DELL EMC

Whitepaper

Exploring Intel® QAT on MX series blade servers

Computing offloads for encryption and compression

Andy Butcher, Technical Staff, Server Advanced Engineering, Dell EMC

Gordon McFadden, Lead Architect, Intel® QuickAssist Technology

Abstract

Integrated Intel® QuickAssist Technology on the MX blade servers provides beneficial CPU offloads for encryption and compression operations. Quantitative examples are described, including information on how to enable and use this feature of the chipset.

Executive Summary

Workload performance is important to Enterprise IT customers. Whether running web servers, distributed storage, data analytics, cloud services, or any custom application, computing performance affects both user satisfaction and cost of ownership. General purpose CPUs are very good at most operations necessary for these workloads. However, some operations can be executed faster by custom-designed circuitry. The cost and benefit tradeoff sometimes makes a compelling case for the usage of accelerator peripherals. Common accelerator peripherals include FPGAs (field programmable gate arrays) and GPUs (graphics processing units). This paper discusses the Intel® QuickAssist Technology (Intel® QAT) peripheral.

Table of contents

1. Introduction

PowerEdge MX is the first Dell EMC server to offer a software licensing option to enable Intel® QuickAssist Technology. It provides a software-enabled foundation for security, authentication, and compression, and significantly increases the performance and efficiency of standard platform solutions. This paper will explore uses of Intel® QAT with two examples.

1.1 Encryption and Key Generation

Many users will be familiar with the “https” prefix on frequently-visited websites. Behind all of these secure websites is an implementation of TLS (transport layer security) or its predecessor SSL (secure sockets layer). Each protocol entails a “handshake” between the client and server that establishes authenticity of the server and creates a session key for encrypting the exchanged data. These Public Key Encryption (PKE) algorithms, historically performed by software, can be offloaded from the CPU into the Intel® QAT engine for providing significant performance gains for Web Server, Content Delivery Networks, eCommerce, VPN, Firewall or Security Load Balancer and Wan Acceleration solutions.

1.2 Data Compression and Decompression

Users of compressed file formats will be familiar with the benefit of another function provided by Intel® QAT. Like cryptography, compression and decompression can be compute-intensive functions. Intel® QuickAssist Technology (Intel® QAT) is comprised of acceleration engines for data compression as well, yielding faster performance, lower latency and higher throughput for software and systems that rely on compressed data such as storage, web compression, big data, or high-performance computing (HPC).

Compressing data before it is stored on a hard drive provides the dual benefit of reducing the time to complete both writes and subsequent reads, and increasing the amount of data that can be stored on the drive system.

Compressing data before transmission over a network improves overall network utilization by reducing the number of TCP/UDP segments and as a result the number of IP packets. This allows more data to be sent and received without the need to expand the network interfaces.

In both of these examples, the benefit is achieved by compressing and decompressing data before it is consumed by a slower component in the compute infrastructure.

1.3 Software Licenses for Intel® QAT in PowerEdge MX

Intel® QAT has a long history with the deliveries of the 8920 model and the subsequent 8955 on PCIe cards. In the Intel® Xeon® Processor Scalable Family, Intel® is making the next generation of Intel® QAT available with significantly improved performance in a chipset-integrated version. Dell EMC is offering hardware-enabling licenses for chipset Intel® QAT on the MX series blade servers (MX740c and MX840c). These licenses can be installed without the need to add hardware to the system and occupy slots. Depending on the license level installed and the performance level desired, the chipset based Intel® QAT will be programmed to offer the bandwidth performance as defined below, mimicking the performance of the latest model 8960 and model 8970 PCIe cards. The licenses are installed through the iDRAC license manager.

QAT license	Compression	Encryption	RSA
40G ‘mid-range’	28 Gb/s	40 Gb/s	40K Ops/s
100G ‘top performance’	65 Gb/s	100 Gb/s	100K Ops/s

1.4 Software

Software for the Intel® QuickAssist Technology is provided through the Intel open source site.[1] The applicable drivers are associated with the C62x chipset. Application and library examples are posted on 01.org along with the Quick Start Guide API Programmer's Guide and other useful collateral allowing users to build upon these open source libraries and examples or build their own applications. Release notes identify operating system compatibility.

Alternatively, the software can be installed onto a Linux system with an RPM manager such as yum, as shown in Appendix section 6.2. In this example, Intel® QAT and its companion application and library QATzip were installed.

