

QUIC TX Offload on Linux: Chelsio T7 vs. Broadcom for High-Throughput HTTP/3

Executive Summary

HTTP/3 now carries a growing share of web, streaming and content-delivery traffic, and QUIC is the transport underneath it. That success comes with a cost on the server. QUIC runs in user space over UDP and encrypts every packet it sends, so the server CPU spends its cycles on packet protection and segmentation instead of serving content. At 100 Gbps and above, that overhead grows fast enough to cap how much traffic a single server can deliver.

Chelsio's T7 and S7 ASICs power a family of PCIe Gen5 adapters, including DPUs, SmartNICs and storage adapters, at speeds of up to 400GbE. On supported adapters from either family, QUIC TX offload takes that work off the host: transmit-side QUIC processing runs in adapter hardware, which gives the CPU cores back to the application. This paper compares a Chelsio T72200-FH adapter (T7 ASIC) with QUIC TX offload against a Broadcom BCM57608 adapter running QUIC in host software with UDP GSO on Linux, using an nginx HTTP/3 server and the h2load load generator.

Key results: At 128 concurrent connections, an nginx HTTP/3 server on the Chelsio T7 with QUIC TX offload delivers **183 Gbps** at MTU 9000 and **164 Gbps** at MTU 1500, against **167 Gbps** and **162 Gbps** on the Broadcom BCM57608 with UDP GSO and no QUIC offload. At MTU 9000, Chelsio delivers about 10% more throughput while cutting server CPU utilization from **44% to 26%**. At MTU 1500, Chelsio delivers slightly higher throughput (164 vs. 162 Gbps) and uses **28% CPU instead of 50%**. Across all six test points, Chelsio uses **35% to 51% less** server CPU.

Introduction

QUIC is an encrypted, multiplexed transport built on UDP. It builds TLS 1.3 security into the protocol, sets up connections faster than TCP with TLS, and keeps streams independent so that a lost packet on one does not stall the others. Those benefits have a price. Because QUIC runs in user space and protects every packet, a software implementation consumes a substantial amount of CPU at high data rates.

Chelsio's Terminator 7 (T7) ASIC is a seventh-generation Unified Wire Data Processing Unit that supports 1/10/25/40/50/100/200/400 Gbps. It already offloads a broad set of protocols, including crypto (IPsec, TLS/SSL), TCP, UDP, NVMe/TCP, NVMe-oF, iSCSI and RDMA (iWARP and RoCEv2). QUIC TX offload adds QUIC to that list. The T7 handles transmit-path work in hardware, which frees host cores for the application, and applications such as nginx can keep serving HTTP/3 while it does so.

The sections that follow compare the Chelsio T72200-FH with QUIC TX offload against the Broadcom BCM57608 with UDP GSO and no QUIC offload, present the test results, describe the test setup and the tuning applied to the server, and close with conclusions.

Test Results

Figures 1 and 2 compare transmit throughput and server CPU utilization of a Chelsio T72200-FH adapter with QUIC TX offload and a Broadcom BCM57608 adapter with UDP GSO and no QUIC offload at 32, 64 and 128 concurrent connections, at MTU 9000 (Figure 1) and MTU 1500 (Figure 2).

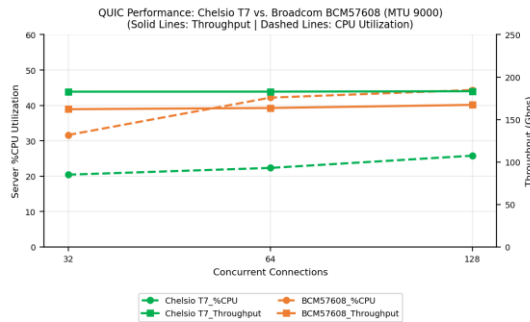


Figure 1 – QUIC Performance and CPU Utilization, Chelsio T7 vs. Broadcom BCM57608, at MTU 9000

