Chelsio Communications

Accelerate

Chelsio Unified Wire for Linux

Installation and User Guide



This document and related products are distributed under licenses restricting their use, copying, distribution, and reverse-engineering.

No part of this document may be reproduced in any form or by any means without prior written permission by Chelsio Communications.

All third-party trademarks are copyright of their respective owners.

THIS DOCUMENTATION IS PROVIDED “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

THE USE OF THE SOFTWARE AND ANY ASSOCIATED MATERIALS (COLLECTIVELY THE “SOFTWARE”) IS SUBJECT TO THE SOFTWARE LICENSE TERMS OF CHELSIO COMMUNICATIONS, INC.



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, send an email to sales@chelsio.com

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

Copyright © 2026. Chelsio Communications. All Rights Reserved.
Chelsio ® is a registered trademark of Chelsio Communications.
All other marks and names mentioned herein may be trademarks of their respective companies.

Document History

Version	Revision Date
1.5.7	01/31/2022
1.5.8	03/15/2022
1.5.9	07/29/2022
1.6.0	12/02/2022
1.6.1	04/07/2023
1.6.2	08/04/2023
1.6.3	01/22/2024
1.6.4	04/30/2024
1.6.5	11/22/2024
1.6.6	11/28/2025
1.6.7	01/30/2026
1.6.8	03/02/2026
1.6.9	03/31/2026
1.7.0	04/15/2026
1.7.1	04/30/2026
1.7.2	06/30/2026
1.7.2	07/31/2026
1.7.3	08/31/2026
1.7.4	10/08/2026
1.7.5	11/02/2026

TABLE OF CONTENTS

I. CHELSIO UNIFIED WIRE 16

1. Introduction 17

1.1. Features 18

1.2. Hardware Requirements 19

1.3. Software Requirements 19

1.4. Package Contents 19

2. Hardware Installation 22

2.1. Installing the Chelsio T7 Adapters 22

2.2. Installing the Chelsio T6 Adapters 23

3. Software/Driver Installation 25

3.1. Pre-requisites 25

3.2. Installing Chelsio Unified Wire 26

3.3. Firmware flash and init/vpd update 32

3.4. Mounting debugfs 34

3.5. Collect all adapter logs using cudbg_app 34

4. Configuring Chelsio Network Interfaces 35

4.1. Configuring Chelsio T7 Adapters 35

4.2. Configuring Chelsio T6 Adapters 35

4.3. Configuring NetworkManager keyfiles 39

4.4. Creating Network Manager keyfiles manually 40

4.5. Configuring IPv6 41

4.6. Checking Link 41

5. Performance Tuning 42

5.1. Generic 42

5.2. Throughput 42

5.3. Latency 43

6. Software/Driver Update 44

7. Software/Driver Uninstallation 45

7.1. Uninstalling Chelsio Unified Wire from source 45

II. NETWORK (NIC/TOE) 46

1. Introduction 47

1.1. Hardware Requirements 47

1.2. Software Requirements 48

2. Software/Driver Installation 49

3. Software/Driver Loading 50

3.1. Loading in NIC mode (without full offload support) 50

3.2. Loading in TOE mode (with full offload support) 50

4. Software/Driver Configuration 51

4.1. Enabling TCP Offload 51

4.2. Precision Time Protocol (PTP) 51
4.3. VXLAN Offload 52
4.4. HMA 55
4.5. Performance Tuning 55
5. Software/Driver Unloading 61
5.1. Unloading the NIC Driver 61
5.2. Unloading the TOE Driver 61

III. PORT FORWARDING MODE (HARDWARE BYPASS MODE/HAIRPIN MODE/LAN BYPASS) 63
1. Introduction 64
1.1. Hardware Requirements 64
1.2. Software Requirements 64
2. Software/Driver Installation 65
3. Firmware and init/vpd update 66
4. Software/Driver Loading 68
5. Hardware bypass on power-on 68
6. Hardware bypass with driver deadman watchdog 69
7. Hardware bypass with IOCTL-Based deadman watchdog 71
7.1. cxgb4 driver parameter 71
7.2. Reference Application 72
8. Software/Driver Unloading 75
8.1. Unloading the Driver 75

IV. VIRTUAL FUNCTION NETWORK (VNIC) 76
1. Introduction 77
1.1. Hardware Requirements 77
1.2. Software Requirements 78
2. Software/Driver Installation 79
2.1. Pre-requisites 79
2.2. Installation 79
3. Software/Driver Loading 80
3.1. Instantiate Virtual Functions (SR-IOV) 80
3.2. Loading the Driver 80
4. Software/Driver Configuration and Fine-tuning 81
4.1. VF Communication 81
4.2. VF Link state 82
4.3. VF Rate Limiting 82
4.4. Bonding 83
4.5. High Capacity VF Configuration 85
5. Software/Driver Unloading 87
5.1. Unloading the Driver 87

V. IWARP RDMA OFFLOAD 88
1. Introduction 89
1.1. Hardware Requirements 89
1.2. Software Requirements 89
2. Software/Driver Installation 90
2.1. Pre-requisites 90
2.2. Installation 90
3. Software/Driver Loading 91
3.1. Loading iWARP Driver 91
4. Software/Driver Configuration and Fine-tuning 92
4.1. Testing connectivity with *ping* and *rping* 92
4.2. Enabling various MPIs 93
4.3. Setting up NFS-RDMA 97
4.4. HMA 98
4.5. Performance Tuning 99
5. Software/Driver Unloading 100
VI. ROCEV2 101
1. Introduction 102
1.1. Hardware Requirements 102
1.2. Software Requirements 102
2. Software/Driver Loading 103
2.1. Loading RoCEv2 Driver 103
3. Software/Driver Configuration and Fine-tuning 104
3.1. Testing connectivity with *ping* and *rping* 104
3.2. Setting up NFS-RDMA 105
3.3. Performance Tuning 106
4. Software/Driver Unloading 107
VII. NVME-OF IWARP 108
1. Introduction 109
1.1. Hardware Requirements 109
1.2. Software Requirements 109
2. Kernel Configuration 110
3.1. Pre-requisites 111
3.2. Installation 111
4. Software/Driver Loading 112
5. Software/Driver Configuration and Fine-tuning 113
5.1. Target 113
5.2. Initiator 114
5.3. HMA 115
5.4. Performance Tuning 115

6. Software/Driver Unloading 117

VIII. NVME-OF ROCEV2 118

1. Introduction 119

1.1. Hardware Requirements 119

1.2. Software Requirements 120

2. Kernel Configuration 121

3. Software/Driver Loading 122

4. Software/Driver Configuration and Fine-tuning 123

4.1. Target 123

4.2. Performance Tuning 124

5. Software/Driver Unloading 125

IX. SPDK (USER SPACE) NVME-OF IWARP 126

1. Introduction 127

1.1. Hardware Requirements 127

1.2. Software Requirements 127

2. Kernel Configuration 128

3. Software/Driver Loading 129

4. Software/Driver Configuration and Fine-tuning 130

4.1. Target 130

4.2. Initiator 131

4.3. Performance Tuning 131

5. Software/Driver Unloading 132

X. SPDK (USER SPACE) NVME-OF ROCEV2 133

1. Introduction 134

1.1. Hardware Requirements 134

1.2. Software Requirements 134

2. Kernel Configuration 135

3. Software/Driver Loading 136

4. Software/Driver Configuration and Fine-tuning 137

4.1. Target 137

4.2. Initiator 138

4.3. Performance Tuning 138

5. Software/Driver Unloading 139

XI. NVME-OF TOE 140

1. Introduction 141

1.1. Hardware Requirements 141

1.2. Software Requirements 142

2. Kernel Configuration 143

3. Software/Driver Installation 144
3.1. Installation 144
4. Software/Driver Loading 145
5. Software/Driver Configuration and Fine-tuning 146
5.1. Target 146
5.2. Initiator 146
5.3. HMA 147
5.4. Performance Tuning 148
6. Software/Driver Unloading 149

XII. SPDK (USER SPACE) NVME-OF TOE 150
1. Introduction 151
1.1. Hardware Requirements 151
1.2. Software Requirements 151
2. Kernel Configuration 152
3. Software/Driver Installation 153
3.1. Installation 153
4. Software/Driver Loading 154
5. Software/Driver Configuration and Fine-tuning 155
5.1. Target 155
5.2. Initiator 156
6. Software/Driver Unloading 157

XIII. NVME-OF/TCP PDU OFFLOAD 158
1. Introduction 159
1.1. Hardware Requirements 159
1.2. Software Requirements 159
2. Kernel Configuration 160
3. Software/Driver Loading 161
4. Software/Driver Configuration and Fine-tuning 162
4.1. Target 162
4.2. Initiator 163
5. Software/Driver Unloading 164

XIV. SPDK (USER SPACE) NVME/TCP PDU OFFLOAD 165
1. Introduction 166
1.1. Hardware Requirements 166
1.2. Software Requirements 166
2. Driver Installation 167
3. Software/Driver Loading 168
4. Configuration 169
4.1. Configuring SPDK NVMe/TCP PDU Offload 169

4.2. SPDK NVMe/TCP PDU Offload Target 169
4.3. SPDK NVMe/TCP PDU Offload Initiator 170
5. Software/Driver Unloading 171

XV. SPDK NVME/TCP OVER MEMFREE TOE 172
1. Introduction 173
1.1. Hardware Requirements 173
1.2. Software Requirements 173
2. Compiling and installing spdk 174
3. Software/Driver Loading 175
4. Configuration 176
4.1. Configuring SPDK NVMe/TCP over Memfree TOE 176
4.2. SPDK NVMe/TCP over Memfree TOE Target 176
4.3. SPDK NVMe/TCP Over Memfree TOE Initiator 177

XVI. SOFTIWARP 178
1. Introduction 179
1.1. Hardware Requirements 179
1.2. Software Requirements 180
2. Kernel Configuration 181
3. Software/Driver Loading 182
4. Software/Driver Configuration and Fine-tuning 183
4.1. Initiator/Client 183
5. Software/Driver Unloading 184

XVII. LIO ISCSI TARGET OFFLOAD 185
1. Introduction 186
1.1. Hardware Requirements 186
1.2. Software Requirements 187
2. Kernel Configuration 188
3. Software/Driver Installation 190
3.1. Pre-requisites 190
3.2. Installation 190
4. Software/Driver Loading 192
5. Software/Driver Configuration and Fine-tuning 193
5.1. Configuring LIO iSCSI Target 193
5.2. Offloading LIO iSCSI Connection 193
5.3. Running LIO iSCSI and Network Traffic Concurrently 193
5.4. Performance Tuning 194
6. Software/Driver Unloading 195
6.1. Unloading the LIO iSCSI Target Offload Driver 195
6.2. Unloading the NIC Driver 195

XVIII.ISCSI PDU OFFLOAD INITIATOR 196
1. Introduction 197
1.1. Hardware Requirements 198
1.2. Software Requirements 198
1.3. Pre-requisites 198
2. Software/Driver Installation 199
2.1. Pre-requisites 199
2.2. Installation 199
3. Software/Driver Loading 200
4. Software/Driver Configuration and Fine-tuning 201
4.1. Accelerating open-iSCSI Initiator 201
4.2. HMA 203
4.3. Auto login from cxgb4i initiator at OS bootup 204
4.4. Performance Tuning 204
5. Software/Driver Unloading 205
XIX. SPDK (USER SPACE) ISCSI PDU OFFLOAD 206
1. Introduction 207
1.1. Hardware Requirements 207
1.2. Software Requirements 207
2. Driver Installation 208
3. Software/Driver Loading 209
4. Configuration 210
4.1. Configuring SPDK iSCSI PDU Offload target/initiator 210
4.2. SPDK iSCSI PDU Offload Target 210
4.3. SPDK iSCSI PDU Offload Initiator 211
5. Software/Driver Unloading 211
XX. SPDK (USER SPACE) ISCSI OVER MEMFREE TOE 212
1. Introduction 213
1.1. Hardware Requirements 213
1.2. Software Requirements 213
2. Compiling and installing spdk 214
3. Software/Driver Loading 215
4. Configuration 216
4.1. Configuring SPDK iSCSI over Memfree TOE target 216
4.2. SPDK iSCSI over Memfree TOE Target 216
XXI. CRYPTO OFFLOAD 217
1. Introduction 218
1.1. Hardware Requirements 218
1.2. Software Requirements 218

2. Kernel Configuration 219
3. Software/Driver Installation 221
3.1. Pre-requisites 221
3.2. Installation 221
4. Software/Driver Loading 222
4.1. Inline 222
4.2. Co-processor 222
5. Software/Driver Configuration and Fine-tuning 224
5.1. Configuring OpenSSL 224
5.2. Co-processor IPsec Offload 226
5.3. Inline TLS Offload 230
5.4. Co-processor 233
5.5. Performance Tuning 236
6. Software/Driver Unloading 237

XXII. QUIC TX OFFLOAD 238
1. Introduction 239
1.1. Hardware Requirements 240
1.2. Software Requirements 240
2. Software/Driver Installation 241
3. Software/Driver Loading 242
4. Software/Driver Configuration 243
4.1. Compilation of QUIC enabled OpenSSL 243
4.2. Compilation of Chelsio QUIC HW offload nginx 243
5. Client Application 246
5.1. Compilation of curl with QUIC enabled OpenSSL 246
5.2. Using custom cURL to download files 248
5.3. QUIC Offload Counters on Server-end 248
6. Software/Driver Unloading 248

XXIII. INLINE IPSEC OFFLOAD 249
1. Introduction 250
1.1. Hardware Requirements 250
1.2. Software Requirements 250
2. Kernel Configuration 251
2.1. Kernel Configuration 251
3. Software/Driver Loading 252
4. Configuration 253
4.1. Using ip xfrm 253
5. Software/Driver Unloading 258

XXIV. ISER 259

1. Introduction **260**
1.1. Hardware Requirements 260
1.2. Software Requirements 260
2. Kernel Configuration **261**
3. Software/Driver Installation **262**
3.1. Pre-requisites 262
3.2. Installation 262
4. Software/Driver Loading **263**
5. Software/Driver Configuration and Fine-tuning **264**
5.1. HMA 265
5.2. Performance Tuning 265
6. Software/Driver Unloading **266**

XXV. WD-UDP **267**
1. Introduction **268**
1.1. Hardware Requirements 268
1.2. Software Requirements 268
2. Software/Driver Installation **269**
2.1. Pre-requisites 269
2.2. Installation 269
3. Software/Driver Loading **270**
4. Software/Driver Configuration and Fine-tuning **271**
4.1. Accelerating UDP Socket Communications 271
5. Software/Driver Unloading **276**

XXVI. DATA CENTER BRIDGING (DCB) **277**
1. Introduction **278**
1.1. Hardware Requirements 278
1.2. Software Requirements 279
2. Software/Driver Installation **280**
3. Software/Driver Loading **281**
4. Software/Driver Configuration and Fine-tuning **282**
4.1. Configuring Cisco Nexus 5010 switch 282
4.2. Configuring the Brocade 8000 switch 284
5. Running NIC and iSCSI Traffic together with DCBx **286**

XXVII. FCOE FULL OFFLOAD INITIATOR **287**
1. Introduction **288**
1.1. Hardware Requirements 288
1.2. Software Requirements 289
2. Software/Driver Installation **290**
3. Software/Driver Loading **291**

4. Software/Driver Configuration and Fine-tuning 292
4.1. Configuring Cisco Nexus 5010 and Brocade switch 292
4.2. FCoE fabric discovery verification 292
4.3. Formatting the LUNs and Mounting the Filesystem 295
4.4. Creating Filesystem 296
4.5. Mounting the formatted LUN 297
5. Software/Driver Unloading 298

XXVIII. OFFLOAD BONDING 299
1. Introduction 300
1.1. Hardware Requirements 300
1.2. Software Requirements 300
2. Software/Driver Installation 301
3. Software/Driver Loading 302
4. Software/Driver Configuration and Fine-tuning 303
4.1. Offloading TCP traffic over a bonded interface 303
5. Software/Driver Unloading 304

XXIX. UDP SEGMENTATION OFFLOAD AND PACING 305
1. Introduction 306
1.1. Hardware Requirements 307
1.2. Software Requirements 307
2. Software/Driver Installation 308
3. Software/Driver Loading 309
4. Software/Driver Configuration and Fine-tuning 310
4.1. Modifying the Application 310
4.2. Configuring UDP Pacing 312
4.3. Enabling Offload 313
5. Software/Driver Unloading 314

XXX. OFFLOAD IPV6 315
1. Introduction 316
1.1. Hardware Requirements 316
1.2. Software Requirements 317
2. Software/Driver Installation 318
2.1. Pre-requisites 318
2.2. Installation 318
3. Software/Driver Loading 319
4. Software/Driver Configuration and Fine-tuning 320
5. Software/Driver Unloading 321
5.1. Unloading the NIC Driver 321
5.2. Unloading the TOE Driver 321

XXXI. WD SNIFFING AND TRACING 322
1. Theory of Operation 323
1.1. Hardware Requirements 325
1.2. Software Requirements 325
2. Software/Driver Installation 326
2.1. Pre-requisites 326
2.2. Installation 326
3. Usage 327
3.1. Installing Basic Support 327
3.2. Using Sniffer (wd_*sniffer)* 327
3.3. Using Tracer (wd_tcpdump_trace) 328
XXXII. CLASSIFICATION AND FILTERING 329
1. Introduction 330
1.1. Hardware Requirements 330
1.2. Software Requirements 331
2. LE-TCAM Filters 332
2.1. Configuration 332
2.2. Creating Filter Rules 335
2.3. Listing Filter Rules 336
2.4. Removing Filter Rules 336
2.5. Layer 3 Example 337
2.6. Layer 2 Example 338
2.7. Filtering VF traffic 340
3. Hash/DDR Filters 342
3.1. Configuration 342
3.2. Creating Filter Rules 344
3.3. Listing Filter Rules 346
3.4. Removing Filter Rules 347
3.5. Filter Priority 347
3.6. Swap MAC Feature 347
3.7. Traffic Mirroring 347
3.8. Packet Tracing and Hit Counters 349
4. NAT Filtering 356
XXXIII. MESH TOPOLOGY 357
1. Introduction 358
1.1. Hardware Requirements 358
1.2. Software Requirements 358
1.3. Mesh topology 359
2. Software/Driver Installation 360
3. Software/Driver Configuration and Fine-tuning 361

XXXIV. TRAFFIC MANAGEMENT 362
1. Introduction 363
1.1. Hardware Requirements 363
1.2. Software Requirements 365
2. Software/Driver Loading 366
3. Software/Driver Configuration and Fine-tuning 367
3.1. Traffic Management Rules 367
3.2. Configuring Traffic Management 369
4. Usage 372
4.1. Non-Offloaded Connections 372
4.2. Offloaded Connections 372
4.3. Offloaded Connections with Modified Application 373
4.4. Inline TLS Offload Connections 374
5. Software/Driver Unloading 375

XXXV. UNIFIED BOOT 376
1. Introduction 377
1.1. Hardware Requirements 377
1.2. Software Requirements 379
1.3. Pre-requisites 379
2. Secure Boot 379
3. Chelsio T7 Adapter 380
3.1. Flashing firmware, option ROM and boot.cfg 380
3.2. Configuring PXE Server 394
3.3. PXE Boot Process 399
3.4. OS Installation 407

XXXVI. NC-SI (NETWORK CONTROLLER SIDEBAND INTERFACE) 410
1. Introduction 411
1.1. Hardware Requirements 411
1.2. Software Requirements 412
2. Firmware and init/vpd update 413
3. Package and Channel Configuration 415
3.1. Pass-through Traffic 415
4. Recommended BMC Bring-up Sequence 416
4.1. AEN Configuration 416

XXXVII. APPENDIX 417
1. Limitations 418
2. Troubleshooting 418
3. Chelsio End-User License Agreement (EULA) 421

I. Chelsio Unified Wire

1. Introduction

Thank you for choosing Chelsio Unified Wire adapters. These high-speed, single-chip, multicore firmware cards provide enterprises and data centers with high-performance solutions for various network and storage-related requirements. Designed to meet the demands of modern workloads, Chelsio adapters offer advanced offloads, scalable throughput, and low latency to enhance performance across diverse applications.

The **Terminator series** represents Chelsio's next generation of highly integrated, hyper-virtualized adapters built around a programmable protocol-processing engine. These Unified Wire Adapters provide full offload support for a complete Unified Wire solution, integrating **NIC, TOE, iWARP, RoCEv2, and iSCSI** functionalities within a single platform. They deliver scalable, true line-rate performance across **10/25/40/50/100/200/400GbE**, efficiently supporting workloads ranging from a single TCP connection to thousands of concurrent connections. Leveraging multiple physical channels within the ASIC, the architecture enables ultra-low latency and high-bandwidth operation while allowing unmodified IP sockets, Fibre Channel, and InfiniBand applications to run simultaneously over Ethernet.

Building on this innovation, the **Chelsio T7 adapter family** provides advanced hardware acceleration, protocol offloads, and scalable throughput to address modern networking demands. The family includes **T7 Storage and DPU Adapters** and **S7 SmartNICs**. The T7 Storage and DPU Adapters and S7 SmartNICs are high-performance Ethernet Unified Wire ASICs available in quad/dual/single-port configurations, supporting **10/25/40/50/100/200/400GbE**, along with **PCIe Gen5** and advanced offload features. In comparison, the Chelsio T6 adapter family supports configurations up to 10/25/40/50/100GbE. The T7 platform sets new benchmarks in bandwidth, latency, CPU offload, and feature integration, making it an ideal solution for Linux-based data centers, storage systems, and high-performance computing environments.

The **T7 Storage and DPU Adapters** have integrated ARM cores and support all port configurations and a full range of offloads, including **NIC, TOE, iWARP, RoCEv2, and iSCSI** functionalities. The **Chelsio S7 SmartNICs** are an Ethernet Unified Wire SmartNIC ASIC optimized for base offloads—including **TOE, RDMA, Crypto, and iSCSI**—using host memory and supporting a wide range of data center networking applications.

The **Chelsio Unified Wire software** is provided in source code format, allowing users to compile the package to generate necessary binaries—an approach recommended when using a custom-built kernel.

This document describes the installation, use, and maintenance of the Unified Wire software and its various components, helping users configure and operate Chelsio adapters to achieve optimal performance in their network and storage environments.

Disclaimer: This initial release may contain corner-case and platform-specific issues.

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**
- **RoCEv2**
- **NVMe-oF iWARP (Target and Initiator)**
- **NVMe-oF RoCEv2 (Target)**
- **SPDK (user space) NVMe-oF iWARP (Target and Initiator)**
- **NVMe-oF TOE (Target and Initiator)**
- **SPDK (user space) NVMe-oF TOE (Target)**
- **NVMe-oF/TCP PDU Offload (Target and Initiator)**
- **SPDK (user space) NVMe/TCP PDU Offload (Target and Host)**
- **SPDK (user space) NVMe/TCP over Memfree TOE (Target and Initiator)**
- **SoftiWARP Initiator**
- **LIO iSCSI Target Offload**
- **iSCSI PDU Offload Initiator**
- **SPDK (user space) iSCSI PDU Offload (Target and Initiator)**
- **SPDK (user space) iSCSI over Memfree TOE (Target and Initiator)**
- **Crypto Offload**
- **QUIC TX Offload**
- **Inline IPSec Offload**
- **iSER (Target and Initiator)**
- **WD-UDP**
- **Data Center Bridging (DCB)**
- **FCoE Full Offload Initiator**
- **Offload Bonding**
- **UDP Segmentation Offload and Pacing**
- **Offload IPV6**
- **WD Sniffing and Tracing**
- **Classification and Filtering**
- **Mesh Topology**
- **Traffic Management feature (TM)**
- **Unified Boot Software**
- **Utility Tools (cop,cxgbtool,t4_perftune)**

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

1.3. Software Requirements

The Chelsio Unified Wire software has been developed to run on 64-bit Linux-based platforms; a 64-bit Linux platform is therefore a base requirement for running the driver. To know more about the complete list of operating systems supported by each driver, 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 Guide (this document) for the software.
- **libs**: This directory is for iWARP and WD-UDP libraries. The libibverbs library has an 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. 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. Find more details on this tool in its manual page (run man cxgbtool command).
- **nvme_utils**: This directory contains *nvmecli*, *nvmetcli* and *targetcli* installation files, and dependent components.
- **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 whether the Linux platform is supported or not.
- **wdload**: UDP acceleration tool.
- **wdunload**: Used to unload all the loaded Chelsio drivers.

- **Uboot**: This directory contains Unified Boot Option ROM image (cubt4.bin), PXE only Option ROM image (*cpbt4.bin*), uEFI driver (*ChelsioUD.efi* and *ChelsioPXE.efi*) and default boot configuration file (*boot.cfg*), which can be used to flash the Option ROM onto Chelsio adapters. It also contains the T7 Firmware file.

Note *The Unified Boot OptionROM image cubt4.bin at present supports only PXE. Support for iSCSI and FCoE will be added in subsequent releases.*

- **install.py**, **dialog.py**: Python scripts needed for the GUI installer.
- **EULA**: Chelsio's End User License Agreement.

- **install.log**: File containing installation summary.
- **Makefile**: The Makefile for building and installing from the source.

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. That is README, Release Notes, and User 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 Linux Driver (*cxgb4.ko*) files for supported OSes.

2. Hardware Installation

This section consists of the following topics:

- Installing the Chelsio T7 Adapters
- Installing the Chelsio T6 Adapters

2.1. Installing the Chelsio T7 Adapters

Follow these steps to install the Chelsio T7 adapter in your system:

1. Shutdown/power off your system.
2. Power off all remaining peripherals attached to your system.
3. Unpack the Chelsio adapter and place it on an anti-static surface.
4. Remove the system case cover as per the system manufacturer's instructions.
5. Remove the PCI filler plate from the slot where you will install the Ethernet adapter.
6. For maximum performance, it is highly recommended to install the adapter into a PCIe x8/x16 slot.
7. 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.
8. Secure the Chelsio adapter with a screw, or other securing mechanism, as described by the system manufacturer's instructions. Replace the case cover.
9. 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.
10. Connect a fiber optic cable to the Chelsio T7 adapter port.
11. Power on your system.
12. 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.
```

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

```
[root@ ~]# lspci | grep -i Chelsio
c7:00.0 Ethernet controller: Chelsio Communications Inc Device 7007
c7:00.1 Ethernet controller: Chelsio Communications Inc Device 7007
c7:00.2 Ethernet controller: Chelsio Communications Inc Device 7007
c7:00.3 Ethernet controller: Chelsio Communications Inc Device 7007
c7:00.4 Ethernet controller: Chelsio Communications Inc Device 7407
c7:00.5 SCSI storage controller: Chelsio Communications Inc Device 7507
c7:00.6 Fibre Channel: Chelsio Communications Inc Device 7607
```

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:

```
[  497.356490] cxgb4 0000:16:00.4: Chelsio T72200 rev 1
[  497.356493] cxgb4 0000:16:00.4: S/N: PT11260163, P/N: 11013067005
[  497.356494] cxgb4 0000:16:00.4: Firmware version: [redacted]
[  497.356496] cxgb4 0000:16:00.4: Bootstrap version: 2.0.4.0
[  497.356497] cxgb4 0000:16:00.4: TP Microcode version: 0.1.30.0
[  497.356498] cxgb4 0000:16:00.4: Expansion ROM version: 3.0.0.5
[  497.356499] cxgb4 0000:16:00.4: Serial Configuration version: 0x1100002
[  497.356501] cxgb4 0000:16:00.4: VPD version: 0x2
```

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

2.2. Installing the Chelsio T6 Adapters

Follow these steps to install the Chelsio T6 adapter in your system:

1. Shutdown/power off your system.
2. Power off all remaining peripherals attached to your system.
3. Unpack the Chelsio adapter and place it on an anti-static surface.
4. Remove the system case cover as per the system manufacturer's instructions.
5. Remove the PCI filler plate from the slot where you will install the Ethernet adapter.
6. For maximum performance, it is highly recommended to install the adapter into a PCIe x8/x16 slot.
7. 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.
8. Secure the Chelsio adapter with a screw, or other securing mechanism, as described by the system manufacturer's instructions. Replace the case cover.
9. 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.
10. 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.
11. Power on your system.
12. 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.
```

13. 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:
[ 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 appears as the first network interface. For these adapters, the port nearest to the motherboard will appear as the first network interface.*

3. Software/Driver Installation

Chelsio Unified Wire package can be installed using source only. The source package supports CLI mode.

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.	Network (NIC/TOE), Precision Time Protocol (PTP),Port Forwarding Mode (Hardware Bypass Mode/Hairpin Mode/LAN Bypass),Virtual Function Network (vNIC), iWARP RDMA Offload, RoCEv2,NVMe-oF iWARP (Target and Initiator),NVMe-oF RoCEv2 (Target),SPDK (user space) NVMe-oF iWARP (Target and Initiator),NVMe-oF TOE (Target and Initiator),SPDK (user space) NVMe-oF TOE (Target),NVMe-oF/TCP PDU Offload (Target and Initiator),SPDK (user space) NVMe/TCP PDU Offload (Target and Host),SPDK (user space) NVMe/TCP over Memfree TOE (Target and Initiator)*,SoftiWARP Initiator,LIO iSCSI Target Offload,iSCSI PDU Offload Initiator,SPDK (user space) iSCSI PDU Offload (Target and Initiator)*,SPDK (user space) iSCSI over Memfree TOE (Target and Initiator)*,Crypto Offload,Inline IPSec Offload,iSER (Target and Initiator),WD-UDP,Data Center Bridging (DCB),FCoE Full Offload Initiator,Offload Bonding,UDP Segmentation Offload and Pacing,Offload IPV6,WD Sniffing and Tracing,Classification and Filtering,Mesh Topology,Traffic Management feature (TM),Unified Boot Software,Utility Tools (cop,cxgbtool,t4_perftune), NC-SI (Network Controller Sideband Interface)

3.1. Pre-requisites

- Package Manager yum should be configured using any of the OS recommended ways to resolve and install missing packages.
- Install *python3* to run the Chelsio Unified Wire package scripts.
- The *make, gcc, kernel-devel, kernel-headers, openssl-devel, libibverbs, libibverbs-devel, librdmacm, librdmacm-devel, rdma-core-devel, cmake, libnl3, libnl3-devel, fuse3, fuse3-devel, fuse3-libs, numactl-devel, libuuid-devel, libaio-devel, ncurses-devel, pciutils-devel, meson, patchelf, wheel, pyelftools, libpsl-devel, uv, and ninja-build* packages should be installed for the compilation of drivers and utilities.
- Uninstall any OFED present in the machine.
- Installation must be attempted as 'root' user.
- Compilation of drivers can be attempted as non-root user using the following command:

```
[user@host ~]$ make KSRC="<kernel source path>" KOBJ="<kernel object path>"
```

OR

```
[user@host ~]$ make KDIR="<kernel directory path>"
```

Example 1:

```
[user@host ChelsioUwire-x.x.x.x]$ make KSRC=/home/linux-5.14.0-611.5.1.el9_7/
KOBJ=/lib/modules/5.14.0-611.5.1.el9_7.x86_64/build
```

Example 2:

```
[user@host ChelsioUwire-x.x.x.x]$ make KDIR=/usr/src/kernels/linux-6.12.95/
```

Example 3:

```
[user@host ChelsioUwire-x.x.x.x]$ make KSRC=/root/linux-source-6.8.0/
KOBJ=/lib/modules/6.8.0-136-generic/build
```

3.2. Installing Chelsio Unified Wire

3.2.1. GUI mode (with Dialog utility)

1. Download the Unified Wire driver package (tarball) from Chelsio Download Center.
2. Extract the downloaded package using the following command:

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

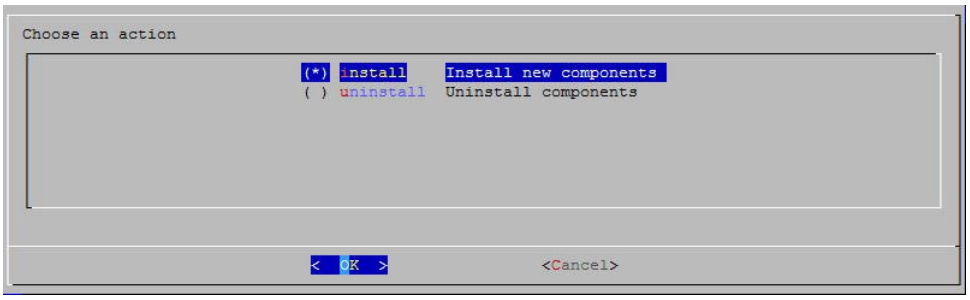
3. Change your current working directory to the Chelsio Unified Wire package directory.

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

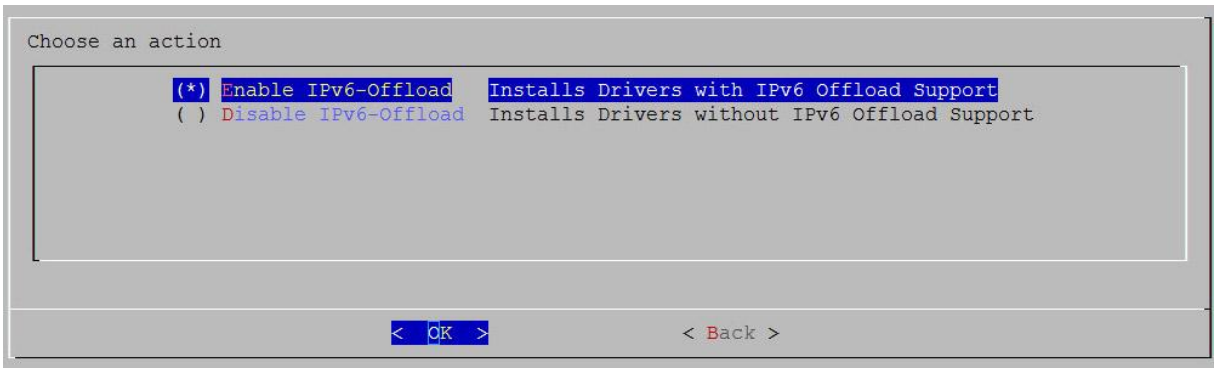
4. Run the following script to start the GUI installer:

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

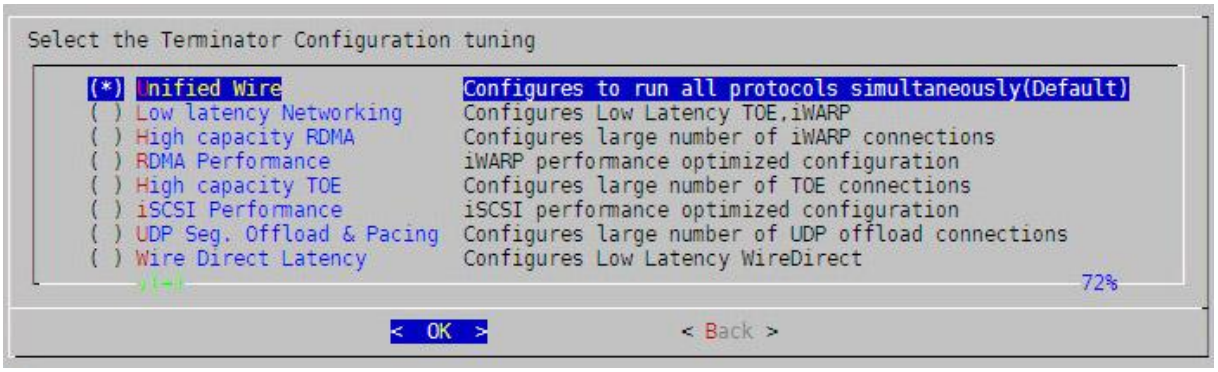
5. If **Dialog** utility is present, you can skip to step 6. If not, press *y* to install it when the installer prompts for input.
6. Select **install** under **Choose an action**.



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

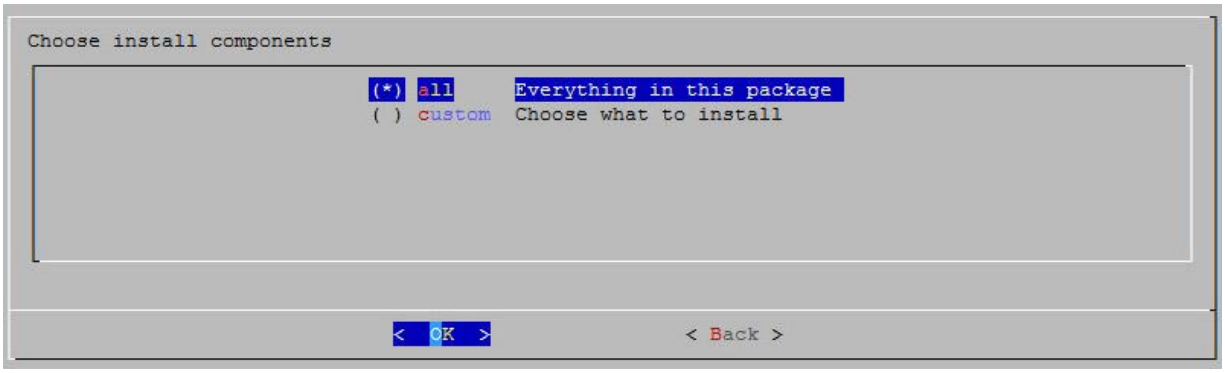


8. Select the required configuration tuning option.



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

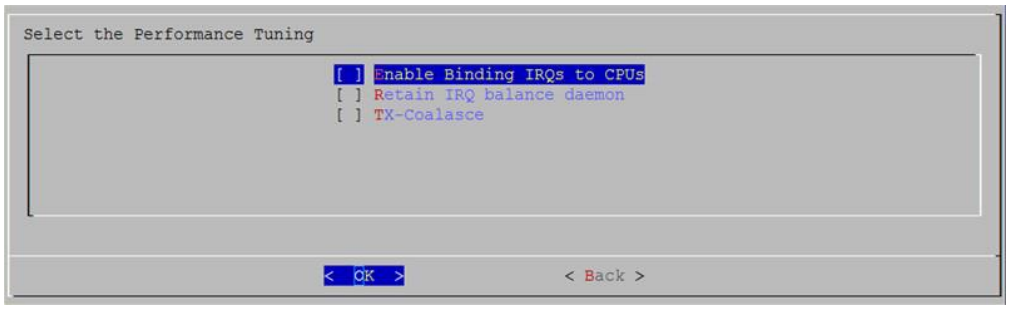
9. Under **Choose install components**, select **all** to install all the related components for the option chosen in step 8 or select **custom** to install specific components.