1.4.1 DPDK (Data Plane Development Kit)

An open source project consisting of a set of libraries and drivers for fast packet processing, DPDK employs PMDs (Poll Mode Drivers) to interact with user space software, avoiding latency expensive context switches between kernel and user space. Instructions on installing the Intel® QAT PMD can be found on the DPDK web site.[2] Using DPDK, performance benefit has been demonstrated for IPsec (Internet Protocol Security), which provides security at a lower level in the protocol stack than TLS. For further reading on IPSEC, see the Getting Started Guide[3] and Sample application usage.[4]

1.4.2 Compression and Decompression

The primary vehicle for application development or application integration for data compression and decompression for Linux and Windows Server is QATZip, which is a user space library that produces data in standard gzip format. See the most recent release notes for the drivers and the API application guides for more information on data compression.

2. Intel® QAT on MX7000

This section includes several examples of Intel® QAT applications, but this list of potential uses of Intel® QAT is by no means exhaustive. Drivers and APIs are available for custom applications that require encryption or compression. Not included in this section is the NGINX web server, which has been integrated with Intel® QAT encryption and PKE for superior performance measured in connections per second. Also, the commonly used encryption library Openssl has been modified to utilize Intel® QAT.

2.1 Lab Setup

The basis for these experiments was the MX7000 series blade server product line. The blade chassis hosts eight two-socket sleds in a 7U space, depicted in Figure 1. For the compression experiment, only one sled was necessary. For the IPSec experiment, two blades were connected through the IOM (input/output module, which serves as the aggregate networking switch for the blades in the chassis). In both cases, an MX740c was used, depicted in Figure 2. It should be noted that there are many other beneficial applications of this technology, as stated (for example, web server applications that offload key exchange and encryption).

Figure 1 – MX7000 chassis

Figure 2 – MX740c

The pertinent internal connections are represented in Figure 3 and Figure 4. Each sled plugs in directly to the IOMs; there is no midplane. This ensures signal integrity for speed upgrades well into the future. Fabric A was used as the "east-west" connection between the two blades in the IPSec experiment.

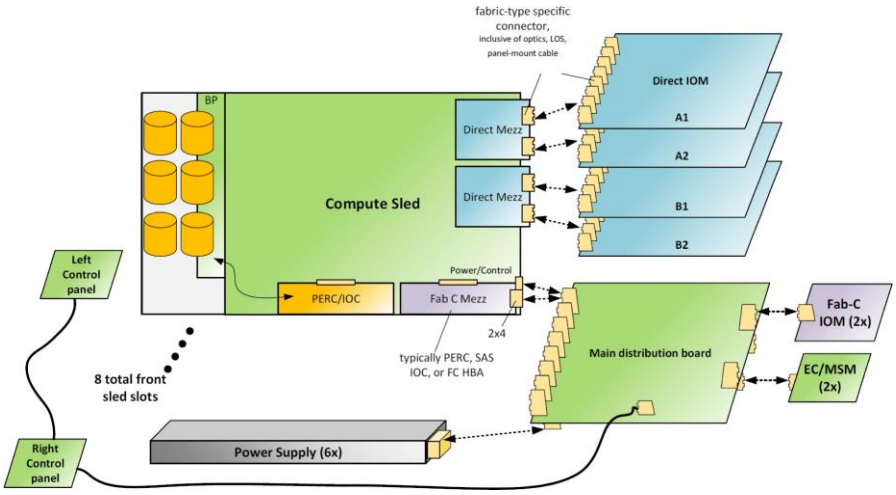

Figure 3 – MX7000 components

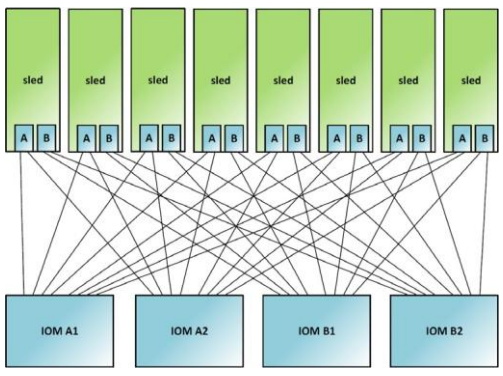

Figure 4 – Fabric A and B Interconnect

3. Example 1 – Compression

3.1 Background – Platform Hardware and Capability

The latest generation the Intel® QAT device provides three separate PCIe end-points, each with 10 compression/ decompression engines. In the Dell EMC MX740c, the device is integrated into the Intel® C62x Chipset Platform Hub Controller (PCH) and is connected to the CPU with a 16-lane PCIe3 link. High speed DMA (Direct Memory Access) transactions are used by the engines to transfer data. The compression engines support the creation of both static and dynamic Huffman headers. Various search depths provide access to increased compression ratios. Both Adler32 checksum and the CRC32 hash are supported.