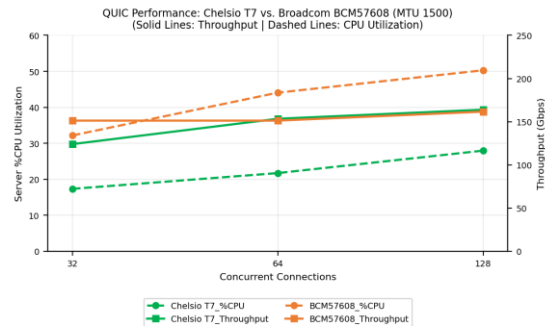


Figure 2 – QUIC Performance and CPU Utilization, Chelsio T7 vs. Broadcom BCM57608, at MTU 1500

Up to 51% Less Server CPU, Higher Throughput on Jumbo Frames, and Equal or Higher Throughput on Standard Frames as Connections Scale.

At MTU 9000 (jumbo frames)

- **Higher throughput, less CPU:** Chelsio sustains about **183 Gbps**, 10% to 13% above Broadcom (162 to 167 Gbps), while using up to **47% less** server CPU.
- **Lower server CPU at every load:**
 - 32 connections: 20% CPU with Chelsio vs. 32% with Broadcom (35% lower)
 - 64 connections: 22% CPU with Chelsio vs. 42% with Broadcom (47% lower)
 - 128 connections: 26% CPU with Chelsio vs. 44% with Broadcom (42% lower)

At MTU 1500 (standard frames)

- **Equal or higher throughput from 64 connections onward:** Chelsio delivers **124 to 164 Gbps** and passes Broadcom (151 to 162 Gbps) from 64 connections, at roughly half the CPU.
 - 32 connections: 124 Gbps at 17% CPU vs. 151 Gbps at 32% CPU for Broadcom
 - 64 connections: 153 Gbps with Chelsio vs. 151 Gbps with Broadcom (1% higher)
 - 128 connections: 164 Gbps with Chelsio vs. 162 Gbps with Broadcom (1% higher)
- **Lower server CPU at every load:**
 - 32 connections: 17% CPU with Chelsio vs. 32% with Broadcom (46% lower)
 - 64 connections: 22% CPU with Chelsio vs. 44% with Broadcom (51% lower)
 - 128 connections: 28% CPU with Chelsio vs. 50% with Broadcom (44% lower)
- **Scales with connection count:** At MTU 1500, Chelsio throughput rises 32% from 32 to 128 connections (124 to 164 Gbps), while Broadcom rises 7% (151 to 162 Gbps).
- **Higher CPU efficiency:** At MTU 1500, Chelsio delivers 1.5x to 2.1x the throughput per percentage point of CPU utilization.

Across both MTU sizes

- **Lower server CPU in every test:** Reductions range from 35% to 51%, leaving more CPU for application workloads.
- **Equal or higher throughput in five of six tests, with lower CPU in all six:** At 32 connections and MTU 1500, Chelsio delivers 124 Gbps at 17% CPU, and its throughput passes Broadcom as connections increase.
- **Higher CPU efficiency:** Chelsio delivers 1.5x to 2.1x the throughput per percentage point of CPU utilization across all six test points.
- **Strong performance on standard frames:** At 128 connections, Chelsio delivers 164 Gbps on 1500-byte frames, about 89% of its jumbo-frame throughput and more than Broadcom's 162 Gbps, at 28% CPU instead of 50%.

Test Setup

Topology

The server and the client each connect to a switch over a single 200 GbE link, using one port on each adapter; the second port is unused. Tests were run at MTU 9000 (jumbo frames) and MTU 1500 (standard frames), with the MTU set on both hosts and the switch. The server was fitted with the Chelsio T72200-FH for the Chelsio runs, with QUIC TX offload enabled, and with the Broadcom BCM57608 for the Broadcom runs, which processed QUIC in host software with UDP generic segmentation offload (UDP GSO) enabled, as shown in Figure 3.

