SSLSARD: A Request Distribution Technique for Distributed SSL Reverse Proxies

Hai-Tao Dong[1,2,3], Lei Song[2], Jin-Lin Wang[2], and Jun Yang[1]
[1] Key Laboratory of Noise and Vibration Research, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China
[2] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China
[3] University of Chinese Academy of Sciences, Beijing 100190, China
Email: {donght, songl, wangjl}@dsp.ac.cn; jyang@mail.ioa.ac.cn

Abstract —Although Secure Sockets Layer (SSL) and its successor Transport Layer Security (TLS) are the for transport layer security, their cryptographic operations tend to be highly CPU intensive. Web systems that support SSL/TLS often deploy several locally or globally distributed SSL reverse proxies in front of Web servers to offload SSL/TLS operations from Web servers and improve the execution performance of the SSL/TLS protocol. A particularly obvious problem is the distribution strategy of incoming requests to the SSL reverse proxies. In this paper, we propose a request distribution technique to improve the overall performance of SSL reverse proxy system. This technique is called SSL-Session-Aware Request Distribution (SSLSARD), consisting of a real-time load estimation algorithm and an SSL-session-aware request distribution algorithm. Our experimental results show that SSL session resumption is critical in improving the performance of a SSL reverse proxy system. And comparing with the client-granularity distribution strategy of SSL_session_only, SSLSARD can deal with more concurrent requests and further increase system throughput.

Index Terms—Secure Sockets Layer (SSL), Web system, SSL reverse proxy, distributed system, request distribution

I. Introduction

At present, SSL protocol [1] and its successor TLS protocol [2]-[4] are widely used to provide a secure transmission channel between a client and a server on the Internet, and have become the de facto standards for transport layer security [5]-[7]. In the rest of this paper, the term “SSL” is used to refer to both SSL and TLS. SSL runs on top of some reliable transport protocol (e.g. TCP), while under various kinds of application layer protocols. Many application layer protocols, such as HTTP, TELNET, FTP, etc. can run transparently over SSL. However, SSL is most widely used to ensure the security of HTTP traffic [7], namely HTTP Secure (HTTPS). The execution process of SSL is composed of two stages: the establishment of an SSL connection and the secure transmission of application data.

Although SSL is the most popular protocol to provide a secure transmission channel between a client and a server on the Internet, the cryptographic operations, particularly public key operations, tend to be highly CPU intensive [1]-[5], [8]-[10]. Due to this reason, SSL’s popularity rate over the Internet is rather low. In fact, among the network traffic in people’s everyday life, only a rather small part is transmitted in cipher text, the most of which is traffic carrying sensitive information (such as e-mail and online shopping) [11]. So far, the number of Web sites on the Internet has exceeded 850 million [12]. However, only less than 3 million of those offer SSL security [13]. Results of another survey [14] show that among Alexa’s list of top 1,000,000 Web sites, only about 450,000 offer SSL security. Despite all this, nowadays SSL tend to achieve a universalization within the range of the whole Internet. More and more Internet traffic is carried through as cipher text. Many research works devote to improve the execution performance of SSL, starting from the directions of speeding up cryptographic algorithms [11], [15], [16], optimizing the SSL protocol [7], [17], [18], SSL offloading [8]-[10], utilizing SSL session resumption [5], [8]-[10], etc. SSL session resumption is a mechanism provided by the SSL protocol that allows the two parties to establish an SSL connection using a previous session within its lifetime instead of negotiating a new one, so as to immensely reduce the overhead in establishing SSL connections. In fact, SSL session resumption ratio has already become a new metric to measure SSL performance [9].

Web systems that support SSL often use SSL reverse proxies [8]-[10], [19]-[21] to offload CPU exhausting SSL operations from Web servers and improve the execution performance of the SSL protocol. As is shown in Fig. 1, one or several SSL reverse proxies are deployed between clients and Web servers. The SSL reverse proxies get content in plain text from the Web servers through HTTP and provide HTTPS services to the clients. Such SSL reverse proxies usually possess hardware devices dedicated to process cryptographic operations at great speed.

Manuscript received November 18, 2015; revised April 14, 2016.
This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06010302 and the Knowledge Innovation Project of the Institute of Acoustics, Chinese Academy of Sciences under Grant No. Y154191601.
Corresponding author email: donght@dsp.ac.cn.
doi:10.12720/jcm.11.4.374-382

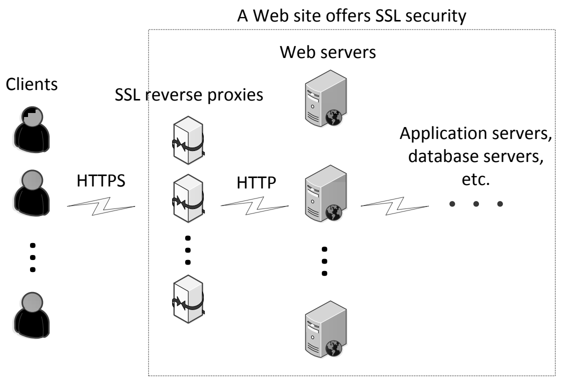

A Web site offers SSL security
Clients
SSL reverse proxies
Web servers
Application servers, database servers, etc.
HTTPS
HTTP

Fig. 1. Architecture of a Web site offers SSL security using distributed SSL reverse proxies

In order to accept more clients that request Web content protected by SSL, Web sites usually deploy several locally or globally distributed SSL reverse proxies as a collaboratively working system, as is shown in Fig. 1. A particularly obvious problem is the distribution strategy of incoming requests to the SSL reverse proxies. In order to improve the overall performance of SSL reverse proxy system, we propose a request distribution technique for SSL reverse proxies, called SSL-session-aware request distribution (SSLSARD), with which the request distributor distributes the incoming requests at a fine granularity of SSL session, taking both load balancing and high SSL session resumption ratio into consideration. This paper presents the following contributions:

- An efficient, highly real-time load estimation algorithm for SSL reverse proxies;
- An SSL-session-aware request distribution algorithm for SSL reverse proxies that makes compromise between load balancing and high SSL session resumption ratio by adjusting several parameters;
- A set of simulation that evaluates the performance of SSLSARD.