3.2 Software Features - QATzip

QATzip is built on the Intel® QAT software interface, providing additional features and capabilities, as well as being focused on ease-of-use. QATzip initializes hardware resource on the first call, eliminating the need to initialize the QAT device separately. If QATzip is installed on a platform that is not accelerated, it will seamlessly use software libraries to provide compression services. This allows an operator to install a common software stack on a set of platforms with or without Intel® QAT enabled. QATzip was designed to work in multithreaded environments, potentially with a large number threads starting and stopping throughout a workload's life cycle. To support multithreading, QATzip:

- intelligently acquires instance and memory resources each time a QATzip API is invoked. A single lightweight lock is used to acquire resources.
  On subsequent operations, the thread will attempt to acquire the same resources first. Thus, if there are fewer threads than resources, then there will be no lock contention.
- Releases the lock on the instance and memory resource, allowing them to be used by a different thread.
- Registers a termination function, allowing the last thread that terminates to free the memory resources back to the operating system, and release the Intel® QuickAssist Technology instances.

These actions allow a large number of threads to seamlessly and efficiently share a small number of endpoints and engines.

For environments with high network traffic or file caching, where system memory contention is likely to occur, the memory driver that ships with Intel® QAT can be configured to consume memory from allocated huge pages.

QATzip is optimized for larger files and can be configured to use software-based compression if the clear-text size is small enough that the trip to the accelerator and back would outweigh the benefits of offloading the job to the accelerator.

By including RFC compliant gzip headers with extensions as illustrated in Figure 5, the QATzip library can effectively perform parallel decompression on file portions, concurrently taking advantage of the multiple engines to provide high data throughput.

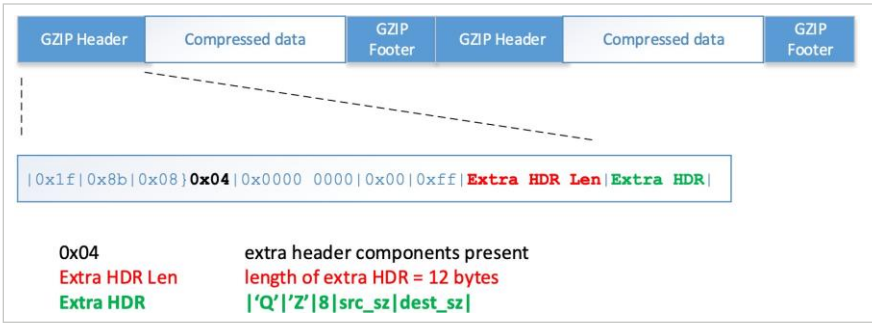

Figure 5 – QATzip header extensions for increased decompression performance

During the compress operation, the input is divided into chunks – default size is 64 kilobytes – and passed to the compression engine in parallel. Each gzip structure contains the source and destination size, meaning they can be simultaneously decompressed. The extra header adds 12 bytes. In most circumstances, this does not materially impact the compression ratio. This multi-gzip structure complies with RFC 1952 and is correctly decompressed by both QATzip and software decompression utilities. QATzip can be configured to produce a single gzip header without the extension, which is used in webservers, such as Nginx, for consumption by browsers.

3.3 Programming Example

The QATzip has been carefully designed to provide a simple programming API. The following code segment is complete:

```
[gmcfadde@localhost small]$ cat Makefile

INC_DIR=/opt/intel/QATzip/include/

smallQz: main.c
	gcc -g -O0 -I$(INC_DIR) main.c -o smallQz -lqatzip

[gmcfadde@localhost small]$ cat main.c
#include <stdio.h>
#include <stdlib.h>

#include “qatzip.h”

int main( int argc, char *argv[ ] )
{
	int rc;
	unsigned int i_len, o_len;
	unsigned char *in, *out;
	QzSession_T sess = {0};

	i_len = 128*1024;
	o_len = 128*1024;

	in = malloc(i_len);
	out = malloc(o_len);
	if ( NULL == in || NULL == out )
	{
		printf( “memory failure\n” ); return 2;
	}
	rc = qzCompress( &sess, in, &i_len, out, &o_len, 1 );

	printf( “In len = %d, out len = %d, rc = %d\n”, i_len, o_len, rc );

	return rc;
}
[gmcfadde@localhost small]$
[gmcfadde@localhost small]$ touch main.c
[gmcfadde@localhost small]$ make
gcc -g -O0 -I/opt/intel/QATzip/include/ main.c -o smallQz -lqatzip
[gmcfadde@localhost small]$ ./smallQz
In len = 131072, out len = 234, rc = 0
[gmcfadde@localhost small]$
```

As can be seen, there is no explicit requirement to initialize access to hardware. This is taken care of in the first QATzip API call.

Applications can explicitly initiate and setup the QATzip library to configure specific session parameters used in the compress and decompress calls. Examples are shown in Table 1.

