**RC22323 (W0202-003) February 1, 2002**
**Computer Science**

# IBM Research Report

## Using SSL Session ID Reuse for Characterization of Scalable Secure Web Servers

**Ronald Mraz, Karen Witting, Paul M. Dantzig**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

**Research Division**
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Using SSL Session ID Reuse for Characterization of Scalable Secure Web Servers.*

Ron Mraz
mraz@us.ibm.com

Karen Witting
witting@us.ibm.com

Paul Dantzig
dantzig@us.ibm.com

IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532

February 1, 2002
Revised February 8, 2002

**Abstract**

This paper describes a new methodology to characterize a web servers ability to serve secure content. Specifically, measuring a server's scalability and sensitivity to dynamic variations in content due to multiple site serving and/or changes in end user interactions. This is done by defining and measuring SSL (TSL) performance with a new metric for evaluation, namely, "Percent ID Reuse". Varying this parameter and monitoring the resultant Transactions per Second will provide quantitative results describing the effectiveness of a secure server architecture to support specific types of content and/or access pattern profiles. A single value of the Percent ID Reuse at the "half serving point" provides an indication of the servers sensitivity to variations in content. To demonstrate the new technique we measure our ability to "desensitize" our content server to SSL activity by offloading it to a dedicated array of SSL Protocol specific servers and measuring their performance.

**Subject Areas:** E-Commerce, Middleware, Networks, Secure Engineering, Secure Communications, Encryption, Innovative hardware/software tradeoffs, Secure Socket Layer (SSL) High-Performance I/O architectures, Secure Networking, commercial and industrial security, Novel architectures for emerging applications, Simulation and performance evaluation, TCP/IP State Migration and Management Mechanism, Multi-Cluster Application Architecture.

## 1 Introduction

Encrypted serving results in higher computational loading, increased network latencies and distorts the traditional Internet client/server model of computing to the point that traditional analysis and measurement techniques for scaling of services cannot be applied to other content with repeatable success. Typical analysis is done for a specific set of content and site parameters such as encryption strength, specific server architectures, and accelerator hardware is applied and varied. This is not

*Submission to the 11th USENIX Security Symposium, August 5-9, 2002 in San Francisco, CA.

| | Clear Text Serving | Encrypted Text Serving |
|---|---|---|
| Client Overhead | High | Higher |
| Server Overhead | Low | High |
| Client Functionality | Low | High |
| Server Functionality | Single | High |
| Content Type | Static | Dynamic |

Figure 1: Table of Showing Content Serving Attributes of Clear vs. Encrypted Serving

a realistic way to analyze the performance of a site since content will change more often than the site architecture parameters. Additionally, Internet Service Providers typically serve content from many customers from the same array of servers. Finally, content for secure sites is typically created dynamically and has different requirements for each connection. Such that analysis with repeatable static content is an approximation, at best.

Secure content serving is fundamentally different than plain text serving. Extending existing methodologies, such as Web Stone [19] and SpecWeb99 [20] to provide a peak serving value for SSL vs. client loading may not be fully representative of a systems secure serving potential since it does not show the servers systems sensitivity to changes in content or a clients access patterns. The Table in Figure 1 provides us with some of the major difference between the two serving techniques. Both client and server overheads and required functionality are increased due the the encryption and handshake overhead. The most significant parameter is that secure content is personalized and in nearly all cases is dynamically generated at each request. This provides even higher CPU loading on the server and dramatically changes the serving dynamics over traditional static plain text serving.

Existing Secure Socket Layer (SSL) communication protocols [5] [6] are intended to provide a mechanism for the secure transfer of customer and payment information when purchasing consumer goods over the Internet. The layering of the secure operations on top of TCP/IP [17] allows the web clients to easily invoke a secure session from an HTML link. Initially, there is a high amount of set-up overhead as encryption keys are created and negotiated. Once set-up is complete, data can be transferred across the encrypted link. Hardware accelerators may be used during actual data transfer to enable encryption and decryption of data at Internet transfer speeds.

SSL implements a timer which is used to limit the "Session ID Lifetime" [6] of the encryption key state. Expiration of the timer forces a renegotiation of the encryption keys after a specified "timeout" period. Once the initial SSL negotiation is completed, there is usually ample time for a consumer to review and edit entries in an e-commerce shopping cart (as well as provide shipping and credit card information) before the keys in the connection must be refreshed. However, problems arise when an e-commerce application requires the SSL connection to be active for an extended period of time.

The requirement to uniquely create and periodically renew the encryption keys of SSL makes this protocol effective for short time duration shopping cart evaluations and purchase execution. Conversely, this periodic renegotiation makes SSL cumbersome for the following long term operations; extended duration stock portfolio monitoring as well as analysis and trading, the continuous reporting required for auction services, or the scalable connection requirements of Internet based voting. This is because each long term continuous connection requires periodic renewal of the SSL encryption keys whereas a client accessing the site multiple times for short periods of time would most likely reuse the previous key state by preserving the session ID across invocations, effectively stretching the duration of key renewal over an extended period of wall clock time.

