LSWS: a Linux-based Secure Web Switch

C. Edward Chow, Ganesh Godavari and Yu Cai
Department of Computer Science,
University of Colorado at Colorado Springs,
Colorado Springs, CO 80933-7150, USA
{chow, gkgodava, ycai}@cs.uccs.edu
Tel: (719)262-3110
FAX: (719)262-3369

Abstract

In this paper we present the design of a Linux application level secure web switch, called LSWS, which is based on OpenSSL package. It is intended to be used a front end switch for a secure web server cluster. It allows flexible specification and dynamic update of content switch rules. Two versions of LSWS were implemented. One based on dynamic forking of child processes and other based on prefork idea similar to Apache. The overheads of content switching and SSL processing are analyzed using WebBench. We also compare its performance with Intel 7280 XML director.

Keywords: Content Switch, Cluster, SSL, Linux, Secure web server, Secure web switch.

1. Introduction

The explosion of the Internet from a small network of known individuals to a huge, heterogeneous anonymous network has brought several troubles in its wake. Right from the case of a mail account password being sniffed and acquired to credit card numbers and other confidential business data being observed, attacks can occur on transmitted information in many ways. The number of commercial transactions and private data transmission that occur and the ability of malicious elements to observe and manipulate data anonymously has necessitated the growth of security measures to protect Internet users. A need to have standard security protocols that are platform and network type independent and easily implementable, usable and secure was felt.

Along with security another major issue is handling of the large volumes of data present in today's networks. Many approaches have been devised in order to provide a solution to this problem. One solution to reduce the load is to have a paid subscription to one of the Content Delivery Network (CDN) providers such as Akamai [1], Speedera [2] and Digital Island [3]. Another approach is to distribute the large volume of requests among a group of servers where a master controller, that can be a dedicated host or a process, first receives the requests and delegates it to one of a group of servers for processing. A *content switch* (CS) [4] is such a load balancing system that distributes load based on the content of the received requests. A *Web-switch* is a content switch that distributes load based on Web requests.

Consider the case of an e-commerce site with a large amount of traffic. The users who are accessing the site may be performing various functions like browsing, signing in or doing some profitable activity like purchasing. It makes good business sense to provide better and faster access to paying customers rather than casual surfers. One way of doing this is to provide some kind of preferential treatment like routing them to faster servers. This segregation implies that requests are routed to different servers based on their content. This kind of routing based on request content cannot be achieved by traditional layer4 and below switches, which route requests based on request characteristics like port number, or IP address, but not on the content of the request. A better approach in this direction would be to develop a mechanism, which can route request based on content, in other words, a content based switch. Some of the other functions that a content switch can be used for are:

Load Balancing: As a Load Balancer, a content switch can segregate incoming requests based on the HTTP meta header, URL or even the application layer payload and route them to the back end servers in the server cluster.

Firewall: As a firewall, a content switch can either allow or reject requests based on their content or their IP address.

Email Filtering: a content switch can function as an efficient spam guard or work as an anti virus device by verifying the sender and content of emails.

E-commerce transactions normally involve the transfer of sensitive or private data like personal information or credit card numbers, which are liable to active or passive attacks during transmission.

There are several approaches to provide Web security. The approaches are similar in the services they provide and, to some extent, in the mechanisms that they use, but they differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack.

One way to provide Web security is to use IP Security or IPSEC [5] as it is generally called. The advantage of using IPSec is that it is transparent to end users and applications and provides a general-purpose solution. Further, IPSec includes a filtering capability so that only selected traffic need incur the overhead of IPSec processing.

Another relatively general-purpose solution is to implement security just above TCP. The foremost example of this approach is the Secure Sockets Layer (SSL) [6] and the follow-on Internet standard of SSL known as Transport Layer Security (TLS) [7]. At this level, there are two implementation choices. For full generality, SSL (or TLS) could be provided as part of the underlying protocol suite and therefore be transparent to applications. Alternatively, SSL can be embedded in specific packages. For example, Netscape [8] and Microsoft Explorer [9] browsers come equipped with SSL, and most Web servers have implemented the protocol.