Parameter	Range	Default	Notes
Huffman Encoding	Dynamic | Static	Dynamic	
Headers	None | gzip | gzip with extenstions	gzip with extensions	
Compression Level	1-9	1	level nine invokes a software compression algorithm for the best possible compression ratio
Size of data passed to hardware	1 KB - 512 KB	64 KB	
Min size for hardware	128 bytes and higher	1024 bytes	no upper limit

Table 1 – Selected QATzip session parameters

Finally, the QATzip library includes its own allocation and free APIs – qzMalloc() and qzFree(). If appropriate, an application can allocate memory that will be compressed or decompressed using these APIs. qzMalloc will returned pinned memory if available. This will lead to improved latency as the data is already in DMA-friendly memory and will not have to be copied. The qzCompress and qzDecompress APIs accept both pinned memory and regular memory without the caller having to indicate which type is being used in a specific call.

3.4 Example Software stack

Figure 6 shows the integration of QAT into NGINX, providing both encryption and compression for the web server. This diagram puts the QATzip library in context, showing its relation to application software and the driver.

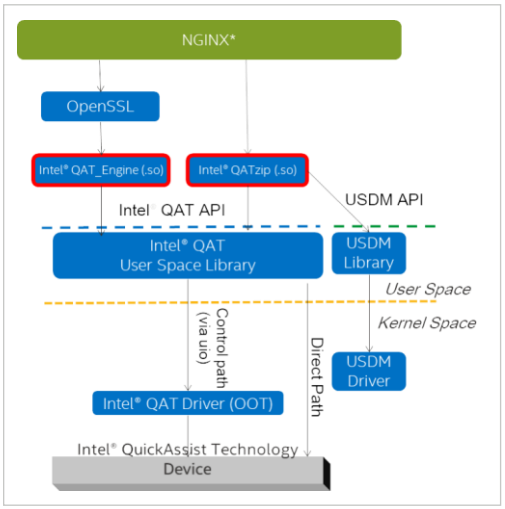

Figure 6 – QAT integration into the NGINX web server

3.5 Experiment Results

The results shown in Table 2, Figure 7, and Figure 8 were obtained on an MX740c blade server with two Intel® Xeon® Gold 5117 CPUs running at 2 GHz. There was 281GB of RAM on the machine, and files were cached so disk operations would not skew the results. The operating system was Red Hat Enterprise Linux 7.6. The files under test were created from arbitrary data. Intel has already published results from the Calgary Corpus dataset, so we chose to confirm performance with randomly selected binary and text data,[1] which was also useful in creating large files so the performance could be easily observed over an extended time. For the comparison, gzip performance was collected alongside QATzip performance. We chose a compression level (4) that offered the most similar ratios. The times were captured as the total of "user" plus "system" time for the threads executing the compression. Commands used are shown in Appendix section 6.1.

		gzip/gunzip			QATzip			
Test file	**Size**	**Comp. ratio**	**gzip comp. time(s)**	**gzip decomp. time (s)**	**comp. ratio**	**qzip comp. time(s)**	**qzip decomp. time (s)**	**% accel.**
file #1	1GB	1.86	16.76	5.89	1.87	1.75	1.08	89.6%
file #2	1GB	8.45	38.28	8.90	9.16	2.43	2.08	93.6%
file #3	1GB	1.25	43.17	9.61	1.26	2.59	2.29	94.0%
file #4	1GB	1.03	39.74	6.46	1.03	2.71	2.32	93.2%

Table 2 – Compression and Decompression comparison between a software-only (CPU) operation vs. a QAT assisted operation

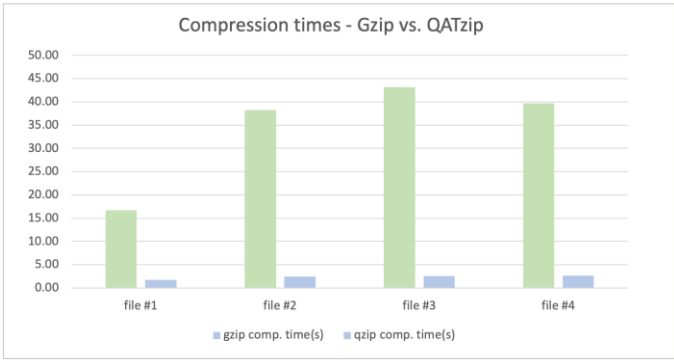

Figure 7 – Compression time comparison

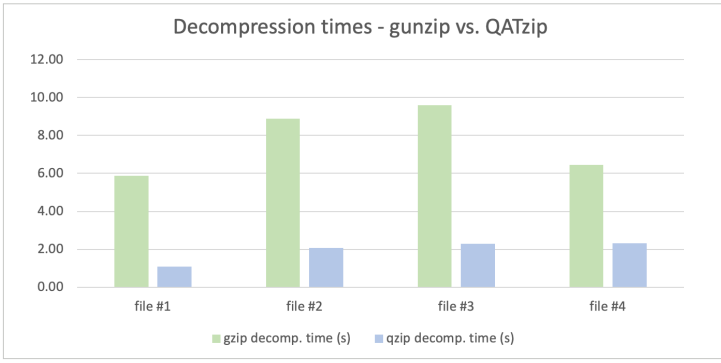

Figure 8 – Decompression time comparison

1 The exact content was less important in this experiment as investigating some variety of compressible vs. uncompressible data. For this, some .tar files for FPGA tool installers were used (not compressible), and there was some text data (source code) mixed with the binary data. The data that was compressible is evident in the results.