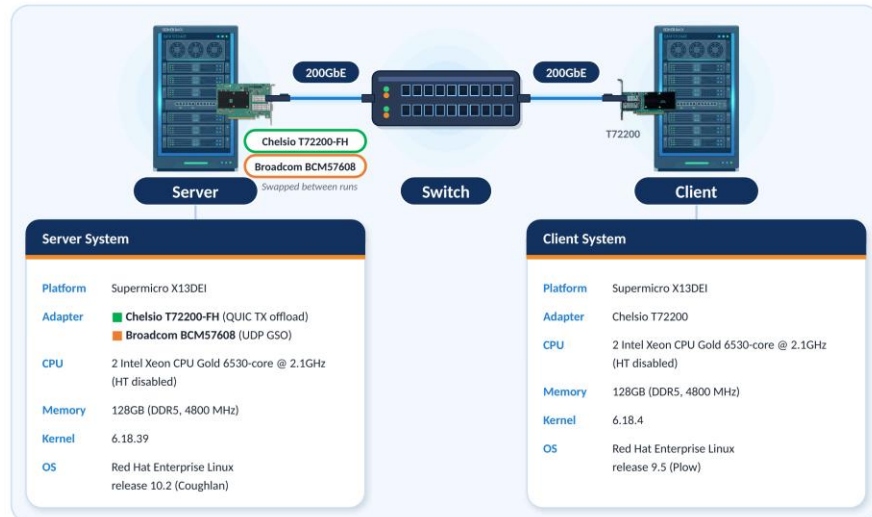


Figure 3 – Test Setup

Setup Configuration

Common Configuration

The following BIOS settings and Kernel parameters were configured on both the Server and the Client systems.

BIOS settings:

- Disable Hyper-Threading Technology.
- Disable Sub-NUMA Clustering (SNC).
- Disable CPU Power Management Features including C-states, P-states, and Turbo Boost.
- Configure System power management profile in Maximum Performance Mode.

Kernel parameters:

- Add 'intel_idle.max_cstate=0 processor.max_cstate=0 intel_pstate=disable selinux=0' to the kernel command line and reboot the machine.

Server Configuration (Chelsio T7)

Perform the following steps on the Server machine:

- Install RHEL 10.2 and compile 6.18.39 kernel.
- Disable Indirect Branch Tracking (IBT) and reboot.

```
[root@host ~]# grubby --update-kernel=ALL --args="ibt=off"
```

```
[root@host ~]# reboot
```
- Compile and install the latest [Chelsio Unified Wire package](#) and reboot the machine.

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

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

```
[root@host ~]# reboot
```
- Set the TunedD profile:

```
[root@host ~]# tuned-adm profile network-throughput
```
- Initializing the driver and interface.
 Load the required Chelsio driver modules, bring up the interface, and pin NIC interrupts to CPU cores using the performance tuning script:

```
[root@host ~]# modprobe -v cxgb4 max_eth_qsets=64 enable_ulds=0x1
```

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

```
[root@host ~]# ifconfig ens3f4 101.1.1.235/24 up
```

```
[root@host ~]# t4_perftune.sh -n -s
```
- Configuring QUIC and nginx Build.
 Clone and build QuicTLS (the OpenSSL fork with QUIC APIs), generate the server SSL certificate, and compile nginx with HTTP/3 and Chelsio QUIC offload support:

```
[root@host ~]# BASE_DIR="/root/OPENSSL"
```

```
[root@host ~]# mkdir -p $BASE_DIR
```

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

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

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

```
[root@host ~]# ./config -O3 -march=native -flto shared \
```

```
--prefix=$BASE_DIR/openssl --openssldir=$BASE_DIR/openssl \
```

```
-Wl,-rpath,$BASE_DIR/openssl/lib64
```

```
[root@host ~]# make -j${nproc} && make install_sw install_ssldirs
```

```
[root@host ~]# cd $BASE_DIR/openssl/bin
```

```
[root@host ~]# ./openssl req -nodes -new -x509 -keyout /root/server.key -out
```

```
/root/server.crt -days 365 -subj "/CN=$(hostname -f)"
```

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

```
[root@host ~]# git clone https://github.com/chelsiocrypto-26/nginx_quic.git
```

```
[root@host ~]# cd nginx_quic
[root@host ~]# ./auto/configure --prefix=$BASE_DIR/nginx --with-http_v3_module
--with-cc-opt="-O3 -march=native -flto -I$BASE_DIR/openssl/include" --with-ld-
opt="-flto -L$BASE_DIR/openssl/lib64 -L/usr/local/lib64 -Wl,-
rpath,$BASE_DIR/openssl/lib64 -Wl,-rpath,/usr/local/lib64 -lchquic"
[root@host ~]# make -j${nproc} && make install
```

vii. Configuring nginx.

