# Analysis of Transport Optimization Techniques

Kevin J. Ma
Cisco Systems, Inc.
Boxborough, MA 01719, USA
Email: kema@cisco.com

Radim Bartoš
Department of Computer Science
University of New Hampshire,
Durham, NH 03824, USA
Email: rbartos@cs.unh.edu

*Abstract*— **The popularity of Web-based transactions and the need for more sophisticated content distribution methods has helped to fuel the rapid growth of Web Service adoption, specifically, HTTP-bound Web Services. Secure and efficient content delivery has long been a requirement of traditional Web-based distribution schemes, and existing the Web infrastructure provides numerous options for securing and optimizing HTTP. Two exemplary technologies are SSL/TLS and HTTP compression. While efforts to solidify the more granular WS-Security standards are ongoing, and methods for XML message compression schemes continue to be investigated, HTTP provides an interim solution, supporting transactional security and message compression. The SSL/TLS and HTTP compression technologies have become commoditized and pervasive. And with the trend in content delivery toward hardware offload for these functions, modern data centers have begun to raise the bar for performance. In this paper, we examine three different paradigms for implementing SSL/TLS and HTTP compression: software-based functionality, server-resident hardware accelerators, and centralized network-resident hardware accelerators. We discuss the trade-offs between the two different offload techniques (i.e., PCI accelerator vs. network proxy) and explore their relationship to the current performance activities, in the field of Web Services. In analyzing the results for SSL/TLS offload, and the effects of compression, in conjunction with SSL/TLS, we draw parallels with the efforts of WS-Security and XML message compression. Although optimizations for software-based cryptography will continue to advance, the potential for hardware-based acceleration should not be overlooked. We discuss our results and address deployment scenarios for optimizing Web-based transactions, and the future optimization of Web Service transactions.**

## I. Introduction

Web Services are quickly gaining in popularity and acceptance as the method of choice for implementing Web-based transactions. As Web Services increase penetration into the data center and as content distribution networks become more reliant on Web Service technologies, the Web Service infrastructure must continue to evolve and achieve higher throughput and greater capacity. To that end, numerous research activities involving XML parser enhancements (e.g., DOM vs. SAX vs. XPP [1]) and implementation differences [2], [3]), data size reduction (e.g., HTTP compression [4], binary XML [5], differential encoding [6], [7], etc.), and transport protocol enhancements [9] are ongoing. All three techniques have been shown to provide significant performance benefits.

Similar strategies have been applied to WS-Security performance research efforts [8], where individual cryptographic operations are typically regarded with a "black box" approach. And while cryptographic research is beyond the scope of Web Services themselves, privacy and trust are essential components of the Web Service paradigm. Thus, cryptography is a critical component of Web Service infrastructure. The impact of software-based cryptography should not be overlooked when evaluating core security performance enhancements. High performing and secure transactions are integral to the success of Web Services. Likewise, activities related to reducing the size of XML data representations should not overlook the effects of software-based compression implementations.

The popular HTTP SOAP transport binding intrinsically supports options for both data size reduction (i.e., HTTP Compression) and transaction authentication and privacy (i.e., SSL/TLS). While efforts to enhance infrastructure protocols are ongoing, the existing HTTP infrastructure protocol offers proven enhancement options for existing Web Service deployments. The support and expertise in these areas is already well developed, and significant resources have been invested into the advancement of these features.

One concern with any post-processing technique, is the significant CPU performance penalties associated with multi-pass data parsing. To address such concerns, two offload paradigms are generally employed: server-resident and network-resident hardware accelerators.

The former is typically implemented in a standard PCI form-factor (either as standalone devices, or integrated into NIC cards, etc.) or integrated into the system chipset. The latter comprises a multitude of switches, routers, load balancers, and custom network appliances. In general, both paradigms are effective in reducing the burden on the main CPU, but each has its own strengths weaknesses. The following discussion examines the trade-offs between the two offload paradigms, compared with the purely software implementation.

This paper explores the performance characteristics of these different infrastructure offload techniques using commercial off-the-shelf (COTS) solutions. Comparing both the macroscopic and microscopic properties of these two approaches, we extrapolate some of the generic tendencies in hardware acceleration, as we look to apply them to Web Service optimization research. Having a better understanding of the lessons learned with HTTP, we can begin to define new baselines for Web Service quality and performance, and proceed with confidence.

## II. Methodology

The test strategy was to look at three aspects of the offload techniques: client experienced latency, server throughput, and server CPU usage.

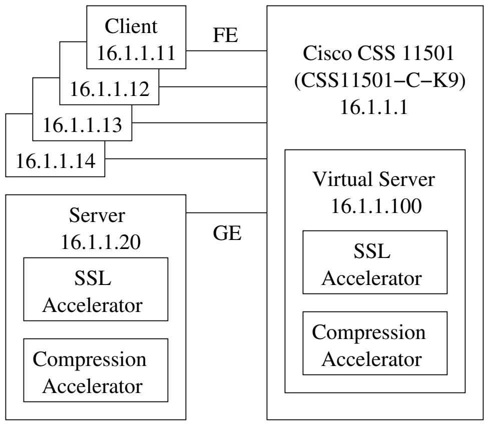

Fig. 1. Network Test Configuration.

Our test network was comprised of client machines connected through a Cisco® CSS 11501 Content Services Switch [16] to a single server. The Cisco CSS 11501 is a server load balancing Ethernet switch with routing and offload capabilities for SSL/TLS and HTTP compression. Fig. 1 depicts the test setup. The clients were directly connected to the Cisco CSS 11501, via built-in 100BASE-T Ethernet links. The server was direct connected to the Cisco CSS 11501, via a built-in 1000BASE-T Ethernet link.