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

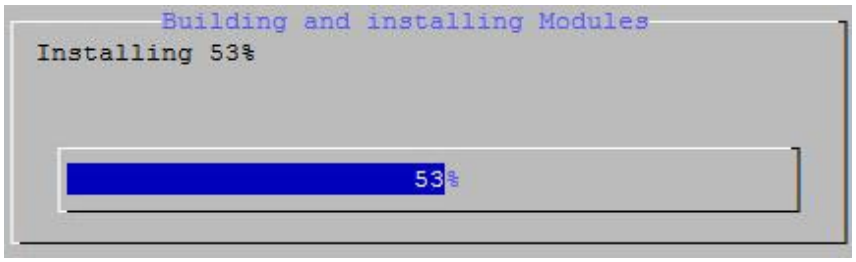
10. Select the required performance tuning option.
    i. Enable Binding IRQs to CPUs: Bind MSI-X interrupts to different CPUs and disable IRQ balance daemon.

ii. *Retain IRQ balance daemon*: Do not disable IRQ balance daemon.
iii. *TX-Coalesce*: Write tx_coal=2 to modprobe.d/conf.

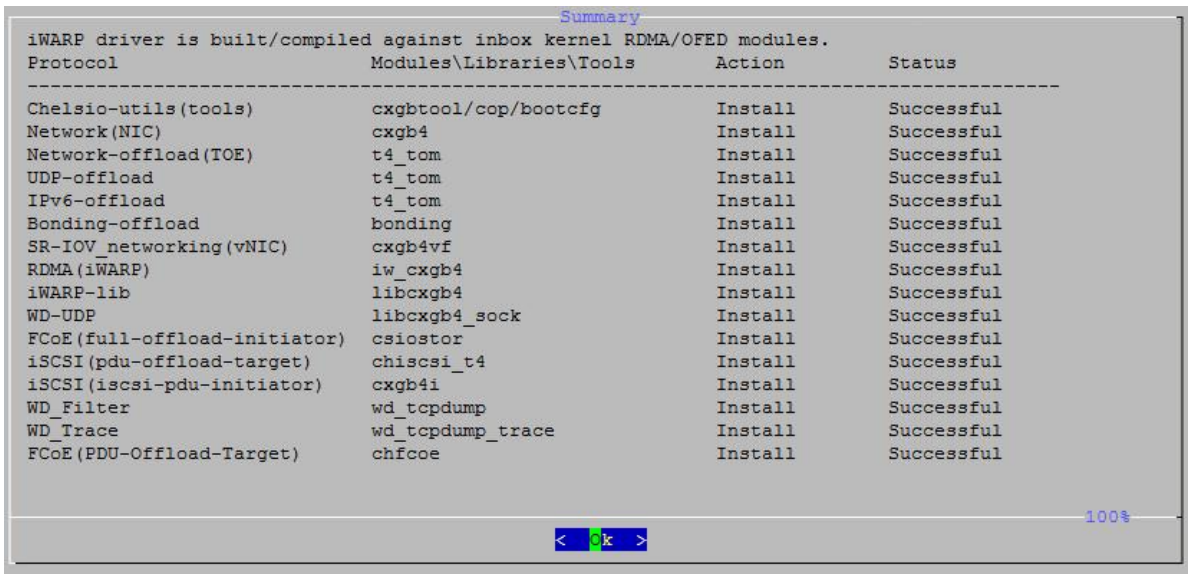


Note *For more information on the Performance tuning options, refer to the* Performance Tuning *section of the* ***Network (NIC/TOE)*** *chapter.*

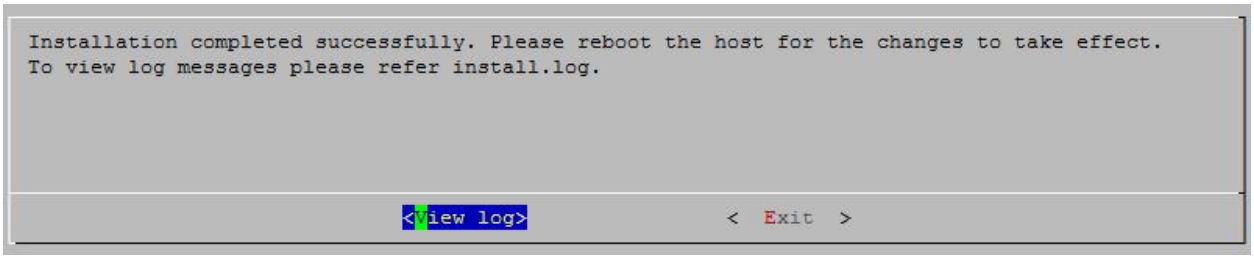
11. The selected components will now be installed.



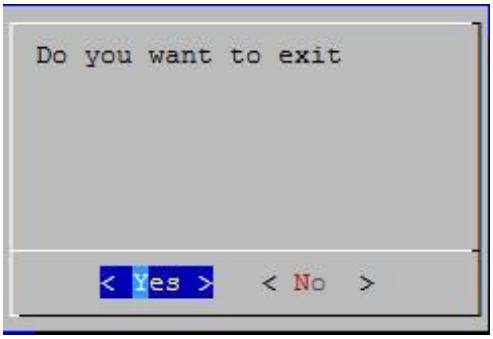
12. After successful installation, a summary of installed components are displayed.



13. Select **View log** to view the installation log or **Exit** to continue.



14. Select **Yes** to exit the installer or **No** to go back.



15. Reboot your machine for changes to take effect.

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

3.2.2. 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:

1. Download the Unified Wire driver package from the Chelsio Download Center.
2. Extract the downloaded package using the following command:

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

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

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

4. Run the following script to start the installer.

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

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

6. The selected components will now be installed.
   After successful installation, press *1* to view the installation log. Press any key to exit from the installer.

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

7. Reboot your machine for changes to take effect.

iWARP driver installation on Cluster nodes

Important *Ensure 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:

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

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

2. 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.

3. Now, execute the following command:

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

4. Select the required configuration tuning option. The tuning options may vary depending on the Linux distribution.
5. Select the required Cluster Configuration.
6. 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.

3.2.3. CLI mode

1. Download the tarball specific to your operating system and architecture from the Chelsio Download Center.
2. Extract the downloaded package. Example, for RHEL/Rocky/AlmaLinux 9.7:

```
[root@host~]# tar zxvfm ChelsioUwire-x.x.x.x-RHEL9.7-x86_64.tar.gz
```

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

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

4. 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.*

5. 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`

6. Reboot your machine for changes to take effect.

iWARP driver installation on cluster nodes

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

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

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

2. 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.

3. 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.

4. Reboot your machine for changes to take effect.

3.3. Firmware flash and init/vpd 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>
```

3.3.1. Firmware flash and init/vpd update

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

2. Extract the downloaded package using the following command.

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

3. Change your current working directory to the Chelsio Unified Wire package directory and build and install the drivers, tools and libraries from source.

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

Note *If a previous version of driver package is installed, execute the following commands before attempting installation:*

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

4. Unload NIC driver and its dependencies.

```
[root@host~]# rmmod iw_cxgb4 cxgb4
```

5. Flash the updated init and vpd on the Chelsio T7 adapter.

```
[root@host~]# t7seeprom -flash -b <pci-id of chelsio card>  write -f:< path
to seeprom.bin file >
```

Example:

- For T72200-FH:

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/t72200_fh_init_gen5x16_2_mem_2ch_4800_2x200g_vp
d_mfg.bin
```

- For T72200:

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/t72200_init_gen5x16_mem_2ch_4800_2x200g_vpd_mfg
.bin
```

- For T7450:

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/t7450_init_gen5x16_mem_2ch_4800_4x50g_vpd_mfg.b
in
```

- For S72200:

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/s72200_init_gen5x16_no_mem_2x200g_vpd_mfg.bin
```

- For S71400:

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/s71400_init_gen5x16_no_mem_1x400g_vpd_mfg.bin
```

- For S7450:

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/s7450_init_gen5x16_no_mem_4x40g_vpd_mfg.bin
```

- For S7250:

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/s7250_init_gen5x16_no_mem_4x40g_vpd_mfg.bin
```

6. Update the firmware on the Chelsio T7 adapter.
   Run the following commands to update the firmware manually:

```
 [root@host~]# modprobe cxgb4 fw_attach=0
 [root@host~]# cxgbtool ethX loadfw
/lib/firmware/cxgb4/config/t7fwbootstrap-2.0.4.0.bin
[root@host~]# cxgbtool ethX loadfw /lib/firmware/cxgb4/t7fw.bin
```

7. Power cycle your machine for changes to take effect.

8. Verify the firmware version after the host boots.
   The firmware (v2.1.x.x is installed on the system, typically in */lib/firmware/cxgb4*. The firmware version can be verified using:

```
[root@host~]# ethtool -i ethX
driver: cxgb4
...
firmware-version: 2.1.x.x,TP 0.1.30.0
```

3.4. Mounting debugfs

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

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

3.5. Collect all adapter logs using cudbg_app

To collect all adapter logs using *cudbg_app,* run the following command.

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

4. Configuring Chelsio Network Interfaces

To effectively test the features of Chelsio adapters, it is recommended to use two machines, each installed with Chelsio's network adapters. 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.

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, refer to the operating system documentation for manually configuring the network device.*

4.1. Configuring Chelsio T7 Adapters

T7 Adapters operate at a speed of **10/25/40/50/100/200/400GbE** with optic cables connected to Arista Switches with optics modules.

For the best performance, use JDSU optic module for 10G, Finisar optic module for 25G, SM100G optic module for 100G, and FS.COM optic module for 50G, 200G, and 400G.

Note *T7 Unified Wire adapters do not support auto-negotiation and FEC.*

4.2. Configuring Chelsio T6 Adapters

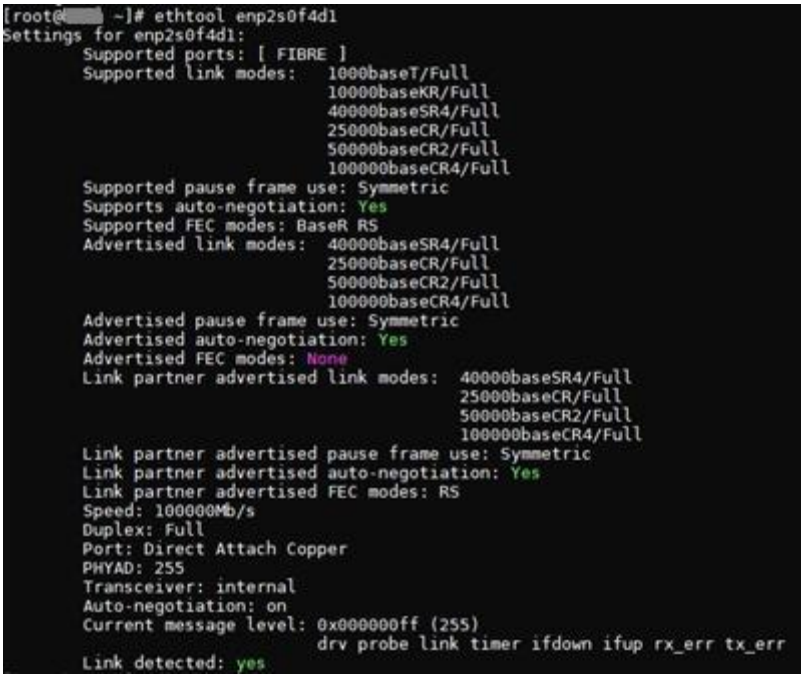
T6 Unified Wire adapters support auto-negotiation (enabled by default) which allows link parameters like speed, duplex, FEC, and Pause to be negotiated with the PEER.

4.2.1. 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 *Ensure that ethtool v4.8 or higher is required.*

Below is a sample output for T6 100G port:



- **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:

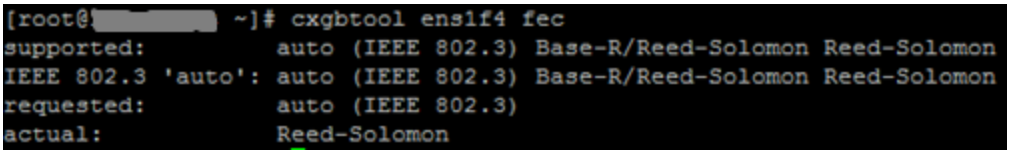
```
[root@host~]# ethtool -s <ethX> speed <speed> autoneg off
```

4.2.2. 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
```

The following is a sample output on the T6 100G port:



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 following 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.*

4.2.3. 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
```

4.2.4. Spider and QSA Modes

- **T6 Adapters**

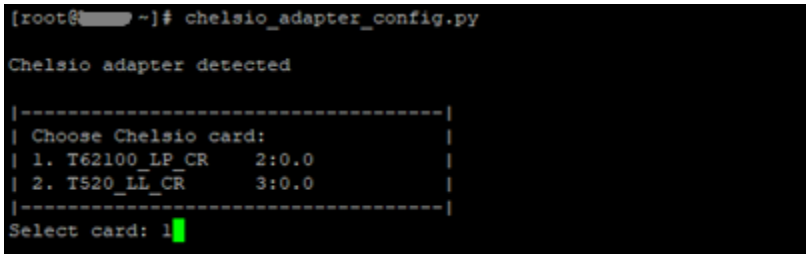
Chelsio T6 100G adapters can be configured in the following two modes:

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

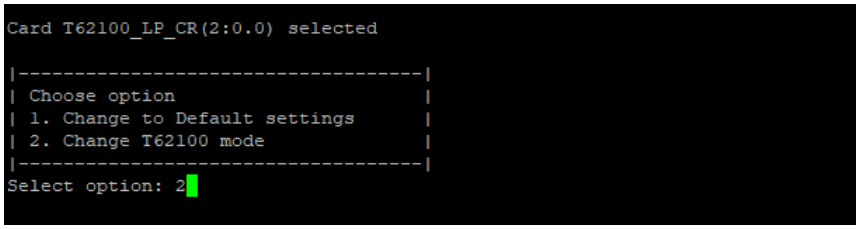
Note *QSA modules work in the default mode.*

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

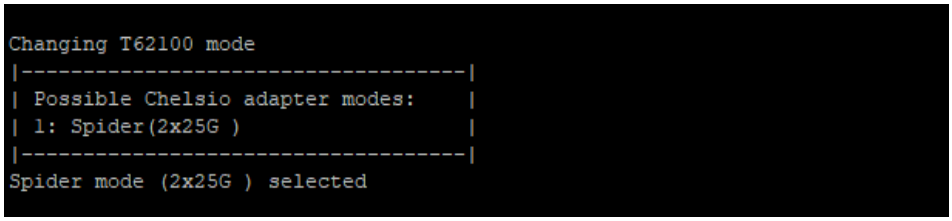
1. Run the *chelsio_adapter_config.py* command to detect all the Chelsio adapter(s) present in system. Select the adapter to configure by specifying the adapter index.



2. Select *Change adapter mode.*



3. Select the required mode.



4. Reload the network driver to make the changes.

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

Note *If the default option is selected in step 2, reboot the machine to apply the changes.*

4.3. Configuring NetworkManager keyfiles

A typical interface NetworkManager keyfiles (Example: enp3s0f4) on RHEL 9.X looks like the following:

```
# cat /etc/NetworkManager/system-connections/enp3s0f4.nmconnection
[connection]
id=enp3s0f4
uuid=a52c3985-3fc3-41cd-bec0-77c19fab5cae
type=ethernet
autoconnect=true
interface-name=enp3s0f4
timestamp=1740991901

[ethernet]
mac-address=00:07:43:3d:d9:e0

[ipv4]
address1=103.1.1.77/24
method=manual

[ipv6]
addr-gen-mode=eui64
dhcp-timeout=90
method=link-local
```

In the case of DHCP addressing use the following template:

```
# cat /etc/NetworkManager/system-connections/enp3s0f4.nmconnection
[connection]
id=enp3s0f4
uuid=a52c3985-3fc3-41cd-bec0-77c19fab5cae
type=ethernet
autoconnect=true
interface-name=enp3s0f4
timestamp=1740991901

[ethernet]
mac-address=00:07:43:3d:d9:e0

[ipv4]
method=auto

[ipv6]
addr-gen-mode=eui64
dhcp-timeout=90
method=link-local
```

4.4. Creating Network Manager keyfiles manually

To spot the new interfaces, ensure 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 must display the adapter interfaces as:

```
[root@host~]# ifconfig -a | grep HWaddr
eth0 Link encap:Ethernet HWaddr 00:30:48:32:6A:AA
enp3s0f4 Link encap:Ethernet HWaddr 00:07:43:3D:D9:E0
enp3s0f4d1 Link encap:Ethernet HWaddr 00:07:43:3D:D9:E8
```

For each interface you can write a configuration file in */etc/NetworkManager/system-connections*. The *enp3s0f4.nmconnection* could look like:

```
# cat /etc/NetworkManager/system-connections/enp3s0f4.nmconnection
[connection]
id=enp3s0f4
uuid=a52c3985-3fc3-41cd-bec0-77c19fab5cae
type=ethernet
```

```
autoconnect=true
interface-name=enp3s0f4
timestamp=1740991901

[ethernet]
mac-address=00:07:43:3d:d9:e0

[ipv4]
address1=103.1.1.77/24
method=manual

[ipv6]
addr-gen-mode=eui64
dhcp-timeout=90
method=link-local
```

Reload and bring up:

```
# nmcli connection reload
# nmcli connection up enp3s0f4
```

4.5. Configuring IPv6

The interfaces should come up with a link-local IPv6 address for complete and fully functional IPv6 configuration. Update the Interface NetworkManager keyfile with *method=link-local*.

4.6. Checking Link

Once the NetworkManager keyfiles are created for the interfaces you should check the link. That is, ensure 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 T6 adapter into a PCIe Gen3 x8/x16 slot. Ensure that **T7 400GbE adapters** are placed in PCIe **Gen5 x16** slots and not in x8_in_x16 slots.
- Ensure that the system is populated with the balanced memory configuration (Refer to the system manual for the recommended configurations). At least one DIMM per channel should be populated for maximum performance.
- Make sure the following BIOS settings are configured:
  1. Disable Hyper-Threading Technology.
  2. Disable Sub-NUMA Clustering (SNC).
  3. Disable virtualization, C-states technology, VT-d, Intel I/O AT, and SR-IOV.
  4. Disable CPU Power Management Features including C-states, P-states, and Turbo Boost.
  5. Configure System power management profile in Maximum Performance Mode.
- 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 following 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 following tuned-adm profile:

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

- 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, refer to their respective chapters.

6. Software/Driver Update

For any distribution-specific problems, check **README** and **User Guide** (this document) included in the release for possible workarounds.

Visit the 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 the source code. By using the source, you can uninstall the package using the CLI mode.

7.1. Uninstalling Chelsio Unified Wire from source

7.1.1. CLI mode

1. Change your current working directory to the *Chelsio Unified Wire package* directory.

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

2. Uninstall using the following command:

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

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-up CPU resources can be used for other computing tasks. 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S71400
  - S72200
  - S7450
  - S7250
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T62100-SO-CR
  - T62100-SO-OCP3
  - T6225-CR
  - T6225-SO-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR
  - T520-SO-CR

Note *The S71400, S72200, S7450, S7250, T62100-SO-CR, T62100-SO-OCP3, T6225-SOCR, and T520-SO-CR adapters support only NIC functionality.*

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the Network driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Software/Driver Installation

Change your current working directory to the 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, run* `make help` *or* `install.py -h`.

Reboot your machine for changes to take effect.

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

3. Software/Driver Loading

Important *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.*

```
[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 now offloaded over the Chelsio interface. 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. Precision Time Protocol (PTP)

The Precision Time Protocol (PTP) standard defines a protocol for precise synchronization of a 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.

4.2.1. Synchronizing Clocks

The *ptp4l* tool is used to synchronize clocks.

Note *The "linuxptp" package must be installed before proceeding.*

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

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

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

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

4. On the master, synchronize the PTP hardware clock with the system clock.

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

5. 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 to the ptp4l help manual.*

6. On the slave, synchronize the system clock with the PTP hardware clock.

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

4.3. 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 when loading the driver.

4.3.1. Host Configuration

1. Load the network driver and vxlan driver.

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

2. 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
```

3. 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
```

4. Bring up the VXLAN interface.

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

5. Create the bridge interface and bring it up.

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

6. Add the VXLAN interface to the bridge interface.

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

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

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

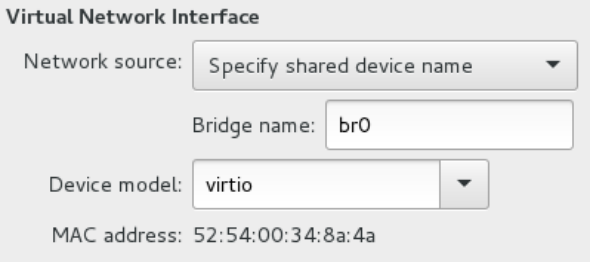
8. For better performance, configure the NIC settings of the Performance Tuning section.

4.3.2. Guest (VM) Configuration

1. Open the Virtual Machine Manager.

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

2. Add a Virtual Network Interface to the VM, by specifying the Bridge name configured in Step 5. of the Host Configuration section and Device Model as *virtio*.



3. 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:

1. 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'/>
```

2. 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>
```

3. 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
```

4. Bind the Virtual Network Interface Queues to different CPUs.

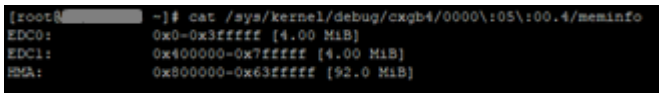
5. Increase the TCP buffers by configuring the sysctl variables mentioned in NIC settings of the **Performance Tuning** section.

4.4. HMA

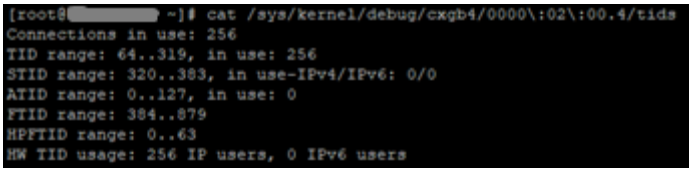
To use HMA, ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option. Currently, 256 IPv4/128 IPv6 TOE connections are supported on T6 25G SO adapters.

Note *The HMA feature is applicable only for T6 adapters.*

The following image shows the HMA reserved memory.



The following image shows the number of TOE offloaded connections.



4.5. Performance Tuning

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

- **TOE**

1. Run the performance tuning script to update few kernel parameters using sysctl and map TOE queues to different CPUs.

```
[root@host~]# t4_perftune.sh -i ethX -s
```

2. 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**

1. Run the performance tuning script to update few kernel parameters using sysctl and map NIC queues to different CPUs.

```
[root@host~]# t4_perftune.sh -i ethX -s
```

2. Enable adaptive-rx.

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

- **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 following sysctl:

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

- **Receiver Side Scaling (RSS)**

The Receiver Side Scaling (RSS) 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/T6 devices ...
Configuring Chelsio T4/T5/T6 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 a 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 the 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 10/25/40/50/100/200/400 Gbps, 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
```

- **Generic Receive Offload**

Generic Receive Offload (GRO) is a performance improvement feature at the receiving side. It aggregates the received packets that belong to the 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 the same amount of data. This helps to reduce the receive host CPU load and improve throughput in a 10/25/40/50/100/200/400Gb network environment where CPU can be the bottleneck. It is available from kernel 2.6.29 onwards.

Chelsio's card supports both hardware assisted GRO and Linux-based GRO. `t4_tom` is the kernel module that enables the hardware assisted GRO. 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 implementation is enabled.

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

```
[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 [fixed]
```

Note that `generic-receive-offload` option is ON. This means GRO is enabled. 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 into larger packets. By default, this number is zero.

`GROMerged` is the number of packets that are 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 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 [fixed]
```

The output above shows a disabled GRO.

Note that if your Linux system has IP forwarding enabled. That is acting as a bridge or router, the GRO needs to be disabled. This is due to a known kernel issue.

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 steps below:

1. Load *t4_tom* driver with the *unsupported_allow_unload* parameter.

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

2. 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 3.

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

a. Create a policy file which disables 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
```

3. Unload the driver.

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

III. Port Forwarding Mode (Hardware Bypass Mode/Hairpin Mode/LAN Bypass)

1. Introduction

Port forwarding enables automatic packet forwarding between ports of the same adapter through firmware, even when the driver is not loaded. This feature is intended to support packet forwarding during driver absence, driver unload, or driver failure.100% hair-pin is supported by which all traffic from Port0 is sent to Port1, and traffic from Port1 is sent to Port0.

Following modes are supported:

- Hardware bypass on power-on
- Hardware bypass with driver deadman watchdog

Hardware bypass with IOCTL-Based deadman watchdog.

Note *To enable this feature please flash the latest init/vpd present in this package, refer "**Firmware and init/vpd update**" section in this chapter for steps.*

Note *Port Forwarding Mode (Hardware Bypass Mode/Hairpin Mode/LAN Bypass) is provided as an Alpha quality release.*

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S72200

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the Network driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.6.144

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

2. Software/Driver Installation

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

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

- To build and install driver,

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

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

Reboot your machine for changes to take effect.

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

3. Firmware and init/vpd update

Note *The init/VPD files are only available in the source software package; while the 't7seeprom' utility used to flash init/VPD is available in both the source and RPM packages.*

1. Download Chelsio Unified Wire driver package.
2. Extract the downloaded package.

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

3. Change your current working directory to the Chelsio Unified Wire package directory and build and install the drivers, tools and libraries from source.

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

4. Unload NIC driver and its dependencies.

```
[root@host~]# rmmod iw_cxgb4 cxgb4
```

5. Flash the updated init and vpd on the Chelsio T7 adapter.

```
[root@host~]# t7seeprom -flash -b <pci-id of chelsio card>  write -f:< path
to seeprom.bin file >
```

Example:

- For T72200-FH:

For Gen5 -

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config5
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t72200_fh_init_gen5x16_FW_wire_mode_2_mem_2ch_4800_2x200g_vpd_mfg.bin
```

For Gen4 -

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config5
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t72200_fh_init_gen4x16_FW_wire_mode_2_mem_2ch_4800_2x200g_vpd_mfg.bin
```

- For T72200:

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config5
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t72200_init_gen5x16_FW_wire_mode_mem_2ch_4800_2x200g_vpd_mfg.bin
```

- For T7450:

For Gen5 -

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config5
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t7450_init_gen5x16_FW_wire_mode_mem_2ch_4800_4x50g_vpd_mfg.bin
```

For Gen4 -

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config5
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t7450_init_gen4x16_FW_wire_mode_mem_2ch_4800_4x50g_vpd_mfg.bin
```

- For S72200:

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config5
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:s72200_init_gen5x16_FW_wire_mode_no_mem_2x200g_vpd_mfg.bin
```

6. Update the firmware on the Chelsio T7 adapter.
   Run the following commands to update the firmware manually:

```
[root@host~]# modprobe cxgb4 fw_attach=0
[root@host~]# cxgbtool ethX loadfw /lib/firmware/cxgb4/config/t7fwbootstrap-
x.x.x.x.bin
[root@host~]# cxgbtool ethX loadfw /lib/firmware/cxgb4/t7fw.bin
```

7. Power cycle your machine for changes to take effect.
8. Verify the firmware version (v2.1.x.x) after the host boots.
   The firmware (v2.1.x.x) is installed on the system, typically in */lib/firmware/cxgb4*. The firmware version can be verified using:

```
[root@host~]# ethtool -i ethX
 driver: cxgb4
 ...
 firmware-version: 2.1.x.x,TP 0.1.30.0
 ...
```

4. Software/Driver Loading

Important *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.

Please refer to the Deadman watchdog section for further details.

5. Hardware bypass on power-on

Upon adapter power-on, hardware bypass is enabled by default.

After power-up:

- Firmware will bring up the links automatically
- Firmware will configure switch filters automatically
- Firmware will start packet forwarding between ports without waiting for the driver to load as below:
    - On a dual-port adapter:
        - Packets from Port0 are forwarded to Port1
        - Packets from Port1 are forwarded to Port0
    - On a quad-port adapter:
        - Packets from Port0 are forwarded to Port1
        - Packets from Port1 are forwarded to Port0
        - Packets from Port2 are forwarded to Port3
        - Packets from Port3 are forwarded to Port2

6. Hardware bypass with driver deadman watchdog

Once the cxgb4 driver registers, hardware bypass is disabled and firmware-based packet forwarding is disabled.

Hardware bypass is re-enabled under the following circumstances:

- cxgb4 driver unregisters/unloads
- cxgb4 driver stops sending deadman watchdog updates
- A driver or firmware crash occurs

In order to enable hardware bypass with deadman watchdog feature follow the steps below:

```
[root@host~]# modprobe -v cxgb4 deadman_watchdog=x,2
[root@host~]# ifconfig ethA up
[root@host~]# ifconfig ethB up
```

Note

Here,

x is the deadman watchdog timer interval in milliseconds ranging from 1000ms to 60000ms.

ethA refers to Port0 of the adapter.

ethB refers to Port1 of the adapter.

Example #1:

For a dual port adapter:

```
[root@host~]# modprobe -v cxgb4 deadman_watchdog=2000,2
[root@host~]# ifconfig eth0 up
[root@host~]# ifconfig eth1 up
```

Note

Here, driver sends watchdog sync every 1000ms (2000ms/2) to the firmware. If the firmware fails to receive a sync then it switches to hardware bypass mode and packets are forwarded from port0 to port1 and port1 to port0.

Example #2:

For a quad port adapter:

```
[root@host~]# modprobe -v cxgb4 deadman_watchdog=2000,2
[root@host~]# ifconfig eth0 up
[root@host~]# ifconfig eth1 up
[root@host~]# ifconfig eth2 up
[root@host~]# ifconfig eth3 up
```

Note *Here, driver sends watchdog sync every 1000ms (2000ms/2) to the firmware. If the firmware fails to receive a sync then it switches to hardware bypass mode and packets are forwarded from port0 to port1, port1 to port0, port2 to port3 and port3 to port2.*

7. Hardware bypass with IOCTL-Based deadman watchdog

The driver provides a private IOCTL interface that enables a userspace application to configure, start, and periodically refresh the driver watchdog. The application configures the watchdog by issuing an ioctl() request using SIOCDEVPRIVATE on the target network interface. The watchdog configuration is passed through a dedicated structure that includes the watchdog operation command identifier (cmd) and the watchdog timeout value (timeout) in milliseconds. Once configured, the watchdog remains active as long as the application continues to periodically refresh it using the same IOCTL interface.

To configure the watchdog, the application must first create a network socket and populate the watchdog configuration structure with the required command and timeout parameters. An ifreq structure must then be initialized with the target network interface name and a pointer to the watchdog configuration structure assigned to ifr_data. The application should invoke ioctl() with SIOCDEVPRIVATE to start the watchdog and continue issuing the same IOCTL periodically to refresh it. The refresh interval must be set to half of the configured watchdog timeout value to ensure the watchdog is refreshed well before expiration and to avoid unintended timeout events. For example, with a configured watchdog timeout of 20 seconds, the refresh IOCTL should be issued every 10 seconds.

The 'cxgb4' driver provides support for IOCTL-based hairpin mode. Once the driver is loaded and the network interfaces are brought up, the application can configure the driver timer using 'set_timer' and periodically refresh it using 'refresh_timer'. If the application crashes or stops refreshing the timer, the timer will eventually expire, and the adapter will automatically switch to hairpin mode.

7.1. cxgb4 driver parameter

To enable hairpin mode immediately when the interface is brought up after driver load, the 'cxgb4' driver must be loaded with the following driver parameter:

```
[root@host~]# modprobe cxgb4 hairpin_enable=1
```

Note *The 'hairpin_enable=1' parameter is required only when hairpin mode needs to be enabled immediately during interface bring-up after driver load.*

This parameter is _not required_ for the timer-based application-failure mechanism. Even if 'hairpin_enable=1' is not specified, the application can configure 'set_timer' and periodically call 'refresh_timer'. If the application subsequently crashes or stops refreshing the timer, the timer will expire and the adapter will automatically switch to hairpin mode.

7.2. Reference Application

The **test_app_v2** is provided as an example/reference application to demonstrate how to use the driver-based hairpin functionality. End-users can use the **test_app_v2** implementation and usage as a reference and modify their application accordingly to integrate and support driver-based hairpin functionality.

Location

The **test_app_v2.c** application can be found at:

```
ChelsioUwire-x.x.x.x/support/config5/test_app_v2.c
```

Compilation of the test application

```
[root@host ~]# gcc test_app_v2.c -o test_app_v2
```

Usage

```
[root@host~]# ./test_app_v2 <ifname> set_timer <time in ms>
[root@host~]# ./test_app_v2 <ifname> refresh_timer <time in ms>
[root@host~]# ./test_app_v2 <ifname> del_timer
[root@host~]# ./test_app_v2 <ifname> port_mode
[root@host~]# ./test_app_v2 <ifname> hairpin
[root@host~]# ./test_app_v2 <ifname> default
```

Command description

- **set_timer <time in ms>**: Configures the driver timer. The supported range is 1000–60000 ms.
- **refresh_timer <time in ms>**: Refreshes the configured driver timer.
  - **set_timer** must be configured before starting the timer refresh operation using **refresh_timer**.
    - Once the timer is configured and periodically refreshed, the adapter will automatically switch to hairpin mode if the application crashes or stops refreshing the timer and the timer expires.
    - The **refresh_timer** must be called after configuring **set_timer** and before the timer expires. **refresh_timer** is used to maintain a heartbeat mechanism between the application and the driver. It should be called at regular intervals to inform the driver that the application is still alive. It is recommended to call **refresh_timer** within half of the configured timeout period; for example, if the timer is configured for 60 seconds, **refresh_timer** should be called within 30 seconds.
- **del_timer**: Deletes the configured timer.
  This removes the timer used to trigger the switch to hairpin mode. Therefore, even if the application crashes, the adapter will not switch to hairpin mode due to the timer mechanism.

- **port_mode**: Displays the current adapter mode and timer status, including:
  - **Current adapter mode**: NORMAL/HAIRPIN
  - **Configured timer timeout**: in ms
  - **Last timer refresh status**: SUCCESS/FAILURE/NEVER REFRESHED
  - **Timer state**: ACTIVE/INACTIVE/EXPIRED
- **hairpin**: Immediately switches the adapter to hairpin mode.
- **default**: Immediately switches the adapter back to normal/default mode.

8. Switching back to Normal Mode

To disable Port Forwarding Mode (Hardware Bypass Mode/Hairpin Mode/LAN Bypass) and return the adapter to normal operation, flash the normal mode init/vpd on the adapter in place of the port forwarding (FW_wire_mode) init/vpd, and then power cycle the machine.

Note: The 'default' command of test_app_v2 only switches the adapter back to normal mode at run time. It does not change the init/vpd flashed on the adapter, so hardware bypass will be enabled again on the next power-on as long as the port forwarding init/vpd remains flashed. To permanently switch back to normal mode, follow the steps below.

1. Make sure the Chelsio Unified Wire package is built and installed on the host. Refer to steps 1 to 3 of the "Firmware and init/vpd update" section in this chapter for details.

2. Unload NIC driver and its dependencies.

```
[root@host~]# rmmod iw_cxgb4 cxgb4
```

3. Flash the normal mode init and vpd on the Chelsio T7 adapter.

```
[root@host~]# t7seeprom -flash -b <pci-id of chelsio card> write -f:< path
to seeprom.bin file >
```

   The Gen5 normal mode init/vpd files are installed in /lib/firmware/cxgb4/config. The Gen4 normal mode init/vpd files are available in the source package at ChelsioUwire-x.x.x.x/support/config5.

   Example:

   - For T72200-FH:

   For Gen5 -

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/t72200_fh_init_gen5x16_2_mem_2ch_4800_2x200g_vp
d_mfg.bin
```

   For Gen4 -

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config5
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t72200_fh_init_gen4x16_2_mem_2ch_4800_2x200g_vpd_mfg.bin
```

   - For T72200:

For Gen5 -

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/t72200_init_gen5x16_mem_2ch_4800_2x200g_vpd_mfg
.bin
```

- For T7450:

For Gen5 -

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/t7450_init_gen5x16_mem_2ch_4800_4x50g_vpd_mfg.b
in
```

For Gen4 -

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config5
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t7450_init_gen4x16_mem_2ch_4800_4x50g_vpd_mfg.bin
```

- For S72200:

For Gen5 -

```
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:/lib/firmware/cxgb4/config/s72200_init_gen5x16_no_mem_2x200g_vpd_mfg.bin
```

4. Power cycle your machine for changes to take effect.

   After the power cycle, the adapter operates in normal mode and firmware no longer forwards packets between the ports when the driver is not loaded.

9. Software/Driver Unloading

9.1. Unloading the Driver

To unload the NIC driver,

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

IV. Virtual Function Network (vNIC)

1. Introduction

The ever-increasing network infrastructure of IT enterprises has led 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. Users 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T62100-SO-CR
  - T62100-SO-OCP3
  - T6225-CR
  - T6225-SO-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR
  - T520-SO-CR

1.2. Software Requirements

1.2.1. Linux Requirements

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

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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 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** through Hypervisor VM initiation commands.

2.1. Pre-requisites

Ensure that the following requirements are met before driver 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

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

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

2. On the host, install network driver.

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

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

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

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

4. 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, refer to 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 *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 can 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:

1. 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
```

2. The 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
```

3. 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 communicate 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. Refer to the above section for VF to physical port mappings. To override this and always enable the VF link, follow the below procedure. This enables VF to VF communication irrespective of the physical port link status.

1. 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
```

2. 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
```

3. 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 the 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 the 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).

1. 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.

2. 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:

1. Four VFs are instantiated on PF0.

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

2. Two VMs are configured with two VFs each. Two 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)
```

3. 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):

1. Load the Virtual Function network driver.

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

2. Create a bond interface.

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

3. 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.*

4. 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:

1. Two 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
```

2. One 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)
```

3. 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 failover to enp8s1f5d1.

4.5. High Capacity VF Configuration

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

Important *This feature is currently supported only on T6225-SO-CR adapter.*

1. Change your current working directory to the 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, run* `make help` *or* `install.py -h`.

2. Update the 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
```

3. 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}.

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

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

5. The virtual functions can now be assigned to virtual machines (guests).
6. For each PF on which VFs are instantiated, one 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
```

7. To set a VLAN ID on Virtual Function,

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

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

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

8. 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 may fail.

To unload the driver, execute the following command:

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

V. 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 the 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the iWARP driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104*
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* Kernel must be compiled on RHEL 10.X/9.8/9.7/9.6.

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

2. Software/Driver Installation

2.1. Pre-requisites

Ensure that the following requirements are met before driver installation:

- Uninstall any OFED present in the machine.
- *The rdma-core-devel* package should be installed on RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4 systems.

2.2. Installation

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

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

2. Install iWARP drivers and libraries.

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

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

3. Reboot your machine for changes to take effect.

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

3. Software/Driver Loading

Important *Ensure that all inbox drivers are unloaded before proceeding with the 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, 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, and core RDMA modules as mentioned in the 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 through 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 the user to remotely log on or executing commands without a password.

Quick steps to set up user authentication:

1. Change to the user home directory.

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

2. Generate authentication key.

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

3. Press *[Enter]* upon prompting to accept the default setup and empty password phrase.
4. Create an authorization file.

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

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

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

4.2.2. Installation of libfabrics for Open MPI 5.X or higher version

Open MPI iWARP support is only available in Open MPI version 5.X or greater.

Follow the steps mentioned below to install libfabrics:

1. Download **libfabric-2.3.0.tar.bz2** from https://github.com/ofiwg/libfabric.
2. Extract it using the following command:

```
[root@host~]# tar -xvf libfabric-2.3.0.tar.bz2
```

3. Navigate to **libfabric-2.3.0** folder.

