Accelerated Inline IPsec Communication with T7

A Competitive Performance Analysis

Chelsio Terminator 7

The Terminator 7 (T7) ASIC from Chelsio Communications, Inc. is a seventh generation, high performance 1/10/25/40/50/100/200/400 Gbps, Unified Wire Data Processing Unit (DPU) which offers offload support for a wide range of Crypto (IPsec, TLS/SSL), TCP, NVMe/TCP, NVMe-oF, iSCSI, RDMA (iWARP and RoCEv2) protocols. It is designed specifically to perform computationally intensive cryptographic operations more efficiently than general-purpose CPUs. Servers with system load, comprising of cryptographic operations, see great performance improvement by offloading crypto operations on to the Chelsio T7 adapters. With concurrent support for offloading multiple protocols and crypto operations, Chelsio has taken the Unified Wire solution to the next level.

Chelsio Inline IPsec Acceleration

Internet Protocol Security (IPsec) is an end-to-end security scheme that provides protocols to ensure the authenticity, privacy, and integrity of data in transit. Chelsio's T7 Inline IPsec solution enables the offloading of encryption and decryption operations onto the adapter, delivering accelerated Inline IPsec communication that is well suited for secure site-to-site connectivity over WAN environments. Both Tunnel and Transport modes of IPsec are supported with the ESP protocol, along with Tx and Rx offload capabilities.

Test Results

The following graph compares the throughput and CPU usage using Chelsio T7 crypto accelerator in Inline IPSec offload mode. The numbers are collected using **iperf3** tool with 64 and 128 connections across 32 ipsec tunnels.

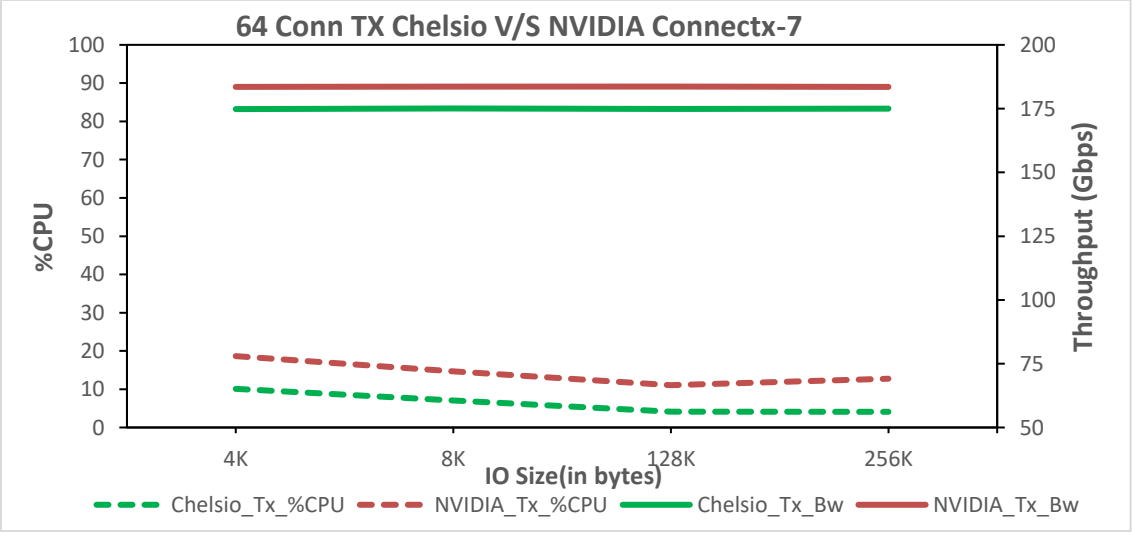

Figure 1 – TX %CPU Usage and Throughput: Chelsio T72200-FH-DPU vs Nvidia CX7 with 64 connections

