ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/276527759

A load scheduler for SIP proxy servers: Design, implementation and evaluation of a history weighted window approach

Article *in* International Journal of Communication Systems · May 2015

DOI: 10.1002/dac.2980

CITATIONS
17

READS
373

4 authors:

Ahmadreza Montazerolghaem
University of Isfahan
52 PUBLICATIONS **303** CITATIONS
SEE PROFILE

Kazem Shekofteh
Universität Heidelberg
19 PUBLICATIONS **104** CITATIONS
SEE PROFILE

Mohammah-H Yaghmaee
Ferdowsi University Of Mashhad
246 PUBLICATIONS **3,274** CITATIONS
SEE PROFILE

M. Naghibzadeh
Ferdowsi University Of Mashhad
225 PUBLICATIONS **3,072** CITATIONS
SEE PROFILE

All content following this page was uploaded by Ahmadreza Montazerolghaem on 15 October 2020.

The user has requested enhancement of the downloaded file.

INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/dac.2980

A load scheduler for SIP proxy servers: design, implementation and evaluation of a history weighted window approach

Ahmadreza Montazerolghaem, S.-Kazem Shekofteh, M. H. Yaghmaee*,† and Mahmoud Naghibzadeh

Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

SUMMARY

The widespread use of *Session Initiation Protocol* as a signalling protocol has created various challenges. An important one is that its throughput can be severely degraded when an overload happens in the proxy server because of several retransmissions from the user agent. One common approach to overcome this problem is 'load balancing'. A balancer needs to know the status of proxy servers, which are continuously gathered implicitly or explicitly. Implicit methods have averagely less overhead than explicit ones. This paper attempts to prevent throughput reduction by balancing the loads among available proxy servers properly using an implicit mechanism called *History Weighted Average Response time*. The proposed algorithm is robust because it incurs no extra processing to proxy servers. The novelty of the mechanism is making use of 'response time history' to estimate the load being currently processed on servers. By implementing in a real testbed, throughput and scalability are improved compared with an important state-of-the-art similar algorithm. This improvement stems from no need for modification in SIP protocol, easy implementation and application, simple computations for making decision and no need for extra feedback between servers and load balancer. Copyright © 2015 John Wiley & Sons, Ltd.

Received 29 June 2014; Revised 11 February 2015; Accepted 25 March 2015

KEY WORDS: load balancer; scheduler; session initiation protocol; asterisk; overload

1. INTRODUCTION

Voice over Internet Protocol (VoIP) networks are widely used nowadays by different levels of users, such as professional organizations, academic institutions, and home users [1, 2]. One of the most important elements of these networks is the signalling protocol. Session Initiation Protocol (SIP) is the most efficient and commonly used signalling protocol [3]. In spite of many advantages such as mobility and being text-based, in networks connecting millions of users, it cannot perform its function satisfactorily. This protocol is responsible for the following: (1) establishing, (2) managing, and (3) terminating a call session [3]. The call session can be of three types: voice, video and text, or any combination of these. In the protocol stack, SIP is an application protocol and is independent of the transport layer protocol [4]. In these protocols, there are three components: User Agent, proxy server and different messages.

The overview of how messages are transferred between a user agent and a proxy server and the order of doing so is presented in Figure 1. Dashed arrows show the message communications between network hops, and solid arrows indicate the media stream, which is transferred end-to-end after a call is established.

Figure 2 depicts the occurrence of an overload for a proxy server. In this situation, `INVITE` messages are dropped and not processed because of the full queue. This occurs when the proxy does not

*Correspondence to: Yaghmaee Moghaddam, Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran.
†E-mail: hyaghmae@ferdowsi.um.ac.ir

Copyright © 2015 John Wiley & Sons, Ltd.

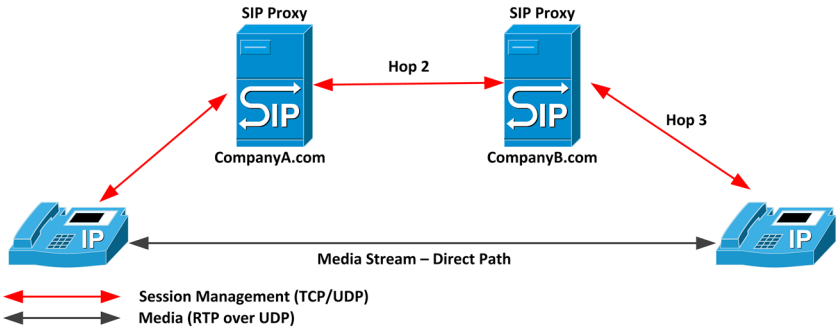

Figure 1. Establishing a session using Session Initiation protocol.

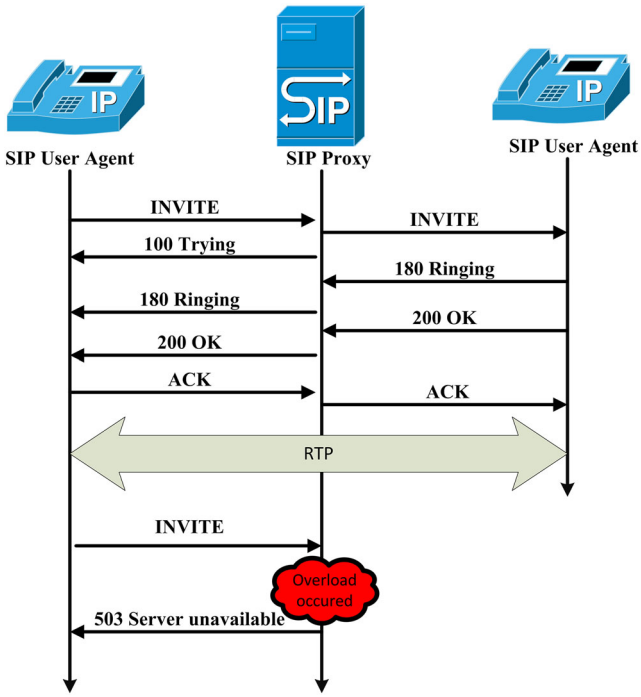

Figure 2. The occurance of an overload.

have sufficient processing resources. A `503 Service Unavailable` response message is then sent from the proxy back to the user agent, which will immediately start to retransmit its `INVITE` message. This will cause the congestion of not only the proxy server but also of the whole SIP network.

The categories of solutions to this problem are (1) the overload control and (2) the message distribution mechanisms. In the former, two types of decisions prevent the overload situation: local and distributed. In the local, each proxy will independently choose its strategy for overload. In the distributed decision, the proxy servers cooperate with each other regarding the decision. In the latter, there exists a third party entity (i.e. the balancer) that is responsible for the proper distribution of messages among proxy servers.

