Architectural Impact of Secure Socket Layer on Internet Servers

Krishna Kant and Ravishankar Iyer
Server Architecture Lab
Intel Corporation, Beaverton, OR

Prasant Mohapatra*
Department of Computer Science and Engineering
Michigan state University
East Lansing, MI 48824

November 18, 1999

Abstract

Secure socket layer (SSL) is the most popular protocol used in the Internet for facilitating secure communications through authentication, encryption, and decryption. Although the use of SSL provides adequate security, the performance degradation has been drastic compared to non-secured data retrieval. In this paper, we analyze the performance and architectural impact of SSL on the servers in terms of various parameters such as throughput, utilization, cache sizes, cache miss ratios, number of processors, control dependencies, file access sizes, bus transactions, network load, etc. The experimental study is based on an Intel Pentium III Xeon based server; although many of the results should apply to other processors as well.

The major conclusions from this study are as follows: The use of SSL increases computational cost of the transactions by a factor of 5-7. SSL transactions do not benefit much from a larger L2 cache, but a larger L1 cache would be helpful. A complex logic for handling control dependencies is not useful for SSL transaction as the frequency of is branches is very low. It may be possible to enhance SSL performance by using a number of architectural features such as a large L1 cache, deeper pipelines, alternative cache organization to avoid cache pollution, security coprocessors, and efficient streaming of dirty data out of the cache onto the I/O bus.

Keywords: E-commerce, Internet Server Architecture, Secure Socket Layer (SSL), Secure Transactions, Server Performance.

---

[*] This work was done while Mohapatra was working for Intel Corporation as a Visiting Professor during the summer of 1999.

1 Introduction

The explosive growth in the usage of Internet in the past two years has put additional burden on the infrastructure that support the information superhighway. The original design of Internet was based on a noncommercial (research and educational) environment which was considered trustworthy. Recently, the Internet usage has proliferated in the commercial environment which makes security considerations of paramount importance.

Currently the most popular Internet application is World Wide Web, which provides a client-server environment for exchange of multimedia documents. Almost exclusively, this application forms the front-end for rapidly expanding e-commerce applications that run in the background. With increasingly sensitive information being exchanged via the Web, or more accurately via the hypertext transfer protocol (HTTP), the need for security has become critical. These security needs are being addressed in parallel at two levels:

1. At network (or IP) level via the Internet Protocol Security Protocol (IPSEC) [7]. This protocol is intended to be implemented in the network interface cards (NICs) and ensures the privacy, integrity and authenticity of all NIC to NIC communications. IPSEC allows a secured private network to be physically spread over the entire Internet.

2. At session (or transport) level via the Secure Sockets layer (SSL) [6]. This protocol secures an individual communication session and thus must reside above the transport (e.g., TCP) layer. Typically, SSL is implemented on the application processor, although it could be offloaded to auxiliary or coprocessors.

This paper concentrates on SSL, although many of the performance issues are common to both SSL and IPSEC. SSL is beginning to be used widely in e-commerce communications because of its capability in providing adequate security. However, it is believed to have a severe performance impact on the web server, and consequent long client response times. It is estimated that 10-25% of e-commerce transactions are aborted because of unduly long client response times, which translates into 1.9 billion dollar lost revenue [12]. In addition, the tardy response experienced by non-secured transactions on the same server (e.g., browsing transactions) may cause further abandonment by customers who may have otherwise moved on from “browsing” to the “ordering” step. Thus, the performance of the security protocol plays a significant role in the performance of web servers used in the e-commerce environment.

Recent media reports and also our own measurements confirm a precipitous drop in throughput when SSL is turned on a server running at its maximum capacity [12]. For example, it is reported that an Intel Pentium-II web server throughput drops from 322 ops/sec to 2.4 ops/sec under SSL, whereas a Sun450 throughput drops from 500 ops/sec to 3 ops/sec.[1] Based on such measurements, it has been claimed that SSL may degrade performance by a factor of 70 to 100. Such conclusions are false and misleading, and do not help in understanding what is really going on and how to get better performance. SSL secured communications that involve handshake each time could be

---

[1] Reference [12] reported 2.4 ops/sec on Intel Pentium-II erroneously as 24 ops/sec.

anywhere from 3-8 times more processing intensive than unsecured communications depending on the use of various options and the type of documents transferred. Beyond this level of performance degradation, any additional degradation is usually a result of number of "system" factors including lack of overload control, lack of priority scheduling, inappropriate engineering procedures, etc.

In this paper, we have done an experimental study of the SSL performance and its architectural impact using Intel Pentium III Xeon based servers. Intel processors provide detailed hardware level measurements via a tool called EMON, and these were used to understand low-level details of server performance for a number of configurations. System level information was collected using the PERFMON tool of NT O/S. The configurations analyzed varied the following parameters: (a) number of processors in the SMP server (uniprocessor, dual processor and quad processor configuration), (b) three different L2 cache sizes (512 KB, 1 MB and 2 MB), and (c) three different file sizes in order to differentiate the impact of the handshaking protocol from the bulk encryption and transfer cases. The study exposes various architectural features that impact the performance of secure transactions in the Internet based on SSL protocol. The results of this study could be used as a preliminary guide for designing high-performance Internet servers used for secure transactions.

Several techniques could be used to improve the performance of the SSL-based secure communication. It is possible to introduce some additional architectural features such as, efficient caching structure, support for additional instructions, and perhaps even a security coprocessor. These issues are detailed in Section 4. Looking beyond the architectural aspects, the most crucial issues in obtaining good performance from e-commerce servers (with or without SSL) are (a) a decent overload control scheme suitably aided by hardware mechanisms such as intelligent NICs, and (b) good server engineering practices. In contrast to "servers" in traditional telecommunications systems [8], these issues have been largely ignored for Web servers. A limited amount of studies have been reported on the architectural issues of E-commerce servers. An analysis of resource management policies on the basis of revenue generation is reported in [9]. Other related works on web server performance includes service differentiation [1, 10] and operating system support issues [2, 3, 5].

The rest of the paper is organized as follows. An overview of the SSL protocol is presented in Section 2. The resource requirements for SSL and the experimental setup are discussed in Section 3. The data analysis and the architectural issues are detailed in Section 4. Inferences and the concluding remarks are outlined in Section 5.

2 Overview of SSL

SSL is a protocol for securing client-server communications and includes mechanisms for authentication, encryption and decryption [6, 11]. It has two important functions: (a) authentication of the server and client at the beginning of the session, and (b) encryption/decryption of data exchanged between the two parties during the session. The authentication is performed via the SSL handshake protocol, which typically involves 3 phases, with one or more messages sent in each direction in each phase. Following is a somewhat simplified description of the three handshake phases:

1. Parameter Negotiation: This phase is initiated via a *client-hello* message that provides some challenge data, session-id, and the ciphers that the client can use (e.g., RC4, DES, 3DES, IDEA, etc.). The server responds to this with a *server-hello* message that provides the connection-id, the subset of ciphers it can accept, a key exchange method (e.g., RSA, fixed Diffie-Hellman, etc.), authentication algorithm (MD5 or SHA1), and some other parameters.

2. Mutual Authentication: Following a server-hello, the server establishes its credentials to the client in one of several ways. One such method is to send a pre-signed server certificate that also serves to inform the client of server's public key, which will be needed for subsequent exchanges. The server may optionally request client certificate also, to which the client responds either with a certificate or with a no-certificate alert.

