... Done
Writing Image at Base 00000000 ... Done
Writing Image at Base 00008000 ... Done
Writing Image at Base 00010000 ... Done
Writing Image at Base 00018000 ... Done
Writing Image at Base 00020000 ... Done
Writing Image at Base 00028000 ... Done
Writing Image at Base 00030000 ... Done
Writing Image at Base 00038000 ... Done
Writing Image at Base 00040000 ... Done
Writing Image at Base 00048000 ... Done
Writing Image at Base 00050000 ... Done
Writing Image at Base 00058000 ... Done
Writing Image at Base 00060000 ... Done
Writing Image at Base 00068000 ... Done
Erasing serial flash sector(s) ... Done
Writing Image at Base 00070000 ... Done
Reboot machine for changes to take effect
```

ix. In case of multiple adapters in the sytem, please repeat the steps from iv. to viii. to update/flash the firmware and Option ROM on all the adapters.

x. To configure the base MAC address (optional), use the below command:

```
C:\CHELSIO>cfut4 -d <idx> -um <Hex MAC Address>
```

Example:

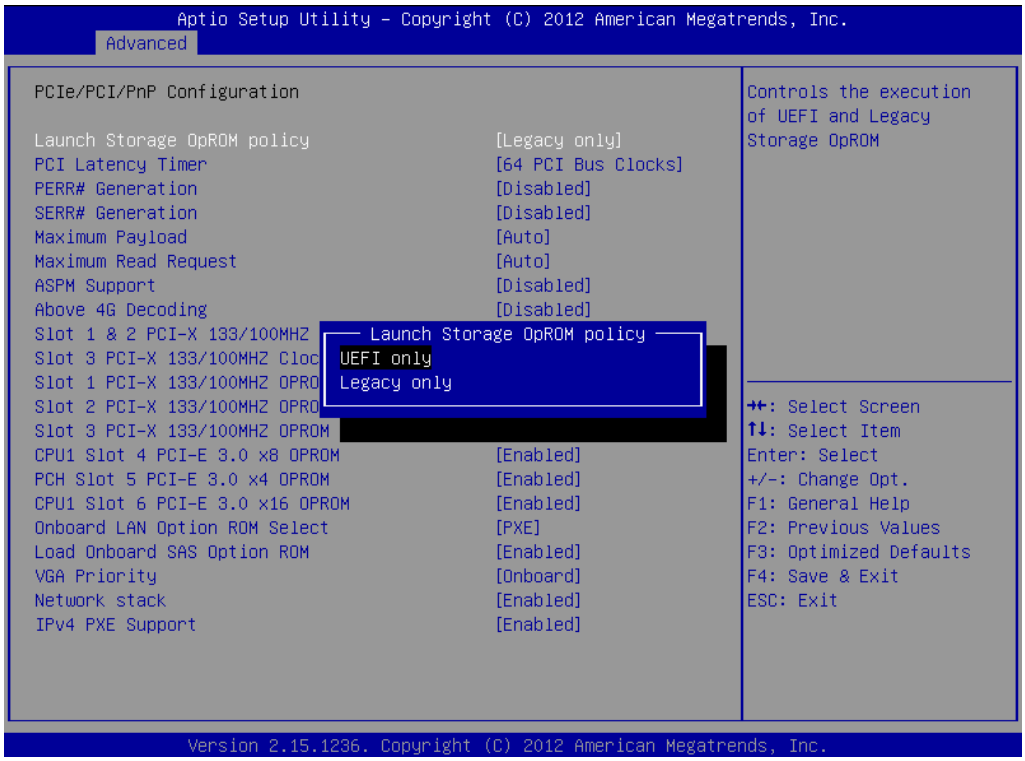
```
C:\CHELSIO>cfut4 -d 0 -um 000743000123
```

xi. Reboot the system for changes to take effect.

3.3. uEFI

3.3.1. Loading uEFI driver

i. In BIOS, configure the system having Chelsio adapter to boot in uEFI mode.

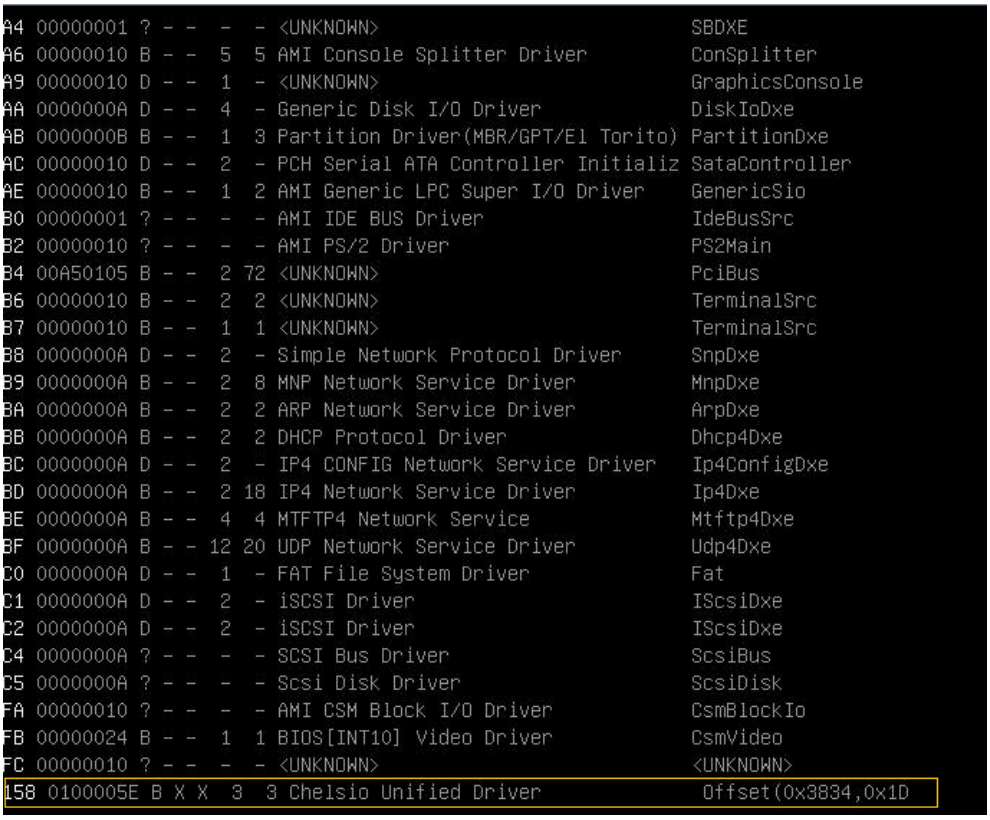


Note *For Supermicro systems, enable **Network Stack*** as well before proceeding.

ii. Boot to EFI Shell.

```
EFI Shell version 2.31 [4.654]
Current running mode 1.1.2
Device mapping table
  fs0  :Removable HardDisk - Alias hd83b0f0b blk0
        PciRoot(0x0)/Pci(0x1d,0x0)/USB(0x1,0x0)/USB(0x5,0x0)/HD(1,MBR,0x0fdb738d,0x800,0x78b800)
  blk0 :Removable HardDisk - Alias hd83b0f0b fs0
        PciRoot(0x0)/Pci(0x1d,0x0)/USB(0x1,0x0)/USB(0x5,0x0)/HD(1,MBR,0x0fdb738d,0x800,0x78b800)
  blk1 :HardDisk - Alias (null)
        PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0x0,0x0)/HD(1,MBR,0x00092b0c,0x3f,0x9c25fe)
  blk2 :HardDisk - Alias (null)
        PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0x0,0x0)/HD(2,MBR,0x00092b0c,0x9c263d,0x88b8fdc)
  blk3 :HardDisk - Alias (null)
        PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0x0,0x0)/HD(3,MBR,0x00000000,0x927be19,0x14019e7)
  blk4 :HardDisk - Alias (null)
        PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0x0,0x0)/HD(4,MBR,0x00000000,0xa67d83f,0x13fe849)
  blk5 :BlockDevice - Alias (null)
        PciRoot(0x0)/Pci(0x1f,0x2)/Sata(0x0,0x0)
  blk6 :Removable BlockDevice - Alias (null)
        PciRoot(0x0)/Pci(0x1d,0x0)/USB(0x1,0x0)/USB(0x5,0x0)

Press ESC in 1 seconds to skip startup.nsh, any other key to continue.
Shell> _
```

iii. Issue command `drivers` to determine if Chelsio uEFI driver is already loaded. The below image shows that the driver is loaded.



If the driver is not loaded, continue to step (v)

iv. Note the handle and unload the driver.

```
fs0:\CHELSIO\> unload -n <driver_handle>
```

Example:

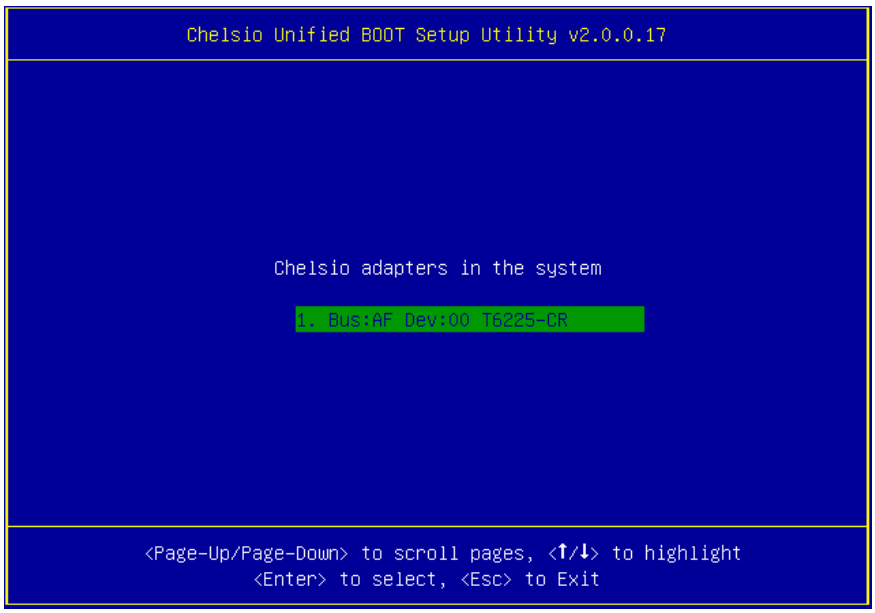
v. Load the uEFI driver (*ChelsioUD.efi*) present in the *CHELSIO* directory.

3.3.2. drvcfg

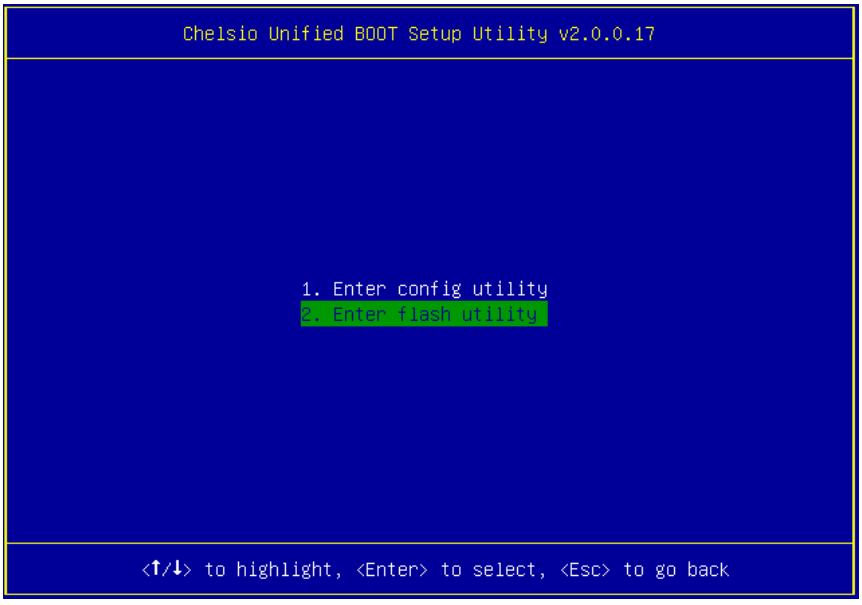
i. Please ensure that Chelsio uEFI driver is loaded correctly as mentioned in Loading uEFI driver section.
ii. Run the following command to launch the Unified Boot Setup utility.

```
fs0:\> drvcfg -s_
```

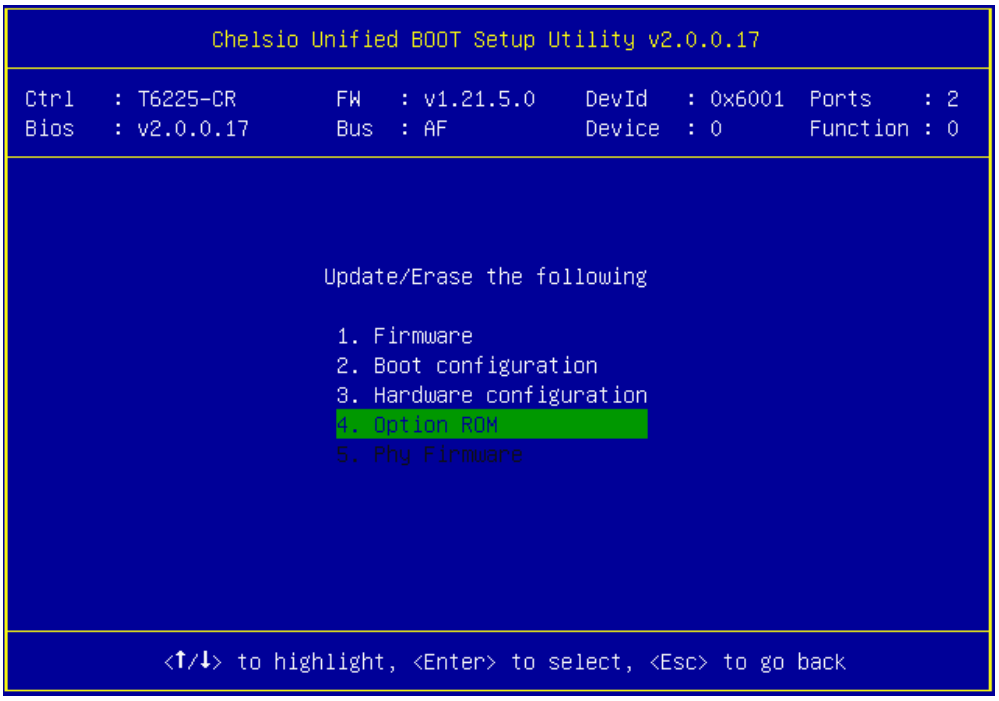
iii. Choose the Chelsio adapter which needs to be configured.



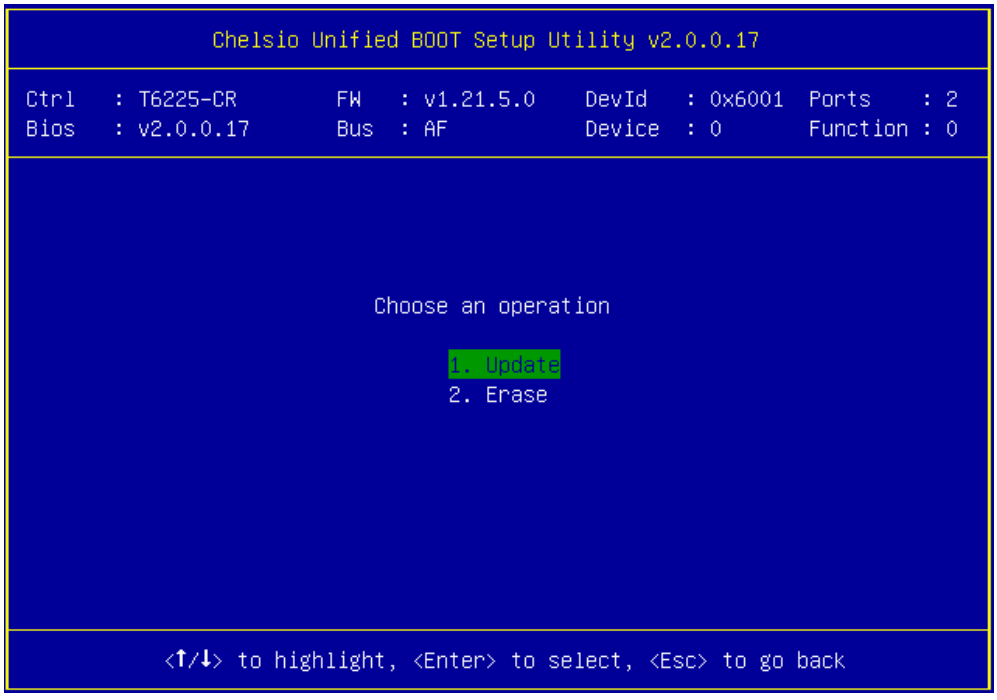
iv. Highlight **Enter flash utility** and press [Enter].



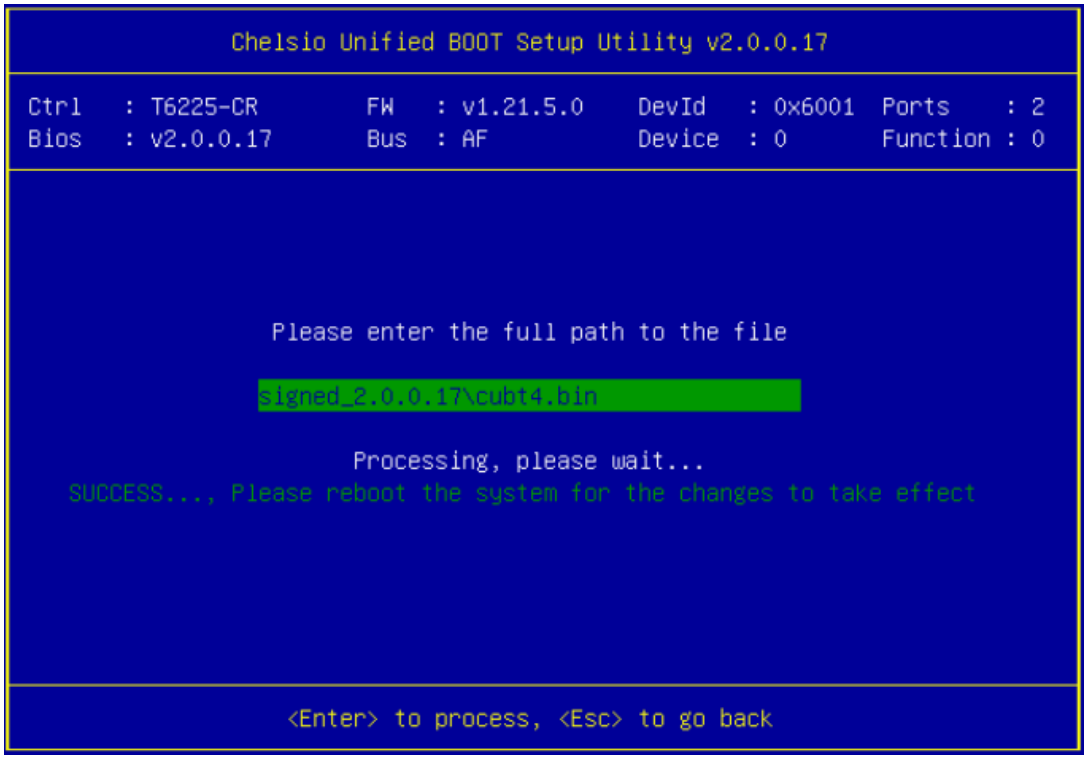
v. Highlight **Option ROM** and press [Enter].



vi. Highlight **Update** and press [Enter].



vii. Enter the path to the Option ROM file and press [Enter].



viii. Similarly, you can use the above method to update Firmware present in the *CHELSIO* directory.

ix. Reboot the machine for changes to take effect.

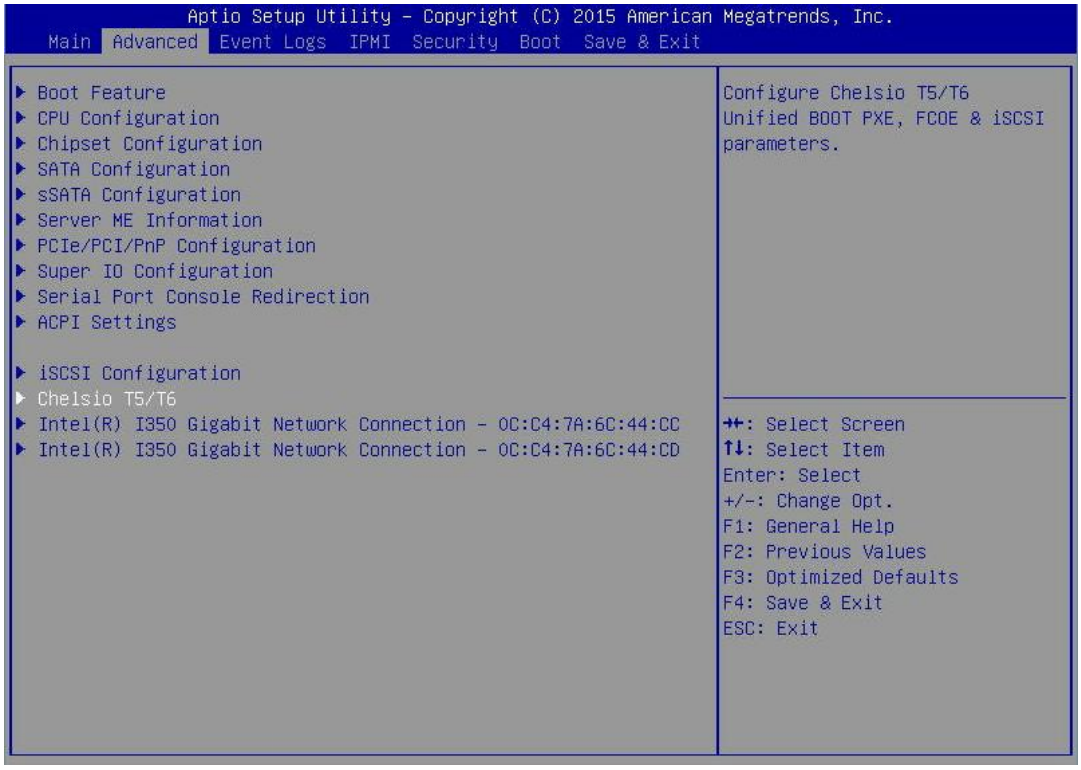
3.3.3. HII

i. Go into the BIOS setup.

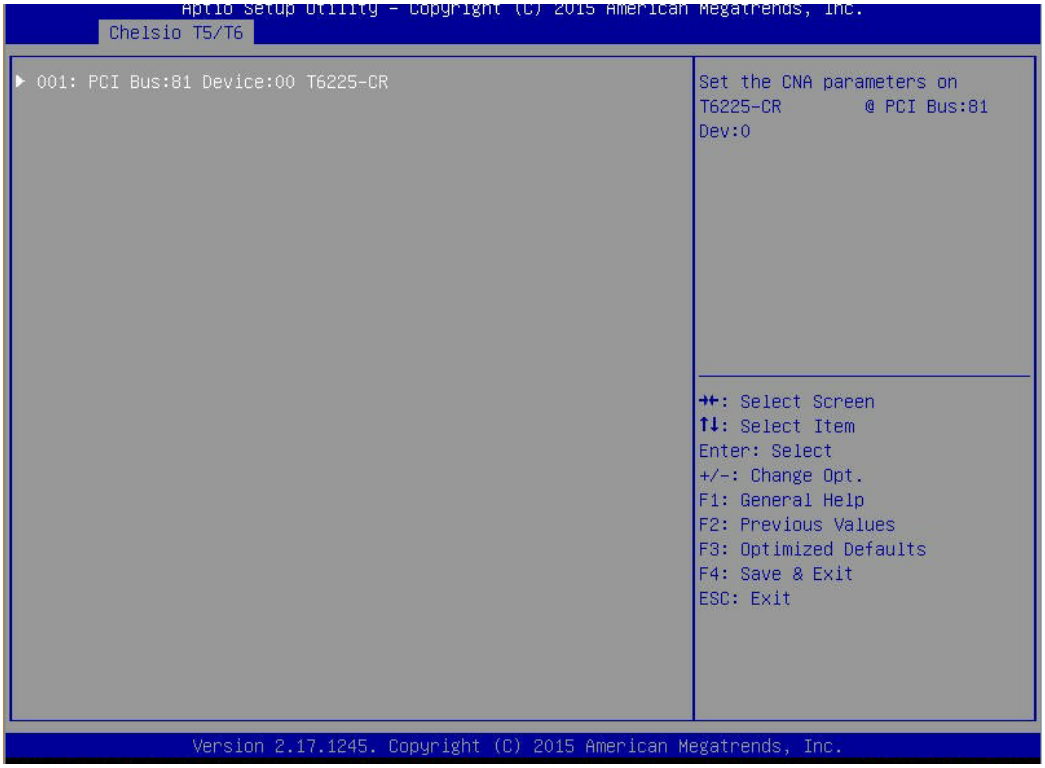
ii. Chelsio HII should be listed as **Chelsio T5/T6** as shown below. Highlight it and press [Enter].

If Chelsio T5/T6 is not listed,

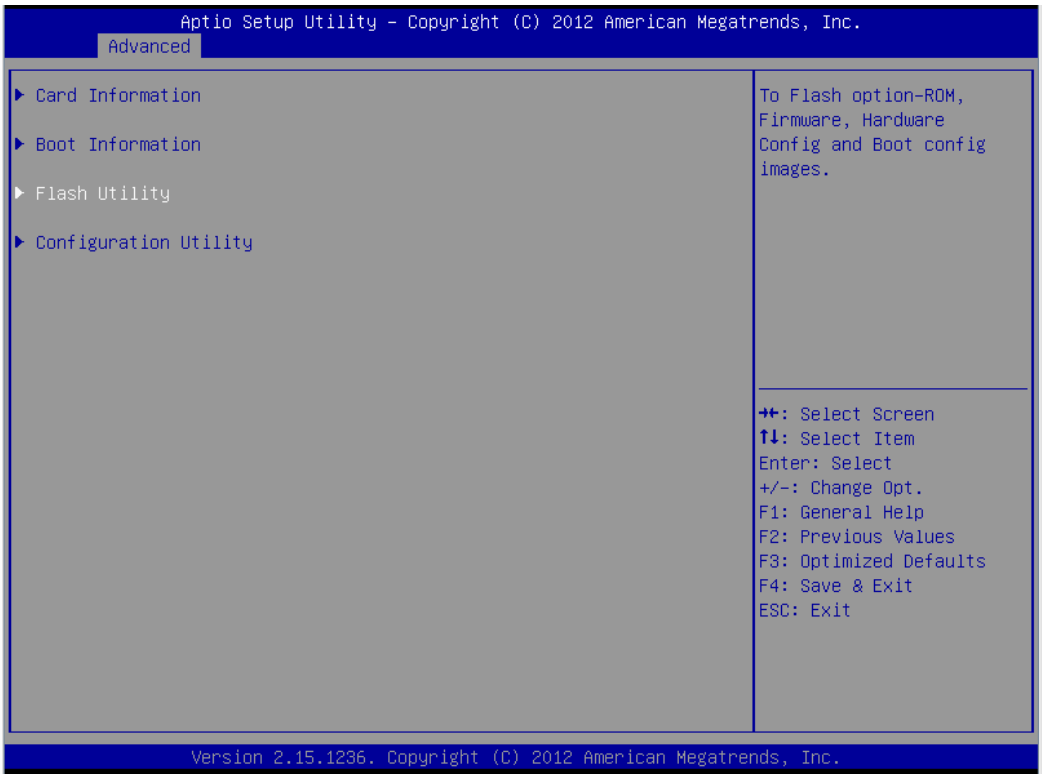
- Load the Chelsio uEFI driver as mentioned in Loading uEFI driver section.
- Flash the Option ROM and Firmware as mentioned in drvcfg section.



iii. Highlight the Chelsio adapter to be configured and press [Enter].



iv. Highlight **Flash Utility** and press [Enter].



v. Erase or update firmware using the methods explained below:

   a. Erase existing firmware

      i. Select [Erase] as Flash Operation
      ii. Select [FW File] as Flash File Type
      iii. Select Update/Erase
      iv. Press [Y] to confirm

   b. Update firmware

      i. Select [Update] as Flash Operation
      ii. Select [FW File] as Flash File Type
      iii. Enter full path to the firmware file for Enter File Name, e.g., CHELSIO\t6fw-1.16.29.0.bin.
      iv. Press [Enter]
      v. Select Update/Erase
      vi. Press [Y] to confirm

vi. Similarly, you can use the above method to update/erase Option ROM present in the *CHELSIO* directory.

vii. Reboot the machine for changes to take effect.

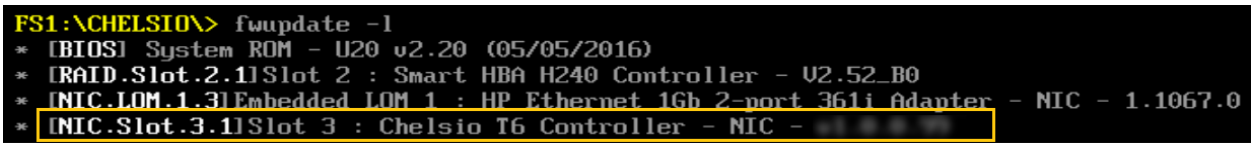
3.3.4. Firmware Management Protocol (FMP)

HP machines support Firmware Management Protocol (FMP) interface, in addition to HII. This can be used to update the Option ROM on Chelsio adapters.

- **Enabling FMP**

i. Please ensure that Chelsio uEFI driver is loaded correctly as mentioned in Loading uEFI driver section

ii. Run the command `fwupdate -l` and Chelsio T6 adapter should be listed as shown below:



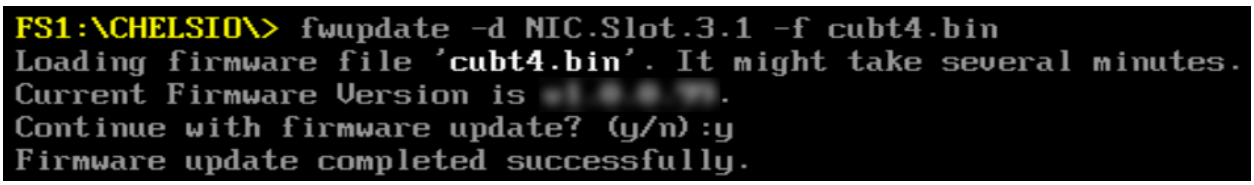
- **Upgrading Firmware**

▪ **Using CLI**

i. Use the adapter's device name to update the firmware:

```
FS1:\CHELSIO\> fwupdate -d <device_name> -f cubt4.bin
```

Example:

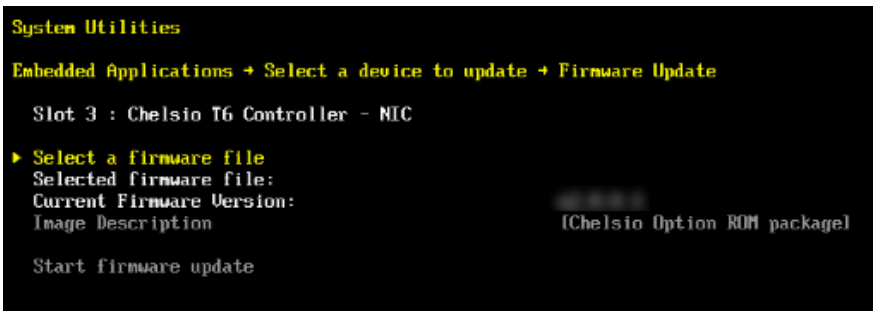


ii. Reboot machine for changes to take effect.

▪ **Using FMP**

i. Reboot system and press F9 to access **System Utilities**

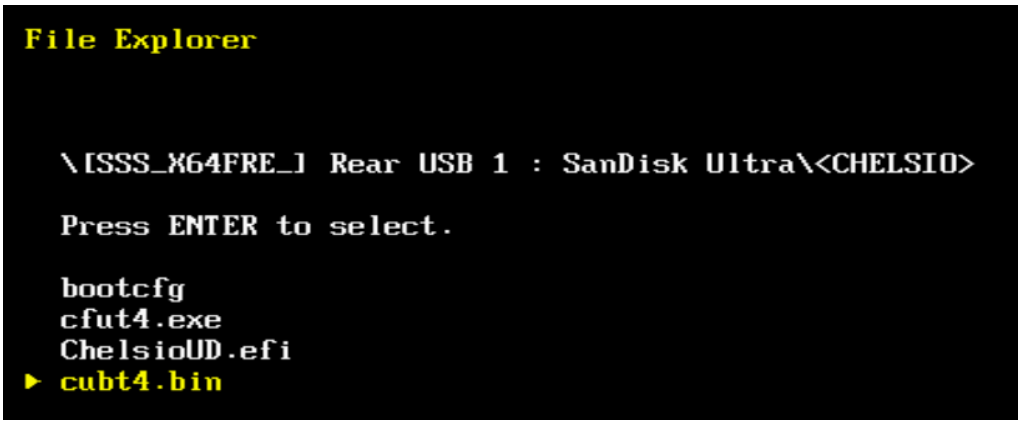
ii. Go to **Embedded Applications → Firmware Update → Chelsio T6 Controller**



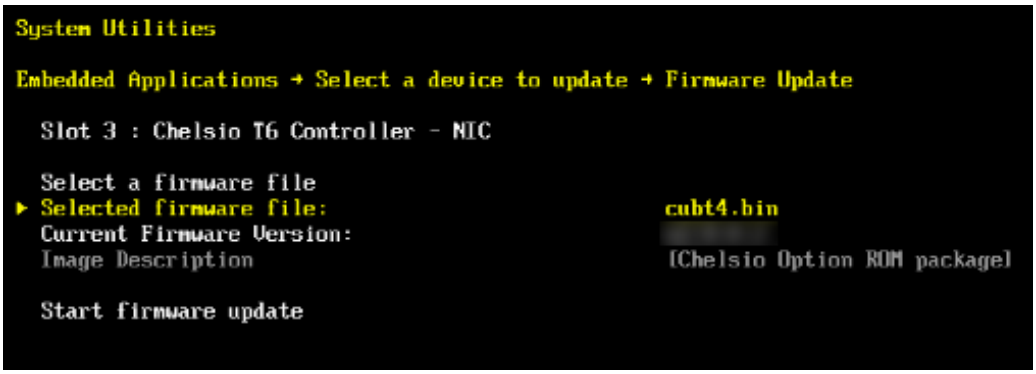
iii. Highlight **Select a firmware file** option and hit [Enter].

iv. Select the USB flash drive which contains the latest Option ROM and hit [Enter].

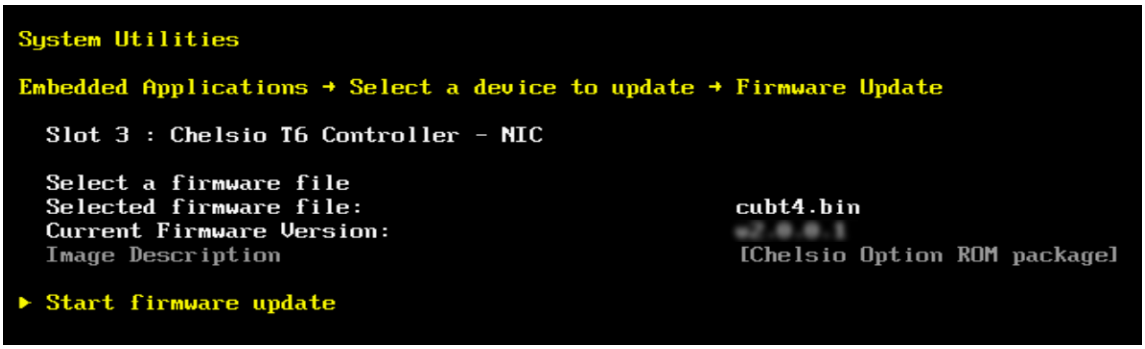
v. Select Option ROM file *cubt4.bin* and hit [Enter].



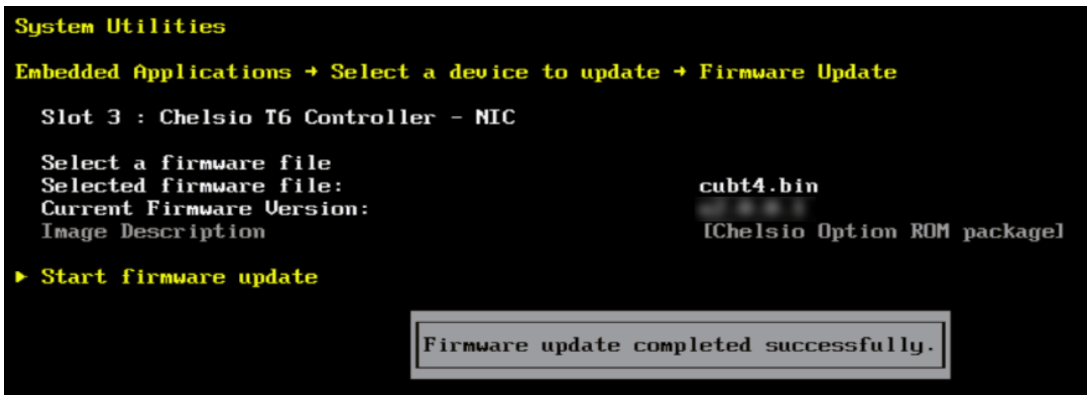
The file should show up in the **Selected firmware file** field.



vi. Select **Start firmware update** and hit [Enter].



vii. After **Firmware update completed successfully** prompt appears, reboot the machine for changes to take effect.



3.4. cxgbtool (OS Level)

Follow the steps mentioned below to flash the Option ROM onto Chelsio adapters using *cxgbtool* utility:

i. If not done already, install the Network driver and cxgbtool.

```
[root@host~]# cd ChelsioUwire-x.x.x.x
[root@host~]# make install
```

ii. Load the Network driver.

```
[root@host~]# modprobe cxgb4
```

iii. Delete any previous version of Option ROM flashed onto the adapter.

```
[root@host~]# cxgbtool ethX loadboot clear
```

iv. Flash the Option ROM onto the Chelsio adapter

```
[root@host~]# cd ChelsioUwire-x.x.x.x/Uboot/OptionROM/
[root@host~]# cxgbtool ethX loadboot cubt4.bin
```

v. Flash the default boot configuration onto the adapter.

