Secure Blue: An Architecture for a Scalable, Reliable High Volume SSL Internet Server

Ron Mraz
IBM T.J. Watson Research Center
30 Saw Mill River Road, Hawthorne, NY 10532
mraz@us.ibm.com

Abstract

Although there exist accelerator products to increase throughput of encrypted transactions produced by an Internet HTTP server, there are no current architectures that provide a truly coordinated and scalable solution for Secure Socket Layer (SSL) encrypted communications. This paper presents an architecture that facilitates high volume SSL Internet serving, scaling from thousands to millions of independently active SSL sessions. Reliability, availability, serviceability, and on-line error recovery requirements for such an application are also addressed.

Our approach is to offload SSL set-up protocol activity that was traditionally executed by Transaction Engines (and dedicated co-processors), to a scalable array of SSL Handshake Protocol specific servers. This significantly reduces utilization on the Transaction Engines since SSL session set-up is a CPU intensive operation. Additionally, the actual encryption/decryption processing is offloaded as well, to a dedicated and scalable array of In-Line Encryption Engine(s). The In-Line Encryption Engine is architected such that requests and responses flowing to and from the Transaction Servers are in clear text. A benefit of this arrangement is that Transaction Engines (as well as Web Accelerator Proxies) will retain the ability to cache web objects, while firewalls will retain the ability to perform packet level inspection of all traffic directed to the transaction engines. Such features have been sacrificed in prior SSL implementations.

1 Introduction

Existing Secure Socket Layer (SSL) communication protocols [6] [7] 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 allows 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 exchanged. 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" [7] 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 shared secret encryption keys of SSL makes this protocol effective for short duration shopping cart evaluations and purchase execution. Conversely, this periodic renegotiation makes SSL cumbersome for long term operations, such as 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 1 provides definitions for different classes of SSL connectivity. The Gaant Chart shows there are three types of connections supported by SSL today. They are the One-Time Connection, the Intermittent Connection and the Continuous Connection. The One-Time connection is the 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.

The timeout parameter shown in Figure 1 can be set from 5 seconds to 24 hours in SSL products as referenced in the Mozilla on-line documentation, see our reference [5]. 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 (longer) 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 [10] 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 1 for even a single page transfer. HTTP Version 1.0 would not map to the One-Time Connection (unless only a single object is requested) and Continuous Connect Traffic Pattern of Figure 1 since a separate connection is required for each object. HTTP Version 1.1 [11] 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 1.

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 operations that will determine 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. The first is the addition of hardware accelerator cards, such as the IBM 4197 [8], 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.

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 of transaction servers to staggering levels to handle peak loading periods. Although, this practice is desirable from a server vendors 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.

Load balancing affinity is an issue for traditional web site systems that rely on sniffing the packet and viewing the cookie for user identification. When SSL is invoked, cookies are encrypted with the rest of the data. In our Secure

Blue architecture all data appears as plain text to the load balancers and servers. This allows for any number of methods to be used for server affinity mapping.

Additionally, the reason and justification for employing an SSL implementation is the protection of data while in-transit across the open and public Internet. It is ironic that the use of SSL eliminates ones' ability to provide content examination with a packet sniffing firewall. Encrypting the data packets with advanced encryption and information hiding techniques (such as default packet sizes) drastically reduces the effectiveness of current Intrusion Detection Systems and packet filtering firewalls. In fact, hacker resources on the Internet boast that given the choice between Port 80 and Port 443, the hacker will use the SSL Port 443 connection every time to gain access to the site.

This paper analyzes the requirements for, and presents the architecture and design of a high volume SSL server that achieves scalability by partitioning and optimizing functionality involved in serving SSL content. The major functions we designate for high volume serving are: in-line encryption and decryption, SSL handshake processing, and TCP/IP socket redirect or hand-off. It is our position that SSL processing is problematic when competing for CPU resources in the transaction server of the site. Redirecting or handing off SSL operations, at appropriate times, to specialized parallel resources not only scale the operations, but, provides wall clock speed-up for the operations. Since the resources for each function are parallel and redundant in nature, effective system reliability and servicing of error conditions without significant loss of performance is provided as an artifact of the system architecture. In addition, there are natural, plain text interfaces, that allow the architecture to incorporate web caching techniques to improve object access time and packet inspection firewalls for hardening of Internet Security.