The rest of this paper is organized as follows: In Section II we describe SSLSARD in detail. In Section III we describe the design of simulation used to evaluate the performance of SSLSARD. In Section IV we present the results of our simulation. We describe the related work in Section V and then we conclude this paper in Section VI.

II. The SSLSARD Strategy

In this section, we describe the SSLSARD request distribution strategy in detail. SSLSARD consists of a real-time load estimation algorithm and an SSL-session-aware request distribution algorithm. To describe intuitively, first we provide an example of system architecture utilizing SSLSARD. Next, we describe the SSLSARD load estimation algorithm and request distribution algorithm respectively.

A. An Example of System Architecture Utilizing SSLSARD

Fig. 2 shows an example of system architecture utilizing SSLSARD. A request distributor is connected between clients and several locally or globally distributed SSL reverse proxies in series. First, the request distributor establishes a TCP connection with a client. Second, the request distributor receives a ClientHello and checks it for information on session resumption, then chooses a target SSL reverse proxy node according to the SSLSARD strategy. Third, the request distributor transfers its end point of the TCP connection to the target node using the technique of TCP hand-off [22]. Then the client can establish SSL connection with the target node and send HTTPS request to it. This system architecture enables the request distributor to distribute the incoming requests at the granularity of SSL session, which is essential to SSLSARD.

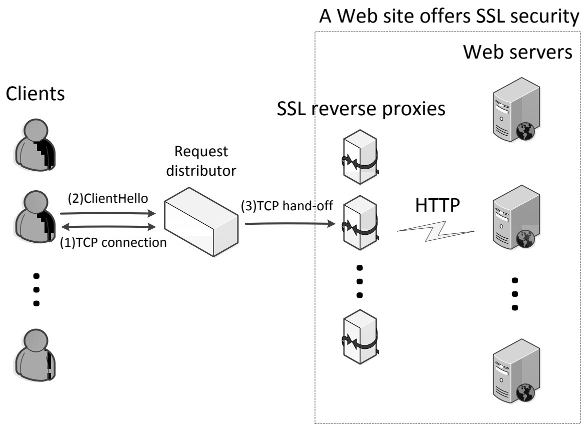


Fig. 2. An Example of System Architecture Utilizing SSLSARD

B. SSLSARD Load Estimation Algorithm

The load of SSL reverse proxies is an important input parameter of the request distribution algorithm. The SSL workload has high computational complexity. Moreover, its amount of computation differs greatly depend on whether session resumption takes place. In order to utilize the SSL reverse proxies' computational resource efficiently, the calculation of load should be fast and highly real-time. In this subsection we firstly present our method of measuring SSL reverse proxy's load. Then we describe the proposed real-time load estimation algorithm. Some symbols used in this paper are defined as follows:

- N is the total number of the SSL reverse proxies;
- G_i are the SSL reverse proxies, $i = 1, 2, 3, \dots, N$;
- $l_i(t)$ is the load of G_i at time t.

SSL reverse proxy's load is caused mainly by SSL, consisting of the overhead of SSL handshake and that of bulk encryption. The overhead of SSL handshake includes public key operations. Some related work indicates that when providing a service of a small Web page (1KB-32KB) using HTTPS, an SSL server spends about 70%-90% of its processing time on public key operations [7], [23]. In this paper, we assume that the HTTPS request content size is small enough that the difference in bulk encryption overhead among requests can be neglected. Only considering the difference in SSL handshake overhead, we divide the requests into two categories: the requests that willing to negotiate a new SSL session and the requests willing to resume an existed session s. Each request of the first category is expected to cause a load of $p_{i,new}$ to G_i, while each request of the second category is expected to cause a load of $p_{i,reu}(s)$ to G_i. If s can be resumed on G_i, then $p_{i,reu}(s) = p_{i,reu} < p_{i,new}$; or otherwise $p_{i,reu}(s) = p_{i,new}$. $p_{i,new}$ and $p_{i,reu}$ stands for the processing ability of G_i, where a smaller value means a greater processing ability.

Our load estimation algorithm adopts a model of discrete sliding window. As is shown in Fig. 3, we split the time axis into time slices of duration ΔT. The sliding window at time t covers the time slice t belongs to (end with t) and the former time slice. Suppose that t_d is the beginning of the time slice t belongs to; $l_i(t_d)$ is the total load distributed to G_i during the time interval $[t_d - \Delta T,\ t_d]$ with $l_i(0) = 0$; while $\Delta l_i(t)$ is the total load distributed to G_i during the time interval $[t_d,\ t]$, then $l_i(t)$ is estimated as follows:

$$l_i(t) = \frac{l_i(t_d) + \Delta l_i(t)}{\Delta T + t - t_d} \tag{1}$$

That is, the load of G_i at current time t is estimated using the normalized load distributed to G_i during the former time slice and the current one. Like traditional load estimation algorithms, we have a load information updating period ΔT to wipe off old load information. Moreover, in our algorithm estimating load between updating intervals is enabled, so the goal of real-time estimation is guaranteed. In our work, the way the request distributor obtains the nodes' load information is proactively estimating, instead of periodically receiving load information from the nodes. The monitoring of the nodes' state is not adopted by us, either. As a result, the network delay, bandwidth consumption and extra work on the SSL reverse proxies are minimized.