```
[root@host~]# cd ChelsioUwire-x.x.x.x/Uboot/OptionROM/
[root@host~]# cxgbtool ethX loadboot-cfg boot.cfg
```

vi. In case of multiple adapters in the system, please repeat the steps from iii. to v. to update/flash the Option ROM on all the adapters.

vii. Reboot the system for changes to take effect.

4. Configuring PXE Server

The following components are required to configure a PXE Server:

- DHCP Server
- TFTP Server

PXE server configuration steps for different operating systems can be found on following links:

Note *Chelsio Communications does not take any responsibility regarding contents given in below mentioned links. They are provided for example purposes only.*

Linux

- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/installation_guide/chap-installation-server-setup

Windows

- http://technet.microsoft.com/en-us/library/cc771670%28WS.10%29.aspx
- http://tftpd32.jounin.net/ (Use port # 67, set PXE option and provide bootable file name in settings)
- http://unattended.sourceforge.net/pxe-win2k.html

VMware

- http://www.vstellar.com/2017/07/25/automating-esxi-deployment-using-pxe-boot-and-kickstart/
- http://fdo-workspace.blogspot.in/2016/11/building-tftp-dhcp-for-pxe-esxi-65.html

5. PXE Boot Process

Before proceeding, please ensure that the Chelsio adapter has been flashed with the provided firmware and Option ROM (See Flashing Firmware and option ROM).

5.1. Legacy PXE Boot

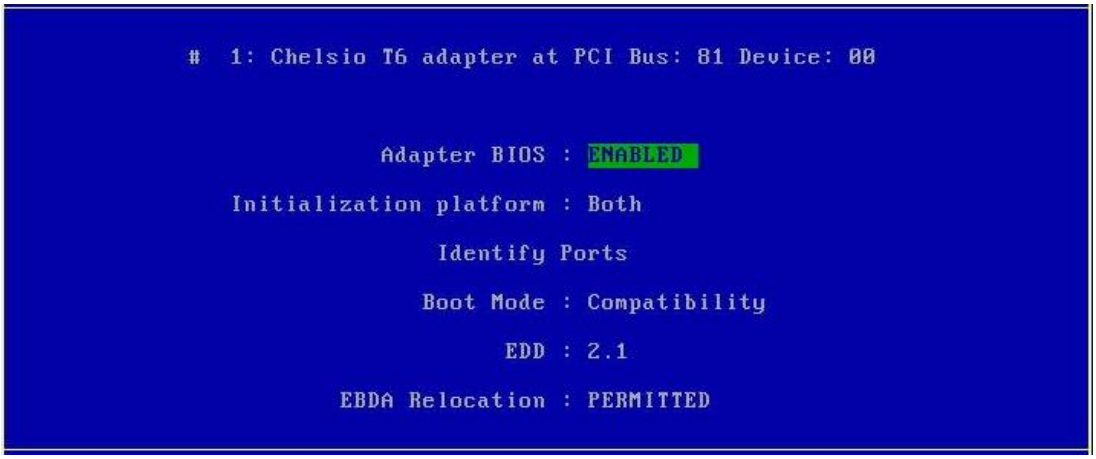
i. After configuring the PXE server, make sure the PXE server works. Then reboot the client machine.

ii. Press [Alt+C] when the message to configure Chelsio adapters appears on the screen.

iii. The configuration utility will appear as below:

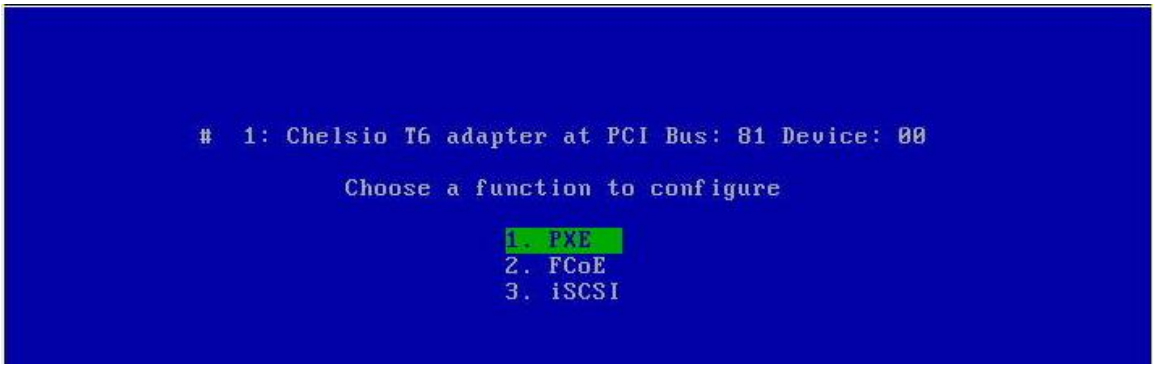
Choose the adapter on which you flashed the option ROM image. Hit [Enter].

iv. Enable the adapter BIOS using arrow keys if not already enabled. Hit [Enter].

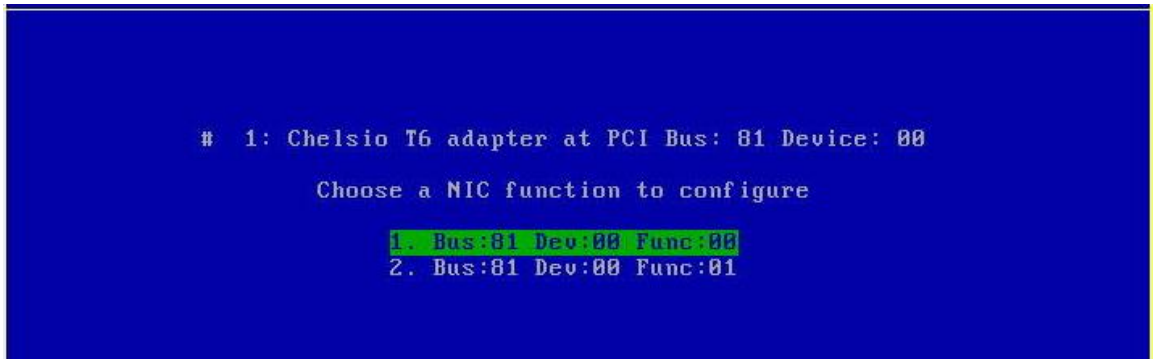


Note *Use the default values for Boot Mode, EDD and EBDA Relocation parameters, unless instructed otherwise.*

v. Choose *PXE* from the list to configure. Hit [Enter].



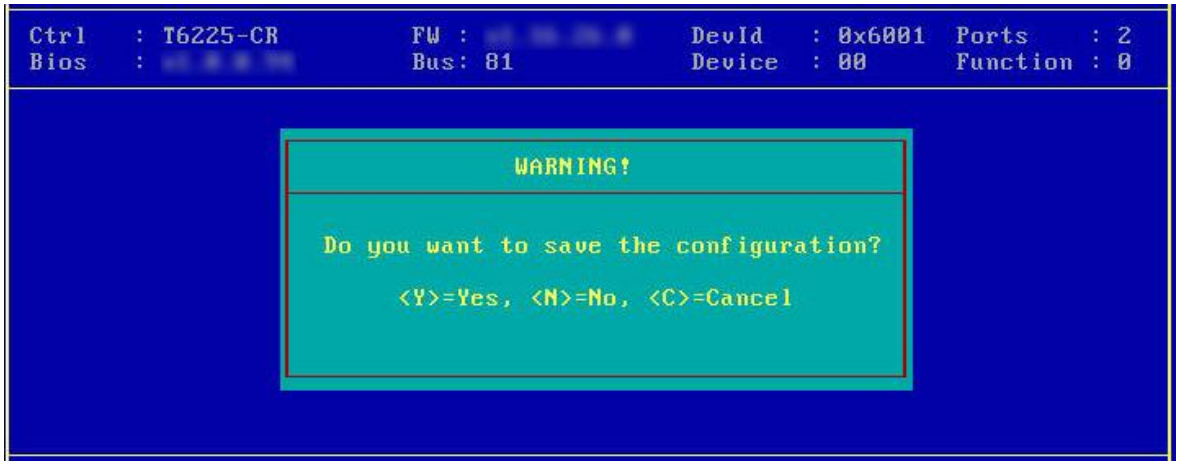
vi. Use the arrow keys to highlight the appropriate function among the supported NIC functions and hit [Enter] to select.



vii. Enable NIC function bios if not already enabled.

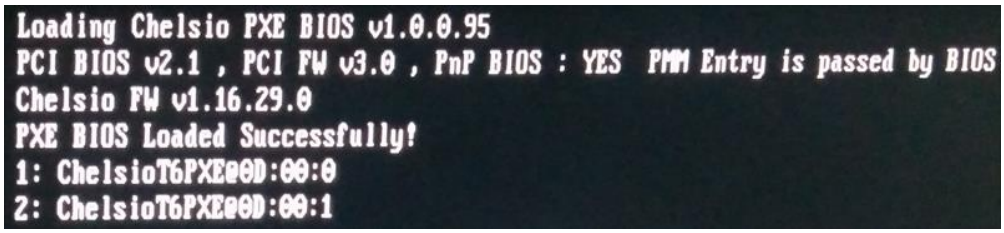
Choose the boot port to try the PXE boot. It is recommended to only enable functions and ports which are going to be used. Please note that enabling NIC Func 00 will enable port 0 for PXE, enabling NIC Func 01 will enable port 1 and so on for NIC function.

viii. Hit [F10] or [Esc] and then [Y] to save configuration changes.

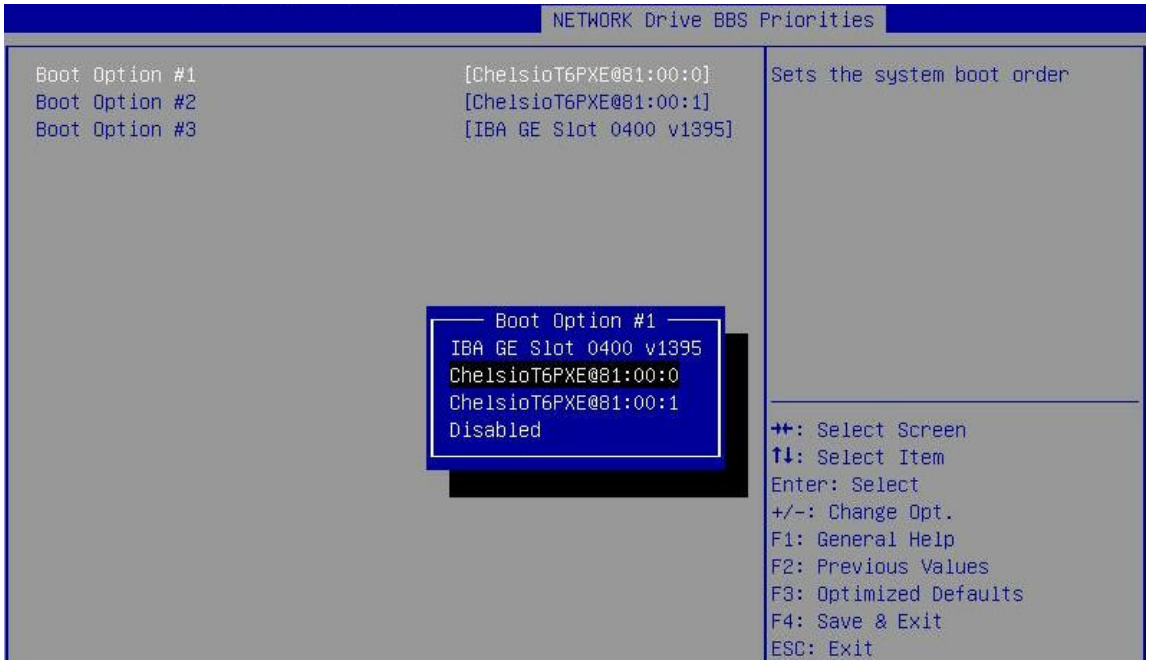


ix. Reboot the system.

x. Allow the Chelsio option ROM to initialize and setup PXE devices. DO NOT PRESS ALT-S to skip Chelsio option ROM.



xi. In the system setup, choose any of the Chelsio PXE devices as the first boot device.



xii. Reboot. DO NOT PRESS ALT-S to skip Chelsio option ROM, during POST.

xiii. Hit [F12] key when prompted to start PXE boot.

5.2. uEFI PXE Boot

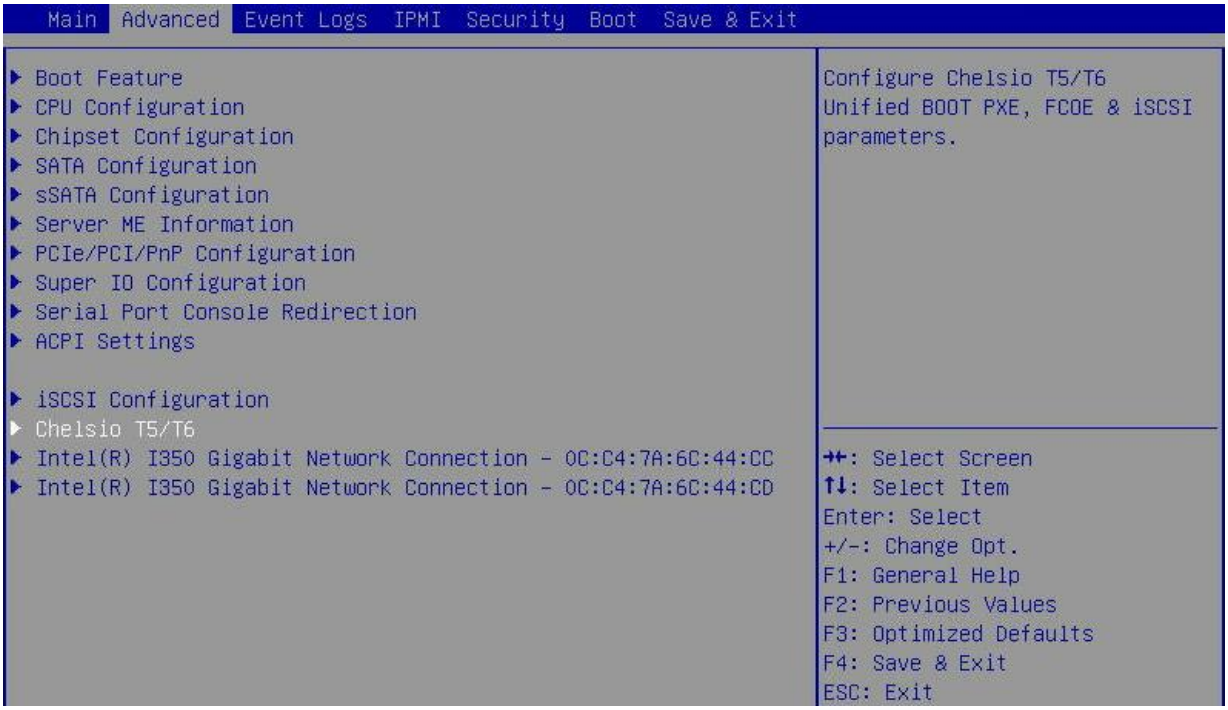
Important

- *Only uEFI v2.3.1, v2.4 and v2.5 supported.*
- *Any other uEFI version is NOT SUPPORTED and may render your system unusable.*

5.2.1. HII

This section describes the method to configure and use Chelsio uEFI PXE interfaces using HII.

i. Reboot the system and go into the BIOS setup.
ii. Chelsio HII should be listed as **Chelsio T5/T6**. Highlight it and press [Enter].

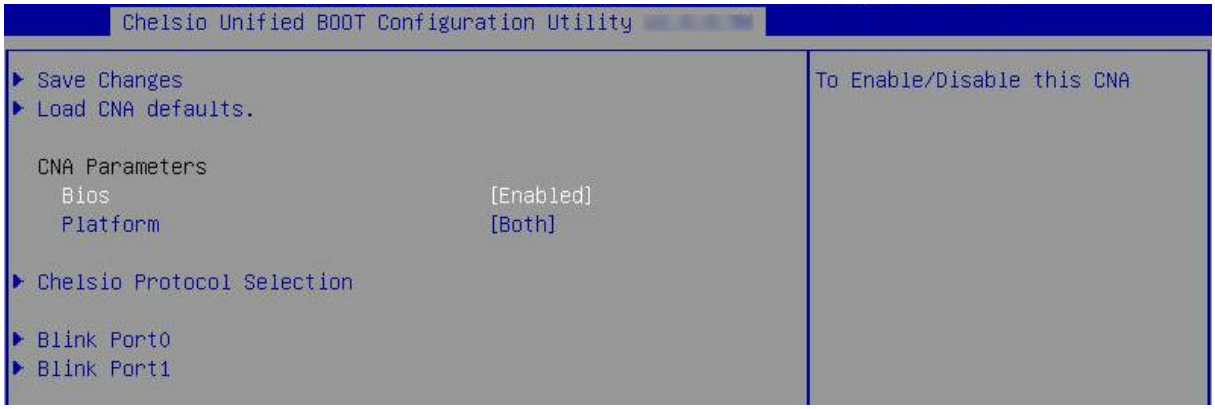


Note *Please ensure that Chelsio uEFI driver is loaded correctly as mentioned in Loading uEFI driver section.*

iii. Select the Chelsio adapter to be configured and press [Enter].

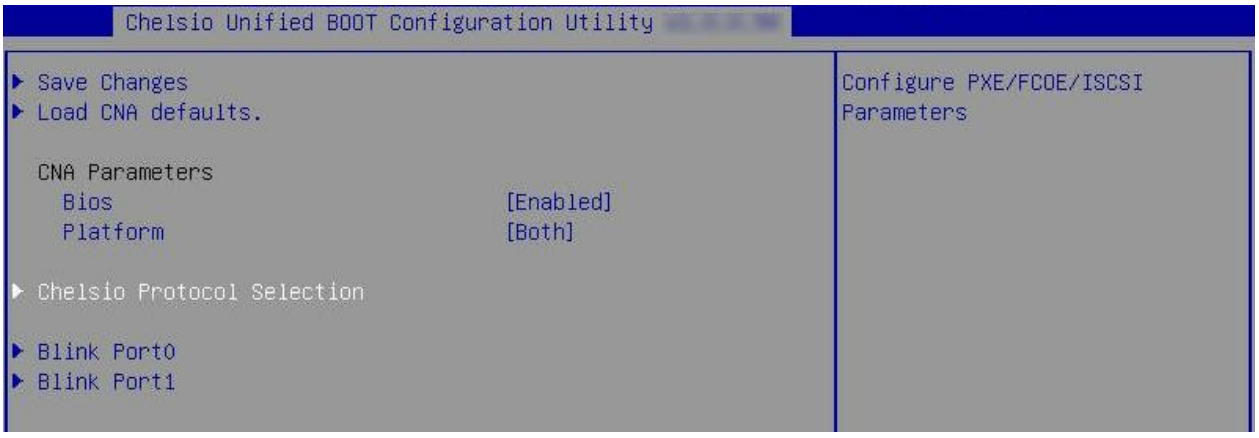
iv. Select **Configuration Utility** and press [Enter].

v. Enable adapter BIOS if not already enabled.

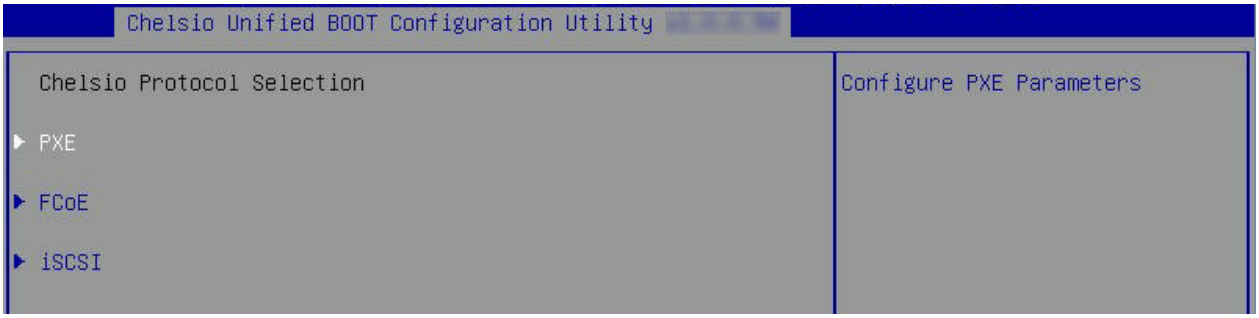


Note *It is highly recommended that you use the **Save Changes** option every time a parameter/option is changed.*

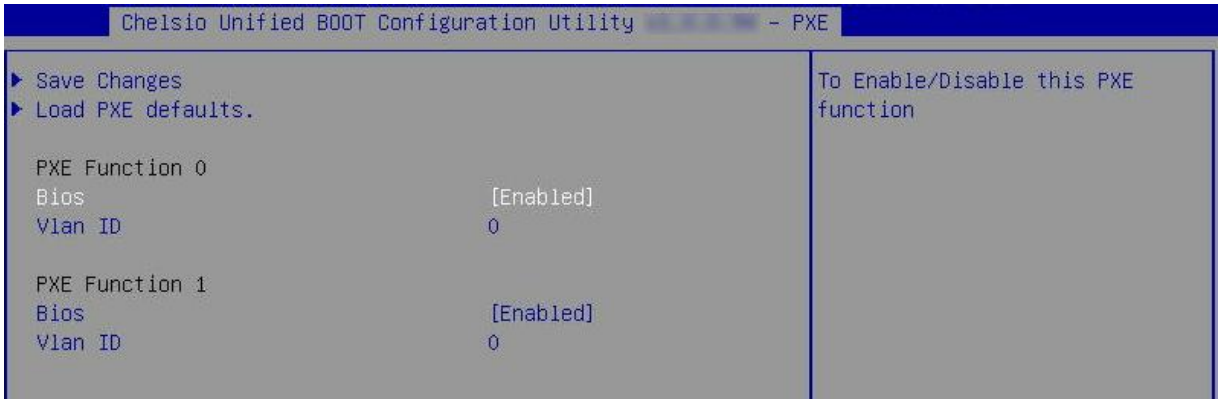
vi. Select **Chelsio Protocol Selection** and press [Enter].



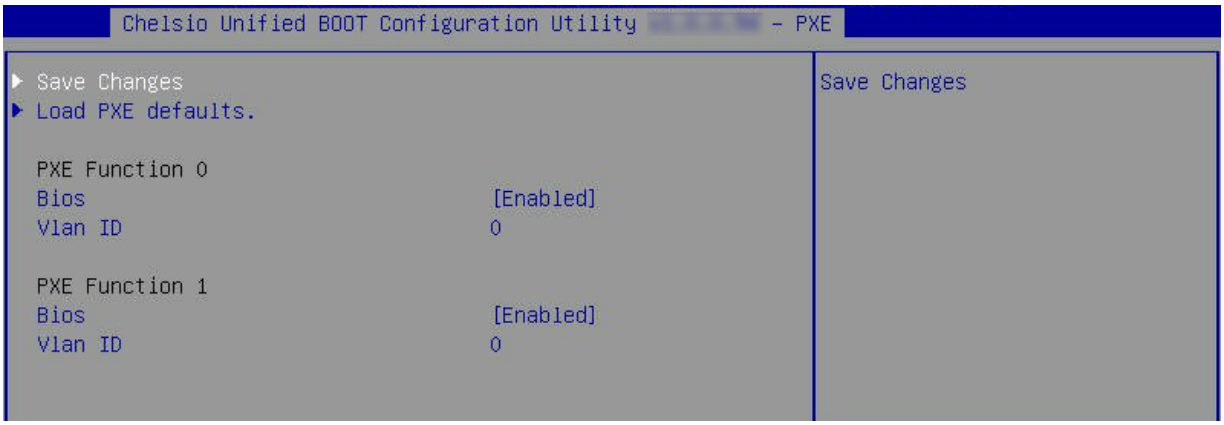
vii. Select **PXE** and press [Enter].



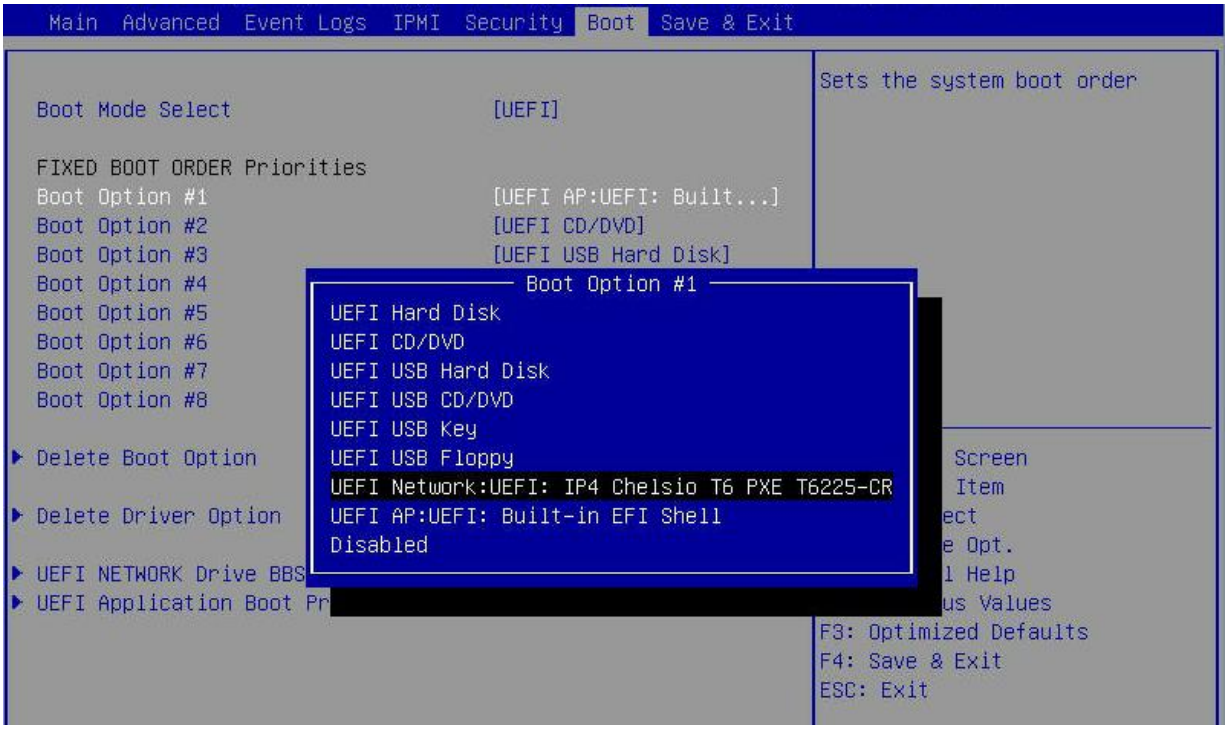
viii. Choose the boot port to try PXE boot. It is recommended to enable only those functions and ports which are going to be used. Please note that enabling PXE Function 0 will enable port 0 for PXE, enabling PXE Function 1 will enable port 1 and so on, for NIC function.



ix. Select **Save Changes** and press [Enter].



x. Reboot the system and in BIOS, choose any of the available Chelsio PXE devices.



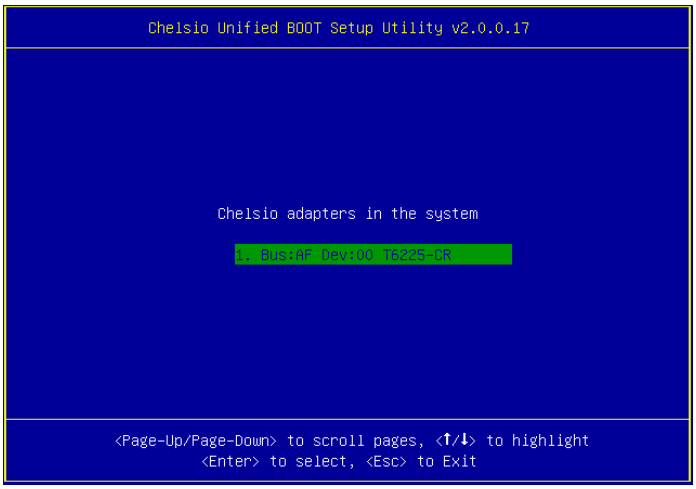
xi. Reboot and hit [F12] key when prompted to start PXE boot.

5.2.2. drvcfg

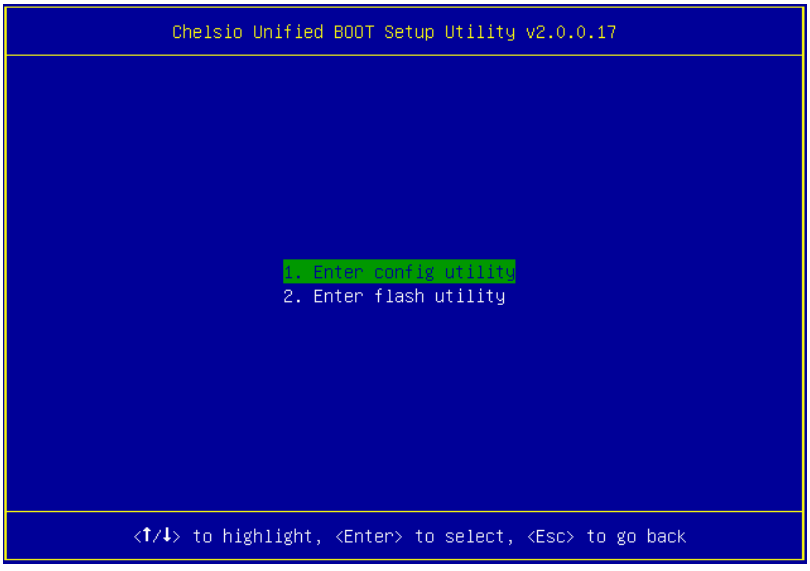
This section describes the method to configure and use Chelsio uEFI PXE interfaces using drvcfg.

i. Boot the system into EFI shell.
ii. Run the following command to launch the Unified Boot Setup utility.

iii. Choose the Chelsio adapter which needs to be configured.



iv. Highlight **Enter config utility** and press [Enter].



v. Further configuration steps are similar from step (iv) of Legacy PXE Boot section.

6. FCoE Boot Process

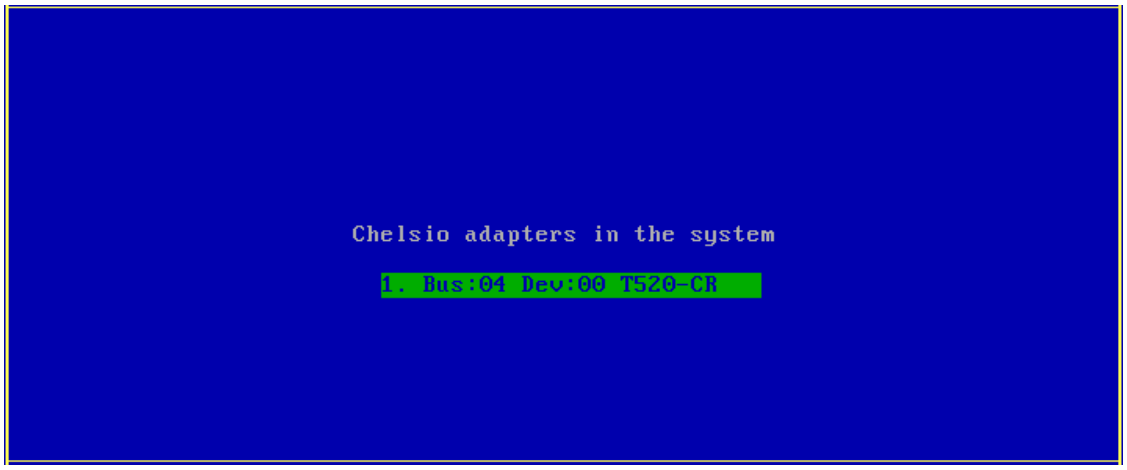
Before proceeding, please ensure that the Chelsio CNA has been flashed with the provided firmware and option ROM (See Flashing firmware and option ROM).

6.1. Legacy FCoE Boot

i. Reboot the system.

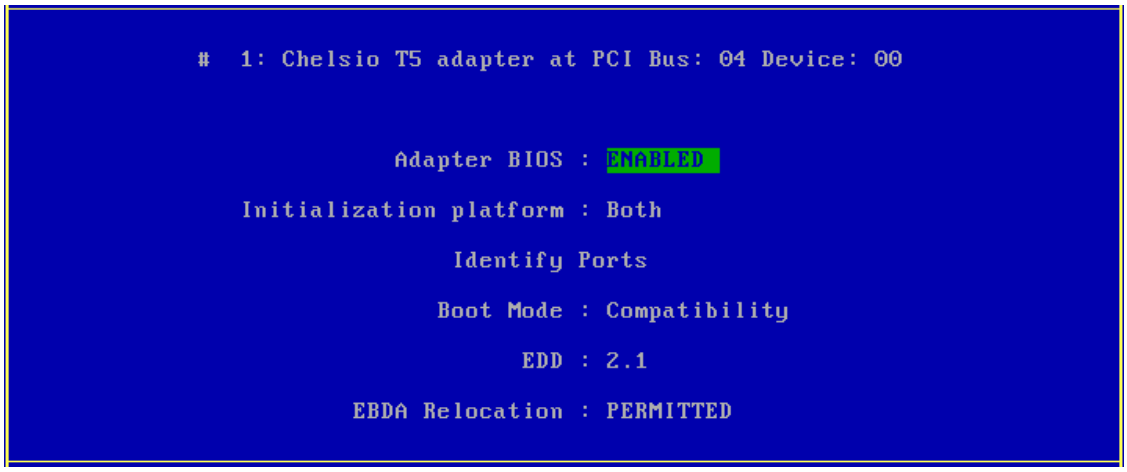
ii. Press [Alt+C] when the message to configure Chelsio adapters appears on the screen.

iii. The configuration utility will appear as below:



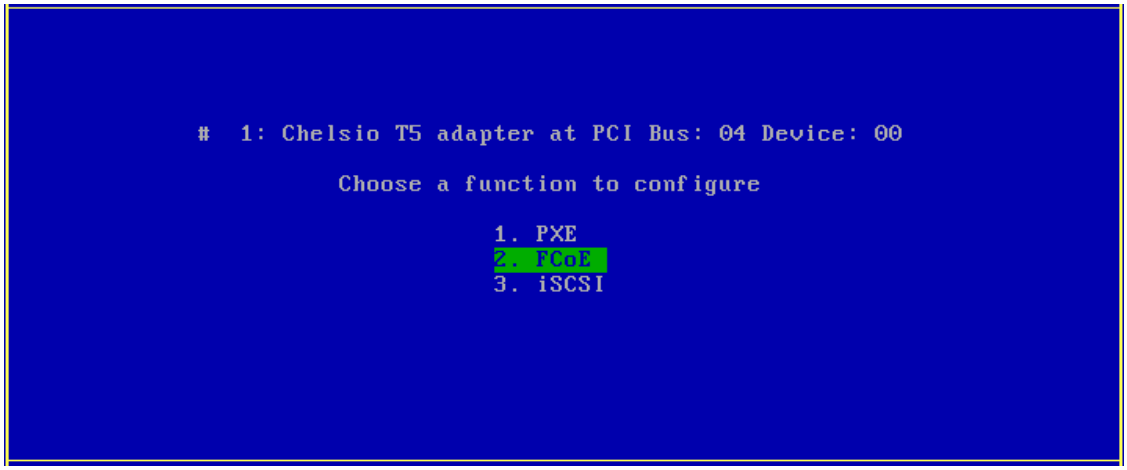
Choose the adapter on which you flashed the option ROM image. Hit [Enter].

iv. Enable the adapter BIOS if not already enabled. Hit [ENTER].

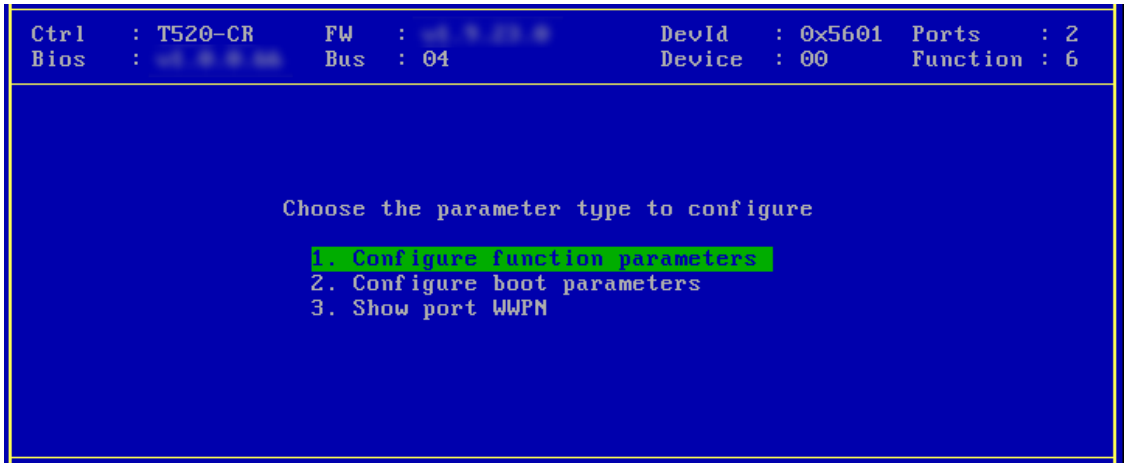


Note *Use the default values for Boot Mode, EDD and EBDA Relocation parameters, unless instructed otherwise.*

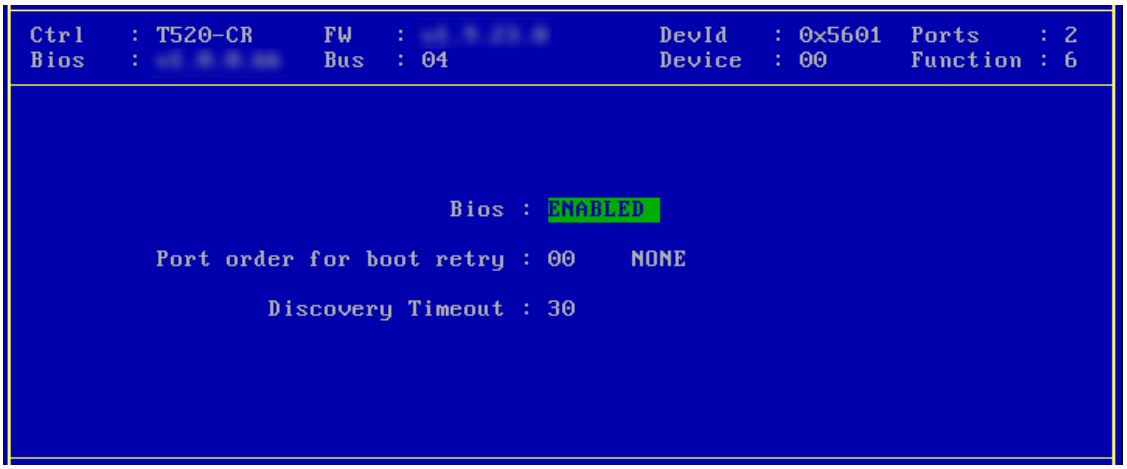
v. Choose FCoE from the list to configure and hit [Enter].



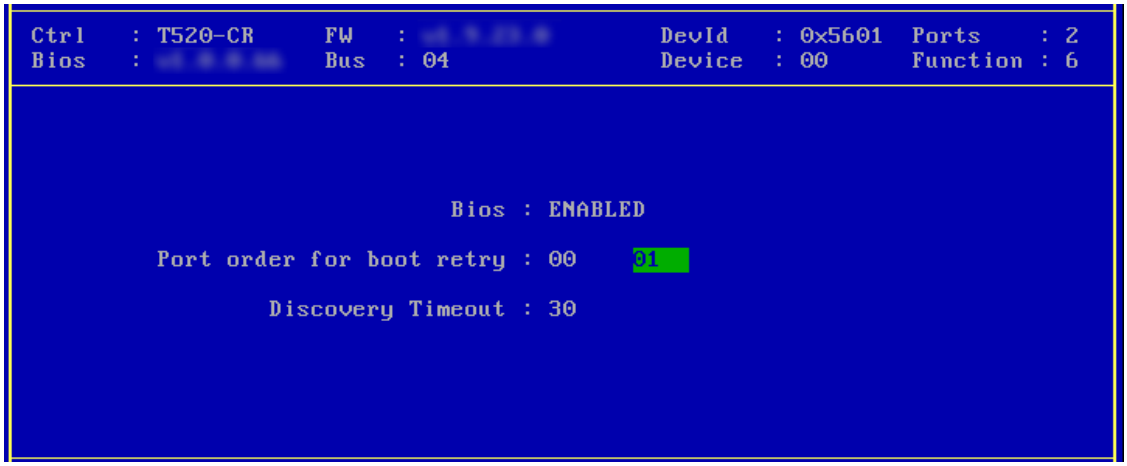
vi. Choose the first option, **Configure function parameters**, from the list of parameter type and hit [Enter].



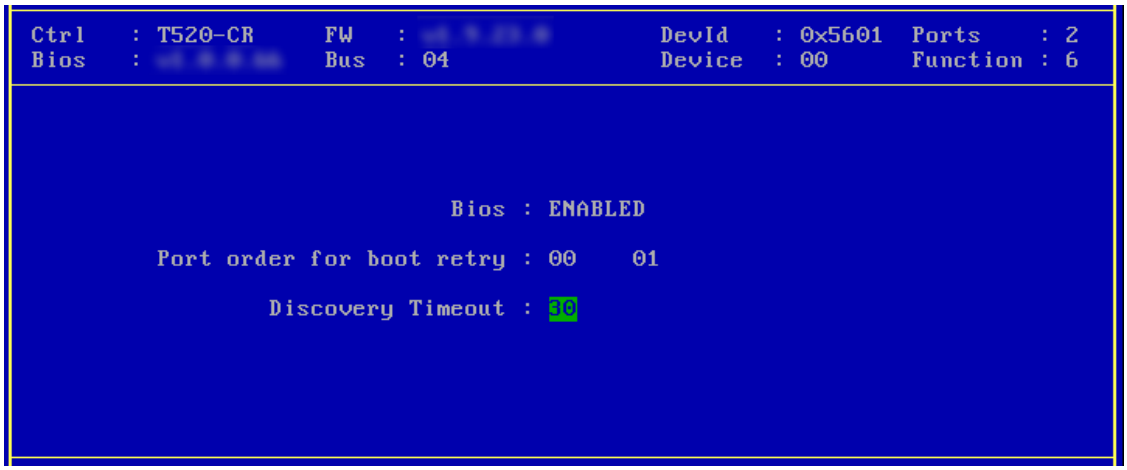
vii. Enable FCoE BIOS if not already enabled.



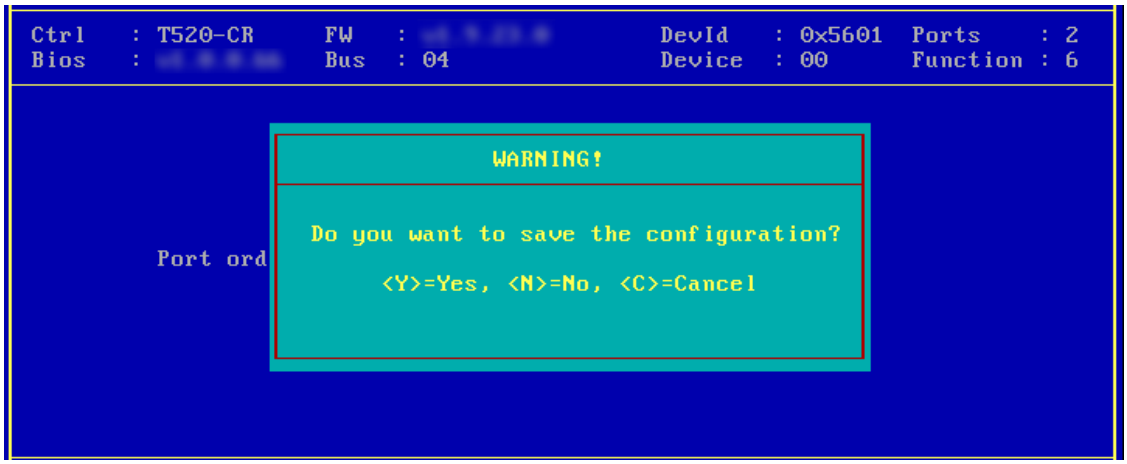
viii. Choose the order of the ports to discover FCoE targets.



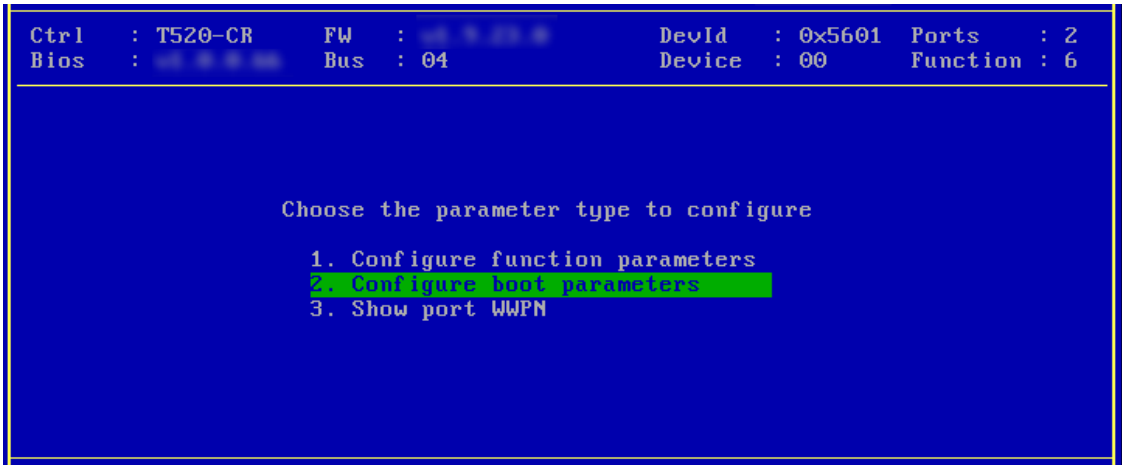
ix. Set discovery timeout to a suitable value. Recommended value is >= 30.



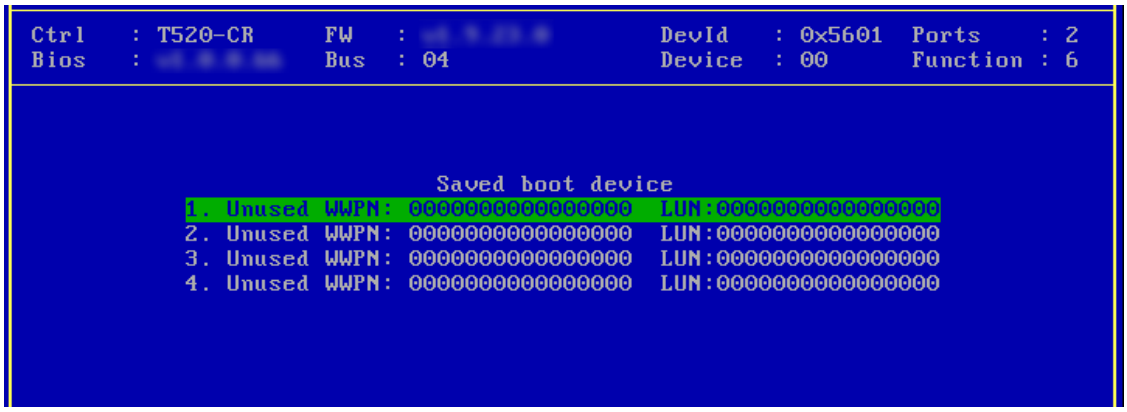
x. Hit [F10] or [Esc] and then [Y] to save the configuration.



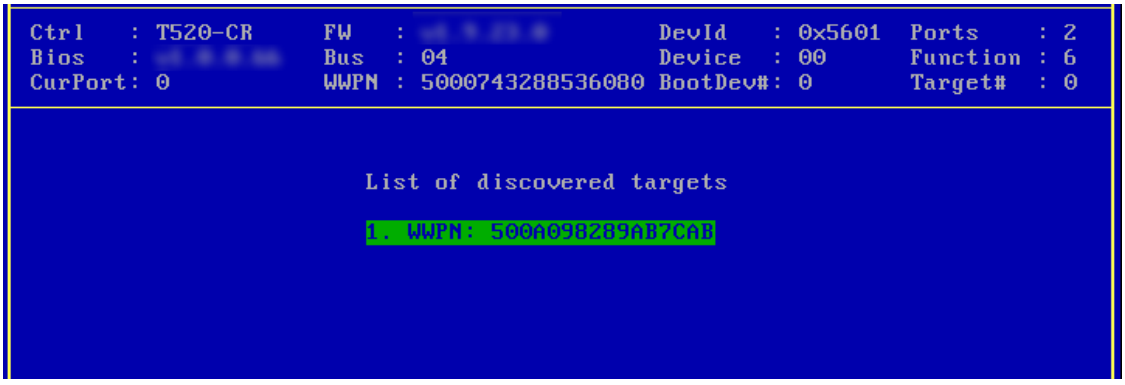
xi. Choose **Configure boot parameters**.



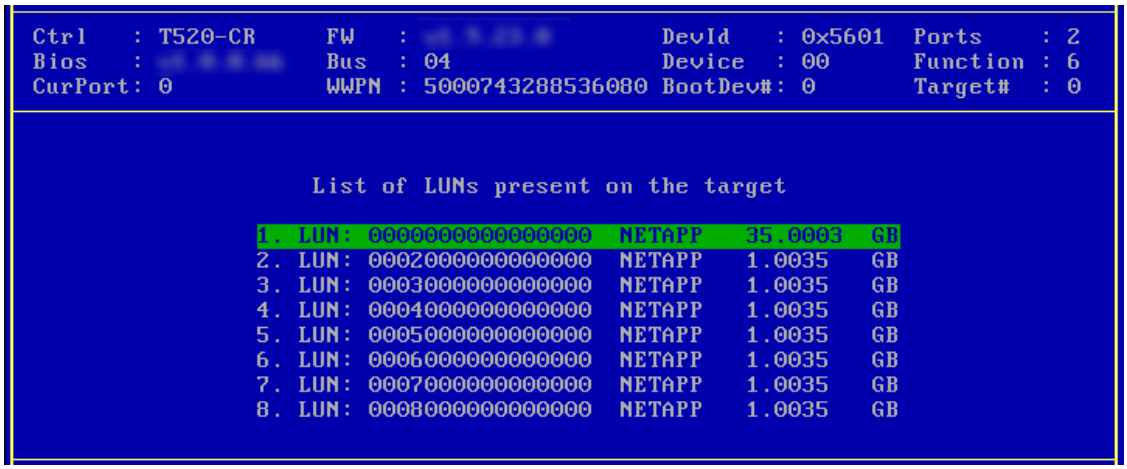
xii. Select the first boot device and hit [Enter] to discover FC/FCoE targets connected to the switch. Wait till all reachable targets are discovered.

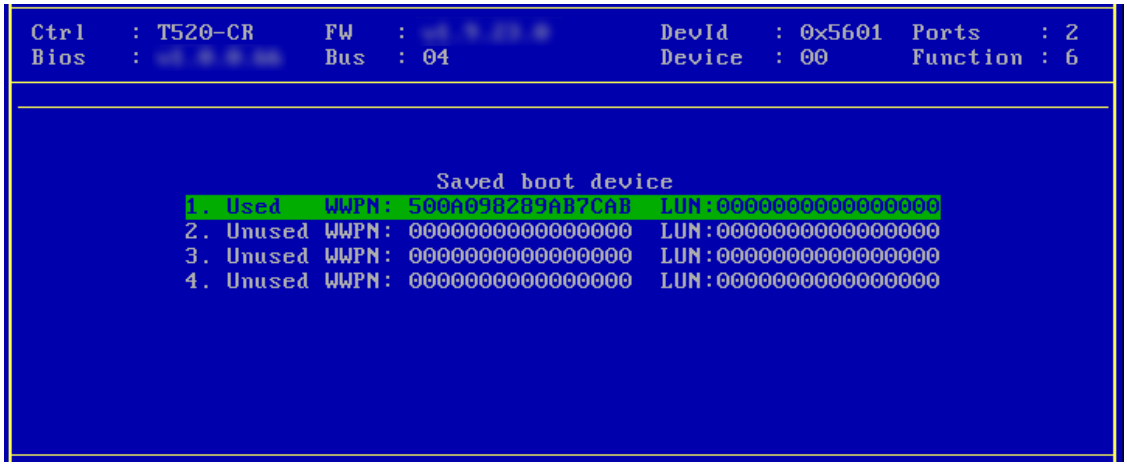


xiii. List of discovered targets will be displayed. Highlight a target using the arrow keys and hit [Enter] to select.

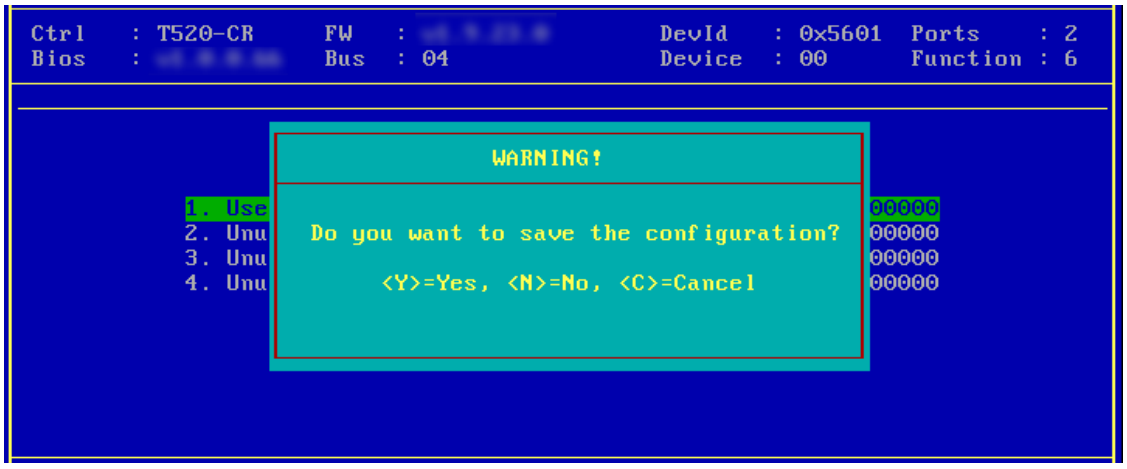


xiv. From the list of LUNs displayed for the selected target, choose one on which operating system has to be installed. Hit [Enter].





xv. Hit [F10] or [Esc] and then [Y] to save the configuration.

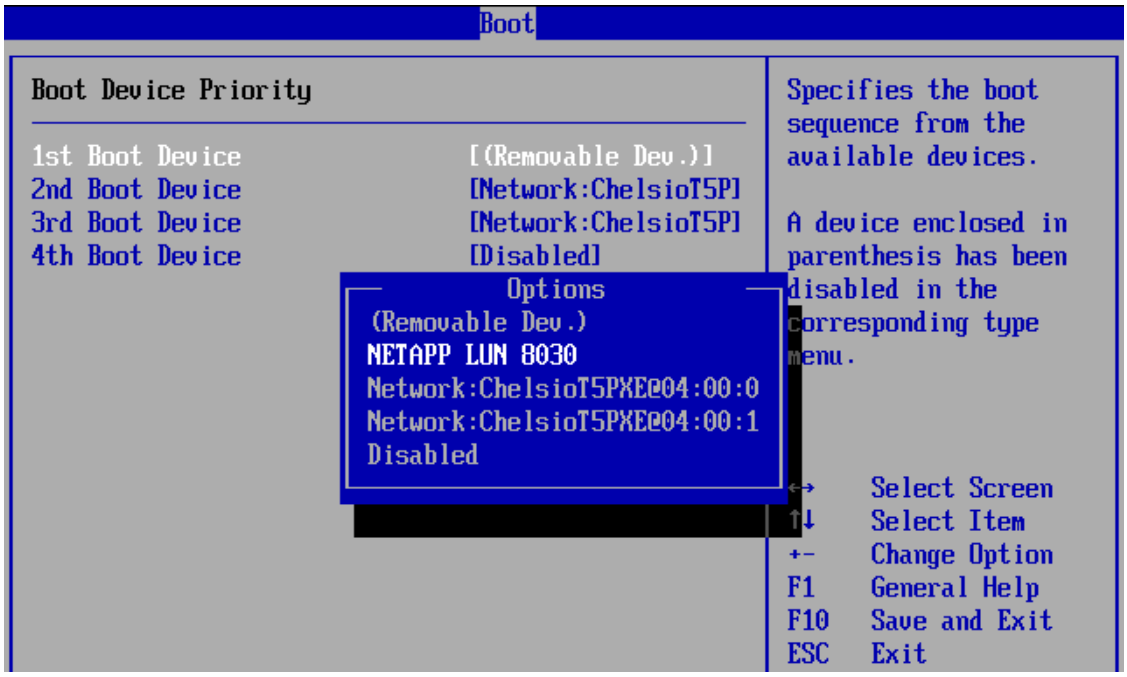


xvi. Reboot the machine.

xvii. During POST, allow the Chelsio Option ROM to discover FCoE targets.