This paper provides a system overview of a High Volume SSL Server Architecture and the remainder is organized as follows. Section 2 provides a high level description of the High Volume SSL Server Architecture and defines requirements for each of the components functionality based on the SSL protocol. Namely, in-line encryption and decryption, SSL handshake processing, and the TCP/IP State Migration and Management Mechanism. Section 3 describes the TCP/IP socket redirection and hand-off function. Section 4 describes the role of the Network Dispatcher in the architecture. Section 5 describes the scalable SSL handshake processing function. Section 6 describes the design and requirements for our in-line encryption/decryption engine. Section 7 provides performance estimates and Section 8 provides a discussion of Reliability and Availability. Section 9 provides a review of Related Work and Section 10 includes a brief Summary and Conclusions as well as recommendations for future work.

2 High Volume SSL Server Architecture

The architecture we are proposing scales by function rather than arrays of system components. In other words, rather than scaling the server components, we split apart and scale the major operations that the servers perform. The major components of our Secure Blue System are: Transaction Servers, the SSL Handshake Engines, the In-Line Crypto-Engines and the site load balance mechanism termed the Network Dispatcher. These components and their relationships are shown in Figure 2. These components support scalable operation of our 3 functions, namely, in-line encryption and decryption, SSL handshake processing, and the TCP/IP State Migration and Management Mechanism. It is possible that further partitioning of these major components can be achieved to allow more efficient system scaling.

The Transaction Servers are the customer's e-commerce HTML site servers. These provide the page or fragment serving of content and queries to the back-office database system for transaction execution. Whenever a secure socket connection is referenced, this TCP/IP connection is then handed off to a scalable set of SSL Handshake engines. The use of socket hand-off for Internet related operations was introduced by Song in [2] and independently by Tracey in [3] to support scalable load balancing. This is an implementation of a TCP/IP State Migration and Management Mechanism. These engines provide the initial handshake required for processing and storing of the encryption state for secure operation. The engines perform this operation masquerading as the original TCP/IP address. The state required for supporting TCP/IP State Migration also affords them historical references of socket origination. This is useful to transfer the socket back to the original Transaction Server when Transaction Machine affinity is required for content serving.

Once a session has been negotiated, 2 operations are then initiated to begin the SSL session. They are 1) the Key/Management Control state set up and 2) TCP/IP socket transference back to the original Transaction Server. The Key/Management Control state is migrated to our in-line Encryption/Decryption Module for action on that specific TCP/IP socket connection. From this point forward, all incoming packets for this TCP/IP port will be decrypted and forwarded by plain text to the Network Dispatcher. The TCP/IP Socket is then transferred back to the original Transaction Server for plain text operation.

The Transaction Server continues operation in plain text which is routed through the Network Dispatcher and encrypted through the In-line Crypto- Engine. These operations are transparent to the End User Clients who are seeing a single system image for SSL web page serving. Content

serving continues until the SSL session requires an update to the session keys. At this time we require another set of key exchanges. At this time, the TCP/IP socket is again handed off to a SSL Handshake engine for update and creation of a new set of encryption keys. Once a new encryption state is determined, the Key/Management State is updated and the TCP/IP socket is transferred back to the original Transaction Server.

The Network Dispatcher provides the initial load balancing routing for the initial socket connection. Since the handshake operation is offloaded to the SSL Handshake Engines, affinity to any existing SSL state can be done when the socket is redirected to the appropriate handshake engine.

This cycle is repeated until the Client or Transaction Server terminates the TCP/IP socket. If the SSL Handshake Engine determines that the socket should be terminated it terminates the socket and provides a termination notice to the originating Transaction Server for clean-up of reserved resources within the TCP/IP stack.