4. Example 2 – IPsec

In this experiment, the performance benefit of offloading encryption to the Intel QAT device in a simulated VPN tunnel is demonstrated. The tunneling machine, running VPP,[6] was exercised with a traffic generator running TRex. To perform the IPsec encryption, three methods were compared, employed by VPP at the tunnel: 1) openssl library without offload 2) Intel® AES-NI, and 3) Intel® QAT offload.

4.1 Lab Setup

Figure 10 shows the lab setup. The MX740c machines were configured as follows:

- Tunneling Server: Intel® Xeon® Gold 5117 CPUs @ 2GHz, 30GB memory, Intel® Ethernet 25G 2P XXV710 Mezzanine card, RHEL 7.6
- Traffic Generator: Intel® Xeon® Gold 6146 CPUs @ 3.2GHz, 376GB memory, Intel® Ethernet 25G 2P XXV710 Mezzanine card, RHEL 7.6

The p1p1 and p1p2 are attached to Fabric A1 and A2 switches respectively in the MX7000 chassis.

The configuration files and procedures for this experiment are detailed in the Appendices, and additional reference material and links are listed in the References section.

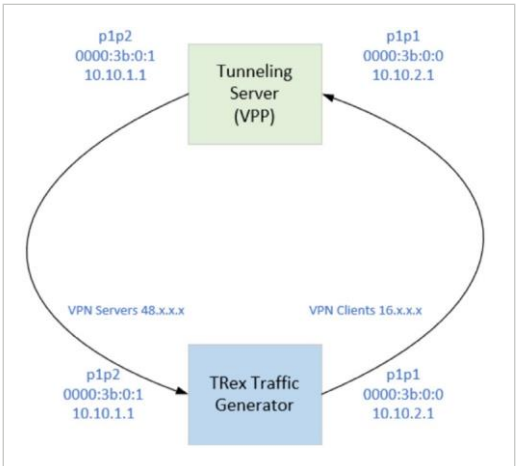

Figure 9 – Test setup for the IPsec performance measurement

4.2 IPSec Performance Results

This section provides summary of results observed. The graph in Figure 9 shows that the benefit of offload, measured by the data throughput per second, increased with packet size. This is intuitive because the overhead of the offload operation is incurred more times per unit of data with the smaller packets. The figures measured by TRex are reported in Table 3.

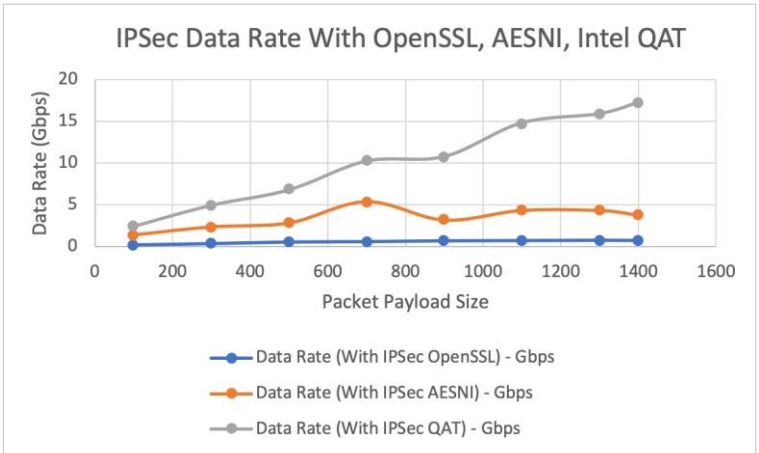

Figure 10 – IPsec tunnel performance for encryption options

Packet Size (Bytes)	Data Rate (With IPSec OpenSSL) - Gbps	Data Rate (With IPSec AESNI) - Gbps	Data Rate (With IPSec Intel® QAT) - Gbps
100	0.242	1.43	2.46
300	0.39	2.39	4.97
500	0.569	2.86	6.82
700	0.615	5.39	10.25
900	0.706	3.2	10.79
1100	0.735	4.34	14.81
1300	0.768	4.34	15.91
1400	0.745	3.78	17.24

Table 3 – Resulting data throughput for various encryption calculation methods

5. Conclusion

Acceleration and offload functions of Intel® QuickAssist Technology (QAT) demonstrated in this paper are encryption/decryption and compression/decompression. IPsec tunneling was used as a workload to show the benefit of encryption and decryption offload. QATzip was used to demonstrate the performance of accelerated compression and decompression. In both cases, faster performance was offered by offloading operations onto the QAT engines. Other workloads not shown are also beneficiaries of this technology, including web server applications.