```
Installing Chelsio T5 Storage FCoE BIOS
PCI BIOSv3.0 PCI FWv2.1 PnP BIOS: YES PMM Entry is passed by BIOS
Bringing up link on PCI:04:00:6 Port 0 ... Done
Discovering FCoE Target(s) on PCI:04:00:6 Port 0 ... Done
sd(1): T520-CR         PCI:04:00:6 P(0) WWPN:500A098289AB7CAB Lun(00)
       NETAPP  LUN             8030 35.0003 GB
Storage FCoE BIOS Installed Successfully!
```

xviii. Enter BIOS setup and choose FCoE disk discovered via Chelsio adapter as the first boot device.



xix. Reboot and boot from the FCoE disk or install the required OS using PXE.

6.2. uEFI FCoE Boot

Important

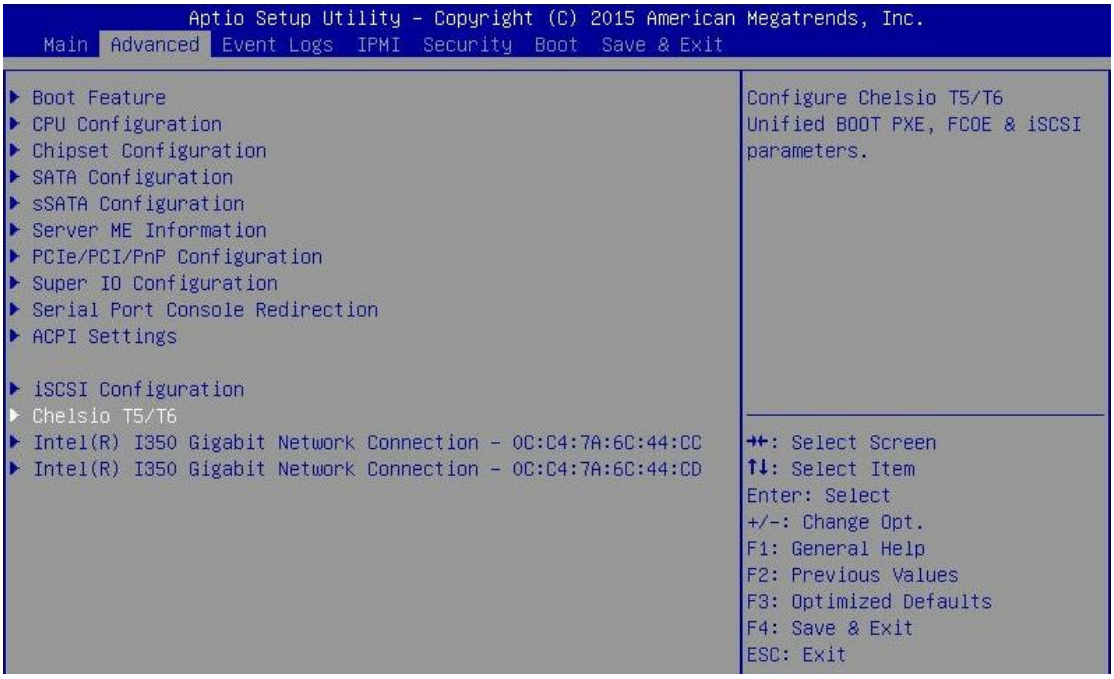
- *Only uEFI v2.3.1, v2.4 and v2.5 supported.*
- *Any other uEFI version is NOT SUPPORTED and may render your system unusable.*

6.2.1. HII

This section describes the method to configure and use Chelsio uEFI FCoE interfaces using HII.

i. Reboot the system and go into BIOS setup.

ii. Select **Chelsio T5/T6** and press [Enter]

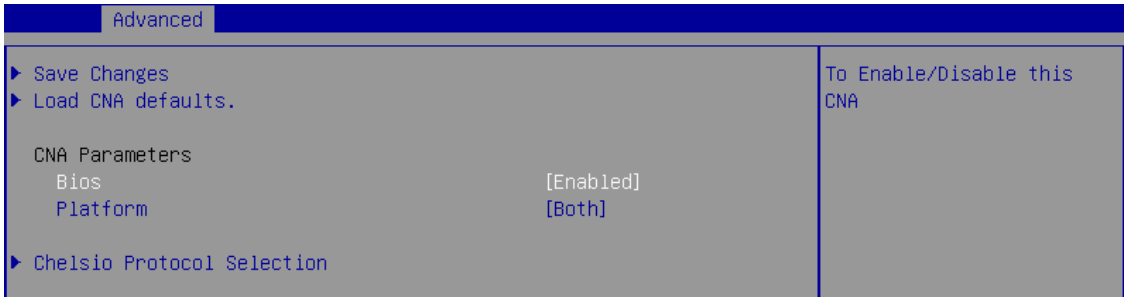


Note *Please ensure that Chelsio uEFI driver is loaded correctly as mentioned in Loading uEFI driver section.*

iii. Select the Chelsio adapter to be configured and press [Enter].

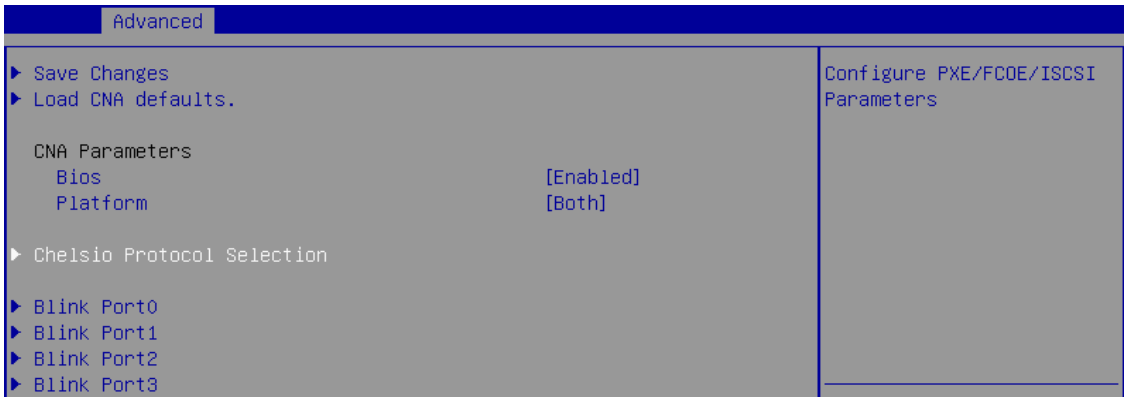
iv. Select **Configuration Utility** and press [Enter].

v. Enable adapter BIOS if not already enabled.



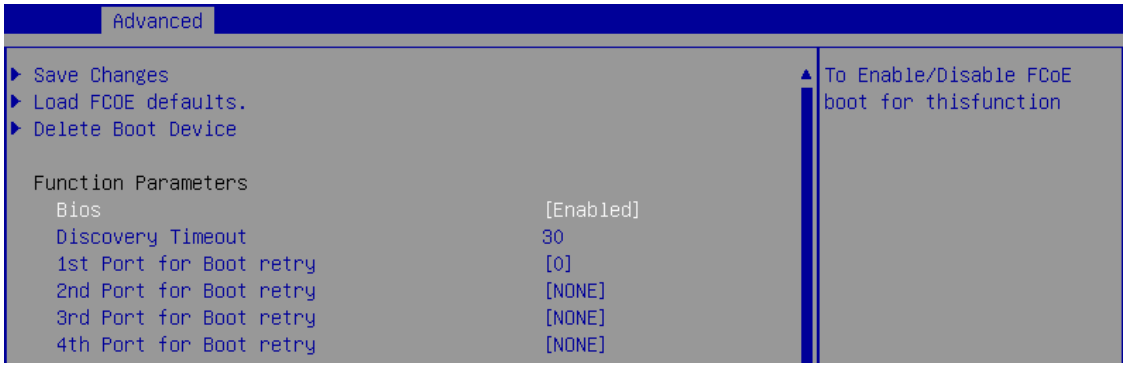
Note *It is highly recommended that you use the **Save Changes** option every time a parameter/option is changed.*

vi. Select **Chelsio Protocol Selection** and press [Enter].

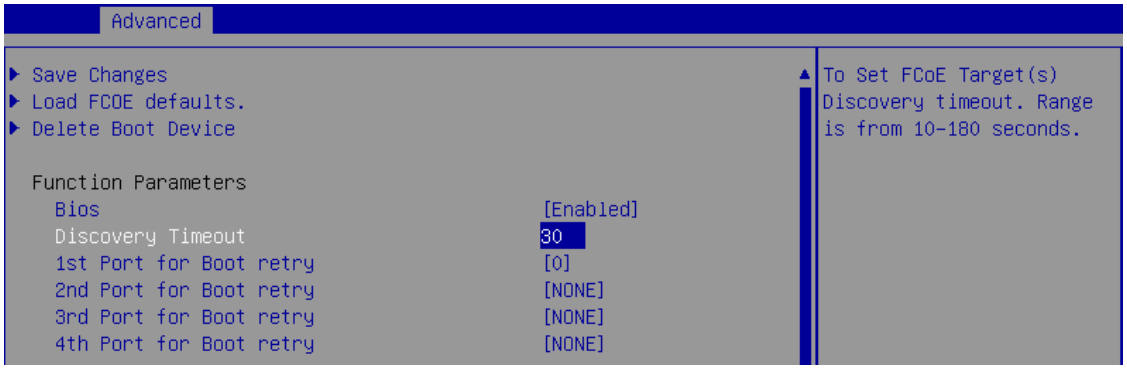


vii. Select **FCoE** and press [Enter].

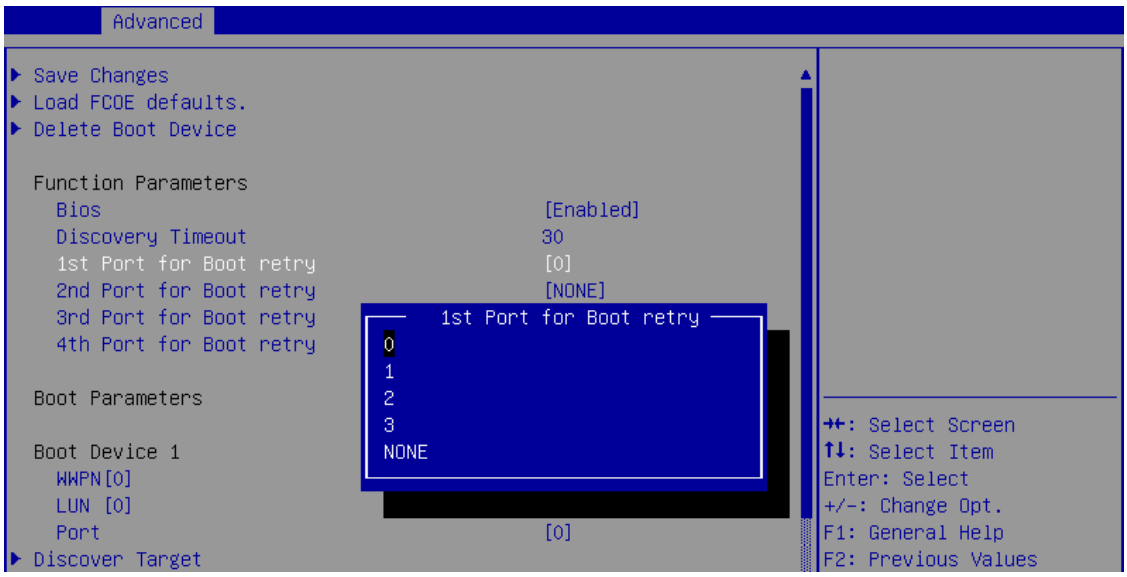
viii. Under **Function Parameters**, enable FCoE BIOS, if not already enabled.



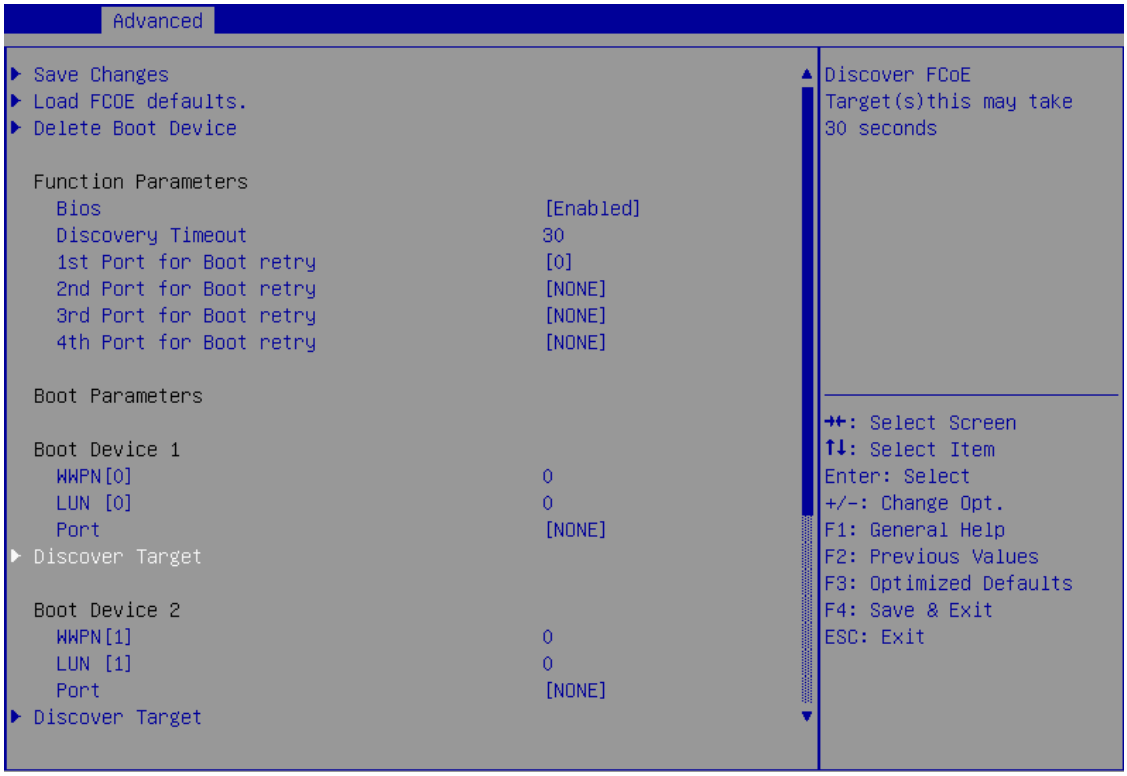
ix. Set discovery timeout to a suitable value. Recommended value is >= 30



x. Choose the order of the ports to discover FCoE targets.

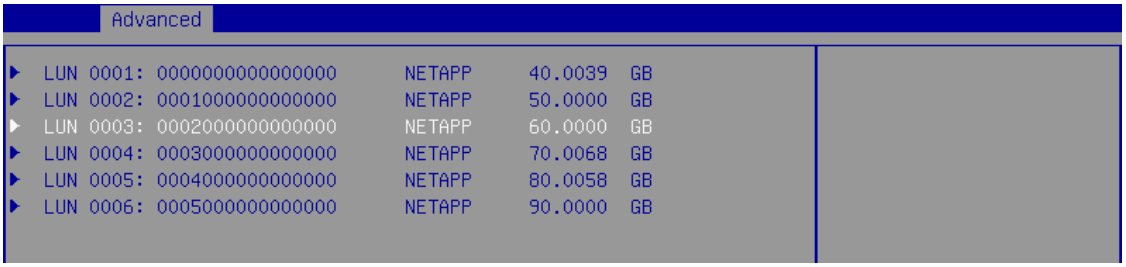


xi. Under the first boot device, select **Discover Target** and press [Enter] to discover FC/FCoE targets connected to the switch. Wait till all reachable targets are discovered.

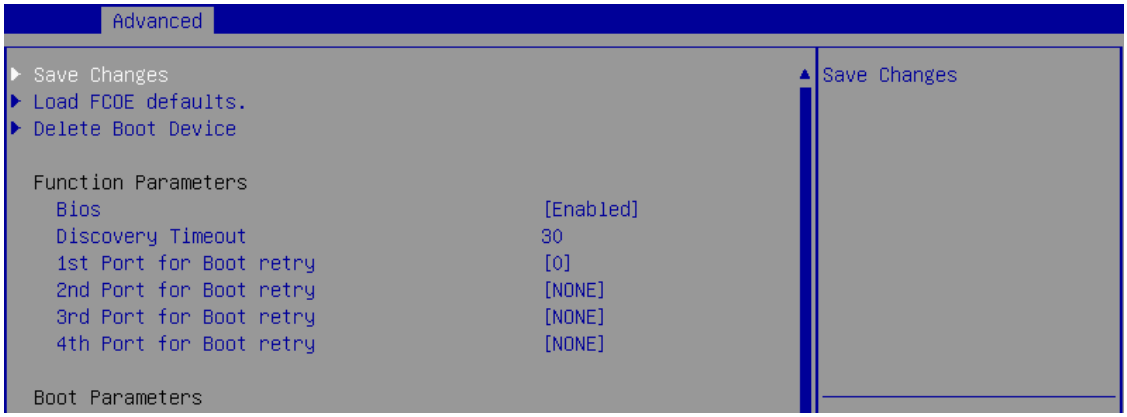


xii. List of discovered targets will be displayed. Highlight a target to select it and hit [Enter].

xiii. List of LUNs for the selected target will be displayed. Highlight a LUN to select it and hit [Enter].

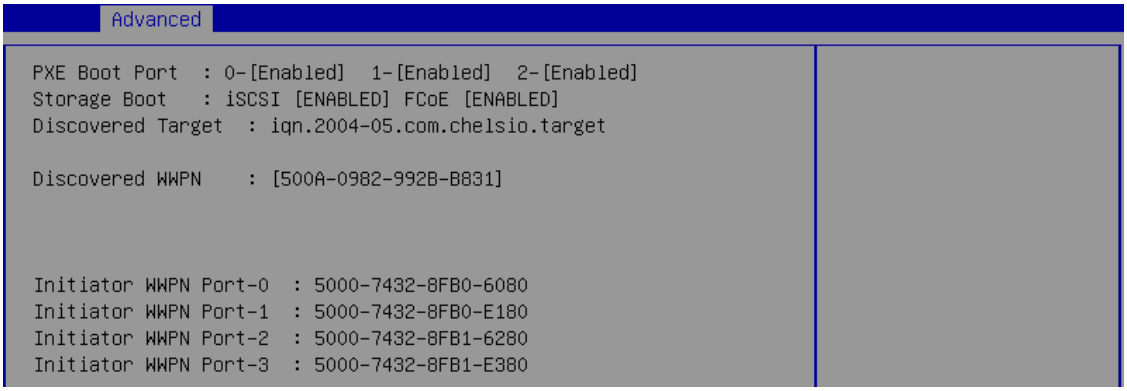


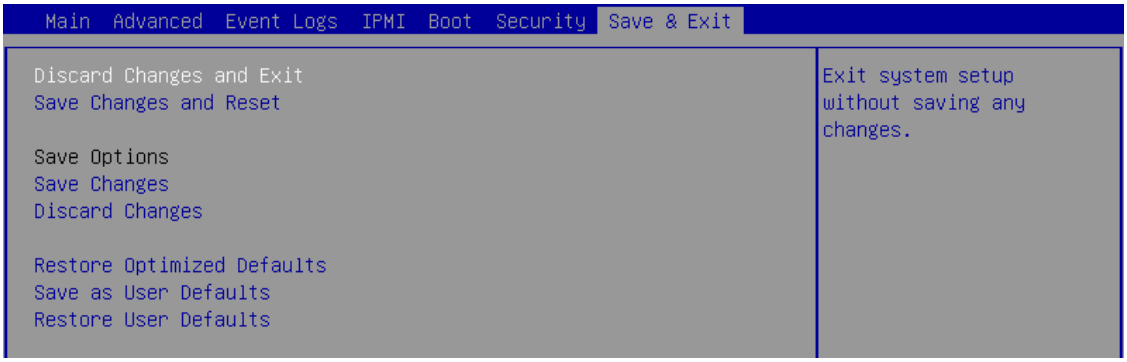
xiv. Select **Save Changes** and press [Enter].



xv. Reboot the system for changes to take effect.

xvi. The discovered LUN should appear in the **Boot Configuration** section and system BIOS section.





xvii. Select the LUN as the first boot device and exit from BIOS.

xviii. Either boot from the LUN or install the required OS.

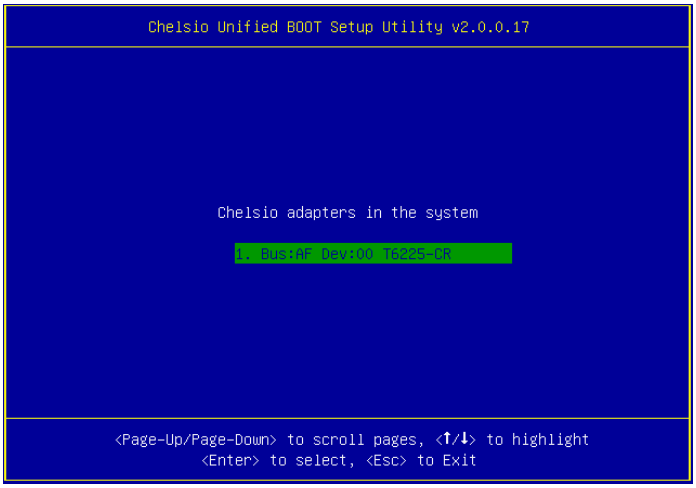
6.2.2. drvcfg

This section describes the method to configure and use Chelsio uEFI FCoE interfaces using drvcfg.

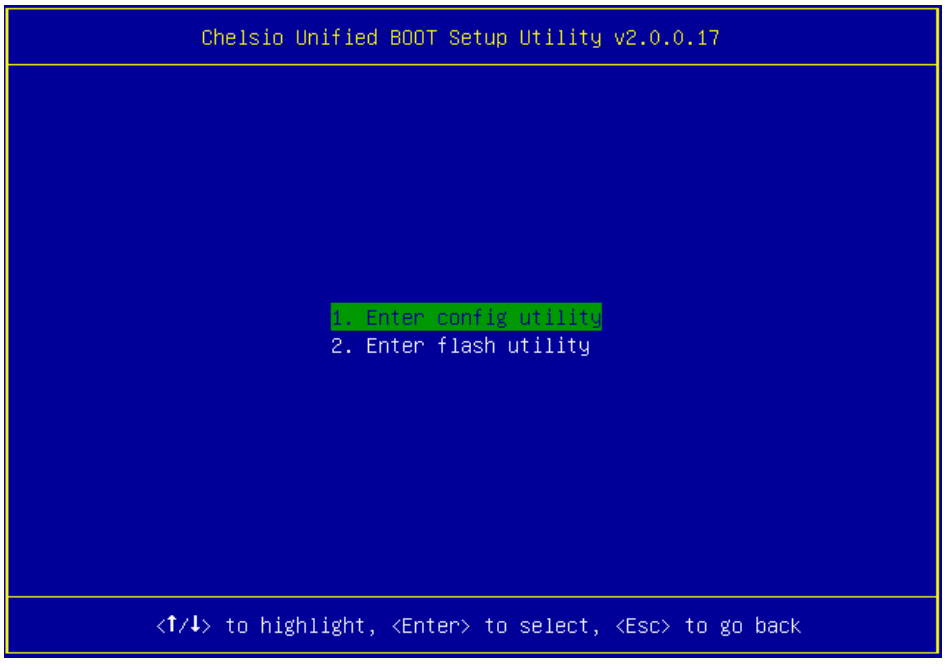
i. Boot the system into EFI shell.
ii. Run the following command to launch the configuration utility.