## A. Server Configuration

The server used in our experiments is a dual core Xeon 3.00 GHz machine. Each core has 1 MB of L2 cache and the system has 1 GB of total RAM. The operating system installed is Fedora Core 2, kernel version 2.6.5-1.358smp. Apache 2.2 [10] and OpenSSL-0.9.8a [11] were installed for the purposes of these tests. Apache was built with the included `mod_ssl` and `mod_deflate` modules. The `mod_ssl` `SSLRandomSeed` directive was configured to restrict the usage to only `/dev/urandom`, to avoid blocking delays that can be caused by lack of entropy, when using `/dev/random` [12], and the `SSLCipherSuite` directive was set to `RC4+RSA+MD5`, to match the Cisco CSS 11501 configuration.

The server was also equipped with a Broadcom BCM5821 Crypto Accelerator, rev 02 [13] and a Comtech AHA AHA362-PCIX Compression Accelerator [14]. The BCM5821 Co-Processor is installed on a standard PCI bus. The AHA362 shared the PCI-X bus with the build-in Gigabit Ethernet NIC card. Broadcom provided a driver with the Broadcom 582x Linux SDK Version: 2.51. Support for the BCM582x has been included in OpenSSL since version 0.9.6c, through the standard `ENGINE` framework, using the `ubsec` engine type. The Apache HTTP Server supports for OpenSSL `ENGINE`s, via the `SSLCryptoDevice` directive. Comtech AHA provided both a driver and an Apache module `mod_deflate_aha`. The Apache HTTP Server supports the `SetOutputFilter` directive for enabling the included `DEFLATE` type and the Comtech AHA `DEFLATE_AHA` type.

TABLE I
TEST FILE SIZES.

| *File Name* | *File Size (bytes)* | *Compression* | | |
|---|---|---|---|---|
| | | *Method* | *Size (bytes)* | *Ratio* |
| index.html | 44 | N/A | N/A | N/A |
| cisco.html | 66613 | AHA | 28650 | 2.33:1 |
| | | CSS | 16118 | 4.13:1 |
| | | software | 11951 | 5.57:1 |
| xml.html | 232081 | AHA | 115768 | 2.00:1 |
| | | CSS | 76071 | 3.05:1 |
| | | software | 53626 | 4.33:1 |

Table I provides details on the three test files used in our experiments. It shows the original files sizes, as well as the compressed file sizes when compressed via the AHA362, the Cisco CSS 11501[1], and the `mod_deflate` software implementation. No compressed sizes are reported for the file `index.html`, as the small size makes compression impractical. All of the files should be fully cachable, by the server. Problems with cache thrashing and disk access latency should not influence test results.

*1) SSL/Compression Accelerators:* There are numerous silicon vendors offering cryptographic offload solutions (e.g., Broadcom and Cavium Networks), compression offload solutions (e.g., Comtech AHA and Indra Networks), as well as combination solutions (e.g., Cavium Networks), with and without NIC support. For cryptographic offload we chose the Broadcom BCM5821 for its Apache and OpenSSL support, and ease of installation. The BCM5821 also happens to be the same device used by the Cisco CSS 11501 SSL module. We chose the Comtech AHA AHA362 also for its Apache support and ease of availability. The Cisco CSS 11501 uses a proprietary compression solution not available for server-side testing, however, the AHA362 is used in competitive network appliances, making it a suitable candidate.

*2) Network Appliance:* For our network-resident acceleration appliance, we chose the Cisco CSS 11501 Content Service Switch, from Cisco Systems®. Numerous load balancer vendors offer solutions with SSL/TLS and compression offload (e.g., Cisco Systems, F5 Networks, Citrix Netscaler, Juniper Networks, etc.). The Cisco CSS 11501 was made available to us for testing, by Cisco Systems.

## B. Client Configuration

Four client machines were used in our test configuration. The clients are dual core Pentium 4 3.00 GHz machines. Each core has 1 MB of L2 cache and each system has 256 MB of system RAM. The operating system installed is Red Hat 8, kernel version 2.4.20-28.7.

*1) Latency Testing:* Latency tests were performed by issuing individual requests, from a single client, to the server, using

[1]The compressed size of files compressed by the Cisco CSS 11501 SSL and compression module may vary, from run to run. The Cisco CSS 11501 uses a streaming approach to reduce the buffering requirement and memory resource impact involved in the compression process. Compression occurs as data becomes available. Slight variances in the output of the server TCP stack can cause variances in the Cisco CSS 11501 compression performance.

cURL [15], version 7.15.1. Scripts were used to repeat cURL requests 1000 times. Data was then collected at the client via `tcpdump`, version 3.6.3. Traces were filtered, dumped to file, and then subsequently post-parsed, to determine the transaction time for each request.

*2) Throughput Testing:* Throughput tests were performed using the Tarantula test tool. Tarantula was developed by Arrowpoint Communications (acquired by Cisco Systems, Inc., in 2000) for generating simultaneous HTTP(S) connections using multiple client threads, from individual physical machine. Tarantula was licensed for evaluation, but was never made publicly available. Today many vendors (e.g., Ixia and Spirent Communications) offer network test tools, for generating large numbers of requests from large numbers of simulated clients. We are evaluating numerous freely available software packages (e.g. Apache benchmark, Hammerhead 2, SSL Client, http_load, etc.). For convenience, however, we chose to use Tarantula, for the purposes of these tests.

All four clients were used in all cases, with the number of threads tuned to reduce errors. In over-subscribed cases, TCP timeouts and resets will occur, skewing test results. The number of total requests was then appropriately size to create an approximate test duration of 20 minutes. The Tarantula test tool provides connection per second data for each machine.

*3) CPU Usage Monitoring:* CPU usage was approximated, by observing the output of the `top` command, on the server, and the `show system-resources cpu-summary` command, on the Cisco CSS 11501.