3. Secret Key Exchange: The client selects a secret key for bulk data exchange (using the selected cipher such as RC4, DES, etc.), encrypts it using server's public key and sends it out to the server. The client verifies the master key by sending connection-id encrypted with this key. In turn, the server verifies by sending encrypted challenge data followed by encrypted session-id. The connection id is changed periodically to avoid replay attack. The session-id can be cached by the server for subsequent use.

Following the handshake protocol, the data exchange takes place using the chosen private key bulk-data encryption algorithm. For maximum security, the recommended key length for bulk data encryption is 128 bits, although many sites currently use smaller keys. If a block cipher is used, the data is encrypted in blocks of 64 bits. SSL also decomposes large messages into fragments of size at most 16 KB in order to facilitate message authentication. Each fragment is appended with the message authentication code (MAC) which is computed as follows; a copy of the fragment is first appended with the secret key and message sequence number, and then whole thing is reduced to a fixed size MAC using the negotiated secure hash function (MD5 or SHA1).

In the classical OSI reference model, SSL sits in the presentation layer, and thus can be used transparently by the application layer components such as Telnet, FTP, HTTP, etc. for secure communication across a network irrespective of the application level message size. Unlike IPSEC which secures the network at the IP layer, SSL only secures a given client-server session.

Next, we briefly discuss the computational costs associated with using SSL. The handshake protocol is obviously expensive in terms of the number of message exchanges; however, a much more time consuming activity is the public key encryption and decryption needed for protecting the exchange of the private key. The private key encryption/decryption and secure hashing functions are also quite expensive operations, and they all tend to make secure transactions far slower than the unsecured ones.

RSA is by far the most commonly used public key encryption algorithm in practice currently, and we describe its essential characteristics here briefly [11]. RSA belongs to the class of exponentiation ciphers and involves exponentiation of two large integers modulo another large integer.

Let (e, d) denote the encryption/decryption key-pair, and $N = pq$ where p and q are large primes. Then, encrypted message C and plain-text message M are related as follows:

$$C = M^e \mathrm{mod} N, \qquad M = C^d \mathrm{mod} N \tag{1}$$

Here the public key consists of the pair (N, e) and the private key consists of the triplet (p, q, d). The strength of this cipher is essentially controlled by the difficulty of factoring N into the factors p and q, which necessitates key lengths of 512 or 1024 bits. The algorithm to efficiently evaluate equation (1) involves unsigned arithmetic operations (multiplication, addition, shift and rotate) on large integers. Thus, processors with longer word length unsigned integer instructions will automatically provide a significant performance boost to RSA. In fact, since a 2n×2n-bit multiplication requires four n×n-bit multiplications, doubling the word length of unsigned operations will provide 4-times performance boost so long the cycles per instruction remains the same. It turns out that even with a 512-bit key length, almost 80% of the SSL handshake time may be spent in the RSA exponentiation, which makes the efficient coding of exponentiation critical for good performance. For the same reason, availability of special hardware features to expedite exponentiation can have a tremendous influence on handshake performance.

Public key algorithms are inappropriate for bulk data encryption because of very long key sizes and associated computational complexity of the algorithms. Currently, the most popular bulk data encryption algorithms used in e-commerce are DES (U.S. government's Data Encryption Standard) and RC4 from RSA Inc. There is currently an effort underway to find a suitable replacement for DES as the next U.S. standard and five algorithms are currently being investigated. Although these algorithms have different characteristics in terms of their most efficient implementations, they are all highly sequential in nature. DES encrypts blocks of size 64-bits using an initial permutation, followed by 16 steps of rather complex substitution/transposition using the key, and finally the inverse of the initial permutation [11]. Let $T_i = L_i R_i$ denote the block at ith iteration with L_i and R_i as left and right 32-bit halves. Then,

$$L_i = R_{i-1}, \qquad R_i = L_{i-1} \oplus f(R_{i-1}, PC(K_i)) \tag{2}$$

where $\oplus$ is exclusive-OR operation, f is substitution/transposition function, and K_i is the key for ith step. The key K_i is obtained from K_{i-1} by left-rotation of its left and right halves. (K_0 is the original DES key). The function PC is substitution that yields a 48-bit result. The function f expands the 32-bit block R_{i-1} into a 48-bit block, exclusive-OR's it with $PC(K_i)$, splits the result into subblocks of size 6 bits, and converts each 6-bit subblock into a 4-bit subblock using a bit substitution function ("S-box").

It is clear from this description that DES is not only highly computationally intensive but also very sequential in nature and does not have much inherent parallelism that can be exploited by modern superscalar or vector-oriented (e.g., MMX-like) architectures. Most other bulk data encryption algorithms also possess these characteristics and thus run very slow on modern microprocessors.

3 SSL Resource Requirements

The dismal SSL performance quoted in [12] is a cumulative result of a number of factors as already indicated above. Examining all these effects is beyond the scope of this paper. Instead, this paper only concentrates on the inherent cost of SSL in terms of computation and communication and attempts to find architectural features that would improve the performance.

It should be clear from the discussion above that SSL handshake is an extremely expensive operation both in terms of computational requirements (e.g., exponentiation of long integers) and in terms of user response time (about 8 message transfers between client and server each of which may experience long transport delays as it travels through the wide area network intervening the client and the server). The frequency of SSL handshake vis a vis bulk data transfer is very much dependent on the degree of sensitivity of various transactions. In a typical e-tailing environment, a customer browses through the product descriptions using non-secured transactions. Eventually, when the customer decides to purchase some product, secured transactions are used to exchange sensitive information such as the credit card number. In this application, a SSL handshake will occur for one or a few secure transactions. On the other extreme, in certain environments such as online banking or stock trading, even the “browsing” transactions (e.g., checking account balance or checking portfolio) may be considered too sensitive to be run in non-secure mode. In such a case, SSL handshake is necessary only when the customer initially logs into the service, and from then only encrypted data transfer ensues. (In order to minimize possibility of long exposures, the private key may be renegotiated occasionally.) In these “one-time handshake” situations, the cost of SSL handshake becomes irrelevant and one must concentrate on the efficiency of bulk data encryption/decryption. Accordingly, we consider the following 3 cases in studying SSL performance:

1. SSL handshake followed by a very small data transfer. Indeed we consider only a 30 byte data transfer (the smallest possible using our traffic generation tool). This case is intended for studying the handshake performance only.[2]
2. SSL handshake followed by encrypted transfer of a huge web-page. The web page size chosen here was 1 MB. In this case, the handshake overhead becomes negligible, and hence we get to see only the bulk data encryption performance.[3]
3. SSL handshake followed by 36 KB web-page transfer. This size was determined to be the average size based on data from several large e-commerce sites.

In order to characterize these cases, we ran experiments where the clients were repeatedly retrieving web-pages of the three sizes listed above. The traffic generator provided the capability to run the request with or without SSL. In the SSL case, the public-key encryption used 512-bit

---

[2] Even a very small transaction such as providing credit card number amounts to a POST operation with a few Kbytes of data; therefore, the 30 byte case is *not* representative of small information transfer case.

[3] This case was intended to approximate a one-time handshake case since the traffic generator used did not have the capability of a true one-time handshake.