This paper proposes a method for the message distribution mechanism. The balancer is in charge of scheduling the incoming messages to be transmitted to a proxy server. The most important benefit of the proposed load balancer is its message scheduler component, which decides the best destination proxy server based on an effective factor. The decision factor will be then specifically and mathematically analysed.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

The rest of the paper is organized as follows. In Section 2, several related works in this area are briefly discussed. The proposed method and its evaluation are presented in detail in Section 3. Section 4 covers the evaluation of the proposed method in different experiments and presents their analysis. The paper is then summarized in Section 5, where future work is also discussed.

2. RELATED WORKS

There are so many researches about the performance aspects of SIP proxy servers. Because of SIP's popularity and rapidly growing deployments in the Internet Protocol (IP) networks, the SIP server overload [5–12], its security [1, 2, 13–16] and session mobility [17–19] are the main challenges of the network. Mishra *et al.* presents a cut-off priority queuing scheduling with a timeout method [5]. In [8], two PI controllers are used to propose an overload control system, for adjusting the queue length and the proxy servers' CPU load. This is a local method that operates without any knowledge about status of the network. Shen *et al.* introduce an explicit algorithm for window adjustment in [10, 20], but its results can be optimized by considering delay and loss properties of the links.

In [12] a timer-based overload control algorithm is introduced in which in case of using User Datagram Protocol (UDP) as the transport layer protocol, the `INVITE` retransmission timer is adjusted dynamically. To this end, an adaptive `INVITE` retransmission timer is proposed, and the time of establishing a session is enhanced. In [21], Hilt *et al.* provide a detailed analysis of the status of SIP proxy servers under overload and then propose a local method, which can be hard to be applied in a real distributed environment.

The authors of [22] and [23] implemented SIP in both Transmission Control Protocol (TCP) and User Datagram Protocol and compared their performance utilizing OpenSER in case of overload. The results of their experiments show that because application context is not aware of real-time requirements at the transport layer for session-based load, SIP overload congestion collapse cannot be prevented by TCP flow control feedback. In [24] and [25], four implementations of proxies with different memory allocation methods were tested. The results showed that the effective parameters in proxy performance could be divided into two groups: (1) parameters related to the protocol, such as the message length, and (2) parameters related to the implementation of the server, such as the way by which the operating system allocates resources to transactions. SIPperformer, a tool for evaluating the effect of user response latency on SIP server performance, is presented and analysed in [26].

SIPstone [27] is a benchmark in which various metrics are proposed for evaluating the throughput of SIP proxy servers. A similar benchmark for measuring the effect of the operating system, hardware, database and transport layer on SIP proxy performance is provided in [28].

Cheng *et al.* [29] proposed a dependable SIP-based clustered architecture for VoIP and multimedia applications that reduces the number of failed calls when one of the dispatchers or the SIP proxy servers are down; hence, it can balance the proxy servers' load and achieve a fast failover by adjusting the distribution method using CPU loadings of SIP proxy servers. To achieve CPU load status of proxy servers, this architecture uses an external middleware. Applicability of this method depends on the middleware, which may be unreachable in some cases.

The need to solve this problem in other related domains has been already considered, and Harchol-Balter and Schroeder [30] and Crovella, Harchol-Baltera and Murtac [31] propose the Least-Work-Left (LWL) and Join-Shortest-Queue, respectively, for the problem of assigning tasks to servers. A load balancing technique for real websites, especially high accessed ones, is described in [32], which depends mostly on redundancy and avoids sending repeated data and responses. Ciardo *et al.* used the request size to a web server for the load balancing of clustered web servers in [33] which is similar to a 'locality-aware' allocation policy. Some of the existing web server redundancy techniques are involved in presenting a load sharing algorithm in [34]. The paper compares various failover and load sharing methods for registration and call routing servers based on SIP. Server failover techniques are based on the clients, Domain Name Service and replication. Load sharing techniques are based on DNS, SIP identifiers, network address translators and servers with same IP addresses. A two-stage reliable and scalable SIP server architecture is proposed that,

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

in the first stage, the requests are peroxided to one of the second stage server group based on the destination user identifier.

In [35], a load balancer is featured along with three algorithms, which are based on counting the number of transactions or processed sessions on a server. The load balancer performs a session-aware request assignment for routing SIP transactions to the proper back-end node that contains the appropriate session state. The best algorithm among them is Transaction Least-Work-Left (TLWL) [35], which is employed in Section IV as a competitor with which this paper's proposed algorithm is compared. TLWL assigns different *weights* to different transactions depending on their relative costs. An important aspect of SIP protocol is that several types of transactions, especially `INVITE` and `BYE`, can incur significant overheads. `INVITE` transactions have 75% more processing cost than `BYE` ones.

3. LOAD SCHEDULING SCHEME

The overall architecture is illustrated in Figure 3. Clients send their request messages to the load scheduler. The load scheduler then selects the *best* SIP server to process these requests. Choosing the *best* server is the main discriminator among the various solutions to this problem. This paper tries to minimize the average response time of the SIP servers.

The first role of load scheduling is classifying the input request messages. There are two classes of request messages: `INVITE` and `Non-INVITE`. The proposed algorithm focuses on the `INVITE` messages and, hence, the `Non-INVITE` messages remain unscheduled. The main reasons for this approach are as follows: (1) an `INVITE` message is a starting message to establish a session; therefore, it consumes more computation time than a `Non-INVITE` message and (2) when a session is established between a client and a SIP server, the subsequent messages must be sent to the same server from which the corresponding `INVITE` message was sent.

A sample `INVITE` message is presented in Figure 4(a). As illustrated in Figure 4 (b), it is obvious that there are more important field headers to be processed than a `Non-INVITE` message (an `ACK` message). The `Call-ID` field header is the session identifier, and the `Cseq` field header is the sequence number of the messages [36].

The second role takes care of the response messages. Because the destinations of these messages are known, the load scheduler plays the role of a relay in this step.

