

QUIC TX Offload for High-Throughput HTTP/3 on Linux with Chelsio T7

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 measures the effect on Linux, using an nginx HTTP/3 server and the h2load load generator.

Key results: With QUIC TX offload, an nginx HTTP/3 server on the T7 delivers **183 Gbps** at MTU 9000 and **164 Gbps** at MTU 1500 with 128 concurrent connections. At MTU 9000, offload matches non-offload throughput while cutting server CPU utilization from **44% to 26%**. At MTU 1500, it delivers about twice the non-offload throughput (164 vs. 81 Gbps) at 28% CPU instead of 44%.

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 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 QUIC TX offload and non-offload transmit throughput and server CPU utilization on a Chelsio T7 adapter at 32, 64 and 128 concurrent connections, at MTU 9000 (Figure 1) and MTU 1500 (Figure 2).

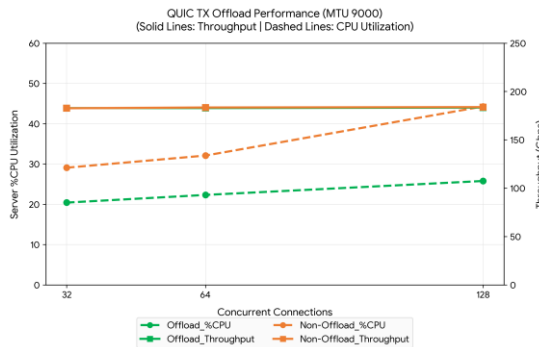


Figure 1 – QUIC TX Offload Performance and CPU Utilization Comparison at MTU 9000

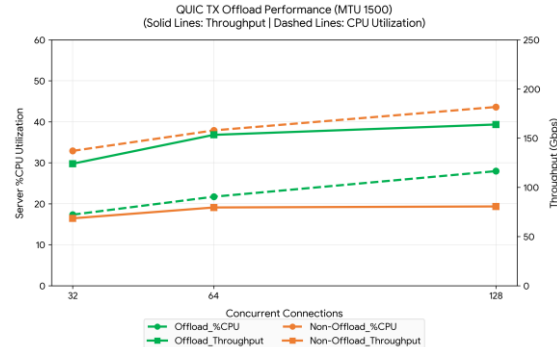


Figure 2 – QUIC TX Offload Performance and CPU Utilization Comparison at MTU 1500

Matching Throughput on Jumbo Frames, About 2x on Standard Frames, and Up to 47% Less Server CPU.

At MTU 9000 (jumbo frames)

- **Same throughput, less CPU:** Offload sustains about **183 Gbps**, matching non-offload, while using up to **42% less** server CPU.
- **Lower server CPU at every load:**
 - 32 connections: 20% CPU with offload vs. 29% without (30% lower)
 - 64 connections: 22% CPU with offload vs. 32% without (30% lower)
 - 128 connections: 26% CPU with offload vs. 44% without (42% lower)

At MTU 1500 (standard frames)

- **Much higher throughput:** Offload delivers **124 to 164 Gbps**, while non-offload stays at **68 to 81 Gbps**.
 - 32 connections: 124 Gbps with offload vs. 68 Gbps without (81% higher)
 - 64 connections: 153 Gbps with offload vs. 80 Gbps without (93% higher)
 - 128 connections: 164 Gbps with offload vs. 81 Gbps without (more than 2x)
- **Lower server CPU at every load:**
 - 32 connections: 17% CPU with offload vs. 33% without (47% lower)
 - 64 connections: 22% CPU with offload vs. 38% without (43% lower)
 - 128 connections: 28% CPU with offload vs. 44% without (36% lower)
- **Scales with connection count:** At MTU 1500, offload throughput rises 32% from 32 to 128 connections (124 to 164 Gbps), while non-offload plateaus at about 80 Gbps from 64 connections onward.
- **Higher CPU efficiency:** At MTU 1500, offload delivers about 3 times the throughput per percentage point of CPU utilization.