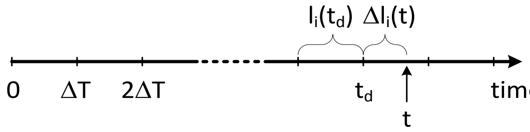


Fig. 3. The model of discrete sliding window.

C. SSLSARD Request Distribution Algorithm

Load balancing and high SSL session resumption ratio are two core concerns when designing request distribution algorithm for SSL reverse proxies. Unfortunately, the two concerns are contradictory. Load balancing focuses on distributing requests to the least loaded node, while high SSL session resumption ratio focuses on distributing requests from the same client to the same target node. Our request distribution algorithm is aimed at making compromise between the two concerns. In the rest of this subsection, we discuss the request distribution decision-making models for both categories of requests described in Section II. B. Some symbols used in this subsection are defined as follows:

- $\mathbf{p}_{new} = (p_{1,new}, p_{2,new}, \ldots, p_{N,new})^{\mathrm{T}}$ is the performance vector of the SSL reverse proxies when a new SSL session is willing to be negotiated;
- $\mathbf{p}_{reu}(s) = (p_1, p_2, \ldots, p_N)^{\mathrm{T}}$, where

  $$p_i = \begin{cases} p_{i,reu}, \text{ if } G_i \text{ has cached } s \\ p_{i,new}, \text{ or else} \end{cases}$$

  is the performance vector of the SSL reverse proxies when SSL session s is willing to be resumed;
- $\mathbf{l}(t) = (l_1(t), l_2(t), \ldots, l_N(t))^{\mathrm{T}}$ is the load vector of the SSL reverse proxies;
- $\mathbf{x} = (x_1, x_2, \ldots, x_N)^{\mathrm{T}}$, where

  $$x_1, x_2, \ldots, x_N \in \{0, 1\}, \sum_{i=1}^{N} x_i = 1$$

is the request distribution decision-making vector, in which the only "1" element indicates the selected node.

- Problem 1:

The first problem is the distributing strategy of requests that willing to negotiate a new SSL session. In our work, the measurement of SSL reverse proxy's load takes performance difference among the nodes into consideration so that the estimated load is normalized. Therefore, the only object of this problem is balancing load among the nodes.

Firstly we quantify this object. The load vector after request distribution is:

$$\begin{aligned} \mathbf{l}'(t) &= \mathbf{l}(t) + \frac{1}{\Delta T + t - t_d} \cdot \mathbf{p}_{new} \circ \mathbf{x} \\ &= (l_1'(t), l_2'(t), \ldots, l_N'(t))^{\mathrm{T}} \end{aligned} \tag{2}$$

Let

$$\bar{l}'(t) = \frac{1}{N} \sum_{i=1}^{N} l_i'(t) \tag{3}$$

We define objective function $f(\mathbf{x})$ as the SSL reverse proxy system's load balance degree, which is measured by the standard deviation of all the SSL reverse proxies' load:

$$f(\mathbf{x}) = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} \left| l_i'(t) - \bar{l}'(t) \right|^2} \tag{4}$$

So we can describe the object as: work out the $\mathbf{x}$ that minimizes $f(\mathbf{x})$. The determination of the optimal solution is described as follows:

Step 1: Work out the $\mathbf{x}$ that minimizes $\mathbf{p}_{new}^{\mathrm{T}} \cdot \mathbf{x}$ (if there are more than one such solutions, select the one that the node it represents is least loaded);

Step 2: Find out the solutions that the nodes they represent are less loaded than the solution worked out in Step 1;

Step 3: Calculate $f(\mathbf{x})$ for all the solutions found in Step 1 and 2, determine the $\mathbf{x}$ that minimizes $f(\mathbf{x})$ as the optimal solution.

- Problem 2:

Considering a request willing to resume session s, a problem is whether distribute it to the node that has cached s or to some node that is much less loaded but hasn't cached s. There are two objects in this problem:

Object 1: Minimize the load caused by this time's request distribution;

Object 2: Balance the load among the nodes.

Our goal is to make compromise between these two objects. We define objective function $f_1(\mathbf{x})$ as the load caused by this time's request distribution:

$$f_1(\mathbf{x}) = \mathbf{p}_{reu}(s)^{\mathrm{T}} \cdot \mathbf{x} \tag{5}$$

So we can describe Object 1 as: work out the $\mathbf{x}$ that minimizes $f_1(\mathbf{x})$.

The load vector after request distribution is:

$$\begin{aligned}\mathbf{l}'(t) &= \mathbf{l}(t) + \frac{1}{\Delta T + t - t_d} \cdot \mathbf{p}_{reu}(s) \circ \mathbf{x} \\ &= \left(l_1'(t), l_2'(t), \ldots, l_N'(t)\right)^{\mathrm{T}}\end{aligned} \tag{6}$$

The definition of objective function $f_2(\mathbf{x})$ here is the same with $f(\mathbf{x})$ in (3) and (4). So Object 2 can be described as: work out the $\mathbf{x}$ that minimizes $f_2(\mathbf{x})$.