Figure 2 provides definitions for different classes of SSL connectivity. The Gaant Chart shows there are three types of connection profiles supported by SSL today. They are the One-Time Connection, Intermittent Connection and a Continuous Connection. The One-Time connection is the

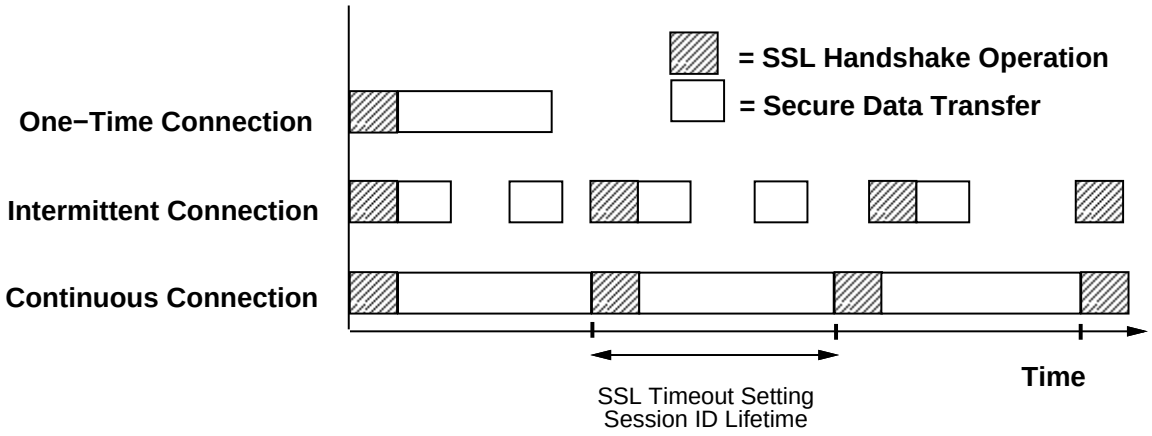


Figure 2: Definition of SSL Client-Server Traffic Pattern Profiles.

typical e-commerce shopping cart model. In this variant, the client establishes an SSL connection and then uses it to transfer information in a session that is shorter than the SSL timeout parameter. The Intermittent Connection is one in which the client makes many short connections over an extended period of time. The SSL protocol attempts to reuse the encryption session state from one TCP/IP connection to another. If reconnection occurs prior to the timeout, the session information is reused, if not, a renegotiation handshake is required. The Intermittent Connection is the model used for transactions associated with an e-commerce auction site over an extended period of time. The Continuous Connection is one where the client and server continuously transfer data over an extended period of time. This model could be applied by a brokerage information system in which an on-line trader continually monitors quotes and on-line stock transactions over an extended period of time. [1]

The timeout parameter shown in Figure 2 can be set from 5 seconds to 24 hours in SSL products as referenced in the Mozilla on-line documentation, see our reference [4]. Typical sites maintain a 100-300 second timeout default and a transaction count limit of 500 is used as well. The duration of this timer exposes a vulnerability in that someone sniffing the session is provided this amount of time to extract the SSL keys and subsequently decrypt data. Shortening the timer duration will strengthen security of the connection but increases the computational load on the servers due to additional key generation activities. Lengthening the timer duration will weaken site security but usually reduces server loading. However, it may be necessary to use stronger (more bits) encryption keys to compensate for the exposures introduced by lengthening the timer duration. Additionally, the load balancing operations are more deterministic when the traffic is of the One-Time Connection type. This connection type is characterized by high computational loads during the initialization phase followed by sustained lower demands during the data transfer phase of a connection. The other connection types provide periodic surges in computational loads when multiple connections are active for extended periods of time. When these surge periods align, there can be overloading of the server and significant delays in client-server interaction even to the point of TCP/IP connection retry and or timeout.

The version of HTTP traffic coming to the web site also impacts the performance of SSL operations. HTTP Version 1.0 [8] requires each web page component or object to be transported over a separate TCP socket connection. This most closely maps to the Intermittent Connect model shown in Figure 2 for even a single page transfer. HTTP Version 1.0 would not map to the One-Time

[1] SSL accelerator vendors have adopted use the continuous connection profile with the Session ID Lifetime Set to Infinity to measure peak SSL encryption performance.


Figure 3: A high level view of clients accessing a Secure Server.

Connection (unless only a single object is requested) and Continuous Connect Traffic Pattern of Figure 2 since a separate connection is required for each object. HTTP Version 1.1 [9] provides for multiple objects to be requested in a single session. This allows Version 1.1 to map to each of the Traffic Patterns shown in our Figure 2.