```
[root@host~]# cd libfabric-2.3.0/
```

4. Configure libfabrics using the following command:

```
[root@host~]# ./configure --prefix=/usr/local
```

5. Install libfabrics using the following command:

```
[root@host~]# make -j$(nproc) && make install
```

4.2.3. Installation of OpenMPI 5.X or higher version with libfabrics support

Follow the steps mentioned below to install OpenMPI with libfabrics support:

1. Download **openmpi-5.0.8** file from https://download.open-mpi.org/release/open-mpi/v5.0/openmpi-5.0.8.tar.bz2.
2. Extract it using the following command:

```
[root@host~]# tar xvf openmpi-5.0.8.tar.bz2
```

3. Navigate to **openmpi-5.0.8** folder.

```
[root@host~]# cd openmpi-5.0.8
```

4. Configure **OpenMPI** using the following command:

```
[root@host~]# ./configure CC=gcc CXX=g++ F77=gfortran FC=gfortran --
prefix=/usr/mpi/gcc/openmpi-5.0.8 --libdir=/usr/mpi/gcc/openmpi-5.0.8/lib64
--with-libfabric=/usr/local --with-libfabric-libdir=/usr/local/lib --enable-
mpi-thread-multiple --enable-mpirun-prefix-by-default --with-contrib-vt-
flags=--disable-iotrace --enable-mca-no-build=btl-uct
```

5. Install **OpenMPI** using the following command:

```
[root@host~]# make -j$(nproc) && make install
```

4.2.4. Installation of IMB-MPI benchmark

Follow the steps mentioned below to install IMB-MPI benchmark:

1. Download **IMB-v2021.8** file from https://github.com/intel/mpi-benchmarks/archive/refs/tags/IMB-v2021.8.tar.gz.
2. Extract it using the following command:

```
[root@host~]# tar -xvf mpi-benchmarks-IMB-v2021.8
```

3. Edited */root/mpi-benchmarks-IMB-v2021.8/src_cpp/Makefile* as follows.

```
#ifeq ($(origin CC),default)
#CC=mpiicx
#endif
#ifeq ($(origin CXX),default)
#CXX=mpiicpx
#endif
#override CPPFLAGS += -I.
#override CFLAGS += -g -O0 -Wall -Werror
#override CXXFLAGS += -g -O0 -Wall -Wextra -Werror
#ifdef WITH_OPENMP
#ifeq ("$(CXX)",$(filter "$(CXX)", "mpiicpx" "icpx"))
#override CXXFLAGS += -qopenmp
#endif
#ifeq ("$(CC)",$(filter "$(CC)", "mpiicx" "icx"))
#override CFLAGS += -qopenmp
#endif
#ifeq ("$(CXX)",$(filter "$(CXX)", "mpiicpc" "icpc"))
#override CXXFLAGS += -qopenmp
#endif
#ifeq ("$(CC)",$(filter "$(CC)", "mpiicc" "icc"))
#override CFLAGS += -qopenmp
#endif
#ifeq "$(CXX)" "g++"
#override CXXFLAGS += -fopenmp
#endif
#ifeq "$(CC)" "gcc"
#override CFLAGS += -fopenmp
#endif
#ifeq "$(CXX)" "mpicxx"
#override CXXFLAGS += -fopenmp
#endif
#ifeq "$(CC)" "mpicc"
#override CFLAGS += -fopenmp
#endif
#endif
# ICC ICPC
```

```
ifeq ($(origin CC),default)                t
CC=/usr/mpi/gcc/openmpi-5.0.8/bin/mpicc
endif
ifeq ($(origin CXX),default)
CXX=/usr/mpi/gcc/openmpi-5.0.8/bin/mpicxx
endif
override CPPFLAGS += -I.
override CFLAGS = -g -Wall -Wextra -pedantic -Wno-long-long
CXXFLAGS = -g -Wall -Wextra -pedantic -Wno-long-long
```

4. Build and install IMB-MPI1 using the below command:

```
[root@host~]# make IMB-MPI1
```

5. Create *IMB* folder using the following command:

```
[root@host~]# mkdir -p /usr/mpi/gcc/openmpi-5.0.8/tests/IMB/
```

6. Copy the files using the following command:

```
[root@host~]# cp IMB-MPI1 /usr/mpi/gcc/openmpi-5.0.8/tests/IMB/
```

4.2.5. Running Open MPI

Perform the following the steps to run Open MPI:

1. Load **cxgb4, iw_cxgb4,** and **rdma_ucm** and then bring up the interface.
2. Edit */root/hosts.txt* file with machine IPs.

```
root@host~]# cat /root/hosts.txt
10.193.80.216 slots=8
10.193.80.121 slots=8
```

3. Run the following command:

```
[root@host~]# /usr/mpi/gcc/openmpi-5.0.8/bin/mpirun -np 8 --hostfile
/root/hosts.txt --allow-run-as-root --map-by node --bind-to core --mca pml
cm --mca mtl ofi --mca mtl_ofi_provider_include verbs --mca osc rdma --mca
btl self,vader --mca mtl_ofi_use_device_rdma 1 /usr/mpi/gcc/openmpi-
5.0.8/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:

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

```
/share/rdma1     *(fsid=1,rw,async,insecure,no_root_squash)
/share/rdma2     *(fsid=2,rw,async,insecure,no_root_squash)
```

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

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

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

4. Start the *nfs-server* service.

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

   All services in NFS should start without errors.

5. 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
```

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

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

Now the NFS-RDMA server is ready.

- **Client-side settings**

Follow the steps mentioned below on the client side:

1. Load the iwarp modules and make sure peer2peer is set to 1. Make sure that you are able to ping and ssh to the server Chelsio interface through which directories are exported.
2. Load the `xprtrdma` module.

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

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

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

4. Once the exported directories are listed, mount the directories.

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

To use NFSv4, specify vers=4.

4.4. HMA

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

Note *The HMA feature is applicable only for T6 adapters.*

The following image shows the HMA reserved memory.

```
[root@[redacted] ~]# 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@[redacted] ~]# 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

1. Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.
2. Run the performance tuning script to update few kernel parameters using sysctl and map iWARP queues to different CPUs.

```
[root@host~]# t4_perftune.sh -i ethX -s
```

5. Software/Driver Unloading

To unload the iWARP driver, run the following command:

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

VI. RoCEv2

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.

RoCEv2 (RDMA over Converged Ethernet v2) enables low-latency, high-throughput data transfer by allowing direct memory access over standard Ethernet networks. It operates over UDP/IP at Layer 3, making it routable and scalable across modern data center networks. RoCEv2 reduces CPU overhead and improves performance.

Note *This is supported only on T7 family of adapters.*

Note *RoCEv2 is provided as an Alpha quality release.*

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S72200*
  - S71400*
  - S7450*
  - S7250*

  *Limited number of connections supported

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the RoCEv2 driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104*
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* Kernel must be compiled on RHEL 10.X/9.8/9.7/9.6.

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

2. Software/Driver Loading

Important *Ensure that all inbox drivers are unloaded before proceeding with the unified wire drivers.*

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

2.1. Loading RoCEv2 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 RoCEv2 driver, load the NIC driver and core RDMA drivers first. Run the following commands:

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe iw_cxgb4 roce_mode=1
[root@host~]# modprobe rdma_ucm
```

3. Software/Driver Configuration and Fine-tuning

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

Load the NIC, RoCEv2, and core RDMA modules as mentioned in the 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 through 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...
#
```

3.2. Setting up NFS-RDMA

3.2.1. Starting NFS-RDMA

- **Server-side settings**

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

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

```
/share/rdma1     *(fsid=1,rw,async,insecure,no_root_squash)
/share/rdma2     *(fsid=2,rw,async,insecure,no_root_squash)
```

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

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

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

4. Start the *nfs-server* service.

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

   All services in NFS should start without errors.

5. 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
```

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

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

Now the NFS-RDMA server is ready.

- **Client-side settings**

Follow the steps mentioned below on the client side:

1. Load the iwarp modules and make sure peer2peer is set to 1. Ensure that you are able to ping and ssh to the server Chelsio interface through which directories are exported.
2. Load the `xprtrdma` module.

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

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

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

4. Once the exported directories are listed, mount the directories.

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

To use NFSv4, specify vers=4.

3.3. Performance Tuning

1. Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.
2. Run the performance tuning script to update few kernel parameters using sysctl and map iWARP queues to different CPUs.

```
[root@host~]# t4_perftune.sh -i ethX -s
```

4. Software/Driver Unloading

To unload the RoCEv2 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

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

- Kernel.org linux-6.12.104*
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* Kernel must be compiled on RHEL 10.X/9.8/9.7/9.6.

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

2. Kernel Configuration

Kernel.org linux-6.18.X/6.12.X

1. Ensure that the following NVMe-oF iWARP components are enabled in the kernel configuration file:

```
CONFIG_BLK_DEV_NVME
CONFIG_NVME_RDMA
CONFIG_NVME_TARGET
CONFIG_NVME_TARGET_RDMA
CONFIG_BLK_DEV_NULL_BLK
CONFIG_CONFIGFS_FS
```

2. If the NVMe-oF iWARP components are not enabled, enable them as follows:

```
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
```

3. Compile and install the kernel, then boot into the new kernel and install the Chelsio Unified Wire.

RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4

No additional kernel configuration is required.

3. Software/Driver Installation

3.1. Pre-requisites

Ensure that the following requirements are met before driver installation:

- Uninstall any OFED present in the machine.
- The *rdma-core-devel* package should be installed on *RHEL/Rocky/AlmaLinux 10.X/9.7/9.6* and *Ubuntu-24.04.X* systems.

3.2. Installation

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

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

2. Install iWARP RDMA Offload driver and NVMe utilities.

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

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

3. Reboot your machine for changes to take effect.

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

4. Software/Driver Loading

Important *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:

1. Load the following drivers:

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

2. Bring up the Chelsio interface(s).

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

3. Mount configfs.

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

4. 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

1. The following commands 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
```

2. Save the target configuration to a file.

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

3. To clear the targets,

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

5.2. Initiator

1. Discover the target.

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

2. Connect to the 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
```

3. List the connected targets.

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

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

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

Note *The HMA feature is applicable only for T6 adapters.*

The following image shows the HMA reserved memory.

```
[root@[redacted] ~]# 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@[redacted] ~]# 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.

1. Ensure that Unified Wire is installed with *NVMe Performance* configuration tuning.
2. Run the performance tuning script to update few kernel parameters using sysctl and map iWARP queues to different CPUs.

```
[root@host~]# t4_perftune.sh -i ethX -s
```

3. 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. NVMe-oF RoCEv2

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.

RoCEv2 (RDMA over Converged Ethernet v2) enables low-latency, high-throughput data transfer by allowing direct memory access over standard Ethernet networks. It operates over UDP/IP at Layer 3, making it routable and scalable across modern data center networks. RoCEv2 reduces CPU overhead and improves performance.

Note *This feature is only supported on Chelsio T7 family of adapters.*

Note *NVMe-oF RoCEv2 is provided as an Alpha quality release.*

Note *Enable IOMMU before proceeding.*

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S72200*
  - S71400*
  - S7450*
  - S7250*

  *Limited number of connections supported

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the NVMe-oF RoCEv2 driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104*
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* Kernel must be compiled on RHEL 10.X/9.8/9.7/9.6.

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

2. Kernel Configuration

Kernel.org linux-6.18.X/6.12.X

1. Ensure that the following NVMe-oF RoCEv2 components are enabled in the kernel configuration file:

```
CONFIG_BLK_DEV_NVME
CONFIG_NVME_RDMA
CONFIG_NVME_TARGET
CONFIG_NVME_TARGET_RDMA
CONFIG_BLK_DEV_NULL_BLK
CONFIG_CONFIGFS_FS
```

2. If the NVMe-oF iWARP components are not enabled, enable them as follows:

```
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
```

3. Compile and install the kernel, then boot into the new kernel and install the Chelsio Unified Wire.

RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4

No additional kernel configuration is required.

3. Software/Driver Loading

Important *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:

1. Load the following drivers:

```
[root@host~]# modprobe iw_cxgb4 roce_mode=1
[root@host~]# modprobe rdma_ucm
```

2. Bring up the Chelsio interface(s).

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

3. Mount configfs.

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

4. 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
```

4. Software/Driver Configuration and Fine-tuning

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

4.1. Target

1. The following commands 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
```

2. Save the target configuration to a file.

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

3. To clear the targets,

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

4.2. Performance Tuning

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

1. Ensure that Unified Wire is installed with *NVMe Performance* configuration tuning.
2. Run the performance tuning script to update few kernel parameters using sysctl and map NVMe-oF RoCEv2 queues to different CPUs.

```
[root@host~]# t4_perftune.sh -i ethX -s
```

3. 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
```

5. 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
```

IX. SPDK (user space) NVMe-oF iWARP

1. Introduction

SPDK (Storage Performance Development Kit) is 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

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

- Kernel.org linux-6.12.104*
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* Kernel must be compiled on RHEL 10.X/9.8/9.7/9.6.

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

2. Kernel Configuration

Kernel.org linux-6.18.X/6.12.X

1. Ensure that the following SPDK NVMe-oF iWARP components are enabled in the kernel configuration file:

```
CONFIG_BLK_DEV_NVME
CONFIG_NVME_RDMA
CONFIG_NVME_TARGET
CONFIG_NVME_TARGET_RDMA
CONFIG_BLK_DEV_NULL_BLK
CONFIG_CONFIGFS_FS
```

2. If the SPDK NVMe-oF iWARP components are not enabled, enable them as follows:

```
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
```

3. Compile and install the kernel, then boot into the new kernel and install the Chelsio Unified Wire.

RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4

No additional kernel configuration is required.

3. Software/Driver Loading

Important *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:

1. Load the iWARP RDMA Offload drivers.

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

2. Bring up the Chelsio interface(s).

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

4. Software/Driver Configuration and Fine-tuning

4.1. Target

1. Download SPDK v26.01 LTS.

```
[root@host~]# git clone https://github.com/spdk/spdk
[root@host~]# cd spdk
[root@host~]# git checkout v26.01
[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
```

2. Run the following script to check that the minimum SPDK dependencies are installed.

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

3. Compile SPDK with RDMA and install it.

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

4. 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
```

5. Start the SPDK NVMe-oF iWARP target.

```
[root@host~]# spdk/build/bin/nvmf_tgt -m 0xFF --wait-for-rpc &
[root@host~]# spdk/scripts/rpc.py  iobuf_set_options --large-pool-count 8192
[root@host~]# spdk/scripts/rpc.py framework_start_init
```

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

```
[root@host~]# spdk/scripts/rpc.py nvmf_create_transport -t RDMA -c 4096 -u
131072 -n 8192 -b 700
[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
```

4.2. Initiator

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

1. Follow steps 1 to 4 of the SPDK Target section above to configure and install SPDK.
2. Connect to the target using the fio plugin.

```
[root@host~]# LD_PRELOAD=/root/spdk/build/fi/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
```

4.3. Performance Tuning

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

1. Ensure that Unified Wire is installed with *NVMe Performance* configuration tuning.
2. Run the performance tuning script to update few kernel parameters using sysctl and map iWARP queues to different CPUs.

```
[root@host~]# t4_perftune.sh -i ethX -s
```

5. Software/Driver Unloading

Follow the steps mentioned below to unload the SPDK NVMe-oF iWARP driver:

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

X. SPDK (user space) NVMe-oF RoCEv2

1. Introduction

SPDK (Storage Performance Development Kit) is 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 RoCEv2 RDMA transport.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S72200*
  - S71400*
  - S7450*
  - S7250*

    *Limited number of connections supported

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the SPDK NVMe-oF RoCEv2 driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104*
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* Kernel must be compiled on RHEL 10.X/9.8/9.7/9.6.

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

2. Kernel Configuration

Kernel.org linux-6.18.X/6.12.X

1. Ensure that the following SPDK NVMe-oF RoCEv2 components are enabled in the kernel configuration file:

```
CONFIG_BLK_DEV_NVME
CONFIG_NVME_RDMA
CONFIG_NVME_TARGET
CONFIG_NVME_TARGET_RDMA
CONFIG_BLK_DEV_NULL_BLK
CONFIG_CONFIGFS_FS
```

2. If the SPDK NVMe-oF iWARP components are not enabled, enable them as follows:

```
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
```

3. Compile and install the kernel, then boot into the new kernel and install the Chelsio Unified Wire.

RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4

No additional kernel configuration is required.

3. Software/Driver Loading

Important *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:

1. Load the iWARP RDMA Offload drivers.

```
[root@host~]# modprobe iw_cxgb4 roce_mode=1
[root@host~]# modprobe rdma_ucm
```

2. Bring up the Chelsio interface(s).

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

4. Software/Driver Configuration and Fine-tuning

4.1. Target

1. Download SPDK v26.01 LTS.

```
[root@host~]# git clone https://github.com/spdk/spdk
[root@host~]# cd spdk
[root@host~]# git checkout v26.01
[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
```

2. Run the following script to check that the minimum SPDK dependencies are installed.

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

3. Compile SPDK with RDMA and install it.

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

4. 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
```

5. Start the SPDK NVMe-oF iWARP target.

```
[root@host~]# spdk/build/bin/nvmf_tgt -m 0xFF --wait-for-rpc &
[root@host~]# spdk/scripts/rpc.py  iobuf_set_options --large-pool-count 8192
[root@host~]# spdk/scripts/rpc.py framework_start_init
```

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

```
[root@host~]# spdk/scripts/rpc.py nvmf_create_transport -t RDMA -c 4096 -u
131072 -n 8192 -b 700
[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
```

4.2. Initiator

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

1. Follow steps 1 to 4 of the SPDK Target section above to configure and install SPDK.
2. Connect to the target using the fio plugin.

```
[root@host~]# LD_PRELOAD=/root/spdk/build/fi/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
```

4.3. Performance Tuning

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

1. Ensure that Unified Wire is installed with *NVMe Performance* configuration tuning.
2. Run the performance tuning script to update few kernel parameters using sysctl and map iWARP queues to different CPUs.

```
[root@host~]# t4_perftune.sh -i ethX -s
```

5. Software/Driver Unloading

Follow the steps mentioned below to unload the SPDK NVMe-oF iWARP driver:

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

XI. 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. That is 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the NVMe-oF TOE driver is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Kernel Configuration

Kernel.org linux-6.18.X/6.12.X

1. Ensure that the following NVMe/TCP components are enabled in the kernel configuration file:

```
CONFIG_NVME_CORE
CONFIG_NVME_FABRICS
CONFIG_NVME_TCP
CONFIG_NVME_TARGET
CONFIG_NVME_TARGET_TCP
CONFIG_BLK_DEV_NVME
CONFIG_BLK_DEV_NULL_BLK
CONFIG_CONFIGFS_FS
```

2. If the NVMe/TCP components are not enabled, enable them as follows:

```
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
```

3. Compile and install the kernel, then boot into the new kernel and install the Chelsio Unified Wire.

RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4

No additional kernel configuration is required.

3. Software/Driver Installation

3.1. Installation

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

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

2. Install TOE driver and NVMe utilities.

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

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

3. Reboot your machine for changes to take effect.

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

4. Software/Driver Loading

Important *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:

1. Load the TOE driver.

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

2. Bring up the Chelsio interface(s).

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

3. Mount configfs.

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

4. 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
```

5. 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

1. The following commands configure the 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
```

2. Save the target configuration to a file.

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

3. To clear the targets,

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

5.2. Initiator

1. Discover the target.

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

2. 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
```

3. List the connected targets.

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

4. Format and mount the NVMe disks shown with the above command.
5. 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 (for example, nvme0n1) and not the device path.*

5.3. HMA

To use HMA, ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option. Currently, 256 IPv4/128 IPv6 NVMe-oF TOE Initiator 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 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 are used.

5.4. Performance Tuning

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

1. Run the performance tuning script to update few kernel parameters using sysctl and map TOE queues to different CPUs.

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

2. Set the following 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 the TOE driver, refer to the Software/Driver Unloading section in the **Network (NIC/TOE)** chapter.

XII. SPDK (user space) 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. The Storage Performance Development Kit (SPDK) provides an accelerated user space NVMe-oF target (RDMA and TCP transports), which provides much better performance compared with the 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.

Note *This is supported only on T6/T5 family of adapters.*

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the SPDK NVMe-oF TOE driver is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Kernel Configuration

Kernel.org linux-6.18.X/6.12.X

1. Ensure that the following NVMe/TCP components are enabled in the kernel configuration file:

```
CONFIG_NVME_CORE
CONFIG_NVME_FABRICS
CONFIG_NVME_TCP
CONFIG_NVME_TARGET
CONFIG_NVME_TARGET_TCP
CONFIG_BLK_DEV_NVME
CONFIG_BLK_DEV_NULL_BLK
CONFIG_CONFIGFS_FS
```

2. If the NVMe/TCP components are not enabled, enable them as follows:

```
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
```

3. Compile and install the kernel, then boot into the new kernel and install the Chelsio Unified Wire.

RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4

No additional kernel configuration is required.

3. Software/Driver Installation

3.1. Installation

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

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

2. Install SPDK NVMe-oF TOE driver and NVMe utilities.

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

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

3. Reboot your machine for changes to take effect.

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

4. Software/Driver Loading

Important *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:

1. Load the SPDK NVMe-oF TOE driver.

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

2. 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

1. SPDK v24.05 LTS, customized to support TCP/IP offload and kernel bypass for SPDK NVMe-oF TCP Target is part of the Chelsio Unified Wire package. Change your current working directory to the Chelsio SPDK directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x/build/src/chspdk/user/spdk/
```

2. 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
```

3. 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.
```

4. Below are the sample configuration steps to create a LUN with the 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. Refer to the 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 the target machine, run the following commands:

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

XIII. NVMe-oF/TCP PDU Offload

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. The vital aspect of the transport layer is process-to-process communication. It moves NVMe command, data, and completion PDU processing from the host CPU to the NIC. By handling segmentation, reassembly, and data placement in hardware, it delivers predictable low latency and higher storage IOPS at scale.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the NVMe-oF/TCP PDU Offload driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39[#]

[#] Only Initiator is supported

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

2. Kernel Configuration

Kernel.org linux-6.18.X/linux-6.12.X

1. Ensure that the following NVMe/TCP components are enabled in the kernel configuration file:

```
CONFIG_NVME_CORE
CONFIG_NVME_FABRICS
CONFIG_NVME_TCP
CONFIG_NVME_TARGET
CONFIG_NVME_TARGET_TCP
CONFIG_BLK_DEV_NVME
CONFIG_BLK_DEV_NULL_BLK
CONFIG_CONFIGFS_FS
```

2. If the NVMe/TCP components are not enabled, enable them as follows:

```
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
```

3. Compile and install the kernel, then boot into the new kernel and install the Chelsio Unified Wire.

3. Software/Driver Loading

Important

Kernel and user mode NVMe/TCP PDU offload is not supported simultaneously. Configure either Kernel mode NVMe/TCP or user mode NVMe/TCP.

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:

1. Unload the existing drivers.

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

2. To load NIC driver (cxgb4):

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

3. To load NVMe-oF/TCP PDU Offload Target driver (cnvmet-tcp):

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

4. To load NVMe-oF/TCP PDU Offload Host driver (cnvme-tcp):

```
[root@host~]# rmmod nvme-tcp nvme-fabrics nvme-core
[root@host~]# modprobe cnvme-tcp
```

4. Software/Driver Configuration and Fine-tuning

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

4.1. Target

1. The following commands configure the 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
```

2. Save the target configuration to a file.

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

3. To clear the targets,

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

4.2. Initiator

1. Discover the target.

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

2. Connect to the 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
```

3. List the connected targets.

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

4. Format and mount the NVMe disks shown with the above command.
5. 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 (for example, nvme0n1) and not the device path.*

5. Software/Driver Unloading

Follow the steps mentioned below to unload the NVMe drivers:

On target, run the following commands:

```
[root@host~]# rmmod cnvmet-tcp
```

On initiator, run the following commands:

```
[root@host~]# rmmod cnvme-tcp
```

To unload the TOE driver, refer to the Software/Driver Unloading section in the **Network (NIC/TOE)** chapter.

XIV. SPDK (user space) NVMe/TCP PDU Offload

1. Introduction

The SPDK (Storage Performance Development Kit) NVMe/TCP PDU Offload enables high-performance, low-latency data transfers by moving NVMe/TCP protocol data unit (PDU) processing from the kernel to user space, allowing direct hardware acceleration on both target and host sides. By leveraging user-space drivers, zero-copy data paths, and poll-mode I/O, SPDK eliminates kernel context switching overhead and improves CPU efficiency. When combined with Chelsio adapters supporting NVMe/TCP offload, this approach delivers line-rate throughput, reduced latency, and improved scalability for storage and cloud applications, making it ideal for modern data centers requiring efficient NVMe over TCP performance.

Note *This feature is only supported on Chelsio T7 family of adapters.*

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the supported Chelsio adapters:

- T7 adapter:
  - T72200-FH*
  - T72200*
  - T7450*

  *Limited number of connections supported

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the SPDK NVMe/TCP PDU Offload driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Driver Installation

Perform the following steps to install the driver:

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

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

To build and install SPDK (user space) NVMe/TCP PDU Offload (Target and Initiator) driver,

```
root@host~]# make cstor_pdu_spdk_install
```

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

2. Reboot your machine for changes to take effect.

3. Software/Driver Loading

Important *Kernel and user mode NVMe/TCP PDU offload is not supported simultaneously. Configure either Kernel mode NVMe/TCP or user mode NVMe/TCP.*

Perform the following steps to load the driver:

1. Unload the existing drivers.

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

2. To load NIC driver (cxgb4),

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

3. To load User Space NVMe/TCP PDU Offload driver (cstor),

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

The library (libcstor.so) will be installed by the package at */lib64/*.

4. Configuration

4.1. Configuring SPDK NVMe/TCP PDU Offload

Follow the steps mentioned below on the SPDK NVMe/TCP PDU Offload target/initiator machines:

1. SPDK v26.01, customized to support NVMe/TCP PDU Offload and kernel bypass for SPDK NVMe/TCP Target and Initiator is part of Chelsio Unified Wire package. Change your current working directory to the Chelsio SPDK directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x/build/src/chspdk/spdk-26.01
```

2. 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
```

3. Start the reactor.

```
[root@host spdk]# ./build/bin/nvmf_tgt -m <cpu_mask>
```

4.2. SPDK NVMe/TCP PDU Offload Target

Below are the sample configuration steps to create a LUN with Malloc bdev of 256MB.

```
SPDK_PATH=$'ChelsioUwire-x.x.x.x/build/src/chspdk/spdk-26.01'
$SPDK_PATH/scripts/rpc.py nvmf_create_transport -t TCP --iface <interface
name>
$SPDK_PATH/scripts/rpc.py bdev_malloc_create -b Malloc1 256 512
$SPDK_PATH/scripts/rpc.py nvmf_create_subsystem nqn.2016-06.io.spdk:cnode1 -
a -s SPDK00000000000001 -d SPDK_Controller1
```

```
$SPDK_PATH/scripts/rpc.py nvmf_subsystem_add_ns nqn.2016-06.io.spdk:cnode1
Malloc1
$SPDK_PATH/scripts/rpc.py nvmf_subsystem_add_listener nqn.2016-
06.io.spdk:cnode1 -t tcp -a 102.11.11.45 -s 4420 -f ipv4de1 Malloc1
$SPDK_PATH/scripts/rpc.py nvmf_subsystem_add_listener nqn.2016-
06.io.spdk:cnode1 -t tcp -a 102.11.11.45 -s 4420 -f ipv4
```

The SPDK NVMe/TCP PDU Offload Initiator, Kernel NVMe/TCPInitiators can connect to the configured target.

4.3. SPDK NVMe/TCP PDU Offload Initiator

SPDK NVMe/TCP PDU Offload Initiator can connect to an NVMe/TCP Target using the following steps:

```
SPDK_PATH=$'ChelsioUwire-x.x.x.x/build/src/chspdk/spdk-26.01'
$SPDK_PATH/scripts/rpc.py bdev_nvme_attach_controller -b < bdev name > -t TCP
-a < Target IP > -f IPv4 -s 4420 -n < target NQN >
```

Example:

```
$SPDK_PATH/scripts/rpc.py bdev_nvme_attach_controller -b Nvme1 -t TCP -a
102.11.11.86 -f IPv4 -s 4420 -n nvme-ram1
```

5. Software/Driver Unloading

To unload the driver, execute the following commands:

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

XV. SPDK NVMe/TCP over Memfree TOE

1. Introduction

The SPDK (Storage Performance Development Kit) NVMe/TCP over Memfree TOE enables high-performance, low-latency data transfers by moving NVMe/TCP protocol data unit (PDU) processing from the kernel to user space, allowing direct hardware acceleration on both target and host sides. By leveraging user-space drivers, zero-copy data paths, and poll-mode I/O, SPDK eliminates kernel context switching overhead and improves CPU efficiency. When combined with Chelsio adapters supporting NVMe/TCP offload, this approach delivers line-rate throughput, reduced latency, and improved scalability for storage and cloud applications, making it ideal for modern data centers requiring efficient NVMe over TCP performance.

Note *This feature is only supported on Chelsio T7 family of adapters.*

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the supported Chelsio adapters:

- T7 adapters:
  - S72200
  - S71400
  - S7450
  - S7250

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the SPDK NVMe/TCP over Memfree TOE driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Compiling and installing spdk

1. Clone the spdk folder to local.

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

2. Open spdk folder.

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

3. Checkout v26.01 git tag.

```
[root@host~]# git checkout v26.01
[root@host~]# git submodule update --init
```

4. Apply a patch.

```
[root@host~]# patch -p1 < ChelsioUwire-
x.x.x.x/build/src/chspdk/patches/spdk-26.01/spdk_disable_tx_zero_copy.patch
[root@host~]# ./configure
[root@host~]# make
```

3. Software/Driver Loading

Important *Kernel and user mode NVMe/TCP PDU offload is not supported simultaneously. Configure either Kernel mode NVMe/TCP or user mode NVMe/TCP.*

Perform the following steps to load the driver:

1. Unload the existing drivers.

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

2. To load Offload driver (t4_tom),

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe t4_tom
```

4. Configuration

4.1. Configuring SPDK NVMe/TCP over Memfree TOE

Follow the steps mentioned below on the SPDK (user space) NVMe/TCP over Memfree TOE target/initiator machines:

1. Use the above compiled SPDK v26.01.

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

2. 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 /root/spdk
[root@host~]# NRHUGE=8192 scripts/setup.sh
```

3. Start the reactor.

```
[root@host~]# build/bin/nvmf_tgt -m <cpu_mask>
```

4.2. SPDK NVMe/TCP over Memfree TOE Target

Below are the sample configuration steps to create a LUN with Malloc bdev of 256MB.

```
[root@host~]# cd spdk
[root@host~]# scripts/rpc.py nvmf_create_transport -t TCP
[root@host~]# scripts/rpc.py bdev_malloc_create -b Malloc1 256 512
[root@host~]# scripts/rpc.py nvmf_create_subsystem nqn.2016-
06.io.spdk:cnode1 -a -s SPDK00000000000001 -d SPDK_Controller1
[root@host~]# scripts/rpc.py nvmf_subsystem_add_ns nqn.2016-
06.io.spdk:cnode1 Malloc1
```

```
[root@host~]# scripts/rpc.py nvmf_subsystem_add_listener nqn.2016-
06.io.spdk:cnode1 -t tcp -a 102.11.11.45 -s 4420 -f ipv4
```

The SPDK NVMe/TCP over Memfree TOE Initiator, Kernel NVMe/TCP Initiators can connect to the configured target.

4.3. SPDK NVMe/TCP Over Memfree TOE Initiator

SPDK (user space) NVMe/TCP over Memfree TOE Initiator can connect to an NVMe/TCP Target using the following steps:

```
[root@host~]# cd spdk
[root@host~]# scripts/rpc.py bdev_nvme_attach_controller -b < bdev name > -
t TCP -a < Target IP > -f IPv4 -s 4420 -n <target NQN>
```

Example:

```
[root@host~]# cd spdk
[root@host~]# scripts/rpc.py bdev_nvme_attach_controller -b Nvme1 -t TCP -a
102.11.11.45 -f IPv4 -s 4420 -n nvme-ram1
```

XVI. SoftiWARP

1. Introduction

SoftiWARP (siw) is a software iWARP kernel driver and user library for Linux that 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 following advantages:

- Provides a simple path for transition of RDMA applications to the cloud platform.
- Is useful for Client/Initiator side applications such as iSER, NVMe-oF, NFSoRDMA, and LustreoRDMA 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S71400
  - S72200
  - S7450
  - S7250
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T62100-SO-CR
  - T62100-SO-OCP3
  - T6225-CR
  - T6225-SO-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the SoftiWARP driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Kernel Configuration

Kernel.org linux-6.18.X/6.12.X

1. Ensure that the following SoftiWARP component is enabled in the kernel configuration file:

```
CONFIG_RDMA_SIW
```

2. If the SoftiWARP components are not enabled, enable them as follows:

```
CONFIG_RDMA_SIW=m
```

3. Compile and install the kernel, then boot into the new kernel and install Chelsio Unified Wire.

3. Software/Driver Loading

Important *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:

1. Load the network driver (cxgb4).

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

2. Load the SoftiWARP driver (siw).

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

3. Unload the iWARP RDMA offload driver (iw_cxgb4).

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

4. 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.

4.1. Initiator/Client

Important *Disable iWARP Port Mapper (iwpmd) service on Target and Initiator.*

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

1. RDMA tool (*rdma*) is used to configure the siw device. It is installed by default, from RHEL distribution. If not present in the machine, then install it from the latest iproute2 package.

2. Configure the siw device.

```
[root@host~]# rdma link add <siw_device> type siw netdev <ethX>
[root@host~]# ifconfig ethX <IP address> up
```

3. Verify the configuration using *ibv_devices.*

```
[root@ ~]# ibv_devices
    device                 node GUID
    ------              ----------------
    siw_enp6s0f4        0007433ddb400000
```

4. The initiator/client can now connect to the target/server machines.
   Refer to the NVMe-oF iWARP initiator and iSER initiator sections for steps to connect to the respective targets.

5. Software/Driver Unloading

Follow the steps mentioned below to unload the SoftiWARP and network drivers:

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

XVII. 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 10/25/40/50/100/200/400Gb
- 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the LIO iSCSI Target Offload driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Kernel Configuration

RHEL/Rocky/AlmaLinux RHEL 10.X/9.8/9.7/9.6

To use RHEL/Rocky/AlmaLinux RHEL 10.X/9.8/9.7/9.6 version, perform the following steps:

1. Download the kernel source RPM kernel-6.12.0- xxx.el10_1.src.rpm for your distribution.
2. Install the kernel source.

```
[root@host~]# rpm -ivh kernel-6.12.0-xxx.el10_1.src.rpm
```

3. Prepare the kernel source.

```
[root@host~]# cd /root/rpmbuild/SPECS/
[root@host~]# rpmbuild -bp kernel.spec --nodeps
[root@host~]# cd /root/rpmbuild/BUILD/kernel-6.12.0-xxx.el10_1/linux-
6.12.0-xxx.el10_1.x86_64/
[root@host~]# make prepare
```

4. Copy the source to /usr/src directory.

```
[root@host~]# cp -r linux-6.12.0-xxx.el10_1 /usr/src
```

5. Proceed with driver installation as directed in the **Driver Installation** section.

Kernel.org linux-6.18.X/6.12.X

Follow the below steps to use a 6.12.X kernel version,

1. Download the required kernel version from kernel.org.
2. Extract the tar-ball.
3. Change your working directory to the kernel directory and run the following command to invoke the installation menu.

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

4. Select **Device Drivers > Generic Target Core Mod (TCM) and ConfigFS Infrastructure**.
5. Enable **Linux-iSCSI.org iSCSI Target Mode Stack** as a Module (if not already enabled).
6. Select **Save**.
7. Exit from the installation menu.
8. Continue with kernel installation as usual.
9. Boot into the new kernel and proceed with driver installation as directed in the **Software/Driver Installation** section.

Ubuntu 24.04.4

To use Ubuntu 24.04.4 version, perform the following steps:

1. Install the source using apt-get.

```
[root@host~]# apt install linux-source-6.8.0*
[root@host~]# cd /usr/src/linux-source-6.8.0
[root@host~]# tar xf /usr/src/linux-source-6.8.0.tar.bz2
```

2. Proceed with driver installation as directed in the **Driver Installation** section.

```
[root@host~]# rpm -ivh kernel-6.12.0-xxx.el10_1.src.rpm
```

Pre-requisites

Ensure that the following requirements are met before driver installation:

- targetcli is automatically installed by the Chelsio Unified Wire installer using the package manager yum/apt, if missing from the system. If you wish to use a different version, it is highly recommended to install v2.1.fb44 or higher versions.

3. Software/Driver Installation

3.1. Pre-requisites

Ensure that the following requirements are met before driver installation:

- *targetcli* will be automatically installed by the Chelsio Unified Wire installer using the package manager yum/apt/zypper, if missing from the system. If you wish to use a different version, it is highly recommended to install v2.1.fb44 or higher versions.

3.2. Installation

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

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

2. Install LIO driver and targetcli utilities.

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

In case of RHEL/Rocky/AlmaLinux 10.X/9.7.9.6, the kernel source (KSRC), and kernel object (KOBJ) must be specified.