RSA and the private key encryption used 128-bit RC4. In all cases, the web-pages were static files; in fact, all clients were attempting to retrieve the same web-page. All experiments were run an Intel Pentium III Xeon based SMP server running NT4.0 O/S and Microsoft Internet Information Server (IIS) v4.0 web server. The original server had 2 MB of L2 cache. In order study the impact of L2 cache size, experiments were also run with 1 MB and 512 KB L2 cache. The next section discusses the results obtained from these experiments.

In order to get a clear picture of the transaction processing operations, we analyzed the break-up of the transaction time. Assuming one SSL handshake per HTTPS transaction, the total transaction time can be broken into the following three components:

T_h**:** SSL handshake part.

T_e**:** Data encryption/decryption part (including its cache impact).

T_n**:** Non-SSL part (everything else).

Note that in case of static web pages, encryption/decryption will cause cache pollution and network transfer of data from the cache which would increase CPI. All these impacts are captured in T_e, and do not affect T_n, which is simply the transaction time if SSL is not used.

Experiments were conducted in the setup described earlier using a number of sizes for the requested web pages. It was originally postulated, and then verified using these experiments that the total transaction processing time at the server depends almost linearly on the requested file-size. That is, the total processing time can be expressed as a function of file-size η (in Kbytes):

$$T_x = T_{x0} + \eta\, T_{x1}, \qquad x = s \text{ for SSL case}, \quad x = n \text{ for non-SSL case} \tag{3}$$

where T_{x0} is the constant part, and T_{x1} is the per-Kbyte part. Since the SSL handshake cost is independent of the response size, whereas the cost of encryption/decryption is proportional to the response size, we can estimate the T_h and T_e components as follows:

$$T_h = T_{s0} - T_{n0}, \qquad T_e = (T_{s1} - T_{n1})\eta \tag{4}$$

Note that $T_h + T_e = T_s - T_n$, as desired. Based on a dual processor, 2 MB L2 measurements, we determined that:

$$T_s = 1.51 + 0.04\eta, \qquad T_n = 0.24 + 0.01\eta \tag{5}$$

Therefore, for a 36 Kbyte file, $T_h/T_s = 0.43$, i.e., 43.0% of the total time is spent in SSL handshake. Also, $T_e/T_s = 0.366$, i.e., another 36.6% time is spent in encryption/decryption, and remaining 20.3% in other activities. This further implies that if SSL is not used, the per-transaction time is about 1/5th, and hence the achievable throughput is 5-times as much. Undoubtedly, this breakup will change somewhat with system parameters (e.g., number of processors, cache size, etc.); however, the numbers here give some idea of relative importance of optimizations to various parts in achieving better SSL performance.

req file size	No of procs	Throughput		path-length		cycles/inst		DBUS util	
		w/ SSL	w/o SSL	w/ SSL	w/o SSL	w/ SSL	w/o SSL	w/ SSL	w/o SSL
30 B	1	353	2496	734	49	1.94	4.13	6.7%	3.2%
	2	661	4166	698	47	2.17	5.14	18.2%	13.9%
	4	1197	6531	706	52	2.37	5.86	38.8%	30.8%
1 MB	1	12	75	35077	2828	1.18	2.39	10.0%	11.3%
	2	23	110	34896	3151	1.27	2.96	20.8%	24.3%
	4	41	144	34389	3659	1.42	3.84	37.5%	44.3%
36 KB	1	201	1120	1834	136	1.39	3.30	8.0%	7.4%
	2	345	1643	2064	158	1.50	3.85	17.6%	19.3%
	4	548	2442	2348	186	1.57	4.42	32.9%	34.6%

Table 1: Throughput, path-length, CPI and Data bus utilization (2 MB L2 cache)

4 Data Analysis and Architectural Impact

In this section, we analyze the data obtained from all the counters in our experimental setup. We also discuss the architectural implications relating to the behavior of the secure communication protocol. In this analysis, we examine the data both from the perspective of comparing SSL performance against the non-SSL case and in terms of architectural features that would help boost SSL performance (irrespective of corresponding non-SSL performance).

4.1 Overall Performance

Table 1 shows the overall performance parameters for the three web-page sizes, and varying number of processors in the server system (1P, 2P and 4P). The first parameter of interest is the achieved throughput. The goal of the experiments was to subject the server to a load such that all processors are nearly 100% utilized, and yet the number of connection errors, timeouts and retransmissions remains negligible. Although in most cases, 95% or higher processor utilization was achieved, there were some significant exceptions where maximum achieved load was only in 60-80% range. Table 1 does not directly list the achieved throughput; instead, to ensure a fair comparison, it lists the throughput scaled up to 100% processor utilization, i.e., achieved throughput divided by the observed processor utilization. The third column in Table 2 shows the ratios of scaled throughput for SSL and non-SSL case. It is seen that for 1P, non-SSL throughput is 5.6 to 7.1 times that of SSL throughput; however, as the number of processors increase, the ratio goes down. This is to be expected in view of the fact that more processors mean more coherency traffic in both SSL and non-SSL cases.

Table 1 also shows the path length (number of instructions retired per transaction) and processor cycles per instruction (CPI) with and without SSL. These parameters and indeed most others need to be corrected for the idle time, especially in cases where the achieved processor utilization was low. The correction is necessary since the characteristics of instructions executed during

Req	No	Rel non-SSL		L1 inst MR		L1 data MR		L2 MR	
file	of	tput	cycle	w/	w/o	w/	w/o	w/	w/o
size	procs		/op	SSL		SSL		SSL	
30 B	1	7.07	7.02	0.93%	1.77%	2.71%	5.54%	12.1%	1.9%
	2	6.30	6.29	0.65%	1.20%	3.45%	6.83%	23.1%	7.3%
	4	5.46	5.46	0.45%	0.90%	4.13%	7.40%	35.8%	10.7%
1 MB	1	6.14	6.10	0.35%	0.27%	2.46%	4.57%	11.2%	23.3%
	2	4.89	4.77	0.30%	0.10%	2.85%	5.14%	16.3%	44.0%
	4	3.49	3.47	0.22%	0.07%	2.86%	5.92%	19.7%	57.0%
36 KB	1	5.58	5.67	0.52%	0.77%	3.03%	5.00%	7.1%	8.7%
	2	4.77	5.06	0.49%	0.49%	3.03%	6.03%	13.4%	15.4%
	4	4.46	4.49	0.37%	0.31%	3.14%	6.27%	27.2%	21.4%

Table 2: Relative throughput and L1/L2 Cache miss ratios (2 MB L2 cache)

idle periods (NOPs, scanning of device interrupt vectors, maintenance activities, etc.) have very different characteristics than during non-idle periods. We excluded the idle impact by recording values of hardware counters during idle period and subtracting out an appropriate proportion of those from the non-idle measurements. It is seen that SSL can increase path length 10-15 fold over the non-SSL case, however, the CPI drops by more than a factor of 2, thereby resulting in a net increase in processing cost of about 5-7 fold. The fourth column in Table 2 shows the ratio of cycles per transaction for SSL and non-SSL case. A very close tracking of columns 3 and 4 (relative throughput and relative cycles per operation or cycles/op) gives credence to the estimated throughput scaling factors.