A detailed study [1] shows that if an Internet content server can serve 250-300 HTML plaintext (port 80) transactions per second, the same server can support 5-80 encrypted (port 443) pages per second depending on the frequency of handshake operations required. If the same server is used to support SSL handshake operations, the performance of the server will be further degraded since the number of handshakes this server can support is 20-40 per second. It is the frequency of the SSL handshake operation that will determine an Internet Server SSL connection capacity since a typical server takes 48.5 milli-seconds to 24.2 milli-seconds, depending on the strength of the encryption method, for processing of the TLS Handshake Protocol. Furthermore, when SSL processing coexists within the content transaction servers, due to the natural affinity of the SSL session state and the user transaction state, site operations can slow beyond the 5/250 ratio of encrypted pages to plain text pages served.

Two distinct approaches are being applied to this offload SSL problem. This first being the addition of hardware accelerator cards, such as the IBM 4197 [7], to the server to improve computational encryption capabilities. Although, this reduces the computational overhead, there is little benefit in overall latency reduction of the initial SSL handshake. A second approach uses an in-line companion proxy that intercepts SSL operations on their way to the transaction server and responds to all requests in such a way that this is transparent to the server. In this way, the transaction server is responding to all requests with plain text. These proxy servers typically contain a machine with a hardware accelerator engine(s) to offload encryption processing. This approach forces connection affinity of both content and reusable SSL state to a specific companion proxy pair and has a potentially negative impact on load balancing as the site scales to 1,000,000's of active SSL connections. Research [15] and [16] has shown that an in-line encryptor can handle large amounts of traffic.

Performance of these SSL servers cannot be helped by traditional Internet cache appliances when scalability is required. Since all SSL clients have a unique set of encryption keys for their connection, the requested objects cannot be reused from one client to another, even if the served content is static. This forces the site architect to further increase the number additional transaction servers many times that typically required to serve similar amounts of plain text content. Although, this practice is desirable from a server vendors suppliers point of view, supporting and managing the transactions from a site of several hundred servers with attached appliances can increase maintenance and can lead to configuration errors. Details of this and other methods are described in [2] and [3]. The problem of high volume SSL proxy serving is highlighted in the press at [10].

Ipivot's (Intel acquisition) [11] and [14], solution provides an inline SSL encryption engine that handles both the protocol and encryption/decryption of SSL processing. It is unclear from the marketing literature how the system scales for high volume (millions of active connections) use. Our Secure Blue architecture is different since we split the SSL processing into the In-Line Crypto Engine and the SSL Handshake Engines. Having separate clusters to support SSL Handshake and Encryption functions allows the site administrator to tune the number of Handshake Processors and encryption processors to support specific aggressive SSL ID Timeouts and encryption strength. Finally, the use of socket handoff provides a scalable solution with low overhead as well as Transaction

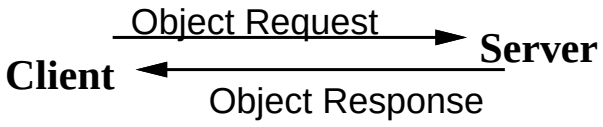


Figure 4: Definition of a Transaction.

Server connection affinity.

Alteon's solution [12] is to provide an SSL Offload Engine in parallel with the Internet Traffic Manager. All SSL related packets are sent to the SSL Offload Engine for encryption processing. HTTPS requests appear as plain text HTTP requests to the Transaction Engines. Our Secure Blue architecture is different since we split the SSL processing into The In-Line Crypto Engine and the SSL Handshake Engines. An advantage is that In-Line decryption of the packets at entrance to the site allows the the traffic manager to provide cookie based traffic management. (as cookies are encrypted as will in traditional SSL packets) Additionally, having separate handshake engines gives the site administrator to scale the number of Handshake Processors to support aggressive SSL ID Timeouts. Finally, the use of socket handoff provides a scalable solution with low overhead as well as Transaction Server connection affinity.

Therefore, it is essential that general analysis of a secure sites serving capability be done to support current and future serving characteristics. Additionally, there are Internet Service Providers that use the same set of hardware for multiple site serving. Having such a generic analysis would help allocate and balance content serving resources for anticipated and unanticipated changes in site load.

This paper describes a new methodology to characterize secure content servers for scalability and sensitivity to changes in site content heirarchary or customer interaction. This is done by defining and measuring SSL (TSL) performance with a new metric for evaluation, namely, "Percent ID Reuse". This parameter demonstrates the effectiveness of a secure server architecture to scale independent of SSL access patterns. To "desensitize" our secure server to Percent ID Reuse variations, we offload all SSL protocol activity, that was traditionally executed by Transaction Engines (and dedicated co-processors), to an array of SSL Protocol specific servers. Plots of transactions per second vs. Percent ID Reuse clearly shows how additional SSL offload engines support additional activity for any content characteristics.

The remainder of this paper is organized as follows. Section 2 provides a set of definitions to develop the measurement methodology Section 3 and provides a theoretical example of the measurement methodology. Section 4 describes the operation of the scalable array of SSL Protocol Processors. Experimental results for the methodology are presented. Section 5 provides a Summary and Conclusions. After the Acknowledgments and Bibliography, an Appendix provides detailed flowcharts of the client, secure proxy and server components.