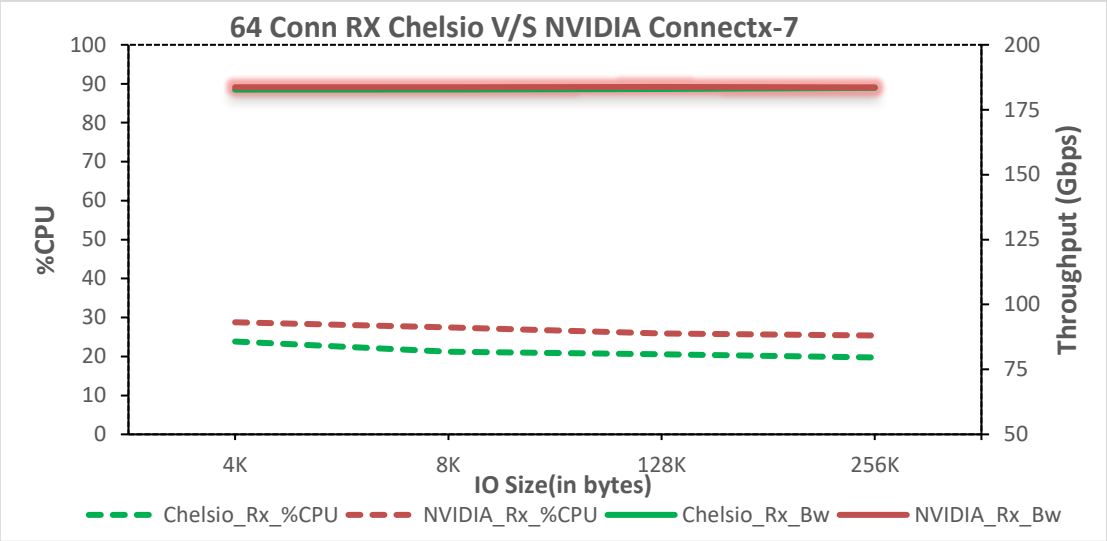

Figure 2 – RX %CPU Usage and Throughput: Chelsio T72200-FH-DPU vs Nvidia CX7 with 64 connections

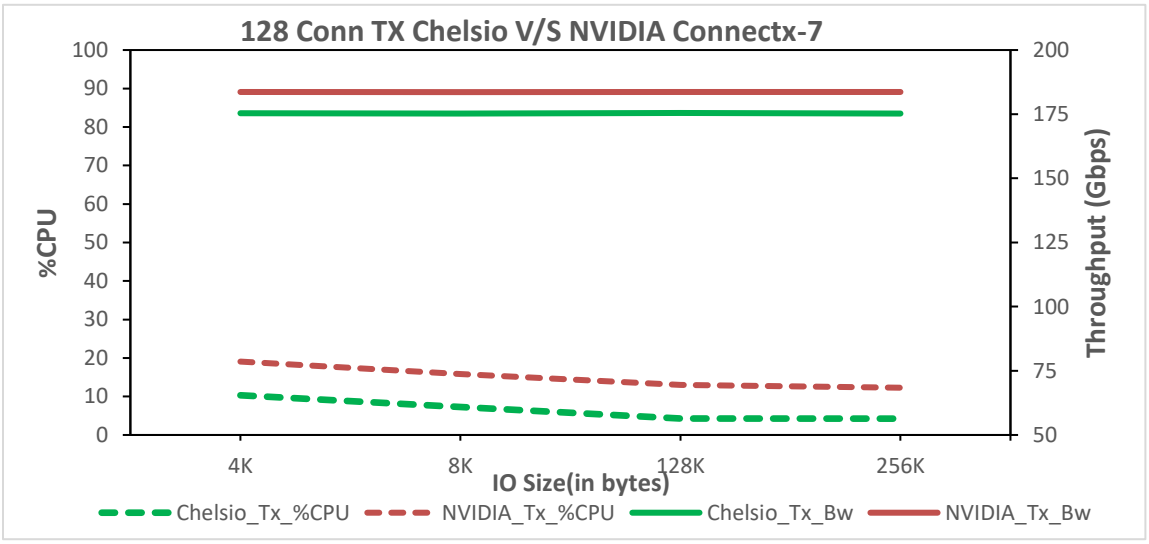

Figure 3 – TX %CPU Usage and Throughput: Chelsio T72200-FH-DPU vs Nvidia CX7 with 128 connections