We define the utility function of this problem as follows:

$$\begin{aligned}U(\mathbf{x}) &= \alpha_1 \cdot \left\{\frac{f_1(\mathbf{x}) - \min\left[f_1(\mathbf{x})\right]}{f_1(\mathbf{x})}\right\}^{\lambda_1} \\ &\quad + \alpha_2 \cdot \left\{\frac{f_2(\mathbf{x}) - \min\left[f_2(\mathbf{x})\right]}{f_2(\mathbf{x})}\right\}^{\lambda_2}\end{aligned} \tag{7}$$

Parameters α_1, α_2, λ_1 and λ_2 are constants whose values depend on the number and processing ability of the nodes. Through adjusting these parameters, the relative weights on load balance and SSL session resumption can be changed freely, so as to adapt the algorithm to SSL reverse proxy systems of different specifications. Among all the noninferior solutions, the one that minimizes $U(\mathbf{x})$ is the optimal solution of our algorithm. The determination of the noninferior solutions is described as follows:

Step 1: Among all the nodes that have cached s, find out the $\mathbf{x}$ that minimizes $f_1(\mathbf{x})$ (if there are more than one such solutions, select the one that the node it represents is least loaded) as a noninferior solution;

Step 2: Find out the solutions that the nodes they represent are less loaded than the noninferior solution determined in Step 1;

Step 3: Calculate $f_2(\mathbf{x})$ for the noninferior solution determined in Step 1 as a benchmark;

Step 4: Calculate $f_2(\mathbf{x})$ for the solutions found in Step 2, if any $f_2(\mathbf{x})$ is smaller than the benchmark described in Step 3, determine $\mathbf{x}$ as a noninferior solution.

This request distribution algorithm improves the quality of request distribution at the expense of calculation complexity. And the expense of (7) increases as N increases. However, the procedure of determining noninferior solutions greatly reduces the complexity of calculation, providing our request distribution algorithm a wider range of application. In general, the expense of (7) is acceptable for a normal-sized SSL reverse proxy system. For a larger distributed system, a more powerful request distributor is needed.

III. Simulation Model

In this section, we describe the simulation model we designed and implemented based on the system architecture shown in Fig. 2. Based on this simulation model, we will evaluate the performance of SSLSARD in Section IV.

TABLE I: SYSTEM PARAMETERS IN THE SIMULATION MODEL

Parameters of the SSL reverse proxy system	
Number of the SSL reverse proxies (N)	4
Lifetime of SSL session	5min
Public key algorithm	RSA, key size = 2048 bits
Symmetric key algorithm	AES, key size = 256 bits
Parameters of the request distributor	
Period of load information updating (ΔT)	2s
Request distribution decision-making parameters	α_1 = 10; α_2 = 1; λ_1 = 1; λ_2 = 2.

TABLE II: PROCESSING ABILITY OF THE SSL REVERSE PROXIES

	Private key decryption	Symmetric encryption (or decryption)	$p_{i,new}$	$p_{i,reu}$
G_1	90 times/s	1050 Mbps	12.39	1
G_2	75 times/s	840 Mbps	14.92	1.25
G_3	60 times/s	750 Mbps	18.49	1.4
G_4	45 times/s	540 Mbps	24.73	1.94

A. System Parameters

Table I provides system parameters in our simulation model, while Table II provides detailed processing ability of the SSL reverse proxies. The SSL reverse proxy system consists of 4 nodes, whose processing ability for private key decryption and symmetric encryption is different from each other. The lifetime of SSL session for a typical Web site can be set from 5s to 24h [8]. In our simulation, we set it as 5 min. In the configuration of request distribution decision-making parameters, we set α_1 and λ_2 relatively large to add the weight of SSL session resumption with respect to load balancing.

TABLE III: CLIENT REQUEST PARAMETERS IN THE SIMULATION MODEL

Newcome clients per second	Poisson distribution, λ = 40-80
Moment when a new client arrives	Uniform distribution
Requests per user session	Negative binomial distribution, r = 1, p = 0.023
User think time	Generalized Pareto distribution, k = 0.714, σ = 1.429, θ = 2
Content size	128 KB

B. Request Parameters

Table III provides client request parameters in our simulation model. Experience suggests that the number of newcome clients per second obeys Poisson distribution, while the moment when a new client arrives obeys Uniform distribution. The mean value of newcome clients per second varies from 40 to 80, representing different levels of load. The study on online user behavior in [24] provided statistical data of average requests per user session. We had done a curve fitting to these statistical data and as a result, we modeled the number of requests per user session according to a negative binomial distribution. The user think time refers to the time between the retrieval of two successive requests from the same client. In our simulation, the user think time was modeled according to a generalized Pareto distribution [25]. The object of each HTTPS request is a 128KB Web page, which conforms to the assumption that the HTTPS request content size is small enough.

IV. EXPERIMENTAL RESULTS

In this paper, on evaluating the performance of SSLSARD, we considered the following metrics: 1) SSL session resumption ratio, which is defined as the percent of requests that resume existed SSL sessions. 2) Load balance degree, which is measured by the standard deviation of all the SSL reverse proxies' load. 3) Throughput, namely the number of requests the SSL reverse proxy system deals with per second.

On evaluating the performance of SSLSARD, we chose the following two algorithms as benchmarks [5]:

- Round Robin with Weight (WRR): Based on the request distribution policy of Round Robin, this algorithm additionally assigns weights to the nodes according to their processing ability. In our simulation, we assign weights to the SSL reverse proxies according to their ability in private key decryption. The weights of G_1, G_2, G_3 and G_4 are 6, 5, 4, 3 respectively.
- SSL_session_only: This algorithm considers SSL session resumption as the only significant factor in request distribution. Requests from the same client are always distributed to the same node. The first request from each client can be distributed using algorithms like WRR, LL, etc. In our simulation we adopt WRR. The weights of the SSL reverse proxies are the same with the WRR algorithm described above.