## 2 Scalable SSL Measurement Methodology

The secure server architecture we are considering for measurement is shown in Figure 3 and is typical of many installations today. It is a generic transaction server connected to the Internet and clients access this content over the Internet via secure socket connections. The client accesses the system by a TCP/IP socket connection to 443 and requests data through the resultant encrypted channel after a dedicated handshake negotiation for a session key.

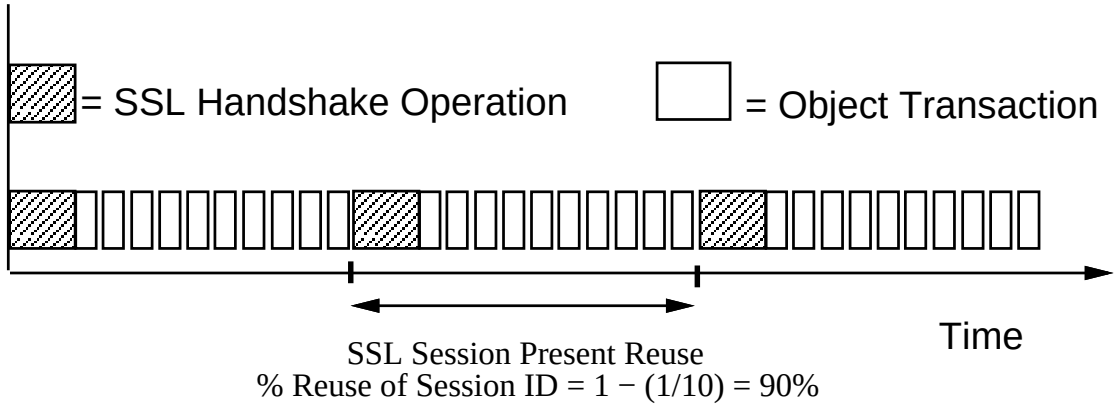


Figure 5: Graphic Representation of Percent Reuse of Session ID.

Figure 4 graphically shows our definition of a Transaction as the completion of the client's object get request and resulting transaction response. Once the Session Key is established, it is reused for each successive object transferred across the socket until it times out or a specific object count per session ID has been exceeded. At that time the Session ID is renegotiated.

Multiple object transactions may be required for construction of a single web page. The results for our object transactions can be applied to any number of web page designs since the number of objects (or an upper and lower bound in the case of fully dynamic generation) is typically known. We have resisted the temptation to define a web page as a completed transaction since our approach offers a more general metric. As an example, 10 objects could produce a single page or can represent 2 or 3 web pages of information. Other sites may use multiple objects to create a background and update a formated text block of information. Finally, the page boundary has no relevance to the Session ID timeout parameter.

We define "Percent Reuse of Session ID" by the following formula,

$$\%\, Reuse\, of\, Session\, ID \;=\; 1 \;-\; \left[\frac{\#\, SSL\, Handshakes}{\#\, Object\, Transactions}\right]. \qquad (1)$$

This formula is represented in Figure 5. Given that we serve 10 objects per SSL Handshake Operation the Percent Reuse of Session ID is 90after the initial use. We see that the typical number of objects served to construct a single web page is on the order of 18. Therefore, the typical operating range of a secure website is around 96 to 98 percent Session ID Reuse. We will show that that this is also the area of greatest change in Transactions per Second.

The methodology we are proposing is to vary the Percent Reuse of Session ID from 0% to near 100% and record the peak sustained serving rate of transactions that can occur over a period of 30 seconds. This produces a curve where we have a rising exponential to the point of plain text serving. Plain text serving being the limit of encrypted serving capability of the server by definition. [2]

In reality, the time to execute the SSL handshake can be 2-100 times longer than the time to provide a transaction object response. Additionally, parts of the handshake require near full CPU utilization of the server for completion. If several clients are attempting a handshake at the same time, there is contention for CPU resources. Having dedicated resources for SSL handshake processing in parallel with the Transaction Server can reduce the serialization effect on the overall response time and resulting Transactions Per Second.

[2] If the SSL part of secure serving could be reduced to zero, the best one could expect to achieve, in Transactions per Second is that of plain text serving.