There are several commercial content switches. The Alteon iSD-SSL accelerator[10] works with an Alteon Web switch to intelligently speed secure e-Commerce transactions by offloading Secure Sockets Layer (SSL) processing from local servers without imposing delays on other traffic in the same data path. **F5** has long been a leader in the Internet Traffic Management space. It has an advanced web switching logic (F5 Networks Big-IP Controller 3.1[11]). Big-IP combines a strong, traditional load-balancing feature set with SSL acceleration ability. It extracts HTTP header information from the request to make traffic management decisions. Foundry (**Foundry Networks ServerIronXL with Internet IronWare 7.0** [12]) equips its ServerIronXL with a strong Layer 2 and Layer 3 switching and Layer 5/7 load-balancing based on cookies and SSL session IDs. Intel provides CEA7110 [13] SSL Accelerator and CEA7280 [14] XML Director which is very close to features that our LSWS provide. It is capable to perform routing based on url and xml tag values.

2. Secure Web Switch

The SSL protocol is a security protocol that sits on top of TCP at the transport layer. In the OSI model, application layer protocols such as HTTP or IMAP handle user application tasks such as displaying Web pages or running email servers. Session layer protocols establish and maintain communications channels. Transport-layer protocols such as TCP and UDP, handle the flow of data between two hosts. Network layer protocols such as IP and ICMP provide hop-by-hop handling of data packets across the network.

SSL operates independently and transparently of other protocols so it will work with any application-layer and any transport-layer protocol. This allows clients and servers to establish secure SSL connections.

An application layer protocol sends unencrypted data to the session layer. TLS/SSL encrypts the data and hands it down to the lower layers. When its peer receives the data at the other end, it passes it up through the layers to the session layer where TLS/SSL decrypts it and hands it to the application layer. Since the client and the server have gone through the key negotiation handshake, the symmetric key used by SSL is the same at both ends.

2.1. SSL Transactions

To illustrate how SSL works, assume a user wants to make a purchase over the Internet and needs to send a credit card number to a secure Web site [15].

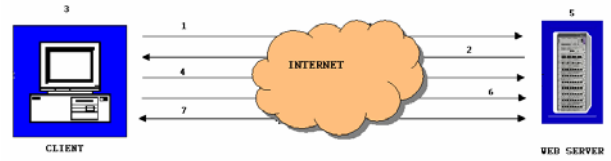

Figure 2.2 illustrates the steps taken during an SSL negotiation

Initially, the request for an SSL session comes from the browser to the Web server.

The Web server then sends the browser its digital certificate. The certificate contains information about the server, including the server's public key.

Once the browser has the server's certificate, the browser verifies that certificate is valid and that the CA is listed in the client's list of trusted CA's. The browser also checks the certificates expiration date and the Web server domain name.

Once a browser has determined that the server certificate is valid, the browser then generates a 48-byte master secret key. This master secret key is encrypted using server's public key, and is then sent to the Web server.

Upon receiving the master secret from the browser, the Web server then decrypts this master secret key using the server's private key.

As both the browser and the Web server have the same master secret key, they use this master secret to create keys for the encryption and MAC algorithms used in the bulk-data process of SSL. Since both participants use the same master key, they now have the same encryption and MAC keys.

As both the browser and Web serve have the same encryption and MAC keys, they use the SSL encryption and authentication algorithms to create an encrypted tunnel. Through this encrypted tunnel, they can now pass data securely through the Internet.

Though the authentication and encryption process may seem rather involved, it happens in less than a second. Generally, the user does not even know it is taking place. However, the user will be able to tell when the secure tunnel has been established since most SSL-enabled Web browsers will display a small closed lock at the bottom or top of their screen when the connection is secure. Users can also identify secure Web sites by looking at the Web site address; a secure Web site's address will begin with https:// rather than the usual http://.