A. SSL Session Resumption Ratio

Fig. 4 shows the SSL session resumption ratio of the SSL reverse proxy system. The SSL session resumption ratio of WRR is extremely low, maintaining a level of a bit lower than 1/4. The reason why it is a bit lower than 1/4 rather than exactly 1/4 is, because WRR is too slow, some requests have to wait for a long time to be processed so that their target sessions go out of date. According to the description of SSL_session_only, its SSL session resumption ratio should be the highest among all the request distribution algorithms. The results of our simulation indicate that SSLSARD has an SSL session resumption ratio almost not lower than SSL_session_only. This illustrates that with SSLSARD, the load of the SSL reverse proxy system is almost always well balanced that the situation where a request willing to resume session is need to be distributed to a node that hasn't cached this session rarely happens. As newcome clients per second increases, the SSL session resumption ratio of SSLSARD and SSL_session_only decline slightly. This is because requests have to wait for a longer time to be processed and more sessions go out of date when the number of concurrent requests is large.

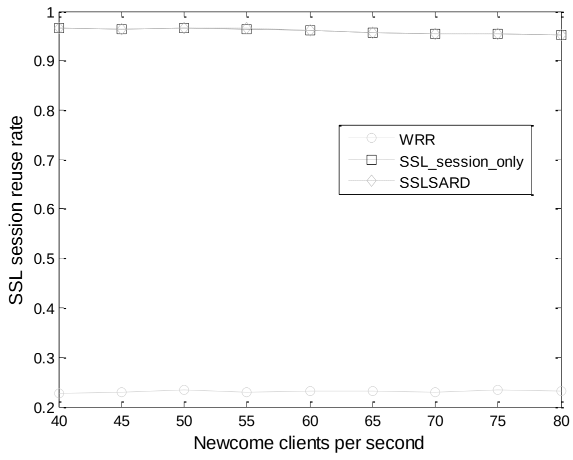
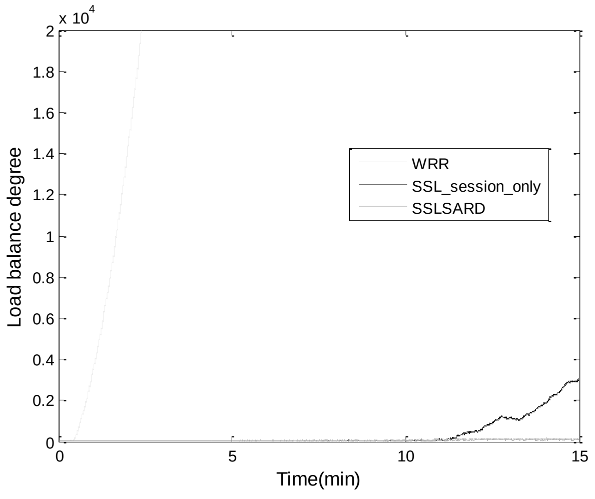


Fig. 4. SSL session resumption ratio of the SSL reverse proxy system

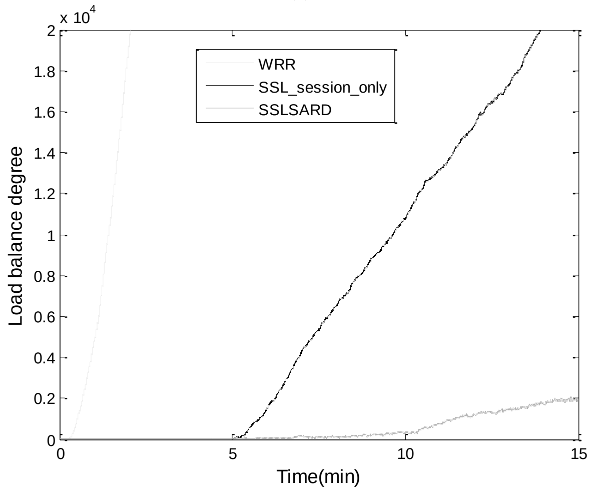
B. Load Balance Degree

We next show the SSL reverse proxy system's load balance degree using the three algorithms. In our simulation, a fixed number of newcome clients visit the Web system per second. Each of these clients generates a number of requests as is described in Section III. B. So at the early stage of the simulation, the number of incoming requests per second increases gradually and finally reaches a steady state. Assuming the queues for requests in the SSL reverse proxies are long enough, Fig. 5 (a) and

(b) provide the SSL reverse proxy system's load balance degree during the first 15 minutes of our simulation using the three algorithms, with the number of newcome clients per second set as 55 and 70, respectively. We can observe that WRR's load balance degree is worst. The amount of computation a request brings to a node differs greatly depend on whether SSL session resumption takes place. Since WRR doesn't take SSL session resumption into consideration, load imbalance is very likely to happen. In SSL_session_only, requests are distributed at the granularity of clients, which may easily cause load imbalance if several clients that request frequently are distributed to some certain node. Due to its excellent load estimation algorithm and SSL-session-aware request distribution algorithm, SSLSARD performs much better than SSL_session_only in load balance degree.



(a)



(b)

Fig. 5. The load balance degree of the SSL reverse proxy system during the first 15 minutes of our simulation. (a) 55 newcome clients per second. (b) 70 newcome clients per second.

C. Throughput

Fig. 6 shows the throughput results of the SSL reverse proxy system using the three algorithms. We can observe that WRR performs far worse than SSL_session_only and SSLSARD. This is because WRR doesn't take SSL session resumption into consideration. On the one hand, requests willing to resume session tend to be distributed to all the nodes, so that the total load of the SSL reverse proxy system is high. On the other hand, not considering SSL session resumption can result in load imbalance. Fig. 6 also shows that, since SSLSARD can achieve high SSL session resumption ratio and load balancing simultaneously, it can deal with more concurrent requests than SSL_session_only and therefore can obtain a relatively large throughput. When the number of newcome clients per second is large, the throughput of SSLSARD and SSL_session_only decrease because their SSL session resumption ratio decrease, as is described in Section IV. A.