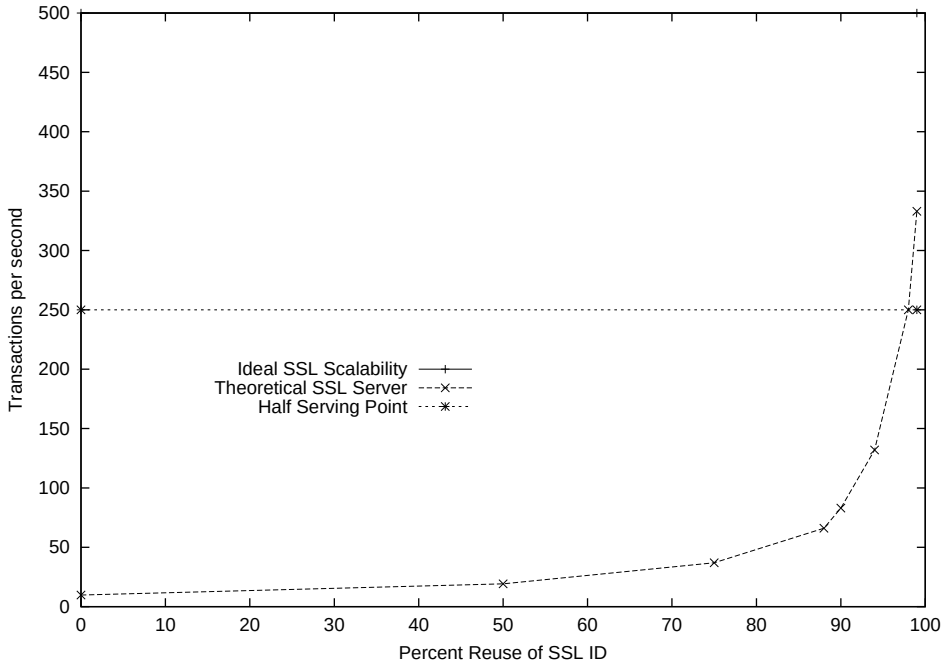


Figure 6: A theoretical graph to illustrate our measurement technique. Our hypothetical server can serve 500 object transactions per second and an SSL handshake requires 50x that of serving a single object. The Half Serving Point is at 98 Percent Reuse of SSL ID.

# 3 Scaling SSL Serving

We define a perfectly scalable SSL server as one that performs at the same level of object Transactions per Second over the entire range (0 to near 100%) of Percent Reuse of SSL ID. In reality, this means that the number of Transactions Served is independent of the SSL Handshake Time or Handshake Time equals zero.

As an example, suppose that that 500 objects can be served in a second. If a SSL handshake takes 50 times that of an object being served, we can construct a hypothetical curve displaying Transactions per Second vs. Percent Reuse SSL ID. For this discussion, we consider that the actual encryption of data is negligible to allow us to focus on the handshake overhead issue.

Figure 6 provides us with a representation of an SSL Server with Ideal Scalability and a Typical SSL Server with the above characteristics. The first step is to determine the plain text serving limit. This is defined by our theoretical definitions above and is set at 500 Transactions per Second. This is a constant value over the graph since it is independent of Percent Reuse of SSL ID. The Half Serving Point is as 250 Transactions per Second.

Using the time parameters above, we can simulate the operation of an SSL server and determine it's hypothetical Transactions per Second at representative points of Percent Reuse of SSL ID. As an example, 0 % reuse is when all objects require a handshake. This makes the serving time of each object equivalent to 51 object transactions. Since we can serve 500 objects in a second, plain text, the number of 019.2 Transactions per second. Similarly, we find 37, 66, 83, 132, 250, and 333 Transactions per Second for 75, 88, 90, 94, 98, and 99 Percent Reuse of SSL ID, respectively.

Defining the intersection of the half serving point and the Transactions per Second graph of the SSL Server is at 98%. This means that all content served with at least 98% reuse of the SSL Session ID can be served at a rate equal to or greater than 250 Transactions per Second. This information

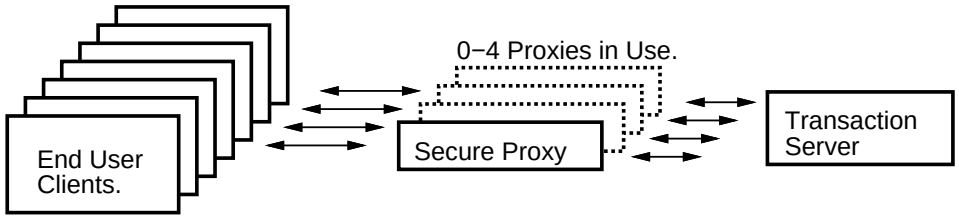


Figure 7: Scalable Proxy Architecture for SSL Performance Measurements.

can then allow site administrators to adjust timeouts and time to live counts for the SSL Session ID to insure operation in this range and maintain security within their necessary guidelines.

# 4 Experimental Testbed

A complete client, proxy and server system was crafted to allow full analysis at any point of the system to verify each systems operation. The following subsections describe the details of this system.

## 4.1 Configuration

The experimental set-up consists of 8 Windows 2000 Server workstations with Service Pack 1 installed. Each system is a single CPU, Pentium III running at 667 MHz with a system memory of 256 MB. The machines are connected with a 100 Mb Ethernet subnet with no known external routing traffic thorough the subnet.

The machines were personalized for function by custom Java servlets to be a Transaction Server (plain text serving), Secure Proxy Server, and a Custom Client (plain and encrypted requesting options). All code is compiled with Java Version 1.1.2 compiler using the Java (TM) Secure Socket Extension (JSSE) 1.0.2. Additional Windows 2000 and Unix machines were enlisted as additional Java Clients to provide additional load as necessary to force the Transaction Server to Peak Utilization.