The small CPI for SSL is indicative of primarily computational type of workload, which immediately implies that a faster CPU core would go a long way in improving SSL performance so long as L1 is large enough to supply much of the code and data need in the computations. (However, since bulk data encryption/decryption algorithms tend to be highly sequential in nature, a wider issue width would not help; but a longer pipeline would.) It is worth noting in this regard that the performance of ordinary Web benchmarks (such as SPECweb96) does not scale very well with the core speed because of significant locking/contention issues. This advantage of SSL over non-SSL case shows up in further analysis in this section.

Figure 1 shows the throughput scaling with respect to the number of processors for the 3 file sizes. It is seen that under SSL, the processor scaling is excellent (about 1.8 from 1P to 2P and also from 2P to 4P) for both extremes (30bytes and 1 Mbytes). For 36 Kbyte file size, the scaling is somewhat poorer because of smaller spatial locality. In contrast, the processor scaling is much poorer for non-SSL case. Figure 2 shows the scaling of CPI with respect to number of processors. Again, the CPI does not change much with number of processors for SSL (unlike the non-SSL case). This means that the locking/contention and other high CPI activities do not become dominant even in a 4P case with SSL. A major consequence of these observations is using a separate server for SSL handshake (or more generally for authentication and key distribution)

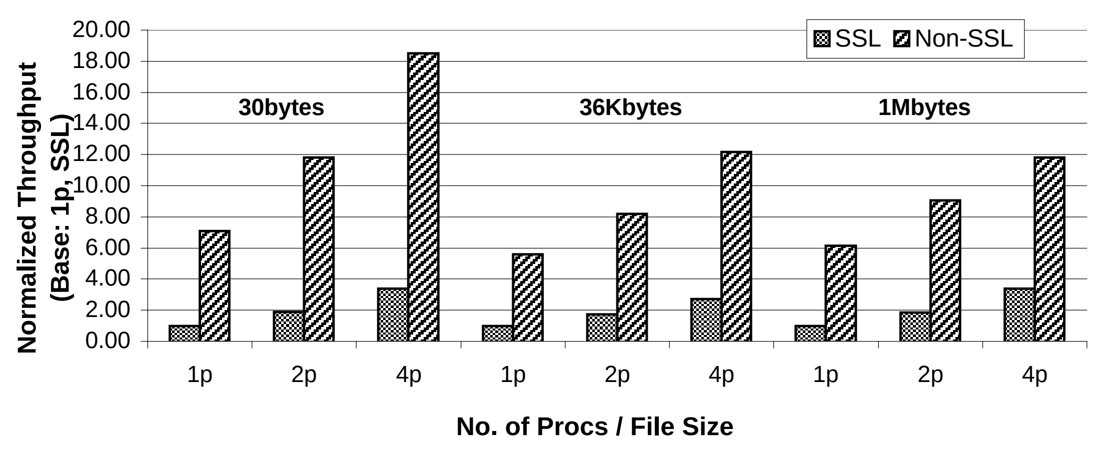

Figure 1: Relative throughput for SSL and Non-SSL cases (2 MB L2 cache)

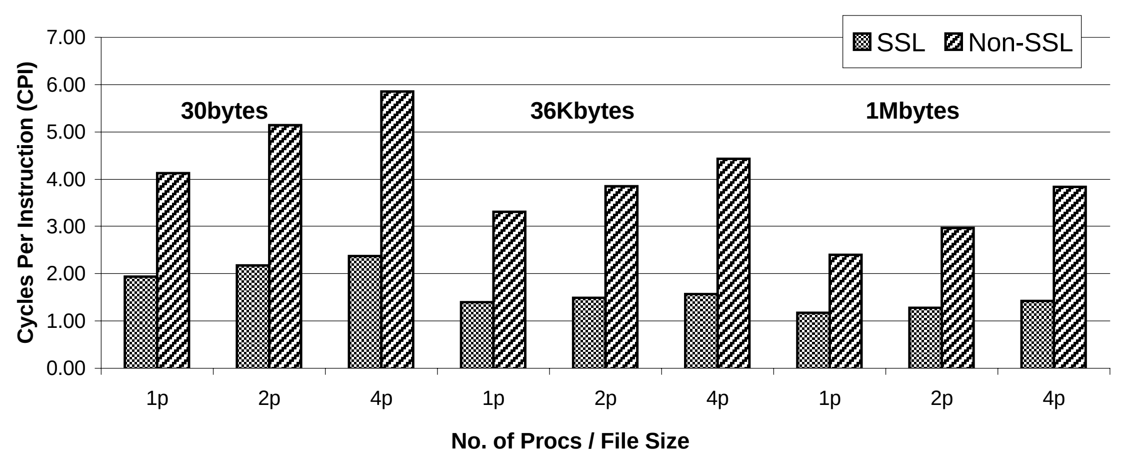

Figure 2: A comparison of CPIs for SSL and Non-SSL cases (2 MB L2 cache)

can exploit a large multiprocessor (e.g., a 4-processor SMP or 8-processor cluster) very well to handle the required computational load.

4.2 L1 Cache Characteristics

Table 2 also shows the L1 instruction and data miss ratios. (Intel Pentium III has separate instruction and data L1 caches, each of size 16 KB, but it has a single unified L2 cache.) It is seen that L1 instruction miss ratios are very low in all cases, but L1 data miss ratios are significant. However, as a function of number of processors, the behavior of instruction and data caches is very different, as can be seen more clearly from Figures 3 and 4. The instruction miss ratio generally decreases with number of processors, but the data miss ratio goes up. This is to be expected because more processors allow a better sharing of code, but the data footprint and coherency misses in data cache increases with the number of processors. In any case, the effect of number of processors in not very pronounced because of small size of L1. A larger L1 would perhaps show more degradation in miss ratios as a function of number of processors.

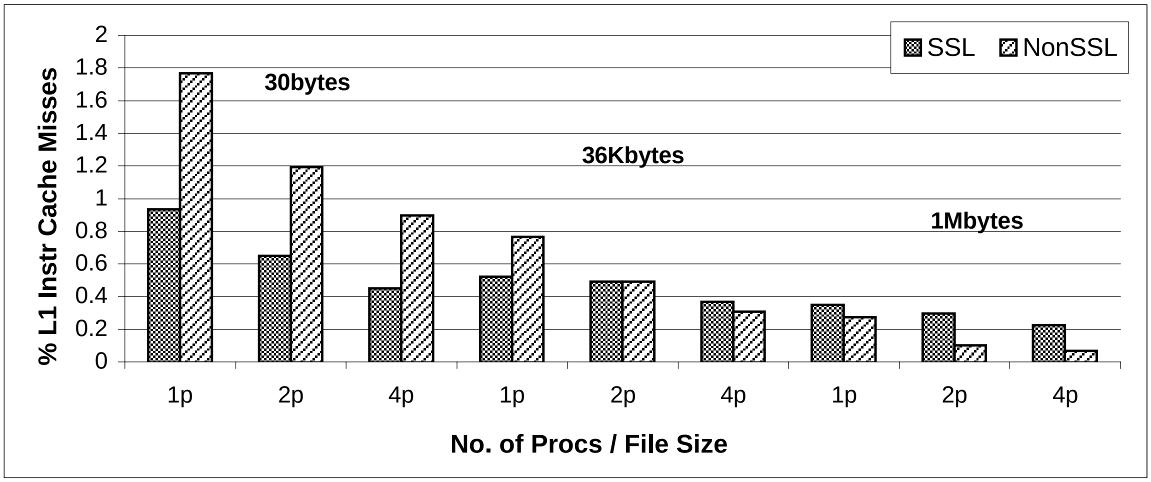

Figure 3: Scaling of instruction L1 miss ratios with file size (2 MB L2 cache)