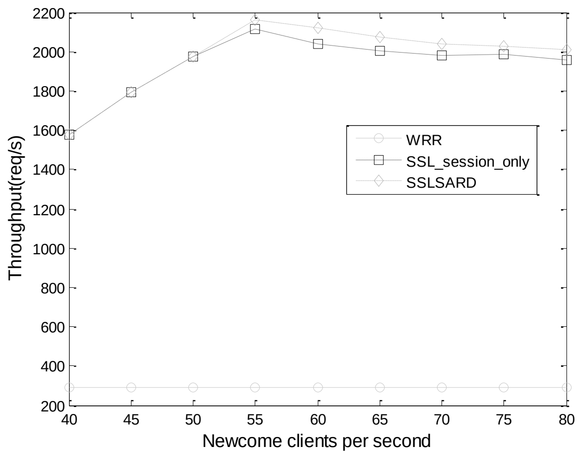


Fig. 6. Throughput of the SSL reverse proxy system

V. Related Work

Some early works studied on the performance of the SSL protocol, pointing out the huge proportion that public key operations take up in SSL overhead in a quantificational way. In [7], [23], the authors presented a detailed description of the anatomy of the SSL protocol and quantified the overhead associated with different components of it. From another point of view, the authors of [26] profiled SSL Web servers with trace-driven workloads, replaced individual components inside SSL with no-ops, then measured the observed increase in server throughput, and provided the upper-bound that may be achieved by optimizing each operation within SSL as well. Many previous works pointed out that the execution performance of SSL can be greatly increased by properly utilizing the SSL session resumption mechanism [5], [7]-[10], [26]. Ref. [9] first suggested defining and measuring SSL performance with a new metric "Percent ID Reuse" (namely the "SSL session resumption ratio" mentioned in this paper) for evaluation.

In the system architecture described in Section II, A, there is a front-end used as request distributor. This architecture has much in common with a Web server cluster. The front-end of Web server cluster can be divided into two categories according to the OSI layer used to distribute the incoming requests, namely layer-4 front-end and layer-7 front-end [27]. Layer-4 front-end selects the target server that is going to attend the request based on the information contained in the TCP SYN packet sent from the client. Then it distributes the request to the target server, so as to establish a TCP connection between the client and the target server. Layer-7 front-end, however, must establish a TCP connection with the client before distributing requests. It then receives HTTP requests from the client and distributes them to the Web servers. The request distributor in this paper should be called "layer-SSL front-end", since it distributes SSL ClientHellos to the back-end nodes. It also needs to establish a TCP connection with the client first before distributing requests, adopting the idea of layer-7 front-end in a Web cluster.

TCP hand-off [22] is one of the most popular request routing mechanisms for layer-7 request distributors [27]. Once the target node is selected, the request distributor hands off its end point of the TCP connection to the target node. With TCP hand-off, responses from the target node can go directly to the client avoiding the request distributor. So this mechanism has very fast response speed. Since SSL can run on top of TCP, request distribution at the granularity of SSL session can also use the TCP hand-off technique.

Among request distribution strategies for Web server system, some are aimed at exploiting the cache locality of the Web servers by distributing requests that ask for a determined Web page to a server that is likely to have it in its cache module. This class of algorithms deals with requests that normally ask for static content in Web pages. The request distribution algorithm proposed in this paper adopted the idea of cache hitting, since the cache of SSL sessions on SSL reverse proxies has much in common with the cache of Web pages on Web servers. The LARD strategy [28], which was proposed by Pai *et al.* in 1998 and designed for Web server cluster, is the first request distribution strategy that takes a request's target content into account. With LARD, a request is always distributed to a Web server node that caches the target, unless this node has a load larger than a threshold. In the latter situation the request is distributed to the least loaded node instead. In 1999, Bunt et al. proposed another request distribution algorithm that concerns cache hitting [29]. In this algorithm, a controllable threshold ε is set. Whenever a request arrives, the request distributor selects the most lightly-loaded Web server that has the object cached as the target Web server, provided that its load is within ε % of the most lightly loaded of all the Web servers. Otherwise, the least loaded Web server is selected. Furthermore, many other works also focused on cache hitting, see our references [30]-[33] for details. There is not much previous research focusing on the request distribution for SSL servers or SSL reverse proxies. Ref. [5] refered to RR and SSL_session_only. Ref. [10] proposed CSSL-SL, a request distribution algorithm for SSL reverse proxies, which introduces a mechanism of broadcasting SSL session information on the basis of RR. Whenever an SSL reverse proxy has negotiated a new SSL session with a client, it broadcasts this session's information to all the other nodes via LAN. As a result, all the nodes can cache this session and the SSL session resumption ratio can be increased. This algorithm needs LAN to broadcast SSL session information. Therefore the SSL reverse proxies can only be deployed at the same geographical location. Moreover, when the number of the SSL reverse proxies is large, the overhead of network communication is tremendous.

