

IMPROVING

High-Bandwidth TLS in the FreeBSD Kernel

In our 2015 paper, “Optimizing TLS for High-Bandwidth Applications in FreeBSD,” we demonstrated the cost of TLS encryption on high-bandwidth video serving on Netflix’s OpenConnect Appliance (OCA) [1] and explored methods for reducing that cost via in-kernel encryption. The results showed that such a concept is feasible, but only a small performance gain was achieved. In this article we describe the improvements made since then to reduce the cost of encryption. We also compare the performance of several different bulk encryption implementations.

INTRODUCTION

The importance of Transport Layer Security (TLS) [2] continues to grow as businesses and their customers come to appreciate the value of communication privacy. Netflix announced in 2015 that it would start on the path to encrypting the video and audio playback sessions of its streaming service in an attempt to help users protect their viewing privacy [3]. Enabling this comes with a significant computational cost to the OpenConnect serving platform, so work continues in exploring and implementing new ways to optimize TLS bulk encryption and thus lower capital and operational costs.

An OCA is a FreeBSD-based appliance that serves movies and television programming to Netflix subscribers. Confidential customer-data-like payment information, account authentication, and search queries are exchanged via an encrypted TLS session between the client and the various application servers that make up the Netflix infrastructure. The audio and video objects are statically encrypted by Digital Rights Management (DRM) that is pre-encoded into the objects prior to them being distributed to the OCA network for serving.

The Netflix OpenConnect Appliance is a server-class computer based on an Intel 64-bit Xeon CPU and running FreeBSD and Nginx 1.9. The platform evolves yearly as commodity components increase in capability and decrease in price; the most recent generation of the platform holds between 10TB and 200TB of multimedia objects, and can accommodate anywhere from 10,000 to 40,000 simultaneous long-lived TCP sessions with customer client systems. The servers are also designed to deliver between 10Gbps and 40Gbps of continuous bandwidth utilization.

BY
RANDALL
STEWART &
SCOTT LONG

Communication with the client is over the HTTP protocol, making the system essentially into a large static-content web server.

Until 2015, these audio and video objects were not encrypted on a per-session basis. Netflix is now in the process of revising and releasing its client playback software to include support for TLS playback encryption, with the goal of updating all playback devices that are both capable of being upgraded and have the computational capacity to support TLS decryption. By the end of 2016 we expect the majority of streaming sessions to be using TLS encryption.

As the number of client sessions doing TLS grows across the OpenConnect network, demand increases on the server side to accommodate these sessions. Building on the results from our work in 2015 [4], we looked for ways to reduce the cost of encryption on our deployed fleet of hardware as well as reduce the number of servers needed to accommodate future growth. This investigation covered three areas: what is the ideal cipher for bulk encryption, what is the best implementation of the chosen cipher, and are there ways to improve the data path to and from that cipher implementation?

BULK CIPHER SELECTION

The Cipher Block Chaining (CBC) is commonly used to implement bulk data encryption for TLS sessions as it is well studied and relatively easy to implement. However, it comes with a high computational cost as the plaintext data must be processed twice, once to generate the encrypted output, and once to generate the SHA hash that verifies the integrity of the encrypted data. The AES-GCM cipher, based on Galois/Counter Mode (GCM) [5], provides adequate data protection and does not require that the plaintext be processed twice. It is included in TLS 1.2 and later, and is available in all modern versions of OpenSSL and its derivatives. Decryption is also computationally cheap, and this combined with ubiquitous availability makes it attractive to both client and server platforms.

We decided that we would transition our TLS platform to prefer GCM, and fall back to CBC only for primitive clients that couldn't be upgraded to support GCM. We estimate that once TLS is rolled out to our entire network only a small percentage of playback sessions will use CBC.

CIPHER IMPLEMENTATION

Synthetic performance testing of the OpenCrypto Framework AES-CBC cipher showed it to be less performant than the equivalent implementation from OpenSSL. We also needed to investigate AES-GCM performance, but found that OpenSSL 1.0.1 as of early 2015 did not have an AESNI-optimized implementation for it. Our search for alternatives led us to BoringSSL [6], which had a well-performing AESNI implementation of AES-GCM.

In mid-2015 we were introduced to the Intel Intelligent Storage Acceleration Library (ISA-L) [7]. It provided an implementation of AES-GCM that was hand-tuned for specific Intel model families and their instruction sets. Testing showed it to be an improvement over the BoringSSL ciphers. Results are included below.