2.2. OpenSSL: The Open Source toolkit for SSL/TLS

OpenSSL is based on the excellent SSLeay [17] library developed by Eric A. Young and Tim J. Hudson. Open Source toolkit implements the Secure Socket Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general-purpose cryptography library.

OpenSSL contains two important libraries:

OpenSSL SSL library implements the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols.

OpenSSL CRYPTO library implements a wide range of cryptographic algorithms used in various Internet standards. The services provided by this library are used by the OpenSSL implementations of SSL, TLS and they have also

been used to implement SSH [18], OpenPGP [19], and other cryptographic standards.

In addition, the OPENSSL program is a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. It can be used for

- Creation of RSA [20], DH [21] and DSA [22] key parameters.
- Creation of X.509 certificates [23] and Certificate Revocation List (CRL).
- Calculation of Message Digests
- Encryption and Decryption with Ciphers.
- SSL/TLS Client and Server Tests.
- Handling of S/MIME signed or encrypted mail.

2.3. Software architecture of secure content switch

The following figure 2.3 shows the architecture of the current design of secure content switch.

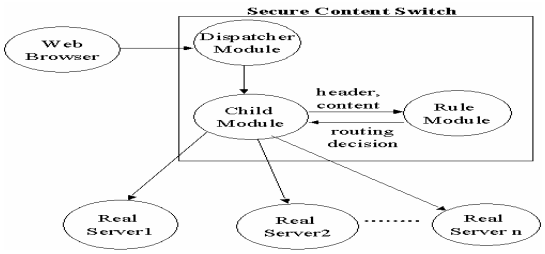

Figure 2.3 Architecture of secure content switch

The following steps describe the process of handling secure requests using the secure content switch.

The web browser makes a request to the secure content switch.

The dispatcher module in the secure content switch forwards the request to the secure content switch child module. In the dynamic forking version of secure content switch the dispatcher module forks a child process. In the preforking version of secure content switch the dispatcher module forwards the request to a free child.

The secure content switch child module performs the handshake with the client and reads in the request.

The secure content switch child module then sends the request to the Rule module, which performs rule matching and returns the name of the server by which the request can be served.

The real server connects to the web browser through the secure content switch child module (not shown in the picture).

2.3.1. Secure Content Switch Child Module.

The child module performs the SSL handshake with the client. This step involves establishing ciphers to use and providing certificate to the client for server-authentication. As part of the SSL Handshake request, the client may provide a current or previously created SessionID to reuse for the current connection. The Secure content switch manages the SessionID and this will be used as appropriate during the SSL Handshake. The child module receives the request for data and decrypts the data according to the negotiated SSL handshake. When the Secure content switch process determines it has fully received an HTTP request, it sends header information to the Rule Matching Module, which determines which Real Server can serve the request. The child module then establishes a connection with the Real Server and forwards the request in plain HTTP.

3. Performance Results

This section presents the performance results of the Linux application secure web switch. Fig shown below shows a block diagram of Secure Linux Application level content switch.

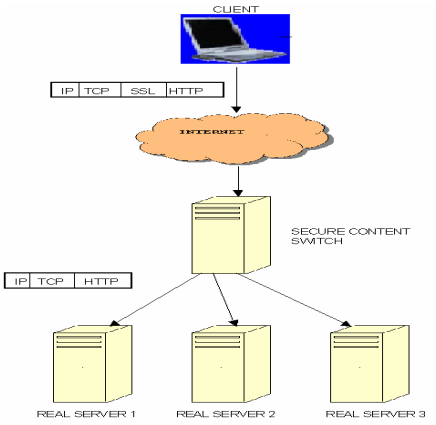

Figure 3.1 the secure web switch test bed

Table 3.1 shows the hardware and software configuration of machines used in the test-bed.

