Article can be accessed online at http://www.publishingindia.com

ACCELERATING OPENSSL OPERATIONS OFFLOADED TO HARDWARE CRYPTO ACCELERATOR

Nitesh Narayan Lal*, Vakul Garg**

**Software Engineer, Freescale Semiconductors India Pvt Ltd, Bengaluru, Karnataka, India.*
Email: nitesh.lal@nxp.com
***Software Architect, Freescale Semiconductors India Pvt Ltd, Bengaluru, Karnataka, India.*
Email: vakul.garg@nxp.com

Abstract *Traditionally, when an user space application tries to access hardware for performing operations, then this whole interaction is done via kernel i.e., a context switch is required from user space to kernel space. In case of cryptography based applications where large number of transactions takes place every second, this proves to be an expensive affair. But that's not the only factor which curtails the overall performance of applications/ libraries such as (OpenSSL). These applications/ libraries access the hardware crypto accelerators of asynchronous nature in a synchronous manner which further deteriorates the performance due to this discrepancy. This paper proposes a scheme to enhance the throughput of such applications by overcoming the drawbacks of existing kernel based approach of offloading crypto operations to hardware accelerators.*

Keywords: *OpenSSL, Cryptography, Security, Hardware Accelerators, User Space Driver*

Introduction

Traditionally, device driver (Agarawal & Malhotra, n.d.) runs in kernel space as handling interrupts (http://www.tldp.org) and mapping hardware resources require privileges that could be attained only after moving into kernel space.

But as every coin has got two sides, this approach also has some drawbacks. Some of them are listed as follows:

- **System call overhead:** Each call to the kernel must perform a switch from user mode to supervisor mode and then back again (http://www.linux.it). There are many applications which require several context switches thereby impacting the performance (Li, Ding, & Shen, n.d.).
- **Intricate to understand:** Kernel API's are not straightforward to understand and require certain level of adroitness.
- **Strenuous to debug** (Yehuda, n.d.): In kernel space, bugs could have severe impact on the entire system. A crashing kernel tends to have a more severe impact on the system, affecting the robustness of the whole solution.

Most of the issues with kernel-space drivers are solved by having the driver in user space but the issue with interface stability is only true for very simple user-space drivers. The advantages of user-space drivers are:

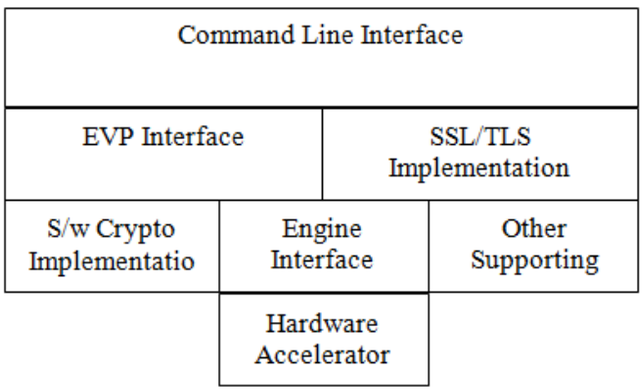
User space drivers are easy to develop and debug (Liljegren, n.d.). The programmer can run a conventional debugger on the driver code without having to go through contortions of debugging a running kernel. If a user space driver hangs, the user can simply kill it. Problems with user space drivers are unlikely to hang the entire system unless the hardware being controlled is really misbehaving.

- User memory is swappable, unlike kernel memory. An infrequently used device with a huge driver won't occupy RAM that other programs could be using, except when it is actually in use.
- Modifying a user space driver doesn't require kernel re-compilation. The code can be modified and run while the kernel is still up.
- A significant performance gain (Yehuda, Leslie, Chubb, Fitzroy-Dale, Gotz, Gray, Macpherson, Potts, Shen, Elphinstone, & Heiser, n.d.) can be achieved by overcoming the issue of frequent context switches as compared to kernel space solution.

Cryptographic operations such as symmetric key algorithms (Michael, n.d.) especially RSA are used during the initial handshake while establishing a SSL/TLS connection are CPU

intensive (Cristian, Peter, & Dan, n.d.). When processing a large number of TLS connections on a HTTPS server that handles millions of connections, the encryption/decryption being performed in the software could easily slow down the processing on the server, even affecting its core functionality. A typical architecture of OpenSSL (Satapathy, n.d.) along with the hardware crypto accelerator is shown in Fig. 1.