### *C. Cisco CSS 11501 Configuration*

The Cisco CSS 11501 (CSS11501-C-K9) is equipped with a built-in SSL/TLS/Compression acceleration module. The Cisco CSS 11501 was configured with separate virtual servers for: no SSL/TLS, front-end SSL/TLS, and front-end SSL/TLS with compression.

## III. RESULTS

The following sections summarize the results of our testing. They are divided into three sections: latency, throughput, and CPU usage. Each table and graph is broken down by the file name and SSL/TLS and/or compression offload method employed. Graphs use the notation: "encryption method/compression method" to distinguish between sample sets. Note: compression tests using `index.html`, and tests combining Cisco CSS 11501 compression with server-side encryption were omitted, because of the impracticality of compressing small and/or encrypted data, respectively.

### *A. Latency Results*

Table II shows the average latency measurements for each of the files and offload methods. Fig. 2 and Fig. 3 offer graphical representations of the average latency data from Table II grouped by compression offload method, and cryptographic offload method, respectively.

Most notably, Fig. 2 clearly displays large latency impact of cryptographic operations. It also depicts the relatively small difference in latency, between the cryptographic offload options, when using the same compression offload method. The hardware acceleration appears to have no significant effect on reducing individual transaction latency. However, as Table III will show, the hardware acceleration greatly improves bulk throughput. This is likely due to the parallelization allowed for by hardware offload, freeing up the CPU for other tasks.

TABLE II
AVERAGE SINGLE TRANSACTION LATENCY.

| *File Name* | *Crypto* | *Compress* | *Latency (s)* |
|---|---|---|---|
| index.html | none | none | 0.003297 |
| | CSS | none | 0.023177 |
| | BCM | none | 0.022008 |
| | software | none | 0.023161 |
| cisco.html | none | none | 0.007767 |
| | none | CSS | 0.006949 |
| | none | AHA | 0.004482 |
| | none | software | 0.008294 |
| | CSS | none | 0.029119 |
| | CSS | CSS | 0.026685 |
| | CSS | AHA | 0.017385 |
| | CSS | software | 0.029576 |
| | BCM | none | 0.028097 |
| | BCM | AHA | 0.016154 |
| | BCM | software | 0.029143 |
| | software | none | 0.029302 |
| | software | AHA | 0.017272 |
| | software | software | 0.030314 |
| xml.html | none | none | 0.021826 |
| | none | CSS | 0.015512 |
| | none | AHA | 0.011738 |
| | none | software | 0.032399 |
| | CSS | none | 0.043557 |
| | CSS | CSS | 0.035454 |
| | CSS | AHA | 0.024968 |
| | CSS | software | 0.054249 |
| | BCM | none | 0.042194 |
| | BCM | AHA | 0.023609 |
| | BCM | software | 0.052634 |
| | software | none | 0.043598 |
| | software | AHA | 0.025266 |
| | software | software | 0.054726 |

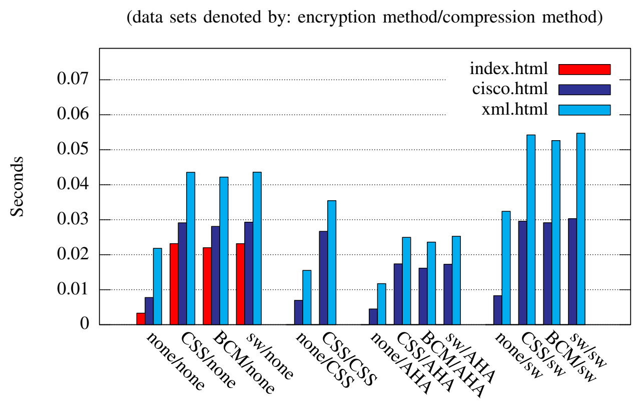

Fig. 2. Average Latency per Transaction (grouped by compression method).

Fig. 3 shows the effect compression has on reducing latency, with each cryptographic offload method. The hardware accelerated cases show marked improvement in latency reduction. For the software compression cases, however, latency increases as the additional CPU processing apparently outweighs savings from encrypting and sending less data. The AHA362 shows a clear speed advantage over the Cisco CSS 11501, however, it comes at a cost. The trade of is a significantly lower compression ratio, as was shown in Table I.

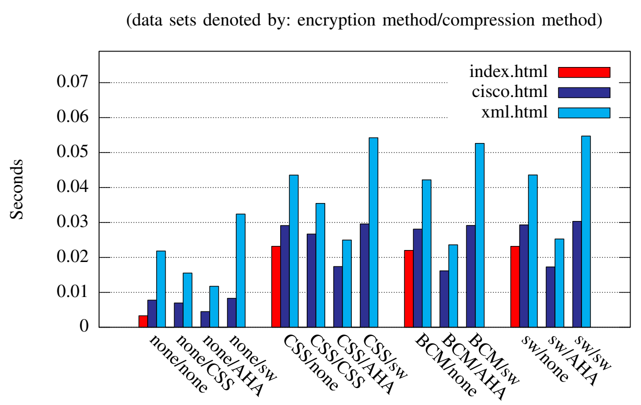

Fig. 3. Average Latency per Transaction (grouped by encryption method).

Figs. 4 through 10 show the 1000 sample run data for each of the cases shown in Table II. Though in many cases there is an overlap in the samples due to the comparable latencies across groups of test cases, these graphs show the experimental consistency under which our averages were derived.