Machine Spec.	IP Address	O/S	Web Server
a)CALVIN.uccs.edu *DELL Dimension-4100, 933 MHz, 512MB*	128.198.192.184	Redhat 7.2 (2.4.9-21)	Apache 1.3.22
b) oblib.uccs.edu *HP Vectra VL 512 MHz, 512MB* (Content switch)	128.198.60.195	Redhat 7.2 (2.4.9-21)	Apache 1.3.22
a) dilbert.uccs.edu b) wait.uccs.edu c) wind.uccs.edu (Client)	128.198.60.23 128.198.60.202 128.198.60.204	a) Windows NT, 4.0 b), c) Windows-2000,Advanced Server	N/A
a) eca.uccs.edu b) frodo.uccs.edu c) bilbo.uccs.edu d) odorf.uccs.edu e) walrus.uccs.edu f) wallace.uccs.edu *HP Kayak Machines, 233 MHz, 96MB RAM* (Real Server)	128.198.60.188 128.198.60.183 128.198.60.182 128.198.60.196 128.198.60.197 128.198.60.208	Redhat 7.1 (2.4.3-12)	Apache 1.3.19

The tests were performed using Web bench [24] and the negotiated cipher suite for these tests are DES-CBC3-SHA, SSLv3, Kx=RSA, Au=RSA Enc=3DES(168), Mac=SHA1 Figure 3.2 and Table 3.2 shows that the performance of the Dynamic forking secure content switch is better than Pre-forked secure content switch. I found out that after a while the children are created and killed immediately with out serving multiple requests. The Pre-forked server is designed to see that child SSL Processes are created ahead of time and kill them if their number exceeds a certain threshold. The reason could be that more child processes are free which implies number of requests sent by the web-bench is irregular, there

by affecting the overall performance of the pre-forked secure content switch.

Figure 3.3 shows the impact of rules on the performance of the linux secure content switch. There is no major impact on the performance of our secure content switch with respect to the number of rules. This clearly shows us SSL processing is the major hurdle.

Cluster with secure content switch vs. standalone Apache web server

The performance of a dynamic forking secure content switch was pretty good when compared to a standalone apache server. Taking into fact that amount of computation overhead involved with extracting the HTTP headers and performing a rule matching on the HTTP header data, and routing the request to one of the Real Servers

4. Performance Comparison with Intel SSL accelerator and XML director.

Intel 7280 XML Director is configured as content switcher (named lizzie), and it directs XML content to eca and frodo respectively, according to the pre-defined rules. We implement a software XML content switch, installed on calvin.uccs.edu. It directs XML content to eca and frodo respectively, according to the pre-defined rules.

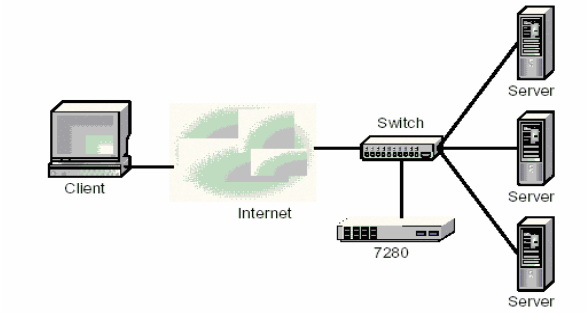

Figure 4.1 Network topology for Intel 7280 unit

We run a perl script on testing machine. It sends XML content to lizzie (Intel 7280 XML Content Switch) and calvin (Software XML Content Switch).

By repeating the same XML requests 30 times, we get the average XML file content switch processing speed. We then increase the XML length, by adding more content in the XML file, and see if the processing time increases or not. We do the test on both plain text request scenario and SSL request scenario.

It seems that Intel 7280 unit and Software Content Switch, both process XML data pretty fast. Amazingly enough, Intel 7280 is only a little bit faster than LSWS, and if the XML document size is not large enough, there are very little difference on performance.

For the XML data with SSL encryption, Intel 7280 process SSL request by itself, and then redirect plain text request to eca/frodo. On the other hand, the Software Content Switch process SSL request on Calvin by software, then redirect plain text request to eca/frodo too. We expect Intel 7280 should run a lot faster than the Software Content Switch. But we didn't observe that in our test.