Fig. 1: OpenSSL Architecture



New security threats (https://www4.symantec.com) are getting added everyday to the already growing list of threats. Also, the load on the servers has tremendously increased therefore crypto operations are generally offloaded on to the hardware accelerators. In this paper we have proposed an approach by which overall throughput could be enhanced. The experiments were performed on Freescale Q or IQ platforms which have a built-in security engine hardware known as SEC (Security Engine) (http://cache.freescale.com) on the SoCs.

Literature Review

Cryptographic algorithms are performance intensive, and as more countermeasures are added to thwart increasing security attacks (listed by symantec (https://www4.symantec.com)), the space and memory requirements grow even more. For these reasons, cryptographic algorithms have traditionally been offloaded to dedicated hardware engines. As with most of the cryptographic accelerators, device driver is located within the kernel space until now. Recently there is a great shift towards moving the device driver to the userspace due to several advantages such as robust and flexible process management, standardised system-call interface, simpler resource management, a large number of libraries for XML, and regular expression parsing and others which are explained in detail in paper on device drivers in user space (Agarawal & Malhotra, n.d.). Since all the interrupt handling is done within the kernel, Linux provides a standard UIO (user I/O) framework for developing user-space-based device drivers for this purpose. The UIO framework defines a small kernel space component that primarily performs device memory region mapping and interrupt indication to the userspace. This particular concept is explained in detail in the chapter Interrupts and interrupt handling (http://www.tldp.org).

Whenever we talk about performance, context switch is one of the key parameter to be considered. When device driver is placed in the kernel space and is accessed from the user space, several context switches are involved causing unnecessary overheads resulting in performance degradation. A detailed introspection of the same is covered in Kernel system calls (http://www.linux.it) and Quantifying the Cost of Context Switch (Li, Ding, & Shen, n.d.). Keeping the driver in user space provides additional advantages from developers perspective such as ease of debugging, least system impact due to crash, easy to understand API's and others as explained by Mats Liljegren in his work on user v space drivers in Linux (Liljegren, n.d.). Some of the researchers from Australia and Germany have also shown (in their work on User-level Device Drivers: Achieved Performance (Yehuda *et al.*, n.d.)) that one of the primary concern related to the userspace driver of poor I/O performance can also be mitigated.

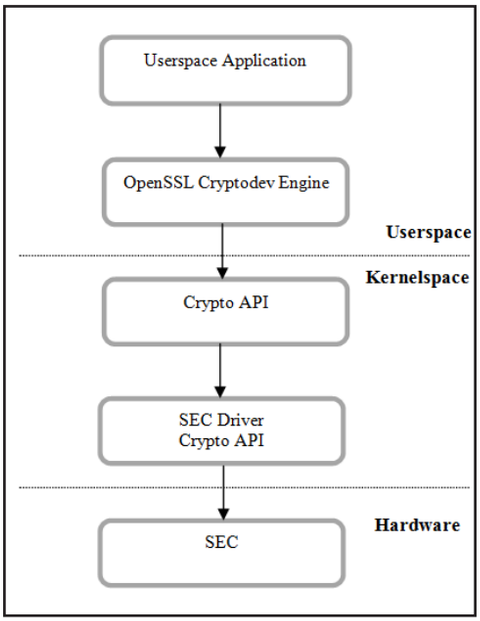
For our demonstration we have tried to accelerate the OpenSSL crypto operations which are offloaded to Freescale's security engine (http://www.nxp.com) whose driver resides within the kernel space. The OpenSSL (Architectural analysis of OpenSSL crypto algorithms is comprehensively explained in the paper (Satapathy, n.d.)) project provides an open source implementation of the SSL/TLS protocols and is a commonly deployed library for SSL/TLS world-wide. The SSL/TLS protocols consist of two phases: an initial session-initiation/handshake and a bulk data transfer. These operations are one of the most CPU intensive operations and their detailed performance analysis with respect to the webservers is been done by Cristian, Peter, and Dan (n.d.) in their work on Performance analysis of TLS web servers . Today when millions of secure transactions are processed every second, we simply cannot overlook the importance of performance just for the sake of security.

Existing Implementation

The existing implementation is diagrammatically shown in Fig. 2. In the existing implementation, all the crypto related calls from userspace using OpenSSL APIs are trapped into the cryptodev engine (http://openssl.com) in the user space. Cryptodev's kernel driver receives these requests from the engine, creates a request for SEC and sends it to the SEC driver (CAAM) which later passes them on to the hardware after required restructuring. During this time no other request can be submitted to SEC through this application context. Until SEC processes the request and sends a response, the context sleeps. As the application context is simply waiting for the response, the time taken by SEC for processing the request is referred to as idle time for the application context. In this analysis it was observed that kernel doesn't utilize this time slice efficiently. The method of calculation of this

idle time is shown under the PERFORMANCE ANALYSIS & RESULTS section.

