Accelerating SSL with GPUs

Keon Jang* Sangjin Han* Seungyeop Han† Sue Moon* KyoungSoo Park**

*Department of Comptuter Science, KAIST, Korea
{keonjang, sangjin}@an.kaist.ac.kr, sbmoon@kaist.edu

†NHN Corporation, Korea
haneul0318@gmail.com

**Department of Electrical Engineering, KAIST, Korea
kyoungsoo@ee.kaist.ac.kr

ABSTRACT

SSL/TLS is a standard protocol for secure Internet communication. Despite its great success, today's SSL deployment is largely limited to security-critical domains. The low adoption rate of SSL is mainly due to high computation overhead on the server side.

In this paper, we propose Graphics Processing Units (GPUs) as a new source of computing power to reduce the server-side overhead. We have designed and implemented an SSL proxy that opportunistically offloads cryptographic operations to GPUs. The evaluation results show that our GPU implementation of cryptographic operations, RSA, AES, and HMAC-SHA1, achieves high throughput while keeping the latency low. The SSL proxy significantly boosts the throughput of SSL transactions, handling 25.8K SSL transactions per second, and has comparable response time even when overloaded.

Categories and Subject Descriptors

C.2.0 [**General**]: Security and protection; C.2.1 [**Network Architecture and Design**]: Network communications

General Terms

Design, experimentation, performance

Keywords

SSL, CUDA, GPU

1. INTRODUCTION

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) have served as a secure communication channel in the Internet for the past 15 years. SSL provides secure communication against eavesdropping, and enables authentication of end hosts. Nowadays, SSL plays an essential role in online-banking, e-commerce, and other Internet services to protect passwords, credit card numbers, social security numbers and other private information. Despite its great success, today's SSL deployment is limited to security-critical domains. The number of SSL-enabled websites is slightly over one million [1], which reflects only about 0.5% of 200 million active Internet domains [2].

The primary reason behind the low SSL adoption lies in high computation overhead in the server side. For each SSL connection, the server has to perform key exchange that involves expensive public-key cryptography. Public-key decryption quickly becomes the bottleneck when a large number of connections have to be established at the server side [3, 4]. For instance, even the state-of-the-art CPU core can handle only about $1,000$ HTTPS transactions per second (TPS) with 1024-bit RSA while the same core can serve over $10,000$ plain HTTP TPS.

To scale the server-side SSL computation, we propose Graphics Processing Units (GPUs) as a new source of computing power to offload cryptographic operations from CPU. While GPUs have demonstrated significant throughput improvement in cryptographic operations [5, 6], high latency remains as a challenge and is not yet practical in interactive environments [6].

In this work we implement an SSL-accelerator with GPU working as a proxy for web servers, and show that GPUs accelerate SSL with small latency overhead while significantly boosting the throughput at the same time. To achieve low latency and high throughput, we design and implement opportunistic offloading algorithms to balance the load between GPU and CPU in SSL processing. For light workload, we use CPU for low latency, and, when the load increases, we offload cryptographic operations to GPUs to improve throughput with GPU's massive parallelism.

2. A CASE FOR GPU-ACCELERATED SSL

SSL uses three different algorithms for secure communication. At the beginning, a client and a server exchange randomly generated secrets using an asymmetric cipher. Asymmetric ciphers (e.g., RSA, DSA, or DHE) guarantee the secure exchange of the secrets even when an attacker is eavesdropping. After the exchange of secrets, the server and the client generate session keys for symmetric ciphers (e.g., AES, DES, or RC4) and authentication algorithms (e.g., SHA1 or MD5). Symmetric ciphers are used to encrypt the data, and authentication algorithms are used to prevent tampering.

For small SSL transactions, the asymmetric cipher takes up most of the computation, and for large transactions, the symmetric cipher and authentication algorithm consume the most computation. We implement cryptographic operations in GPU and evaluate the effectiveness for SSL acceleration. We choose RSA, AES128-CBC (Cipher-Block Chaining), and HMAC-SHA1 (Hash-based Message Authentication Code), one of the most popular cipher suites in SSL.

GPUs achieve high computing power with hundreds of processing cores and large memory bandwidth, being able to run thousands of hardware threads in parallel. At the coarse-grained level each SSL transaction can run in parallel. At the fine-grained level the cipher algorithm itself may process input data in parallel.

RSA operations rely heavily on the large integer arithmetic such as 1024-bit integer multiplication. Large integer arithmetic is divided into series of native integer operations and executed serially in CPU. In GPU, each independent operation can run in parallel.

SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.
ACM 978-1-4503-0201-2/10/08.

	1024-bit	2048-bit	4096-bit
GPU thruput	66,970 msg/s	9,995 msg/s	1,348 msg/s
CPU thruput	7,268 msg/s	1,160 msg/s	164 msg/s

(a) RSA throughput

	AES-ENC	AES-DEC	HMAC-SHA1
GPU thruput	9,254 Mbps	9,342 Mbps	27,893 Mbps
CPU thruput	4,620 Mbps	4,620 Mbps	10,429 Mbps

(b) AES 128-bit CBC and HMAC SHA1 throughput

Table 1: Performance comparison of RSA, AES, and HMAC-SHA1 on NVIDIA GTX480 and Intel Xeon X5550 (with all four cores)

We implement a parallel algorithm with O(k) time complexity for multiplication of two k-bit integers.

In AES128-CBC encryption, there is no parallelism in the algorithm itself as encryption of each AES block of 128 bits or 16Bdepends on the previous block's encrypted results. On the other hand, AES-128CBC decryption can be parallelized at the block level. HMAC-SHA1 algorithm cannot be parallelized at the block level due to data dependency between blocks and we parallelize HMAC-SHA1 at the SSL record level. We omit further implementation details due to space limitation.