Figure 7 shows the virtual arrangement of the system for our experiments. Experiments were done with no Secure Proxies in place. With no proxies, experiments were run with the Transaction Server providing plain text serving to determine the peak serving limit. Experiments were then run with 1-4 proxies in place to facilitate a scalable SSL serving environment.

## 4.2 Transaction Server

The content server we used was a home grown server, written in Java, which tried to simulate generation of a dynamic page. Secure and clear text requests were handled identically by this server, except for the additional SSL handshaking required for the secure communication. In order to do analysis, we timestamped the beginning and the end of each request through the server. These timestamps were saved in memory and written to a log after the test was complete. A representative flowchart and description appears in the Appendix.

## 4.3 Secure Proxy Server

The proxy server was also a home grown server, written in Java, which received encrypted requests and forwarded them, in clear text, to the configured actual server. The socket used to forward the

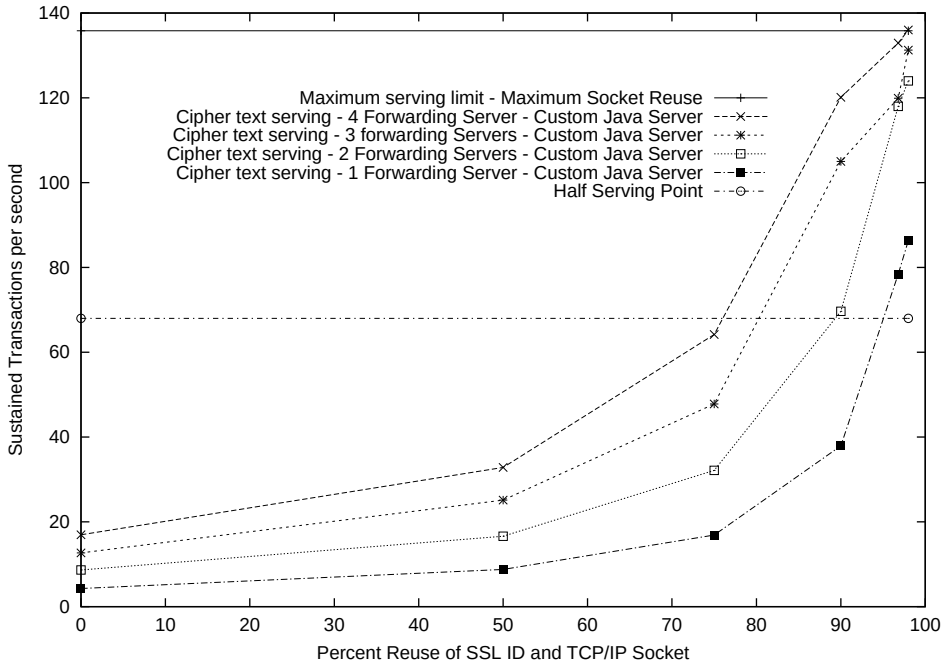


Figure 8: Performance results for a scalable secure server using proxys.

requests was reused whenever possible to avoid socket overhead for the forwarding of the requests. A representative flowchart and description appears in the Appendix.

## 4.4 Client Test Program

The program used to send requests was also written in Java and could be configured to send a variety of types of requests. In particular it could be configured to re-use the socket and the SSL session ID. A representative flowchart and description appears in the Appendix.

## 4.5 Experimental Results

Figure 8 graphs the experimental results of running our testbed with 1-4 secure proxy engines. First we determined the maximum serving capability of our Transaction Server for plain text serving. This was measured at 136 Transactions per Second. From this we can determine our Half Serving Point to be 68 Transactions per Second.

We then did performance results using 1, 2, 3 and 4 Secure Proxies to offload all SSL processing from the Transaction Server. We set the Client Test Programs and Transaction Server to access content at 0, 50, 75, 90, 96 and 98 Percent Reuse of of SSL ID. As the SSL ID expired, we terminated the socket for another access at the client. Therefore, this also represented the percent reuse of the TCP/IP Socket as well.

The results show that the addition of proxies shows a clear ability to scale SSL processing in our testbed. The addition of each proxy shows that the half serving point decreased as the number of proxies were increased as 95, 90, 80 and 77 Percent Reuse of SSL ID for 1, 2, 3, and 4 secure proxy servers, respectively. Therefore, the site using 4 proxies can handle a greater variation in dynamic content variation than a single proxy implementation can.

Our findings are relevant to web site designs of today. Reference [13] provides a example sizing

of an e-commerce site (CDnow) that contains two pages of approximately 11.1 K bytes of content and the first page contains 11 GIF images and one HTML page and the second page contains 12 GIF images and one HTML page. Within the context of HTML Version 1.1 we expect all of these objects to be requested within one HTTP session access. The 23 GIF images and the two HTML pages provide 25 objects referenced per user session. Using Formula 1, this would constitute a Percent Reuse of Session ID of 96%. This would indicate that a single proxy in our test bed could support such a site at the Half Serving Point. If the site content characteristics or user interactions change to lower this Reuse Value below 95%, the single proxy would not be sufficient to support the Half Serving Point.