```
fs0:\> drvcfg -s_
```

iii. Choose the Chelsio adapter on which needs to be configured.



iv. Highlight **Enter config utility** and press [Enter].



v. Further configuration steps are similar from step (iv) of Legacy FCoE Boot section.

7. iSCSI Boot Process

Before proceeding, please ensure that the Chelsio CNA has been flashed with the provided firmware and option ROM (See Flashing firmware and option ROM).

7.1. Legacy iSCSI Boot

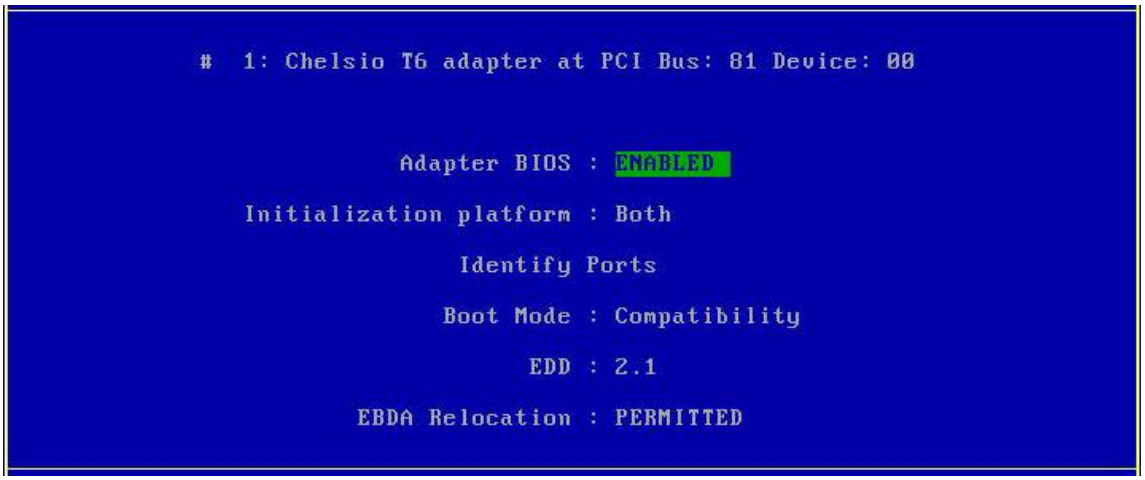
i. Reboot the system.

ii. Press [Alt+C] when the message to configure Chelsio adapters appears on the screen.

iii. The configuration utility will appear as below:

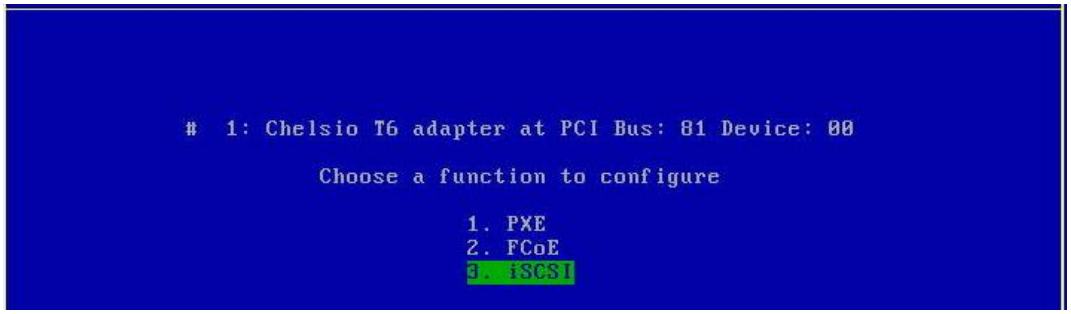
Choose the adapter on which you flashed the option ROM image. Hit [Enter].

iv. Enable the adapter BIOS if not already enabled. Hit [Enter].

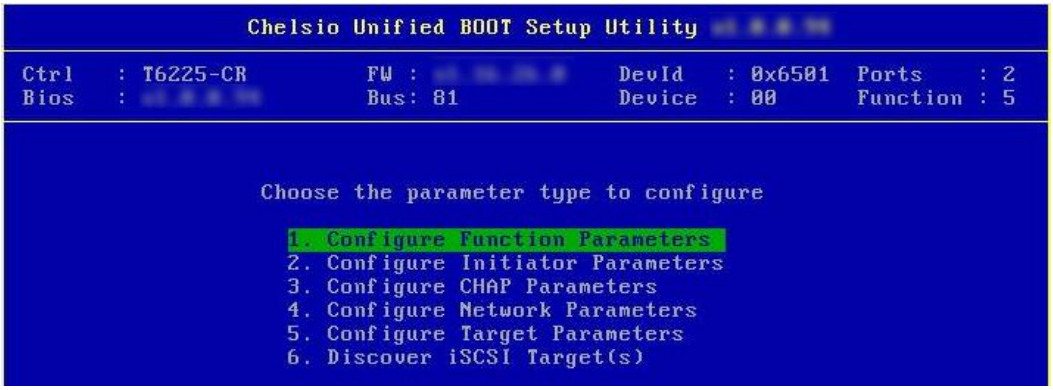


Note *Use the default values for Boot Mode, EDD and EBDA Relocation parameters, unless instructed otherwise.*

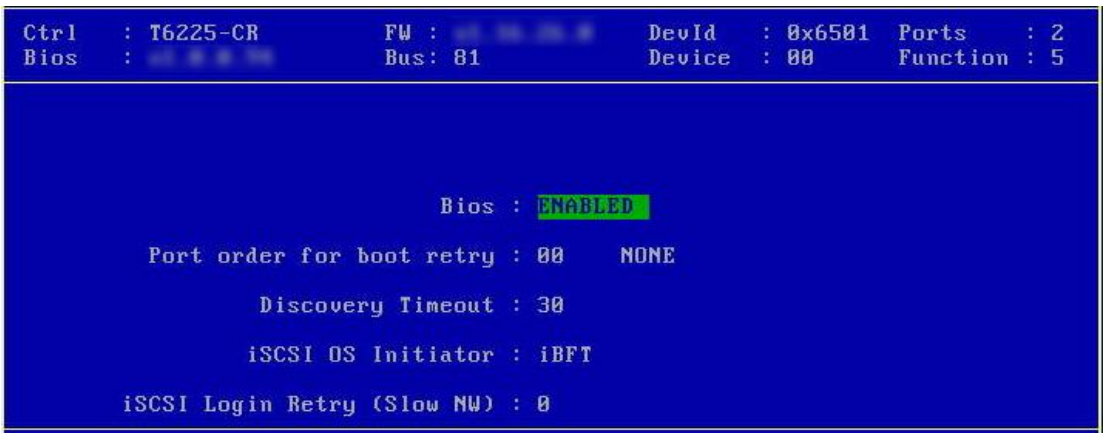
v. Choose *iSCSI* from the list to configure and hit [Enter].



vi. Choose the first option, **Configure Function Parameters**, from the list of parameter type and hit [Enter].

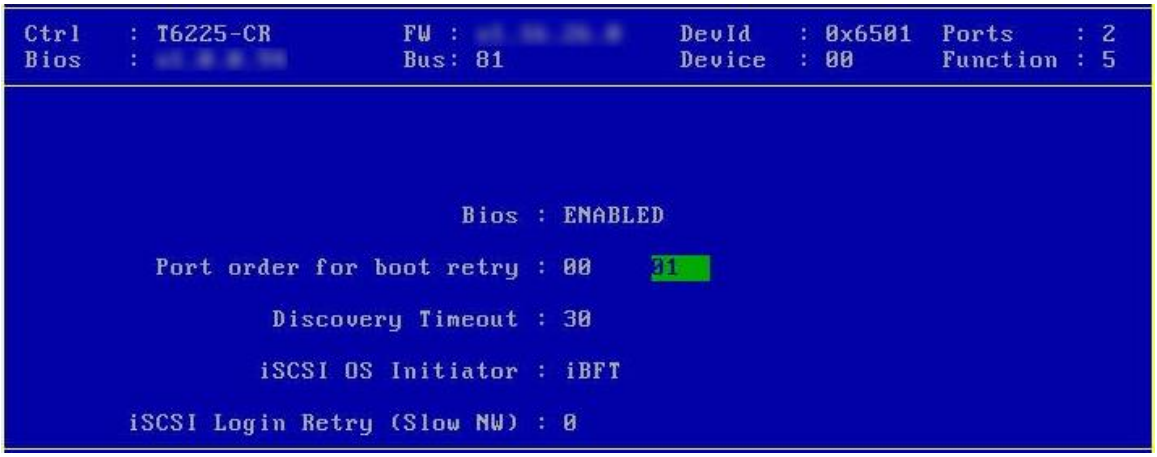


vii. Enable iSCSI BIOS if not already enabled. iBFT (iSCSI Boot Firmware Table) will be selected by default. Only iBFT is supported in Linux.

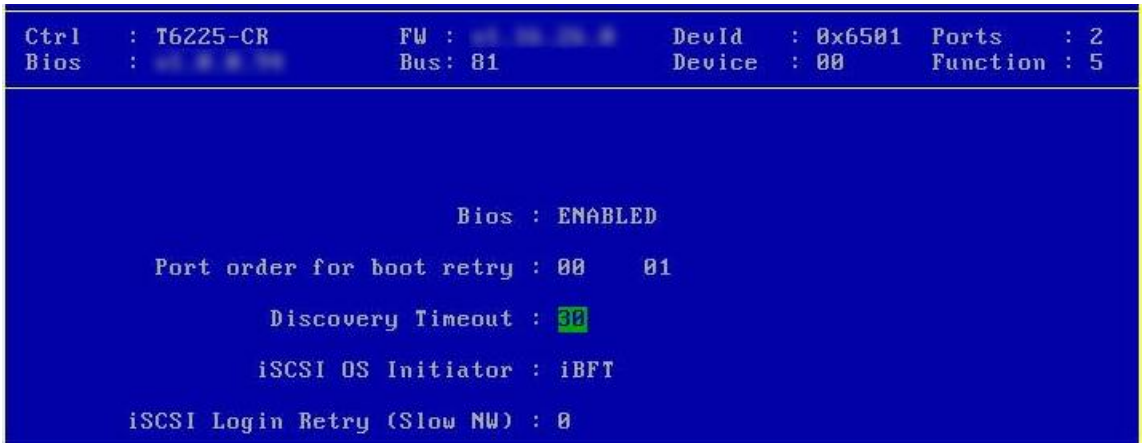


You can also configure the number of iSCSI login attempts (retries) in case the network is unreachable or slow

viii. Choose the order of the ports to discover iSCSI targets.



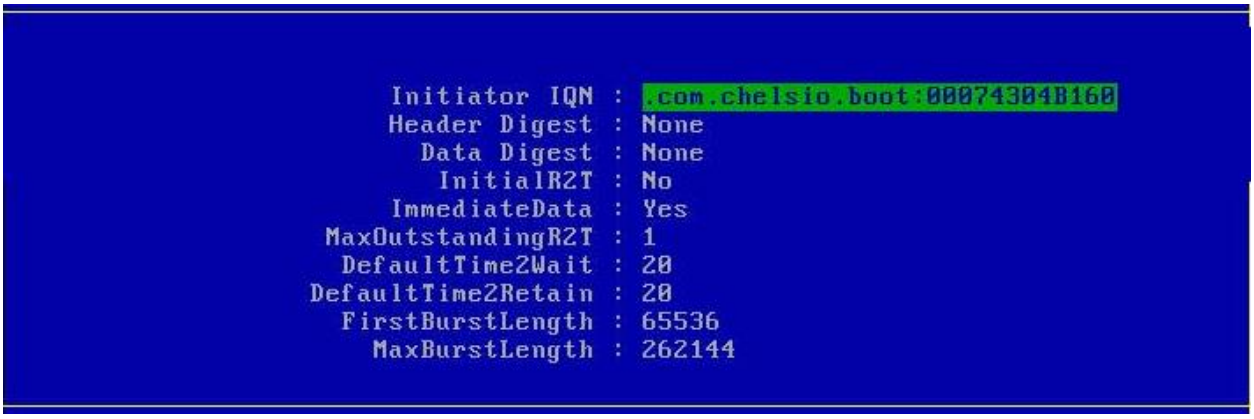
ix. Set discovery timeout to a suitable value. Recommended value is >= 30.



x. Hit [Esc] and then [Y] to save the configuration.

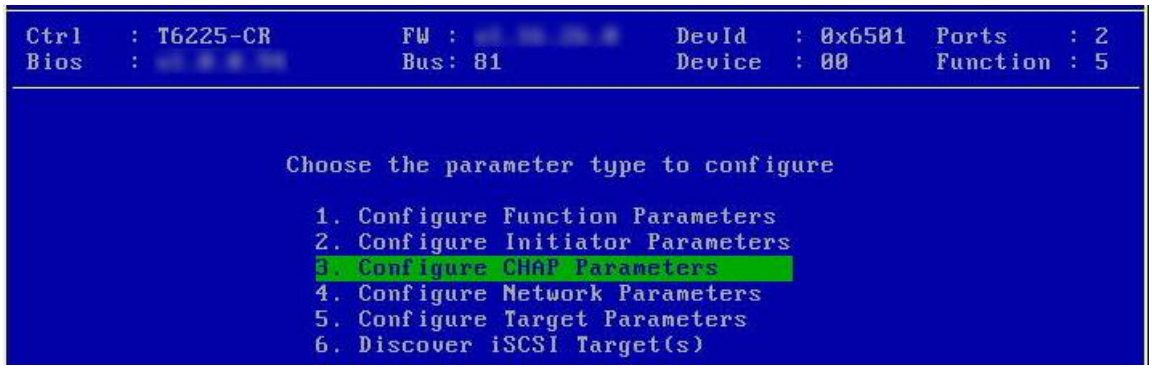
xi. Go back and choose **Configure Initiator Parameters** to configure initiator related properties.

xii. Initiator properties like IQN, Header Digest, Data Digest, etc. will be displayed. Change the values appropriately or continue with the default values. Hit [F10] to save.



Note *MaxBurstLength and FirstBurstLength range from 512 to 16777215 bytes.*

xiii. CHAP authentication is disabled by default. To enable and configure, go back and choose **Configure CHAP Parameters**



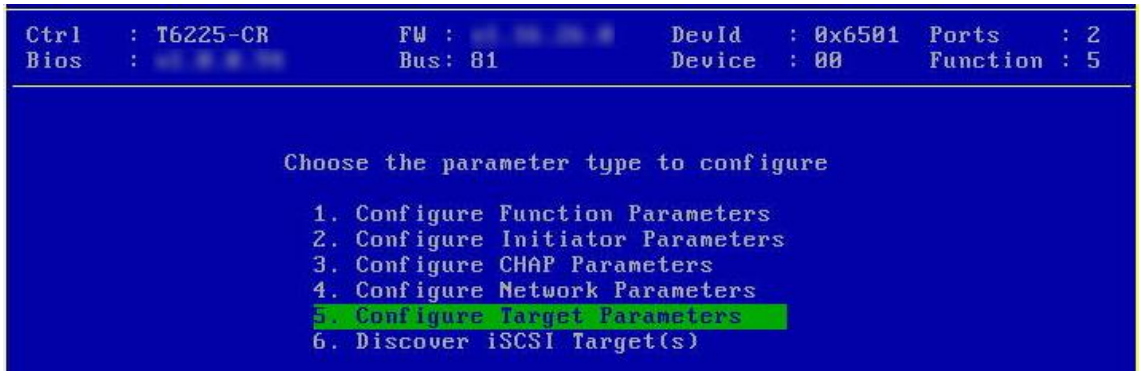
xiv. Enable CHAP authentication by selecting ONE-WAY or MUTUAL in the **CHAP Policy** field. Next, choose the CHAP method. Finally, provide Initiator and Target CHAP credentials as per the authentication method selected. Hit [F10] to save.

xv. Go back and choose **Configure Network Parameters** to configure iSCSI Network related properties.

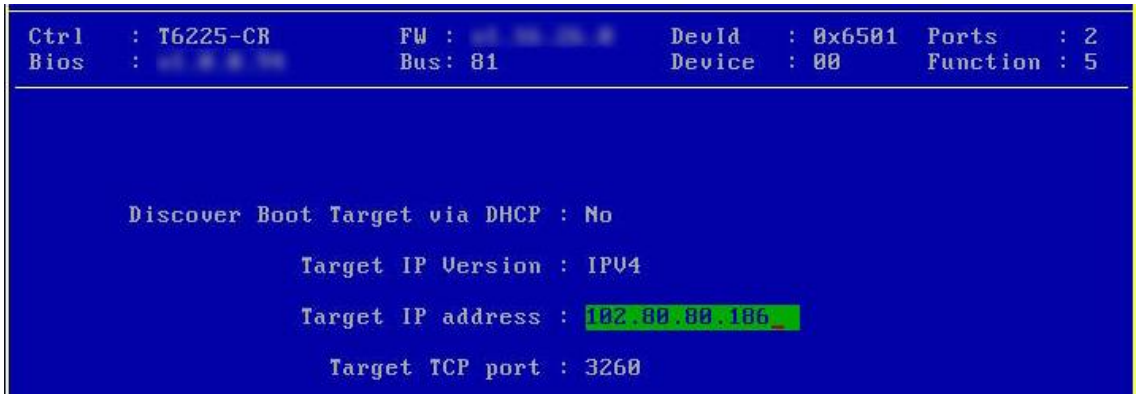
xvi. Select the port using which you want to connect to the target. Hit [Enter].

xvii. Select *Yes* in the **Enable DHCP** field to configure port using DHCP or *No* to manually configure the port. Hit [F10] to save.

xviii. Go back and choose **Configure Target Parameters** to configure iSCSI target related properties.



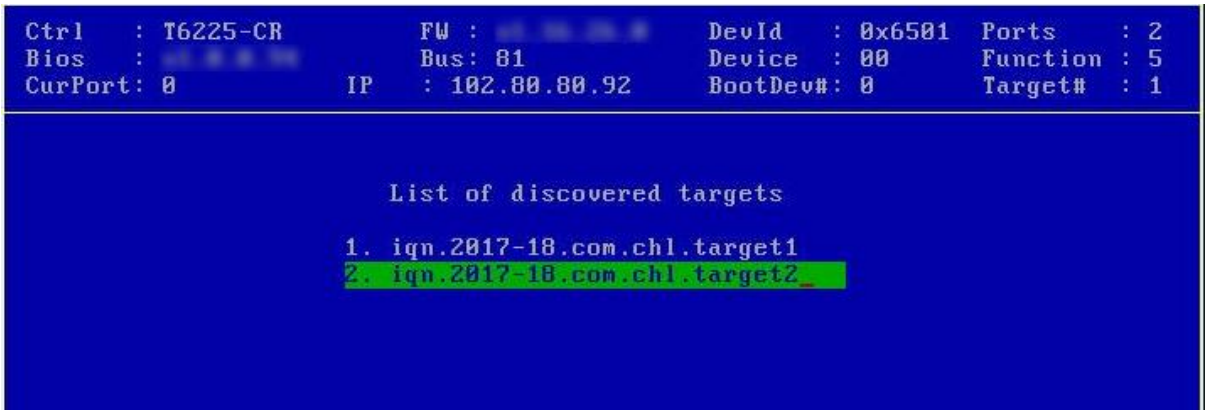
xix. If you want to discover target using DHCP, select *Yes* in the **Discover Boot Target via DHCP** field. To discover target via static IP, select *No* and provide the target IP and Hit [F10] to save. The default TCP port selected is 3260.



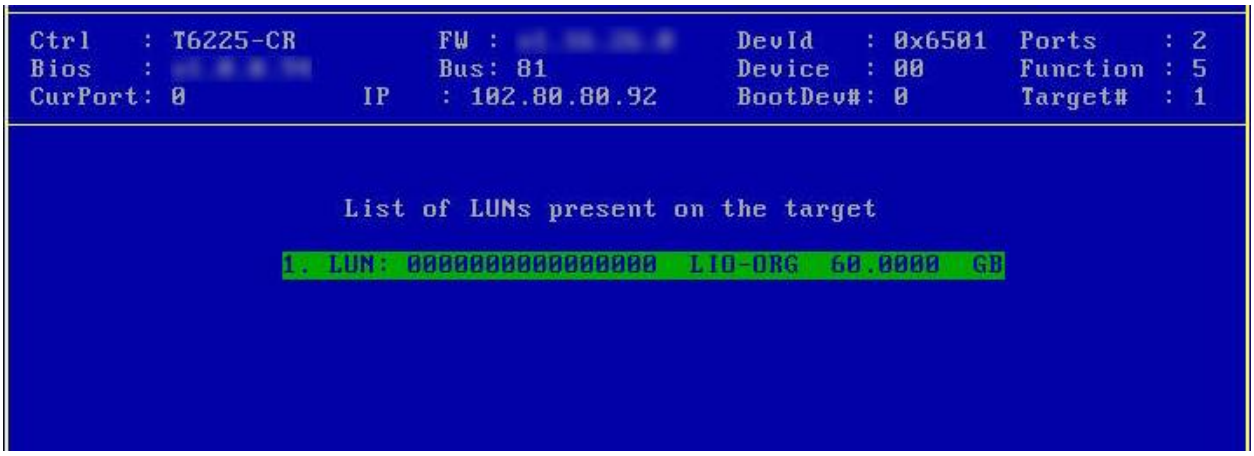
xx. Go back and choose **Discover iSCSI Target (s)** to connect to a target.

xxi. Select the portal group on which iSCSI service is provided by the target.

xxii. A list of available targets will be displayed. Select the target you wish to connect to and hit [Enter].



xxiii. A list of LUNs configured on the selected target will be displayed. Select the LUN you wish to connect to and hit [Enter].

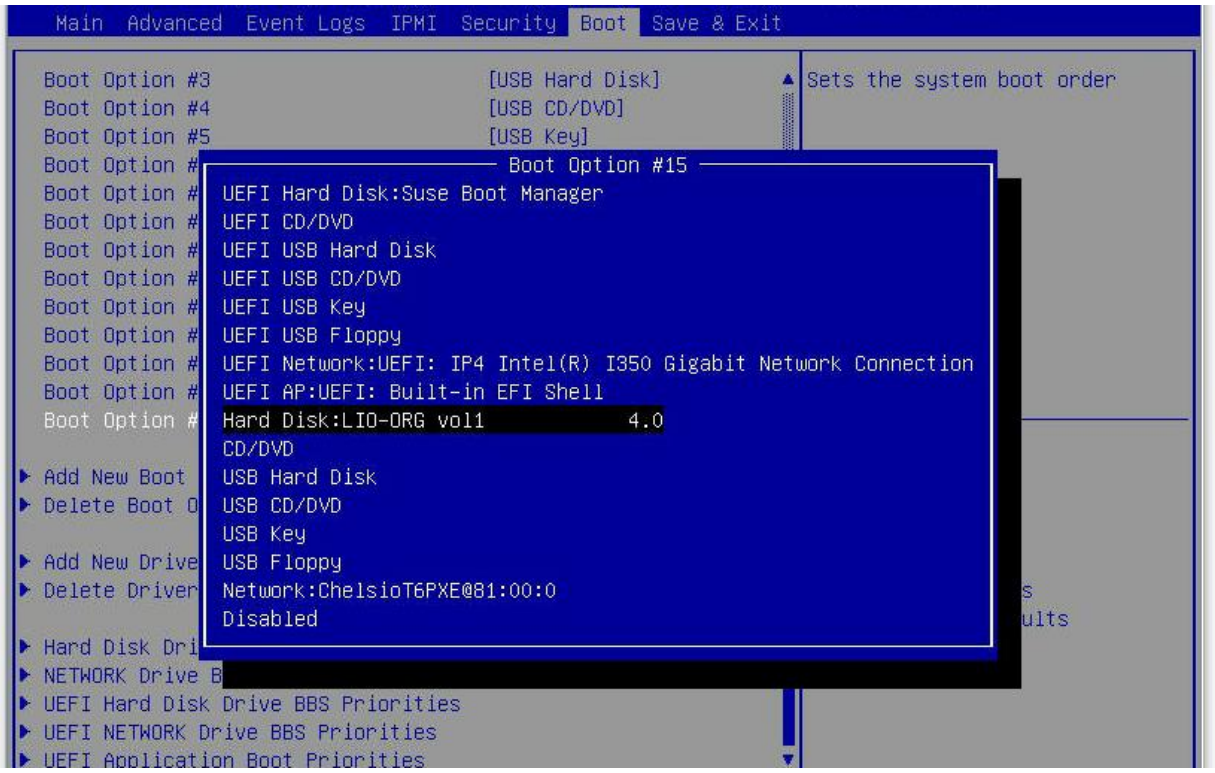


xxiv. Hit [Esc] and then [Y] to save the configuration.

xxv. Reboot the machine.

xxvi. During POST, allow the Chelsio Option ROM to discover iSCSI targets.