One drawback to the ISA-L was that it was written in the YASM dialect, which is not directly compatible with the GCC and LLVM toolchain assemblers in FreeBSD. That required us to modify the kernel makefile infrastructure in a rudimentary way to allow YASM to be called as an external assembler on the needed source files, as so:

```
compile-with "/usr/local/bin/yasm -g dwarf2  
-f elf64 ${INTELISAINCLUDES} -o ${.TARGET}  
${S}/contrib/intel_isa/aes/${.PREFIX}.asm"
```

DATA PATH IMPROVEMENTS

Our initial work in 2015 [4] used the AESNI implementation built into FreeBSD and the Open Crypto Framework (OCF) to perform bulk encryption. The results in our previous paper showed a small improvement in performance, but not nearly the results we had hoped to gain. We knew of several areas where our results could be improved, including:

- Extra copies were being made during kernel data processing due to the encrypt in-place requirement of our AESNI implementation.
- The nginx calls into the TLS code were not passing in correct flags with the `sendfile(2)` call. This meant that hot content was not being properly cached.
- Many times during processing an mbuf chain was walked to gain addresses for encryption; this constant walking of the mbuf linked lists caused added overhead and further polluted the CPU cache.

We decided to pass in to our new ciphers an

array of pointers to encrypt from and to, i.e., an `iovec`. This `iovec` array would be filled in during the initial setup of the `sendfile` call, as each page was set up for I/O, thus eliminating the need for traversing a linked list of `mbufs`. We also redesigned the `mbuf` allocation routines to have the ability, as allocation occurred, to include this new “`mbuf map`.”

Since a large part of our data was being encrypted, we also designed a new special `mbuf` zone that required less overhead during allocation. A typical one-page `mbuf` required three separate allocations (one for the `mbuf`, one for the reference count, and one for the page). We redesigned this to make the page and the `mbuf` an indivisible unit where FreeBSD’s `UMA` would allocate a page and `mbuf` together during the `UMA`’s initialization routine and the `UMA` constructor would only be used to reset pointers within the tied entity. We also embedded the reference count within the `mbuf`. This required some small tricks with copies (we don’t actually free the original `mbuf` until all copies are free), but proved quite effective at reducing `mbuf` overhead.

Switching to the `iovec` array forced us to abandon the `OpenCrypto` Framework API and access the cipher routines directly. We still wanted to be able to fall back to `OpenCrypto` for testing purposes, so we created a layer that abstracts the memory allocation and `iovec` handling for low-level cipher access while still allowing interoperability with `OpenCrypto`. The translation is transparent to the upper layers and is selectable at runtime. This work also gave us the chance to find and fix codepaths that were making unnecessary copies of data. We also fixed the incorrect `sendfile` flag usage.

RESULTS

After adding all the improvements, we deployed our new firmware on three different machines. These machines were

fed live traffic while gathering CPU and bandwidth measurements during busy hours. The same software was used in all measurements, the only difference being changes to the configuration so that the software would:

- Disable all `sendfile` enhancements and use just `OpenSSL`, reading from the file and writing the encrypted data down the TCP connection.
- Use the `sendfile` enhancement with the encryption set to use `boringSSL`.
- Use the `sendfile` enhancement with the encryption set to use Intel’s `ISA` library.

Thus each machine provided us with three sets of results. The machine types were as follows:

- Rev H storage (HDD) platform, CPU E5-2650Lv2 at 1.7Ghz with 20 cores (Hyperthreaded) the `cpu` class being an Ivy Bridge Xeon.
- Rev F Flash (SSD) cache platform, CPU E5-2697v2 at 2.7Ghz with 24 cores (Hyperthreaded) the `cpu` class being an Ivy Bridge Xeon.
- Rev N Flash (SSD) cache platform, CPU E5-2697v3 at 2.6Ghz with 28 cores (Hyperthreaded) the `cpu` class being a Haswell Xeon.

Each set of results will be labeled Rev H, Rev F, or Rev N with the test undergone. We show approximately one hour of traffic during a busy period. For visual clarity, the legends have been removed from the graphs; the green `x` plots are bandwidth in Gbps, and the red `+` plots are CPU system usage percentage.

We see in Figure 1 what happens when only `OpenSSL` is used. The CPU limits we have set are the standard 80%; however, the storage cache is disk bound hitting between 60% and 65% CPU and topping our performance out at about 12–12.5Gps of serving traffic. The `sendfile` feature adds considerable improvement as we see in the next two figures.