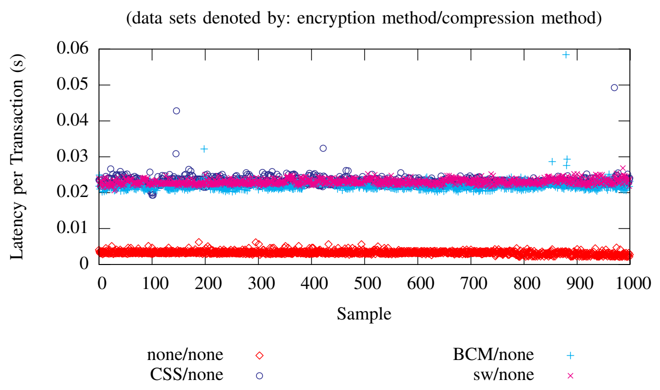

Fig. 4. Encrypted index.html Transfer Latency.

In Figs. 4, 5, and 8, the overhead of SSL/TLS over cleartext transactions can be seen, while Fig. 9 depicts the value of compression offload, and the cost of software compression, in large files. Fig. 4 distinctly shows the inherent latency imposed by the SSL/TLS handshake. The time to software encrypt such a small file should be negligible, therefore, the latency differential may be attributed to the setup and teardown.

Fig. 11 shows the packet flow of a typical HTTP transaction. It includes a TCP handshake and teardown, and application data frames containing HTTP messages. While variations in TCP transmission and ACK processing may occur, the basic structure is constant.

Fig. 12 shows the packet flow for a typical HTTPS transaction. Because variations in SSL/TLS handshake processing is possible, this packet flow was verified using `tcpdump` and `ethereal`, tracing `cURL` transactions between client and server. The extra SSL/TLS handshake messages introduce latency, both from the serialized message exchange, as well as the cryptographic operations require for signing and encrypting those messages [17].

In looking at the network proxy paradigm, the packet flow changes yet again, as depicted in Fig. 13. For network SSL/TLS offload, the client connection must be TCP terminated, SSL terminated, and HTTP terminated at the

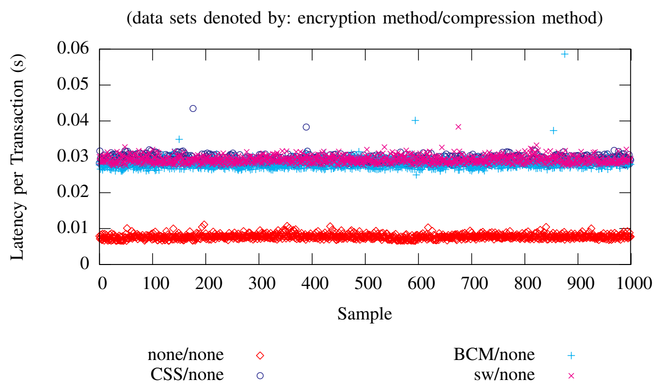

Fig. 5. Encrypted cisco.html Transfer Latency.

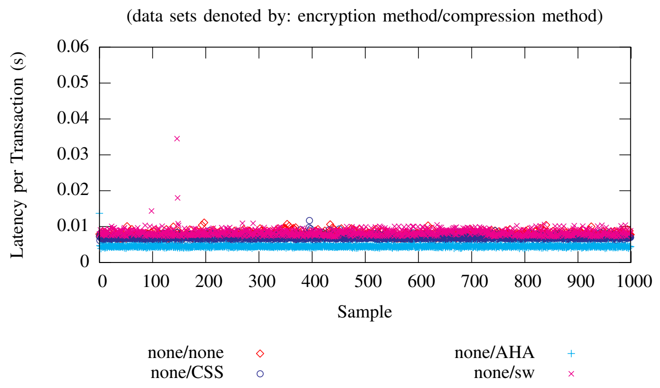

Fig. 6. Compressed cisco.html Transfer Latency.

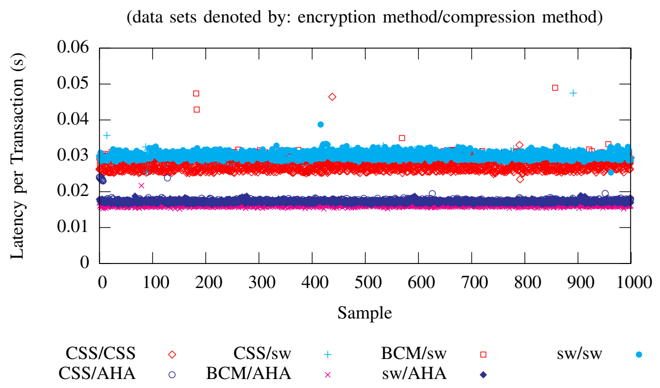

Fig. 7. Encrypted and Compressed cisco.html Transfer Latency.

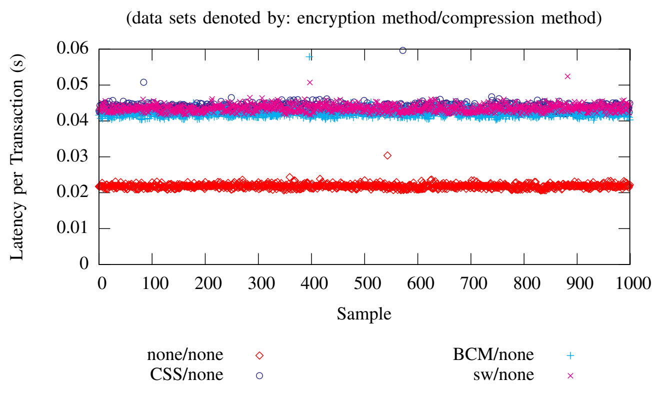

Fig. 8. Encrypted xml.html Transfer Latency.