- CLI mode

```
[root@host~]# make lio_install KSRC="<kernel_source_dir>"
KOBJ="<kernel_object_dir>
```

- CLI mode (without Dialog utility)

```
[root@host~]# ./install.py --ksrc=<kernel_source_dir> --
kobj=<kernel_object_dir>
```

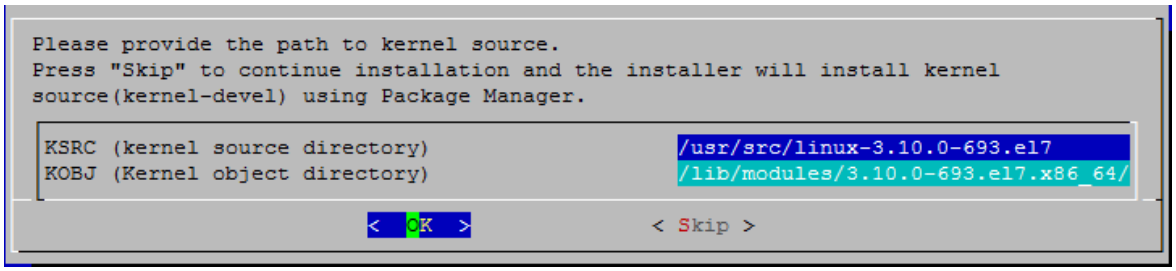
Example: For RHEL 10.1,

```
[root@host~]# ./install.py --ksrc=/usr/src/linux-6.12.0-124.8.1.el10_1 --
kobj=/lib/modules/6.12.0-124.8.1.el10_1.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, run* `make help` *or* `install.py -h`.

3. Reboot your machine for changes to take effect.

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

4. Software/Driver Loading

Important

Kernel and user mode iSCSI PDU offload is not supported simultaneously. Configure either kernel mode iSCSI or user mode iSCSI.

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 may fail.

1. Load network driver (*cxgb4*).

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

2. Bring up the interface.

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

3. Load the LIO iSCSI Target Offload driver (*cxgbit)*.

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

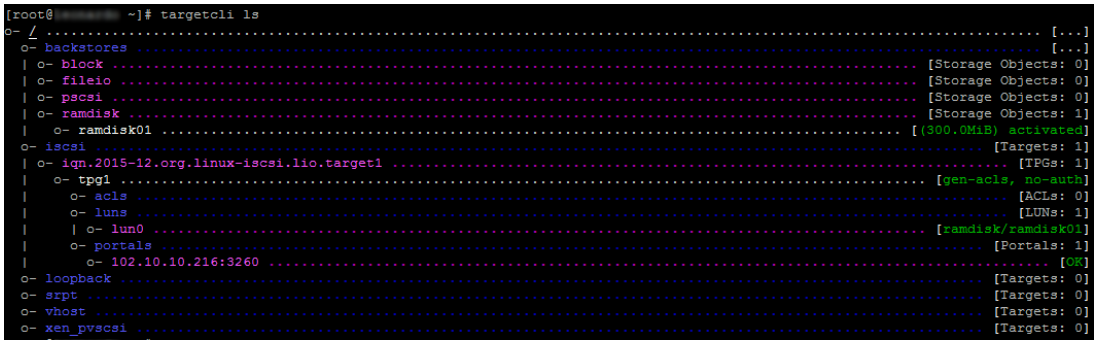
5. Software/Driver Configuration and Fine-tuning

5.1. Configuring LIO iSCSI Target

The LIO iSCSI Target needs to be configured before it can become useful. Refer to the *targetcli* man page using the `man targetcli` command to do so.

5.1.1. Sample Configuration

Here is a sample iSCSI configuration listing a target configured with one 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 the 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,

1. If not done already, load the network driver with offload support (TOE).

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

2. Create a new policy file.

```
[root@host~]# cat <new_policy_file>
```

3. 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
```

4. Compile the policy.

```
[root@host~]# cop -d -o <output_policy_file> <new_policy_file>
```

5. 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

1. Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.
2. Run the performance tuning script to update a few Kernel parameters using sysctl and map LIO Target queues to different CPUs.

```
[root@host~]# t4_perftune.sh -i ethX -s
```

3. For maximum performance, it is recommended to use iSCSI PDU offload initiator.
   - For MTU 9000, no additional configuration is 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:

1. Log out from the initiator.
2. 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 the Chelsio interface.

3. Unload the driver.

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

6.2. Unloading the NIC Driver

To unload the driver in NIC mode (without offload support):

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

XVIII. 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 transmits 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, and then forwarding the header to the host. The payload data will be directly placed into the pre-posted host DDP buffer if possible. 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the iSCSI PDU Offload Initiator driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

1.3. Pre-requisites

Ensure that the following requirements are met before driver 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.
- The *openssl-devel* package must be installed.

2. Software/Driver Installation

2.1. Pre-requisites

Ensure that the following requirements are met before driver 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.
- The *openssl-devel* package must be installed.

2.2. Installation

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

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

2. Install open-iSCSI,iSCSI-initiator, firmware, and utilities.

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

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

3. Reboot your machine for changes to take effect.

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

3. Software/Driver Loading

Important

Kernel and user mode iSCSI PDU offload is not supported simultaneously. Configure either kernel mode iSCSI or user mode iSCSI.

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 the **unknown symbols found** error message in dmesg, follow the steps mentioned below:

1. View all the loaded iSCSI modules.

```
[root@host~]# lsmod | grep iscsi
```

2. Now, unload them using the following command:

```
[root@host~]# rmmod <modulename>
```

3. Finally, reload the *cxgb4i* driver.

4. Software/Driver Configuration and Fine-tuning

4.1. Accelerating open-iSCSI Initiator

To accelerate the open-iSCSI initiator, the following steps need to be taken:

4.1.1. Configuring interface (*iface)* file

Execute the following command to create the file automatically by loading *cxgb4i* driver:

```
[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

a. 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.*

b. 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
```

c. 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.*

d. 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, ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option.

Note *The HMA feature is applicable only for T6 adapters.*

1. Use LIO iSCSI Target in offload mode.
2. Configure MTU 9000 for Chelsio Interfaces.
3. 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 T6 25G SO adapters. The following image shows the HMA reserved memory.

```
[root@[redacted] ~]# 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@[redacted] ~]# 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, 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.

If iSCSI Initiator IRQs pose a bottleneck for multiple connections, you can improve IOPS performance using the steps mentioned below:

1. Enable iSCSI multi-queue. In 3.18+ kernels, add the following entry to the grub configuration file and reboot the machine:

```
[root@host~]# scsi_mod.use_blk_mq=1
```

2. Run the performance tuning script to update a few kernel parameters using sysctl and map iSCSI Initiator queues to different CPUs.

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

3. Load the initiator driver.

```
[root@host~]# modprobe cxgb4i
```

5. Software/Driver Unloading

To unload the driver, execute the following commands:

```
[root@host~]# rmmod cxgb4i
[root@host~]# rmmod libcxgbi
```

XIX. SPDK (user space) iSCSI PDU Offload

1. Introduction

The SPDK (Storage Performance Development Kit) iSCSI PDU Offload enables high-performance, low-latency data transfers by moving iSCSI protocol data unit (PDU) processing from the kernel to user space, allowing direct hardware acceleration on both target and host sides. By leveraging user-space drivers, zero-copy data paths, and poll-mode I/O, SPDK eliminates kernel context switching overhead and improves CPU efficiency. When combined with Chelsio adapters supporting iSCSI offload, this approach delivers line-rate throughput, reduced latency, and improved scalability for storage and cloud applications, making it ideal for modern data centers requiring efficient iSCSI performance.

Note *This feature is only supported on Chelsio T7 family of adapters.*

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S72200
  - S71400
  - S7450
  - S7250

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the SPDK iSCSI PDU Offload driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Driver Installation

Perform the following steps to install the driver:

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

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

To build and install SPDK (user space) iSCSI PDU Offload (Target and Initiator) driver,

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

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

2. Reboot your machine for changes to take effect.

3. Software/Driver Loading

Important *Kernel and user mode iSCSI PDU offload is not supported simultaneously. Configure either kernel mode iSCSI or user mode iSCSI.*

Perform the following steps to load the driver:

1. Unload the existing drivers.

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

2. To load NIC driver (cxgb4),

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

3. To load SPDK (user space) iSCSI PDU Offload driver (cstor),

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

The library (libcstor.so) will be installed by the package at */lib64/*.

4. Configuration

4.1. Configuring SPDK iSCSI PDU Offload target/initiator

Follow the steps mentioned below on the SPDK iSCSI PDU Offload target/initiator machines:

1. SPDK v26.01, customized to support iSCSI PDU Offload and kernel bypass for SPDK iSCSI Target and Initiator is part of the Chelsio Unified Wire package. Change your current working directory to the Chelsio SPDK directory.

```
[root@host~]# cd ChelsioUwire-x.x.x.x/build/src/chspdk/spdk-26.01
```

2. 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
```

3. Start the reactor.

```
[root@host spdk]# ./build/bin/iscsi_tgt -m <cpu_mask>
```

4.2. SPDK iSCSI PDU Offload Target

Below are the sample configuration steps to create a LUN with Malloc bdev of 256MB.

```
 [root@host~]# SPDK_PATH=$'ChelsioUwire-x.x.x.x/build/src/chspdk/spdk-26.01'
 [root@host~]# $SPDK_PATH/scripts/rpc.py  iscsi_create_transport --iface
<interface name>
 [root@host~]# $SPDK_PATH/scripts/rpc.py bdev_malloc_create -b Malloc1 256
512
```

```
 [root@host~]# $SPDK_PATH/scripts/rpc.py iscsi_create_portal_group 1
102.11.11.45:3260
 [root@host~]# $SPDK_PATH/scripts/rpc.py iscsi_create_initiator_group 2 ANY
 [root@host~]# $SPDK_PATH/scripts/rpc.py iscsi_create_target_node disk1
"Data Disk1" "Malloc1:0" 1:2 64 -d
```

The SPDK iSCSI PDU Offload Initiator, Kernel iSCSI PDU Offload or Open-iSCSI Initiators can connect to the configured target.

4.3. SPDK iSCSI PDU Offload Initiator

SPDK iSCSI PDU Offload Initiator can connect to an iSCSI Target using the below steps:

```
 [root@host~]# SPDK_PATH=$'ChelsioUwire-x.x.x.x/build/src/chspdk/spdk-26.01'
 [root@host~]# $SPDK_PATH/scripts/rpc.py bdev_iscsi_create -b <bdev_name> -i
<INITIATOR_IQN> --url <iSCSI_LUN_URL>
```

Example:

```
[root@host~]# $SPDK_PATH/scripts/rpc.py bdev_iscsi_create -b iSCSI1 -i iqn.2016-
06.io.spdk:init1
--url 'iscsi://102.11.11.45/iqn.2000-
01.com.chelsio.host:'"1"'/0?header_digest=crc32c&data_digest=crc32c'
```

5. Software/Driver Unloading

To unload the driver, execute the following commands:

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

XX. SPDK (user space) iSCSI over Memfree TOE

1. Introduction

The SPDK (Storage Performance Development Kit) iSCSI enables high-performance, low-latency data transfers by moving iSCSI protocol data unit (PDU) processing from the kernel to user space, allowing direct hardware acceleration on both target and host sides. By leveraging user-space drivers, zero-copy data paths, and poll-mode I/O, SPDK eliminates kernel context switching overhead and improves CPU efficiency. When combined with Chelsio adapters supporting iSCSI offload, this approach delivers line-rate throughput, reduced latency, and improved scalability for storage and cloud applications, making it ideal for modern data centers requiring efficient iSCSI performance.

Note *This feature is only supported on Chelsio T7 family of adapters.*

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the supported Chelsio adapters:

- T7 adapters:
  - S72200
  - S71400
  - S7450
  - S7250

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the SPDK iSCSI over Memfree TOE driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* SPDK (user space) iSCSI over Memfree TOE (Target) is supported on 6.18.39 only if iommu is enabled.

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

2. Compiling and installing spdk

Perform the following steps to compile and install the driver:

1. Clone the spdk folder to local.

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

2. Open spdk folder.

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

3. Checkout v26.01 git tag.

```
[root@host~]# git checkout v26.01
[root@host~]# git submodule update --init
```

4. Apply a patch on the spdk repo.

```
[root@host~]# cd spdk
[root@host~]# patch -p1 < ChelsioUwire-
x.x.x.x/build/src/chspdk/patches/spdk-26.01/spdk_disable_tx_zero_copy.patch
[root@host~]# ./configure
[root@host~]# make
```

3. Software/Driver Loading

Important *Kernel and user mode iSCSI PDU offload is not supported simultaneously. Configure either kernel mode iSCSI or user mode iSCSI.*

Perform the following steps to load the driver:

1. Unload the existing drivers.

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

2. To load Offload driver (t4_tom),

```
[root@host~]# modprobe cxgb4
[root@host~]# modprobe t4_tom
```

4. Configuration

4.1. Configuring SPDK iSCSI over Memfree TOE target

Follow the steps mentioned below on the iSCSI over Memfree TOE target/initiator machines:

1. Change your current working directory to the SPDK 26.01 directory.

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

2. 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 /root/spdk
[root@host~]# NRHUGE=8192 scripts/setup.sh
```

3. Start the reactor.

```
[root@host~]# build/bin/iscsi_tgt -m <cpu_mask> &
```

4.2. SPDK iSCSI over Memfree TOE Target

Below are the sample configuration steps to create 1 SPDK iSCSI over Memfree TOE Target.

```
[root@host~]# cd spdk
[root@host~]# scripts/rpc.py  bdev_malloc_create -b Malloc1 256 512
[root@host~]# scripts/rpc.py iscsi_create_portal_group 1 102.11.11.45:3260
[root@host~]# scripts/rpc.py iscsi_create_initiator_group 2 ANY
[root@host~]# scripts/rpc.py iscsi_create_target_node disk1 "Data Disk1" "Malloc1:0"
1:2 64 -d
```

The Kernel iSCSI PDU Offload or Open-iSCSI Initiators can connect to the configured target.

XXI. 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 supported Chelsio adapters:

- T7 adapters:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T62100-SO-CR*
  - T62100-SO-OCP3*
  - T6225-CR
  - T6225-SO-CR*

* Only Co-processor supported

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the Crypto Offload driver is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

2. Kernel Configuration

Kernel.org linux-6.18.X/6.12.X

1. Enable the following options in the kernel configuration file:

```
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
```

2. Compile and install the kernel, then boot into the new kernel and install the Chelsio Unified Wire.

RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4

No additional kernel configuration is required.

3. Software/Driver Installation

3.1. Pre-requisites

Ensure that SELinux and the firewall are disabled.

3.2. Installation

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

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

2. Install the Crypto driver.

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

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

3. Reboot your machine for changes to take effect.

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

4. Software/Driver Loading

Important *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

1. To load the Crypto Offload driver in Inline mode,

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

2. Bring up the Chelsio network interface.

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

Where *ethX* is the Chelsio interface.

4.2. Co-processor

1. To load the Crypto Offload driver in Co-processor mode (*chcr*).

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

2. Bring up the Chelsio network interface.

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

Where *ethX* is the Chelsio interface.

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:

- TLS/SSL
- nginx
- IPsec
- ip xfrm
- Strongswan

5. Software/Driver Configuration and Fine-tuning

5.1. Configuring OpenSSL

Install the OpenSSL v3.5.4 with kTLS support and do the configuration, perform the following steps:

1. Download the OpenSSL v3.5.4, compile with kTLS support and install it.

```
[root@host~]# wget
https://github.com/openssl/openssl/releases/download/openssl-3.5.4/openssl-
3.5.4.tar.gz
[root@host~]# tar xf openssl-3.5.4.tar.gz
[root@host~]# cd openssl-3.5.4
[root@host~]# ./config shared enable-ktls --prefix=/root/sslv3-ktls --
openssldir=/root/sslv3-ktls -Wl,-R/root/sslv3-ktls/lib64
[root@host~]# make && make install
```

2. 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
```

Ensure that the following requirements are met for connections to be offloaded:

- TLS version should be v1.2
- Cipher should be AES128-GCM-SHA256

3. Compile and install nginx.

```
[root@host ~]# tar xf nginx-1.28.0.tar.gz
[root@host ~]# cd nginx-1.28.0/
[root@host nginx-1.28.0]# ./configure --prefix=/usr/local/nginx-openssl-ktls
--with-http_ssl_module --with-http_v2_module --with-http_dav_module --with-
cc-opt="-DNGX_SSL_SENDFILE -DOPENSSL_API_COMPAT=10101 -I /root/openssl-
3.5.4/include/" --with-ld-opt="-L/root/sslv3-ktl/lib64 -Wl,-R/root/sslv3-
ktl/lib64"
[root@host nginx-1.28.0]# make && make install
```

4. Configure the nginx server by updating the required settings in /usr/local/nginx/nginx.conf file.

5. Start nginx server.

```
[root@host~]# /usr/local/nginx-openssl-ktls/nginx
```

   The nginx server will be offloaded by Chelsio Co-processor now.

6. The Client can now connect to the Server and download the files.

5.2. Co-processor IPsec Offload

5.2.1. Configuration

Note

- *DUT is the machine equipped with a T7 card.*
- *PEER is the corresponding companion machine.*

5.2.2. Using ip xfrm

1. Establish IPSec SAs and Policies using the commands below.
   a. On DUT

```
[root@DUT~]# ip xfrm state add src $DUT_IP dst $PEER_IP proto esp
spi 0x0a1b2c3d reqid 0x5f7c1a6b mode transport aead
"rfc4106(gcm(aes))" 0x0f83e4cb43bd443a839bef61378d294577386f72 128
sel src 0.0.0.0/0 dst 0.0.0.0/0 replay-window 128 flag esn
[root@DUT~]# ip xfrm state add src $PEER_IP dst $DUT_IP proto esp
spi 0x4e5f6a7b reqid 0x9abcdef0 mode transport aead
"rfc4106(gcm(aes))" 0x0f83e4cb43bd443a839bef61378d294577386f72 128
sel src 0.0.0.0/0 dst 0.0.0.0/0 replay-window 128 flag esn
[root@DUT~]# ip xfrm policy add src $DUT_IP dst $PEER_IP dir out
tmpl src $DUT_IP dst $PEER_IP proto esp reqid 0x5f7c1a6b mode
transport
[root@DUT~]# ip xfrm policy add src $PEER_IP dst $DUT_IP dir fwd
tmpl src $PEER_IP dst $DUT_IP proto esp reqid 0x9abcdef0 mode
transport
[root@DUT~]# ip xfrm policy add src $PEER_IP dst $DUT_IP dir in tmpl
src $PEER_IP dst $DUT_IP proto esp reqid 0x9abcdef0 mode transport
```

Note *ethX refers to the interface of DUT adapters.*

   b. On PEER

```
[root@PEER~]# ip xfrm state add src $DUT_IP dst $PEER_IP proto esp
spi 0x0a1b2c3d reqid 0x5f7c1a6b mode transport aead
"rfc4106(gcm(aes))" 0x0f83e4cb43bd443a839bef61378d294577386f72 128
sel src 0.0.0.0/0 dst 0.0.0.0/0 replay-window 128 flag esn
[root@PEER~]# ip xfrm state add src $PEER_IP dst $DUT_IP proto esp
spi 0x4e5f6a7b reqid 0x9abcdef0 mode transport aead
"rfc4106(gcm(aes))" 0x0f83e4cb43bd443a839bef61378d294577386f72 128
sel src 0.0.0.0/0 dst 0.0.0.0/0 replay-window 128 flag esn
```

```
[root@PEER~]# ip xfrm policy add src $PEER_IP dst $DUT_IP dir out
tmpl src $PEER_IP dst $DUT_IP proto esp reqid 0x9abcdef0 mode
transport
[root@PEER~]# ip xfrm policy add src $DUT_IP dst $PEER_IP dir fwd
tmpl src $DUT_IP dst $PEER_IP proto esp reqid 0x5f7c1a6b mode
transport
[root@PEER~]# ip xfrm policy add src $DUT_IP dst $PEER_IP dir in
tmpl src $DUT_IP dst $PEER_IP proto esp reqid 0x5f7c1a6b mode
transport
```

2. Verify SA and policy status:

```
[root@DUT~]# ip xfrm state list
```

5.2.3. Using StrongSwan

1. Deploy certificates to DUT and PEER.

```
/usr/local/etc/swanctl/private/ # Private Keys of respective hosts
/usr/local/etc/swanctl/x509/    # Host Certificates of respective hosts
/usr/local/etc/swanctl/x509ca/  # CA Certificate
```

2. Configure "/usr/local/etc/strongswan.conf" (enable “make_before_break = yes”).

```
[root@DUT~]# cat /usr/local/etc/strongswan.conf
charon {
        load_modular = yes
        plugins {
                include strongswan.d/charon/*.conf
                make_before_break = yes       # <------- Append
"make_before_break" parameter
        }
}
```

Note *"make_before_break" ensures seamless tunnel rekeying by creating a new SA before tearing down the old one, preventing traffic disruption.*

3. Update "*/usr/local/etc/swanctl/swanctl.conf*" on both the DUT and PEER according to their IP configurations.
    a. DUT Machine

```
[root@DUT~]# cat /usr/local/etc/swanctl/swanctl.conf
connections {
    gw-port0 {
        local_addrs = <DUT_IP>
        remote_addrs = <PEER_IP>

        local {
            auth = pubkey
            certs = dut-Cert.der
            id = C=CH, O=test.ipsec.tunnel, CN=dut-common-name
        }

        remote {
            auth = pubkey
            id = C=CH, O=test.ipsec.tunnel, CN=peer-common-name
        }

        children {
            net-net-0 {
                rekey_time = 4000000
                esp_proposals = aes128gcm128-esn-noesn
                hw_offload = no
                mode = tunnel
            }
        }

        version = 2
        mobike = no
        rekey_time = 8500000
        reauth_time = 9000000
        proposals = aes256-sha1-x25519
    }
}
```

b. PEER Machine

```
[root@PEER~]# cat /usr/local/etc/swanctl/swanctl.conf
connections {
    gw-port0 {
        local_addrs = <PEER_IP>
        remote_addrs = <DUT_IP>
        local {
            auth = pubkey
            certs = peer-Cert.der
            id = C=CH, O=test.ipsec.tunnel, CN=peer-common-name
        }
        remote {
            auth = pubkey
            id = C=CH, O=test.ipsec.tunnel, CN=dut-common-name
        }
        children {
            net-net-0 {
                rekey_time = 5000000
                esp_proposals = aes128gcm128-esn-noesn
                hw_offload = no
                mode = tunnel
            }
        }
        version = 2
        mobike = no
        rekey_time = 11000000
        reauth_time = 12000000
        proposals = aes256-sha1-x25519
    }
}
```

4. Start and load configurations on both DUT and PEER Machines.

```
[root@DUT~]# /usr/local/libexec/ipsec/charon
[root@DUT~]# swanctl --load-creds
[root@DUT~]# swanctl --load-conns
```

5. Initiate connection from DUT.

```
[root@DUT~]# swanctl -i -c net-net-0
```

6. Verify connection.

```
[root@DUT~]# swanctl --list-sas -P
```

5.3. Inline TLS Offload

5.3.1. Configure TLS Offload and TOE Ports

To configure Inline TLS Offload, the connection offload policy should be used with the required TCP port numbers. Perform the following the steps:

1. 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 the 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
```

2. 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
```

3. 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
```

5.3.2. Configuring and running Applications

The following sections assume that openssl with kTLS support is installed at */root/sslv3-ktls* as mentioned in Configuring OpenSSL. If you wish to use your own openssl with kTLS support, replace */root/sslv3-ktls* with the corresponding location.

- **OpenSSL tool**

1. Start TLS offload Server.

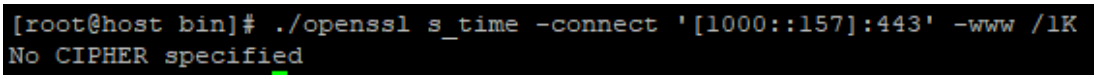
```
[root@host~]# cd /root/sslv3-ktls/bin
[root@host~]# ./openssl s_server -key <key_file> -cert <cert_file> -accept
<server_ip>:<port> -cipher AES128-GCM-SHA256 -WWW -4 -tls1_2
```

2. Start TLS offload Client.

```
[root@host~]# cd /root/sslv3-ktls/bin
[root@host~]# ./openssl s_time -connect <server_ip>:<port> -www /<file>
```



In case of IPv6, the address should be specified within [].

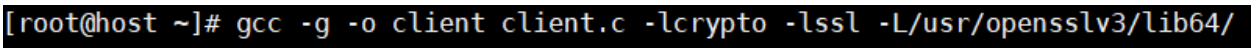


Custom Applications

To compile custom applications using OpenSSL library,

```
[root@host~]# gcc -g -o <server/client output file> <server/client file> -
lcrypto -lssl -L/root/sslv3-ktls/lib64/
```

Client:



Server:



- **nginx server**

1. Download the latest stable version from nginx website.
2. 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 /root/sslv3-ktls/include" --
with-ld-opt="-L/root/sslv3-ktls/lib64 -Wl,-R/root/sslv3-ktls/lib64" && make
&& make install
```

3. Configure the nginx server by updating required settings in */usr/local/nginx/nginx.conf* file.
4. Update the following in */usr/local/nginx/nginx.conf* file.

```
http {
   ...
   ssl_protocols           TLSv1.2;
   ssl_ciphers  AES128-GCM-SHA256;
   ssl_prefer_server_ciphers  on;
   ...
      }
```

5. Load Chelsio Inline drivers and configure nginx server port as a TLS Offload port as described in the Configure TLS Offload and TOE Ports section.
6. Start nginx server.

```
[root@host~]# ./usr/local/nginx/nginx
```

   The nginx server will be Inline TLS offloaded now.

7. The Client can now connect to the Server and download the files.

5.3.3. Inline TLS Counters

To verify whether 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
```

5.4. 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:

- TLS/SSL
  - Nginx
- IPsec
  - ip xfrm
  - Strongswan

5.4.1. Co-processor TLS Offload

This section assumes that openssl with kTLS support is installed at */root/sslv3-ktls* as mentioned in the Configuring OpenSSL section. If you wish to use your own openssl with kTLS support, replace */root/sslv3-ktls* with the corresponding location.

- **nginx server**

1. Download the latest stable version from the nginx website.
2. 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 /root/sslv3-ktls/include" --with-ld-opt="-
L/root/sslv3-ktls/lib64 -Wl,-R/root/sslv3-ktls/lib64" && make && make install
```

3. Configure the nginx server by updating the required settings in */usr/local/nginx/nginx.conf* file.
4. 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
```

5. Start nginx server.

```
[root@host~]# ./usr/local/nginx/nginx
```

The nginx server will be offloaded by Chelsio Co-processor now.

6. The Client can now connect to the Server and download the files.

5.4.2. Co-processor IPsec Offload

Encryption and decryption processing for IPsec can be offloaded to T6 adapters by the coprocessor driver (chcr). Both Tunnel and Transport IPsec modes are supported with ESP and AH protocols. The driver supports both Tx and Rx offload. Follow the steps mentioned below to configure co-processor IPsec offload using ip xfrm.

1. Load the Chelsio Co-processor driver.

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

2. Configure IP Alias on Chelsio interfaces for multiple tunnels (8 in this case).

```
[root@host~]# for i in `seq 1 8`; do ifconfig ethX:$i $i.0.0.1/24 up;done
```

3. Configure *ip xfrm* using below scripts.

DUT

```
for i in `seq 1 8`
do
   local_ip=$i.0.0.1
   remote_ip=$i.0.0.2

ip xfrm state add src $remote_ip dst $local_ip proto esp spi 0x54fa1f$i
reqid 16385 mode transport aead "rfc4106(gcm(aes))"
0x010203047aeaca3f87d060a12f4a4487d5a5c335 96 sel src 0.0.0.0/0 dst
0.0.0.0/0 replay-window 64 flag esn
ip xfrm state add src $local_ip dst $remote_ip proto esp spi 0x53fa1f$i
reqid 16386 mode transport aead "rfc4106(gcm(aes))"
0x010203047aeaca3f87d060a12f4a4487d5a5c335 96 sel src 0.0.0.0/0 dst
0.0.0.0/0 replay-window 64 flag esn
ip xfrm policy add src $local_ip dst $remote_ip dir out tmpl src $local_ip
dst $remote_ip proto esp reqid 16386 mode transport
ip xfrm policy add src $local_ip dst $remote_ip dir fwd tmpl src $local_ip
dst $remote_ip proto esp reqid 16385 mode transport
ip xfrm policy add src $local_ip dst $remote_ip dir in tmpl src $local_ip
dst $remote_ip proto esp reqid 16385 mode transport

done
```

PEER

```
for i in `seq 1 8`
do
   local_ip=$i.0.0.2
   remote_ip=$i.0.0.1

ip xfrm state add src $local_ip dst $remote_ip proto esp spi 0x54fa1f$i
reqid 16385 mode transport aead "rfc4106(gcm(aes))"
0x010203047aeaca3f87d060a12f4a4487d5a5c335 96 sel src 0.0.0.0/0 dst
0.0.0.0/0 replay-window 64 flag esn
```

```
ip xfrm state add src $remote_ip dst $local_ip proto esp spi 0x53fa1f$i
reqid 16386 mode transport aead "rfc4106(gcm(aes))"
0x010203047aeaca3f87d060a12f4a4487d5a5c335 96 sel src 0.0.0.0/0 dst
0.0.0.0/0 replay-window 64 flag esn
ip xfrm policy add src $local_ip dst $remote_ip dir out tmpl src $local_ip
dst $remote_ip proto esp reqid 16385 mode transport
ip xfrm policy add src $local_ip dst $remote_ip dir fwd tmpl src $local_ip
dst $remote_ip proto esp reqid 16386 mode transport
ip xfrm policy add src $local_ip dst $remote_ip dir in tmpl src $local_ip
dst $remote_ip proto esp reqid 16386 mode transport

done
```

4. The ip xfrm policies can be verified.

```
   [root@host~]# ip xfrm state list
```

5. Traffic can be run on the tunnels.

5.4.3. Co-processor counters

To verify whether 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
Cipher_ops: 64750285
Digest_ops: 13241
Aead_ops: 0
Completion: 64763501
Error: 0
Fallback: 0
IPSec Tx Inline: 0
```

5.5. Performance Tuning

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

Inline-TLS

1. Run the performance tuning script to update few kernel parameters using sysctl and map TOE queues to different CPUs.

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

2. Ensure that the application sends 8k PDU for best performance.

Co-processor

1. Run the performance tuning script to update few kernel parameters using sysctl and map crypto queues to different CPUs.

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

6. Software/Driver Unloading

To unload the 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. Refer to the Software/Driver Unloading section in **Network (NIC/TOE)** chapter for more information.

XXII. QUIC TX Offload

1. Introduction

QUIC is a modern transport protocol developed by Google, later standardized by the IETF (RFC 9000). Unlike TCP, it runs on top of UDP, but it provides TCP-like reliability, congestion control, and ordering.

The following are the main features:

- Multiplexed streams: Multiple data streams in a single connection without head-of-line blocking.
- Built-in encryption: Always uses TLS 1.3 for security (no unencrypted QUIC).
- Faster handshakes: 0-RTT/1-RTT connection setup (fewer round trips than TCP+TLS)
- Connection migration: Survives IP/port changes (e.g., switching from Wi-Fi to mobile data).
- Better performance on lossy/variable networks compared to TCP.

It has the following use cases:

- HTTP/3 (the latest version of HTTP) is built on QUIC.
- Streaming, gaming, and mobile apps increasingly use it for low-latency communication.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S72200

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the QUIC TX Offload feature is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Software/Driver Installation

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

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

- On the host, install network driver.

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

- Reboot your machine for changes to take effect.

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

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

3. Software/Driver Loading

Important *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.

4. Software/Driver Configuration

Configuring Chelsio T7 QUIC hardware offload on the server side involves two main steps:

1. Compilation of QUIC enabled OpenSSL
2. Compilation of Chelsio QUIC HW offload nginx

4.1. Compilation of QUIC enabled OpenSSL

Clone and compile QUIC enabled OpenSSL git repo as shown below:

Note *Perform the following configuration once.*

```
[root@host ~]# git clone https://github.com/quictls/openssl
[root@host ~]# cd openssl/
[root@host ~]# ./config shared --prefix=/root/openssl_build --
openssldir=/root/openssl_build -Wl,-rpath,/root/openssl_build/lib64
[root@host ~]# make
[root@host ~]# make install
```

4.2. Compilation of Chelsio QUIC HW offload nginx

The **nginx_quic** git repo must be used to use Chelsio QUIC offload solution. Follow the following steps to configure Chelsio QUIC Offload on the server end:

Note *Steps 1 to 4 need to be executed once.*

1. Clone Chelsio's nginx_quic repo, this has changes for QUIC encryption offload for chelsio T7:

```
[root@host ~]# git clone https://github.com/chelsiocrypto-26/nginx_quic.git
```

2. Compile nginx using the above compiled QUIC enabled OpenSSL.

```
[root@host ~]# cd nginx_quic
[root@host ~]# ./auto/configure --prefix=/usr/local/nginx --with-
http_v3_module --with-cc-opt="-I/root/openssl_build/include" --with-ld-
opt="-L/root/openssl_build/lib64 -L/usr/local/lib64 -lchquic -Wl,-
rpath,/root/openssl_build/lib64 -Wl,-rpath,/usr/local/lib64"
[root@host ~]# make
[root@host ~]# make install
```

3. Verify nginx compilation.

```
[root@host ~]# ldd /usr/local/nginx/sbin/nginx
        linux-vdso.so.1 (0x000074b70709f000)
        libchquic.so => /usr/local/lib64/libchquic.so (0x000074b707091000)
        libcrypt.so.1 => /lib/x86_64-linux-gnu/libcrypt.so.1
(0x000074b70704f000)
        libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0
(0x000074b706d66000)
        libssl.so.81.3 => /root/openssl_build/lib64/libssl.so.81.3
(0x000074b706cb4000)
        libcrypto.so.81.3 => /root/openssl_build/lib64/libcrypto.so.81.3
(0x000074b706600000)
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x000074b707031000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x000074b706200000)
        libnl-genl-3.so.200 => /lib/x86_64-linux-gnu/libnl-genl-3.so.200
(0x000074b707029000)
        libnl-3.so.200 => /lib/x86_64-linux-gnu/libnl-3.so.200
(0x000074b706c92000)
        /lib64/ld-linux-x86-64.so.2 (0x000074b7070a1000)

 [root@host ~]# /usr/local/nginx/sbin/nginx -V
 nginx version: nginx/1.31.4
 built by gcc 13.3.0 (Ubuntu 13.3.0-6ubuntu2~24.04.1)
 built with OpenSSL 3.1.7+quic 3 Sep 2024
 TLS SNI support enabled
 configure arguments: --prefix=/usr/local/nginx --with-http_v3_module --
 with-cc-opt=-I/root/openssl_build/include --with-ld-opt='-
 L/root/openssl_build/lib64 -L/usr/local/lib64 -lchquic -Wl,-
 rpath,/root/openssl_build/lib64 -Wl,-rpath,/usr/local/lib64'
```

4. The nginx.conf file for offload must be configured with below mentioned keywords:
   a. **quic_tx_offload x.**
   b. **quic_offload_dev ethX**

      Where,

      x is on or off

      "on" turns on Chelsio QUIC HW offload

      "off" turns off Chelsio QUIC HW offload

      ethX is the any interface name of a single card, i.e for a 4 port card, any one interface name out of the four interfaces.

Sample nginx.conf config file:

```
worker_processes auto;
events {
    worker_connections 4096;
}

http {
    include       mime.types;
    default_type  application/octet-stream;

    # QUIC-specific settings
    ssl_protocols TLSv1.3;
    ssl_prefer_server_ciphers off;

    server {
        listen 443;
        listen 443 quic reuseport;   # QUIC (HTTP/3) listener on UDP

        quic_gso on;
        # Chelsio T7 QUIC Setting
        quic_tx_offload on;
        quic_offload_dev ens6f4; #ens6f4 is the name of Chelsio T7 interface

        server_name localhost;

        ssl_certificate  /root/server.crt;
        ssl_certificate_key /root/server.key;

        # Required for QUIC
        ssl_session_cache off;
        ssl_early_data on;

     # Add Alt-Svc header to indicate HTTP/3 support
     add_header Alt-Svc 'h3=":443"'; # Required for HTTP/3
        location / {
            root   html;
            index  index.html index.htm;
        }
    }
}
```

5. Start Nginx using the command below. Note that this step must be performed after every host reboot if Nginx is required.

```
[root@host ~]# /usr/local/nginx/sbin/nginx -c
/usr/local/nginx/conf/nginx.conf
```

5. Client Application

The client application on the peer must be compiled with an OpenSSL version that has QUIC support enabled. While discussing all potential client applications is beyond the scope of Chelsio documentation, a single client application is described below for continuity.