The results of `BoringSSL`, in kernel using `sendfile`, are seen in Figure 2. The CPUs tend to be used a bit more (55%–70% CPU utilization)

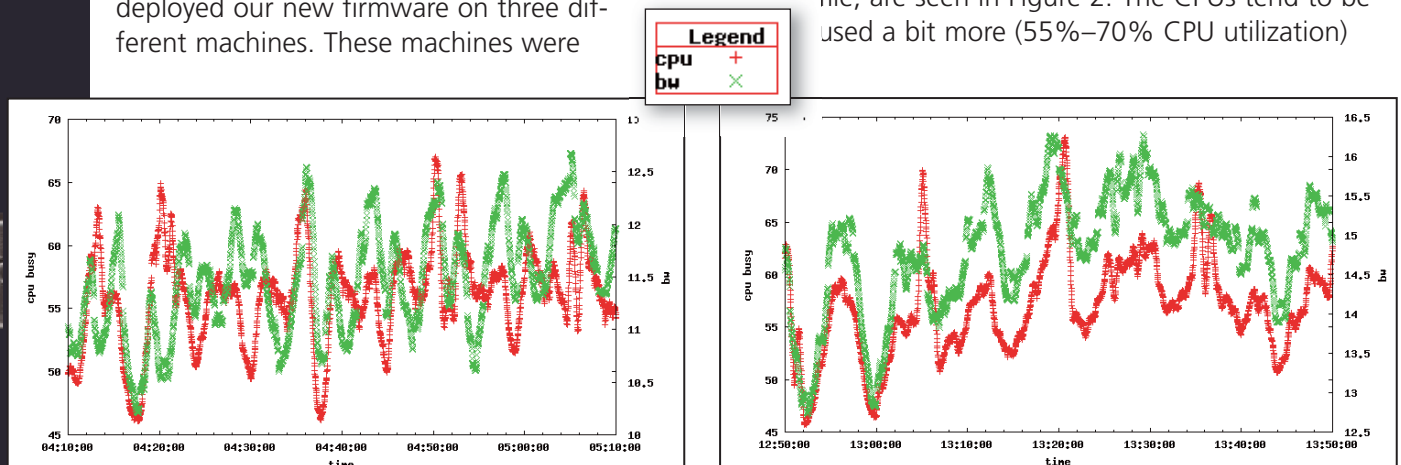


Fig. 1. OCA Rev H Performance using user space `OpenSSL`

Fig. 2. OCA Rev H Performance using in kernel `BoringSSL`

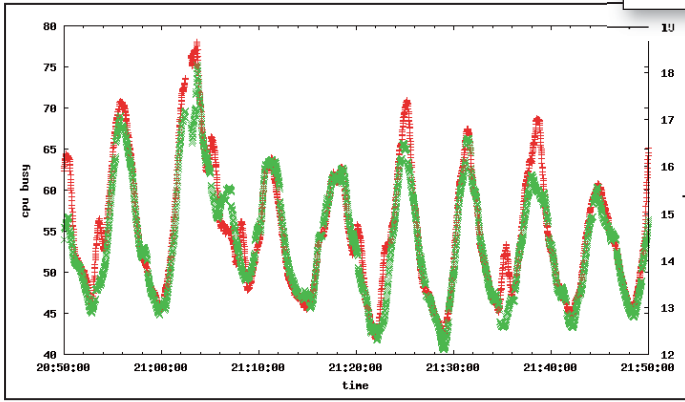
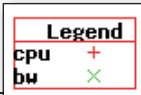