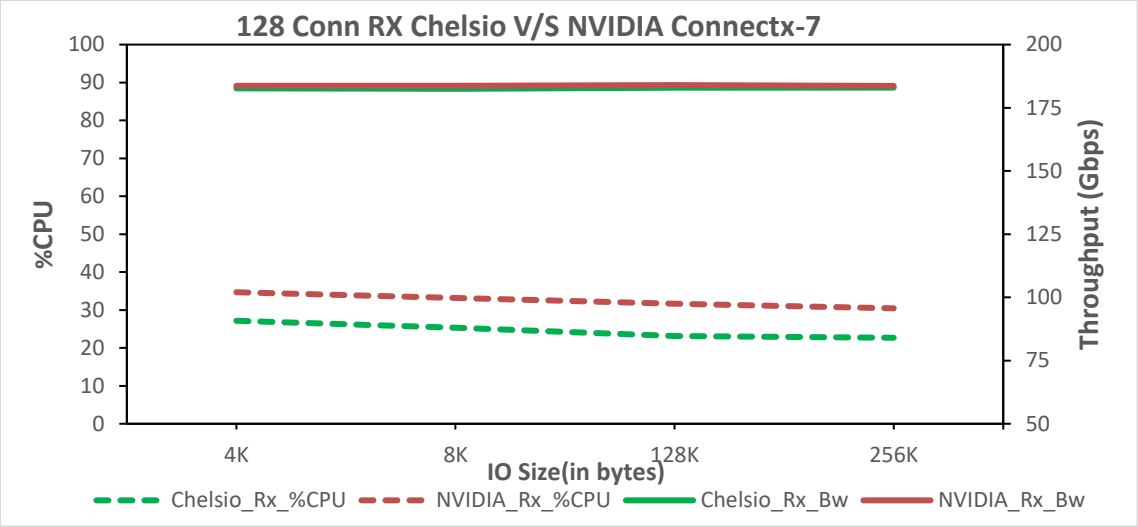

Figure 4 – RX %CPU Usage and Throughput: Chelsio T72200-FH-DPU vs Nvidia CX7 with 128 connections

The graphs illustrate IPsec performance results, demonstrating consistently lower %CPU Usage across both Rx and Tx direction across all connection configurations and I/O sizes. The maximum Rx throughput reached approximately **183 Gbps** in the 2 connection/tunnel configuration with a 256K I/O size while the maximum Tx throughput achieved was approximately 176 Gbps in the 8 connection/tunnel configuration. The results show efficient CPU utilization, with **low %CPU/Gbps** values maintained across different workloads, highlighting the platform's ability to deliver high-performance encrypted networking with optimized resource usage.

Test Configuration

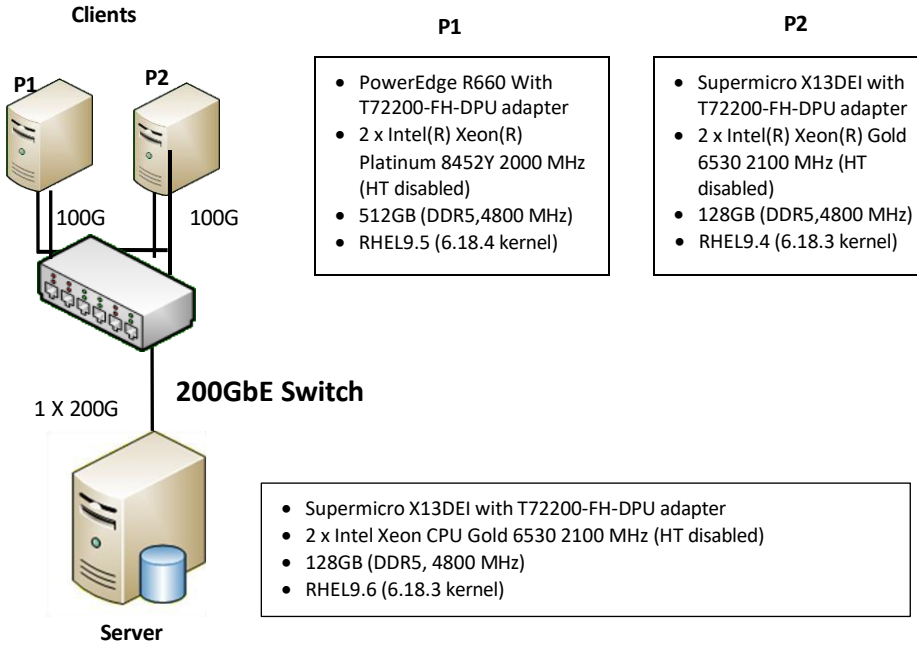

Figure 5 – Test setup

The setup consists of a server connected to three clients using dual ports on each system. The server machine and the clients are configured with the latest Chelsio Unified Wire drivers for Linux. Chelsio T72200-FH-DPU adapter was installed on the Server and Client systems and Port0 of each adapter configured with the standard MTU of 1500B.

Setup Configuration

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

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

iii. Tune profile in the TuneD system tuning service:
```
[root@host ~]# tuned-adm profile network-throughput
```
iv. Load the Chelsio NIC driver (cxgb4)
```
[root@host ~]# modprobe cxgb4 enable_ulds=0x200
```
v. Load the Chelsio Inline IPSec driver
```
[root@host~]# modprobe ch_ipsec
```
vi. Create 32 IP Aliases and assign IP to them
```
[root@host~]# for i in `seq 1 32`;do ifconfig ethX:$i $i.0.0.1/24 up;done
```

Clients – P1 and P2

i. Install RHEL 9.5 and compile 6.18.4 kernel on Peer1 and RHEL 9.4 and compile 6.18.3 kernel on Peer2.