The fundamental advantage of the Secure Blue architecture is it's scalability using traditional servers running specialized software. There is no need to use specialized, single function appliances that require specialized hardware design for performance improvements. Secure Blue scales with parallelism, network bandwidth, and CPU speed increases predicted by Moore's Law.

3 Transaction Servers

These are standard web servers that have their TCP/IP stacks augmented with the ability to migrate or "hand-off" a TCP/IP socket to a known server for handshake. This is done through a user level command to migrate the socket. This feature can be integrated as part of the middle-ware services of the web server.

Since the TCP/IP socket will be returned for service to this Transaction Server, a holding area should be maintained for a deterministic period of time while the secure session state is being constructed. This is because there may be a user affinity to that particular server in prior plain text transaction state. If the socket port assignments are not reserved for the return, they would most likely be reassigned for use to another connection. This feature is independent of the use of a proxy server that renames the global IP address for a single system image address.

All packets for TCP/IP Port 443 would be received by an application proxy on each web server. The proxy would examine the headers and content of each packet for identification of a required SSL handshake. At this time the proxy would migrate the socket to a dedicated handshake engine for key negotiation. Once the negotiation is finished and the bulk in-line encryption engine is set up for this processing, the socket returns to this server for content serving. The protocol and semantics of this operation will be defined as a formal TCP/IP State Migration and Management Mechanism.

The application proxy examines the data (as it is in clear-text) and, if it is not an SSL protocol request, proxies it to the HTTP (port 80) of the local host machine. Note that the web server application does not have any knowledge or requirement to support this feature. All interactions will be in clear text.

4 Network Dispatcher

The Network Dispatcher provides the initial load balancing routing for the initial socket connection. Since the handshake operation is offloaded to the SSL Handshake Engines, affinity to any existing SSL state can be done when the socket is redirected to the appropriate handshake engine.

5 Scalable SSL Handshake Engines

These servers are an array of CPUs optimized (in memory, speed and function) to efficiently perform SSL handshake operations. These are application proxies that use the call functions of SSL to coordinate and negotiate the SSL key exchange. The baseline functions can be performed by an array of CPUs based on PowerPC [9] Embedded Processors. These processors support open OSs such as Linux and have on-chip hardware for Ethernet and a PCI bus interface. Commodity PCI chip sets can be used to support encryption functions or custom CPUs can be provided with on-chip encryption core support.

These machines have TCP/IP stacks that allow socket migration from a hand-off server. This requires the spoofing of the source address in the TCP/IP header to reflect that of the original server machines IP address and TCP port number.

The application proxy will set up the keys for the in-line encryption engine. The TCP/IP socket is then transferred back to the original server for information exchange.

6 High Volume In-line Crypto-Engine

This component provides high speed, in-line encryption and decryption of TCP/IP packet payloads to specifications of a state configuration specified by the SSL Handshake Engines. A key component of this architecture is an associative look-up based on TCP/IP port and address assignments. The lookup provides the state information for cypher state encryption and/or decryption. An example would be a set of DES3 keys for decryption and encryption of packet payloads. This component would have the ability to modify the

TCP/IP packet to conform to encryption rules of the cypher paradigm.

In addition, since incoming information is decrypted, a high performance packet sniffing firewall can be appended to this function to search for specific binary patterns in content and intrusion alerts for hacker penetration attempts.

The engines are based on a multi-threaded encryption engine as described by Pierson in [17] and [18] for high performance and reduced end to end object latency.

7 Performance Estimates and Scalability Requirements

Reference [15] provides a example sizing of an e-commerce site (CDnow) that contains two pages of approximately 11.1 K bytes of content. 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. For the purposes of this discussion we require site performance to allow both pages to be served to users within a typical 100 second handshake SSL timeout period.