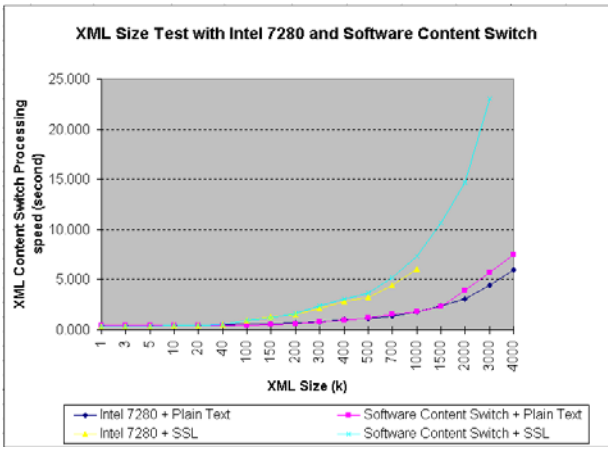

Figure 4.2 Test result on XML content switch processing time on Intel 7280 unit and Software Content Switch

Problem encountered

When the size of XML document increases (above 500k), Intel 7280 gets very picky on the format of the XML document. We always get "403 POSTed data was not welformed" error. But actually the format of the XML document are perfectly OK. Because of this, the result curve named "Intel 7280+SSL" on the testing result chart in Fig 4.2 is incomplete.

5. Conclusion and Future Work

We have developed a web switch with the support of SSL. The software package currently uses OpenSSL version 0.6. The features of our web switch include session id reuse and high encryption strength. We have tested it on a cluster test-bed using the industry standard benchmarking software WebBench and found its performance to be satisfactory.

One of the improvements that are possible is reducing the bottleneck of SSL encryption/decryption. SSL transactions are computation intensive and hence become a drain on web server resources. In cases of web servers handling a large amount of traffic, the problem can become more significant. One way to alleviate the problem is to apply parallel processing techniques. The others is to use SSL hardware accelerator such Intel 7110. But as one ponders about a solution for this problem, we need to think about how all the web servers are going to know about the TLS/SSL session information that was negotiated by another server. There is a message in the openssl group posted by Lutz Jaenicke that talks of a future version OpenSSL 0.9.7 (not yet released) that has a new function (server side) to explicitly choose the session IDs generated (rather than random values as of now). Therefore it will be possible to include a "server ID" into the session ID so that load balancing will become easier. Once the above-mentioned version is released, a solution might be in sight.

Another performance improvement can be the caching of web pages. This aspect must be handled sensibly as most of the web documents that use SSL are dynamic in nature.

One of the ways to overcome the problem of maintaining state across multiple web processes in dealing with TLS/SSL transactions is to make the connections "sticky". The problem with this design is that the IP addresses are constantly shifting for users coming into a site from Internet service providers that use proxy servers. So users must have a "sticky" connection to the SSL server that is independent of their IP address. This can be achieved by having a cookie which identifies which Real Server has serviced the user previously.

Another suggestion for improving the current web switch performance is to have a persistent HTTP connection between the web switch and the real servers, which can be used by all the processes, for all the requests.

Each of these solutions comes with its own combination of cost, performance, and flexibility of scale—proving once again that, when it comes to web site architecture, one size never fits all.

6. References

[1]Akamai technologies http://www.akamai.com/

[2]Speedera http://www.speedera.com/flash_index.html

[3] Digitalisland http://www.digitalisland.com/

[4] George Apostolopoulos, David Aubespin, Vinod Peris, Prashant Pradhan, Debanjan Saha, " Design, Implementation and Performance of a Content-Based Switch**",** Proc. Infocom2000, Tel Aviv, March 26 - 30, 2000, http://www.ieee-infocom.org/2000/papers/440.ps

[5] IP Security Protocol, http://www.ietf.org/html.charters/ipsec-charter.html

[6] SSLv3 Internet Draft (obsolete). ftp://ftp.ietf.org/internet-drafts/draft-ietf-tls-ssl-version3-00.txt.