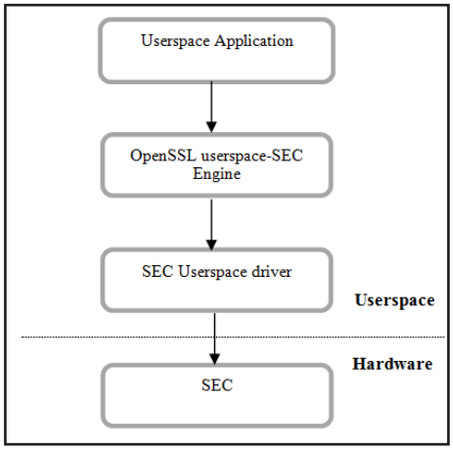
Fig. 2: Existing Implementation Block Diagram



Proposed Solution

In the proposed scheme, all the components from kernel space are removed and hardware is directly accessed from the user space. This approach is shown diagrammatically in Fig. 3.

Fig. 3: Proposed Implementation Block Diagram



Moving the SEC driver to user space reduces the unnecessary context switches which take place from user space to the kernel space every time a packet needs to be sent to SEC. It also provides the flexibility to interleave (https://www.cs.sfu.ca) any other application context such as networking in the idle time to efficiently utilise the bandwidth in the use case. The results and analysis done in this paper are without interleaving any other context and shows the gain just by bringing up the kernel components to the user space.

Test Setup

To verify the performance improvement with the proposed solution, an experiment is been orchestrated in the following manner:

- Algorithm Sampled- AES-CBC (https://townsendsecurity.com) AES stands for Advanced Encryption Standard, a symmetric 128-bit (http://www.axantum.com) block data encryption technique
- Moving to the next step involved lots of scuffling and endeavor. The proposed solution uses an OpenSSL engine which uses the USDPAA API to interact with SEC directly from user space without switching to the kernel. Performance comparison required the following software and hardware component:
  - Hardware
    1. Freescale QorIQ platform-T4240QDS (http://www.nxp.com)
  - Software
    1. *AES_MULTI*: Multi-threaded application to exercise SEC by sending multiple input requests for AES_CBC operation through kernel. It also shows the effective throughput for the specified input size.
    2. *USSEC*: Multi-threaded application which directly accesses SEC to send multiple input requests for AES_CBC operation without switching into kernel As part of result, it shows the throughput for the specified input size.
    3. *OpenSSL engine* (Vasut, n.d.): Engine contains the piece of code required for offloading the crypto operations on to specific hardware attached to the host machine which is running or using OpenSSL libraries.

Performance Analysis and Results

Calculating Idle Time

To calculate the idle time an elementary technique is used where a request to SEC via kernel driver of size 'x' is enquired .' t1' is the time at which the packet is submitted to SEC and t2 is the time at which SEC responds with the output. Now for one packet the idle time will be simply (t2-t1).

To generalize the results, multiple packets are sent consecutively to SEC and then the average time taken is evaluated by using the following formula:

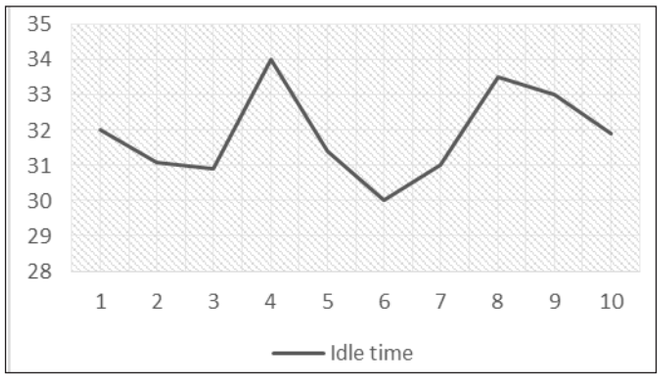
If 'n' is the number of requests sent to SEC in a second

Idle time per second = $\sum$ (t2 (i)-t1 (i))/n

where' i' runs from 1 to n.

The observation is shown in Fig.4 where the y-axis represents the idle time in terms of percentage time consumed for complete processing of a packet and the x axis shows the time slices in which the idle time is calculated; here it is 10 seconds.