Using our performance metric ratios referenced in the Introduction, each Transaction Server, at full capacity, is now capable of serving 2000 static pages per second. At 11K bytes per page, this is 22,000,000 bytes per second or 176 mega bits per second. Over the entire timeout period of 100 seconds, this is a total of 200,000 pages at 11K bytes of information.

The maximum number of static content sessions to be supported over the handshake period is $200,000/2$ or 100,000. By definition, all session encryption keys must be reset during the handshake period. Handshakes can be done at a rate of 20 Handshakes per CPU-second. The number of handshake engines required to support full capacity of a single transaction server is $100,000/20 * 100$ or 50 handshake servers.

The In-Line Crypto-Engine would require in-line processing capability of 176 Mbits/sec per transaction engine. Additionally, state information for 100,000 sessions is required for full scale operation of the SSL Transaction Server.

Since typical transaction oriented databases can sustain transactions at the rate of 1000 per second, this scenario can be envisioned in other industries where erratic swings in load are common place. These would include bid and auction services, stock market trading, and subscribed real-time monitoring of global events such as the Olympics.

These scaling effects have not been seen by typical SSL site administrators because when a transaction server is performing the SSL encryption and decryption its full capacity loading is at 200-400 pages per second. In addition, SSL serving is typically personalized dynamic content which requires higher CPU loading than traditional static content. Each functional element can scale for performance and/or reliability.

8 System Reliability and Availability

The ability to partition the system by function rather than components provided us with scalability. This ability also facilitates redundancy which enhances reliability and availability. We will address issues of reliability for each of our specialized functions independently as they each have unique configurations with respect to functional state.

Transaction server reliability can be treated in the context of an array of processing engines. An interruption in service of one server would require its processing load to be shared among the remaining Transaction Servers. If the interruption occurs during connections, it is possible that some state of a client, not transferred to persistent store, may be lost. Socket communications may be terminated through timeouts forcing clients to re-establish communications with the site.

The SSL Handshake Engines are inherently stateless since the handshake operation relies on the generation of the next set of encryption values. There is no requirement for client or server affinity to be designed into these systems at the architectural level. Therefore, an interruption in service of any one handshake server would require its share of the processing load to be divided among the remaining Transaction Servers.

In a production environment, the above sizing suggests that a dedicated set of 100 Handshake Engines would be assigned to a specific Transaction Server. In addition, a shared pool of Handshake Engines would be available to smooth over any peak loading conditions or service interruptions of the Handshake Engines. Scheduling of the Handshake Engines for operation would be entirely up to the Transaction Servers Handoff Algorithm. Future work will show loading conditions for a variety of scheduling algorithms including round robin, random and load balanced operations.

The keys, encryption state, and the SSL Session ID used to uniquely identify SSL sessions can be in one of 4 states. They are 1) valid and in-use, 2) invalid and in-use, 3) valid and not-in-use, and 4) invalid and not-in-use. The “valid” state represents when a Session ID has not yet expired. An “invalid” state represents when a Session ID has expired. This can occur independently of the session being in or not-in-use. Finally, an in-valid and not-in-use state has only archival value for forensic analysis.

The in-use states would be active within the Handshake Engines and the In-Line Crypto Engines. The valid and not-in-use Session IDs and corresponding state may be kept in a distributed fashion across the handshake engines or in a

centralized dedicated server for look-up. Since all packets in the site are in plain text, the Network Dispatcher can distribute traffic (based on cookies, or user IDs in the packet headers) with affinity. By this affinity association, individual Session ID's would be local to a subset of Handshake Engines reserved for a specific (set) of Transaction Engines.

The In-Line Crypto-Engine represents a potential single point of failure from an architecture perspective. This function will be provided as an array of identical redundant components. Since each TCP/IP connection requires a set of encryption/decryption keys for operation, all connection state information is available on each In-Line Crypto Engine. This is to allow seamless migration of packets from one Engine to another in the event of an interruption in service.