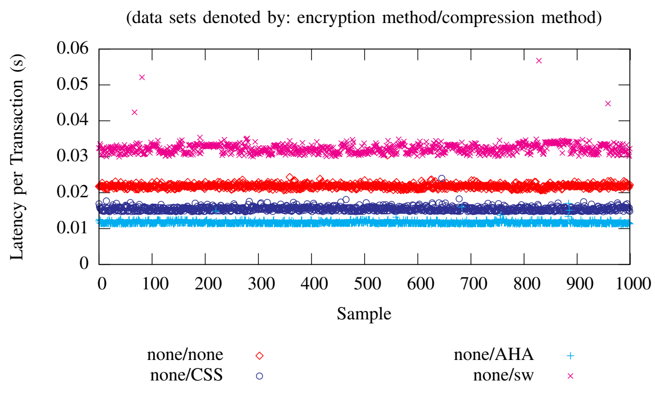

Fig. 9. Compressed xml.html Transfer Latency.

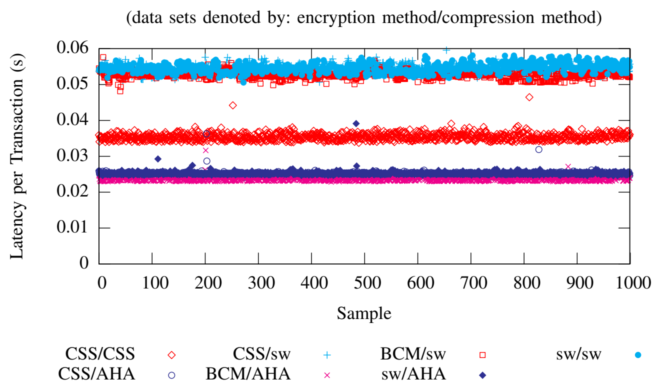

Fig. 10. Encrypted and Compressed xml.html Transfer Latency.

network device. A separate connection must be initiated, by the network device, to the back-end server, resulting in double the amount of data processing, by the Cisco CSS 11501 TCP stack. Another consideration is that the Cisco CSS 11501 SSL module does not have comparable processing capabilities, to the dual core Xeon server. It has been shown that the SSL/TLS handshake processing can be very processor intensive, even without the cryptographic operations [18].

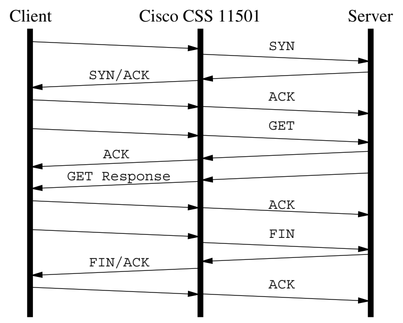

Fig. 11. HTTP `GET` Transaction.

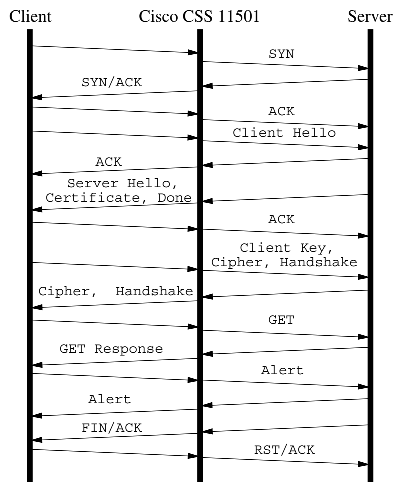

Fig. 12. HTTPS `GET` Transaction.

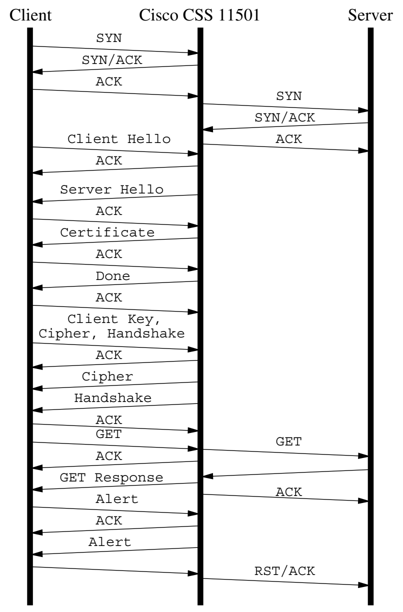

Fig. 13. Cisco CSS 11501 Accelerated HTTPS `GET` Transaction.

## *B. Throughput Results*

Table III shows a summary of the bulk throughput results. The connections per second number is directly reported by the Tarantula test tool. These results are specific to this particular network configuration. Single server scenarios are seldom optimal or practical, and are certainly not ideal for load balancer testing. The single server scenarios serialize requests through one server, eliminating the ability of load balancers to batch process parallel connections. Variations in network topology, client or server hardware or software, cryptographic cipher suite, or compression parameters may have significant impact on the observed results. However, for the purposes of this comparison, using a single server allows us to more easily saturate the server and isolate the key usage statistics. From a relative perspective, this controlled environment allows us to focus on the differences between the offload methods. The megabit per second statistics were computed manually, based on the size of the files. The "link" bandwidth represents the actual physical network load (i.e., conn/sec x compressed size); the "data" bandwidth shows the effective logical link bandwidth, from the client perspective (i.e., conn/sec x uncompressed size). This gross approximation of total bandwidth, that does not take into account protocol overhead, is provided for its relative comparison value. Figs. 14 and 15 show graphical representations of the data provided in Table III. Note: Because of unresolved problems with the AHA362, at the time of this writing, we are unable to provide bulk throughput data, for the `xml.html` file cases.