# 5 Summary and Conclusions

This paper describes a methodology to characterize secure content web servers for scalability and sensitivity to changes in site content heirarchary due to multiple site serving and/or changes in end user interactions. This is done by defining and measuring SSL (TSL) performance with a new metric for evaluation, namely, "Percent ID Reuse". Varying this parameter and monitoring the resultant Transactions per Second will provide quantitative results describing the effectiveness of a secure server architecture to support specific types of content and/or access pattern profiles. To demonstrate the new technique we measured our ability to "desensitize" our content server to SSL activity by offloading it to a dedicated array of SSL Protocol specific servers and measuring their performance.

Steps in the "Half Serving Point" methodology.

1) Measure the system's sustained plain text serving limit for the specific objects to be served.

2) Measure the sustained SSL serving limit for several representative Session ID Reuse levels.

3) Determine the "Half Serving Point" for Secure Serving Operation.

The lower in percentage this value is the better the system can absorb connection anomalies such as lost connections, session retries, timeouts, etc. and maintain a sustained transaction load.

Future investigations will consider the impact of the Half Serving Point on advanced architectures for secure serving such as Secure Blue [18]. We believe that this "Half Serving Point" technique can be applied to any Secure Serving Method that requires a periodic setup and/or agreement of a session key.

# 6 Acknowledgments

Karen Witting coded the client, secure proxy and server and instrumented them to facilitate our measurements. The authors would also like to thank Paul Dantzig, Dilip Kandlur and Nagui Halim who provided management support for this work.

# References

[1] George Apostolopoulos, Vinod Peris, Debanjan Saha "Transport Layer Security: How much does it really cost?", IEEE INFOCOMM 2000, June 2000

[2] Junehwa Song, Eric Levy-Abegnoli, Arun Iyengar, and Daniel Dias "Design Alternatives for Scalable Web Server Accelerators" , Proc. of IEEE International Symposium on Performance Analysis of Systems and Software, 2000

[3] Guerney Hunt, Eric Nahum, and John Tracey "Enabling Content-Based Load Distribution for Scalable Services", Draft document, unpublished.

[4] Sean Cotter, "SSL Reference" *http://www.mozilla.org/projects/security/pki/nss/ref/ssl/-index.html*, netscape public mozilla crypto newsgroup, October, 2000

[5] E. Resorla, "HTTP Over TLS (RFC 2818)" *http://www.ietf.org/rfc/*, IETF RFC 2818, May, 2000

[6] T. Dierksand C. Allen, "The TLS Protocol (RFC 2246)" *http://www.ietf.org/rfc/*, IETF RFC 2246, January, 1999

[7] White Paper, "IBM 4197 Cryptographic Accelerator" *http://www.ibm.com/*, IBM Corporation, 2000

[8] T. Berners-Lee, R. Fielding, H. Frystyk, "HyperText Transfer Protocol (HTTP), Version 1.0 (RFC 1945)" *http://www.ietf.org/rfc/*, IETF RFC 1945, May, 1996

[9] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Kasinter, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol (HTTP), Version 1.1 (RFC 2616)" *http://www.ietf.org/rfc/*, IETF RFC 2616, June, 1999

[10] Paula Musich, "Mega-proxy servers: A load of trouble?" *http://www.zdnet.com/eweek/stories/-general/0,11011,2504237,00.html*, Printed from PC Week, March 31, 2000

[11] Technical Marketing Reference, "Persistence Mechanisms: enabling today's dynamic e-Business transactions" *http://www.intel.com/network/documents/*, Intel Corporation, 2000

[12] Technical Marketing Reference, "Integrated Service Director, iSD - SSL Accelerator" *http://www.alteonwebsystems.com/collateral/isd-ssl.pdf/*, Alteon Corporation, 2000

[13] White Paper, "Designing a Secured Website: What you need to know about SSL benchmarking" *http://www.intel.com/network/white_papers/*, Intel Corporation, 2000

[14] White Paper, "The Mega-proxy Problem for e-Commerce Providers: Persistence during secure sessions" *http://www.intel.com/network/white_papers/*, Intel Corporation, 2000

[15] Tarman, Hutchinson, Pierson, Sholander and Witzke, "Algorithm-Agile Encryption in ATM Networks", IEEE Computer, September 1998.