9 Related Work

This section includes an overview of competing products by Intel (IPivot acquition) [13] and [16], and Alteon (Currently part of Nortel Networks and 3COM). The problem of high volume SSL proxy serving is highlighted in the press at [12].

Ipivot's 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 Server connection affinity.

Alteon's solution [14] 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 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.

Additionally, in these related offerings, the HTTPS socket connects from the client to the Traffic Manager or SSL Accelerator. The HTTPS content requests are forwarded onto the Transaction Servers. In Secure Blue, the use of TCP/IP State Migration or handoff affords the opportunity to manage the socket connection from client to Transaction Server with only the In-Line Encryption Engine and Network Dispatcher adding delays to the request routing. This offers reduced request response latency. In addition, Transaction Servers in Secure Blue are no longer restricted to the amount of SSL processing provided by one or two inline SSL processing engines (that do both Handshake and Encryption). The Secure Blue architecture supports any number of engines within a functional cluster to insure that the Transaction Server can now serve at a pages/second rate equal to that of traditional plain text serving performance.

10 Summary and Conclusions

This paper analyzes the requirements for, and presents the architecture and design of a high volume SSL server that achieves scalability by partitioning and optimizing functionality involved in serving SSL content. The major functions we designate for high volume serving are in-line encryption and decryption, SSL handshake processing, and TCP/IP socket redirect or hand-off. It is our position that SSL processing is problematic when competing for CPU resources in the transaction server of the site. Redirecting or handing off SSL operations, at appropriate times, to specialized parallel resources not only scale the operations, but provides wall clock speed-up for the operations. Since the resources for each function is parallel and redundant in nature, effective system reliability and servicing of error conditions without significant loss of performance is provided as an artifact of the system architecture. In addition, transaction servers now see natural plain text interfaces, that allow the architecture to incorporate site wide web caching and packet inspection firewalls for hardening of Internet Security.

The fundamental advantage of the Secure Blue architecture is its scalability by function using traditional servers running specialized software. Secure Blue scales with parallelism, network bandwidth, and CPU speed increases predicted by Moore's Law. We believe that the methodology of partitioning the SSL application by function and having dedicated clusters of computers has possibilities for many other Internet applications. We call this paradigm a "Multi-Cluster Application Architecture for Internet Servers". This is based on the similarity between this application scaling methodology and a hardware architecture cited in [4].

As computers continue to improve in performance [19] and capability, which may include integrated hardware for encryption, it is feasible that general purpose processors

will have integrated support for cryptographic operations in a few short "Internet Years". Software that allows direct IP packet transmission and receiving, such as the raw socket Pcap library [20] [21], allow much of the specialized Internet router functions to be prototyped in user level software. Since the functions of Secure Blue can scale on commodity components, the implementation of this system, at minimum, is an integrated set of software components that run on scalable functional clusters of server appliances.

Continued investigations will consider the impact of dynamic content generation vs. static content serving, client certificates and the operational aspects of such an architecture for LDAP (Lightweight Directory Access Protocol) with SSL access.

11 Acknowledgments

The authors would like to thank Junehwa Song, (currently on sabbatical in Korea) and John Tracey for their insight into Socket Connection Handoff operation. Karen Witting is prototyping the first version of Secure Blue. The authors would also like to thank Paul Dantzig, Dan Dias and Nagui Halim who provided management support for this work. Robert Filepp provided many helpful comments in the editing of this document. Many thanks to Gabriel Silberman, Kattamuri Ekanadham and Kemal Ebcioglu for suggesting the term, Multi-Cluster Application Architecture, as a formal name for our scaling methodology. Finally the comments of the anonymous reviewers who have strengthened and clarified this paper.

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] Keith I. Farkas, Paul Chow, Norman P. Jouppi, and Zvonko Vranesic "The Multicluster Architecture: reducing Cycle Time Through Partitioning", Proc. of IEEE Micro-30, December 1-3, 1997 Research Triangle Park, North Carolina.