Fig. 4: Idle Time Chart Across 10 Seconds



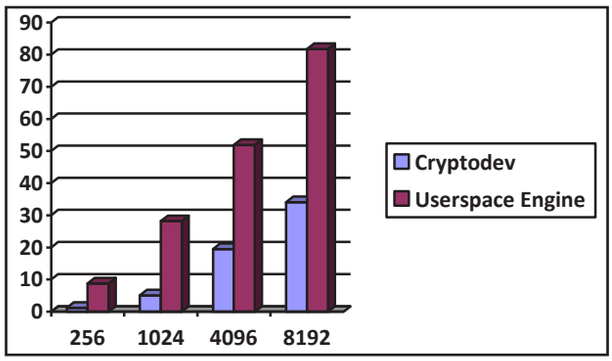
The above experiment shows that the idle time for AES-CBC operations comes out to be around 31-32% on an average.

Comparing the Performance Gain

As it was anticipated, a remarkable gain in the overall throughput after applying the proposed approach on a single core system is noticed. The results of the observation are shown in Fig. 5.

(The graph consists of throughput values in MB/s on the y-axis and chunk size in bytes on the x-axis).

Fig. 5: Performance Comparison on Single Core



Conclusion

From the above results it is quite evident that moving kernel space driver to user space shows significant improvement in the performance.

References

Introduction to AES Encryption by Townsend Security. (2016). Retrieved from https://townsendsecurity.com/sites/default/files/AES_Introduction.pdf

Agarawal, H., & Malhotra, R. (2014). Device drivers in user space. Retrieved from http://www.embedded.com/print/4401769

http://www.embedded.com/design operating -systems/ 4401769 /Device-drivers-in-user-space

Cristian, C., Peter, D., & Dan, S. W. (1996). *Performance analysis of TLS Web Servers*. Retrieved from http://www.cs.rice.edu/~dwallach/ pub/tls-tocs.pdf

Internet security threat report (2015). Retrieved from https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347932_GA-internet-security-threat-report-volume-20-2015-social_v2.pdf

Interrupts and Interrupt Handling. Retrieved from http://www.tldp.org/LDP/ tlk/dd/interrupts.html Kernel System Calls - http://www.linux.it/~rubini/docs/ ksys/ksys.html

Li, C., Ding, C., & Shen, K. (2007). *Quantifying the Cost of Context Switch.* Retrieved from http://www.cs.rochester.edu/u/cli/research/switch.pdf

Liljegren, M. (2014). User-Space Device Drivers in Linux: A First Look. Retrieved from http://www.enea.com/Documents/Resources/Whitepapers/Enea-User-Space-Drivers-in-Linux_Whitepaper_2013.pdf

Michael, H. (1993). Symmetric Key Cryptography. Retrieved from http://www.doc.ic.ac.uk/~mrh/430/03.SymmetricKey.ppt.pdf

Satapathy, P. R. (n.d.). Architectural Analysis of OpenSSL Crypto Algorithms for Network Processor. Retrieved from http://alumni.cs.ucr.edu/~piyush/cs213Report.pdf

Seagate- 128-Bit Versus 256-Bit AES Encryption. (2008). Retrieved from http://www.axantum.com/AxCrypt/etc/seagate128vs256.pdf

Vasut, M. (2014). Utilizing the crypto accelerator. Retrieved from http://events.linuxfoundation.org/sites/events/files/slides/ lcj-2014-crypto-user.pdf

Yehuda, M. (2003). Linux Kernel Debugging. Retrieved from http://www. haifux.org/lectures/63/kernel_oopsing.pdf

Yehuda, M., Leslie, B., Chubb, P., Fitzroy-Dale, N., Gotz, S., Gray, C., Macpherson, L., Potts, D., Shen, Y., Elphinstone, K., & Heiser, G. (2010). User-level Device Drivers: Achieved Performance. Retrieved from https://ssrg.nicta.com.au/publications/papers/Leslie_CFGGMPSEH_05-tr.pdf

http://cache.freescale.com/files/32bit/doc/app_note/AN3085.pdf

"http://openssl.com/testing/validation-2.0/platforms/hwaccel/kernel_crypto.pdf.cs.sfu.ca/~fedorova/Teaching/CMPT886/ Spring2007/papers/laudon-interleaving.pdf

http://www.nxp.com/files/32bit/doc/fact_sheet/T4240QDSFS.pdf