[16] Chris Burroughs, "Sandia Researchers Develop World's Fastest Encryptor - Device encrypts data at more then 6.7 billion bits per second" *Sandia National Labs Press Release*, June, 1999

[17] W. Richard Stevens. "Unix Network Programming: Volume 1" Prentice Hall, Upper Saddle River, NJ 1998

[18] Ronald Mraz "Secure Blue: An Architecture for a High Volume SSL Internet Server", 17th Annual Computer Security Applications Conference, December 2001, New ORleans, Louisiana

[19] Internet URL. "Webstone Benchmarks - Source Code" *http://www.mindcraft.com/webstone/*, 2001

[20] Internet URL. "SpecWeb 99" *http://www.spec.org/osg/web99/*, 2001

# 7 Appendix - Flowcharts

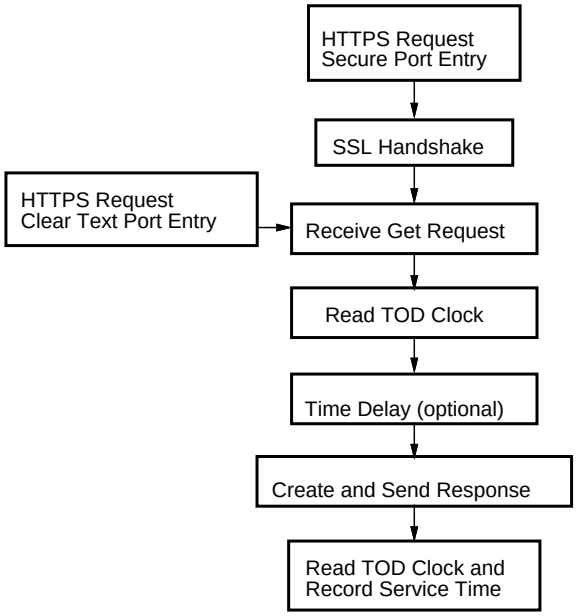


Figure 9: A flowchart of the content server.

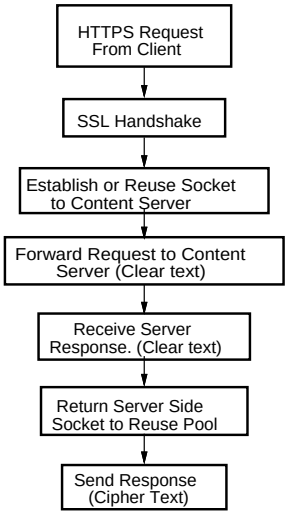


Figure 10: A flowchart of the SSL Proxy Server.

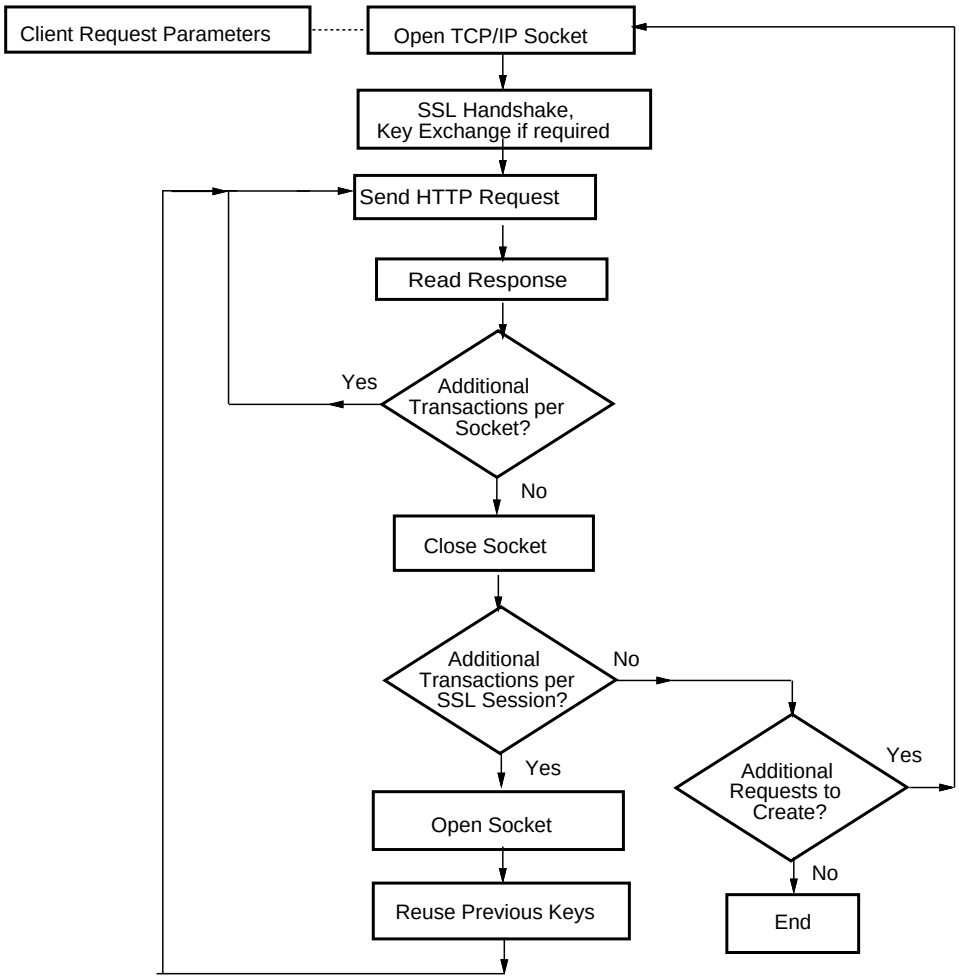


Figure 11: A flowchart of the SSL Client Test Program.