6. Appendices

6.1 Qzip and Gzip commands

```
gzip -4 -k '"$filename"
gunzip -f '"$filename"
qzip -L 4 -k '"$filename"
qzip -d -k '"$filename"
```

6.2 Example installation using yum – QAT and QATzip

```
root@gm-fed-28 ~]# cat /etc/yum.repos.d/intel-qat.repo
[intel-qat]
name=Intel QAT
baseurl=https://download.01.org/QAT/repo/
gpgcheck=0

[root@gm-fed-28 ~]# yum install QATzip
Dependencies resolved.
================================================================================
 Package        Arch          Version               Repository        Size
================================================================================
Installing:
 QATzip         x86_64        0.2.5-02              intel-qat          40 k
Installing dependencies:
 QAT            x86_64        1.7.0-450b34          intel-qat         4.5 M
Transaction Summary
================================================================================
Install 2 Packages

…
  Verifying        : QATzip-0.2.5-02.x86_64                          1/2
  Verifying        : QAT-1.7.0-450b34.x86_64                         2/2
Installed:
  QATzip.x86_64 0.2.5-02              QAT.x86_64 1.7.0-450b34
Complete!
[root@gm-fed-28 ~]#
```

6.3 Server Setup for Traffic Generator

The section describes the steps required to run TRex on the traffic generating server. See links for additional information on DPDK and getting started with IPsec.[9] There is additional reference material for TRex online.[10]

1. Disable the network ports being used for DPDK:
   cd /etc/sysconfig/network-scripts
   ifdown p1p1
   ifdown p1p2
   service network restart

2. Download and untar DPDK in directory "dpdk".
   cd /home/dell
   git clone http://dpdk.org/git/dpdk

3. Compile/Setup DPDK
   cd /home/dell/dpdk
   export DESTDIR=x86_64-native-linuxapp-gcc_destdir
   make T=x86_64-native-linuxapp-gcc install -j100

4. Insert DPDK UIO driver into Kernel:
   modprobe uio
   insmod x86_64-native-linuxapp-gcc/kmod/igb_uio.ko

5. Bind DPDK UIO module with PCI device. (Note: This step is generally not needed as Trex binds to DPDK itself with information from its configuration file with specified PCIe ID's.)
   ./usertools/dpdk-devbind.py -b igb_uio 0000:3b:00.0
   ./usertools/dpdk-devbind.py -b igb_uio 0000:3b:00.1

6. Generate Trex configuration file.
   cd /home/dell/trex-core/scripts
   ./dpdk_setup_ports.py -i
   Follow the prompts to generate trex_cfg.yaml.
   This file needs to be generated only once, or one can use the file generated already.

7. Open 2 Linux terminals

8. On First terminal, launch Trex Traffic generator:
   cp -f trex_cfg.yaml /etc/
   cd /home/dell/trex-core/scripts
   ./t-rex-64 --cfg /etc/trex_cfg.yaml -i

9. On 2nd terminal, launch Trex Console:
   cd /root/my_home/trex-core/scripts
   ./trex-console

10. In Trex Console, execute following commands to start Trex traffic:
    portattr -a --prom on
    start -f qat_tests/udp_1pkt_simple_<payload_size>.py -m 25gbps -p 0 --force
    stop -a
    Where: payload_size varies from 100-1400 bytes.
    Trex traffic files with it are already created within qat_tests directory.

11. Network traffic for each of the tests can be started and stopped with commands in previous step.

6.4 Server Setup for the Tunneling Server

The section describes the steps required to run VPP on Tunneling Server that performs the IPsec encryption. A command line reference for VPP can be found online,[11] with additional reference material from Intel.[12]

6.4.1 Install and Enable Intel® QAT

These steps enable Intel® QAT on then Server-Sun.

1. Check for PCI devices (information only)
   [root@sun dell]# lspci | grep processor
   60:00.0 Co-processor: Intel Corporation C62x Chipset QuickAssist Technology (rev 03)
   61:00.0 Co-processor: Intel Corporation C62x Chipset QuickAssist Technology (rev 03)
   62:00.0 Co-processor: Intel Corporation C62x Chipset QuickAssist Technology (rev 03)

2. Configure and compile QAT
   cd /home/dell/qat
   ./configure --enable-icp-sriov=host
   make -j10
   make install
   lspci -d:37c9 -k

3. Repeating Step 1 will show the Virtual Functions rendered by QAT install step (information only).

6.4.2 Set and Run VPP

These steps provide information on how to run VPP with OpenSSL, Intel AESNI and QAT.