The load scheduler performs the scheduling mechanism in two major phases: *Detection* and *Selection*. In the Detection phase, the scheduler calculates the current load of the servers based on the weighted average response time (WART) of processing the request messages previously sent to each server. The response time for each request is the time period between sending the `INVITE`

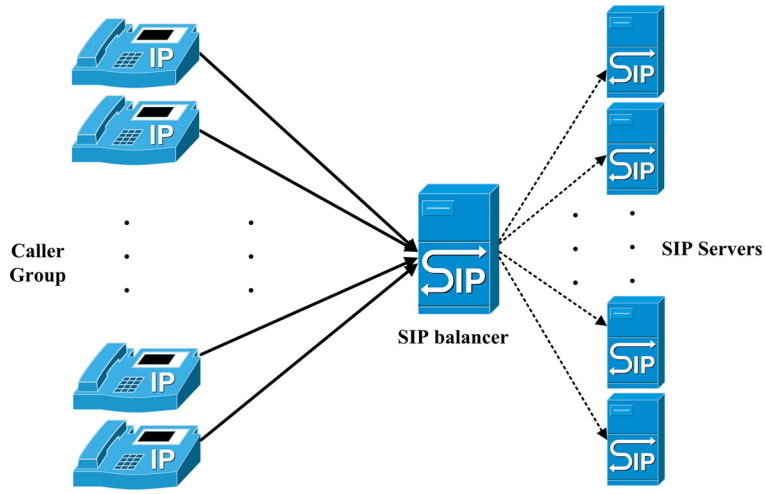

Figure 3. The overall architecture of the proposed algorithm.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