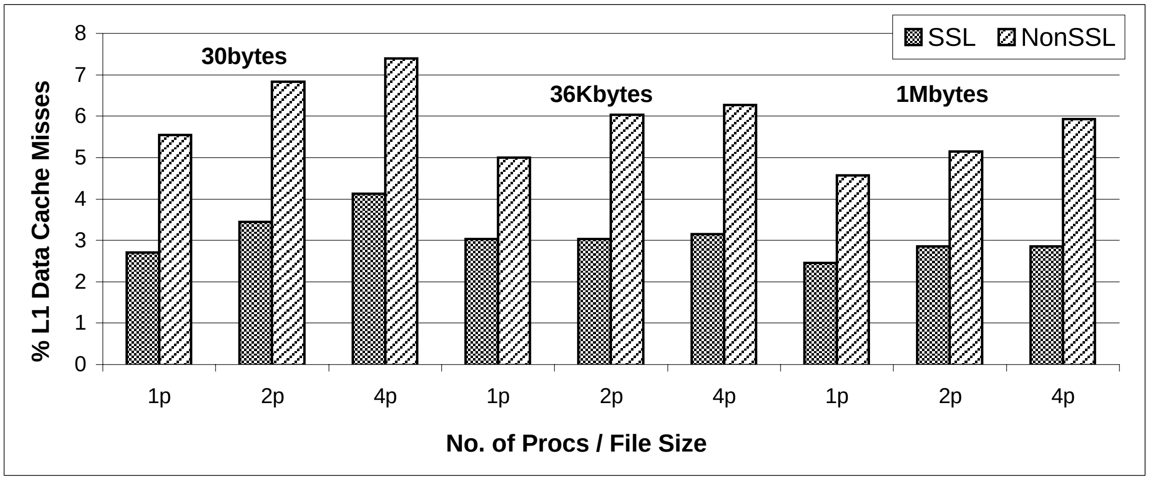

Figure 4: Scaling of data L1 miss ratios with file size (2 MB L2 cache)

For the SSL handshake case (30 byte file sizes), the miss ratio for instruction and data seem to be much lower (about one-half) in the SSL case compared to the non-SSL case. Although the data miss ratio retains the same behavior for all file sizes and processor configurations, the instruction miss ratio becomes very poor with the SSL traffic for bulk transfers (1MB file sizes). This behavior of L1 cache can be explained as follows. The lower data cache miss ratio in case of SSL is primarily because of the frequent reuse of the data during the encryption and decryption

req file size	No of procs	Throughput		path-length		cycles/inst		L2 MR	
		w/	w/o	w/	w/o	w/	w/o	w/	w/o
		SSL		SSL		SSL		SSL	
30 B	1	340	2410	750	49	1.97	4.33	16.9%	3.6%
	2	652	3929	673	47	2.29	5.39	31.7%	8.1%
	4	1142	6292	674	53	2.60	6.00	37.0%	11.7%
1 MB	1	12	71	34695	2845	1.23	2.48	14.1%	28.9%
	2	21	107	36060	3283	1.30	2.82	19.4%	57.2%
	4	36	134	35501	3881	1.55	3.90	22.6%	63.1%

Table 3: Throughput, path-length, CPI and L2 miss ratio (1 MB L2 cache)

process. For the instruction stream, the locality in the instruction relating to the handshaking process is very high, but there is not much temporal locality in the bulk transfer case. Moreover, the working set of instructions in the bulk transfer case probably does not fit within the L1 cache. This also implies that to improve bulk data encryption performance, a larger instruction L1 would help. A larger data L1 should also help because of low CPI of SSL computations which makes misses out of L1 very expensive. In particular, in Pentium III Xeon a miss out of L1 must encounter additional 12 clock cycles of delay (assuming that the data is available in L2).

4.3 Performance Scaling with L2 size

Tables 3 and 4 show the SSL and non-SSL throughput, path-length and CPIs for 1 MB and 512 KB L2 cache. Table 5 summarizes all the data by showing the throughput and L2 misses per transaction for 1 MB and 2 MB cache cases relative to that for the 512 KB case. It is seen that impact of L2 size on SSL performance is very modest for 1P and 2P cases, but significant for 4P case. In contrast, larger cache helps significantly for the non-SSL case for all 3 cases (1P, 2P and 4P). This is to be expected considering the fact that without SSL, the L2 working set size is much smaller and even with the cache pollution caused by TCP checksums, a 2MB cache is large enough contain the working set. In contrast, SSL working set is much bigger and data encryption not only brings in the entire web page in the cache, but also modifies all of it. It is expected that if the cache size is made really large (e.g., 8 MB), SSL will also see a considerable drop in miss ratio. However, such a large cache is unlikely to be seen without a substantial increase in the CPU processing power. An increase in CPU processing power implies an increase in working data size, and so the *effective* cache size is unlikely to increase much in future. In view of this, it is a reasonable conclusion that large cache size does not help SSL performance.

4.4 L2 Cache Characteristics

In this subsection, we examine L2 miss ratios in Tables 2, 3, and 4 more closely. This data is shown more clearly in Figures 5-7. Figure 5 shows the scaling of miss ratios as a function of number of processors and file-size for the 2MB L2 cache size. In contrast, Figures 6 and 7 show

req file size	No of procs	Throughput		path-length		cycles/inst		L2 MR	
		w/	w/o	w/	w/o	w/	w/o	w/	w/o
		SSL		SSL		SSL		SSL	
30 B	1	339	2284	714	49	2.07	4.52	16.7%	5.0%
	2	642	3776	694	48	2.25	5.47	28.5%	9.7%
	4	1084	6079	650	52	2.84	6.28	44.7%	13.2%
1 MB	1	12	70	35380	2785	1.21	2.57	15.6%	32.7%
	2	22	97	34737	2971	1.32	3.47	20.3%	46.1%
	4	35	124	37246	3952	1.56	4.11	25.7%	66.3%

Table 4: Throughput, path-length and CPI and L2 miss ratio (512 KB L2 cache)

Req file size	No of procs	Throughput				L2 Misses per trans			
		SSL		Non-SSL		SSL		Non-SSL	
		1 MB	2 MB	1 MB	2 MB	1 MB	2 MB	1 MB	2 MB
30 B	1	1.004	1.042	1.055	1.093	1.019	0.601	0.696	0.370
	2	1.016	1.030	1.040	1.103	0.934	0.821	0.830	0.710
	4	1.054	1.104	1.035	1.074	0.874	0.853	0.891	0.835
1 MB	1	1.003	1.031	1.015	1.063	0.881	0.715	0.910	0.726
	2	0.980	1.033	1.098	1.131	0.976	0.808	0.977	0.786
	4	1.047	1.180	1.082	1.157	0.875	0.721	0.938	0.843

Table 5: Scaling of Throughput and L2 misses/op relative to 512KB L2 cache

the scaling with respect to number of processors and L2 cache sizes for the two extreme cases (SSL handshake dominated and encryption dominated situations). It may be noted that L2 miss ratios are very high without SSL even though we have an extremely simple situation (i.e., a single static web page that is being requested by all clients). This is especially true for large web pages, as shown in the 1 Mbyte file-size case. For example, with 512 KB L2 cache, the miss ratio is 66% for the 4 processor case. For 2 MB L2, this reduces to 57%, which is still very high. Such a behavior is also observed in simple Web benchmarks such as SPECweb96. One major reason for high miss ratios is high degree of locking/contention in TCP processing. The other reason is the cache pollution because of TCP checksums, since checksum computation involves a sequential reading of the packet data and essentially a "one touch" access to it. Currently, TCP checksums are typically performed by the main processor, but with newer NICs and appropriate support in the operating system, the trend is to offload this functionality to the NIC itself. In particular, Intel Gigabit NICs operating under the upcoming Windows 2000 O/S can offload TCP checksums to the NICs. This offloading should reduce non-SSL miss ratios considerably. We plan to conduct experiments shortly to confirm this.