Configure the nginx server parameters for HTTP/3 and enable QUIC transmit offload as follows:

```
user root;

worker_processes 32;
worker_priority -20;

events {
    worker_connections 65536;
    multi_accept on;
}

error_log logs/error.log error;

http {
    access_log off;

    include mime.types;
    default_type application/octet-stream;

    ssl_certificate /root/server.crt;
    ssl_certificate_key /root/server.key;
    ssl_protocols TLSv1.3;
    ssl_session_cache off;
    ssl_early_data on;

    quic_gso on;                # For non-offload mode
    quic_tx_offload on;         # For offload mode
    quic_offload_dev ens3f4;

    http3_stream_buffer_size 1m;

    keepalive_requests 10000000;
    keepalive_time 24h;
    keepalive_timeout 120s;
    keepalive_min_timeout 10s;

    server {
        listen          443 quic reuseport;
        server_name     localhost;
        add_header Alt-Svc 'h3=":443"';

        location / {
            root    html;
            index  index.html index.htm;
        }
    }
}
```

viii. Creating the test file.

Mount a 10 GB tmpfs on the nginx html directory and create a 5 GB test file:

```
[root@host ~]# mount -t tmpfs -o size=10G,noatime,nodiratime tmpfs
$BASE_DIR/nginx/html
[root@host ~]# cd $BASE_DIR/nginx/html
[root@host ~]# dd if=/dev/urandom of=1G_file oflag=direct bs=1M count=1024
```

ix. Starting the nginx Server.

```
[root@host ~]# taskset -c 0-31 $BASE_DIR/nginx/sbin/nginx -c
$BASE_DIR/nginx/conf/nginx.conf
```

Server Configuration (Broadcom BCM57608)

Perform the following steps on the Server machine when it is fitted with the Broadcom BCM57608 adapter:

- i. Download and install the Broadcom driver package bcm_239.1.170.0a.tar.gz from the [Broadcom Support Portal](#).
- ii. Initializing the driver and interface.
Load the Broadcom bnxt_en driver, set the queue count and ring sizes, bring up the interface, and map interface IRQs to CPU cores:

```
[root@host ~]# modprobe -v bnxt_en
[root@host ~]# ethtool -L ens3f0np0 combined 64
[root@host ~]# ethtool -G ens3f0np0 rx 2047 tx 2047
[root@host ~]# ifconfig ens3f0np0 101.1.1.235/24 up
```

iii. Verifying the driver and firmware versions.

Confirm the driver and firmware versions in use:

```
[root@host ~]# ethtool -i ens3f0np0
driver: bnxt_en
version: 1.10.3-239.1.170.0
firmware-version: 239.1.168.0/pkg 239.1.170.0
```

iv. Verifying UDP GSO.

Confirm that UDP generic segmentation offload (UDP GSO) is enabled on the interface:

```
[root@host ~]# ethtool -k ens3f0np0 | grep -E "^generic-seg|tx-udp-seg"
generic-segmentation-offload: on
tx-udp-segmentation: on
```

v. Configuring OpenSSL and nginx build.