(a)

```
ACK sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asvjh
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 ACK
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

(b)

Figure 4. Contents of (a) an `INVITE` message and (b) an `ACK` message.

message to the selected server and the receipt of the corresponding `200 OK` message. In the Selection phase, the scheduler selects the server that has the minimum average response time for forwarding the input `INVITE` message.

A challenge in the implementation of the proposed algorithm is that the client is registered in a specified server a priori and, in some cases, the field header of the messages should be modified by the load balancer.

In the following, the proposed algorithm is to be defined, but firstly a nomenclature is illustrated in Table I to introduce variables and symbols used in this context.

3.1. The proposed algorithm

The major novelty of the current algorithm is its making use of the average response time factor for increasing the overall throughput. To this end, the load scheduler uses a separate response time window for each server. Each window contains the response time history of a server over time. Because the farthest values of a server's response time are less effective than its recent ones, the size of the window is restricted to a fixed number S_W. The internal architecture is depicted in Figure 5.

The structure of load balancer is such that it should navigate the request messages from users to servers and, vice versa, it should return the messages from servers to users. Internal components of the load balancer are 'Parse Message', 'Classifier', 'Queuing', 'Scheduler' and 'Forward and Drop message'. As a message arrives, the 'Parse Message' component detects the type of the message (request or response). The behaviour of the load balancer for each of them is different, which is described in the following.

- If the received message is a request one,

The 'Classifier' component classifies the request into to two categories: '`INVITE`' and '`Non-INVITE`'.

If the request is a `Non-INVITE` one, it will be forwarded to the server with no extra processing, since it is related to an active session. The server was previously assigned to this session based on the status of the server at the time of establishing the call and it is responsible for all the subsequent requests corresponding to that session. This operation is performed by 'Forward' component.

Table I. Nomenclature.

Variable/symbol	Definition	Variable/symbol	Definition
S_W	size of window	n	number if window slots
$< server_\zeta, session_\varsigma >$	A pair represents that load balancer selects $server_\zeta$ for processing `INVITE` message of $session_\varsigma$	$< server_\xi, session_\varrho >$	A pair represents that $session_\varrho$ was assigned to $server_\xi$ and all its messages are processed by $server_\xi$
r_i	i^{th} response time value	w_i	weight of r_i
$\overline{r}$	average response time	m_i	i^{th} window slot value
r_{new}	new `200 OK` response time value	$\overline{r}_t$	previous average response time value

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

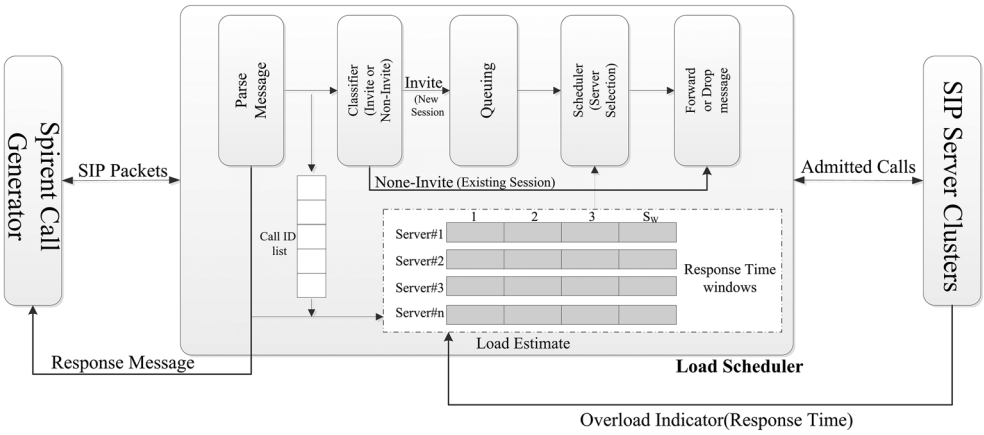

Figure 5. Internal architecture of a load scheduler.

The call ID for active sessions is stored in a list entitled 'Call ID list'. If the received message is an `INVITE` one (representative of a new session), its Call ID field will be stored in the list for detection of this session's future messages. This ID will be kept in the list until its corresponding `BYE` message is received.

After that, all `INVITE` messages are queued for evaluation and determination of the server to be forwarded to. This is the task of the 'Queuing' component. In this paper, a single queue is used and there is no discrimination between `INVITE` messages. In future works, however, it is possible to consider separate queues based on necessity or network status (e.g. for emergency calls) and consequently consider different levels of priorities for each.

Deciding about assigning the best available server to an `INVITE` message is done by the 'Scheduler' component. To this end, this component uses the information provided by the 'Load Estimation' component.

After selecting the best server, the 'Forward' component is responsible for sending `INVITE` message to the selected server.

If the 'Scheduler' component notices that no server is available (i.e. all the servers' capacity is probably full), the `INVITE` message is delivered to 'Drop' component.

- If the received message is a response one,

If it is a message other than `200 OK`, it will be forwarded to the defined user without any extra processing, according to the information stored in Call ID list and 'To' and 'From' field inside the message.

If it is a `200 OK` message, response time parameter will be calculated by HWAR and stored in corresponding 'Response Time Window' for further decisions. 'Load Estimation' component performs this operation. After that, the `200 OK` message is sent to the defined user.

The load scheduler calculates the average value for each window and selects the server with the minimum average value. The Call ID list contains pairs of $< server_\zeta, session_\varsigma >$. Such a pair indicates that the $server_\zeta$ is chosen for the processing $session_\varsigma$. Once the $server_\xi$ finishes processing of `INVITE` message of $session_\varrho$, it sends a response back to the load scheduler. The load scheduler then plays its second role as a relay by adding the $< server_\xi, session_\varrho >$ pair to the Call ID list and updating the window corresponding to that server.

(1) *Fixed-Sized Window Average Response Time Algorithm*: The first load scheduler is equipped with a fixed-sized window and the average response time is calculated for its slots. 5, 10, 15 and 20 are chosen for S_W and the results are evaluated in Section IV.
(2) *Fixed-Sized Window Weighted Average Response Time* (*FWAR*) *Algorithm*: The most important limitation of the Fixed-Sized Window Average Response Time Algorithm method is that all of the slots have an identical effect on the average response time. Because recent response time values indicate the current processing power of the server, a WART is used, that is, the

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

recent response time values have a larger weight than those of the older ones. It is calculated using equation (1).