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
```
vii. Load the Chelsio NIC driver (cxgb4)
```
[root@host ~]# modprobe cxgb4 enable_ulds=0x200
```
viii. Load the Chelsio Inline IPSec driver
```
[root@host~]# modprobe ch_ipsec
```
ix. Create 16 IP Aliases on each peer and assign IP to them

On Client P1 -
```
[root@host~]# for i in `seq 1 16`;do ifconfig ethX:$i $i.0.0.2/24 up;done
```

On Client P2 -
```
[root@host~]# for i in `seq 17 32`;do ifconfig ethX:$i $i.0.0.2/24 up;done
```

Nvdia CX7

i. Install RHEL 9.6 and compile 6.18.3 kernel.

ii. Tune profile in the TuneD system tuning service:
```
[root@host ~]# tuned-adm profile network-throughput
```
iii. Load the NIC driver (mlx5_core)
```
[root@host ~]# modprobe mlx5_core
```
iv. Create 32 IP Aliases and assign IP to them
```
[root@host~]# for i in `seq 1 32`;do ifconfig ethX:$i $i.0.0.1/24 up;done
```

Tunnel Configuration

i. Run the following script on the Server. Here ethX is the interface on the Server while ethY and ethZ are the interfaces on Clients P1 and P2 respectively

```
declare -a spi
for i in {1..33}; do
  spi[i]=$(xxd -p -l 4 /dev/random)
done
declare -a reqid
for i in {1..33}; do
  reqid[i]=$(xxd -p -l 4 /dev/random)
done

dev=ethX
for i in {1..32}; do
 local_ip=$i.0.0.1
 remote_ip=$i.0.0.2
#If Server is Chelsio T72200-FH-DPU
ip xfrm state add src $remote_ip dst $local_ip proto esp spi 0x${spi[$i]}
reqid 0x${reqid[$i]} mode transport aead \"rfc4106(gcm(aes))\"
0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 sel src 0.0.0.0/0 dst 0.0.0.0/0
offload dev $dev dir in replay-window 128
#If Server is NVDIA CX7
ip xfrm state add src $remote_ip dst $local_ip proto esp spi 0x${spi[$i]}
reqid 0x${reqid[$i]} mode transport aead \"rfc4106(gcm(aes))\"
0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 sel src 0.0.0.0/0 dst 0.0.0.0/0
offload dev $dev dir in
#The following commands apply to both Chelsio T72200-FH-DPU and NVDIA CX7
ip xfrm state add src $local_ip dst $remote_ip proto esp spi 0x${spi[$((i +
1))]} reqid 0x${reqid[$((i + 1))]} mode transport aead \"rfc4106(gcm(aes))\"
0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 sel src 0.0.0.0/0 dst 0.0.0.0/0
offload dev $dev dir out
 ip xfrm policy add src $local_ip dst $remote_ip dir out tmpl src $local_ip
dst $remote_ip proto esp reqid 0x${reqid[$((i + 1))]} mode transport
 ip xfrm policy add src $local_ip dst $remote_ip dir fwd tmpl src $local_ip
dst $remote_ip proto esp reqid 0x${reqid[$i]} mode transport
 ip xfrm policy add src $local_ip dst $remote_ip dir in tmpl src $local_ip dst
$remote_ip proto esp reqid 0x${reqid[$i]} mode transport
done

dev=ethY
for i in {1..16}; do
 local_ip=$i.0.0.2
 remote_ip=$i.0.0.1
ssh $PEER1 " ip xfrm state add src $local_ip dst $remote_ip proto esp spi
0x${spi[$i]} reqid 0x${reqid[$i]} mode transport aead \"rfc4106(gcm(aes))\"
0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 sel src 0.0.0.0/0 dst 0.0.0.0/0
offload dev $dev dir out"
ssh $PEER1 " ip xfrm state add src $remote_ip dst $local_ip proto esp spi
0x${spi[$((i + 1))]} reqid 0x${reqid[$((i + 1))]} mode transport aead
\"rfc4106(gcm(aes))\" 0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 sel src
0.0.0.0/0 dst 0.0.0.0/0 offload dev $dev dir in replay-window 128"
ssh $PEER1 " ip xfrm policy add src $local_ip dst $remote_ip dir out tmpl src
$local_ip dst $remote_ip proto esp reqid 0x${reqid[$i]} mode transport"
ssh $PEER1 " ip xfrm policy add src $local_ip dst $remote_ip dir fwd tmpl src
$local_ip dst $remote_ip proto esp reqid 0x${reqid[$((i + 1))]} mode
transport"
ssh $PEER1 " ip xfrm policy add src $local_ip dst $remote_ip dir in tmpl src
$local_ip dst $remote_ip proto esp reqid 0x${reqid[$((i + 1))]} mode
transport"
sleep 1
```