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

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

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

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

[9] White Paper, "The PowerPC 405 Core" *http://-www.-chips.-ibm.-com/products/powerpc/cores/405cr_wp.pdf*, IBM Corporation, 1998

[10] 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

[11] 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

[12] Paula Musich, "Mega-proxy servers: A load of trouble?" PC Week, March 31, 2000

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

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

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

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

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

[18] 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

[19] Ray Kurzweil, "The Age of Spiritual Machines" Penguin Books, New York, NY 1999

[20] Van Jacobson, Craig Leres, and Steve McCanne. "libpcap" *ftp://-ftp.-ee.-lbl.-gov/*, 1994

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

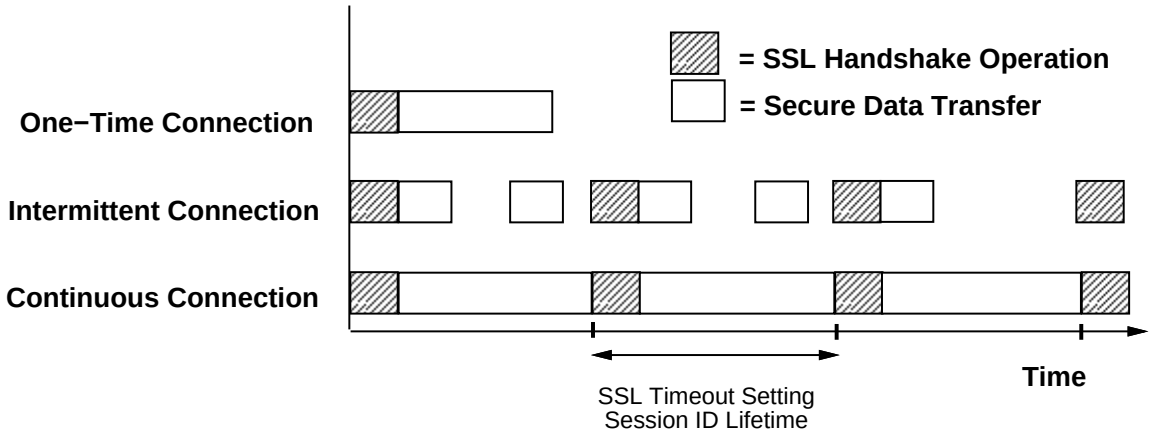

Figure 1. Definition of SSL Client-Server Traffic Pattern Types.

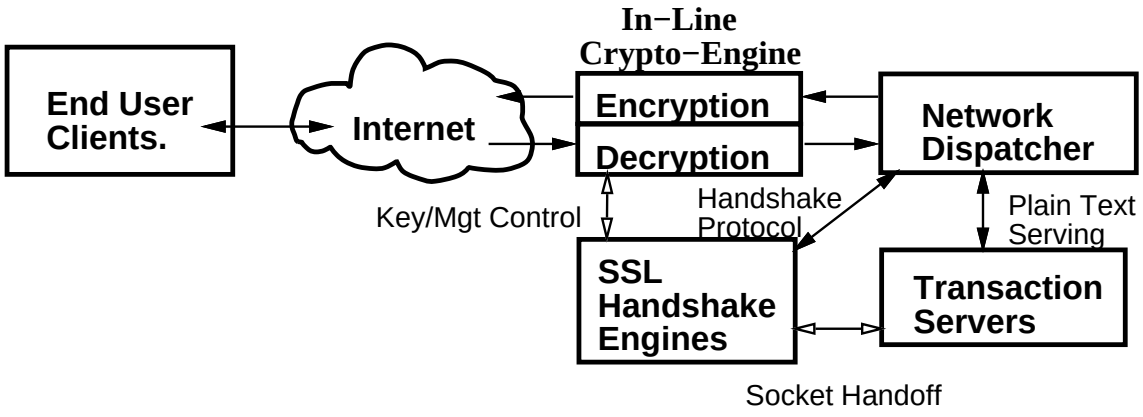

Figure 2. A high level view of the Secure Blue Architecture.