$$\overline{r} = \frac{\sum_{i=1}^{S_W} w_i \cdot r_i}{\sum_{i=1}^{S_W} w_i} \tag{1}$$

$\overline{r}$ indicates the average response time, w_i is the weight of r_i and r_i represents the i^{th} response time value. In this paper's experiments, relative weights are used, which mean that the weight of the most recent response time is S_W and the weight of the least recent one is 1. This is expressed in equation (2).

$$w_{S_W} = S_W, w_{S_W-1} = S_W - 1, \ldots, w_1 = 1 \tag{2}$$

(3) *History Window Weighted Average Response Time* (*HWAR*) *Algorithm*: The most important drawback of the two previous algorithms is determining the optimal value of S_W. This optimal value is very sensitive to the underlying application and environment. In this paper, hence, the problem is solved from a mathematical point of view. The proposed solution keeps track of the whole history while requiring low computations for calculating the WART. In other words, the *FWAR* method is used to its advantage while its weakness is also solved.

To this end, upon the receipt of a new response time value (r_{new}) at time instant t, the previous average response time value $\overline{r}_t$ would be considered as a single value of response time, and the new average response time would be calculated using r_{new} and $\overline{r}_t$. Therefore, just two slots need to be held and the computations will decrease remarkably. It is obvious that $\overline{r}_t$ includes all the previous values of the response time with different weights. 1 and $\frac{1}{2}$ are utilized as the weights of r_{new} and $\overline{r}_t$, respectively. The approach is inspired by a similar one successfully employed by a Linux operating system for scheduling non-real-time processes. In each time epoch, the number of CPU time quantum given to a processor is equal to its priority level plus its credit. The credit of a process is computed based on its CPU usage in previous epochs [37]. These weight values are such that very old response time values will have a negligible effect on the WART because they are multiplied many times by $\frac{1}{2}$; its impact, however, will never become zero, so the proposed solution is expected to have high accuracy.

According to the aforementioned discussions, a recursive relation for the new WART, with respect to previous response time values, is described in the following. Figure 6(a) depicts a sample window in which m_0 represents the first response time value, m_1 represents the second response time value and so on. In Figure 6(b), it is shown that how these values (m_0, m_1, …) are summarized using only two slots.

$$r_0 = m_0 \tag{3}$$

$$r_1 = \frac{\frac{1}{2}r_0 + m_1}{\frac{3}{2}} = \frac{2}{3}\left(\frac{1}{2}r_0 + m_1\right) = \frac{1}{3}r_0 + \frac{2}{3}m_1 \tag{4}$$

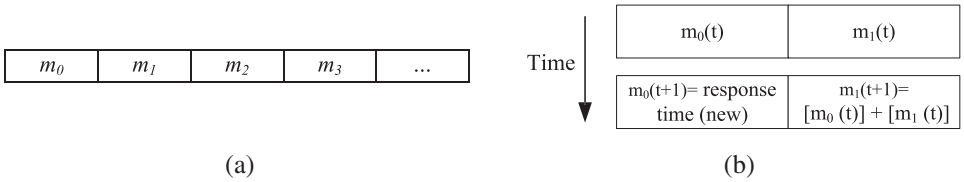

Figure 6. (a) A sample window and (b) how window slots are set in History Weighted Average Response time.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

$$r_2 = \frac{\frac{1}{2}r_1 + m_2}{\frac{3}{2}} = \frac{1}{3}r_1 + \frac{2}{3}m_2 \tag{5}$$

In (3), r_0 is the first WART which is equal to m_0 because there is no other response time value available. r_1 is the second WART which is calculated using (4). This will be repeated in (5). To show the recursive relation, the calculation of r_3 is shown in the following with more details (6).

$$r_3 = \frac{\frac{1}{2}r_2 + m_3}{\frac{3}{2}} = \frac{\frac{1}{2}\left(\frac{\frac{1}{2}r_1 + m_2}{\frac{3}{2}}\right) + m_3}{\frac{3}{2}} = \ldots = \frac{2}{3}\left(\frac{\frac{1}{2}m_0}{9} + \frac{m_1 + 3m_2 + 9m_3}{9}\right) \tag{6}$$

It is evident that r_3 is affected by m_0, m_1, m_2 and m_3, which have increasing weights. From (3) to (6), it can be concluded that the general recursive relation of r_n can be written as (7).

$$r_n = \frac{1}{3^n}m_0 + \frac{2}{3^n}\sum_{i=1}^{n} 3^{i-1} \cdot m_i \tag{7}$$

To show that the recursive relation in (7) converges to the WART, it is proven that the sum of coefficients equals 1 (8) when employing the principle of induction.

$$\textit{Sum of coefficients}: \frac{1}{3^n} + \frac{2}{3^n}\sum\nolimits_{i=1}^{n} 3^{i-1} = 1 \tag{8}$$

$$n = 1 \rightarrow \frac{1}{3} + \frac{2}{3}\left(3^0\right) = 1 \tag{9}$$

$$n = k \rightarrow \frac{1}{3^k} + \frac{2}{3^k}\sum_{i=1}^{k} 3^{i-1} = 1 \tag{10}$$

$$n = k + 1 \xrightarrow{?} \frac{1}{3^{k+1}} + \frac{2}{3^{k+1}}\sum_{i=1}^{k+1} 3^{i-1} = 1 \tag{11}$$

By expanding (11) and multiplying (10) by $\frac{1}{3}$, the correctness of the relation is proved in (12).

$$\begin{aligned} &n = k \Rightarrow \frac{1}{3^k} + \frac{2}{3^k}\sum_{i=1}^{k} 3^{i-1} = 1 \Rightarrow \frac{1}{3^{k+1}} + \frac{2}{3^{k+1}}\sum_{i=1}^{k} 3^{i-1} = \frac{1}{3} \\ &\frac{1}{3^{k+1}} + \frac{2}{3^{k+1}}\sum_{i=1}^{k+1} 3^{i-1} - \left(\frac{1}{3^{k+1}} + \frac{2}{3^{k+1}}\sum_{i=1}^{k} 3^{i-1}\right) = \frac{2}{3} \Rightarrow \frac{2}{3^{k+1}}3^k = \frac{2}{3} \Rightarrow \frac{2}{3} = \frac{2}{3} \end{aligned} \tag{12}$$

■

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

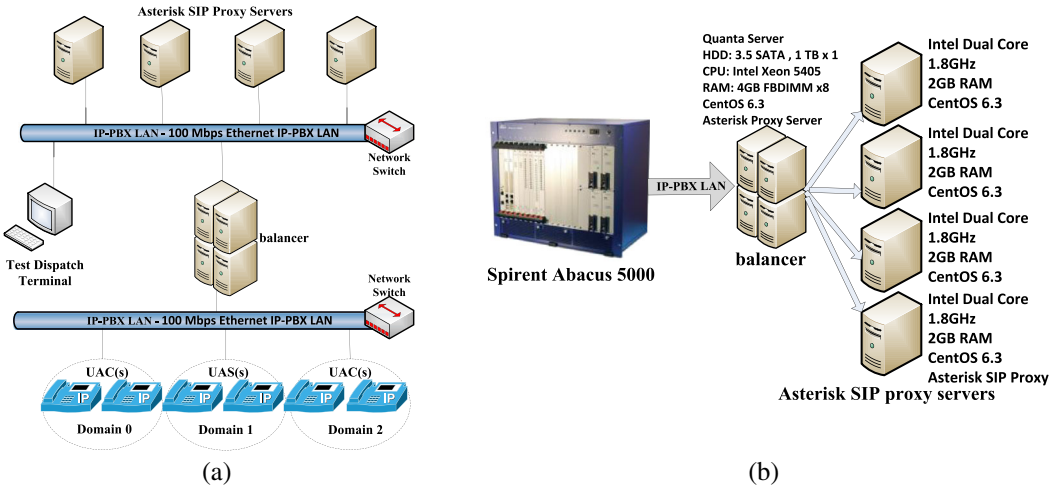

Figure 7. (a) Network configuration details and (b) specification of proxy servers.

Table II. Hardware testbed characteristics.

Feature	Load balancer	Servers
Quantity	1	8
Central processing unit	Intel Xeon 5405	Intel Dual Core 1.8GHz
Random-access memory	4 GB	2 GB
Operating system	CentOS 6.3	CentOS 6.3
Session initiation protocol proxy	Asterisk 1.4	Asterisk 1.4

4. PERFORMANCE EVALUATION

4.1. Specifications of the environment

In this section, the details of the network configurations and proposed method's implementation are given. Figure 7 and Tables II and III illustrate the details of the network configuration used to perform the experiments. A Spirent Abacus 5000 device is a powerful call generator, which is capable of generating 10000 calls per second. In order to evaluate the performance of the proposed method in a condition very close to that of the real world, commercial Asterisk SIP proxy servers are utilized.

Both servers and balancers are Asterisk, and all the required modifications for implementing different mechanisms for scheduling and load balancing among servers are performed on an open source Asterisk proxy server. To profile the results of these changes, the open source tool Oprofile is used.