5.1. Compilation of curl with QUIC enabled OpenSSL

This process involves the following two steps:

Note *Perform the following configuration once.*

1. Compile OpenSSL with QUIC support, nghttp3 and ngtcp2
2. Compile cURL using the custom OpenSSL build

5.1.1. Compilation of QUIC enabled OpenSSL, nghttp3 and ngtcp2

Perform the following steps to compile of QUIC enabled OpenSSL, nghttp3 and ngtcp2

1. Clone and compile QUIC enabled OpenSSL git repo as shown below:

```
[root@host ~]# git clone https://github.com/quictls/openssl
[root@host ~]# cd openssl/
[root@host ~]# ./config shared --prefix=/root/openssl_build --
openssldir=/root/openssl_build -Wl,-rpath/root/openssl_build/lib64
[root@host ~]# make
[root@host ~]# make install
```

2. Compile and install nghttp3.

```
[root@host ~]# git clone https://github.com/ngtcp2/nghttp3
[root@host ~]# cd nghttp3
[root@host ~]# git submodule update --init
[root@host ~]# autoreconf -fi
[root@host ~]# ./configure --prefix=/root/nghttp3_build --enable-lib-only
[root@host ~]# make
[root@host ~]# make install
```

3. Compile and install ngtcp2.

```
[root@host ~]# git clone https://github.com/ngtcp2/ngtcp2
[root@host ~]# cd ngtcp2/
[root@host ~]# autoreconf -fi
[root@host ~]# ./configure
PKG_CONFIG_PATH=/root/openssl_build/lib64/pkgconfig:/root/nghttp3_build/lib/
pkgconfig LDFLAGS="-Wl,-rpath,/root/openssl_build/lib64" --
prefix=/root/ngtcp2_build/ --enable-lib-only --with-openssl
[root@host ~]# make
[root@host ~]# make install
```

5.1.2 Compilation of cURL

1. Run the following command:

```
[root@host ~]# git clone https://github.com/curl/curl
[root@host ~]# cd curl/
[root@host ~]# autoreconf -fi
[root@host ~]# LDFLAGS="-Wl,-rpath,/root/openssl_build/lib64" ./configure --
with-openssl=/root/openssl_build/ --with-nghttp3=/root/nghttp3_build/ --
with-ngtcp2=/root/ngtcp2_build --prefix=/root/curl_build
[root@host ~]# make
[root@host ~]# make install
```

2. Verify cURL installation.

```
[root@host ~]# ldd /root/curl_build/bin/curl
        linux-vdso.so.1 (0x00007fccad0ab000)
        libcurl.so.4 => /root/curl_build/lib/libcurl.so.4
(0x00007fccacfc6000)
        libnghttp3.so.9 => /root/nghttp3_build/lib/libnghttp3.so.9
(0x00007fccacf9a000)
        libngtcp2_crypto_quictls.so.2 =>
/root/ngtcp2_build/lib/libngtcp2_crypto_quictls.so.2 (0x00007fccacf8e000)
        libngtcp2.so.16 => /root/ngtcp2_build/lib/libngtcp2.so.16
(0x00007fccacf39000)
        libpsl.so.5 => /lib64/libpsl.so.5 (0x00007fccacf13000)
        libssl.so.81.3 => /root/openssl_build/lib64/libssl.so.81.3
(0x00007fccace64000)
        libcrypto.so.81.3 => /root/openssl_build/lib64/libcrypto.so.81.3
(0x00007fccac800000)
        libzstd.so.1 => /lib64/libzstd.so.1 (0x00007fccacdb3000)
        libbrotlidec.so.1 => /lib64/libbrotlidec.so.1 (0x00007fccacda5000)
        libz.so.1 => /lib64/libz.so.1 (0x00007fccacd82000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fccac627000)
        libunistring.so.5 => /lib64/libunistring.so.5 (0x00007fccac47a000)
        libidn2.so.0 => /lib64/libidn2.so.0 (0x00007fccacd5e000)
        libbrotlicommon.so.1 => /lib64/libbrotlicommon.so.1
(0x00007fccacd3b000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fccad0ad000)

[root@host ~]# /root/curl_build/bin/curl --version
curl 8.22.0-DEV (x86_64-pc-linux-gnu) libcurl/8.22.0-DEV quictls/3.1.7
zlib/1.3 zstd/1.5.5 libpsl/0.21.2 ngtcp2/1.25.0-DEV nghttp3/1.1
Release-Date: [unreleased]
Protocols: dict file ftp ftps gopher gophers http https imap imaps ipfs ipns
mqtt mqtts pop3 pop3s rtsp smtp smtps telnet tftp ws wss
Features: alt-svc AsynchDNS HSTS HTTP2 HTTP3 HTTPS-proxy IPv6 Largefile libz
PSL SSL threadsafe TLS-SRP UnixSockets zstd
```

5.2. Using custom cURL to download files

Download files from QUIC HW offload enabled nginx server using the following command:

```
[root@host ~]# /root/curl_build/bin/curl -v -k --http3-only
https://102.0.0.72/test1M.bin -o test1M.bin
```

5.3. QUIC Offload Counters on Server-end

To verify whether Chelsio QUIC Offload is used, run the following command:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<PF4_id>/tls
Chelsio Inline TLS and DTLS Stats
TLS PDU Tx: 0
TLS PDU Rx: 0
DTLS PDU Tx: 0
DTLS PDU Rx: 0
TLS Keys (DDR) Count: 0

Chelsio Inline QUIC Stats
QUIC Tx NIC: 2052
QUIC Tx OFLD: 594872
QUIC Tx LSO: 39285549
QUIC No of conns: 1000
```

6. Software/Driver Unloading

A reboot is required to unload the TOE driver.

To avoid rebooting, please follow steps mentioned in Unloading the TOE Driver section.

XXIII. Inline IPSec Offload

1. Introduction

Inline IPsec Offload is a hardware acceleration feature that enables secure, high-speed data transmission by performing IPsec encryption and decryption directly on the network adapter, rather than in the host CPU. By offloading cryptographic processing, packet classification, and encapsulation tasks to the adapter, Inline IPsec Offload significantly reduces CPU utilization, minimizes latency, and enhances overall throughput. This approach ensures end-to-end data confidentiality and integrity while maintaining line-rate performance, making it ideal for secure communication in data centers, cloud infrastructures, and enterprise networks.

The following applications can be used to configure IPSec tunnels:

- ip xfrm
- strongSwan

Note *This is supported only on T7 family of adapters*.

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the Inline IPSec driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Kernel Configuration

2.1. Kernel Configuration

2.1.1. Kernel.org linux-6.18.X/6.12.X

If the IPSec components are not enabled, enable them as follows:

```
CONFIG_XFRM=y
CONFIG_XFRM_OFFLOAD=y
CONFIG_INET_ESP=y
CONFIG_INET_ESP_OFFLOAD=m
CONFIG_INET6_ESP=m
CONFIG_INET6_ESP_OFFLOAD=m
CONFIG_INET6_ESPINTCP=y
CONFIG_CRYPTO=y
CONFIG_CRYPTO_AEAD=y
CONFIG_CRYPTO_GCM=m
```

Note

- *DUT is the machine equipped with a T7 card.*
- *PEER is the corresponding companion machine.*
- *The PEER must run without hw offload support.*

RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4

No additional kernel configuration is required.

3. Software/Driver Loading

Perform the following steps to load the driver:

1. Load the Inline IPSec crypto offload driver.
   To run IPSec Offload in NIC Mode (without tcp offload support):

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

2. Bring up the Chelsio interface(s).

```
[root@DUT~]# ifconfig ethX <IP> up
```

Where *ethX* refers to the interface of T7 adapter.

4. Configuration

4.1. Using ip xfrm

1. Establish IPSec SAs and Policies using the commands below.
   a. On DUT

```
[root@DUT~]# ip xfrm state add src $DUT_IP dst $PEER_IP proto esp
spi 0x0a1b2c3d reqid 0x5f7c1a6b mode transport aead
"rfc4106(gcm(aes))" 0x0f83e4cb43bd443a839bef61378d294577386f72 128
sel src 0.0.0.0/0 dst 0.0.0.0/0 offload dev ethX dir out replay-
window 128 flag esn
[root@DUT~]# ip xfrm state add src $PEER_IP dst $DUT_IP proto esp
spi 0x4e5f6a7b reqid 0x9abcdef0 mode transport aead
"rfc4106(gcm(aes))" 0x0f83e4cb43bd443a839bef61378d294577386f72 128
sel src 0.0.0.0/0 dst 0.0.0.0/0 offload dev ethX dir in replay-
window 128 flag esn
[root@DUT~]# ip xfrm policy add src $DUT_IP dst $PEER_IP dir out
tmpl src $DUT_IP dst $PEER_IP proto esp reqid 0x5f7c1a6b mode
transport
[root@DUT~]# ip xfrm policy add src $PEER_IP dst $DUT_IP dir fwd
tmpl src $PEER_IP dst $DUT_IP proto esp reqid 0x9abcdef0 mode
transport
[root@DUT~]# ip xfrm policy add src $PEER_IP dst $DUT_IP dir in tmpl
src $PEER_IP dst $DUT_IP proto esp reqid 0x9abcdef0 mode transport
```

Note *ethX refers to the interface of DUT adapters.*

   b. On PEER

```
[root@PEER~]# ip xfrm state add src $DUT_IP dst $PEER_IP proto esp
spi 0x0a1b2c3d reqid 0x5f7c1a6b mode transport aead
“rfc4106(gcm(aes))” 0x0f83e4cb43bd443a839bef61378d294577386f72 128
sel src 0.0.0.0/0 dst 0.0.0.0/0 replay-window 128 flag esn
[root@PEER~]# ip xfrm state add src $PEER_IP dst $DUT_IP proto esp
spi 0x4e5f6a7b reqid 0x9abcdef0 mode transport aead
“rfc4106(gcm(aes))” 0x0f83e4cb43bd443a839bef61378d294577386f72 128
sel src 0.0.0.0/0 dst 0.0.0.0/0 replay-window 128 flag esn"
[root@PEER~]# ip xfrm policy add src $PEER_IP dst $DUT_IP dir out
tmpl src $PEER_IP dst $DUT_IP proto esp reqid 0x9abcdef0 mode
transport
```

```
[root@PEER~]# ip xfrm policy add src $DUT_IP dst $PEER_IP dir fwd
tmpl src $DUT_IP dst $PEER_IP proto esp reqid 0x5f7c1a6b mode
transport
[root@PEER~]# ip xfrm policy add src $DUT_IP dst $PEER_IP dir in
tmpl src $DUT_IP dst $PEER_IP proto esp reqid 0x5f7c1a6b mode
transport
```

2. Verify SA and policy status:

```
[root@DUT~]# ip xfrm state list
```

4.1.1. Using StrongSwan

1. Deploy certificates to DUT and PEER.

```
/usr/local/etc/swanctl/private/ # Private Keys of respective hosts
/usr/local/etc/swanctl/x509/    # Host Certificates of respective hosts
/usr/local/etc/swanctl/x509ca/  # CA Certificate
```

2. Configure "/usr/local/etc/strongswan.conf" (enable “make_before_break = yes”).

```
[root@DUT~]# cat /usr/local/etc/strongswan.conf
charon {
        load_modular = yes
        plugins {
                include strongswan.d/charon/*.conf
                make_before_break = yes       # <------- Append
"make_before_break" parameter
        }
}
```

Note *"make_before_break" ensures seamless tunnel rekeying by creating a new SA before tearing down the old one, preventing traffic disruption.*

3. Update */usr/local/etc/swanctl/swanctl.conf* on both the DUT and PEER according to their IP configurations.

a. DUT Machine

Note *Ensure "hw_offload = yes" on DUT.*

```
[root@DUT~]# cat /usr/local/etc/swanctl/swanctl.conf
connections {
    gw-port0 {
        local_addrs = <DUT_IP>
        remote_addrs = <PEER_IP>

        local {
            auth = pubkey
            certs = dut-Cert.der
            id = C=CH, O=test.ipsec.tunnel, CN=dut-common-name
        }

        remote {
            auth = pubkey
            id = C=CH, O=test.ipsec.tunnel, CN=peer-common-name
        }

        children {
            net-net-0 {
                rekey_time = 4000000
                esp_proposals = aes128gcm128-esn-noesn
                hw_offload = yes
                mode = tunnel
            }
        }

        version = 2
        mobike = no
        rekey_time = 8500000
        reauth_time = 9000000
        proposals = aes256-sha1-x25519
    }
}
```

b. PEER Machine

Note *Ensure `hw_offload = no` on PEER.*

```
[root@PEER~]# cat /usr/local/etc/swanctl/swanctl.conf
connections {
    gw-port0 {
        local_addrs = <PEER_IP>
        remote_addrs = <DUT_IP>

        local {
            auth = pubkey
            certs = peer-Cert.der
            id = C=CH, O=test.ipsec.tunnel, CN=peer-common-name
        }

        remote {
            auth = pubkey
            id = C=CH, O=test.ipsec.tunnel, CN=dut-common-name
        }

        children {
            net-net-0 {
                rekey_time = 5000000
                esp_proposals = aes128gcm128-esn-noesn
                hw_offload = no
                mode = tunnel
            }
        }

        version = 2
        mobike = no
        rekey_time = 11000000
        reauth_time = 12000000
        proposals = aes256-sha1-x25519
    }
}
```

4. Start and load configurations on both DUT and PEER Machines.

```
[root@DUT~]# /usr/local/libexec/ipsec/charon
[root@DUT~]# swanctl --load-creds
[root@DUT~]# swanctl --load-conns
```

5. Initiate connection from DUT.

```
[root@DUT~]# swanctl -i -c net-net-0
```

6. Verify connection.

```
[root@DUT~]# swanctl --list-sas -P
```

5. Software/Driver Unloading

To unload the driver, execute the following commands:

```
[root@DUT~]# rmmod ch_ipsec
```

XXIV. 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 supported Chelsio adapters:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the iSER driver is available for the following kernel versions:

- Kernel.org linux-6.12.104*
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* Kernel must be compiled on RHEL 10.X/9.8/9.7/9.6.

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

2. Kernel Configuration

Kernel.org linux-6.18.X/6.12.X

1. Ensure that the following iSER components are enabled in the kernel configuration file:

```
CONFIG_ISCSI_TARGET
CONFIG_INFINIBAND_ISER
CONFIG_INFINIBAND_ISERT
```

2. If the iSER components are not enabled, enable them as follows:

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

3. Compile and install the kernel, then boot into the new kernel and install the Chelsio Unified Wire.

RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4

No additional kernel configuration is required.

3. Software/Driver Installation

3.1. Pre-requisites

Ensure that the following requirements are met before driver installation:

- Uninstall any OFED present in the machine.
- The *rdma-core-devel* package should be installed on RHEL/Rocky/AlmaLinux 10.X/9.7/9.6, Ubuntu 24.04.4 systems.
- *targetcli* will be automatically installed by the Chelsio Unified Wire installer using the package manager yum/apt/zypper, if missing from the system. If you plan to use a different version, it is highly recommended to install **v2.1.fb29 or later**.

3.2. Installation

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

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

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

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

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

3. Reboot your machine to apply the changes.

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

4. Software/Driver Loading

Important *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:

1. Unload the Chelsio iWARP driver if previously loaded.

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

2. Load the following modules.

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

3. Start the iWARP Port Mapper Daemon.

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

4. Bring up the Chelsio interface(s).

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

5. 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

1. 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
```

2. Discover LIO target using the OpeniSCSI initiator.

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

3. Enable iSER support in the LIO target.

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

4. 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, ensure that Unified Wire is installed using the *Unified Wire (Default)* configuration tuning option. Currently, 256 IPv4/128 IPv6 iSER Initiator 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

1. Apply the performance settings mentioned in the Performance Tuning section in the **Unified Wire** chapter before proceeding.
2. Run the performance tuning script to update a few kernel parameters using sysctl and map iWARP queues to different CPUs.

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

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
```

XXV. 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 a 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 supported Chelsio adapters:

- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

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

- Kernel.org linux-6.12.104*
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* Kernel must be compiled on RHEL 10.X/9.8/9.7/9.6.

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

2. Software/Driver Installation

2.1. Pre-requisites

Ensure that the following requirements are met before driver installation:

- Uninstall any OFED present in the machine.
- The *rdma-core-devel* package should be installed on RHEL/Rocky/AlmaLinux 10.X/9.7.9.6 and Ubuntu 24.04.X systems.
- IOMMU should be disabled by adding *intel_iommu/amd_iommu=off* to the grub/grub2 kernel command line.

2.2. Installation

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

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

2. Install iWARP driver and WD-UDP libraries.

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

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

3. Reboot your machine for changes to take effect.

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

3. Software/Driver Loading

Important *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 may 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, 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 to 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 generates 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 *To offload IPv6 UDP sockets, select **low latency networking** as the 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:

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

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

2. Restart the service.

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

3. 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 more than 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
```

XXVI. 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.

Note *This is supported only on T6/T5 family of adapters*.

1.1. Hardware Requirements

1.1.1. Supported adapters

The following are the supported Chelsio adapters:

- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the DCB feature is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Software/Driver Installation

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

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

2. Build and install all drivers with DCB support.

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

Note *For more installation options, run* `make help`.

3. Reboot your machine for changes to take effect.

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

3. Software/Driver Loading

Important *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, 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 bring-up, 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:

1. Open a terminal configuration setting.

```
switch# config terminal
switch(config)#
```

2. 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
```

3. Configure queuing class-maps.

```
switch(config)#class-map type queuing class-nic
switch(config-cmap-que)#match qos-group 2
```

4. Configure network-qos class-maps.

```
switch(config)#class-map type network-qos class-nic
switch(config-cmap-nq)#match qos-group 2
```

5. 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
```

6. 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
```

7. 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.*

8. 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.

1. The following steps 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
```

2. The following steps help in creating a virtual fibre channel (VFC) and binds that VFC to an 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 ensure the ethernet port is part of both default VLAN and FCoE VLAN.*

3. 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
```

4. 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
```

5. 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.
```

6. If you have not created a zone, then ensure 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 following command:

```
switch(config)# zone default-zone permit vsan 2
```

4.2. Configuring the Brocade 8000 switch

1. 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
```

2. 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
```

3. 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
```

4. 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.*

5. Save the Configuration.

```
switch#copy running-config startup-config
```

5. Running NIC and iSCSI Traffic together with DCBx

Note *Refer to the iSCSI PDU Offload Initiator chapter to configure iSCSI Initiator.*

Use the following procedure to run NIC and iSCSI traffic together with DCBx enabled.

1. Identify the VLAN priority configured for NIC and iSCSI class of traffic on the switch.
2. Create VLAN interfaces for running NIC and iSCSI traffic and configure the corresponding VLAN priority.

Example:

The 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
```

XXVII. 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 integrates with the existing Fibre Channel networks and management software.

Note *This is supported only on T6/T5 family of adapters.*

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the FCoE full offload Initiator driver is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Software/Driver Installation

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

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

2. Install FCoE full offload initiator driver.

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

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

3. Reboot your machine for changes to take effect.

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

3. Software/Driver Loading

Important *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, refer to the 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.*
- *In order to identify a Chelsio fc_host from another vendor's 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, perform the following steps:

1. 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
```

2. 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 through the 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

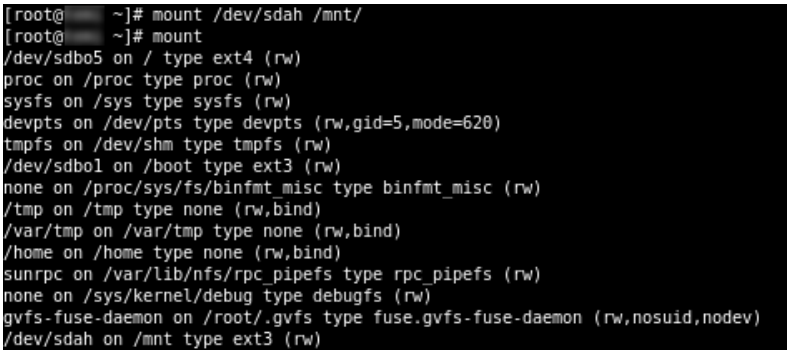
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

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.*

XXVIII. 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.

Note *This is supported only on T6/T5 family of adapters*.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the Offload Bonding driver is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Software/Driver Installation

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

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

2. Install the Chelsio Offload bonding driver.

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

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

3. Reboot your machine for changes to take effect.

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

3. Software/Driver Loading

Important *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:

1. Load the network driver with TOE support.

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

2. Create a bond interface.

```
[root@host~]# modprobe bonding mode=1 miimon=100 max_bonds=1
```

3. 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.*

4. 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
```

5. Disable FRTO on the PEER.

```
[root@host~]# sysctl -w net.ipv4.tcp_frto=0
```

6. 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
```

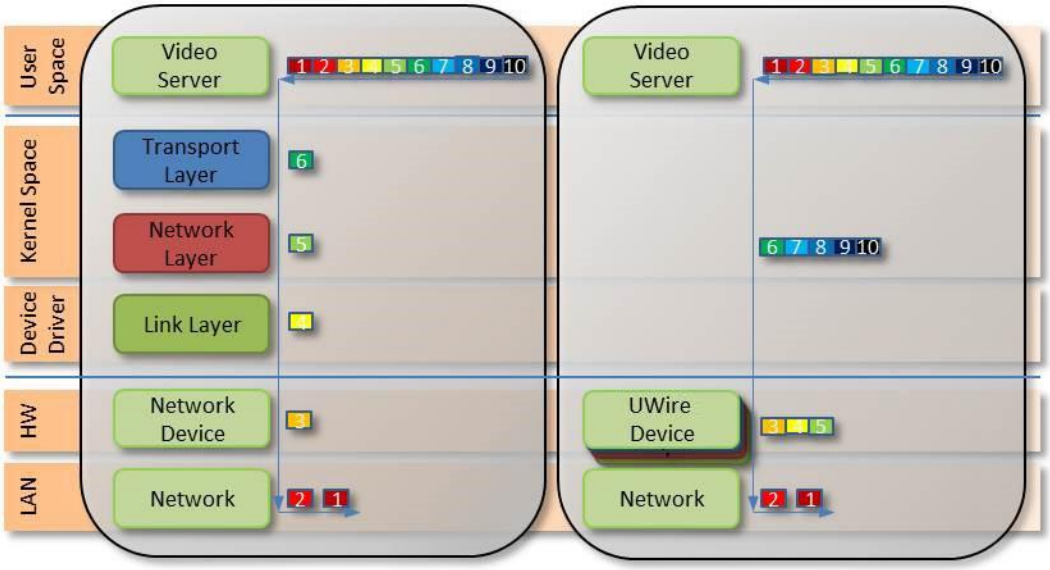
XXIX. UDP Segmentation Offload and Pacing

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).

Note *This is supported only on T6/T5 family of adapters*.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the UDP Segmentation Offload and Pacing driver is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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.*

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

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

2. Run the following command:

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

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

3. Reboot your machine for changes to take effect.

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

3. Software/Driver Loading

Important *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 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:

1. Determine the UDP socket file descriptor in the application through which data is sent.

2. 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 functions as per the parameters set for that traffic class.

3. Add socket option definitions.

To use *setsockopt()* to set the options to the UDP socket, set the following three definitions:

- 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
```

4. 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

5. Now, compile the application.

4.1.1. UDP offload functionality for RTP data

For the 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:

1. In step 2, 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.

2. In step 3, define a new macro, *UDP_RTPHEADERLEN* used for setting RTP header length with the value 292.

```
# define UDP_RTPHEADERLEN 292
```

3. In step 4, 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.

1. Bring up the network interface.

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

2. 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. 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 to the Unloading the TOE driver section of **Network (NIC/TOE)** chapter.

XXX. Offload IPv6

1. Introduction

The growth of the Internet has created a need for more addresses than 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 supported Chelsio adapters:

- T7 adapters:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the Offload IPv6 feature is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Software/Driver Installation

2.1. Pre-requisites

Ensure that the following requirements are met before driver 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

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

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

2. Install Unified Wire with IPv6 support.

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

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

3. Reboot your machine for changes to take effect.

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

3. Software/Driver Loading

Important *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

1. Load the Offload capable drivers.

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

2. 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.

3. Configure the required IPv6 address.

```
[root@host~]# ifconfig ethX inet6 add <IPv6 address>
```

4. 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

Reboot the system to unload the TOE driver. To unload without rebooting, refer to the Unloading the TOE driver section of **Network (NIC/TOE)** chapter.

XXXI. WD Sniffing and Tracing

1. Theory of Operation

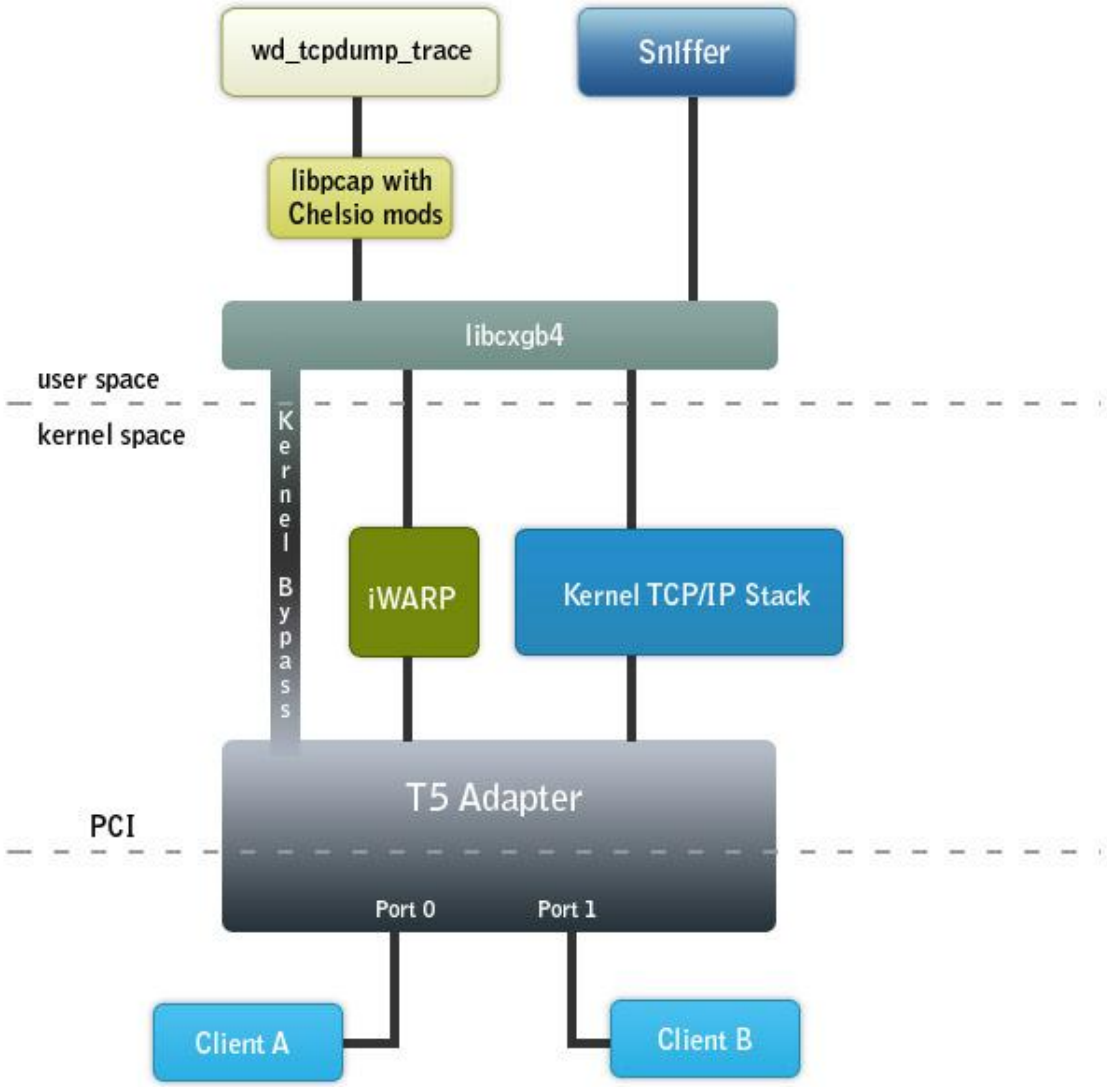
The objective of these utilities (*wd_sniffer* and *wd_tcpdump_trace*) is to provide sniffing and tracing capabilities by making use of the Chelsio adapter's hardware features.

- Sniffer is a tool to measure bandwidth and involves targeting specific multicast traffic and sending it directly to user space.
    1. Get a Queue (raw QP) idx.
    2. 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 through the internal loop back mechanism.
    1. Get a Queue (raw QP) idx.
    2. Set the adapter in loop back.
    3. Connect Client A and B to ports 0 and 1 or ports 2 and 3.
    4. 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.

Note *This is supported only on T6/T5 family of adapters*.



Schematic diagram of sniffer and tracer

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the WD Sniffing and Tracing utility is available for the following kernel versions:

- Kernel.org linux-6.12.104*
- Kernel.org linux-6.18.39*
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

* Kernel must be compiled on RHEL 10.X/9.8/9.7/9.6.

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

2. Software/Driver Installation

2.1. Pre-requisites

Ensure that the following requirement is met before driver installation:

- IOMMU should be disabled by adding *intel_iommu/amd_iommu=off* to the grub/grub2 kernel command line.

2.2. Installation

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

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

2. Install Sniffer and Tracer utilities and iWARP driver.

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

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

3. 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 three systems with one machine having a Chelsio card with two or more ports. The machines should be set up 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 it is 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 following options can be provided additionally to capture more packets.

```
[root@host~]# wd_tcpdump_trace -i <iface> -s 64 -B 1024000 -w capture.pcap
```

Note *Refer to* `wd_tcpdump_trace -h` *for more information on the above options.*

XXXII. Classification and Filtering

1. Introduction

Classification and Filtering feature enhances network security by controlling incoming traffic as they pass through the 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 supported Chelsio adapters:

- T7 adapters:
  - T72200-FH
  - T72200
  - T7450
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the Classification and Filtering feature is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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. For 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.

1. If you have not done already, run the Unified Wire Installer with the appropriate configuration tuning option to install the network driver.
2. Load the network driver and bring up the Chelsio interface.

```
[root@host~]# modprobe cxgb4
[root@host~]# ifconfig ethX up
```

3. 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 to the 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

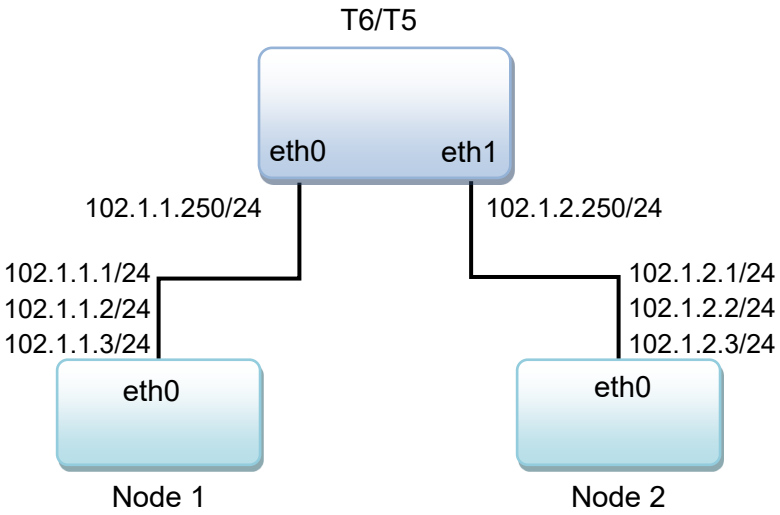
To remove a filter, run the following command with the corresponding filter rule index:

```
[root@host~]# cxgbtool ethX filter <index> <delete|clear>
```

Note *For more information on additional parameters, refer to the cxgbtool manual by running the `man cxgbtool` command*

2.5. Layer 3 Example

The following example illustrates how to achieve Layer 3 (L3) routing functionality:



- **Follow these steps on Node 1**

1. 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
```

2. 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**

1. Configure IP address and enable the three 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
```

2. 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**

1. 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
```

2. 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.

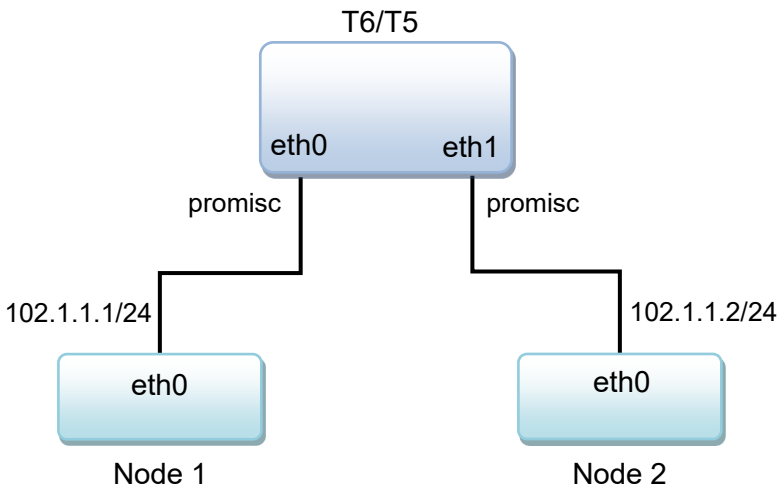
3. Create filter rule to send packets for 102.1.1.0/24 network out via eth0 interface.

```
[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 is an example 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**

1. 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
```

2. Unload and re-load the *cxgb4* driver.
3. Enable promiscuous mode on both the interfaces on T6/T5 adapter machine.

```
[root@host~]# ifconfig eth0 up promisc
[root@host~]# ifconfig eth1 up promisc
```

4. Build and install the latest iproute2 package.
5. 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
```

6. 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
```

7. 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
```

8. 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
```

9. 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 instantiating them, refer to the* 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 ffffffffffff    -       -      -    N      -        0      0     Y   0x3   7 104
  1 00:00:00:00:00:00 ffffffffffff    -       -      -    N      -        0      0     Y   0x3   4  68
  2 00:07:43:3c:a8:40 ffffffffffff    -       -      -    N      -        0      0     Y   0x1   4  68
  3 01:00:5e:00:00:01 ffffffffffff    -       -      -    N      -        0      0     Y   0x3   4  68
  4 00:07:43:3c:a8:48 ffffffffffff    -       -      -    N      -        0      0     Y   0x2   4  69
  5 33:33:00:00:00:01 ffffffffffff    -       -      -    N      -        0      0     Y   0x3   4  68
  6 33:33:ff:3c:a8:40 ffffffffffff    -       -      -    N      -        0      0     Y   0x1   4  68
  7 33:33:ff:3c:a8:48 ffffffffffff    -       -      -    N      -        0      0     Y   0x2   4  69
  8 06:44:3c:a8:40:00 ffffffffffff    -       -      -    N      -        0      0     Y   0x1   0   0
  9 06:44:3c:a8:40:01 ffffffffffff    -       -      -    N      -        0      0     Y   0x1   0   1
 10 06:44:3c:a8:40:02 ffffffffffff    -       -      -    N      -        0      0     Y   0x1   0   2
 11 06:44:3c:a8:40:03 ffffffffffff    -       -      -    N      -        0      0     Y   0x1   0   3
 12 06:44:3c:a8:40:04 ffffffffffff    -       -      -    N      -        0      0     Y   0x1   0   4
 13 06:44:3c:a8:40:05 ffffffffffff    -       -      -    N      -        0      0     Y   0x1   0   5
 14 06:44:3c:a8:40:06 ffffffffffff    -       -      -    N      -        0      0     Y   0x1   0   6
 15 06:44:3c:a8:40:07 ffffffffffff    -       -      -    N      -        0      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:

1. 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
```

2. 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)
```

3. Observe the VF id for the MAC address *06:44:3c:a8:40:02* in mps_tcam.
4. 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

To create the large number of filters, select one of the following 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.

1. If you have not done already, run the Unified Wire Installer with the *High Capacity Hash Filter* or *Unified Wire (Default)* configuration tuning option to install the drivers.
2. 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
```

3. 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. For 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 an 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 do not 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. The 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**

1. Select a filterMode from the supported filterMode table based on your requirement.

```
filterMode = fragmentation, mpshittype, protocol, vlan, port, fcoe
```

2. Select a filterMask so that it is a subset of or equal to filterMode based on application.

```
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.

3. 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 you do not want to create filter rule based on any particular tuple from filterMode, do not select it in filterMask. For example, if you do not want to create a filter based on VLAN value, then do not select it from filterMode to filterMask.

1. Select a filterMode from the supported filterMode table based on your requirement.

```
filterMode = fragmentation, mpshittype, protocol, vlan, port, fcoe
```

2. Select a filterMask so that it is a subset of or equal to filterMode based on application without VLAN tuple.

```
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.

3. 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 to the 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 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 to the cxgbtool manual by running 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. The following is 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:

1. 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.
2. 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
```

3. 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
```

4. 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:

1. Obtain the PF and VF values of the incoming port from */sys/kernel/debug/cxgb4/<bus-id>/mps_tcam*
2. 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:

1. 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
```

2. 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.

3. 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

3.8.1. For T7/S7 Family of Adapters

- **S7 single-port adapter**

The following is a step-by-step guide to enable packet tracing on S7 single-port adapter:

1. Load the network driver with the following parameters:

```
[root@host~]# modprobe cxgb4 enable_traceq=1
```

2. Enable tracing on T7 adapter.

```
[root@host~]# cxgbtool -i ethX 0x9800=0x13
```

3. Determine the RspQ ID of the queues by looking at Trace QType in */sys/kernel/debug/cxgb4/<bus-id>/sge_qinfo* file.

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<bus_id>/sge_qinfo  | grep -i
trace -A 5
```

Example:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/0000\:99\:00.4/sge_qinfo | grep -i trace -A
5
QType:                TRACE          TRACE           TRACE           TRACE
Interface:          ens5f4         ens5f4          ens5f4          ens5f4
RspQ ID:              53             54              55              56
RspQ size:          1023           1023            1023            1023
RspQE size:           64             64              64              64
RspQ CIDX:             0              0               0               0
```

Note *Use '53' from the above example as the RspID for the next step.*

4. Set up a trace filter.

```
[root@host~]# echo "tx0 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace0
[root@host~]# echo "rx0 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace4
[root@host~]# echo "tx1 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace1
[root@host~]# echo "rx1 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace5
[root@host~]# echo "tx2 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace2
[root@host~]# echo "rx2 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace6
[root@host~]# echo "tx3 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace3
[root@host~]# echo "rx3 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace7
```

This step enables packet tracing for all ingress and egress traffic on port0 for capture at port0.

- **T7/S7 dual-port adapter**

The following is a step-by-step guide to enable packet tracing on T7/S7 Dual-port adapter:

1. Load the network driver with the following parameters:

```
[root@host~]# modprobe cxgb4 enable_traceq=1
```

2. Enable tracing on T7 adapter.

```
[root@host~]# cxgbtool -i ethX 0x9800=0x13
```

3. Determine the RspQ ID of the queues by looking at Trace QType in */sys/kernel/debug/cxgb4/<bus-id>/sge_qinfo* file.

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<bus_id>/sge_qinfo  | grep -i
trace -A 5
```

Example:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/0000\:99\:00.4/sge_qinfo | grep -i trace -A
5
QType:                   TRACE           TRACE           TRACE           TRACE
Interface:             ens5f4          ens5f4        ens5f4d1        ens5f4d1
RspQ ID:                   55              56              57              58
RspQ size:               1023            1023            1023            1023
RspQE size:                64              64              64              64
RspQ CIDX:                  0               0               0               0
```

Note *Use '53' from the above example as the RspID for the next step.*

4. Configure RSS Queue to send trace packets.

```
[root@host~]# cxgbtool -i ethX 0x0a434=<Trace Queue0-RspQ ID>
```

Example:

```
[root@host~]# cxgbtool -i ens5f4 0x0a434=53
```

5. Set up a trace filter.

For port0-

```
[root@host~]# echo "tx0 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace0
[root@host~]# echo "rx0 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace4
[root@host~]# echo "tx1 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace1
[root@host~]# echo "rx1 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace5
```

For port1 –

```
[root@host~]# echo "tx2 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace2
[root@host~]# echo "rx2 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace6
[root@host~]# echo "tx3 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace3
[root@host~]# echo "rx3 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace7
```

This step enables packet tracing for all ingress and egress traffic on port0 and port1 for capture at port0.

- **T7/S7 quad-port adapter**

The following is a step-by-step guide to enable packet tracing on T7/S7 Dual-port adapter:

1. Load the network driver with the following parameters:

```
[root@host~]# modprobe cxgb4 enable_traceq=1
```

2. Enable tracing on T7 adapter.

```
[root@host~]# cxgbtool -i ethX 0x9800=0x13
```

3. Determine the RspQ ID of the queues by looking at Trace QType in */sys/kernel/debug/cxgb4/<bus-id>/sge_qinfo* file.

```
[root@host~]# cat /sys/kernel/debug/cxgb4/<bus_id>/sge_qinfo  | grep -i
trace -A 5
```

Example:

```
[root@host~]# cat /sys/kernel/debug/cxgb4/0000\:99\:00.4/sge_qinfo  | less | grep -i
trace -A 5
QType:                   TRACE            TRACE            TRACE            TRACE
Interface:             ens5f4         ens5f4d1         ens5f4d2         ens5f4d3
RspQ ID:                   53               54               55               56
RspQ size:               1023             1023             1023             1023
RspQE size:                64               64               64               64
RspQ CIDX:                  0                0                0                0
```

Note *Use '53' from the above example as the RspID for the next step. .*

4. Configure RSS Queue to send trace packets.

```
[root@host~]# cxgbtool -i ethX 0x0a434=<Trace Queue0-RspQ ID>
```

Example:

```
[root@host~]# cxgbtool -i ens5f4 0x0a434=53
```

5. Set up a trace filter.

   For port0-

```
[root@host~]# echo "tx0 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace0
[root@host~]# echo "rx0 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace4
```

   For port1 –

```
[root@host~]# echo "tx1 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace1
[root@host~]# echo "rx1 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace5
```

   For port2-

```
[root@host~]# echo "tx2 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace2
[root@host~]# echo "rx2 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace6
```

   For port3 –

```
[root@host~]# echo "tx3 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace3
[root@host~]# echo "rx3 snaplen=2048" > /sys/kernel/debug/cxgb4/<bus_id>/trace7
```

This step enables packet tracing for all ingress and egress traffic on port0, port1, port2 and port3 for capture at port0.