TABLE III
BULK TRANSACTION THROUGHPUT.

| *File Name* | *Crypto* | *Compress* | *Throughput (conn/sec)* | *Bandwidth (Mbps)* | |
|---|---|---|---|---|---|
| | | | | *Link* | *Data* |
| index.html | none | none | 5283.50 | 1.77 | 1.77 |
| | CSS | none | 752.94 | 0.25 | 0.25 |
| | BCM | none | 1185.62 | 0.40 | 0.40 |
| | software | none | 291.90 | 0.10 | 0.10 |
| cisco.html | none | none | 709.53 | 360.60 | 360.60 |
| | none | CSS | 693.25 | 85.25 | 352.32 |
| | none | AHA | 1657.67 | 362.34 | 842.45 |
| | none | software | 250.95 | 22.88 | 127.54 |
| | CSS | none | 245.78 | 124.91 | 124.91 |
| | CSS | CSS | 332.27 | 40.86 | 168.86 |
| | CSS | AHA | 463.13 | 101.23 | 235.37 |
| | CSS | software | 245.50 | 22.38 | 124.77 |
| | BCM | none | 623.51 | 316.88 | 316.88 |
| | BCM | AHA | 744.08 | 162.64 | 378.15 |
| | BCM | software | 207.92 | 18.96 | 105.67 |
| | software | none | 242.05 | 123.01 | 123.01 |
| | software | AHA | 254.60 | 55.65 | 129.39 |
| | software | software | 144.58 | 13.18 | 73.48 |
| xml.html | none | none | 205.01 | 363.00 | 363.00 |
| | none | CSS | 208.84 | 121.20 | 369.77 |
| | none | software | 52.00 | 21.27 | 92.07 |
| | CSS | none | 88.49 | 156.68 | 156.68 |
| | CSS | CSS | 140.68 | 81.65 | 249.09 |
| | CSS | software | 51.49 | 21.07 | 91.17 |
| | BCM | none | 203.81 | 360.87 | 360.87 |
| | BCM | software | 48.94 | 20.02 | 86.66 |
| | software | none | 167.81 | 297.14 | 297.14 |
| | software | software | 45.73 | 18.71 | 80.97 |

Fig. 14 shows the transaction capacity of the server, for the different offload methods. Immediately apparent is the order of magnitude difference, between the simple, small file, HTTP `GET`, and all the other cases. The end-to-end system is highly optimized for the benchmark condition, of index page requests. Even dwarfed below the `index.html` HTTP request, the performance benefits of server and network-resident offload over software, can be seen.

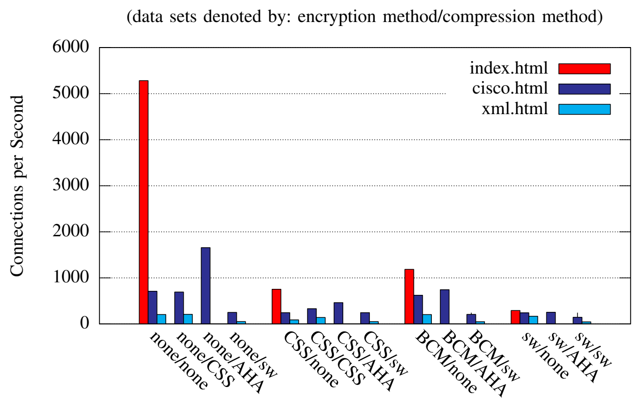

Fig. 14. HTTP(S) `GET` Transaction Throughput.

Fig. 15, shows the relationship of raw data bandwidth, as seen by the physical network links. This view allows us to see the relative compression performance. Note: the uncompressed `index.html` data is plotted, but is too small to see, the AHA compressed `xml.html` data is not available, and the bandwidth is a function of both the compression performance as well as the transaction throughput, e.g. the software compression cases affected by its higher compression ration, but it is also limited by its low transaction throughput.

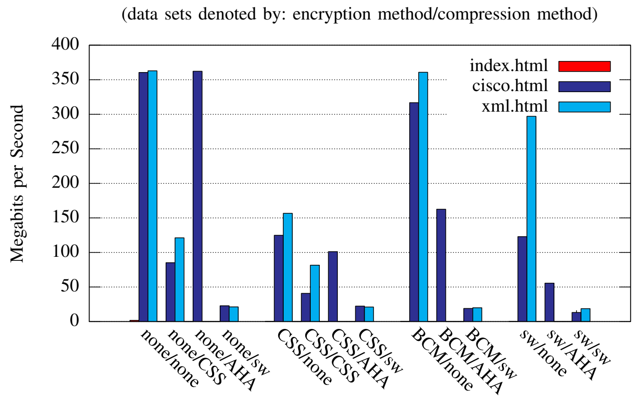

Fig. 15. HTTP(S) `GET` Transaction Physical Network Bandwidth.

It can be seen that the server-based BCM5821 performance exceeds that of the Cisco CSS 11501. Given identical cryptographic accelerators, the performance difference between the server and the Cisco CSS 11501 is explained by the difference in the hardware that feeds the BCM5821. The processing cycles and cache available to the server is an order of magnitude larger than that of the Cisco CSS 11501 SSL module, with the Cisco CSS 11501 also managing twice as many connections.

### *C. CPU Usage*

In the previous sections, we have seen how the different offload techniques affect the end user experience, through how fast they receive the response, and how the different offload techniques affect the ability off the servers to do more work. In this section we look at the server resource cost of these approaches. Table IV and Fig. 16 show the approximated CPU usage observed, during the bulk throughput testing.