It may be noted that the heavy computational workload of SSL helps in reducing the L2 cache miss ratio. Unfortunately, SSL processing itself has certain features that would lead to high L2 cache miss ratios. One of these is the sending and reception of multiple small messages, each of

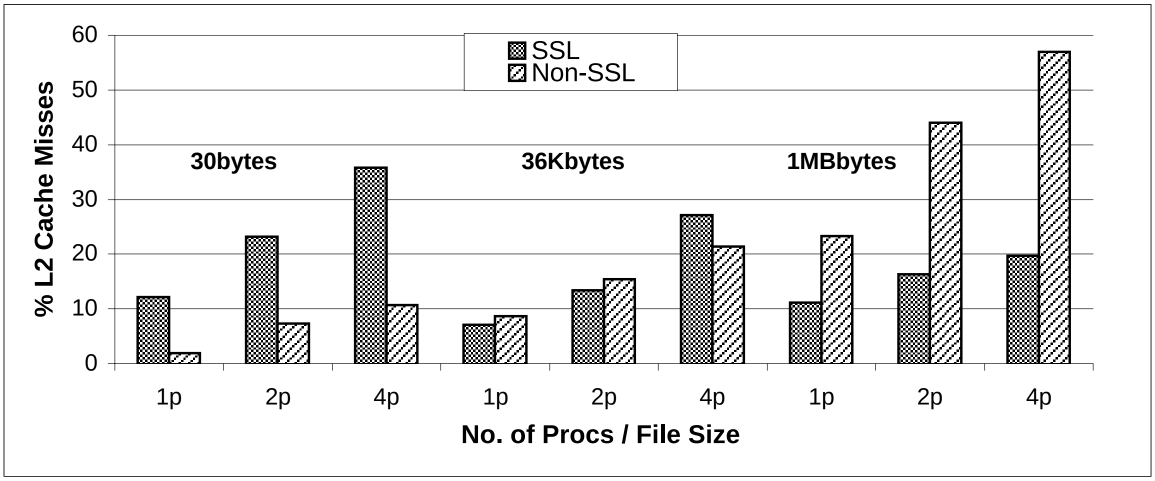

Figure 5: Scaling of L2 miss ratios with file size (2 MB L2 cache)

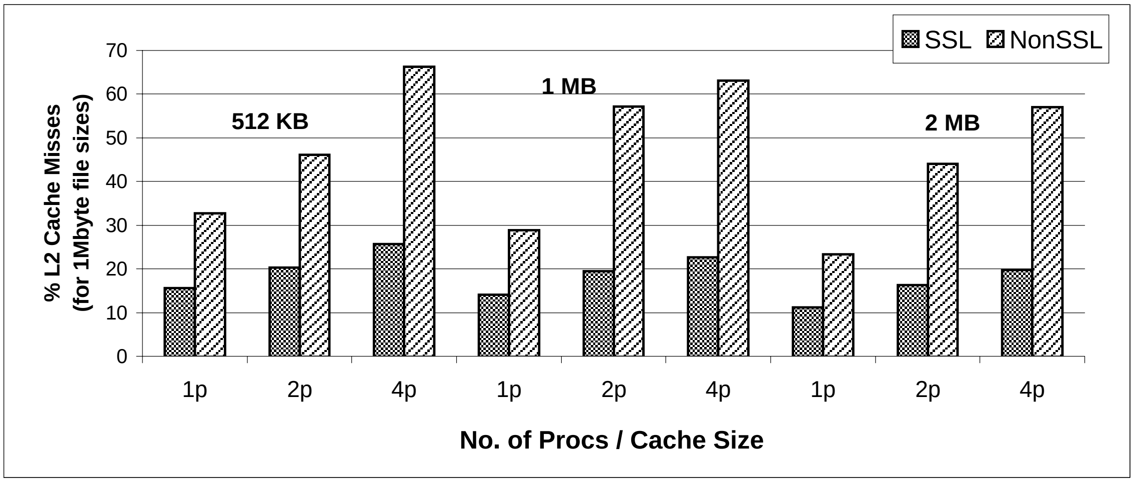

Figure 6: Scaling of L2 miss ratios with L2 size for 1 Mbyte file size

which requires search in transmission control block (TCB) data structure, TCP checksum and TCP header processing. These operations have small locality, as is seen from the performance of current Web benchmarks. Consequently, SSL handshake (30 byte case) shows very high L2 miss ratios; in fact, much higher than the corresponding non-SSL case. (Actually, SSL handshake has high temporal locality within a small range of addresses, which is already exploited at the L1 level, and not much locality is left at the L2 level.) The other aspect relevant for L2 miss ratio is the cache pollution due to bulk data encryption and decryption. As with TCP checksum calculations, the entire packet must be read in the L2 cache for encryption/decryption; however, unlike checksum calculation, the data is now modified thereby increasing chances of coherency misses. On the other hand, because of the sequential nature of encryption and decryption algorithms, bulk data encryption has a very high spatial locality even over large spans, which is exploited by the L2

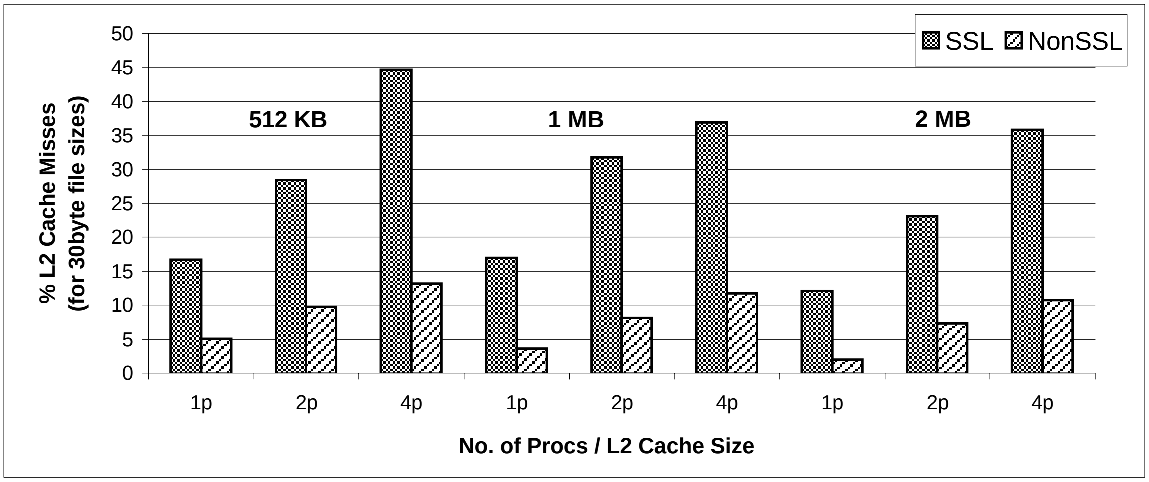

Figure 7: Scaling of L2 miss ratios with L2 size for 30 byte file size

cache. This explains why the L2 miss ratio is smaller for SSL case than for non-SSL case for 1 Mbyte file size. In the 36 KB file-size case, both effects are present and they almost cancel each other out.