Fig. 3. OCA Rev H Performance using in kernel ISA\ in kernel ISA

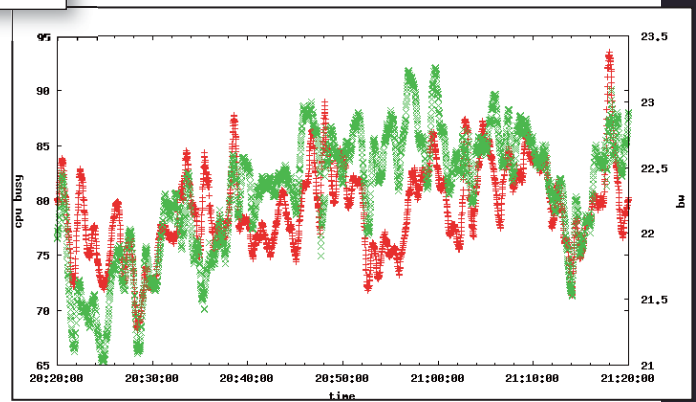


Fig. 4. OCA Rev F Performance using user space OpenSSL

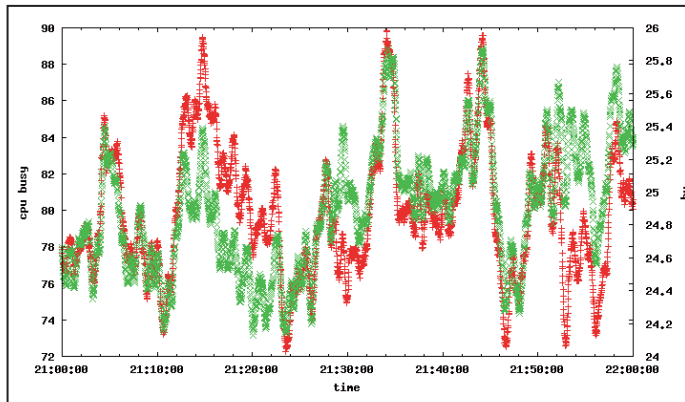


Fig. 5. OCA Rev F Performance using in kernel BoringSSL

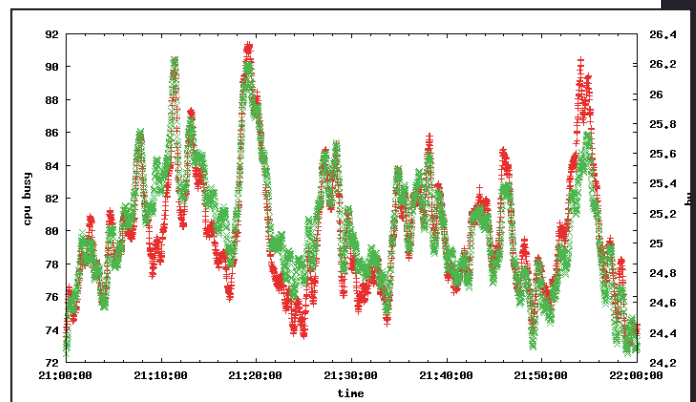


Fig. 6. OCA Rev F Performance using in kernel ISA

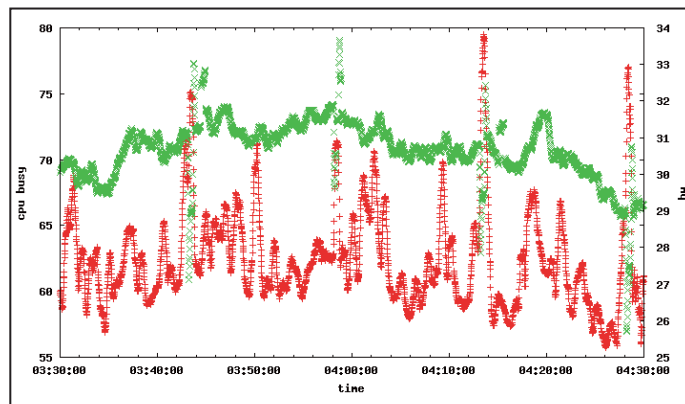


Fig. 7. OCA Rev N Performance using user space OpenSSL

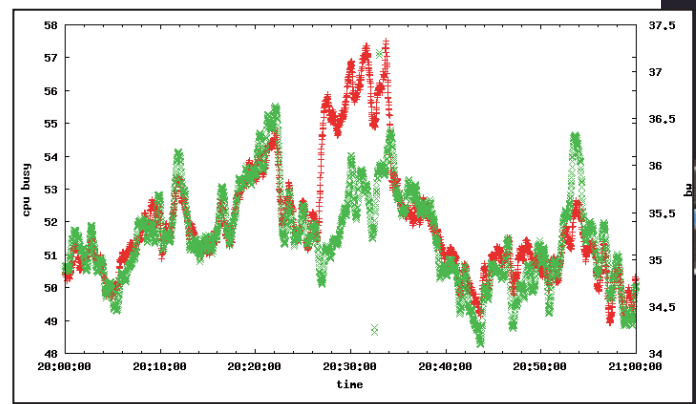


Fig. 8. OCA Rev N Performance using in kernel BoringSSL

with overall output performance increasing to 15–16Gps. This is what we expect with use of the sendfile call to help streamline the I/O.