The IP-PBX laboratory's[‡] testbed for running experiments is illustrated in Figure 8. There is a Spirent call generator that supports different types of load distributions, such as Poisson and Trapezoid, and acts as user agents. The study chose Poisson distribution because it behaves more normally than the Trapezoid. The load profile configuration illustrated in Figure 8 represents the timing parameters of calls. The Asterisk proxy server can report comprehensive statistics about the status of the server during experiments. In the following, the reports of both the Spirent and Asterisk servers are used for evaluating the proposed method.

[‡]http://ippbx-lab.um.ac.ir/

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

Table III. Some of the properties and parameters that can be set in Spirent Abacus 5000.

Workload Generator	Parameter	Description
Spirent Abacus 5000	Number of iterations	3 times for each test
	Internet Protocol (IP)	IP address of the load balancer
	Gateway address	Gateway address (Here the load balancer)
	number of Originate	Number of callers*
	number of Terminate	Number of call-receivers*
	calling profile	Traffic generation method: For example, a user might generate the constant traffic of 100 cps with Poisson distribution or generate the variable rates of 100, 200, and 300 for 100 s.

*Variable for each test – From low load [10 cps] to heavy load [3000 cps]

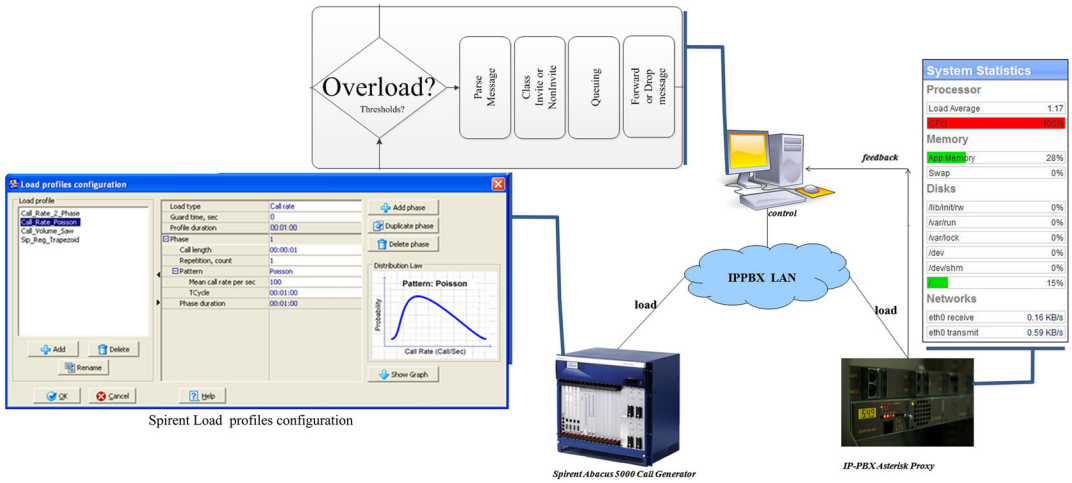

Figure 8. The testbed.

Because the underlying network in which the implementation and evaluations were performed was based on the open source Asterisk system (the most common open source VOIP system which implements SIP proxy servers), it is easy to develop the proposed algorithm as s module that can be installed on any server.

4.2. Performance metrics

(1) *Average Response Time*: The average value of response time over various call rates (call per second).
(2) *Throughput*: The number of successful sessions established per time.
(3) *Scalability*: The metric representing how an increase in the number of SIP servers can affect the proposed method's performance.
(4) *Throughput & Average Response Time* (*heterogeneous back ends*)
(5) *Capacity*: Maximum number of calls that can be handled.
(6) *Network Bandwidth*.
(7) *CPU Utilization* and *Profiling*: Measuring this metric shows the overhead of the load scheduler module imposed on the server's main functionalities.
(8) *Stateful* and *Stateless Proxy Effects on Efficiency*.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

4.2.1. Results for the average response time. In the first set of experiments, the results of the average response time of the three proposed methods are compared with the best proposed algorithm, that is, TLWL-1.75 [35] versus the common round-robin method over eight servers. Different volumes of loads are generated and sent to the load scheduler, starting from 10 cps (low load) to 3000 cps (heavy load). The round-robin method is the worst because it does not note the servers' current load and selects a server unconsciously. TLWL-1.75 dispatches the `INVITE` message to the server with the lowest load of work. A server's work is defined as the number of `INVITE` and `BYE` transactions currently processed on that server, considering a weight of 1.75 and 1 for `INVITE` and `BYE` transactions respectively. Figure 9 illustrates that the average response time decreases in the proposed method because WART is a better reflector of the server's current status rather than the number of active transactions.

The reason is that TLWL is exclusively designed to minimize the workload (i.e. weighted sum of transactions). By employing this strategy, it tries to enhance the average response time, whereas HWAR focuses on the 'response time' parameter and the history of servers. The decision factor in HWAR keeps track of the state of the servers more intelligently and realistically than TLWL.

4.2.2. Results on throughput. The second set of experiments covers the system throughput. Recalling the previous discussion on the average response time, one can realize that the greater the average response time for a server, the longer the proxy queue. A server with less average response time has a shorter proxy queue length. Thus, it is able to process more incoming messages that leads to a higher throughput.

Ideally, it is expected that the performance of eight servers is eight times better than that of a single server. In other words, while the throughput of a single SIP server is about 300 cps, the maximum expected throughput achieved for eight servers is about 2400 cps. Figure 10 presents the throughput of different algorithms employing eight servers.

Among the others, HWAR provides the maximum throughput of 2470 cps. FWAR performance is very close to that of HWAR; however, HWAR has superior performance because it estimates response time better. The three algorithms that provide better response time are also the best in terms of throughput. It is obvious that even using a simple algorithm for load balancing is better than none. This can be seen when the round-robin algorithm shows a 2130 cps performance, which is 86% of HWAR.

In HWAR, no call is lost under the rate of 2470 cps. As long as incoming load is less than the load balancer capacity and proxy servers have the ability of responding to all incoming calls, the load balancer buffer remains almost empty because each packet is processed and serviced once it arrives at the balancer. But upon a sudden increase in load higher than 2470 cps, two reasons will have the buffer become full rapidly: (i) the calls more than proxy servers' capacity and (ii) repeated requests originating from previous cancellation in being serviced. Buffer overflow and packet loss are absolute results of this situation. When the incoming loads decrease to a point less than the servers' capacity and their queues get empty gradually, the throughput increases again.

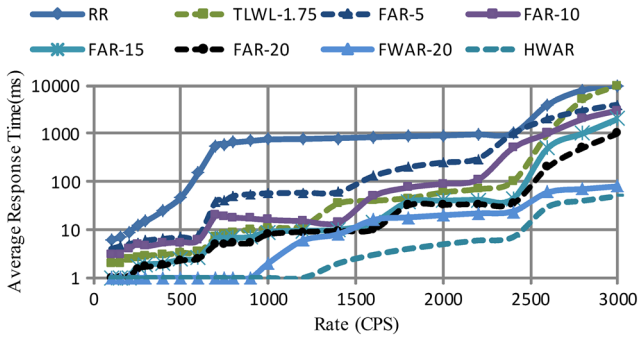

Figure 9. Evaluating the average response time of the proposed method with eight Session Initiation Protocol Proxy Servers.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

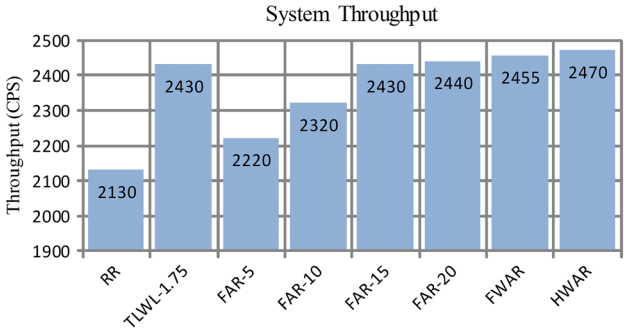

Figure 10. Throughput of various algorithms with eight Session Initiation Protocol proxy servers.