A detailed look at L2 cache data suggests significant L2 cache pollution caused by the "one-touch" processing in encryption/decryption, which displaces data having more favorable reference patterns. Furthermore, in case of encryption, the encrypted data must be sent out over the PCI bus. Thus if the encrypted segment is still sitting in L2 cache when the DMA transfer is requested, it would result in a hit modified (HITM) condition in the cache and a consequent data transfer from the cache. The latencies for dealing with these "dirty hits" are known to be very high. Larger data segments could cause back-to-back HITMs, which are known to be problematic on current Pentium platforms due to extra latencies and dead clocks on the data bus. Note that certain one-touch operations (such as TCP checksum or TCB scan) occur regularly in network oriented workloads even without SSL and affect the performance significantly. For example, it is estimated that in SPECweb96, most of the read misses occur due to TCP checksums, and therefore, the performance can be improved significantly by off-loading TCP checksums to NICs.

In view of the above, a larger L2 cache would help so long as the performance is not I/O latency limited. More important, an architecture that reduces L2 cache pollution should result in a substantial performance improvement. For example, Intel Pentium-III architecture includes special prefetching instructions that can bring data from memory directly into L1 cache without first placing it in the L2 cache. SSL implementation can be coded to take advantage of this by bringing data for encryption/decryption directly into the L1 cache. This should help significantly because (a) other, more frequently referenced data in L2 is not displaced, (b) "dirty hits" and associated problem of long latencies in IO and dead clocks on the data bus are avoided for L2, and (c) L1 being much smaller and faster, the probability of a dirty hit in L1 is considerably smaller. If a significant amount of cache pollution associated with SSL processing can be avoided

req file size	No of procs	branches per instruction		fraction mispredicted		BTB Inefficiency	
		w/ SSL	w/o SSL	w/ SSL	w/o SSL	w/ SSL	w/o SSL
30 B	1	0.116	0.210	0.116	0.219	0.576	1.000
	2	0.103	0.222	0.127	0.202	0.707	0.999
	4	0.084	0.260	0.217	0.157	0.999	0.764
1 MB	1	0.058	0.158	0.067	0.062	0.348	0.475
	2	0.060	0.202	0.059	0.029	0.305	0.273
	4	0.059	0.288	0.052	0.019	0.281	0.165
36 KB	1	0.068	0.191	0.099	0.141	0.475	0.744
	2	0.064	0.203	0.099	0.102	0.476	0.578
	4	0.042	0.286	0.181	0.056	0.875	0.325

Table 6: Comparison of branch frequency, misprediction, and BTB inefficiency

this way, the prefetch instructions should be provide a significant gain in SSL performance. This needs to be verified by actual coding or experimentation.

4.5 Branch and Prediction Behavior

The behavior of control dependencies (branch frequency, branch misprediction rate, and BTB inefficiency) for SSL and non-SSL transactions are summarized in Table 6. It is observed that the frequency of branches in the SSL-based transactions is about one-half of that of the non-SSL-based transactions for the handshaking case, and becomes much less for the bulk transfer case. This fact indicates that there are less control dependencies in the SSL-based transactions. For the bulk transfer case, the encryption process is inherently sequential in nature with minimal branches. Lower control dependencies is another reason for high hit ratio in L1 and lower CPI in case of SSL. The low frequency of branches in SSL can enable exploitation of high degree of pipelining in the processor architecture.

Upon comparing the misprediction rate (proportion of mispredictions per branch instruction retired), it is observed that in case of the 4P configuration, the misprediction rate with SSL is always higher than that of the non-SSL case. To investigate further, we analyzed the proportion of branches that the branch target buffer (BTB) did not predict (we term this as BTB inefficiency, which are listed in the last two columns of Table 6). We observed that the BTB is highly inefficient for 4P cases. For the 1P and 2P cases, it is observed that for the handshaking operations, the misprediction rate with SSL is lower than the non-SSL case. However, in the bulk transfer case, the situation is reverse. This fact also explains a part of the behavior of L1 cache misses discussed earlier. Thus better branch prediction algorithms need to investigated for helping SSL transactions. However, it is cautioned not to use an overly complex branch predictor for SSL transactions since the frequency of branches in these cases are low.

req file size	No of procs	SSL first-byte Response time			Non-SSL first-byte Response time		
		2MB L2	1MB L2	512K L2	2MB L2	1MB L2	512K L2
30B	1	558	579	602	63	67	70
	2	384	401	410	70	62	62
	4	350	348	353	43	44	17
1MB	1	272	212	274	25	39	53
	2	222	224	222	28	33	49
	4	198	209	211	32	51	58

Table 7: Time in milliseconds to receive the first byte of web page

4.6 System-Level Issues

The increase in processing requirements under SSL is not confined to the core only. The nature of SSL processing has a number of other effects as well. In particular, following the trend for L2 miss ratio, the overall bus traffic increases for the SSL handshake, but actually goes down for the data transfer part. (See last two columns in Table 1.) This difference is most apparent for the 1P case. In the multiprocessor case, the effect is much less obvious because of substantial coherency traffic in both SSL and non-SSL cases. Again, for the average sized file case, there seems to be no significant difference in bus traffic with or without SSL.

With a small file-size, the network traffic increases substantially under SSL as a result of about 8 message transfers per handshake plus a single data transfer. This is a 9-fold increase in number of packets to be handled. However, since the throughput also drops by a factor of 5-7, the overall effect is not very substantial. Furthermore, for a more realistic data transfer size, this overhead will go down further since the data part itself will involve a few packets (assuming maximum packet size of 1460 bytes). Thus the overall impact of SSL on required network bandwidth is not significant. This is in contrast with the media reports that seem to indicate a big increase in network bandwidth requirements. Those reports compute bandwidth requirements assuming the same offered load under secured and unsecured traffic — the increase is irrelevant since the processor cannot handle the load anyhow. However, advancements in NIC technology such as interrupt batching, NIC checksums, larger NIC buffers, larger PCI transfer sizes, would have a positive impact. In particular, since SSL handshake data (512 bytes) and perhaps also the exchanged data (a few KB) are small in size, efficient transfer of small packets and interrupt batching are clearly a plus.

One further issue in Microsoft's NT environment has to do with 0-copy vs. 1-copy sends. NT can do 0-copy sends of static files but is limited to 1-copy send of dynamic data. Given static original data, the SSL encryption turns it into dynamic data, which must suffer a user-space to kernel space copy. This increases context switches and results in higher bus/memory load, all of which contribute to poorer performance. A 0-copy I/O solution (such as supported by the VI architecture [4]) would avoid this problem.