Across both MTU sizes

- **Matches or beats non-offload throughput:** Offload delivers the same throughput as non-offload at MTU 9000 and about twice as much at MTU 1500 (164 vs. 81 Gbps at 128 connections).
- **Lower server CPU in every test:** Reductions range from 30% to 47%, leaving more CPU for application workloads.
- **Less dependence on jumbo frames:** At 128 connections, offload keeps about 89% of its jumbo-frame throughput on standard 1500-byte frames (164Gbps vs. 183Gbps). Non-offload keeps only about 44% (81Gbps vs. 184Gbps).

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.

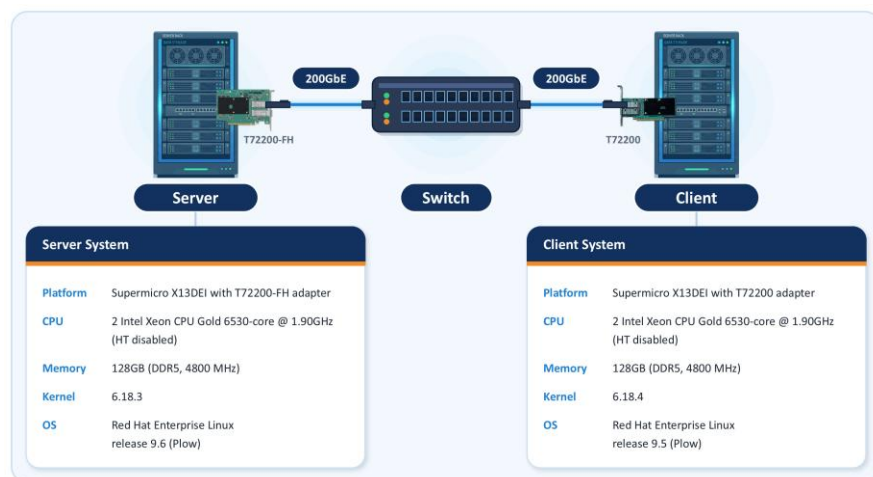


Figure 3 – Test Setup

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

Perform the following steps on the Server machine:

- i. Install RHEL 9.6 and compile 6.18.3 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 quic_offload_install
[root@host ~]# reboot
```

- iii. Tune profile in the TuneD system tuning service:

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

- iv. 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 101.1.1.237/24 up
[root@host ~]# t4_perftune.sh -n -s
```

- v. Configuring QUIC and Nginx Build.

Clone and build QuicTLS OpenSSL, 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
```

vi. **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;
        }
    }
}
```

vii. **Starting the Nginx Server.**

Launch the Nginx server bound to cores 0-31:

```
[root@host ~]# taskset -c 0-31 $BASE_DIR/nginx/sbin/nginx -c
$BASE_DIR/nginx/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. Tune profile in the TuneD system tuning service:

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

- iv. Initializing the driver.

```
[root@host ~]# modprobe -v cxgb4 max_eth_qsets=64
[root@host ~]# ifconfig ens3f4 101.1.1.235/24 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://101.1.1.237:443/<test_file> --warm-up-time 5
```

Example:

```
[root@host ~]# ./h2load --h3 -c 128 -m 1 -t 64 -D 300
https://101.1.1.237:443/1G_test_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 T7 with offload enabled reached **183 Gbps** at MTU 9000 and **164 Gbps** at MTU 1500 with 128 concurrent connections. The same server without offload reached **184 Gbps** and **81 Gbps** respectively. Offload matched non-offload throughput at MTU 9000 and delivered roughly 1.8x to 2.0x the non-offload throughput at MTU 1500, while reducing server CPU utilization by **30% to 47%** across all tests. At 128 connections, CPU utilization fell from **44% to 26%** at MTU 9000 and from **44% to 28%** at MTU 1500.

For operators of content-delivery, streaming and other high-volume HTTP/3 services, offloading the transmit path to the adapter frees server CPU for the application and, on standard 1500-byte frames, delivers up to about twice the traffic from each server. The T7 provides this alongside the other protocols it already offloads, on a single adapter.

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](#)