TABLE IV
BULK TRANSACTION CPU USAGE.

| *File Name* | *Crypto* | *Compress* | *CPU Usage* | | |
|---|---|---|---|---|---|
| | | | *Server* | *CSS* | |
| | | | | *SCM* | *SSL* |
| index.html | none | none | 98 | 57 | 0 |
| | CSS | none | 13 | 12 | 100 |
| | BCM | none | 100 | 12 | 0 |
| | software | none | 100 | 0 | 0 |
| cisco.html | none | none | 29 | 4 | 0 |
| | none | CSS | 27 | 11 | 99 |
| | none | AHA | 83 | 16 | 0 |
| | none | software | 100 | 0 | 0 |
| | CSS | none | 9 | 1 | 78 |
| | CSS | CSS | 13 | 4 | 99 |
| | CSS | AHA | 22 | 7 | 90 |
| | CSS | software | 100 | 1 | 56 |
| | BCM | none | 100 | 4 | 0 |
| | BCM | AHA | 100 | 6 | 0 |
| | BCM | software | 100 | 0 | 0 |
| | software | none | 100 | 0 | 0 |
| | software | AHA | 100 | 0 | 0 |
| | software | software | 100 | 0 | 0 |
| xml.html | none | none | 19 | 0 | 0 |
| | none | CSS | 17 | 0 | 99 |
| | none | software | 100 | 0 | 0 |
| | CSS | none | 7 | 0 | 68 |
| | CSS | CSS | 12 | 0 | 93 |
| | CSS | software | 100 | 0 | 10 |
| | BCM | none | 63 | 0 | 0 |
| | BCM | software | 100 | 0 | 0 |
| | software | none | 100 | 0 | 0 |
| | software | software | 100 | 0 | 0 |

As with the bandwidth numbers in Table III, the CPU usage is provided for comparative analysis, relative to the other test cases. Three CPU usage values are represented: one for the aggregate server use (accounting for both cores of the dual core Xeon), and one for each of the Cisco CSS 11501 internal processors, i.e., the System Control Module (SCM) CPU and the SSL Acceleration Module (SSL) CPU. Fig. 16 depicts a graphical representation of the data shown in Table IV.

From a macroscopic view, it can be seen that the high throughput of the server requires all of the servers resources to maintain. The relative CPU use between the Cisco CSS 11501 and the BCM5821 cases is not proportional to the effective bandwidth differential between the two cases.

Table V shows shows the CPU usage advantage of the Cisco CSS 11501, and the bulk throughput advantage of the server. Although the server-based offload shows a total throughput advantage, it is at a much higher proportional server resource cost. If only basic Web content delivery is required, then server-based acceleration may be optimal. However, server-based offload may prevent network-based content inspection.

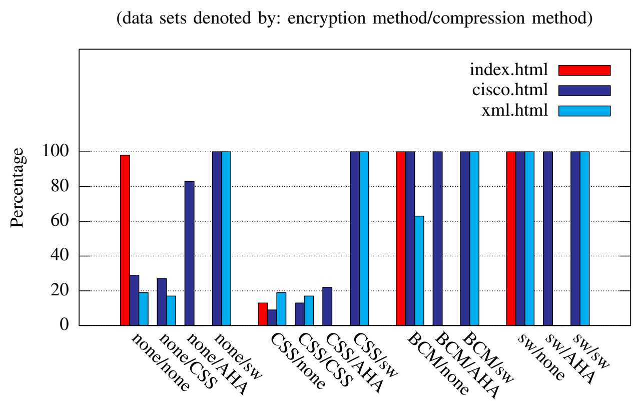

Fig. 16. Server CPU Usage.

TABLE V
SERVER VS. NETWORK OFFLOAD.

| *Comparison* | *File Name* | *CPU* | *Throughput* |
|---|---|---|---|
| CSS/none vs. BCM/none | index.html | 7.69x | 1.57x |
| | cisco.html | 11.11x | 2.54x |
| | xml.html | 9x | 2.30x |
| CSS/CSS vs. BCM/AHA | cisco.html | 7.69x | 2.24x |

If dynamic or computationally intensive content is being delivered, then network-based offload may be more favorable.

## IV. FUTURE WORK

The data provided herein concentrates on the comparison between single purpose cryptographic and compression accelerators, embedded in the server, with a network-based SSL and compression offload solution. Other offload hardware techniques include server-resident SSL/compression NIC cards, and TCP offload engines. Advancements have also been made in cryptographic offload hardware, to help accelerate the handshake process, as well as the bulk cryptographic operations, and in compression offload hardware to enhance dynamic huffman encoding. Investigation of these different paradigms and enhancements, to discover the capabilities of recent advancements in hardware acceleration will help to define a more practical baseline, for Web Service benchmarking.

Multiserver scenarios also need to be evaluated to obtain a better understanding of the system and network dynamics of real world server farms. For more realistic modeling, data center scenarios accounting for the effects of load balancer features and optimizations, should be evaluated. To better quantify these trade-offs, it is also important to implement Web Services with measurable metrics and flexible configurations for assessing the effects of local and remote storage access and complex computations, on the true impact of server-side accelerators. And while the monetary cost of a network appliance will be higher than the monetary cost of an individual PCI card, the total cost of ownership is hard to quantify with single server scenarios, because the network appliance will be able to service multiple servers, where as PCI card costs must be multiplied by the number of servers. Network-based offload tends to mitigate scalability issues, however distributed protocol interdependencies must be investigated.

Additionally, the true value of compression is in network architecture having slow links. Modeling a system in which clients connect through 56-kbps, rather than 100-Mbps, links may provide better insight into the benefits of compression.

Finally, looking specifically at Web Service based offload, there are numerous vendors offering XML and WS-Security acceleration devices, both server-resident offload modules (e.g., from Tirari and Xambala), and network-resident appliances (e.g., from IBM Datapower, Intel Sarvega, Cisco Application Oriented Networking (AON), etc.). While further work still exists in the current area of HTTP acceleration, the future lies in the acceleration of Web Services through Web Service specific protocols. Evaluation of the characteristics of this new class of acceleration devices will provide the best possible baseline, for Web Service benchmarking.