Table 7 shows the transaction response time experienced with and without SSL for the two extreme cases of very small and very large data transfers. The reported time is for the duration from the point the request is sent by the client to the point when the first byte of the response is received. The main reason to include the time to first byte (as opposed to time to last byte) is that it hides the impact of requested file size and thus makes the 30B and 1 MB cases comparable. As expected, the L2 cache size does not have any important role to play here. The most significant result here is that the response time increase by a factor of about 10 under SSL! If we were to look at the time to last byte for a 1 MB file size, we find that a response time of 1-2 secs without SSL (which is quite tolerable) increases to 10-15 secs with SSL (which is large enough to result in significant abandonment and retries).

It may be noted that the response time is significantly higher for 30B case because the handshake involves about 4 round-trip delays between the server and the client. With a large file transfer also involved, the competing connection setups go down drastically and the TCP data transfer becomes much more efficient (by virtue of slow-start mechanism). This results in a significant drop in queuing delays and hence a drop in first-byte response times. A consequence of these observations is that the client response time can be improved by reducing the frequency of SSL handshakes. It is also noted that the response time improves with increasing number of processors. Given the positive scaling of throughput with number of processors, the decrease in response times with number of processors is expected. Basically, the increased processing power reduces CPU delays much more than the increase in queuing delays at other resources due to increased throughput.

5 Architectural Inferences

With a rapid expansion of e-commerce and the corresponding privacy/security concerns, the use of SSL (or the transport level security or TLS, as the post3.0 SSL version is called) is also expected to escalate rapidly. In parallel, IPSEC implementation in the NICs has been receiving a big push from the vendors. It is expected that IPSEC capable NICs will soon become quite inexpensive thereby fueling the rapid deployment of IPSEC in the Internet. In view of these developments, it is essential to examine the entire stack — from low-level architecture up to the application level — to find ways of making secured communications run faster and better. In this section, we briefly outline work that needs to be taken to make secured communications a success.

As stated in section 4.2, some of the current processors already contain features that could be exploited for superior performance in secured communications. We believe that the performance boost obtainable from a recoding of security algorithms that take advantage of such instructions (for array processing and for L2 cache pollution avoidance) may be sufficiently significant to deserve a quick evaluation. In the longer run, processors could provide special instructions that speeds up encryption/decryption. A factor of 2 reduction in CPI under SSL (along with a large increase in path-length) is a clear indication that SSL workload is primarily computational and could be speeded up via special prefetching and computational instructions. A "security coprocessor" or a special purpose processor that sits lower in the architectural hierarchy (e.g., a specialized I/O processor) may also be considered.

As mentioned in section 4.2, a larger L1 cache might help SSL processing. This could be verified using cache simulators exercised using an address trace of SSL transactions, but this has not been done thus far. If larger L1 indeed gives a significant performance boost, future generations of processors may consider tradeoffs between a larger L1 size vs. other area-intensive parts of a processor (e.g., branch predictors).

L2 cache pollution because of "one touch" processing by SSL can also be avoided by using an architectural solution. This solution uses an auxiliary L2 cache, henceforth called $L2'$, for caching one-touch addresses. $L2'$ is a peer of regular $L2$ cache, but could be much smaller in size. On first reference, a cacheline is brought into $L2'$ only. On subsequent references, the cacheline is moved to $L2$. Thus, one touch cachelines stay in $L2'$ and others move to $L2$. This architecture uses $L2$ more effectively. It remains to see what an appropriate $L2'$ size should be relative to $L2$. If the size is small, $L2'$ could be relatively fast, which keeps the dirty hit problem to minimum. The main advantage of this solution is that it does not require any application re-coding. The performance of this architecture can be studied using available bus traces (with L2 turned off) and a cache simulator.

Section 2 indicated that long-word length unsigned and vector operations are highly beneficial to RSA but not DES/RC4. Consequently, SSL handshake will see a significant performance boost from such instructions especially in 64-bit processors, but the data exchange part will not. An immediate consequence of this is that in environments where SSL handshakes are infrequent (e.g., SSL handshake only at the time of a customer login to an e-commerce site), the performance boost to security processing between 64-bit vs. 32-bit or vector operation capable vs. those that don't have vector capabilities may not be very substantial. Specialized instructions that specifically accelerate DES/RC4 type of algorithms may be needed to accelerate bulk data encryption/decryption.

6 Conclusions

In this paper, we have done an experimental analysis to study the impact of SSL transactions on the Internet server architecture. It is well-known that the server performance degrades considerably for SSL transactions compared to the non-SSL case. The counter measurements obtained through our experimentation give a comprehensive view about the reasons for this performance degradation. In addition, we have analyzed the data to identify important architectural issues affecting SSL performance. Some of these issues are as follows.

- Processors with higher core frequency will improve the SSL performance.
- A processor with high pipeline depth can improve the performance of of SSL transactions, whereas the increase in the issue width may not provide any significant performance improvement, especially in cases where the performance is dominated by bulk data encryption.

This behavior is because of the existence of high sequentiality and low control dependencies in SSL code.

- Increasing the size of L1 cache will have a positive impact on the SSL performance.
- The frequency of branches are low, and the efficiency of BTB is also low. Thus a complex logic and large BTB for branch handling will not be beneficial for SSL transactions.
- Increasing the size of L2 caches to any reasonable extent will have minimal impact on the performance of SSL transactions.
- SSL handshake and encryption/decryption of large web-pages has very good scaling with respect to the number of processors in a SMP system, which may promote the use of 4-way or 8-ways systems for these applications.

These observation are useful for designing servers for use in the e-commerce environment and also for directing further studies to support SSL transactions more efficiently. In future, we will continue our effort in further identifying the performance bottlenecks and techniques for architectural improvement in e-commerce environment.

References

[1] J. Almeida, M. Dabu, A. Manikutty, and P. Cao, "Providing differentiated levels of service in web content hosting," Proc. of workshop on internet server performance (WISP), June 1998.

[2] G. Banga, P. Druschel, and J.C. Mogul, "Better Operating System features for faster network servers," Proc. of workshop on internet server performance (WISP), June 1998.

[3] G. Banga and P. Druschel, "Measuring the capacity of a web server," Proc of USENIX symposium on Internet Technologies and Systems, Monterery, CA, Dec 1997.

[4] D. Dunning, G. Regnier, et. al., "The virtual interface architecture: a protected, zero copy user-level interface to networks," IEEE Micro, March 1998, pp. 66-76.

[5] P. Druschel and G. Banga, "Lazy receiver processing (LRP): A Network Subsystem Architecture for Server Systems", Tech Report, Rice University.

[6] A.O. Freier, P. Karlton, P.C. Kocher, "The SSL Protocol, V3.0", IETF draft at www.netscape.com/eng/ssl3/draft302.txt.

[7] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, Nov 1998.

[8] D.R. Manfield, G. Millsteed, and M. Zukerman, "Congestion Controls in SS7 Signalling Networks," IEEE Communications Magazine, June 1993, pp 50-57.

[9] D. A. Menasce, V. A. F. Almeida, R. Fonseca, and M. A. Mendes, "Resource Management in E-commerce Servers," Workshop on Internet Server Performance, 1999.

[10] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled video playback over the Internet," To appear in 1999 ACM SIGCOMM.

[11] W. Stallings, *Cryptography and Network Security: Principles and Practice*, Second Ed, Prentice Hall, 1999.

[12] T. Wilson, "E-Biz bucks lost under SSL strain", Internet Week online, May 20, 1999, at www.internetwk.com/lead/lead052099.