Clone and build QuicTLS (the OpenSSL fork with QUIC APIs), then compile nginx with HTTP/3 support:

```
[root@host ~]# BASE_DIR="/root/OPENSSL"
[root@host ~]# cd $BASE_DIR
[root@host ~]# git clone https://github.com/quictls/openssl
[root@host ~]# cd openssl
[root@host ~]# ./config -O3 -march=native -flto shared \
--prefix=$BASE_DIR/openssl_install --openssldir=$BASE_DIR/openssl_install \
```

```
-Wl,-rpath,$BASE_DIR/openssl_install/lib64
[root@host ~]# make -j${nproc} && make install_sw install_ssldirs
[root@host ~]# cd $BASE_DIR
[root@host ~]# git clone https://github.com/nginx/nginx
[root@host ~]# cd nginx
[root@host ~]# ./auto/configure --prefix=$BASE_DIR/nginx_install \
--with-http_ssl_module --with-http_v2_module --with-http_v3_module \
--with-cc-opt="-I$BASE_DIR/openssl_install/include" \
--with-ld-opt="-L$BASE_DIR/openssl_install/lib64 -L$BASE_DIR/openssl_install/lib -L/usr/lib64 -Wl,-rpath,$BASE_DIR/openssl_install/lib64:$BASE_DIR/openssl_install/lib:/usr/lib64-lbpf" \
--with-threads --with-file-aio
[root@host ~]# make -j${nproc} && make install
```

vi. Configuring nginx.

Use the same nginx.conf as in the Chelsio setup. For the Broadcom run, comment out the two QUIC TX offload directives and keep quic_gso on, so that UDP GSO is used for segmentation:

```
quic_gso on;                # UDP GSO enabled
# quic_tx_offload on;
# quic_offload_dev ens3f4;
```

vii. Creating the test file.

Mount a 10 GB tmpfs on the nginx html directory and create a 5 GB test file:

```
[root@host ~]# mount -t tmpfs -o size=10G,noatime,nodiratime tmpfs
$BASE_DIR/nginx_install/html
[root@host ~]# cd $BASE_DIR/nginx_install/html
[root@host ~]# dd if=/dev/urandom of=1G_file oflag=direct bs=1M count=1024
```

viii. Starting the nginx Server.

```
[root@host ~]# $BASE_DIR/nginx_install/sbin/nginx -c
$BASE_DIR/nginx_install/conf/nginx.conf
```

Client Configuration

Perform the following steps on the Client machine:

- i. Install RHEL 9.5 and compile 6.18.4 kernel.
- ii. Compile and install the latest [Chelsio Unified Wire package](#) and reboot the machine.

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

iii. Set the Tuned profile:

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

iv. Initializing the driver.

```
[root@host ~]# modprobe -v cxgb4 max_eth_qsets=64
[root@host ~]# ifconfig ens3f4 101.1.1.237/24 up
[root@host ~]# t4_perftune.sh -n -s
```

v. Installing OpenSSL 3.5.0, nghttp3, ngtcp2, and h2load with HTTP/3 support.

```
[root@host ~]# BASE_DIR="/root/h2load"
[root@host ~]# mkdir $BASE_DIR
```

- Installing OpenSSL

```
[root@host ~]# cd $BASE_DIR
[root@host ~]# git clone https://github.com/openssl/openssl
[root@host ~]# cd openssl && git checkout openssl-3.5.0
[root@host ~]# ./config shared enable-ktls --
prefix=$BASE_DIR/h2load_install/openssl_build --
openssldir=$BASE_DIR/h2load_install/openssl_build -Wl,-
R$BASE_DIR/h2load_install/openssl_build/lib64
[root@host ~]# make && make install
```

- Installing nghttp3

```
[root@host ~]# cd $BASE_DIR
[root@host ~]# git clone https://github.com/ngtcp2/nghttp3
[root@host ~]# cd nghttp3/ && git submodule update --init && autoreconf -fi
[root@host ~]# ./configure --prefix=$BASE_DIR/h2load_install/nghttp3_build
--enable-lib-only
[root@host ~]# make -j${nproc} && make install
```

- Installing ngtcp2

```
[root@host ~]# cd $BASE_DIR
[root@host ~]# git clone https://github.com/ngtcp2/ngtcp2
[root@host ~]# cd ngtcp2/ && autoreconf -fi
[root@host ~]# ./configure
PKG_CONFIG_PATH=$BASE_DIR/h2load_install/openssl_build/lib64/pkgconfig:$BAS
E_DIR/h2load_install/nghttp3_build/lib/pkgconfig LDFLAGS="-Wl,-
rpath,$BASE_DIR/h2load_install/openssl_build/lib64" --
prefix=$BASE_DIR/h2load_install/ngtcp2_build --enable-lib-only --with-
openssl
[root@host ~]# make -j${nproc} && make install
```