1. Disable the network ports being used for the DPDK:
   cd /etc/sysconfig/network-scripts
   ifdown p1p1
   ifdown p1p2
   service network restart

2. Download and untar DPDK in directory "dpdk".
   cd /home/dell
   git clone http://dpdk.org/git/dpdk

3. Compile/Setup DPDK
   cd /home/dell/dpdk
   export DESTDIR=x86_64-native-linuxapp-gcc_destdir
   make T=x86_64-native-linuxapp-gcc install -j100

4. Insert DPDK UIO driver into Kernel:
   modprobe uio
   insmod x86_64-native-linuxapp-gcc/kmod/igb_uio.ko

5. Bind DPDK UIO module with PCI device:
   ./usertools/dpdk-devbind.py -b igb_uio 0000:3b:00.0
   ./usertools/dpdk-devbind.py -b igb_uio 0000:3b:00.1

   Bind QAT VF's to DPDK:
   ./usertools/dpdk-devbind.py -b igb_uio 0000:60:01.0
   ./usertools/dpdk-devbind.py -b igb_uio 0000:61:01.0

6.4.3 Run VPP with OpenSSL

To run VPP with OpenSSL, on a Linux Shell, execute:
vpp -c vpp_config_hw.txt_withIPSec_withopenssl_works

6.4.4 Run VPP with Intel AES-NI (AES New Instructions set for x86 Processors)

To run VPP with AESNI, on a Linux Shell, execute:
vpp -c vpp_config_hw.txt_withIPSec_withaesni_works

6.4.5 Run VPP with QAT

To run VPP with QAT, on a Linux Shell, execute:
vpp -c vpp_config_hw.txt_withIPSec_withQAT_works

6.5 VPP Configurations For OpenSSL

```
unix {
        exec /home/dell/vpp_manual_cfgs/vpp_with_ipsec/vpp_config_with_ipsec.txt
        nodaemon
        cli-listen /run/vpp/cli.sock
        log /tmp/vpp.log
        interactive
}
cpu {
        main-core 6
        corelist-workers 2,4
}
dpdk {
        socket-mem 2048,2048
        log-level debug
        no-tx-checksum-offload
        dev default{
                num-tx-desc 1024
                num-rx-desc 1024
        }
        dev 0000:3b:00.0
        {
                workers 0
        }
        dev 0000:3b:00.1
        {
                workers 1
        }
        #dev 0000:60:01.0
        #dev 0000:61:01.0
        num-mbufs 370000
        no-multi-seg
}
ip {
        heap-size 2G
}
```

6.6 VPP Configurations For AESNI

```
unix {
        exec /home/dell/vpp_manual_cfgs/vpp_with_ipsec/vpp_config_with_ipsec.txt
        nodaemon
        cli-listen /run/vpp/cli.sock
        log /tmp/vpp.log
        interactive
}
cpu {
        main-core 6
        corelist-workers 2,4
}

dpdk {
        socket-mem 2048,2048
        log-level debug
        no-tx-checksum-offload
        dev default{
                num-tx-desc 1024
                num-rx-desc 1024
        }
        dev 0000:3b:00.0
        {
                workers 0
        }
        dev 0000:3b:00.1
        {
                workers 1
        }
        #dev 0000:60:01.0
        #dev 0000:61:01.0
        vdev crypto_aesni_mb0

        num-mbufs 370000
        no-multi-seg
}

ip {
        heap-size 2G
}
```

6.7 VPP Configurations For Intel® QAT

```
unix {
        exec /home/dell/vpp_manual_cfgs/vpp_with_ipsec/vpp_config_with_ipsec.txt
        nodaemon
        cli-listen /run/vpp/cli.sock
        log /tmp/vpp.log
        interactive
}
cpu {
        main-core 6
        corelist-workers 2,4
}

dpdk {
        socket-mem 2048,2048
        log-level debug
        no-tx-checksum-offload
        dev default{
                num-tx-desc 1024
                num-rx-desc 1024
        }
        dev 0000:3b:00.0
        {
                workers 0
        }
        dev 0000:3b:00.1
        {
                workers 1
        }
        dev 0000:60:01.0
        dev 0000:61:01.0

        num-mbufs 370000
        no-multi-seg
}

ip {
        heap-size 2G
}
```