Among request distribution strategies, the calculation methods of the nodes' load can be divided into two categories. Methods of the first category need to monitor the nodes' state in real time, measuring the nodes' load as the occupation situation of the nodes' one or multiple hardware resources (such as CPU, memory, disk, etc.) or network resources [34]-[37]. However, a mass of works disagreed with monitoring the nodes' state. They held the view that it is difficult to monitor the nodes' state [38], [39], or the nodes' state information is of little use in load balancing [29], or monitoring the nodes' state may reduce the system's performance [40]. In the second category of load calculation methods, the request distributor obtains the nodes' load information by estimating, rather than by monitoring the nodes' state. The basis of the estimation is diversified. For example, it can be the number of active TCP connections on each node [39], the data throughput of each node [38], the HTTP response time and network delay of each node [41], etc. The idea of not monitoring the nodes' state makes this category of methods simple and fast. So we adopted this idea in designing our load estimation algorithm. In previous works, load calculation methods seldom predict the nodes' load between load information updating intervals in real-time. This idea of real-time predicting was proposed in [34]. We adopted this idea in our estimation algorithm as well.

VI. Conclusions

In this paper, we researched on distributing requests among distributed SSL reverse proxies in a Web system. Aiming at improving the overall performance of SSL reverse proxy system, we proposed an SSL-session-aware request distribution technique for SSL reverse proxies called SSLSARD, which takes both load balancing and high SSL session resumption ratio into consideration. Through simulation, we compared the performance of SSLSARD with that of two benchmarks WRR and SSL_session_only, and drew the following conclusions: 1. Considering SSL session resumption is critical to a request distribution strategy for SSL reverse proxies. 2. Comparing with SSL_session_only, SSLSARD's SSL session resumption ratio is almost not lower, while its load balance degree is much better, due to its excellent load estimation algorithm and request distribution algorithm. 3. SSLSARD can deal with more concurrent requests than SSL_session_only and it can achieve a larger throughput.

REFERENCES

[1] A. Freier, P. Karlton, and P. Kocher, RFC 6101: The Secure Sockets Layer (SSL) Protocol Version 3.0, 2011.

[2] T. Dierks and C. Allen, RFC 2246: The Transport Layer Security (TLS) Protocol Version 1.0, 1999.

[3] T. Dierks and E. Rescorla, RFC 4346: The Transport Layer Security (TLS) Protocol Version 1.1, 2006.

[4] T. Dierks and E. Rescorla, RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, 2008.

[5] J. H. Kim, G. S. Choi, and C. R. Das, "An SSL back-end forwarding scheme in cluster-based web servers," *IEEE Trans. on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 946-957, July 2007.

[6] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, "Revisiting SSL/TLS implementations: New bleichenbacher side channels and attacks," in *Proc. 23rd USENIX Security Symposium*, San Diego, 2014, pp. 733-748.

[7] G. Apostolopoulos, V. Peris, and D. Saha, "Transport layer security: how much does it really cost?" in *Proc. INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, New York, 1999, pp. 717-725.

[8] R. Mraz, "Secure blue: an architecture for a scalable, reliable high volume SSL internet server," in Proc. *17th Annual Computer Security Applications Conference*, New Orleans, 2001, pp. 391-398.

[9] R. Mraz, K. Witting, and P. Dantzig, "Using SSL session ID reuse for characterization of scalable secure web servers," Technical Report RC 22323 (Revised May 5, 2002), IBM Research Division, Yorktown Heights, NY, 2002.

[10] R. Hatsugai and T. Saito, "Load-balancing SSL cluster using session migration," in *Proc. 21st International Conference on Advanced Information Networking and Applications*, Niagara Falls, 2007, pp. 62-67.

[11] M. E. Kounavis, X. Kang, K. Grewal, M. Eszenyi, S. Gueron, and D. Durham, "Encrypting the internet," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 135-146, October 2010.

[12] Netcraft. (October 2015). Web Server Survey. [Online]. Available: http://news.netcraft.com/archives/category/web-server-survey/

[13] Netcraft. (April 2014). Half a Million Widely Trusted Websites Vulnerable to Heartbleed Bug. [Online]. Available: http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html

[14] J. Vehent. (January 2014). SSL/TLS analysis of the Internet's top 1,000,000 websites. [Online]. Available: https://jve.linuxwall.info/blog/index.php?post/TLS_Survey

[15] P. Bilski and W. Winiecki, "Multi-core implementation of the symmetric cryptography algorithms in the measurement system," *Measurement*, vol. 43, no. 8, pp. 1049-1060, October 2010.

[16] A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen, and E. Tischhauser, "ALE: AES-Based lightweight authenticated encryption," in *Fast Software Encryption*, S. Moriai, Ed. Berlin Heidelberg: Springer, 2014, pp. 447-466.

[17] Y. Song, K. Beznosov, and V. Leung, "Multiple-channel security architecture and its implementation over SSL," *EURASIP Journal on Wireless Communications and Networking*, vol. 2006, no. 2, pp. 1-14, April 2006.

[18] N. Lim, S. Majumdar, and V. Srivastava, "Engineering SSL-based systems for enhancing system performance," in *Proc. 2nd ACM/SPEC International Conference on Performance Engineering*, Karlsruhe, 2011, pp. 469-474.

[19] NGINX. (October 2015). SSL-Offloader. [Online]. Available: http://wiki.nginx.org/SSL-Offloader

[20] BlueCoat. (October 2015). Reverse Proxy with SSL - ProxySG Technical Brief. [Online]. Available: https://bto.bluecoat.com/sites/default/files/tech_briefs/Reverse_Proxy_with_SSL.b.pdf

[21] Ericom. (October 2015). Ericom Secure Gateway. [Online]. Available: http://www.ericom.com/securegateway.asp

[22] G. Hunt, E. Nahum, and J. Tracey, "Enabling content-based load distribution for scalable services," Technical Report, IBM TJ Watson Research Center, 1997.

[23] L. Zhao, R. Iyer, S. Makineni, and L. Bhuyan, "Anatomy and performance of SSL processing," in *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, 2005, pp. 197-206.