In Table 1, we show the performance of our GPU implementation as well as CPU performance measured with `OpenSSL 1.0.0`. We use NVIDIA GTX480 and Intel Xeon X5550 (quad-core 2.66Ghz). For AES and HMAC evaluation we used 16KB flows. We vary the number of concurrent flows and the batch size for each execution in GPU and show the peak throughput.

The RSA throughput is above 66K msg/s with GTX480. CPU performs about 7,300 msg/s with all four cores, and our GPU implementation is more than 9 times faster than CPU. Even at the peak throughput, latency is under 23 ms. The latency is two orders of magnitude lower than previous work [6] at peak.

GTX480 achieves about 9.2 Gbps for AES encryption and decryption, and 28 Gbps for HMAC-SHA1, faster than CPU by more than a factor of two. The peak throughput is reached at the latency under 60 ms, 4 ms, and 10 ms, respectively; reasonable for interactive applications. AES encryption takes much longer than decryption due to lack of parallelism as explained earlier. Since AES variations such as AES-CTR (Counter Mode) or AES-GCM (Galois/Counter Mode) are parallelizable at the block level, we believe we can reduce the latency farther than our AES128-CBC. We are currently implementing AES-GCM.

3. BASIC DESIGN AND IMPLEMENTATION

Based on our cryptographic algorithms on GPU, we implement an SSL proxy. We choose to implement the SSL proxy instead of a GPU-based crypto library or an OpenSSL engine for two reasons: (*i*) ease of integration with existing servers and (*ii*) support for event-driven applications blocking on cryptographic operations.

We design a simple and novel opportunistic offloading algorithm to achieve high throughput while keeping the latency low. The key idea is to send all requests to CPU when the number of pending cryptographic operations is small enough to be handled by CPU. If requests begin to pile up in the queue, then the algorithm offloads cryptographic operations to GPUs and benefits from parallel execution for high throughput.

4. PRELIMINARY RESULTS

We show preliminary results from our SSL proxy implementation. We run experiments with a dual Intel X5550 (total of 8 cores) system with two NVIDIA GTX480 cards. For comparison, we measure the performance of `lighttpd 1.4.26` with `OpenSSL 1.0.0`. All experiments use 1024 bit RSA, 128 bit AES-CBC, and HMAC-SHA1 as the SSL cipher suite.

Target	TPS
lighttpd with OpenSSL (without proxy)	9,246
SSL proxy GPU (lighttpd in the same machine)	16,497
SSL proxy GPU (separate backend)	25,823

Table 2: Maximum transactions per second

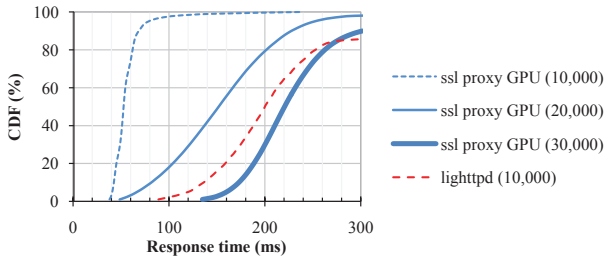

Figure 1: Response time distribution (Number in the parenthesis indicates offered load in HTTPS requests per seconds)

In Table 2, we compare the TPS numbers with 1B content size for three different configurations. We see that 16K TPS is achieved with GPU acceleration, while lighttpd performs about 9K TPS. To identify the HTTP overhead we run the same experiment with the backend lighttpd webserver on a separate machine. In this case GPU acceleration achieves about 25K TPS. We further analyze the CPU consumption with `oprofile` and find that the bottleneck is in the Linux kernel, consuming 50 to 60% of the CPU cycles for TCP/IP processing. Particularly, TCP connection establishment handling in the kernel does not scale with the number of CPU cores.

In Figure 1 we plot the response time distribution by varying the TPS. For latency measurement, we use a separate machine for the SSL proxy. Note that when the offered load is 30K TPS, actual throughput is around 25K TPS. When the offered load is 10K TPS, the GPU-accelerated SSL proxy shows response time even better than the lighttpd with its own SSL support. When the offered load is 30K TPS, our SSL proxy shows comparable response time with lighttpd, and has better 90-th percentile and worst-case latency. These results demonstrate that our opportunistic offloading algorithm balances the load between CPU and GPU effectively depending on the load. We have observed that GPU is not fully utilized due to the bottleneck at connection handling in the Linux kernel. We plan to look into ways to remove the TCP/IP stack bottleneck for further performance improvement.

5. ACKNOWLEDGEMENT

This research was funded by NAP of Korea Research Council of Fundamental Science & Technology, and MKE (Ministry of Knowledge Economy of Repbulic of Korea, project no. N02100053).

6. REFERENCES

[1] Netcraft SSL Survey. `http://news.netcraft.com/SSL-survey`, 2009.

[2] Netcraft Web Server Survey. `http://news.netcraft.com/archives/2010/04/15/april_2010_web_server_survey.html`, 2009.

[3] C. Coarfa, P. Druschel, and D. S. Wallach. Performance Analysis of TLS Web Servers. In *NDSS*, 2002.

[4] G. Apostolopoulos *et al.* Transport Layer Security: How much does it really cost? In *IEEE Infocom*, 1999.

[5] O. Harrison and J. Waldron. Practical Symmetric Key Cryptography on Modern Graphics Hardware. In *USENIX Security*, 2008.

[6] O. Harrison and J. Waldron. Efficient Acceleration of Asymmetric Cryptography on Graphics Hardware. In *Africacrypt*, 2009.