3.8.2. For T5/T6 Family of Adapters

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. The following is a step-by-step guide to enable packet tracing and *hit counters* for T5 Hash/DDR filter rules:

1. Load the network driver with the following parameters:

```
[root@host~]# modprobe cxgb4 use_ddr_filters=1 enable_traceq=1
```

2. Configure the required filter rules.
3. Enable tracing on T5 adapter.

```
[root@host~]# cxgbtool ethX reg 0x09800=0x13
```

4. Set up 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.

5. 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.

1. Enable Tracing as well as MultiRSSFilter.

```
[root@host~]# cxgbtool ethX reg 0x09800=0x33
```

2. Setup a trace filter.

```
[root@host~]# echo tx0 snaplen=40 > /sys/kernel/debug/cxgb4/<bus_id>/trace0
```

3. 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>
```

4. 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, that is 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
```

XXXIII. 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.

Note *This is supported only on T6/T5 family of adapters*.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the supported Chelsio adapters:

- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T6225-CR
- T5 adapters:
  - T540-LP-CR
  - T520-CR

1.2. Software Requirements

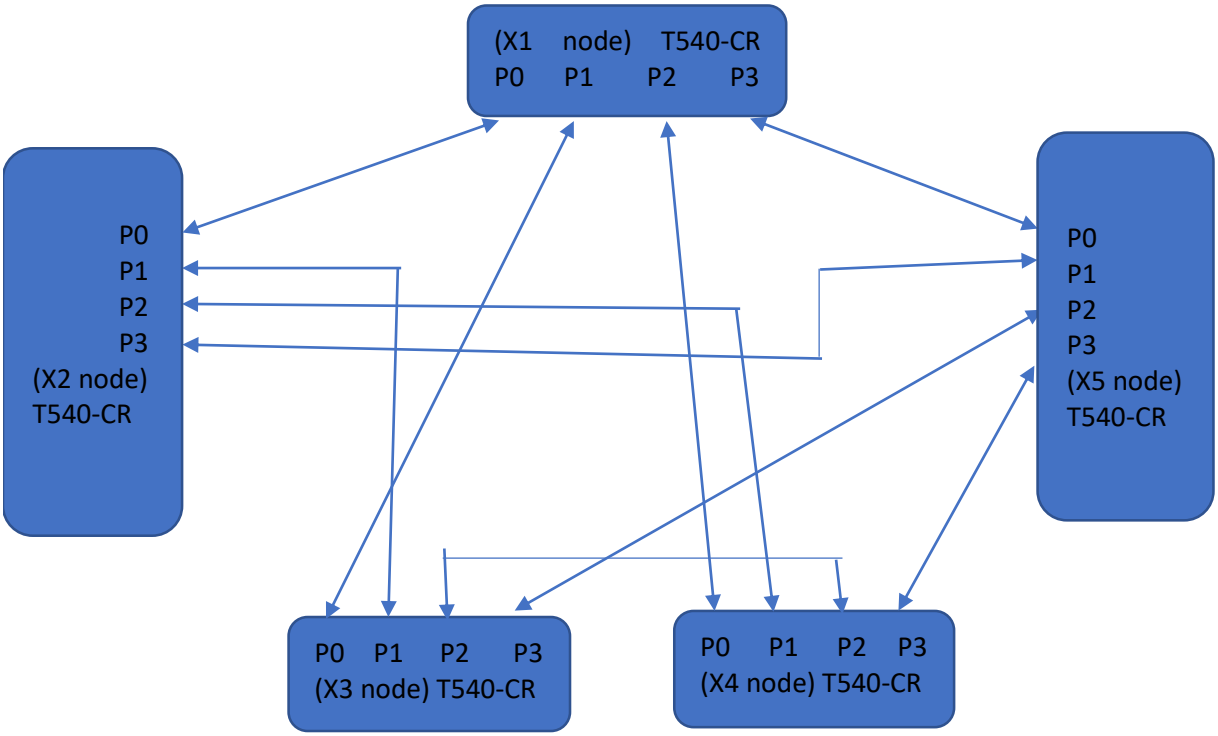
Currently, the Mesh topology is available for the following Linux kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

Other kernel 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. 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.

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

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

2. Install the drivers, tools, and libraries.

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

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

3. 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 following steps:

Important *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
```

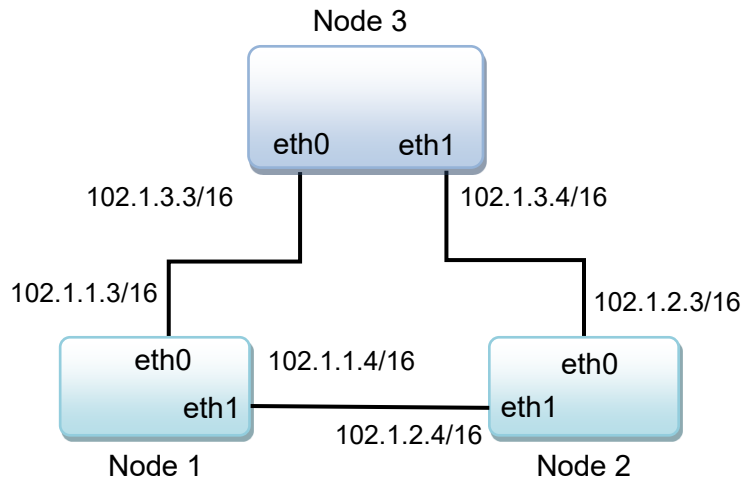
1. Load network driver (*cxgb4*).

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

2. 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, refer to their respective sections for protocol configuration.

Example:
Three 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
```

XXXIV. 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 the 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 supported Chelsio adapters:

- T7 adapters:
  - T72200-FH
  - T72200
  - T7450
  - S72200*
  - S71400*
  - S7450*
  - S7250*
- T6 adapters:
  - T62100-CR
  - T62100-LP-CR
  - T62100-SO-CR*
  - T62100-SO-OCP3*
  - T6225-CR
  - T6225-SO-CR*

- T5 adapters:
  - T540-LP-CR
  - T520-CR
  - T520-SO-CR*

* Only Non-offloaded connections are supported.

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the Traffic Management feature is available for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Software/Driver Loading

Important *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 following 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. At least one class should be specified while configuring Channel Rate Limiting using the following 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 parameters 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 two-step process. In the first step, bind connections to the 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.

1. Load the network driver and bring up the interface.

```
[root@host~]# modprobe cxgb4
[root@host~]# ifconfig ethX up
```

2. 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*

3. 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**

1. Load the TOE driver and bring up the interface.

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

2. 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

3. Compile the policy file using *COP.*

```
[root@host~]# cop -d –o <output_policy_file> <new_policy_file>
```

4. 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:

1. Determine the TCP socket file descriptor in the application through which data is sent.
2. 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.

3. 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
```

4. 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.

5. 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:

1. Load the network driver and bring up the interface.

```
[root@host~]# modprobe cxgb4
[root@host~]# ifconfig eth0 up
```

2. 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
```

3. Bind the NIC TX queue to class 0.

```
[root@host~]# cxgbtool eth0 sched-queue 3 0
```

4. 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:

1. Load the TOE driver and bring up the interface.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig eth0 up
```

2. 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
```

3. Compile the policy file using *COP.*

```
[root@host~]# cop -d -o <output_policy_file> <new_policy_file>
```

4. 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
```

5. 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 the modified application.

1. Load the TOE driver and bring up the interface.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig eth0 up
```

2. Modify the application as mentioned in the Configuring Traffic Management section.

3. 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

Refer to Inline TLS Offload chapter for the configuration steps. To rate limit Inline TLS Offload connections, follow the steps mentioned below:

1. Load the TOE driver and bring up the interface.

```
[root@host~]# modprobe t4_tom
[root@host~]# ifconfig eth0 up
```

2. 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.

3. Compile the policy file using *COP.*

```
[root@host~]# cop -d -o <output_policy_file> <new_policy_file>
```

4. 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
```

5. 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.

XXXV. 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/TOE).

This section of the guide explains how to configure and use Chelsio Unified Boot Option ROM which flashes PXE Option ROM onto Chelsio's T7 adapters. It adds functionalities like PXE boot.

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the currently shipping T7 Chelsio adapters that are compatible with Chelsio Unified Boot software:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S71400
  - S72200
  - S7450
  - S7250

1.1.2. Supported platforms for T7

The following is the list of hardware platforms supported by Chelsio Unified Boot software:

- Supermicro X13SEI-F
- Supermicro X13DEI-F
- Supermicro X13SWA-TF
- Supermicro X11SSL-CF
- Dell PowerEdge R760
- Gigabyte MZ73-LM0-000

  * If system BIOS version is lower than 1.5 and both Legacy and uEFI are enabled, system will hang during POST. Upgrade the BIOS version to 1.5 or higher to avoid this issue.

1.1.3. Supported Switches for T7

The following is the list of network switches supported by Chelsio Unified Boot software:

- Arista DCS-7050DX4-32S-F

  Other platforms/switches have not been tested and are not guaranteed to work.

1.2. Software Requirements

The installation of the following Linux distributions is supported:

- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64

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

1.3. Pre-requisites

- A USB flash drive 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 Chelsio Drivers are in-boxed in major Linux Distributions mentioned in the Software Requirements and can be used for OS installation after enabling Secure Boot in System BIOS.

Note *Currently, the OptionROM does not include a digital signature.*

3. Chelsio T7 Adapter

3.1. Flashing firmware, option ROM and boot.cfg

Chelsio Unified Boot provides utility to flash firmware, optionROM and boot config onto Chelsio T7 adapters. Depending on the boot mode selected, Chelsio Unified Boot provides the following methods to flash Firmware, Option ROM, and boot configuration onto the Chelsio adapters:

- Legacy mode: cfut4
- uEFI mode:
  - drvcfg

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 T7 Chelsio adapters present in the system.*

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:

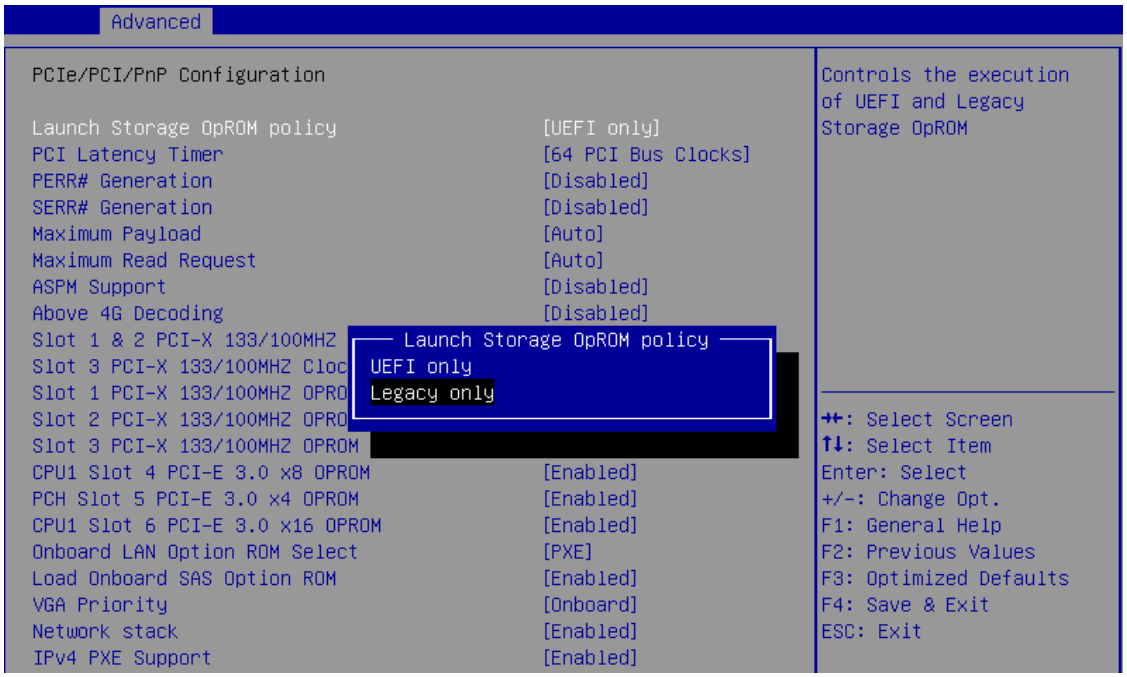
1. Create a DOS bootable USB flash drive. (Click here for instructions)
2. Create a directory *CHELSIO* on the USB flash drive.
3. If you have not done already, then download the driver package from Chelsio Download Center.
4. Extract the downloaded package and change your working directory to *OptionROM* directory.

```
[root@host~]# tar zxvfm ChelsioUwire-x.x.x.x.tar.gz
[root@host~]# cd ChelsioUwire-x.x.x.x/Uboot/OptionROM
```

5. Copy all the files and place them in the *CHELSIO* directory created on the USB flash drive.
6. Plug-in the USB flash drive in the system on which the Chelsio CNA is installed.
7. Reboot the system.

3.1.1. Legacy

1. In BIOS, configure the system which has the Chelsio adapter to boot in Legacy mode.



2. Boot the system from the plugged in USB flash drive and change your working directory to the CHELSIO directory.

```
C:\>cd CHELSIO
```

3. Run the following command to list all the Chelsio adapters present on the system. The list displays a unique index for each adapter found.

```
C:\CHELSIO>cfut4 -l
```

```
C:\CHELSIO>CFUT4.EXE  -l

 Chelsio T5/T6/T7 Flash Utility v1.5


 Index  ChelsioAdaptertype   DevId
 =====  ==================   =====
   [0]   T72200              7005
C:\CHELSIO>_
```

4. Delete any previous version of Option ROM, boot.cfg and firmware flashed onto the adapter.

```
C:\CHELSIO>cfut4 -d <idx> -xb -xc -xf
```

Here, `idx` is the adapter index found in step iii (0 in this case). In the below example “-d” is omitted

```
C:\CHELSIO>cfut4 -xb -xc -xf

 Chelsio T5/T6/T7 Flash Utility v1.5

 Erasing serial flash sector(s) ... Done
 Erasing serial flash sector(s) ... Done
 Erasing serial flash sector(s) ... Done
```

5. 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 -uf t7fw-2~1.bin

 Chelsio T5/T6/T7 Flash Utility v1.5

 Erasing serial flash sector(s) ... 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
 Writing Image at Base 00070000 ... Done
 Writing Image at Base 00078000 ... 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 ... /
```

6. Flash the Unified Boot Option ROM using the following command.

```
C:\CHELSIO>cfut4 -d <idx> -ub cubt4.bin
```

Here, `cubt4.bin` is the Unified Boot Option ROM image file present in the *CHELSIO* directory.

```
C:\CHELSIO>CFUT4.EXE  -ub cubt4.bin

 Chelsio T5/T6/T7 Flash Utility v1.5

 Erasing serial flash sector(s) ... Done
 Writing Image at Base 00200000 ... Done
 Writing Image at Base 00208000 ... Done
 Writing Image at Base 00210000 ... Done
 Writing Image at Base 00218000 ... Done
 Writing Image at Base 00220000 ... Done
 Writing Image at Base 00228000 ... Done
 Writing Image at Base 00230000 ... Done
 Writing Image at Base 00238000 ... Done
 Writing Image at Base 00240000 ... Done
 Writing Image at Base 00248000 ... Done
 Writing Image at Base 00250000 ... Done
 Writing Image at Base 00258000 ... Done
 Writing Image at Base 00260000 ... Done
 Writing Image at Base 00268000 ... Done
 Writing Image at Base 00270000 ... Done
 Writing Image at Base 00278000 ... Done
 Writing Image at Base 00280000 ... Done
 Writing Image at Base 00288000 ... \_
```

7. Flash the boot configuration setting which will enable PXE and disable iSCSI and FCoE.

```
C:\CHELSIO>cfut4 -d <idx> -uc boot.cfg
```

```
C:\CHELSIO>cfut4.exe  -uc boot.cfg

 Chelsio T5/T6/T7 Flash Utility v1.5

 Erasing serial flash sector(s) ... Done
 Writing Image at Base 00300000 ... Done
C:\CHELSIO>_
```

8. Verify the Option ROM, boot.cfg and firmware flashed on the adapter using the following command:

```
C:\CHELSIO>cfut4 -vf -vc -vb
```

```
C:\CHELSIO>cfut4.exe -vf -vc -vb

 Chelsio T5/T6/T7 Flash Utility v1.5

 Controller [0] Fw Version - 2.1.25.0
 Controller [0] OpRom Version - 3.0.0.8
 Controller [0] Config Version - 3.0.0.8
```

9. In case of multiple adapters in the system, repeat the steps from 4 through 8. to update/flash the firmware, Option ROM, and boot configuration on all of them.
10. 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
```

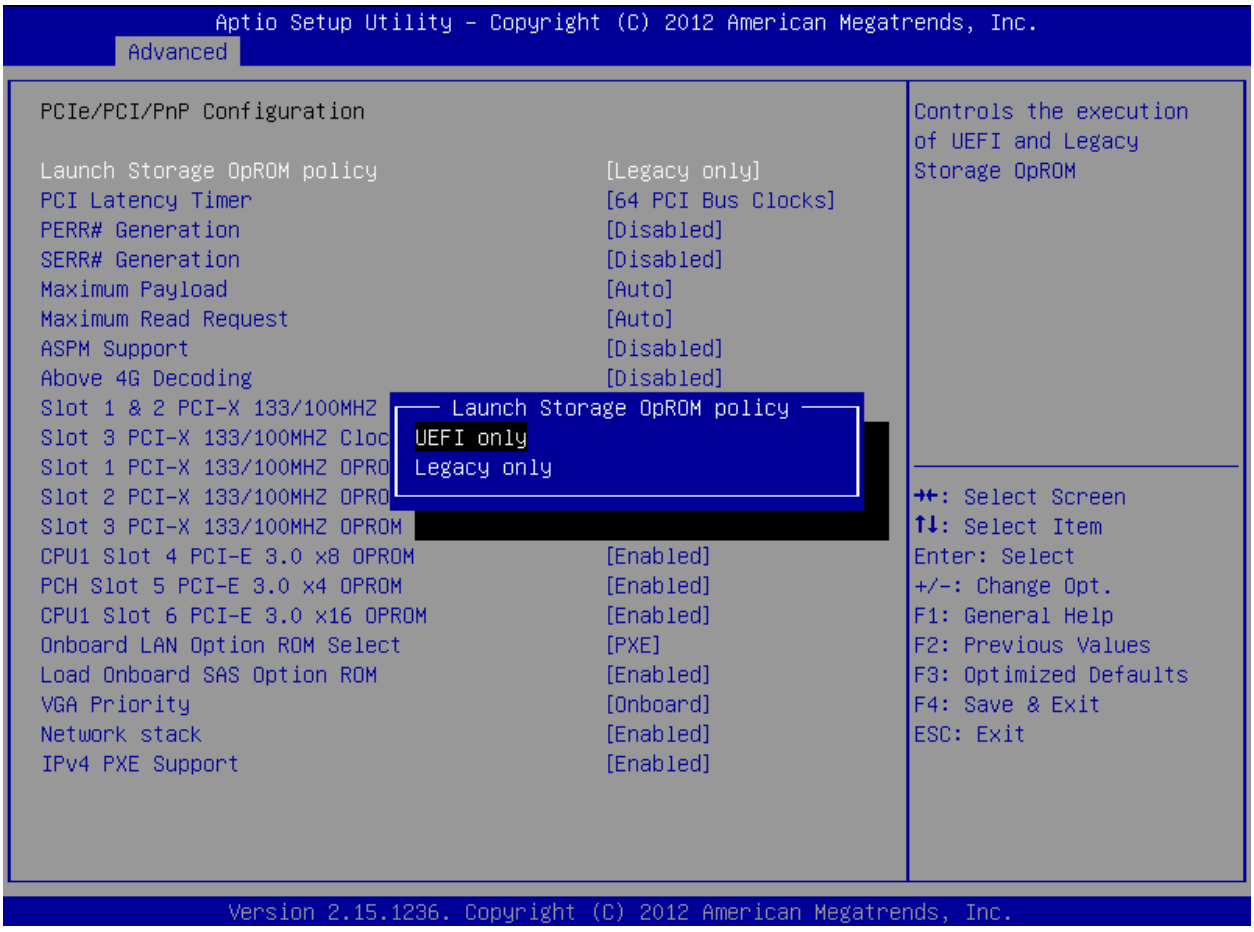
11. Reboot the system for changes to take effect.

3.1.2. uEFI

i. Reboot the system and go into BIOS setup.

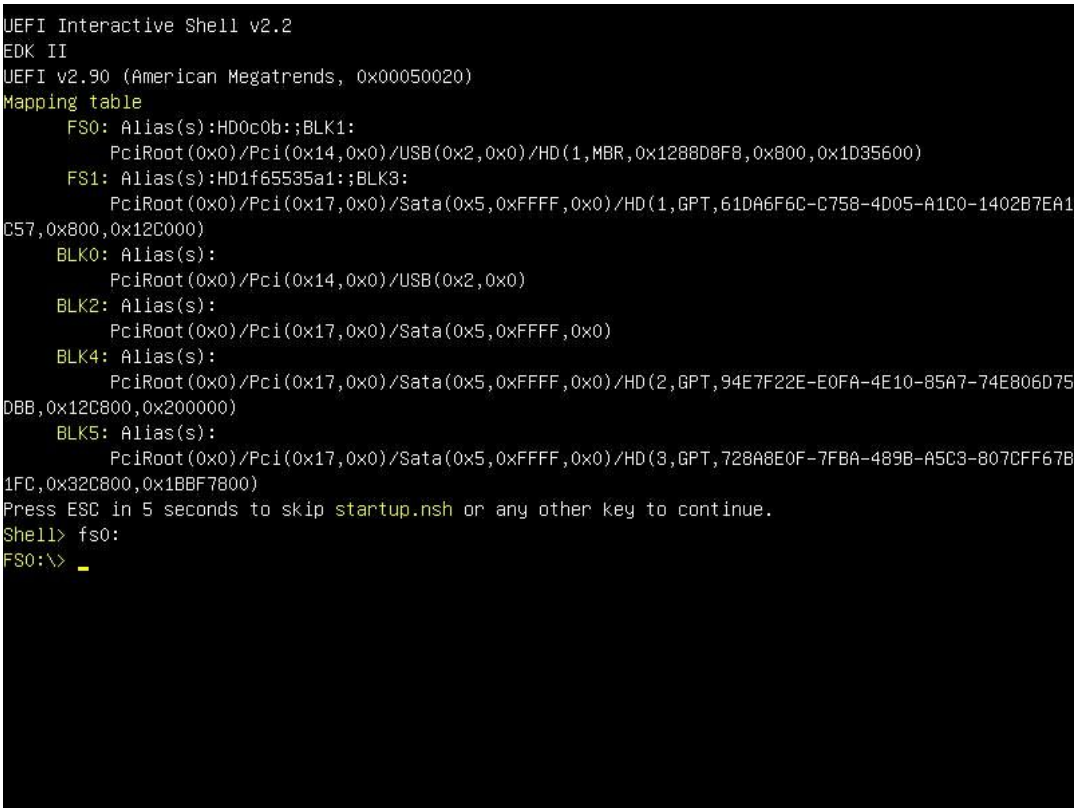
ii. Disable Secure Boot.

iii. Configure the system having Chelsio CNA to boot in uEFI mode.

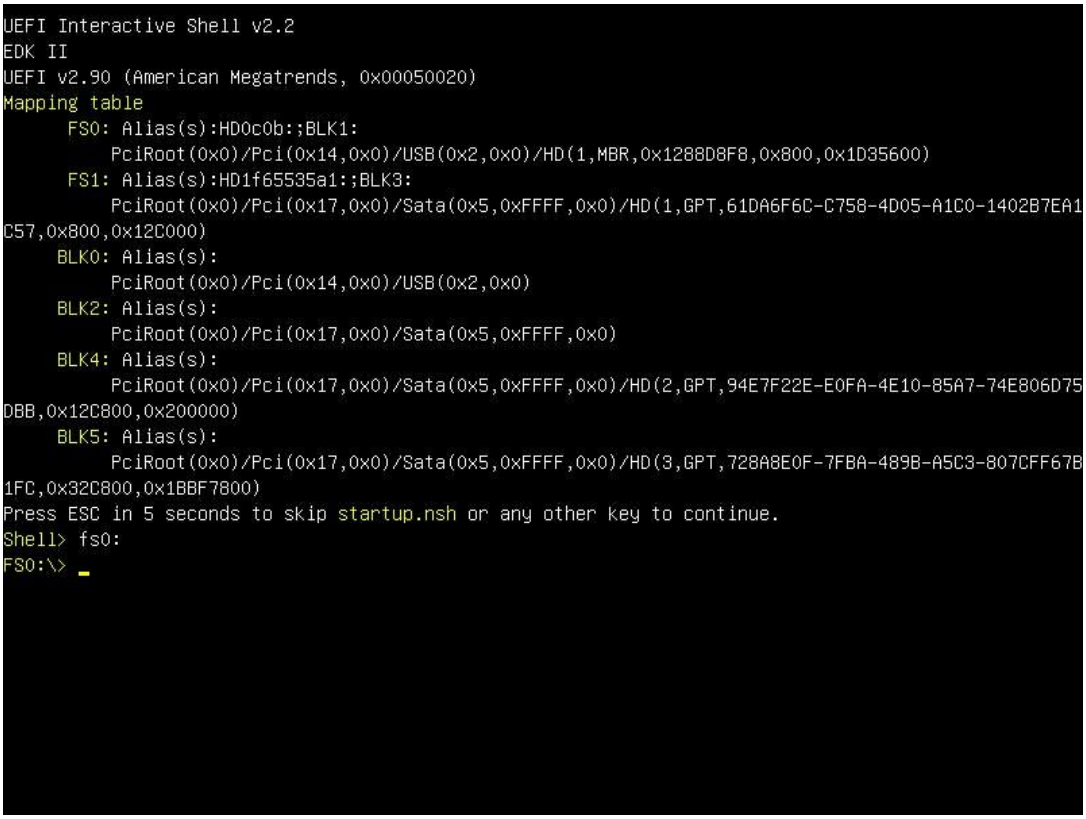


Note *For Supermicro systems, enable **Network Stack*** as well before proceeding.

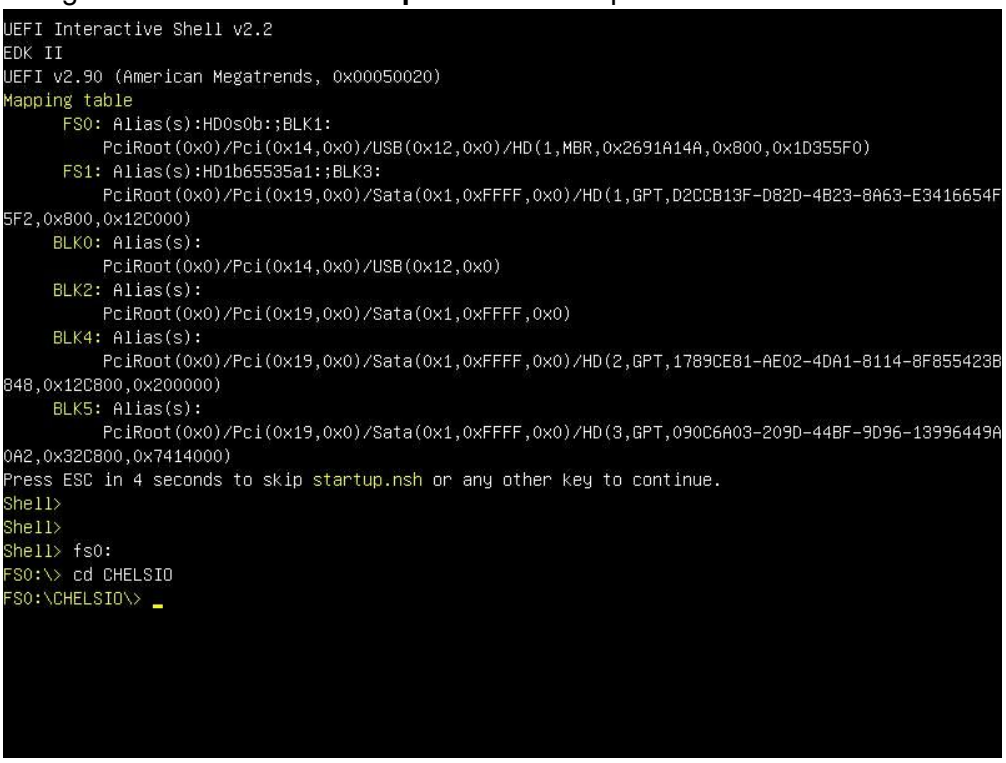
iv. Boot to **EFI Shell**.



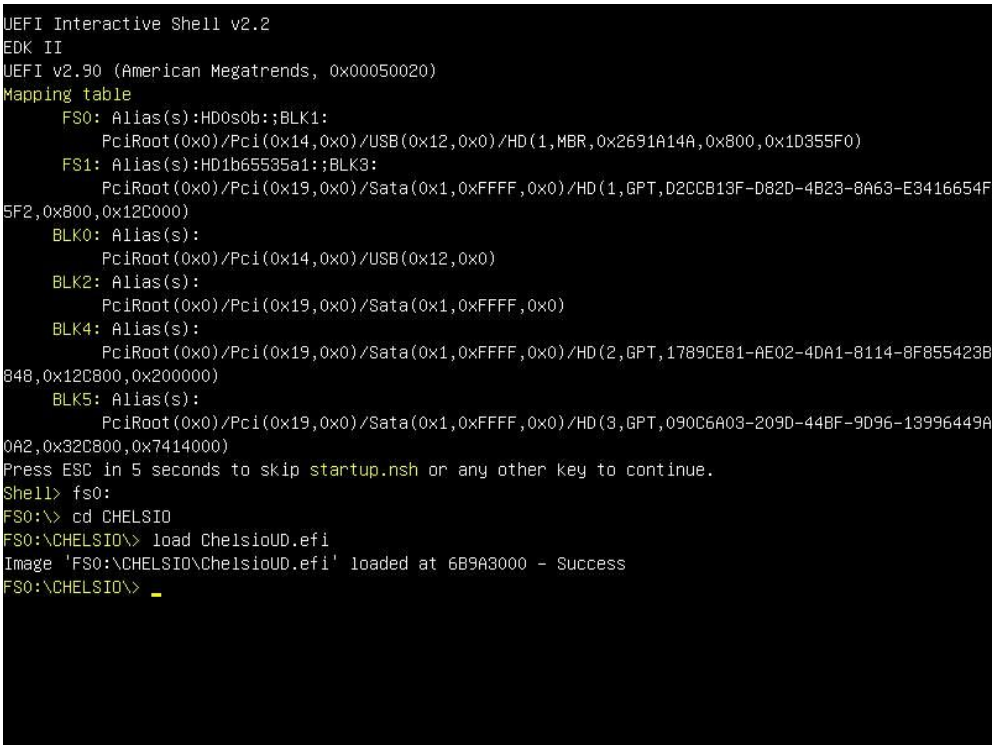
v. Enter into pendrive which is **fs0** in this case.



vi. Navigate to the folder where **OptionROM** is copied.



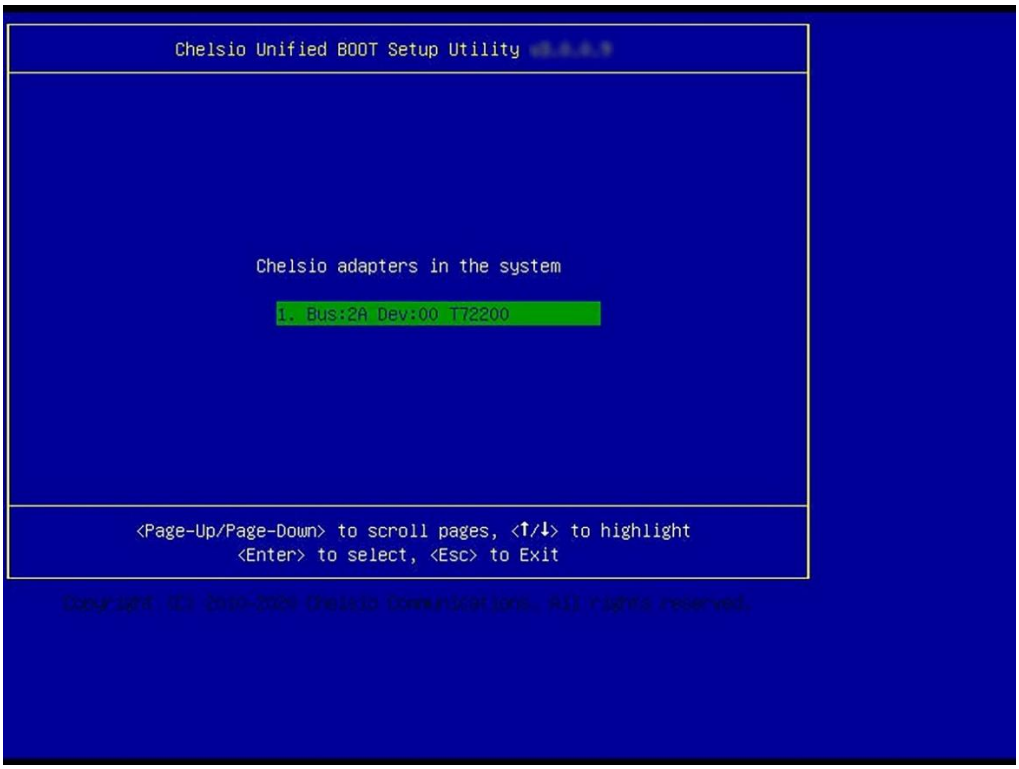
vii. Load the **ChelsioUD.efi** driver.



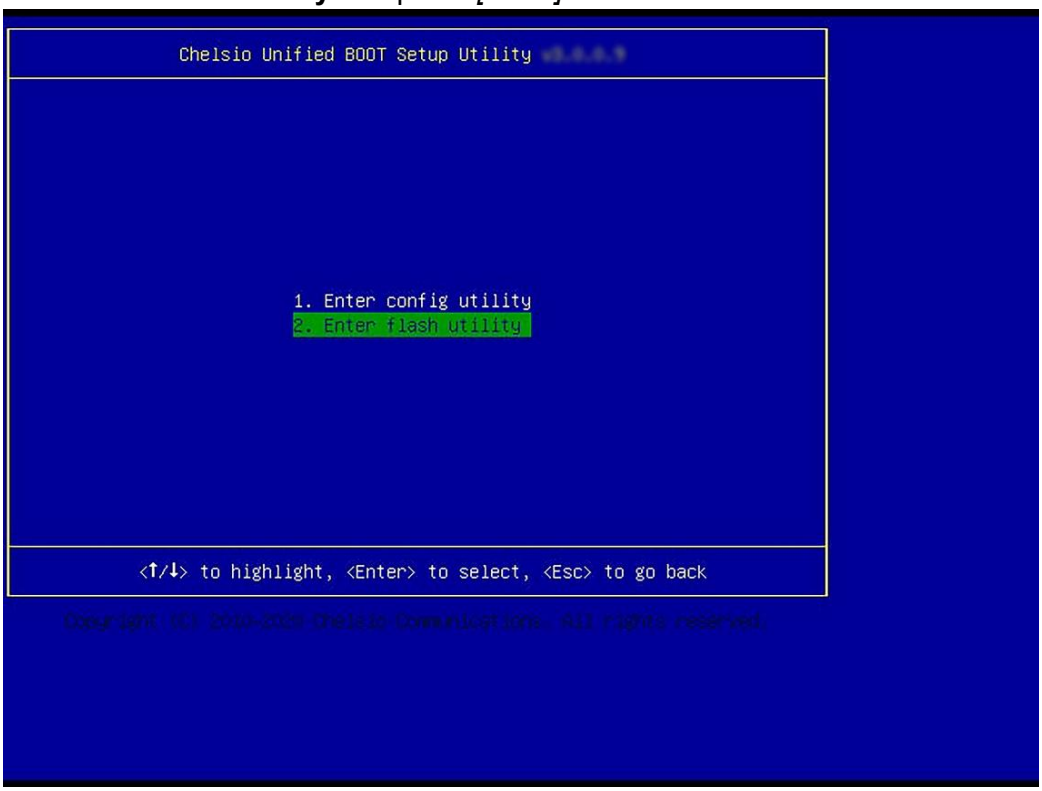
viii. Invoke the **Driver config** utility using ***drvcfg -s*** command.