For final comparison, we put in place the ISA library, again in kernel with sendfile. Results can be seen in Figure 3 and show another improvement moving us to as much as 18G but generally holding around 16–16.5G.

In Figure 4 we see OpenSSL, this time hitting maximum CPU. This is because SSDs have a significantly greater I/O capacity and so we no longer hit the disk limits seen in the Rev H. We see that running with an average of 80% CPU we maintain somewhere between 22 to 23Gbps. This gives us

our baseline to compare any improvements.

In Figure 5 we see the results of using the kernel encryption with sendfile and BoringSSL. Here we are able to maintain between 25 and 25.5Gbps while maintaining our goal of 80% CPU utilization.

The ISA library shown in Figure 6 gives us a slight improvement over the previous results getting us again around 25–25.5G, but tending to stay toward the higher end of the range.

Both the Rev F and Rev H are Ivy Bridge machines (v2 CPUs). We anticipate better performance out of a Haswell machine (v3 CPU). Our Rev N shows promising results in the next set of figures.

Interestingly, the OpenSSL results seen in Figure 7

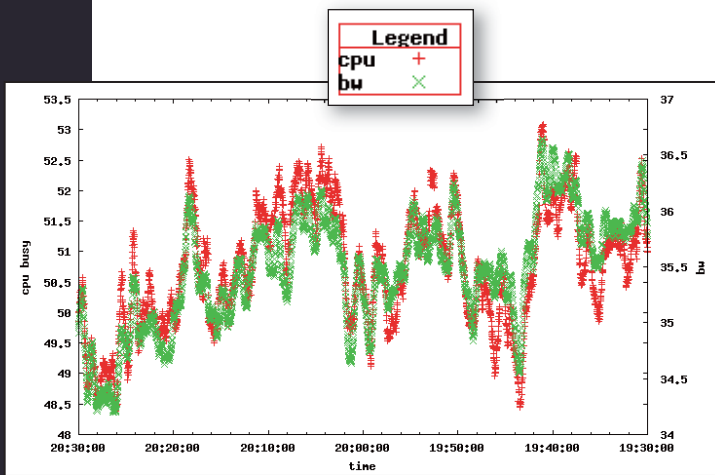


Fig. 9. OCA Rev N Performance using in kernel ISA

do not reach the full CPU target of 80%. Checking the machine health statistics, we found that the SSDs had reached their maximum at around 29–30Gbps that we see maintained in the graph.

In Figure 8 we see us reach the interface’s maximum capacity of 35.5–36Gbps, with the CPU tending to stay around 53% with a burst up to 57% at one point.

The ISA library results can be seen in Figure 9 and show similar results to what we see in the BoringSSL case with the exception that our CPU use is tending to stay toward 50.5%. The results are tabulated in Table I.

We were also forced to set aside the OpenCrypto Framework in order to achieve certain optimization. Similar optimizations might be useful for other crypto consumers in the kernel, so we will explore ways to extend the OCF API as we work to move the code upstream to FreeBSD.

Encrypting in software still requires that data pages get touched by the CPU, causing CPU cache pollution. The data is almost never reused, so the caching is wasted and needlessly evicts other system data that could be reused. We are currently investigating fine-grained use of cache-control features of the Intel CPUs to limit the amount of data that is put into the last-layer caches during encryption. We are also working with the Intel ISA-L team to develop routines that use uncached load/store assembly opcodes for data movement through the cipher.

ACKNOWLEDGMENTS

The authors would like to thank the Intel ISA-L team for reaching out to us to assist in achieving our performance goals, and David Wolfskill and Warner Losh for their support and reviews. We would also like to thank the entire Netflix OpenConnect team for providing an outstanding environment for this work.

ADDENDUM

Shortly after this paper was originally published at AsiaBSDCon in March 2016, Intel publically released the E5-v4 (Broadwell Xeon) series of CPUs. We were already evaluating prerelease samples of these

CPUs under our TLS workload, but could not include our results in the paper at that time due to the pre-release nature of the hardware. We also started experimenting with the Mellanox ConnectX-4 100Gb Ethernet adapter as a replacement for our 40Gb adapters, and HGST SN100 NVMe drives as a replacement for our SATA SSD storage.