6.8 VPP Common IPSec Configuration for OpenSSL/AESNI/Intel® QAT

```
set interface ip address TwentyFiveGigabitEthernet3b/0/1 10.10.1.1/24
set interface ip address TwentyFiveGigabitEthernet3b/0/0 10.10.2.1/24

set ip arp TwentyFiveGigabitEthernet3b/0/1 10.10.1.2 24:6e:96:9c:e5:df
set ip arp TwentyFiveGigabitEthernet3b/0/0 10.10.2.2 24:6e:96:9c:e5:de

ipsec spd add 1
ipsec spd add 2
set interface ipsec spd TwentyFiveGigabitEthernet3b/0/1 1
set interface ipsec spd TwentyFiveGigabitEthernet3b/0/0 2

set int promiscuous on TwentyFiveGigabitEthernet3b/0/1
set int promiscuous on TwentyFiveGigabitEthernet3b/0/0

ipsec sa add 10 spi 1000 esp tunnel-src 10.10.2.1 tunnel-dst 10.10.2.2 crypto-key 4339314b55523947594d6d3547666b45 crypto-alg aes-cbc-128 integ-key 4339314b55523947594d6d3547666b45 integ-alg sha1-96
ipsec policy add spd 1 outbound priority 100 action protect sa 10 remote-ip-range 16.0.0.0-16.255.255.255

ipsec sa add 11 spi 1001 esp tunnel-src 10.10.1.1 tunnel-dst 10.10.1.2 crypto-key 4339314b55523947594d6d3547666b45 crypto-alg aes-cbc-128 integ-key 4339314b55523947594d6d3547666b45 integ-alg sha1-96
ipsec policy add spd 2 outbound priority 100 action protect sa 11 remote-ip-range 48.0.0.0-48.255.255.255

ipsec policy add spd 1 outbound priority 90 protocol 50 action bypass
ipsec policy add spd 2 outbound priority 90 protocol 50 action bypass

ip route add 16.0.0.0/8 via 10.10.1.2 TwentyFiveGigabitEthernet3b/0/1
ip route add 48.0.0.0/8 via 10.10.2.2 TwentyFiveGigabitEthernet3b/0/0

set interface state TwentyFiveGigabitEthernet3b/0/1 up
set interface state TwentyFiveGigabitEthernet3b/0/0 up
```

6.9 VPP Common Settings – Huge Pages

Configure Huge Pages for VPP

Contents of file: /etc/sysctl.d/80-vpp.conf

```
# Number of 2MB hugepages desired
#vm.nr_hugepages=1024
vm.nr_hugepages=4096

# Must be greater than or equal to (2 * vm.nr_hugepages).
#vm.max_map_count=3096
vm.max_map_count=9216

# All groups allowed to access hugepages
vm.hugetlb_shm_group=0

# Shared Memory Max must be greater or equal to the total size of hugepages.
# For 2MB pages, TotalHugepageSize = vm.nr_hugepages * 2 * 1024 * 1024
# If the existing kernel.shmmax setting (cat /sys/proc/kernel/shmmax)
# is greater than the calculated TotalHugepageSize then set this parameter
# to current shmmax value.
#kernel.shmmax=2147483648
kernel.shmmax=8589934592
```

Also use:
/etc/vpp/startup.conf
No changes were made to this file.

6.10 Trex Configuration File

Contents of file: /etc/trex_cfg.yaml

```
### Config file generated by dpdk_setup_ports.py ###

- version: 2
  interfaces: ['3b:00.0', '3b:00.1']
  port_info:
      - ip: 10.10.2.2
        default_gw: 10.10.2.1
      - ip: 10.10.1.2
        default_gw: 10.10.1.1

  platform:
      master_thread_id: 0
      latency_thread_id: 1
      dual_if:
        - socket: 0
          threads: [2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46]
```

This file was generated using:

```
cd /home/dell/trex-core/scripts
./dpdk_setup_ports.py -i
```

7. Acknowledgements

This paper would not have been possible without the capable efforts of Imran Masud.

8. References

[1] https://01.org/intel-quickassist-technology

[2] https://doc.dpdk.org/guides/cryptodevs/qat.html

[3] https://software.intel.com/en-us/articles/get-started-with-ipsec-acceleration-in-the-fdio-vpp-project

[4] https://doc.dpdk.org/guides-16.04/sample_app_ug/ipsec_secgw.html

[5] https://www.ietf.org/rfc/rfc1952.txt

[6] https://fd.io/technology/

[7] https://wiki.fd.io/view/TRex

[8] https://www.dpdk.org/

[9] https://software.intel.com/en-us/articles/get-started-with-ipsec-acceleration-in-the-fdio-vpp-project

[10] https://github.com/cisco-system-traffic-generator/trex-core/blob/master/doc/trex_book.asciidoc

[11] https://wiki.fd.io/view/VPP/Command-line_Interface_(CLI)_Guide

[12] https://software.intel.com/en-us/articles/build-a-fast-network-stack-with-vpp-on-an-intel-architecture-server