```
UEFI Interactive Shell v2.2
EDK II
UEFI v2.90 (American Megatrends, 0x00050020)
Mapping table
      FS0: Alias(s):HD0s0b:;BLK1:
          PciRoot(0x0)/Pci(0x14,0x0)/USB(0x12,0x0)/HD(1,MBR,0x2691A14A,0x800,0x1D355F0)
      FS1: Alias(s):HD1b65535a1:;BLK3:
          PciRoot(0x0)/Pci(0x19,0x0)/Sata(0x1,0xFFFF,0x0)/HD(1,GPT,D2CCB13F-D82D-4B23-8A63-E3416654F
5F2,0x800,0x12C000)
     BLK0: Alias(s):
          PciRoot(0x0)/Pci(0x14,0x0)/USB(0x12,0x0)
     BLK2: Alias(s):
          PciRoot(0x0)/Pci(0x19,0x0)/Sata(0x1,0xFFFF,0x0)
     BLK4: Alias(s):
          PciRoot(0x0)/Pci(0x19,0x0)/Sata(0x1,0xFFFF,0x0)/HD(2,GPT,1789CE81-AE02-4DA1-8114-8F855423B
848,0x12C800,0x200000)
     BLK5: Alias(s):
          PciRoot(0x0)/Pci(0x19,0x0)/Sata(0x1,0xFFFF,0x0)/HD(3,GPT,090C6A03-209D-44BF-9D96-13996449A
0A2,0x32C800,0x7414000)
Press ESC in 5 seconds to skip startup.nsh or any other key to continue.
Shell> fs0:
FS0:\> cd CHELSIO
FS0:\CHELSIO\> load ChelsioUD.efi
Image 'FS0:\CHELSIO\ChelsioUD.efi' loaded at 6B9A3000 - Success
FS0:\CHELSIO\> drvcfg -s
```

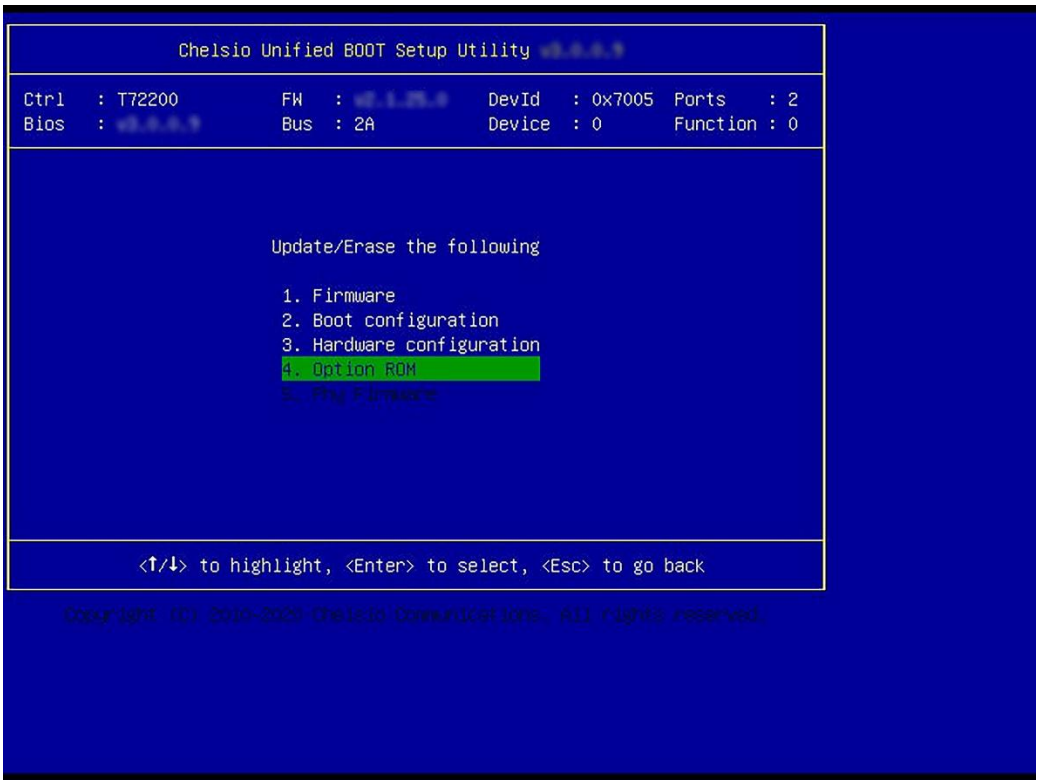
ix. The following configuration utility window appears. Select **Chelsio T7 Adapter** and press *[Enter]*.



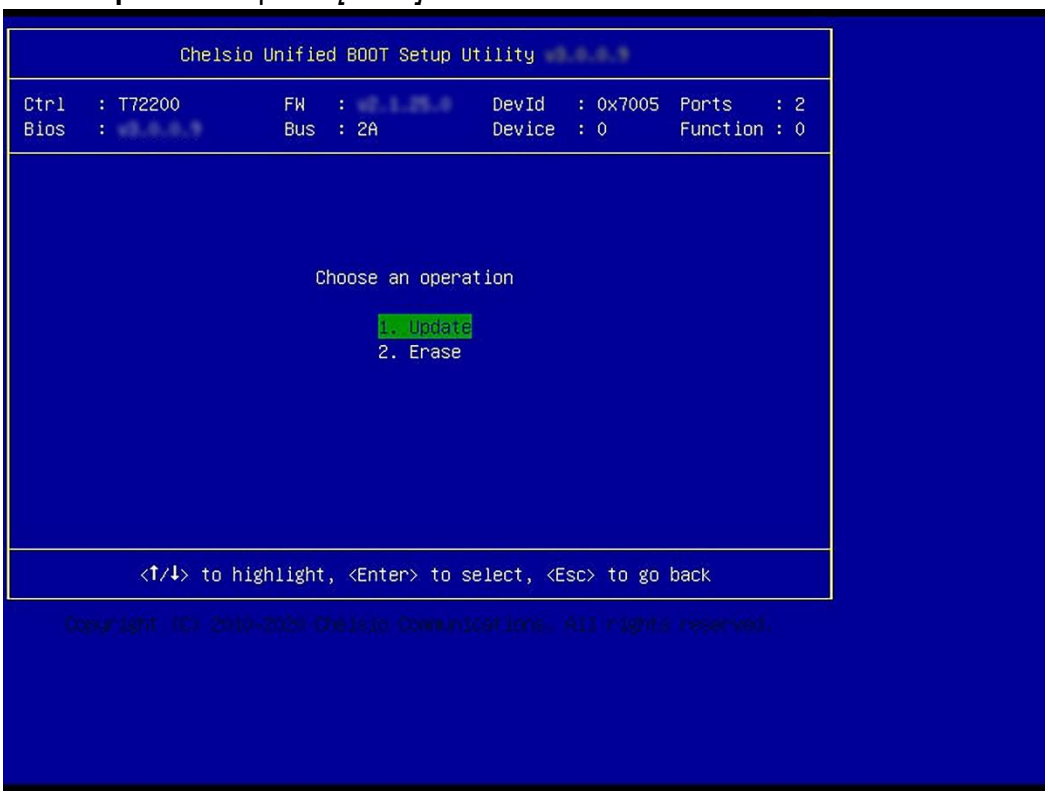
x. Select **Enter flash utility** and press *[Enter]*.



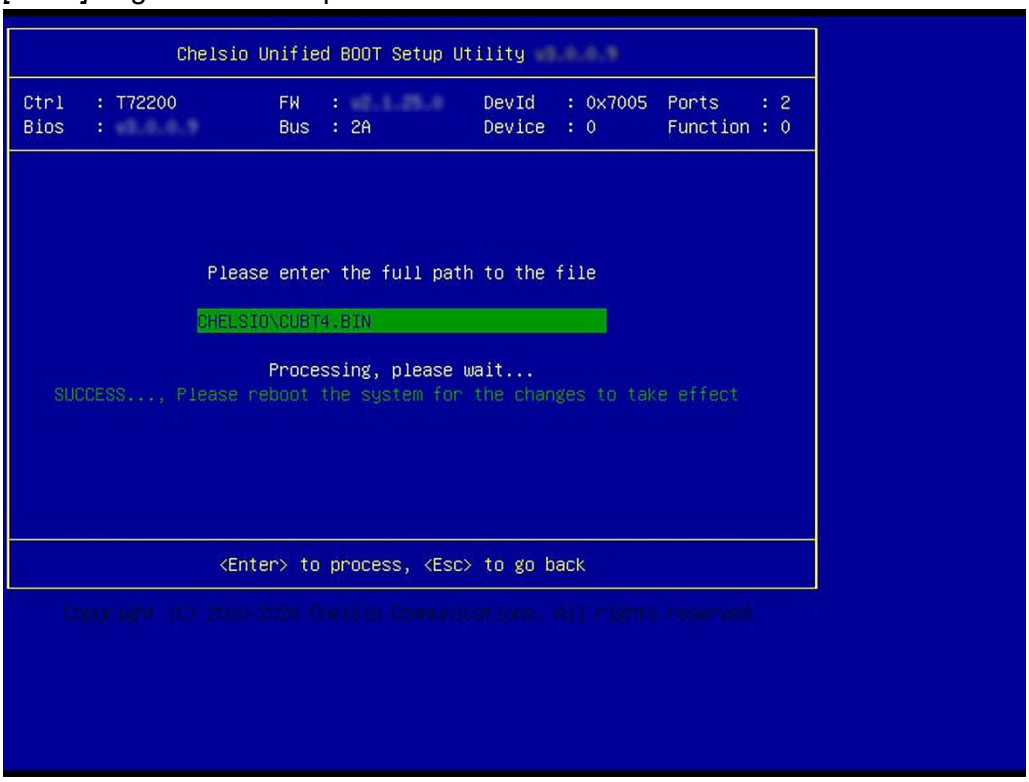
xi. Select **OptionROM** and press *[Enter]*.



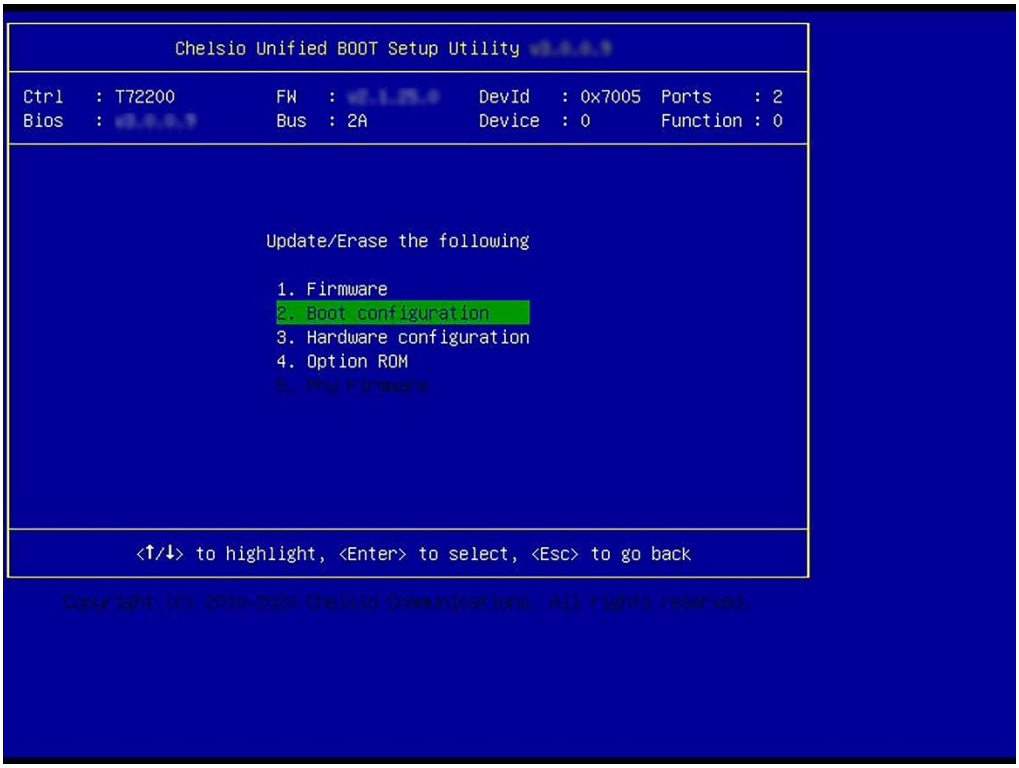
xii. Select **Update** and press *[Enter]*.



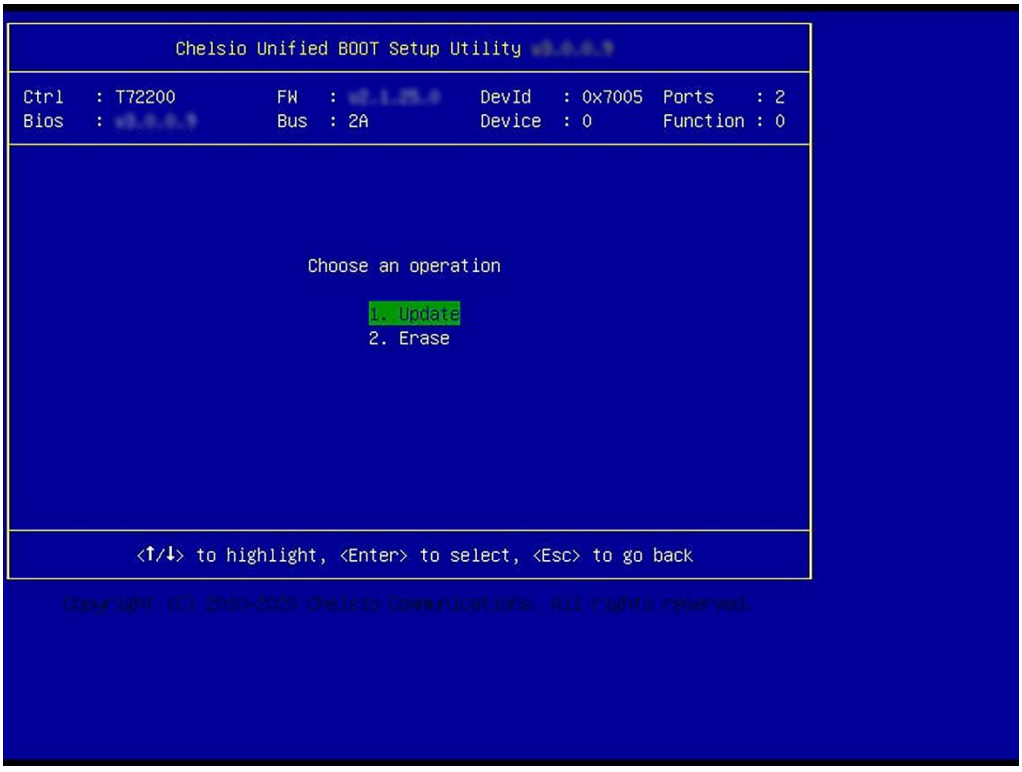
xiii. Provide the path to OptionROM and press *[Enter]*. Once OptionROM flash completes, press *[Enter]* to go back to the previous menu.



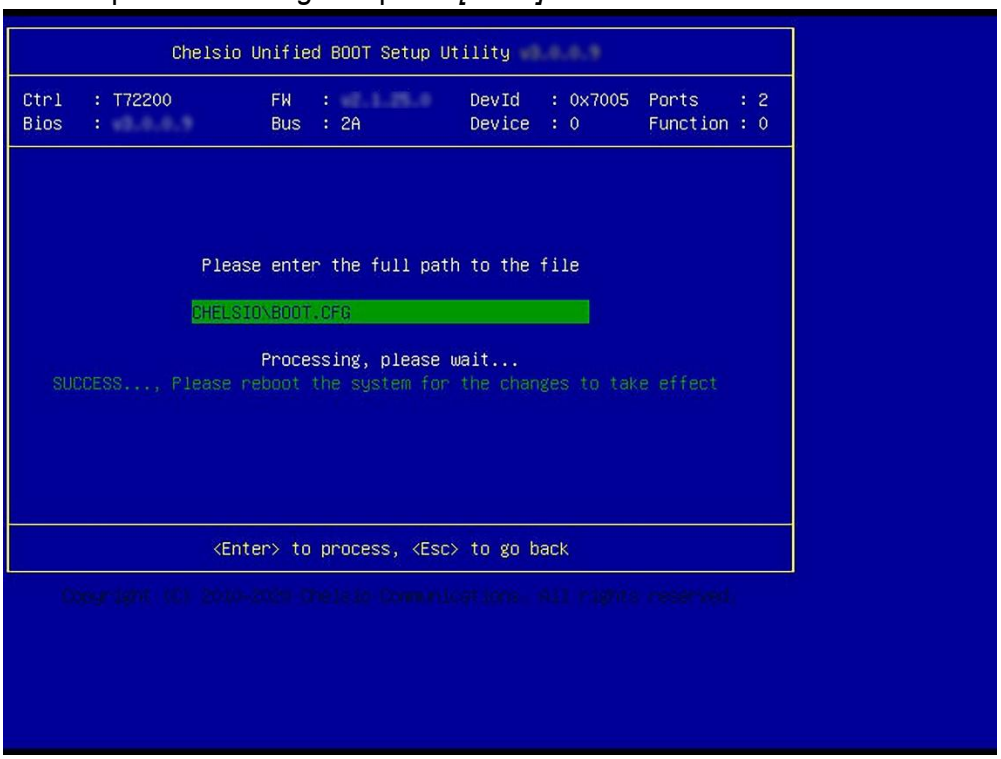
xiv. Select **Boot Configuration** and press *[Enter].*



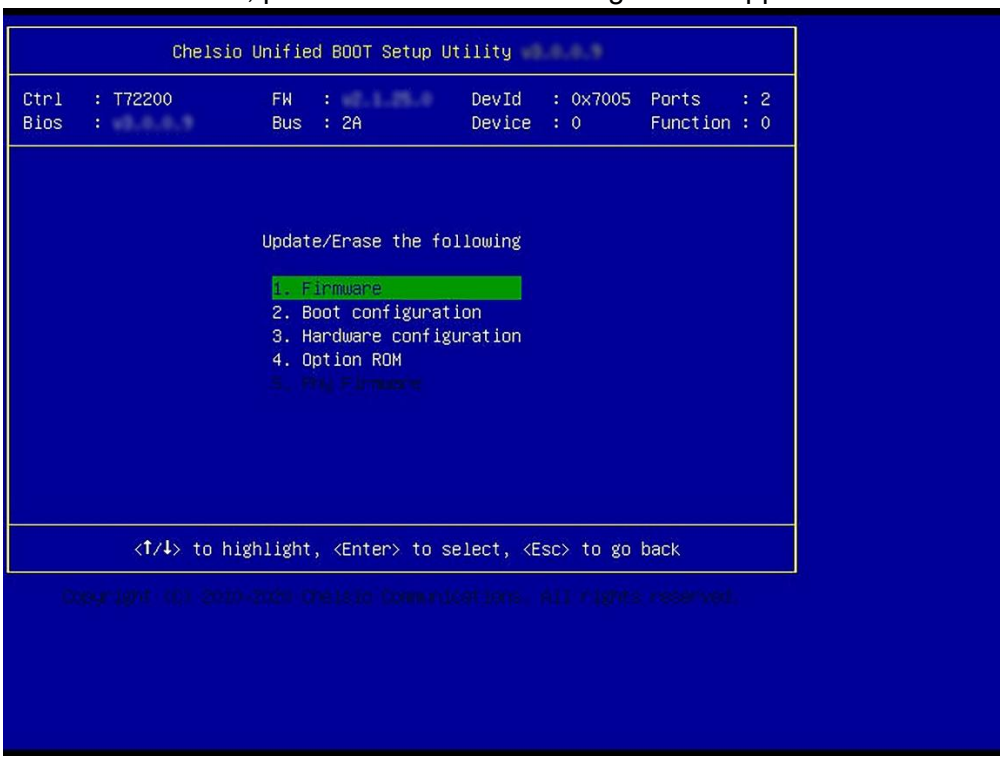
xv. Select **Update** and press *[Enter].*



xvi. Provide path to *boot.cfg* and press *[Enter].*

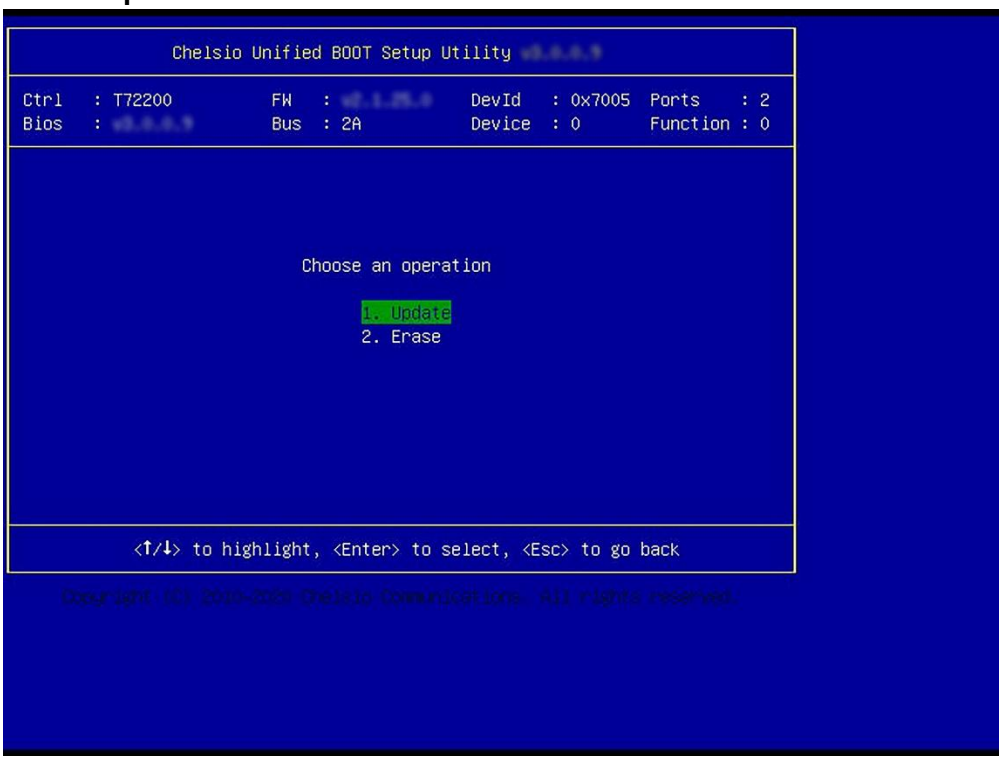


xvii. To flash Firmware, press *Esc* until the following screen appears.

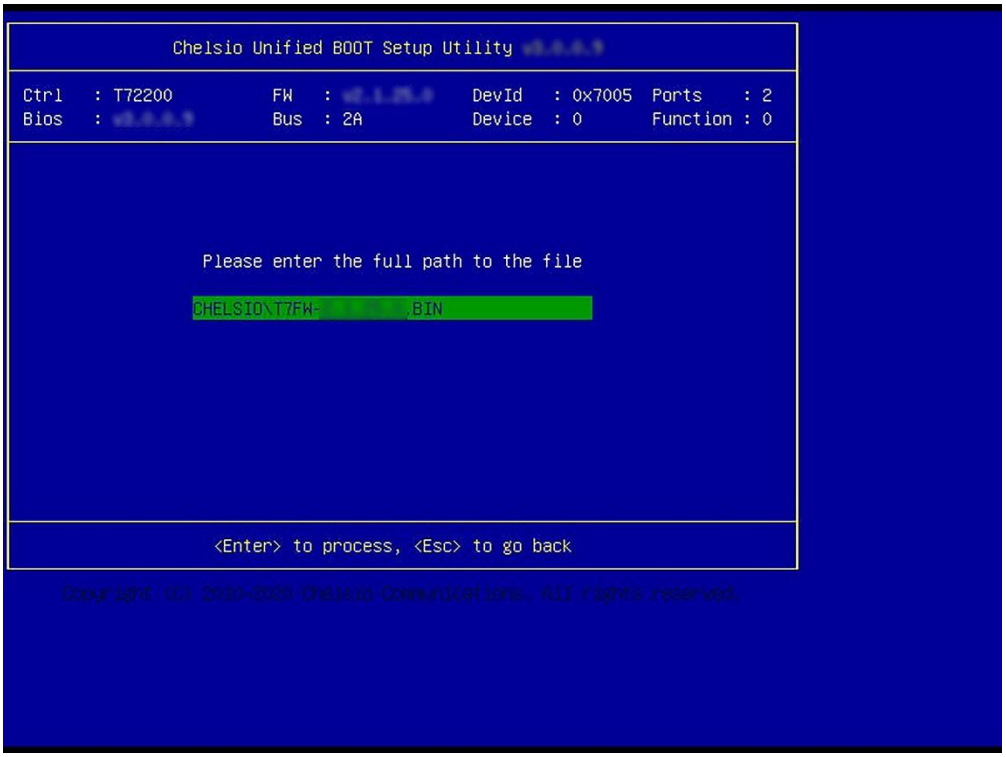


Note *If Chelsio T7 is not listed, please ensure that Chelsio uEFI driver is loaded correctly as mentioned here in the* ***Flashing Firmware and Option ROM*** *section.*

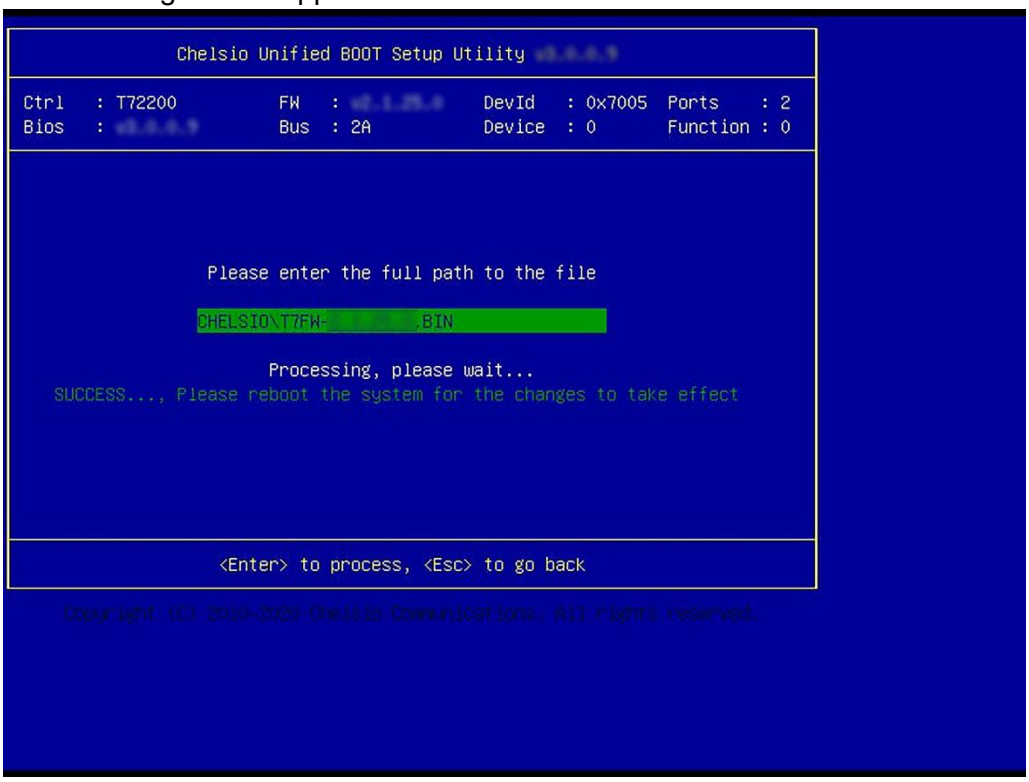
xviii. Select **Update.**



xix. Enter path to the firmware bin file.



xx. The following screen appears with successful firmware flash.



xxi. Reboot the system.

3.2. 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://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/automatically_installing_rhel/preparing-for-a-network-install_rhel-installer
- http://www.howtoforge.com/ubuntu_pxe_install_server

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

3.2.1. Building custom initrd

Perform the following steps to build initrd file:

i. Locate the **initrd.img** file corresponding to the operating system you intend to update. For the purpose of example, RHEL10.1 was used

```
[root@host ~]# mount -o loop,ro RHEL-10.1-20251021.0-x86_64-dvd1.iso /mnt
[root@host ~]# cd /mnt/
[root@host mnt]# ls
AppStream BaseOS EFI EULA extra_files.json GPL images isolinux
media.repo RPM-GPG-KEY-redhat-beta RPM-GPG-KEY-redhat-release
[root@host mnt]# find . | grep initrd.img
./images/pxeboot/initrd.img
./isolinux/initrd.img
```

ii. Create a folder to unzip the **initrd.img** and browse to the following folders:

```
[root@host ~]# mkdir initrd_workdir
```

iii. Unpack the image into the above created directory using the following command:

```
[root@host ~]# cd initrd_workdir
[root@host initrd_workdir]# xzcat /mnt/images/pxeboot/initrd.img | cpio -
idmv
```

Note *Use zcat instead of xzcat in case the image is gzip compressed.*

iv. The **cxgb4.ko** driver files for the required OSes are in the following directory inside the package.

Note This package only supports RHEL9.8/9.7/9.6, RHEL10.2/10.1/10.0, Rocky10.0 (example not mentioned).

```
[root@host ~]# ls -R ChelsioUwire-x.x.x.x/Uboot/LinuxDrivers/
ChelsioUwire-x.x.x.x/Uboot/LinuxDrivers/:
RHEL10.2 RHEL10.1 RHEL10.0 RHEL9.8 RHEL9.7 RHEL9.6
ChelsioUwire-x.x.x.x/Uboot/LinuxDrivers/RHEL10.2:
cxgb4.ko
ChelsioUwire-x.x.x.x/Uboot/LinuxDrivers/RHEL10.1:
cxgb4.ko
ChelsioUwire-x.x.x.x/Uboot/LinuxDrivers/RHEL10.0:
cxgb4.ko
ChelsioUwire-x.x.x.x/Uboot/LinuxDrivers/RHEL9.8:
cxgb4.ko
ChelsioUwire-x.x.x.x/Uboot/LinuxDrivers/RHEL9.7:
cxgb4.ko
ChelsioUwire-x.x.x.x/Uboot/LinuxDrivers/RHEL9.6:
cxgb4.ko
```

v. Replace the driver with your updated **T7 cxgb4.ko** driver. Make sure to replace the kernel version with the kernel version of the target OS:

```
[root@host initrd_workdir]# cp /<path_to_t7_cxgb4.ko_file>/cxgb4.ko
lib/modules/<kernel_version>/kernel/drivers/net/ethernet/chelsio/cxgb4/cxgb4
.ko
```

Example - For RHEL9.8 run the following command:

```
[root@host initrd_workdir]# cp /<path_to_package>/ChelsioUwire-
x.x.x.x/Uboot/LinuxDrivers/RHEL9.8/cxgb4.ko lib/modules/5.14.0-
687.5.3.el9_8.x86_64/kernel/drivers/net/ethernet/chelsio/cxgb4/
```

Example - For RHEL9.7 run the following command:

```
[root@host initrd_workdir]# cp /<path_to_package>/ChelsioUwire-
x.x.x.x/Uboot/LinuxDrivers/RHEL9.7/cxgb4.ko lib/modules/5.14.0-
611.5.1.el9_7.x86_64/kernel/drivers/net/ethernet/chelsio/cxgb4/
```

Example - For RHEL9.6 run the following command:

```
[root@host initrd_workdir]# cp /<path_to_package>/ChelsioUwire-
x.x.x.x/Uboot/LinuxDrivers/RHEL9.6/cxgb4.ko lib/modules/5.14.0-
570.12.1.el9_6.x86_64/kernel/drivers/net/ethernet/chelsio/cxgb4/
```

Example - For RHEL10.2 run the following command:

```
[root@host initrd_workdir]# cp /<path_to_package>/ChelsioUwire-
x.x.x.x/Uboot/LinuxDrivers/RHEL10.2/cxgb4.ko usr/lib/modules/6.12.0-
211.7.3.el10_2.x86_64/kernel/drivers/net/ethernet/chelsio/cxgb4/
```

Example - For RHEL10.1 run the following command:

```
[root@host initrd_workdir]# cp /<path_to_package>/ChelsioUwire-
x.x.x.x/Uboot/LinuxDrivers/RHEL10.1/cxgb4.ko usr/lib/modules/6.12.0-
124.8.1.el10_1.x86_64/kernel/drivers/net/ethernet/chelsio/cxgb4/
```

Example - For RHEL10.0 run the following command:

```
   [root@host initrd_workdir]# cp /<path_to_package>/ChelsioUwire-
   x.x.x.x/Uboot/LinuxDrivers/RHEL10.0/cxgb4.ko usr/lib/modules/6.12.0-
   55.9.1.el10_0.x86_64/kernel/drivers/net/ethernet/chelsio/cxgb4
```

Note *Make sure to replace the kernel version with the kernel version of the target OS.*

vi. Add **t7fw-2.1.x.x.bin** from the package to the firmware directory and create a symlink:

```
[root@host initrd_workdir]# cp /<path_to_package>/ChelsioUwire-
x.x.x.x/Uboot/OptionROM/t7fw-2.1.x.x.bin lib/firmware/cxgb4/
[root@host initrd_workdir]# ln -sf lib/firmware/cxgb4/t7fw-2.1.x.x.bin
lib/firmware/cxgb4/t7fw.bin
```

vii. Create a modprobe configuration file for iw_cxgb4 as below:

```
 [root@host initrd_workdir]# echo "install iw_cxgb4 /bin/true" | tee
etc/modprobe.d/iw_cxgb4.conf
```

viii. Run **depmod** to create the dependencies file for the updated driver:

```
[root@host~]# depmod -a -b . <kernel_version>
```

Example - For RHEL9.8 run the following command:

```
[root@host initrd_workdir]# depmod -a -b . 5.14.0-687.5.3.el9_8.x86_64
```

Example - For RHEL9.7 run the following command:

```
[root@host initrd_workdir]# depmod -a -b . 5.14.0-611.5.1.el9_7.x86_64
```

Example - For RHEL9.6 run the following command:

```
[root@host initrd_workdir]# depmod -a -b . 5.14.0-570.12.1.el9_6.x86_64
```

Example - For RHEL10.2 run the following command:

```
[root@host initrd_workdir]# depmod -a -b . 6.12.0-211.7.3.el10_2.x86_64
```

Example - For RHEL10.1 run the following command:

```
[root@host initrd_workdir]# depmod -a -b . 6.12.0-124.8.1.el10_1.x86_64
```

Example - For RHEL10.0 run the following command:

```
[root@host initrd_workdir]# depmod -a -b . 6.12.0-55.9.1.el10_0.x86_64
```

Note *Make sure to replace the kernel version with the kernel version of the target OS.*

ix. Re-pack the modified initrd image using the following command:

```
  [root@host initrd_workdir]# find . | cpio -o -H newc | gzip >
  /<path_to_new_initrd_with_t7_change>/t7_initrd.img
```

3.2.2. PXE Server Configuration

Use the previously generated custom initrd image to configure PXE Server.

Below is a sample configuration of grub.cfg for RHEL10.1:

```
menuentry 'Install RHEL 10.1' {
 linuxefi <tftp_or_http_path>/vmlinuz modprobe.blacklist=iw_cxgb4,cxgb4i
inst.repo=<IP of the
HTTP server> inst.stage2=<IP of HTTP server>
 initrdefi <tftp_or_http_path>/<generated_initrd_image_name>.img
}
```

Example:

```
menuentry 'Install RHEL 10.1 (verbose)' {
 linuxefi RHEL10.1/vmlinuz modprobe.blacklist=iw_cxgb4,cxgb4i
inst.repo=http://102.50.50.33/RHEL9.6/ inst.stage2=http://102.50.50.33/RHEL10.1/
initrdefi RHEL10.1/t7_initrd.img
}
```

3.3. PXE Boot Process

Before proceeding, ensure that the Chelsio T7 Adapter has been flashed with the provided firmware, OptionROM and boot.cfg (See Flashing Firmware and option ROM).

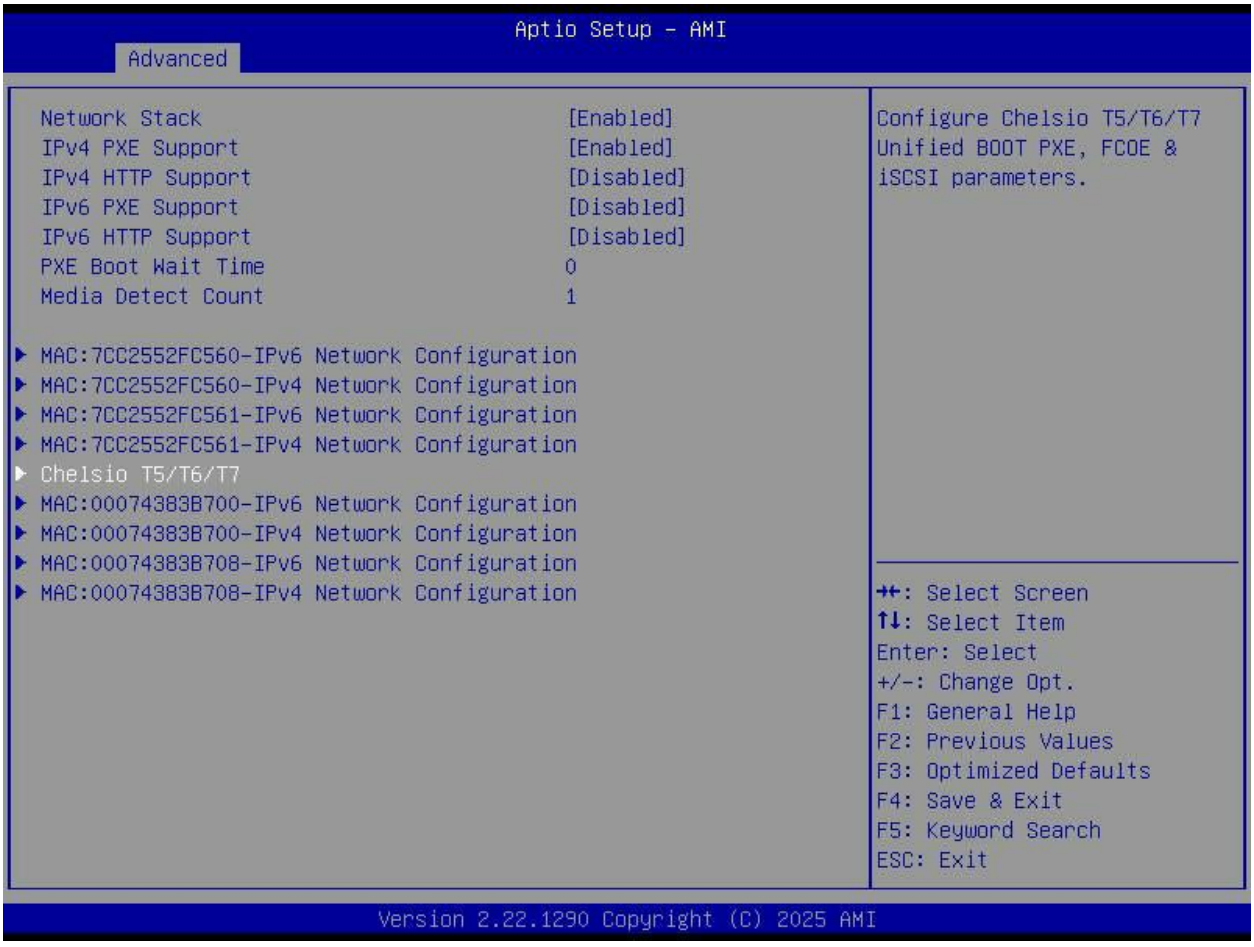
3.3.1. PXE Boot

Important

- *Only uEFI v2.3.1 and above are supported.*
- *Any other uEFI version is NOT SUPPORTED and may render your system unusable.*

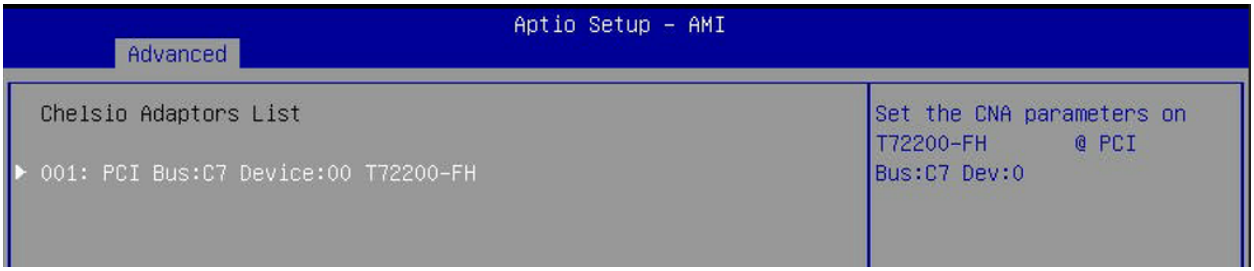
This section describes the method to configure and use Chelsio uEFI PXE interfaces.

i. Reboot the system and go into the BIOS setup.
ii. Disable Secure Boot, if not already done.
iii. Chelsio HII should be listed as **Chelsio T5/T6/T7**. Highlight it and press *[Enter]*.



Note *If Chelsio T5/T6/T7 is not listed, please ensure that Chelsio uEFI driver is loaded correctly as mentioned here in the **Flashing Firmware and Option ROM** section.*

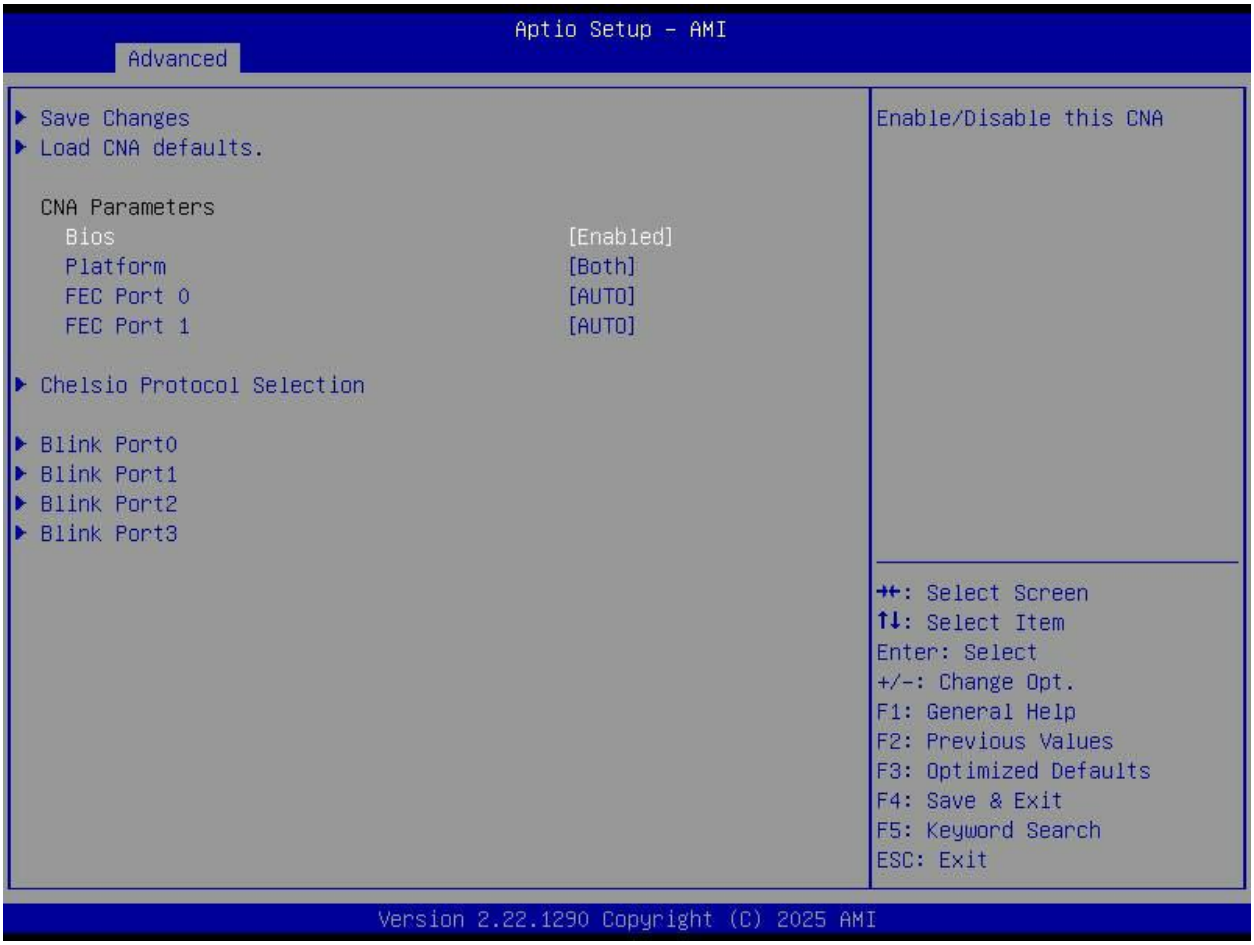
iv. Select the Chelsio adapter to be configured and press *[Enter]*.



v. Select **Configuration Utility** and press *[Enter]*.

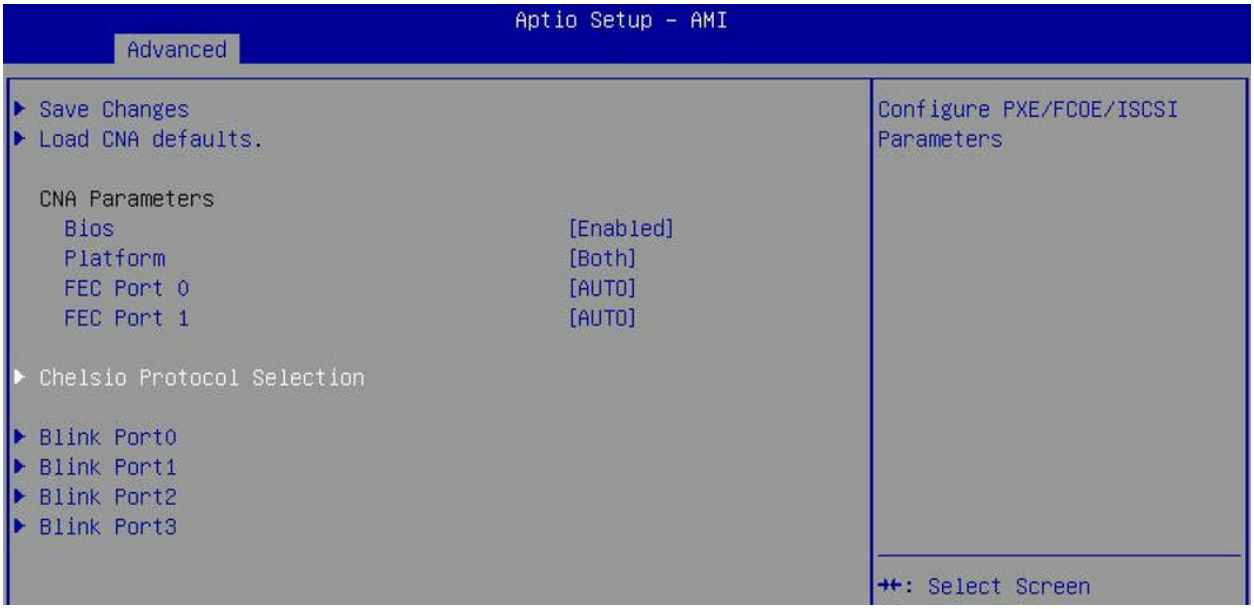


vi. Enable adapter BIOS if not already enabled.

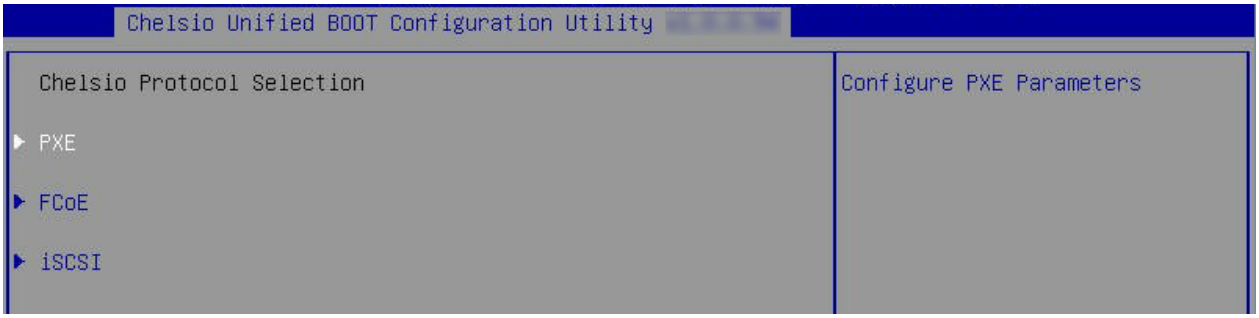


Note *It is highly recommended that you use the **Save Changes** option every time a parameter/option is changed.*

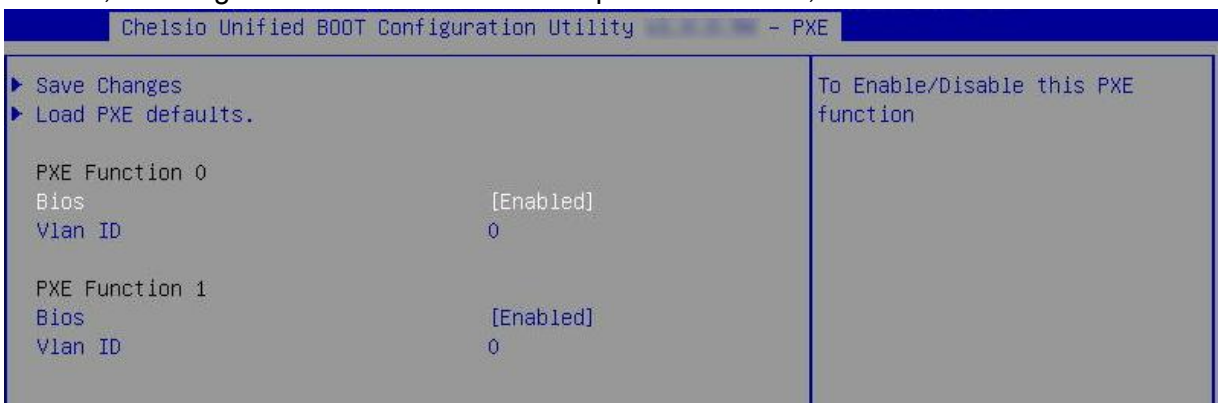
vii. Select **Chelsio Protocol Selection** and press *[Enter]*.



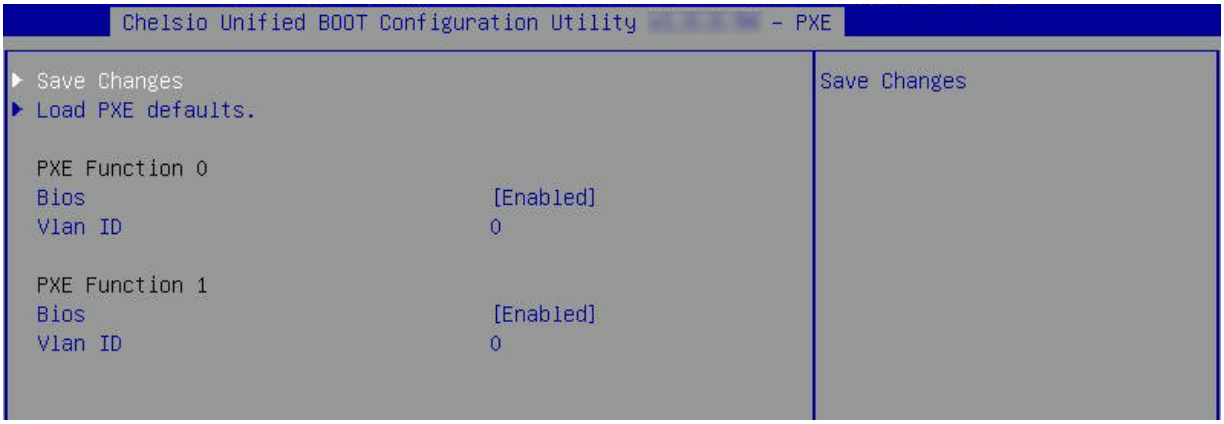
viii. Select **PXE** and press *[Enter]*.



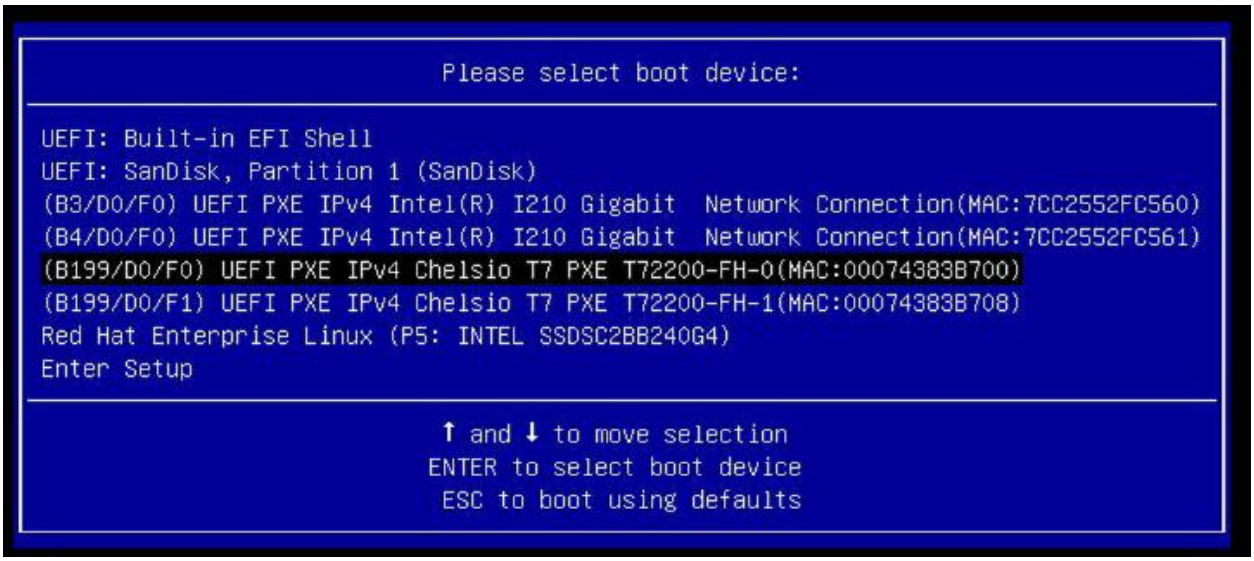
ix. Choose the boot port to try PXE boot. It is recommended to enable only those functions and ports which are going to be used. Note that enabling PXE Function 0 will enable port 0 for PXE, enabling PXE Function 1 will enable port 1 and so on, for NIC function.



x. Select **Save Changes** and press *[Enter]*.



xi. Reboot the system and press **F11** to go to boot manager, choose any of the available Chelsio PXE devices for PXE boot.

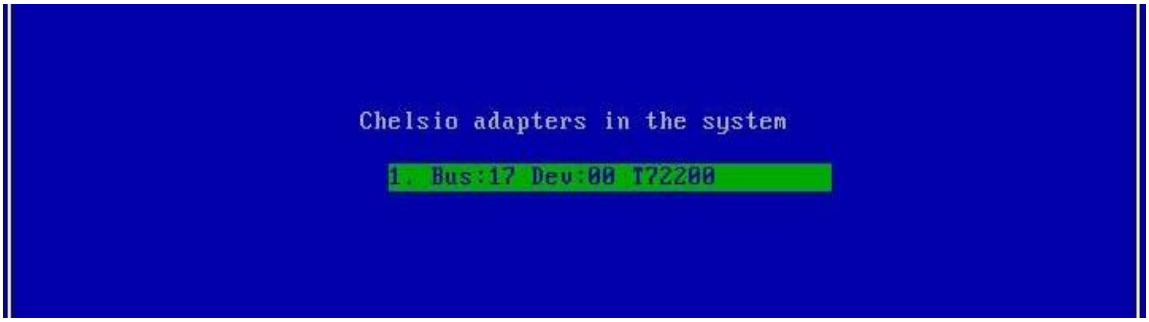


3.3.2. Legacy PXE Boot

1. After configuring the PXE server, make sure the PXE server works. Then reboot the client machine.
2. Press *[Alt+C]* when the banner to configure the Chelsio adapters appears on the screen.



3. The configuration utility will appear as below:



Choose the adapter on which you flashed the option ROM image and press *[Enter]*.

4. Enable the adapter BIOS using arrow keys if not already enabled And press *[Enter]*.

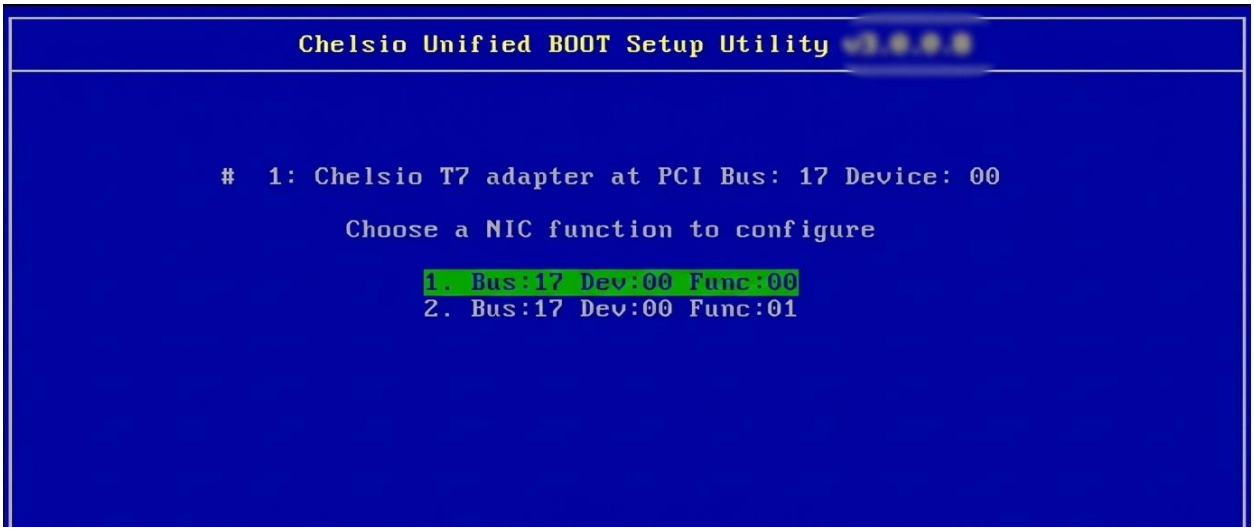


Note *Use the default values for Boot Mode, EDD and EBDA Relocation parameters, unless instructed otherwise.*

5. Choose *PXE* from the list to configure And press *[Enter]*.



6. Use the arrow keys to highlight the appropriate function among the supported NIC functions and press *[Enter]* to select.



7. Enable NIC function bios if not already enabled.

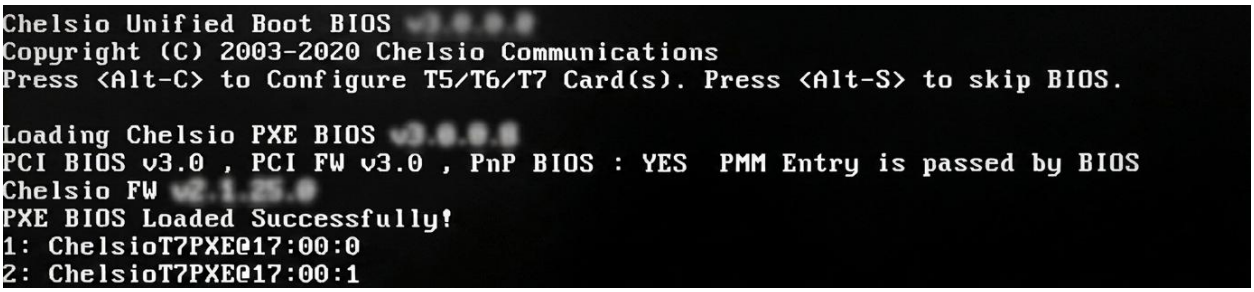


Choose the boot port to try the PXE boot. It is recommended only to enable functions and ports that are going to be used. Note that enabling NIC Func 00 will enable port 0 for PXE, enabling NIC Func 01 will enable port 1, and so on for the NIC function.

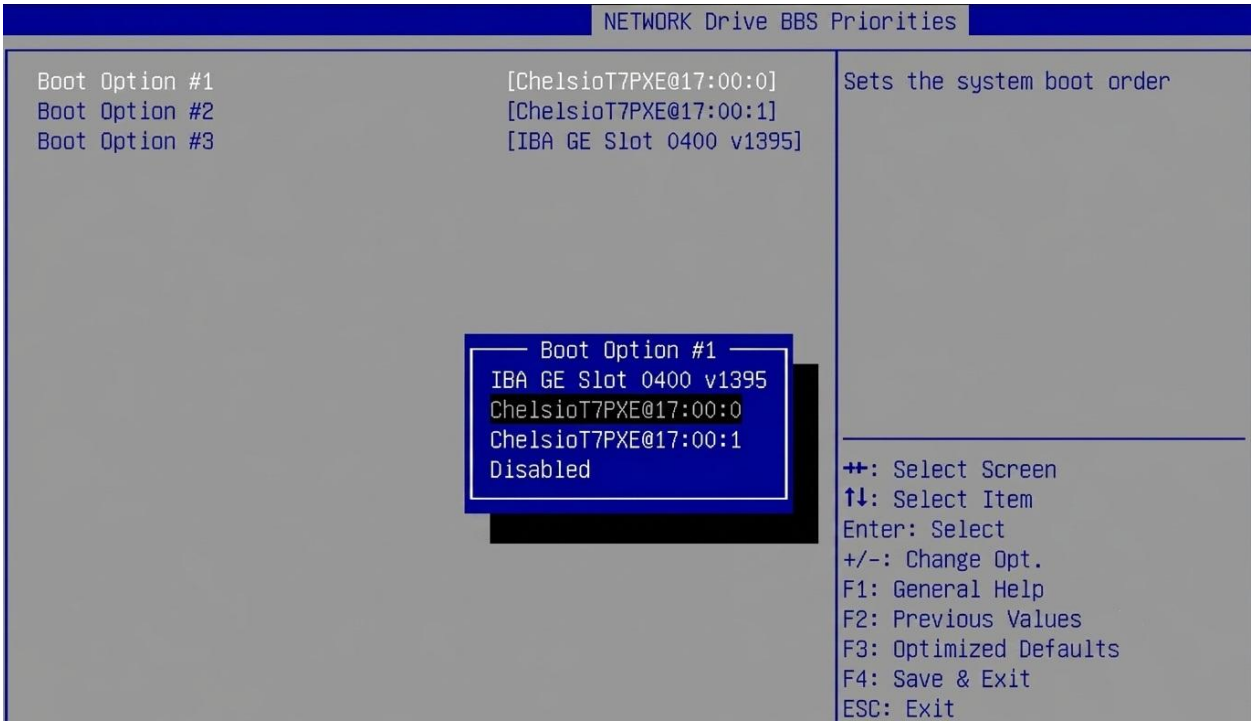
8. Press *[F10]* or *[Esc]* and then *[Y]* to save configuration changes.



9. Reboot the system.
10. Allow the Chelsio option ROM to initialize and setup PXE devices. DO NOT PRESS ALT-S to skip Chelsio option ROM.



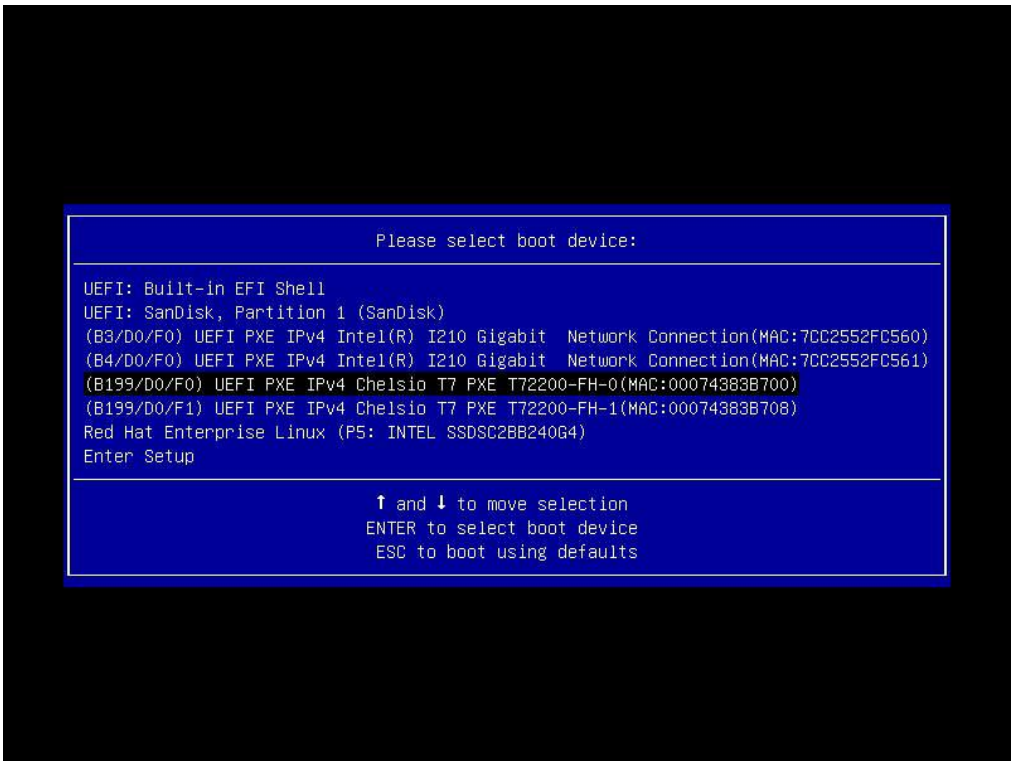
11. In the system setup, choose any of the Chelsio PXE devices as the first boot device.



12. Reboot. DO NOT PRESS ALT-S to skip Chelsio option ROM, during POST.
13. Press *[F12]* key when prompted to start PXE boot.

3.4. OS Installation

i. Reboot the host, press **F11** and select Chelsio Adapter T7 as the boot option.



ii. The following screen appears.

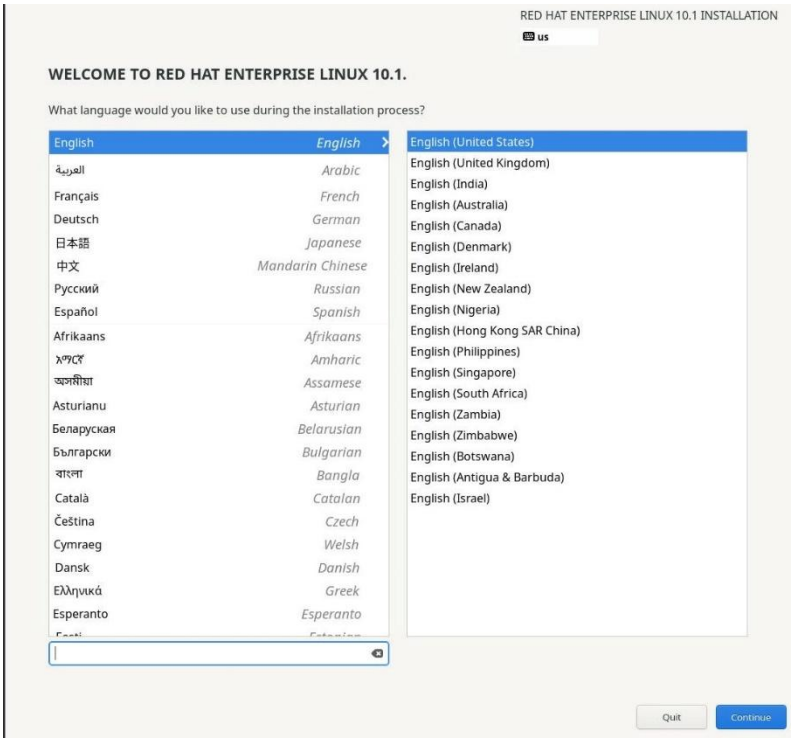


iii. In the following screen, PXE boot begins and an OS menu is listed, select an OS for installation.

Note *For the purpose of this document RHEL9.6 installation is shown.*



iv. Installation of OS begins. Select the installation language and click **Continue**.



v. Follow the onscreen instructions and proceed with the installation.

vi. After installation, reboot the host.

XXXVI. NC-SI (Network Controller Sideband Interface)

1. Introduction

The Chelsio T7 adapter supports the Network Controller Sideband Interface (NC-SI), a DMTF-defined sideband protocol described in the DSP0222 – Network Controller Sideband Interface (NC-SI) Specification. NC-SI allows a Baseboard Management Controller (BMC) to send and receive traffic on the external network through the adapter's ports. The T7 firmware implements version 1.1 of the specification, and the Get Version ID command returns 0xF1F1F000. The feature is implemented entirely in the T7 firmware and does not need any support from the host driver.

The link between the BMC and the adapter is limited to 100 Mbps it is independent of the Ethernet wire speed. Pass-through frames of up to 9000 bytes are supported. Hardware arbitration is not supported.

Each adapter presents a single NC-SI package. The number of NC-SI channels available for pass-through traffic between the BMC and the external network is the same as the number of ports on the adapter, and each channel number is the same as its port number.

Note *The default NC-SI package ID is 0. A vendor can assign a different package ID through the* STATIC_PKG_ID *field (column 3) of the serial init config file.*

Note *NC-SI is now supported only on port0.*

Note *NC-SI is provided as an Alpha quality release.*

1.1. Hardware Requirements

1.1.1. Supported Adapters

The following are the currently shipping T7 Chelsio adapters that are compatible with Chelsio NC-SI:

- T7 adapter:
  - T72200-FH
  - T72200
  - T7450
  - S71400
  - S72200
  - S7450
  - S7250

1.2. Software Requirements

1.2.1. Linux Requirements

Currently, the NC-SI driver is available only for the following kernel versions:

- Kernel.org linux-6.12.104
- Kernel.org linux-6.18.39
- Kernel.org linux-6.6.144
- RHEL/Rocky/AlmaLinux 10.2, 6.12.0-211.7.3.el10_2.x86_64
- RHEL/Rocky/AlmaLinux 10.1, 6.12.0-124.8.1.el10_1.x86_64
- RHEL/Rocky/AlmaLinux 10.0, 6.12.0-55.9.1.el10_0.x86_64
- RHEL/Rocky/AlmaLinux 9.8, 5.14.0-687.5.3.el9_8.x86_64
- RHEL/Rocky/AlmaLinux 9.7, 5.14.0-611.5.1.el9_7.x86_64
- RHEL/Rocky/AlmaLinux 9.6, 5.14.0-570.12.1.el9_6.x86_64
- Ubuntu 24.04.4, 6.8.0-136-generic

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

2. Firmware and init/vpd update

Note *The init/VPD files are only available in the source software package; while the 't7seeprom' utility used to flash init/VPD is available in both the source and RPM packages.*

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

2. Extract the downloaded package.

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

3. Change your current working directory to the Chelsio Unified Wire package directory and build and install the drivers, tools and libraries from source.

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

Note *If a previous version of driver package is installed, execute the following commands before attempting installation:*

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

4. Unload NIC driver and its dependencies.

```
[root@host~]# rmmod iw_cxgb4 cxgb4
```

5. Flash the updated init and vpd on the Chelsio T7 adapter.

```
[root@host~]# t7seeprom -flash -b <pci-id of chelsio card>  write -f:< path
to seeprom.bin file >
```

Example:

- For T72200-FH:

```
root@host~]# cd ChelsioUwire-x.x.x.x/support/config7
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t72200_fh_init_gen5x16_ncsi_2_mem_2ch_4800_2x200g_vpd_mfg.bin
```

- For T72200:

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config7
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t72200_init_gen5x16_ncsi_mem_2ch_4800_2x200g_vpd_mfg.bin
```

- For T7450:

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config7
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:t7450_init_gen5x16_ncsi_mem_2ch_4800_4x50g_vpd_mfg.bin
```

- For S72200:

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config7
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:s72200_init_gen5x16_ncsi_no_mem_2x200g_vpd_mfg.bin
```

- For S71400:

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config7
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:s71400_init_gen5x16_ncsi_no_mem_1x400g_vpd_mfg.bin
```

- For S7450:

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config7
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:s7450_init_gen5x16_ncsi_no_mem_4x40g_vpd_mfg.bin
```

- For S7250:

```
[root@host~]# cd ChelsioUwire-x.x.x.x/support/config7
[root@host~]# t7seeprom -flash -b 11:00.0 write -
f:s7250_init_gen5x16_ncsi_no_mem_4x40g_vpd_mfg.bin
```

6. Update the firmware on the Chelsio T7 adapter.
   Run the following commands to update the firmware manually:

```
[root@host~]# modprobe cxgb4 fw_attach=0
[root@host~]# cxgbtool ethX loadfw /lib/firmware/cxgb4/config/t7fwbootstrap-
x.x.x.x.bin
[root@host~]# cxgbtool ethX loadfw /lib/firmware/cxgb4/t7fw.bin
```

7. Power cycle your machine for changes to take effect.
8. Verify the firmware version (v2.1.x.x) after the host boots.
   The firmware (v2.1.x.x is installed on the system, typically in */lib/firmware/cxgb4*. The firmware version can be verified using:

```
[root@host~]# ethtool -i ethX
driver: cxgb4
...
firmware-version: 2.1.x.x,TP 0.1.30.0
```

3. Package and Channel Configuration

The Chelsio firmware implements the **DMTF DSP0222 NC-SI 1.1.0**. Each adapter presents one NC-SI package. The number of NC-SI channels for pass-through traffic between the BMC and the external network are equal to the number of ports on the adapter, and each NC-SI channel number is the same as its port number.

3.1. Pass-through Traffic

The BMC-NCSI interface runs at a maximum of 100 Mbps, independent of the Ethernet wire speed, and hardware arbitration is not supported.

4. Recommended BMC Bring-up Sequence

Perform the following steps to bring-up BMC:

1. Select package, then clear initial state.
2. Get the **Version ID**.
3. Set the MAC address.
4. Enable **Broadcast Filter** with the ARP bit set. Without this, the BMC does not receive broadcast ARP packets.
5. Set **VLAN Filter** and enable **VLAN** (optional).
6. Set link.
7. Enable **Channel**, then enable **Channel Network TX**.
8. Get **Link Status**.

4.1. AEN Configuration

When the host driver is loaded, the firmware sends a *Configuration Required Asynchronous Event Notification* (AEN) on every NC-SI channel, requesting that the NC-SI configuration be applied again.

Important

After receiving this AEN, the BMC must reissue its channel configuration commands, such as Set MAC Address and Enable Broadcast Filter, even if the channel was already configured. Without this, the BMC may lose connectivity once the host driver loads.

XXXVII. Appendix

1. Limitations

The following is the limitation in this release of Chelsio Unified Wire:

- Some iWARP UDP libraries will not compile unless the user has root privileges.

2. Troubleshooting

- ***Cannot bring up Chelsio interface***

Make sure you have created the corresponding network-script configuration file as stated in Chelsio Unified Wire chapter (See Creating Network Manager keyfiles manually). 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 NetworkManager keyfiles). 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, ensure that 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.

Then, 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, ensure that 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 IP address of 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. Refer to the appropriate documentation for the Linux distribution on how to configure or disable the firewall.

- ***Configuring Ethernet interfaces on the Cisco switch***

Always configure Ethernet interfaces on the Cisco switch in trunk mode.

- ***The Cisco nexus switch reporting "pauseRateLimitErrDisable"***

If 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 up to four Chelsio adapters in a machine. In case of using more than four adapters, it is recommended to disable the Option ROM on the adapters.

- ***Ubuntu OS upgrade***

With the Chelsio Unified Wire package installed, the Ubuntu OS upgrade might run into some Debian package conflicts, because of firmware files. To avoid this,

1. Uninstall the Chelsio Unified Wire Package.

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

2. Upgrade the OS.

```
[root@host~]# apt-get clean
[root@host~]# apt-get update && apt-get upgrade
```

3. Boot to the newly installed kernel/OS and install Chelsio Unified Wire.

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

- ***Installer issues***

In case of any failures while running the Chelsio Unified Wire Installer, collect the following:

- install.log file, if installed using install.py
- Entire make command output, if installed using the makefile

- ***Logs collection***

In case of any driver/firmware issues, run the following 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, the following files need to be provided for analysis:

```
vmcore, vmcore-dmesg.txt, vmlinux, System.map-$(uname -r), Chelsio modules
.ko files
```

- ***Software RPM Package***

If the end-user decides to install RPMs without using the interactive *./install.py* script included in the RPM packages, they must run the following command:

```
[root@localhost~]# rpm -ivh libcxgb4-4.x.x.x-0.x86_64.rpm --nodeps --force
```

3. 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. The license granted hereunder does not constitute a sale of the Software or any copy thereof. 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, except to the extent that such activity is expressly permitted in Section 5 (Authorized Distribution) by applicable law notwithstanding this limitation. You assume full responsibility for the use of the Software and agree to use the Software legally and responsibly. You may not distribute, lease, or lend the Software to third parties except as expressly provided in a separate mutual agreement or under the authorized distribution terms defined in this Agreement.

3. Ownership of Software. As Licensee, you own only the media upon which the Software is recorded or fixed (if applicable), but Chelsio retains all right, title, and interest in and to the Software and all copies of the Software, whether original or subsequent, and regardless of who possesses them. This ownership remains with Chelsio regardless of the form or media in or on which the Software may be embedded.

4. Confidentiality. You agree to maintain the non-public aspects of the Software (including source code and internal logic) in confidence. You shall not disclose the Software to any third party without the express written consent of Chelsio, except where such disclosure is a necessary part of a distribution authorized by Chelsio via mutual agreement. You further agree to take all reasonable precautions to limit access of the Software to those of your employees who reasonably require such access to perform their employment obligations and who arebound by confidentiality agreements with you. In the event of an authorized distribution, you agree to pass on these confidentiality obligations to the recipient.

5. Authorized Distribution. You are granted a limited, non-exclusive right to distribute the Software to your end-users solely for the purpose of operating [Specific Hardware/System Name] products, provided that:

(a) Flow-Down: You ensure that your end-users are subject to a license agreement with terms no less protective of Chelsio’s rights (including Confidentiality and Reverse Engineering restrictions) than those contained herein.

(b) Indemnification: You agree to indemnify and hold Chelsio harmless from any claims arising out of your distribution or your end-users' use of the Software.

6. Term. This license is effective in perpetuity, unless terminated earlier. You may terminate the license at anytime 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.

7. 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.

8. 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.

9. 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.

10. 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.

11. 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.

12. 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.

13. 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