4.2.1. Results on scalability. This paper observed the performance of the system over a different number of servers. Figure 11 illustrates that increasing the number of servers (up to 10) does not have a destructive impact on the performance of the load scheduler nor will it become bottlenecked. The experiments indicate that the proposed method can handle sessions of up to 10 servers.

In this context, scalability appears in terms of the number of servers. In other words, it states the question of how many servers can be added to the network without causing the load balancer to misbehave. Briefly, it can be expressed as the ability to keep the performance, usefulness and usability regardless of the number of servers. In the future, hence, it is possible to support more servers and therefore more users, according to scalability of the load balancer. In our experiments, it is shown that the load balancer equipped with HWAR can support up to 10 servers with no difficulty and distribute the loads among them properly. It is important to remark that by increasing the number of servers, the load balancer, then, needs to store more information about their status. The load balancer should accurately keep track of the servers' status in order to increase the throughput to its possible maximum level. Increasing the number of the servers leads to an increase in the volume of processing and incurred difficulty in decisions until it reaches to a threshold, for which adding a new server makes the load balancer misbehave. Consequently, it loses its processing power and brings about the reduction of throughput and the number of active sessions.

Preparing a load balancer to work on four servers the same as when it works on just two servers is highly dependent on its mechanism of detection and operation. The proposed load balancer can support up to 10 servers with no problem. The main reason is its powerful algorithm HWAR.

The proposed algorithm benefits from intrinsic self-clocking behaviour by using implicit feedback, which results in high stability and scalability because (1) it does not need to perform high computations and (2) it attends to history of response times of each server. Besides, the proposed algorithm is simpler for implementation than similar works because of two reasons: (1) It would not interfere with the processing already being performed on servers and the servers do not need

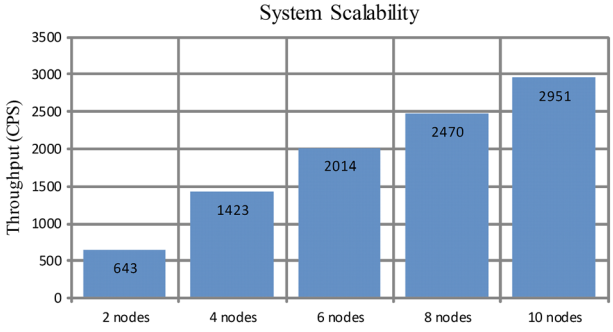

Figure 11. Scalability.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

to supervise their resources and try to detect a potential overload or reject a new incoming call and (2) no extra feedback needs to be transferred between the load balancer and a server and overload detection is done using only regular process of signalling and average response time values.

Furthermore, the proposed algorithm is considered to be an active and a proactive approach since it tries to detect and prevent overload situation before the queues gets full, the messages become overflown, and the resources of the servers be saturated.

Another benefit of the proposed method is that it does not need any cooperation between proxy servers and load balancer nor any modification in SIP protocol.

All of these aspects provide the scalability of the load balancer.

4.2.4. Results on throughput and average response time (heterogeneous back ends). The following sections investigate the functionality of load balancing algorithms when servers have disparate processing powers and capabilities. In most configurations, the expectation that all servers have a similar processing power is unrealistic. Servers are heterogeneous in general. In this experiment, the load balancer dispatches the requests to different servers. The maximum processing power of the first server is about 300 cps and, for the second and third servers, it is 150 cps and 75 cps, respectively. Ideally, the proposed algorithm is expected to have a rate of 1.75 times higher than that of the first server, that is, 525 cps in this heterogeneous environment.

The throughputs of five load balancing algorithms are shown in Figure 12. As is obvious in Figure 12, HWAR exhibits a maximum throughput of 511 cps, which is very close to the optimal rate. TLWL-1.75 ranks second with 491 cps. The throughput of the round-robin algorithm is considerably low. Response times are given in Figure 13. HWAR and FWAR deliver a minimum response time on the average, in contrast to the round-robin's maximum response time. These results indicate that two dynamic algorithms, HWAR and FWAR, adapt to heterogeneous environments much better than other algorithms because they continuously observe server response times and try to balance them. Because the first server responds to requests two times faster than the second one, the ratio of the calls allocated to it is also about two times more than that of the second one and four times more than that of the third one. It should be noted that this happens while the load balancer has no knowledge about this difference in server processing powers.

4.2. 5. Results on capacity. This section focuses on the functionality of the load balancer itself so as to ascertain how much load can be handled before a bottleneck is reached. The Spirent Call Generator acts as both the client and server. In the proposed method, about 10000 calls per second can be managed without any bottleneck. Figure 14 shows that Spirent can operate as both the client and server with the load balancer placed between them.

Figure 15 shows the observed throughput for loads submitted to the HWAR and TLWL-1.75 load balancers. TLWL-1.75 can support 5500 cps before an overload occurs. The corresponding value for HWAR is 6500 cps. If the peak throughout of each SIP server is 300 cps, the load balancer is able to support 18 servers and 21 servers for TLWL-1.75 and HWAR, respectively.

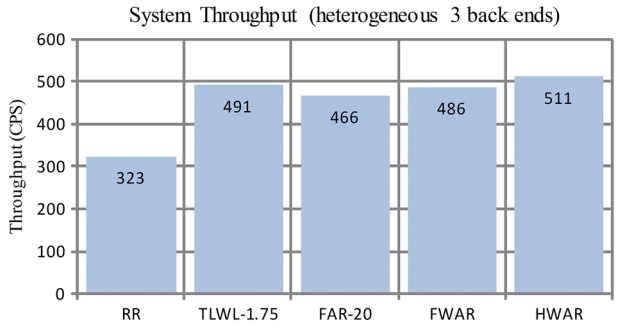

Figure 12. Throughput (heterogeneous back ends).

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

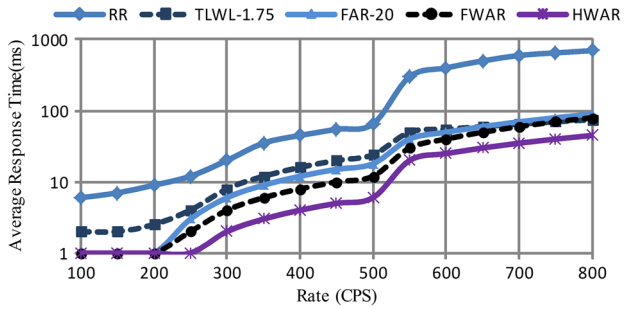

Figure 13. Average response time (heterogeneous back ends).

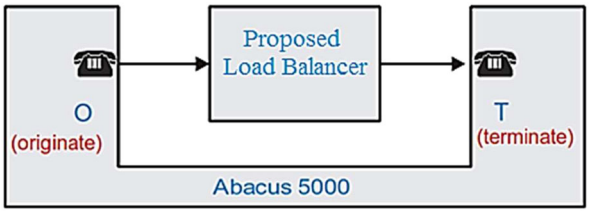

Figure 14. The placement of the load scheduler in the Spirent Abacus 5000 testbed.

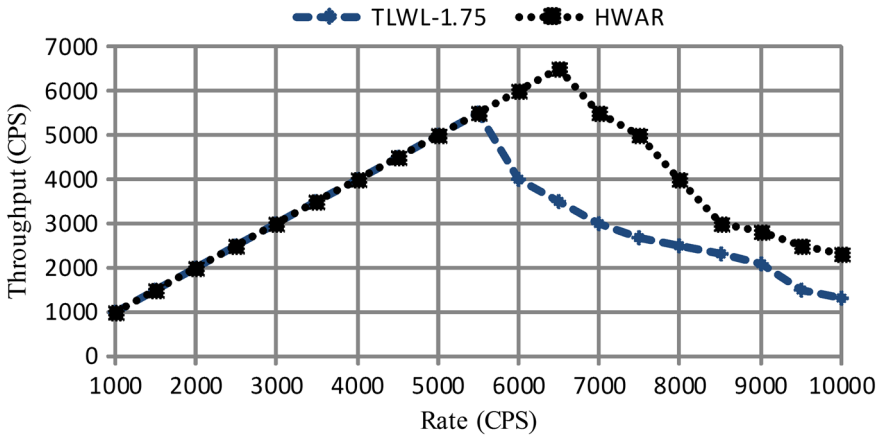

Figure 15. Session Initiation Protocol load scheduler throughput.