[7] RFC2246 - "The TLS Protocol Version 1.0" ftp://ftp.isi.edu/in-notes/rfc2246.txt

[8] Netscape web browser http://www.netscape.com/

[9] Microsoft Internet Explorer web browser of Microsoft Technologies http://www.microsoft.com

[10] Web switch product of nortel networks http://www.nortelnetworks.com/products/01/alteon/isdssl/index.html

[11] web switch product of F5networks http://www.f5networks.com/

[12] Foundry ServIron Installation and Configuration Guide, May 2000. http://www.foundrynetworks.com/techdocs/SI/index.html

[13] Intel NetStructure 7180 e-commerce Director http://www.intel.com/support/netstructure/commerce/index.htm

[14] Intel NetStructure 7280 XML Director http://www.intel.com/support/netstructure/director/index.htm

[15]white paper by cacheflow http://www.cacheflow.com/files/whitepapers/wp_ssl_primer.pdf

[16] OpenSSL: The Open Source toolkit for SSL/TLS (http://www.openssl.org)

[17] The SSLeay package is copyright Eric Young and is available free for commercial and non-commercial use.

[18] ssh prtocol http://www.ietf.org/html.charters/secsh-charter.html

[19] OpenPGP the most widely used email encryption standard in the world. http://www.openpgp.org/

[20] RSA algorithm invented in 1978 by Ron Rivest, Adi Shamir, and Leonard Adleman. http://www.rsasecurity.com/

[21] The Diffie-Hellman Key Agreement, http://www.hamiltonlabs.com/links.htm

[22] NIST, FIPS PUB 186, "Digital Signature Standard", May 1994.

[23] public-key infrastructure (X.509) http://www.ietf.org/html.charters/pkix-charter.html

[24] web bench is the testing tool provided by eTesting Labs Inc. http://webbench.com/

Client	*Request Per Second Prefork NonSecure content switch*	*Request Per Second Dynamic NonSecure content switch*	*Request Per Second Apache NonSSL*	*Request Per Second Secure switch*	*Request Per Second Dynamic content switch*	*Request Per Second Prefork Secure content switch*	*Request Per Second Apache SSL*
1_client	***148.046***	***82.588***	***244.404***	***26.992***	***23.042***	***37.450***
4_client	***146.542***	***84.283***	***241.296***	***26.100***	***20.858***	***36.958***
8_client	***128.688***	***82.642***	***234.867***	***26.113***	***21.704***	***37.479***
12_client	***145.521***	***83.567***	***230.183***	***26.279***	***20.246***	***37.279***
16_client	***148.100***	***82.017***	***236.350***	***26.425***	***21.604***	***37.396***
20_client	***147.946***	***83.433***	***241.475***	***26.333***	***19.462***	***36.962***
24_client	***135.046***	***82.642***	***237.050***	***26.358***	***21.004***	***37.833***
28_client	***148.058***	***83.158***	***234.037***	***26.421***	***20.279***	***38.150***
32_client	***126.621***	***82.767***	***241.037***	***26.275***	***20.358***	***38.346***
36_client	***123.542***	***81.933***	***242.046***	***25.783***	***20.275***	***38.375***
40_client	***148.121***	***81.575***	***239.567***	***25.625***	***21.188***	***37.892***

44_client	***129.762***	***83.112***	***232.988***	***26.033***	***20.163***	***37.804***
48_client	***148.113***	***83.421***	***243.688***	***26.304***	***20.404***	***37.571***
52_client	***147.850***	***81.975***	***244.037***	***26.063***	***21.446***	***37.400***
56_client	***106.900***	***82.254***	***243.258***	***26.350***	***17.363***	***37.063***
60_client	***128.879***	***83.254***	***243.554***	***26.212***	***15.800***	***36.188***