The 2697-v4 CPUs bumped up the CPU core count to 16, an increase of 2 cores from the v3 predecessor. More importantly, the v4 family added a feature called Cache Allocation Technology (CAT) that allows the OS to exercise fine-grained control over the retention of data pages in the L2 and L3 caches. This was attractive since most of the data moving through the CPU

	RevF		RevH		RevN	
	cpu%	BW	cpu%	BW	cpu%	BW
Baseline						
BoringSSL	60–65	12–12.5	80	22–23	70–75	29–30
ISA-L	55–70	15–16	80	25–25.5	53	35.5–36
	55–70	15–16.5	80	25.5	50.5	35.5–36

Table I. CIPHER COMPARISON CHART

SUMMARY AND FUTURE DIRECTIONS

With our latest work, overall performance has improved as much as 30%, a vast improvement from our original results. Still left unexplored is use of offload cards to assist in our efforts to encrypt all data. One question we have is whether a card can be used in our hardware design in such a way that it takes less CPU and PCIe bandwidth than just running the AESNI instructions themselves. As new generations of Intel CPUs become available, it is possible that the cost of feeding data to an auxiliary card and collecting the encrypted results will be more than the actual AESNI instructions themselves.

under our workload is TLS data that is never directly inspected by the CPU once the encryption phase is over and usually never gets reused. Marking this data as non-cachable meant that the limited cache resources could be used for OS data and control structures instead of throw-away TLS data. Supporting CAT policy required only straight-forward code modifications to the TLS socket layer in the kernel. The results are shown in Table II.

These results build on the data from Table I, with the notable difference being that thanks to moving to a 100Gb network interface, the CPU was now the limiting factor on performance, not network port bandwidth. An important data point not shown in the table is that the v4 CPU was also tested in the baseline configuration with userland OpenSSL and no kernel data path. In this configuration, only 26Gbps was achieved before maxing out the CPU. Thus, the difference between the baseline and optimized configuration represents a

	Without SSL		With SSL		With SSL + CAT	
	Gbps	cpu%	Gbps	cpu%	Gbps	cpu%
E5-2697-v3	89	80	55	80	–	–
E5-2697-v4	93	80	58	80	62	80

Table II. NETFLIX VIDEO STREAMING

136% increase in performance on the v4 CPU and a 106% improvement compared to the v3 CPU at baseline. We attribute the lower gain relative to the v3 to the extra CPU cores creating additional resource and cache contention.

Our results show that it is feasible to design and deploy a system running FreeBSD that can fill 2x40Gbps or 1x100Gbps network attachments to a reasonable capacity with 100% TLS traffic using modest hardware components. Performance research continued in 2016, and our results from that will be shared in a future paper. ●

REFERENCES

- [1] Netflix, "Netflix Open Connect," <https://openconnect.netflix.com/en/>. (2016)
- [2] T. Dierks, E. Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246. (August 2008)
- [3] M. Watson. "HTTPS performance experiments for large scale content distribution," <https://lists.w3.org/Archives/Public/www-tag/2015Apr/0027.html>. (April 2015)
- [4] R. Stewart, J. M. Gurney, S. Long. "Optimizing TLS for High-Bandwidth Applications in FreeBSD," https://people.freebsd.org/~rrs/asiabsd_2015_tls.pdf. (March 2015)
- [5] D. A. McGrew, J. Viega. "The Galois/Counter Mode of Operation (GCM)," <http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf>. (2005)
- [6] Google, "BoringSSL," <https://boringssl.googlesource.com/boringssl/>. (2016)
- [7] Intel, "Optimizing Storage Solutions Using the Intel Intelligent Storage Accelerations Library," <https://software.intel.com/en-us/articles/optimizing-storage-solutions-using-the-intel-intelligent-storage-acceleration-library/>. (2015)

RANDALL STEWART currently works for Netflix Inc. as a Senior Software Engineer. His current duties include optimizing and enhancing FreeBSD's network stack within the Netflix Open Connect Appliance. Previously Stewart was a Distinguished Engineer at Cisco Systems. In other lives he has also worked for Adara, Motorola, NYNEX S&T, Nortel and AT&T Communication. Throughout his career he has focused on operating system development, fault tolerance, and call control signaling protocols. Stewart is also a FreeBSD committer having responsibility for the SCTP reference implementation within FreeBSD as well as working with FreeBSD's tcp stack to enhance and extend it.

SCOTT LONG has been a Senior Software Engineer at Netflix Inc. since 2011, focusing on FreeBSD performance and system architecture. Previously he was at Yahoo, Adaptec, and a handful of tech startups. Long has been a FreeBSD committer since 2000 and a user since 1993. Outside of work, he enjoys aviation, hiking, and skiing with his family in Colorado.