- Installing h2load

```
[root@host ~]# cd $BASE_DIR
[root@host ~]# git clone --recursive https://github.com/nghttp2/nghttp2.git
h2load
[root@host ~]# cd h2load && git reset --hard
21d54f4b926d412048038c8fda7f8c499bc3306e && autoreconf -fi
[root@host ~]# ./configure --prefix=$BASE_DIR/h2load_install/h2load_build
PKG_CONFIG_PATH=/usr/local/lib/pkgconfig:$BASE_DIR/h2load_install/openssl
_build/lib64/pkgconfig:$BASE_DIR/h2load_install/nghttp3_build/lib/pkgconfig:
$BASE_DIR/h2load_install/ngtcp2_build/lib/pkgconfig LDFLAGS="-Wl,-
rpath,$BASE_DIR/h2load_install/openssl_build/lib64 -Wl,-
rpath,$BASE_DIR/h2load_install/ngtcp2_build/lib -Wl,-
rpath,$BASE_DIR/h2load_install/nghttp3_build/lib" --enable-http3 --enable-
app --disable-examples
[root@host ~]# make install
```

vi. Run the h2load performance command from the client node:

```
[root@host ~]# cd $BASE_DIR/h2load_install/h2load_build/bin
[root@host ~]# ./h2load --h3 -c <Connections> -m 1 -t <Threads> -D <Time to
run> https://101.1.1.235:443/<test_file> --warm-up-time 5
```

Example:

```
[root@host ~]# ./h2load --h3 -c 128 -m 1 -t 32 -D 300
https://101.1.1.235:443/1G_file --warm-up-time 5
```

Conclusion

QUIC gives HTTP/3 its security and flexibility, but it asks a great deal of the server CPU, especially at 100 Gbps and above. Chelsio's T7 QUIC TX offload addresses that cost directly. It performs transmit-side QUIC processing in the adapter, so the host cores can spend their time serving clients rather than protecting and segmenting packets.

In the tests described in this paper, an nginx HTTP/3 server on the Chelsio T7 with offload enabled reached **183 Gbps** at MTU 9000 and **164 Gbps** at MTU 1500 with 128 concurrent connections, against **167 Gbps** and **162 Gbps** for the Broadcom BCM57608 with UDP GSO and no QUIC offload. Chelsio delivered 10% to 13% more throughput at MTU 9000 and was within 2% of Broadcom at 64 and 128 connections at MTU 1500, while cutting server CPU utilization by **35% to 51%**: at 128 connections, from **44% to 26%** at MTU 9000 and from **50% to 28%** at MTU 1500.

Chelsio throughput also scales with load. At MTU 1500 it rises 32% from 32 to 128 connections (124 to 164 Gbps) and passes Broadcom from 64 connections onward, while server CPU utilization never exceeds 28% at any test point.

For operators of high-volume HTTP/3 services, offloading the transmit path to the adapter delivers higher throughput at MTU 9000 and equal or higher throughput at MTU 1500 from 64 connections onward, with 35% to 51% less server CPU, leaving the difference for the application, alongside the other protocols the T7 already offloads.

Related Links

[Chelsio T7 NIC Performance Leadership Across Tx, Rx, and BiDi Workloads](#)

[Accelerated Inline IPsec Communication with T7](#)

[400G kTLS/SSL Offload & Encryption with FreeBSD](#)

[AI Networking Solution: Chelsio T7 DPU and S7/T6 SmartNICs](#)

[Chelsio T7 Performance Analysis on Linux for 400 GbE Networking in a Multi-Client Setup](#)

[High-Performance NIC Optimization on Linux With Chelsio T7](#)

[AI Networking: The Role of DPUs](#)