## V. Conclusions

This paper presents the results of our testing with hardware cryptographic and compression offload techniques, and the performance of HTTP(S) transactions in server-resident and network-resident SSL/TLS/compression offload scenarios. It is well-known that cryptographic operations are costly when implemented in software. Our results certainly confirm this, and hardware acceleration of such operations has been around for many years. However, multiple deployment options exist for such offload. Our results show distinct trade-offs between the higher throughput and lower CPU usage, for server-side offload. Our testing also showed limited advantage to HTTP compression, as it relates to overall server use and performance. These results pose interesting implications toward the current line of software-centric Web Service optimization research. Existing activities, in the area of Web Service security performance, and XML data compression may be directly analogous to these corresponding HTTP infrastructure results.

The WS-Security standards employ similar algorithms and techniques as SSL/TLS for digital signatures and encryption. While the scope and granularity and flexibility of the WS-Security standards certainly exceed the capabilities of SSL/TLS, many of the same pitfalls and acceleration techniques may still apply. When the WS-Security reaches the wide-spread commoditization that SSL/TLS has achieved, we can expect to see the proliferation of WS-Security specific offload devices. In the interim, the SSL/TLS provides a more easily accessible platform for performance research.

Likewise, XML message compression techniques may also be able to draw from the experiences of HTTP compression deployments. Traditional multi-pass, file compression techniques are computationally intensive and burden server CPUs. However, they are also well encapsulated, easily offloaded, and clearly demarcated, in the data processing flow. Although binary representations and deserialization optimizations allow for a more tightly integrated compression scheme, with possibly lower CPU impact, such coupling affects the ability to offload and accelerate Web Service infrastructure. Consideration should be given to the ultimate option for hardware acceleration of all aspects of Web Service deployment.

We feel that there is significant value to understanding the optimization techniques employed in current Web Service deployments, including traditional HTTP acceleration methods. Beyond the prevailing practical paradigms, the future of high performance Web Services is in the migration to hardware and possibly network offload of Web Service infrastructure. As with most advancements in technology, it is the application of well known and well tested ideas, to new problems and priorities, that leads to innovation.

## References

[1] M. Head, M. Govindaraju, A. Slominski, P. Liu, N. Abu-Ghazaleh, R. van Engelen, K. Chiu, and M. Lewis, "A Benchmark Suite for SOAP-based Communication in Grid Web Services," in *Proc. ACM/IEEE Conference on Supercomputing (SC'05)*, Seattle, WA, USA, Nov. 2005, Page 19.

[2] D. Davis and M. Parashar, "Latency Performance of SOAP Implementations," in *Proc. ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGRID'O2)'*, Berlin, Germany, May 2002, Page 407.

[3] K. Ma and R. Bartoš, "Performance Impact of Web Service Migration in Embedded Environments," in *Proc. IEEE International Conference on Web Services (ICWS'05)*, Orlando, FL, USA, Jul. 2005, Pages 409-416.

[4] H. Liu, X. Lin, and M. Li, "Modeling Response Time of SOAP over HTTP," in *Proc. IEEE International Conference on Web Services (ICWS'05)*, Orlando, FL, USA, Jul. 2005, Pages 673-679.

[5] (2005) XML Binary Characterization Working Group. [Online], Available: http://www.w3.org/XML/Binary/

[6] C. Werner, C. Buschmann, and S. Fischer, "Compressing SOAP Messages by Using Differential Encoding," in *Proc. IEEE International Conference on Web Services (ICWS'04)*, San Diego, CA, USA, Jul. 2004, Pages 540-547.

[7] T. Suzumura, T. Takase, and M. Tatsubon, "Optimizing Web Services Performance by Differential Deserialization," in *Proc. IEEE International Conference on Web Services (ICWS'05)*, Orlando, FL, USA, Jul. 2005, Pages 185-192.

[8] S. Makino, M. Tatsubori, K. Tamura, and T. Nakamura "WS-Security Performance with a Template-Based Approach," in *Proc. IEEE International Conference on Web Services (ICWS'05)*, Orlando, FL, USA, Jul. 2005, Pages 581-588.

[9] C. Werner, C. Buschmann, T. Jäcker, and S. Fischer, "Enhanced Transport Bindings for Efficient SOAP Messaging," in *Proc. IEEE International Conference on Web Services (ICWS'05)*, Orlando, FL, USA, Jul. 2005, Pages 193-200.

[10] (2006) The Apache HTTP Server Project. [Online], Available: http://httpd.apache.org/

[11] (2006) The Open Source toolkit for SSL/TLS. [Online], Available: http://www.openssl.org/

[12] (2006) Apache Module mod_ssl. [Online], http://httpd.apache.org/docs/2.2/mod/mod_ssl.html

[13] (2006) Broadcom Security Processors. [Online], Available: http://broadcom.com/products/Enterprise-Small-Office/Security-Processors/

[14] (2006) Broadcom Security Processors. [Online], Available: http://www.aha.com/show_prod.php?id=32

[15] (2006) cURL and libcurl. [Online], Available: http://curl.haxx.se/

[16] (2006) Cisco CSS 11500 Series Content Services Switches. [Online], Available: http://www.cisco.com/en/US/products/hw/contnetw/ps792/

[17] C. Coarfa, P. Druschel, and D. Wallach, "Performance Analysis of TLS Web Servers," in *Proc. ISOC Network and Distributed System Security Symposium (NDSS'02)'* San Diego, CA, USA, Feb. 2002, Pages 553-558.

[18] G. Apostolopoulos, V. Peris, and D. Saha, "Transport Layer Security: How Much Does It Really Cost?," in *Proc. IEEE International Conference on Computer Communications (INFOCOM'99)*, New York, NY, USA, Mar. 1999, Pages 717-725.