xxvii. Enter BIOS setup and choose iSCSI target LUN discovered via Chelsio adapter as the first boot device.



xxviii. Reboot and boot from the iSCSI Target LUN or install the required OS using PXE.

7.2. uEFI iSCSI Boot

Important

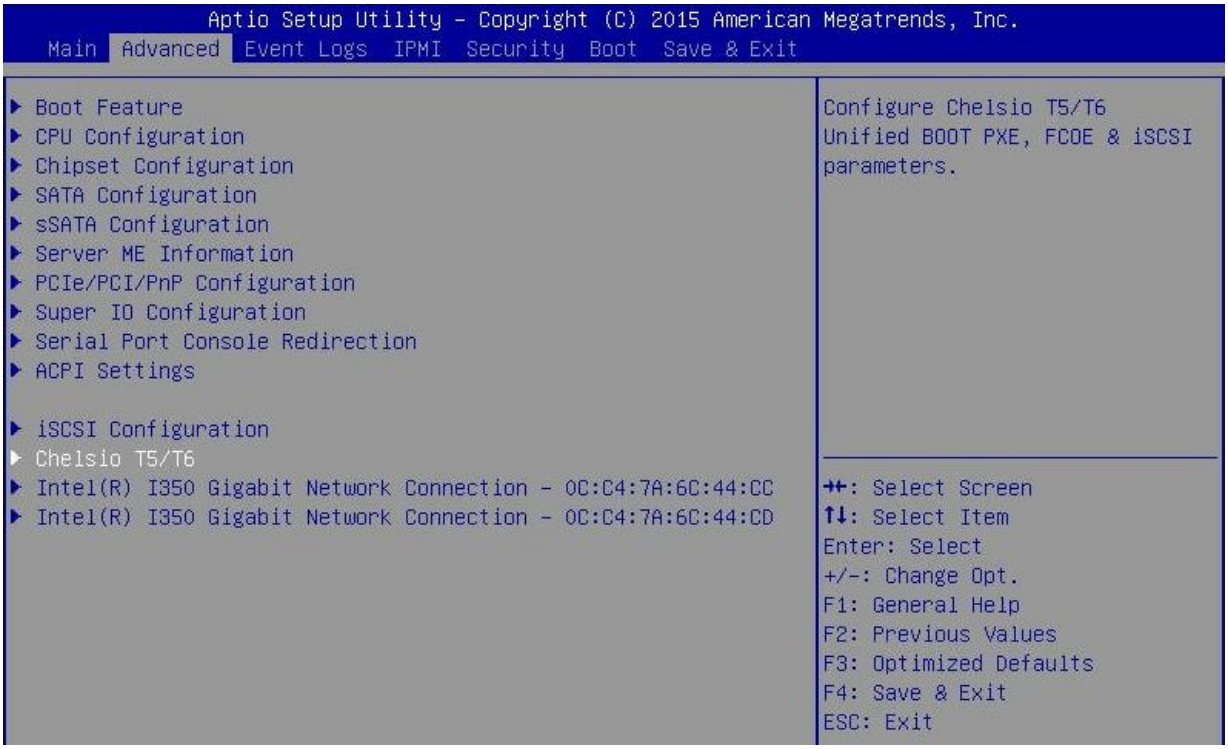
- *Only uEFI v2.3.1, v2.4 and v2.5 supported.*
- *Any other uEFI version is NOT SUPPORTED and may render your system unusable.*

7.2.1. HII

This section describes the method to configure and use Chelsio uEFI iSCSI interfaces using HII.

i. Reboot the system and go into BIOS setup.

ii. Select **Chelsio T5/T6** and press [Enter]

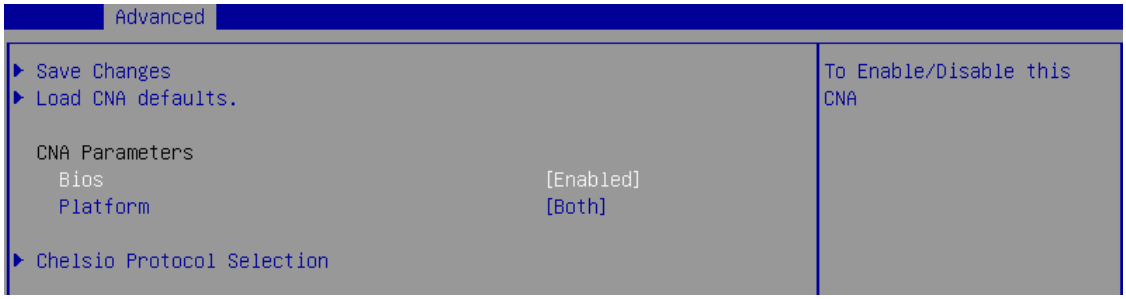


Note *Please ensure that Chelsio uEFI driver is loaded correctly as mentioned in Loading uEFI driver section.*

iii. Select the Chelsio adapter to be configured and press [Enter].

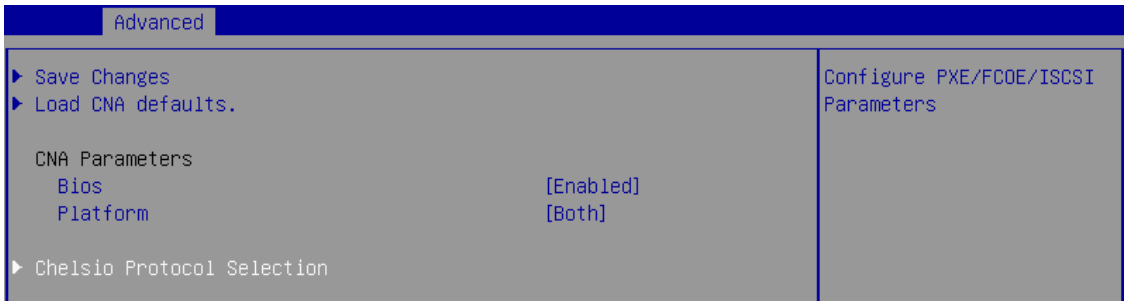
iv. Select **Configuration Utility** and press [Enter].

v. Enable adapter BIOS if not already enabled.

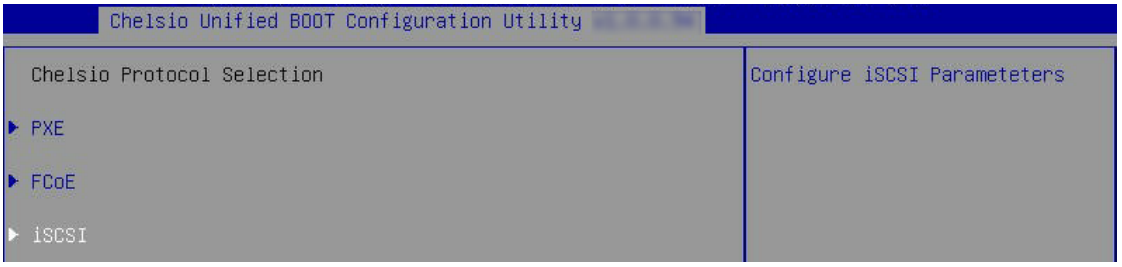


Note *It is highly recommended that you use the **Save Changes** option every time a parameter/option is changed.*

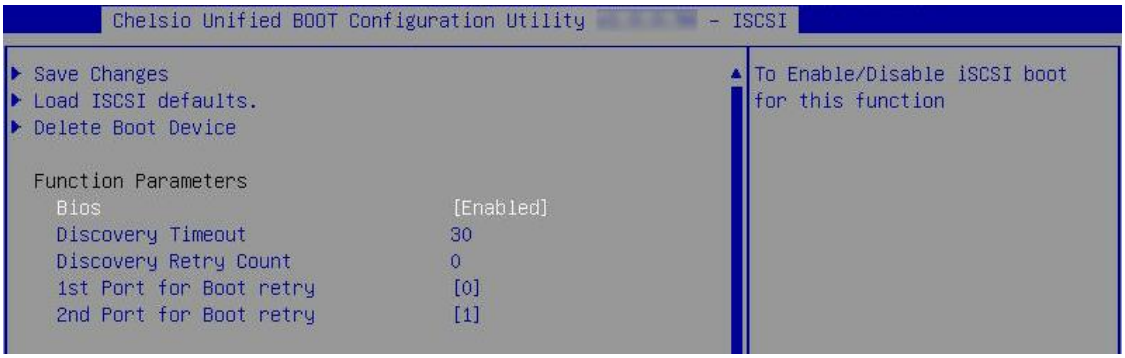
vi. Select **Chelsio Protocol Selection** and press [Enter].



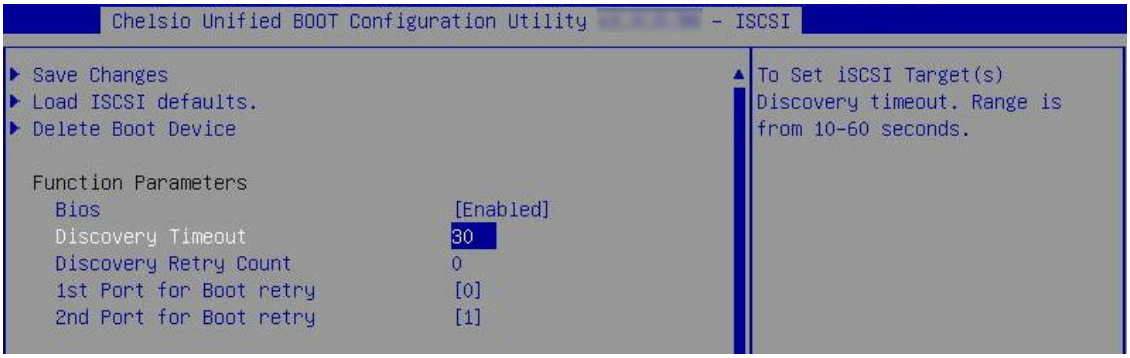
vii. Select **iSCSI** and press [Enter].



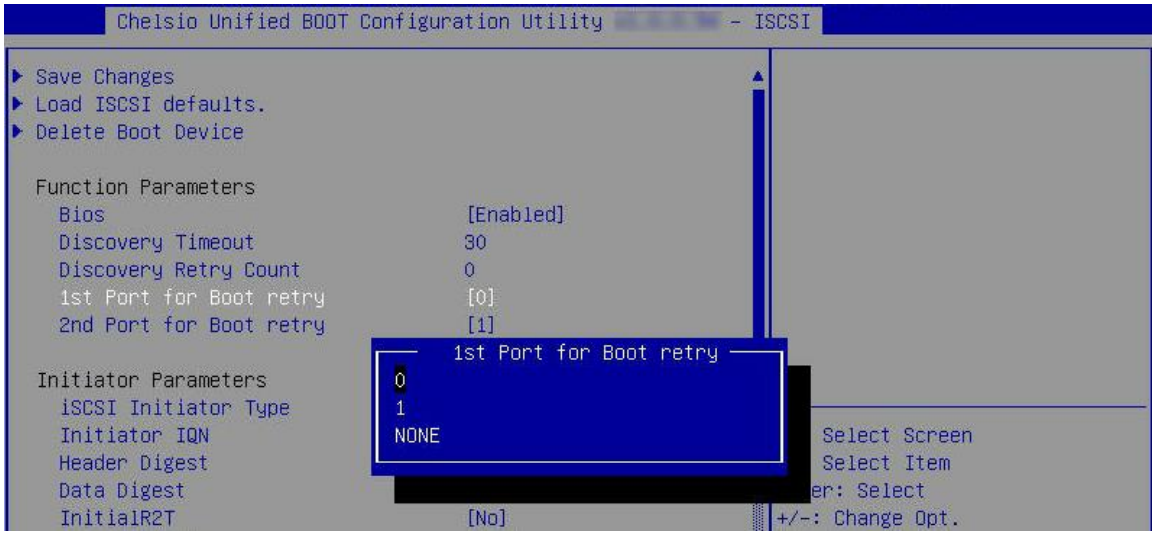
viii. Under **Function Parameters**, enable iSCSI BIOS, if not already enabled.



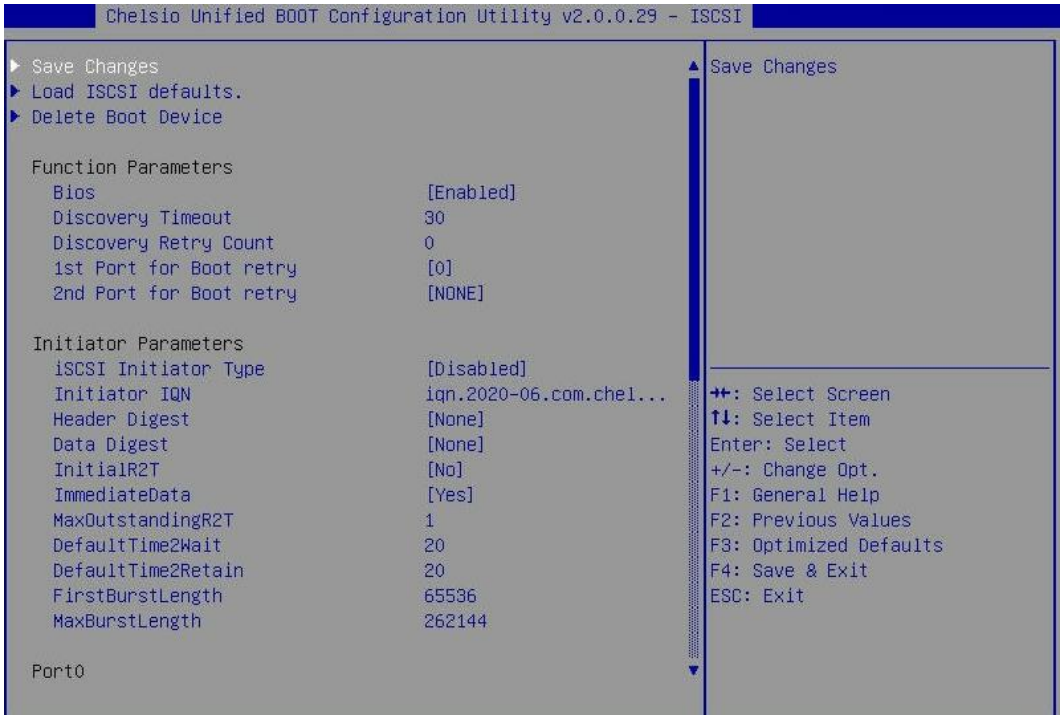
ix. Set discovery timeout to a suitable value. Recommended value is >= 30



x. Choose the order of the ports to discover iSCSI targets.

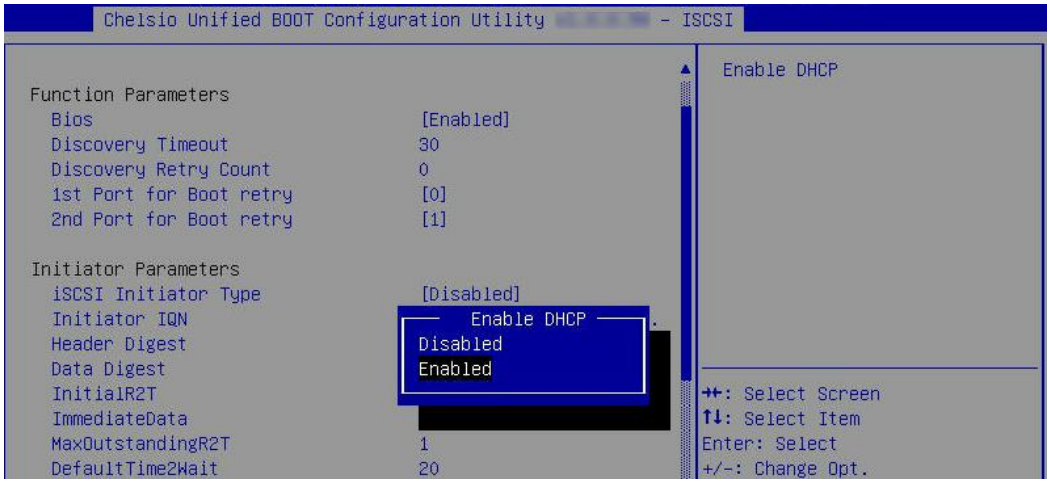


xi. Under **Initiator Parameters**, iSCSI Initiaitor properties like IQN, Header Digest, Data Digest, etc will be displayed. Change the values appropriately or continue with the default values.

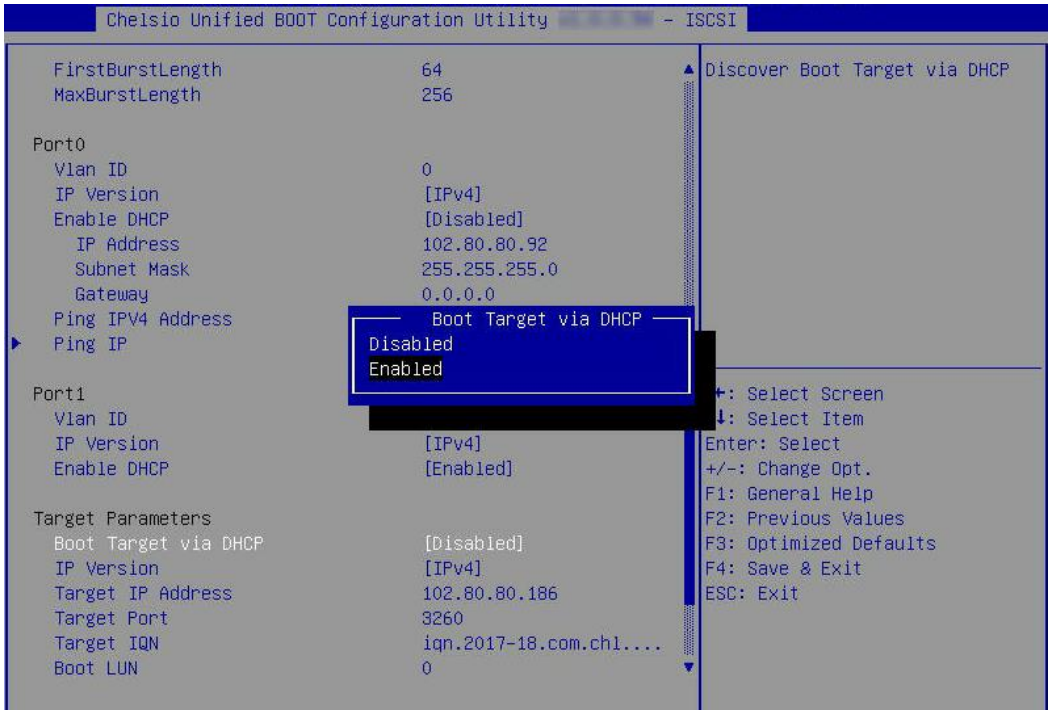


Note *MaxBurstLength and FirstBurstLength range from 512 to 16777215 bytes.*

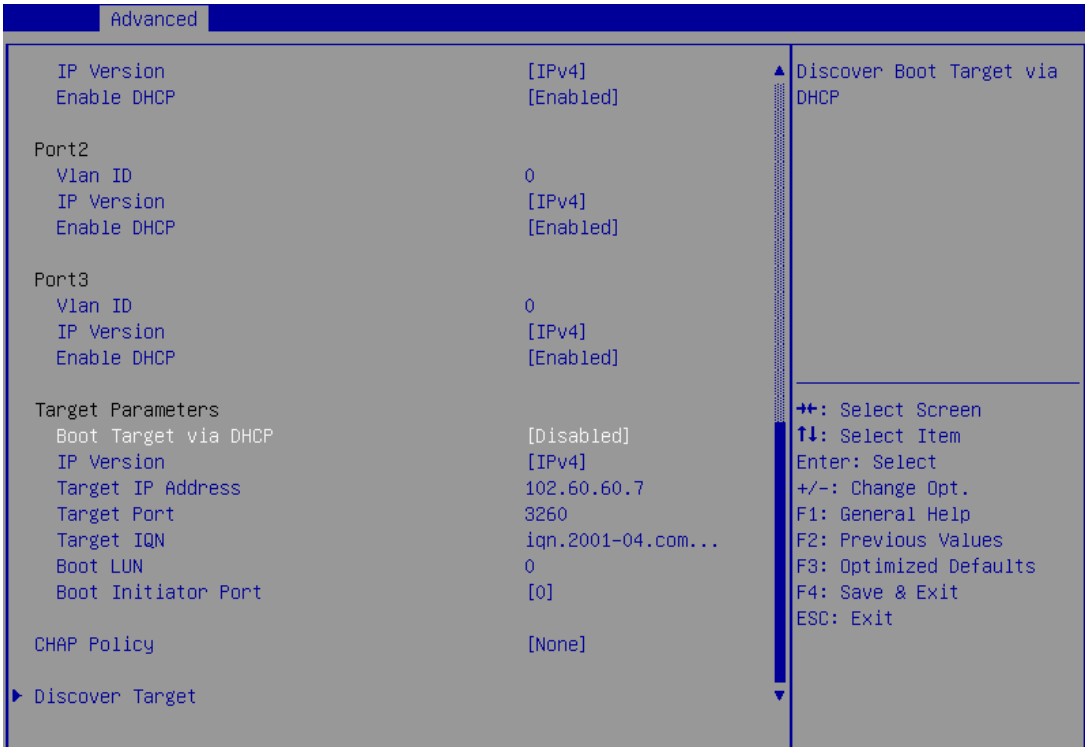
xii. Under the first port, select **Enable DHCP** field, hit [Enter] and select **Enabled**. This will configure port using DHCP. Select **Disabled** to manually configure the port.



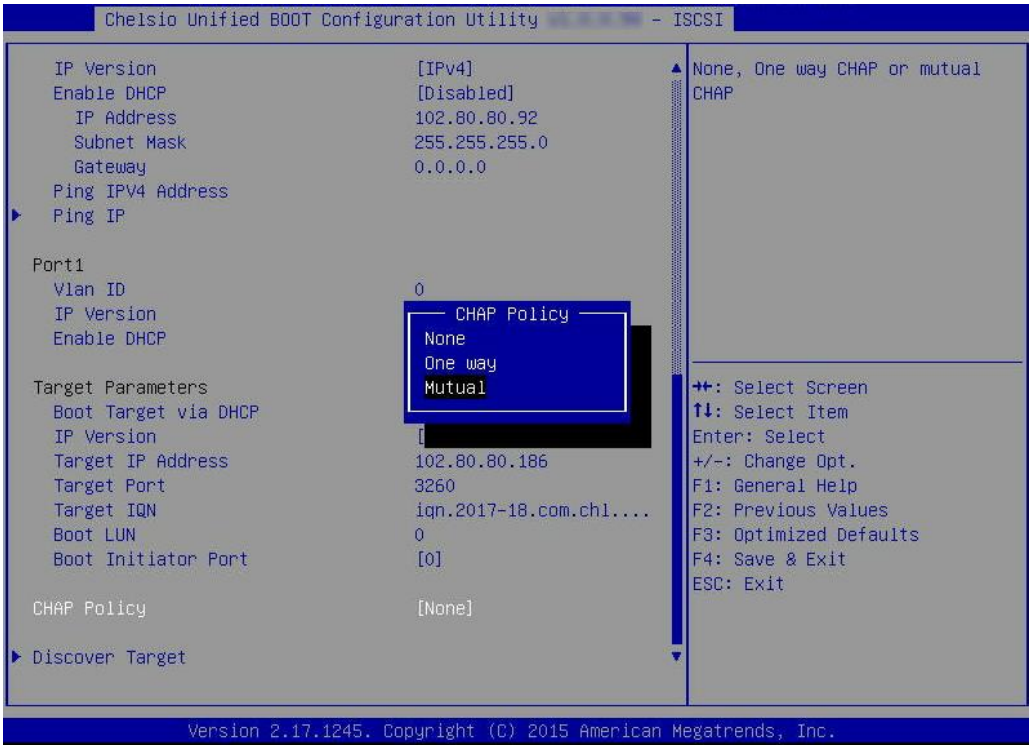
xiii. Under **Target Parameters**, select **Enabled** for the **Boot Target via DHCP** parameter to discover target using DHCP.



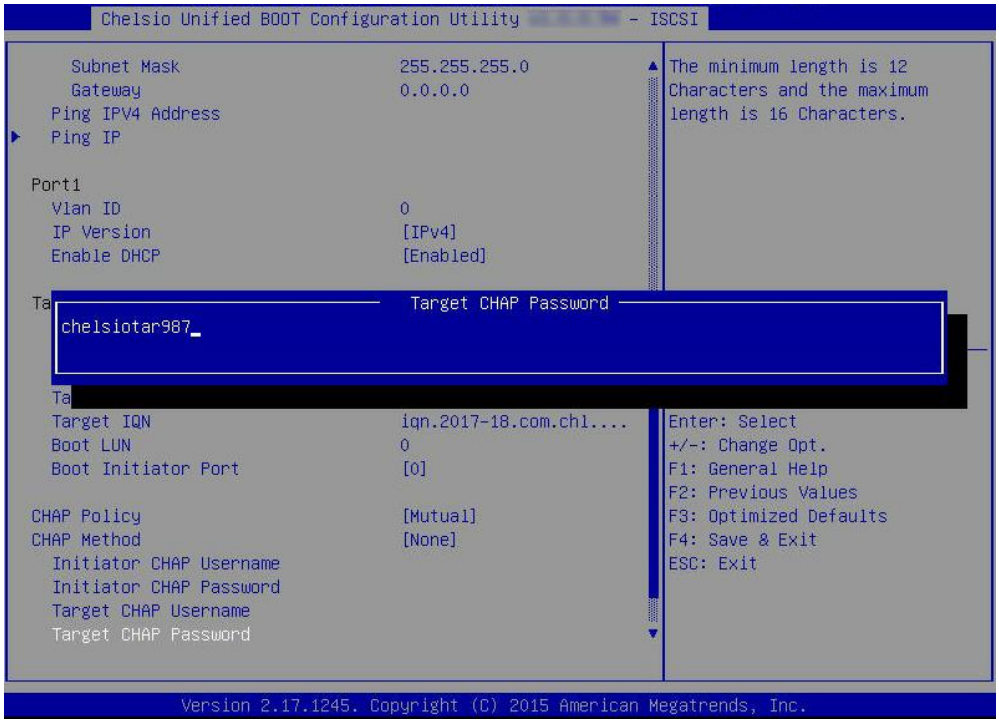
To discover target via static IP, select **Disabled** and provide the target IP.



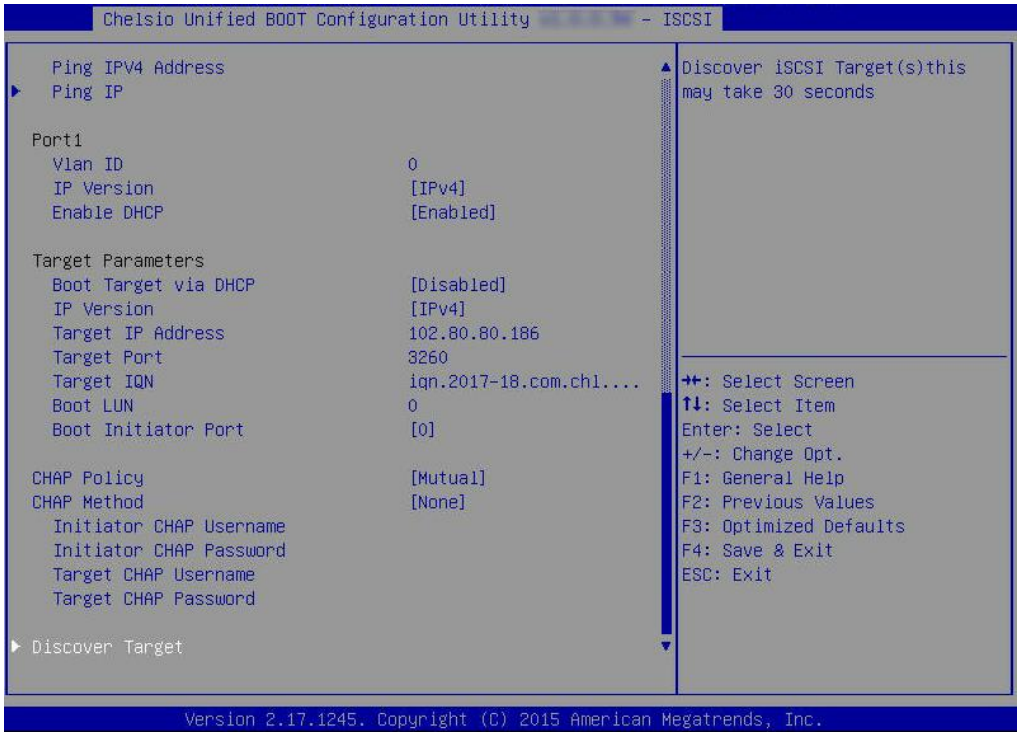
xiv. CHAP authentication is disabled by default. To enable and configure, highlight **CHAP Policy** and hit [Enter]. Select the policy type from the corresponding pop-up and hit [Enter] again.



xv. Provide Initiator and Target CHAP credentials as per the CHAP policy selected.



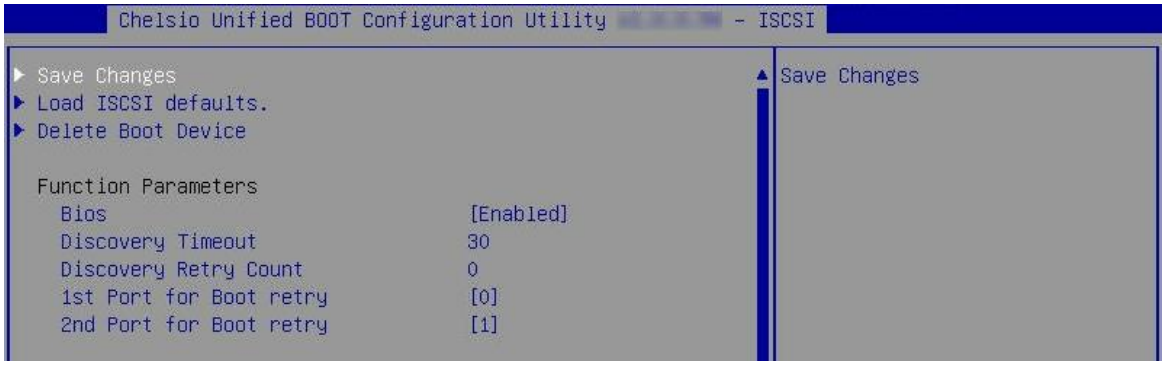
xvi. Select **Discover Target** and press [Enter] to discover iSCSI targets connected to the switch. Wait till all reachable targets are discovered.



xvii. A list of available targets will be displayed. Select the target you wish to connect to and hit [Enter].

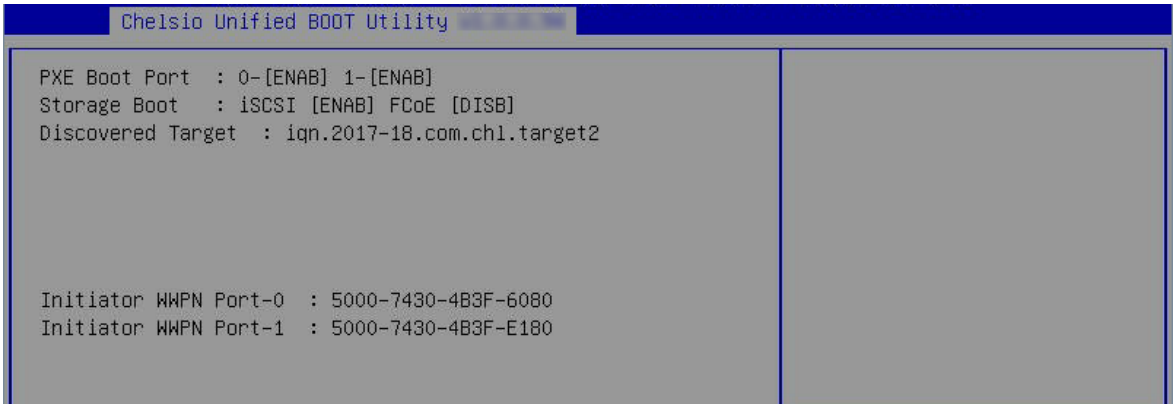
xviii. A list of LUNs configured on the selected target will be displayed. Select the LUN you wish to connect to and hit [Enter].

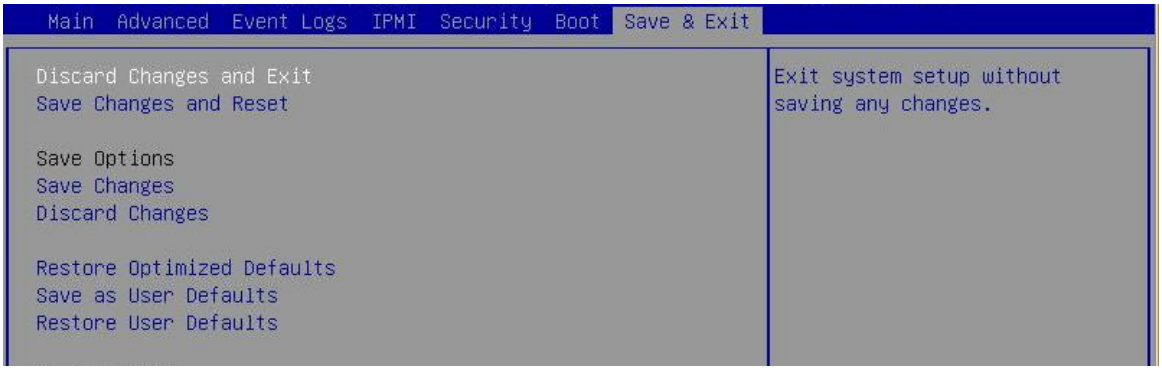
xix. Select **Save Changes** and press [Enter]



xx. Reboot the system for changes to take effect.

xxi. The discovered LUN should appear in the **Boot Configuration/ Boot Information** section and system BIOS.





xxii. Select the LUN as the first boot device and exit from BIOS.

xxiii. Either boot from the LUN or install the required OS.

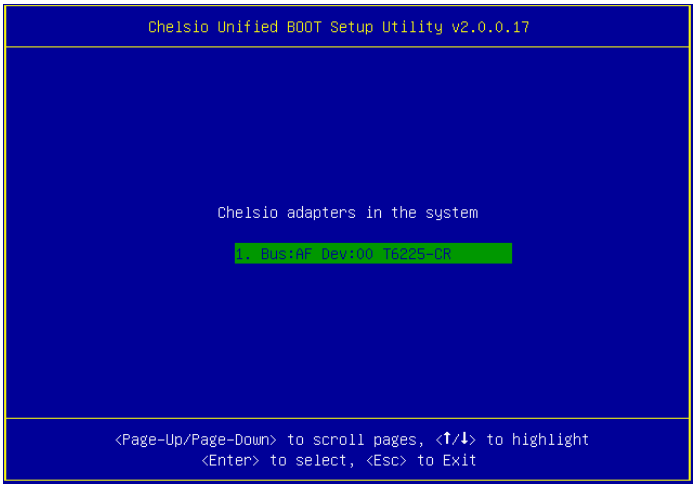
7.2.2. drvcfg

This section describes the method to configure and use Chelsio uEFI iSCSI interfaces using drvcfg.

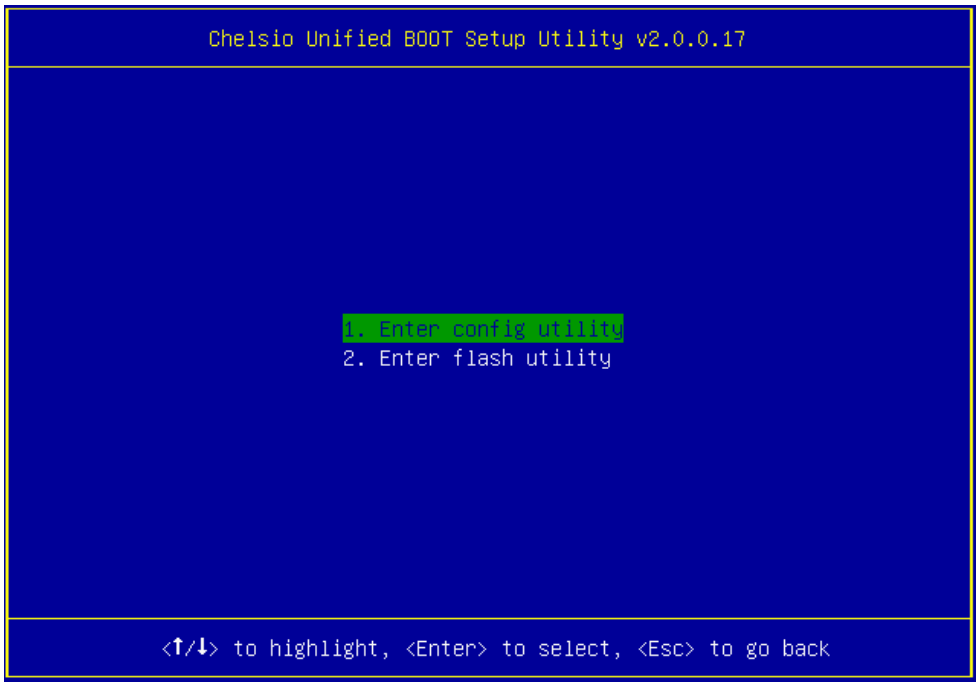
i. Boot the system into EFI shell.
ii. Run the following command to launch the configuration utility.