```
done

dev=ethZ
for i in {17..32}; do
 local_ip=$i.0.0.2
 remote_ip=$i.0.0.1
ssh $PEER2 " ip xfrm state add src $local_ip dst $remote_ip proto esp spi
0x${spi[$i]} reqid 0x${reqid[$i]} mode transport aead \"rfc4106(gcm(aes))\"
0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 sel src 0.0.0.0/0 dst 0.0.0.0/0
offload dev $dev dir out "
ssh $PEER2 " ip xfrm state add src $remote_ip dst $local_ip proto esp spi
0x${spi[$((i + 1))]} reqid 0x${reqid[$((i + 1))]} mode transport aead
\"rfc4106(gcm(aes))\" 0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 sel src
0.0.0.0/0 dst 0.0.0.0/0 offload dev $dev dir in replay-window 128"
ssh $PEER2 " ip xfrm policy add src $local_ip dst $remote_ip dir out tmpl src
$local_ip dst $remote_ip proto esp reqid 0x${reqid[$i]} mode transport"
ssh $PEER2 " ip xfrm policy add src $local_ip dst $remote_ip dir fwd tmpl src
$local_ip dst $remote_ip proto esp reqid 0x${reqid[$((i + 1))]} mode
transport"
ssh $PEER2 " ip xfrm policy add src $local_ip dst $remote_ip dir in tmpl src
$local_ip dst $remote_ip proto esp reqid 0x${reqid[$((i + 1))]} mode
transport"
sleep 1
done
```

ii. Verify the xfrm policies:

```
[root@host~]# ip xfrm state list
```

iii. Start iperf3 servers on Clients P1 and P2:

On Client P1 -

```
[root@host~]# for i in `seq 1 16`; do numactl -N0 -m0 iperf3 -s -p 600$i -D ;
done
```

On Client P2 -

```
[root@host~]# for i in `seq 17 32`; do numactl -N0 -m0 iperf3 -s -p 600$i -D ;
done
```

iv. Start iperf3 client on Server:

Transmit (TX) Test: To measure transmit performance from the Server to Clients,

```
[root@host~]# for i in `seq 1 32`; do iperf3 -c $i.0.0.2 -Z --skip-rx-copy -N
-t 60 -i 0  -l <IO Size> -P <Conn/tunnel> -f g -O 10 -p 600$i -M 1424
```

Receive (RX) Test: To measure receive performance from Clients to the Server,

```
[root@host~]# for i in `seq 1 32`; do iperf3 -c $i.0.0.2 -Z --skip-rx-copy -N
-t 60 -i 0  -l <IO Size> -P <Conn/tunnel> -f g -O 10 -p 600$i -M 1424 -R
```

Notes:

Tests were with “IO Size” of 4K, 8K, 128K and 256K

Conclusion

The performance evaluation demonstrates that Chelsio's T7 Inline IPsec acceleration solution delivers high RX throughput with efficient CPU utilization across a wide range of concurrent connection workloads. By offloading cryptographic processing from the host CPU to the T7 adapter, the platform achieves a low CPU-per-Gbps ratio, **enabling secure, encrypted communication with minimal impact on system resources.** The results highlight the scalability and efficiency of the Chelsio T7 architecture, sustaining stable throughput even at higher connection counts. With support for both Tx and Rx offload, along with Tunnel and Transport modes of IPsec, the Chelsio T7 Unified Wire Adapter provides an optimized solution for secure, high-performance networking deployments in modern data center and WAN environments.

Related Links

AI Networking Solution: Chelsio T7 DPU and S7/T6 SmartNICs
High-Performance NIC Optimization on Linux With Chelsio T7
AI Networking Solution: Chelsio T7 DPU and S7/T6 SmartNICs
T7 Product Brief
AI Networking: The Role of DPUs
Offload Protocols with Inline IPsec demonstration on T7 Emulation Platform
iSCSI JBOF with T7
NVMe/TCP and iSCSI JBOF with T7