Table 3.2 showing the request/sec of different types of secure web switch

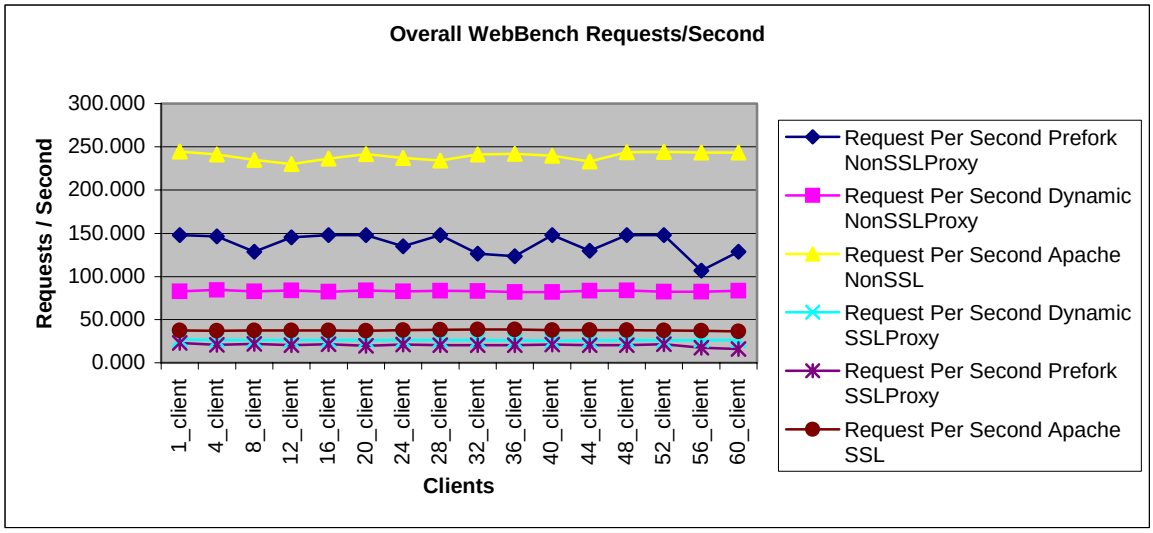

Figure 3.2. Content switching/SSL processing overhead and performance differences between Prefork and Dynamic fork versions.

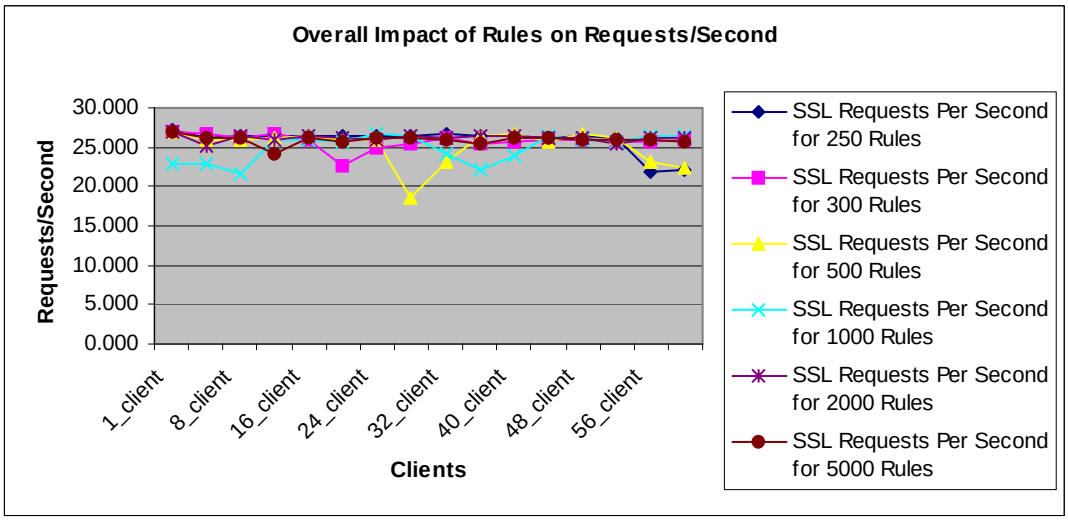

Figure 3.3. Impact of # of content switching rules over performance.