```
fs0:\> drvcfg -s_
```

iii. Choose the Chelsio adapter on which needs to be configured.



iv. Highlight **Enter config utility** and press [Enter].



v. Further configuration steps are similar from step (iv) of Legacy iSCSI Boot section.

8. Update Option ROM settings

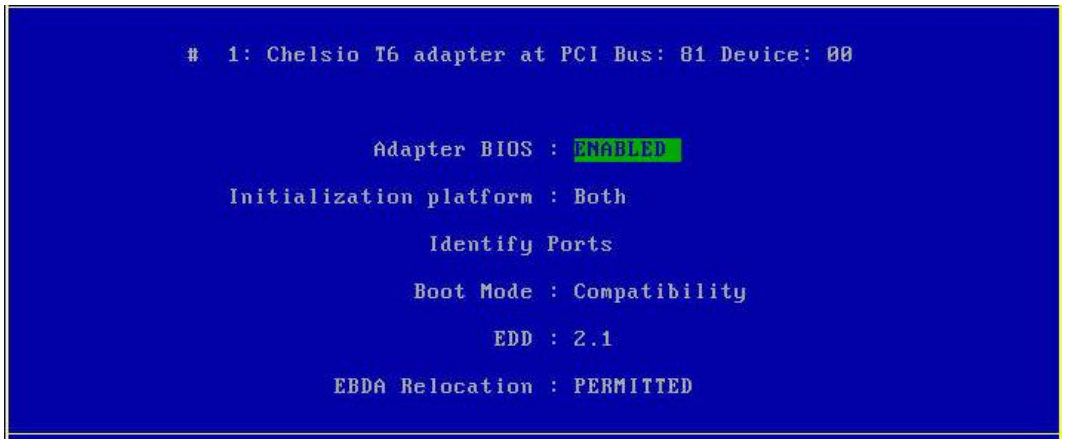
8.1. Default settings

If you wish to restore option ROM settings to their default values, i.e., PXE enabled, iSCSI and FCoE disabled, use any of the methods mentioned below:

8.1.1. Using Option ROM (boot level)

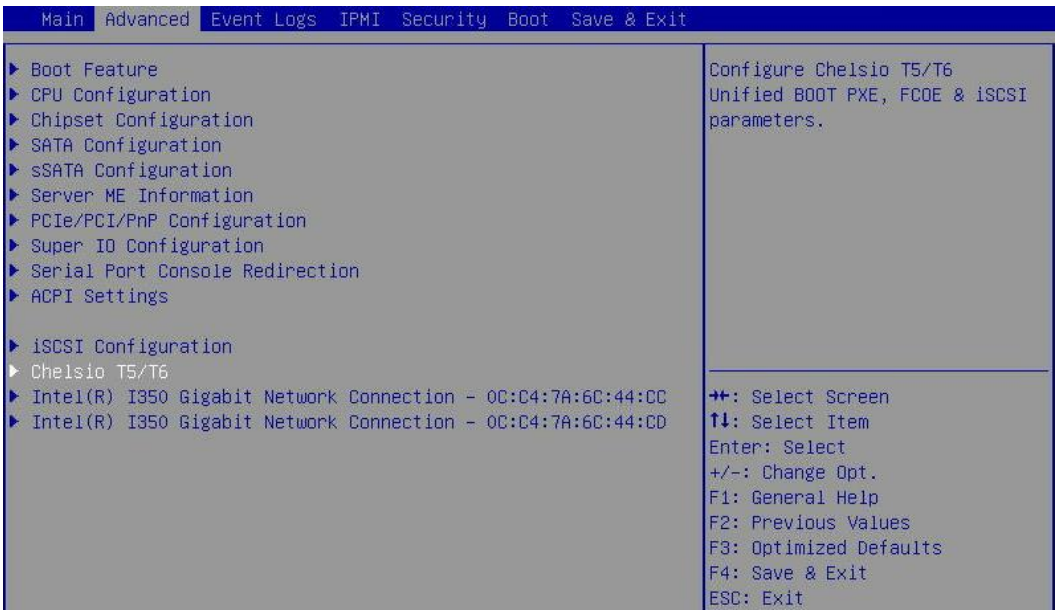
- **Legacy PXE**

Boot system into Chelsio's Unified Boot Setup utility and press F8.



- **uEFI PXE**

Boot system into uEFI mode and press F3.



8.1.2. Using *cxgbtool* (OS level)

Change your working directory to *OptionROM* directory and use *cxgbtool* to flash the default boot configuration onto the adapter.