[24] A. Oke and R. Bunt. "Hierarchical workload characterization for a busy web server," in *Computer Performance Evaluation: Modelling Techniques and Tools*, T. Field, P. G. Harrison, J. Bradley, and U. Harder, Ed. Berlin Heidelberg: Springer, 2002, pp. 309-328.

[25] E. Casalicchio, V. Cardellini, and M. Colajanni, "Content-aware dispatching algorithms for cluster-based web servers," *Cluster Computing*, vol. 5, no. 1, pp. 65-74, January 2002.

[26] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance analysis of TLS web servers," *ACM Transactions on Computer Systems*, vol. 24, no. 1, pp. 39-69, February 2006.

[27] K. Gilly, C. Juiz, and R. Puigjaner, "An up-to-date survey in web load balancing," *World Wide Web*, vol. 14, no. 2, pp. 105-131, March 2011.

[28] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, and W. Zwaenepoel, *et al.*, "Locality-aware request distribution in cluster-based network servers," in *Proc. Eighth International Conference on Architectural Support for Programm*, San Jose, 1998, pp. 205-216.

[29] B. Richard and L. Derek, "Achieving load balance and effective caching in clustered web servers," in *Proc. Fourth International Web Caching Workshop*, San Diego, 1999, pp. 159-169.

[30] L. Cherkasova and M. Karlsson, "Scalable web server cluster design with workload-aware request distribution strategy WARD," in *Proc. Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, San Juan, 2001, pp. 212-221.

[31] Z. Xu, J. Han, and L. Bhuyan, "Scalable and decentralized content-aware dispatching in web clusters," in *Proc. IEEE International Performance, Computing, and*

Communications Conference, New Orleans, 2007, pp. 202-209.

[32] J. Song, E. Levy-Abegnoli, A. Iyengar, and D. Dias, "Design alternatives for scalable Web server accelerators," in *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, 2000, pp. 184-192.

[33] E. V. Carrera and R. Bianchini, "Efficiency vs. portability in cluster-based network servers," in *Proc. Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, Snowbird, 2001, pp. 113-122.

[34] J. Jiang, H. Deng, and X. Liu, "A predictive dynamic load balancing algorithm with service differentiation," in *Proc. 15th IEEE International Conference on Communication Technology*, Guilin, 2013, pp. 372-377.

[35] M. Andreolini, M. Colajanni, and R. Morselli, "Performance study of dispatching algorithms in multi-tier web architectures," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 2, pp. 10-20, September 2002.

[36] X. Zhang, M. Barrientos, J. B. Chen, and M. Seltzer. "HACC: An architecture for cluster-based web servers," in *Proc. 3rd Conference on USENIX Windows NT Symposium*, Seattle, 1999, pp. 155-164.

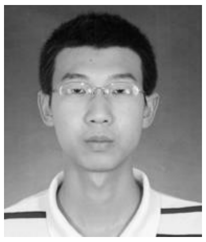
[37] X. Qin, H. Jiang, Y. Zhu, and D. R. Swanson, "Dynamic load balancing for I/O-and memory-intensive workload in clusters using a feedback control mechanism," in *Euro-Par 2003 Parallel Processing*, H. Kosch, L. Böszörményi, and H. Hellwagner, Ed. Berlin Heidelberg: Springer, 2003, pp. 224-229.

[38] K. Dutta, A. Datta, D. VanderMeer, H. Thomas, and K. Ramamritham, "ReDAL: An efficient and practical request distribution technique for application server clusters," *IEEE Trans. on Parallel and Distributed Systems*, vol. 18, no. 11, pp. 1516-1528, November 2007.

[39] V. Ungureanu, B. Melamed, and M. Katehakis, "Effective load balancing for cluster-based servers employing job preemption," *Performance Evaluation*, vol. 65, no. 8, pp. 606-622, July 2008.

[40] E. Casalicchio and M. Colajanni, "A client-aware dispatching algorithm for web clusters providing multiple services," in *Proc. 10th International Conference on World Wide Web*, Hong Kong, 2001, pp. 535-544.

[41] S. Kontogiannis and A. Karakos, "ALBL: An adaptive load balancing algorithm for distributed web systems," *International Journal of Communication Networks and Distributed Systems*, vol. 13, no. 2, 2014, pp. 144-168.



Hai-Tao Dong was born in Heilongjiang Province, China, in 1988. He received the B.S. degree in communication engineering from the University of Electronic Science and Technology of China (UESTC), in 2011. He is currently pursuing the Ph.D. degree in signal and information processing with the Institute of Acoustics, Chinese Academy of Sciences (IACAS). His research interests include network security and broadband network communication.

Lei Song was born in Anhui Province, China, in 1986. He received the B.S. degree from the University of Science and Technology of China (USTC), in 2008 and the Ph.D. degree in signal and information processing from the IACAS, in 2013. He is currently Research Associate in National Network New Media Engineering Research Center, IACAS. His research interests include broadband network communication and network new media.

Jin-Lin Wang was born in Beijing, China, in 1964. He received the B.S. degree in mathematics from the USTC, in 1986 and the M.S. degree in acoustics from the IACAS, in 1989. He is currently a research professor of IACAS, and the director of National Network New Media Engineering Research Center. His current research interests include structure and new service of broadband network, digital media service, and network architecture, etc..

Jun Yang was born in Anhui Province, China, in 1968. He received the B.S. degree and the M.S. degree in engineering from the Harbin Engineering University, in 1990 and 1993, respectively, and the Ph.D. degree in science from the Department of Electronic Science and Engineering of Nanjing University, in 1996. He is currently a research professor of IACAS, and the director of Key Laboratory of Noise and Vibration Research. His research interests include digital signal processing, array signal processing, and acoustic signal processing, etc.