```
[root@host~]# cd ChelsioUwire-x.x.x.x/Uboot/OptionROM/
[root@host~]# cxgbtool <ethX> loadboot-cfg boot.cfg
```

The below command can be used to read the current settings.

```
[root@host~]# cxgbtool <ethX> readboot-cfg
```

```
root@host:~# cxgbtool enp66s0f4 readboot-cfg
current boot setting is : 0x1 (NIC_BOOT: Enabled;  FCoE_BOOT: Disabled;  iSCSI_BOOT: Disabled)

NIC_BOOT: Port[0] : Enabled,  Port[1] : Enabled
Vlan:     Port[0] : 0, Port[1] : 0
```

8.2. Custom Settings (using cxgbtool)

cxgbtool utility can modify/update the following Option ROM settings using *modifyboot-cfg* option:

- PXE, FCoE and iSCSI BIOS
- Per port:
  - PXE Boot
  - VLAN

8.2.1. Updating BIOS value

Use the below command to enable/disable PXE/FCoE/iSCSI boot for all the ports of the adapter.

```
[root@host~]# cxgbtool <ethX> modifyboot-cfg bios <value>
```

Where,

ethX : Chelsio interface.
value : Bitwise OR of boot types that need to be enabled. Ranging from 0x0 – 0x7.
PXE (NIC) = 0x1
FCoE = 0x2
iSCSI = 0x4

Examples:

- To enable NIC and FCoE boot on all the ports,

```
root@host:~# cxgbtool enp66s0f4 modifyboot-cfg bios 0x3
root@host:~# cxgbtool enp66s0f4 readboot-cfg
current boot setting is : 0x3 (NIC_BOOT: Enabled;  FCoE_BOOT: Enabled;  iSCSI_BOOT: Disabled)

NIC_BOOT: Port[0] : Enabled,  Port[1] : Enabled
Vlan:     Port[0] : 0, Port[1] : 0
```

- To enable only iSCSI boot on all the ports,

```
root@host:~# cxgbtool enp66s0f4 modifyboot-cfg bios 0x4
root@host:~# cxgbtool enp66s0f4 readboot-cfg
current boot setting is : 0x4 (NIC_BOOT: Disabled;  FCoE_BOOT: Disabled;  iSCSI_BOOT: Enabled)

NIC_BOOT: Port[0] : Disabled,  Port[1] : Disabled
Vlan:     Port[0] : 0, Port[1] : 0
```

8.2.2. Per Port settings

Use the below command to enable/disable PXE (NIC) boot per port.

```
[root@host~]# cxgbtool <ethX> modifyboot-cfg port <port no.> <param>
```

Where,

ethX : Chelsio interface.
port no. : Port number ranging from 0 – 3.
param : en_nicboot to enable and dis_nicboot to disable NIC boot for the port.

Example:

- To disable NIC boot on Port 0,

```
root@host:~# cxgbtool enp66s0f4 modifyboot-cfg port 0 dis_nicboot
root@host:~# cxgbtool enp66s0f4 readboot-cfg
current boot setting is : 0x1 (NIC_BOOT: Enabled;  FCoE_BOOT: Disabled;  iSCSI_BOOT: Disabled)

NIC_BOOT: Port[0] : Disabled,  Port[1] : Enabled
Vlan:     Port[0] : 0, Port[1] : 0
```

Use the below command to set the VLAN id for the port.

```
[root@host~]# cxgbtool <ethX> modifyboot-cfg port <port no.> vlan <id>
```

Where,

ethX : Chelsio interface.
port no. : Port number ranging from 0 – 3.
id : VLAN id ranging from 0 – 4095.

Example:

- To set vlan id 50 on Port 1,

```
root@host:~# cxgbtool enp66s0f4 modifyboot-cfg port 1 vlan 50
root@host:~# cxgbtool enp66s0f4 readboot-cfg
current boot setting is : 0x1 (NIC_BOOT: Enabled;  FCoE_BOOT: Disabled;  iSCSI_BOOT: Disabled)

NIC_BOOT: Port[0] : Enabled,  Port[1] : Enabled
Vlan:     Port[0] : 0, Port[1] : 50
```

Note *For more information, please refer cxgbtool manpage using* `man cxgbtool`

XXVIII. Appendix

1. Troubleshooting

- ***Cannot bring up Chelsio interface***

Make sure you have created the corresponding network-script configuration file as stated in **Cheslsio Unified Wire** chapter (See Creating network-scripts). If the file does exist, make sure the structure and contents are correct. A sample is given in the **Chelsio Unified Wire** chapter (See Configuring network-scripts). Another reason may be that the IP address mentioned in the configuration file is already in use on the network.

- ***Cannot ping through Chelsio interface***

First, make sure the interface was successfully brought up using ifup ethX (where ethX is your interface) and that it is linked to an IP address, either static or obtained through DHCP.

You then may want to check whether the destination host (i.e. the machine you are trying to ping) is up and running and accepts ICMP requests such as ping. If you get a return value of 0 when doing a `cat /proc/sys/net/ipv4/icmp_echo_ignore_all` on the remote host that means it is configured to reply to incoming pings. Change `ipv4` to `ipv6` in the path if you are using IPv6. Note that this is a Linux-only tip.

If you have more than one interface wired to the network, make sure you are using the right one for your outgoing ping requests. This can be done by using the `-I` option of the ping command, as shown in the following example:

```
[root@host~]# ping -I eth1 10.192.167.1
```

Where 10.192.167.1 is the machine you want to ping.

- ***Configuring firewall for your application***

In many cases the firewall software on the systems may prevent the applications from working properly. Please refer to the appropriate documentation for the Linux distribution on how to configure or disable the firewall.

- ***FCoE link not up***

Always enable LLDP on the interfaces as FCoE link won't come up until and unless a successful LLDP negotiation happens.

- ***priority-flow-control mode on the switch***

On the switch, make sure priority-flow-control mode is always set to auto and flow control is disabled.

- ***Configuring Ethernet interfaces on Cisco switch***

Always configure Ethernet interfaces on Cisco switch in trunk mode.

- ***Binding VFC to MAC***

If you are binding the VFC to MAC address in case of Cisco Nexus switch, then make sure you make the Ethernet interface part of both Ethernet VLAN and FCoE VLAN.

- ***Cisco nexus switch reporting "pauseRateLimitErrDisable"***

If in any case the switch-port on the Cisco nexus switch is reporting "pauseRateLimitErrDisable", then perform an Ethernet port shut/no shut.

- ***"unexpected CM event" messages seen with iWARP traffic***

One reason for this could be port number collisions. To fix this, use iWARP port mapper (iwpmd).

```
[root@host~]# iwpmd
```

Note *iWARP port mapper (iwpmd) has its own issues.*

- ***Multiple Chelsio adapters***

Chelsio Option ROM supports upto 4 Chelsio adapters in a macine. In case of using more than 4 adapters, it is recommened to disable the Option ROM on the adapters.

- ***Installer issues***

In case of any failures while running the Chelsio Unified Wire Installer, please collect the below:

- install.log fille, if installed using install.py
- Entire make command output, if installed using the makefile

- ***Logs collection***

In case of any driver/firmware issues, please run the below command to collect all the necessary log files:

```
[root@host~]# chdebug
```

A compressed tar ball, *chelsio_debug_logs_with_cudbg.tar.bz2* will be created with all the logs.

In case of kernel panics, following files need to be provided for analysis.

```
vmcore, vmcore-dmesg.txt, vmlinux, System.map-$(uname  -r), Chelsio modules
.ko files
```

2. Chelsio End-User License Agreement (EULA)

Installation and use of the driver/software implies acceptance of the terms in the Chelsio End-User License Agreement (EULA).

IMPORTANT: PLEASE READ THIS SOFTWARE LICENSE CAREFULLY BEFORE DOWNLOADING OR OTHERWISE USING THE SOFTWARE OR ANY ASSOCIATED DOCUMENTATION OR OTHER MATERIALS (COLLECTIVELY, THE "SOFTWARE"). BY CLICKING ON THE "OK" OR "ACCEPT" BUTTON YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, CLICK THE "DO NOT ACCEPT" BUTTON TO TERMINATE THE INSTALLATION PROCESS.

1. License. Chelsio Communications, Inc. ("Chelsio") hereby grants you, the Licensee, and you hereby accept, a limited, non-exclusive, non-transferable license to install and use the Software with one or more Chelsio network adapters on a single server computer for use in communicating with one or more other computers over a network. You may also make one copy of the Software in machine readable form solely for back-up purposes, provided you reproduce Chelsio's copyright notice and any proprietary legends included with the Software or as otherwise required by Chelsio.

2. Restrictions. This license granted hereunder does not constitute a sale of the Software or any copy thereof. Except as expressly permitted under this Agreement, you may not:

(i) reproduce, modify, adapt, translate, rent, lease, loan, resell, distribute, or create derivative works of or based upon, the Software or any part thereof; or

(ii) make available the Software, or any portion thereof, in any form, on the Internet. The Software contains trade secrets and, in order to protect them, you may not decompile, reverse engineer, disassemble, or otherwise reduce the Software to a human-perceivable form. You assume full responsibility for the use of the Software and agree to use the Software legally and responsibly.

3. Ownership of Software. As Licensee, you own only the media upon which the Software is recorded or fixed, but Chelsio retains all right, title and interest in and to the Software and all subsequent copies of the Software, regardless of the form or media in or on which the Software may be embedded.

4. Confidentiality. You agree to maintain the Software in confidence and not to disclose the Software, or any information or materials related thereto, to any third party without the express written consent of Chelsio. You further agree to take all reasonable precautions to limit access of the Software only to those of your employees who reasonably require such access to perform their employment obligations and who are bound by confidentiality agreements with you.

5. Term. This license is effective in perpetuity, unless terminated earlier. You may terminate the license at any time by destroying the Software (including the related documentation), together with all copies or modifications in any form. Chelsio may terminate this license, and this license shall be deemed to have automatically terminated, if you fail to comply with any term or condition of this Agreement. Upon any termination, including termination by you, you must destroy the Software (including the related documentation), together with all copies or modifications in any form.

6. Limited Warranty. If Chelsio furnishes the Software to you on media, Chelsio warrants only that the media upon which the Software is furnished will be free from defects in

material or workmanship under normal use and service for a period of thirty (30) days from the date of delivery to you.

CHELSIO DOES NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE SOFTWARE OR ANY PART THEREOF. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, CHELSIO MAKES NO OTHER WARRANTIES, EXPRESS OR IMPLIED, AND HEREBY DISCLAIMS ALL OTHER WARRANTIES, INCLUDING, BUT NOT LIMITED TO, NON-INFRINGEMENT OF THIRD PARTY RIGHTS, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow the exclusion of implied warranties or limitations on how long an implied warranty may last, so the above limitations may not apply to you. This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

7. Remedy for Breach of Warranty. The sole and exclusive liability of Chelsio and its distributors, and your sole and exclusive remedy, for a breach of the above warranty, shall be the replacement of any media furnished by Chelsio not meeting the above limited warranty and which is returned to Chelsio. If Chelsio or its distributor is unable to deliver replacement media which is free from defects in materials or workmanship, you may terminate this Agreement by returning the Software.

8. Limitation of Liability. IN NO EVENT SHALL CHELSIO HAVE ANY LIABILITY TO YOU OR ANY THIRD PARTY FOR ANY INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, HOWEVER CAUSED, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE LICENSE OR USE OF THE SOFTWARE, INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR LOSS OF ANTICIPATED PROFITS, EVEN IF CHELSIO HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL CHELSIO'S LIABILITY ARISING OUT OF OR RELATED TO THE LICENSE OR USE OF THE SOFTWARE EXCEED THE AMOUNTS PAID BY YOU FOR THE LICENSE GRANTED HEREUNDER. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

9. High Risk Activities. The Software is not fault-tolerant and is not designed, manufactured or intended for use or resale as online equipment control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical or environmental damage. Chelsio specifically disclaims any express or implied warranty of fitness for any high risk uses listed above.

10. Export. You acknowledge that the Software is of U.S. origin and subject to U.S. export jurisdiction. You acknowledge that the laws and regulations of the United States and other countries may restrict the export and re-export of the Software. You agree that you will not export or re-export the Software or documentation in any form in violation of applicable United States and foreign law. You agree to comply with all applicable international and national laws that apply to the Software, including the U.S.

Export Administration Regulations, as well as end-user, end-use, and destination restrictions issued by U.S. and other governments.

11. Government Restricted Rights. The Software is subject to restricted rights as follows. If the Software is acquired under the terms of a GSA contract: use, reproduction or disclosure is subject to the restrictions set forth in the applicable ADP Schedule contract. If the Software is acquired under the terms of a DoD or civilian agency contract, use, duplication or disclosure by the Government is subject to the restrictions of this Agreement in accordance with 48 C.F.R. 12.212 of the Federal

Acquisition Regulations and its successors and 49 C.F.R. 227.7202-1 of the DoD FAR Supplement and its successors.

12. General. You acknowledge that you have read this Agreement, understand it, and that by using the Software you agree to be bound by its terms and conditions. You further agree that it is the complete and exclusive statement of the agreement between Chelsio and you, and supersedes any proposal or prior agreement, oral or written, and any other communication between Chelsio and you relating to the subject matter of this Agreement. No additional or any different terms will be enforceable against Chelsio unless Chelsio gives its express consent, including an express waiver of the terms of this Agreement, in writing signed by an officer of Chelsio. This Agreement shall be governed by California law, except as to copyright matters, which are covered by Federal law. You hereby irrevocably submit to the personal jurisdiction of, and irrevocably waive objection to the laying of venue (including a waiver of any argument of forum non conveniens or other principles of like effect) in, the state and federal courts located in Santa Clara County, California, for the purposes of any litigation undertaken in connection with this Agreement. Should any provision of this Agreement be declared unenforceable in any jurisdiction, then such provision shall be deemed severable from this Agreement and shall not affect the remainder hereof. All rights in the Software not specifically granted in this Agreement are reserved by Chelsio. You may not assign or transfer this Agreement (by merger, operation of law or in any other manner) without the prior written consent of Chelsio and any attempt to do so without such consent shall be void and shall constitute a material breach of this Agreement.

Should you have any questions concerning this Agreement, you may contact Chelsio by writing to:

Chelsio Communications, Inc.
735 N Pastoria Avenue,
Sunnyvale, CA 94085
U.S.A