4.2.6. Results for network bandwidth versus load. It is illustrated in Figure 16 that when the load balancer is loaded with traffic (in our experiments, up to 10000 cps), the consumed bandwidth in the testbed will never reach 400 Mbps. Even with 6500 cps, the bandwidth will not exceed 100 Mbps; hence, the bandwidth will never become a bottleneck. This illustrates that the overhead of the proposed algorithm on the connection links is very low. In other words, the proposed algorithm uses the network resources such as bandwidth well and does not perform its task by using the resources more than normal. Two samples of the resources which can be taken into account are bandwidth and CPU of load balancer. In Figures 16 and 17, it is illustrated that HWAR can use them effectively.

As described previously, HWAR does not utilize heavy computations for performing its duties and tries to balance the loads by a simple and easy-to-implement mathematical model alongside monitoring response time parameter. This process needs low CPU computational requirements. Also, HWAR is an implicit mechanism rather than an explicit one. This means that being aware of the status of the servers and being capable of detecting the overload situations within each server, it does not need to transmit any extra messages. By increasing the traffic, only request and ordinary

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

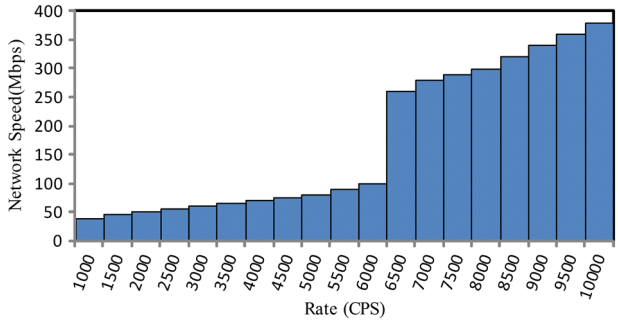

Figure 16. Network bandwidth versus load.

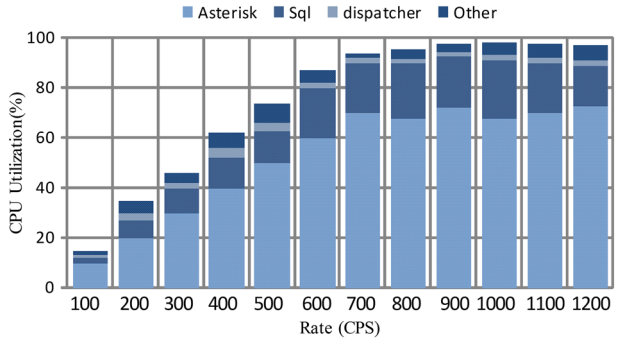

Figure 17. Utilization.

response messages will increase (i.e. HWAR does not incur any extra message), and the bandwidth is increased proportionally. This growth is shown in Figure 16.

4.2.7. Results for central processing unit utilization and profiling. In Figure 17, it is evident that the overhead of the proposed method's dispatcher module is very low in comparison with the other roles of the SIP server. This is because the proposed method uses a simple mechanism to detect overload occurrence and does not involve complex computations and processing. Figure 17 confirms that with a rate of 6500 cps, almost all of the CPU resources are used up.

4.2.8. Stateful and stateless proxy effects on efficiency. It is possible for proxy servers to be configured for various configurations depending on network requirements and other conditions, for example the behaviour of the SIP proxy server with SIP transactions can be either stateful or stateless. A SIP transaction includes a request and all related responses, which are transmitted between two adjacent SIP components. A stateful proxy is required to maintain the information about the state of each transaction for the purpose of performing operations such as possibly retransmitting signalling packets.

In a stateless proxy server, no transaction will be established, and the proxy server will not allot timer for accomplishing the signalling process and retransmission; hence, no `100 TRYING` message will be issued from the server. Processing messages in a stateful server includes creating, searching, updating and deleting of transaction state and also creating and tuning several timers for each transaction. Therefore, each request that is managed in a stateful manner requires more memory, more processing CPU time, keeping track of corresponding timers and consumes more resources overall. Although throughput seems to increase when using stateless servers, delays may occur. For instance, when the loss ratio of packets over network links is low, stateless servers, which are unable to perform retransmission, may take longer time for call establishment. Proxy servers, responsible for some special tasks such as forking, NAT traversing, retransmitting and managing user accounts and invoices, are often stateful, and those accounts for load balancing,

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

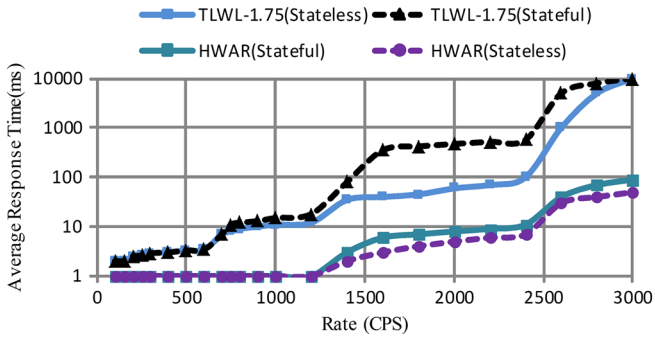

Figure 18. Average response time of stateful and stateless proxies (with eight Session Initiation Protocol proxy servers).

interpretation and retransmission are stateless. Therefore, some methods, such as TLWL-1.75, that need to record information about transactions may be faced with warpath situations. Figure 18 shows the effect of stateless versus stateful servers in regard to delays of establishing a call. It is clear that stateful configurations may lead to an increase in call establishment delays. This is because of the server's consumption of more memory and processing resources for establishing a call. In Figure 18, it can be concluded that the cost of being stateful when using HWAR is much less than when using TLWL-1.75, because the server equipped with TLWL-1.75 may be faced with a bottleneck situation sooner that the other server equipped with HWAR.

Figure 19 represents the throughput of the load balancer in two cases: stateless and stateful. When passing a specified threshold, not only does the throughput not increase but also decreases. This increase is much more when the server is in stateful configuration and uses the TLWL-1.75 algorithm.

Therefore, the load balancing server being stateful will not have positive effect on the increase of throughput, even if TLWL-1.75 is used, which requires maintaining more information.

4.3. Result summarization

A tabular comparison between different methods for mean and standard deviation values is represented in Table IV. As the results show, high value of standard deviation in RR method for all types of experiments comes from not being intelligent compared with HWAR. HWAR shows a more stable behaviour because it is based on a mathematical model and it has the least standard deviation among other methods. RR distributes the incoming requests with no attention to the current load of the servers. The first request will be assigned to the first server, the second one will be assigned to the second server … and the ninth request will be assigned to ninth server, then n + 1th request will be assigned to the first server and this process continues. Hence, it is very possible that a

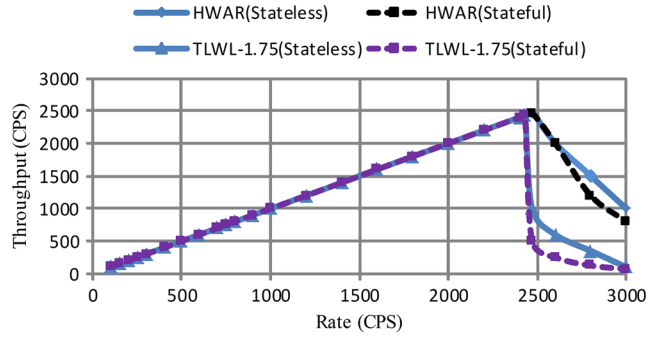

Figure 19. Throughput of stateful and stateless proxies (with eight Session Initiation Protocol proxy servers).

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

Table IV. Comparison of mean and standard deviation between different methods.

	Throughput								Average response time							
	Eizght proxy servers		Heterogeneous back ends		Stateful on 3000 cps		Stateless on 3000 cps		Eight proxy servers on 3000 cps		Heterogeneous back ends on 800 cps		Stateful on 3000 cps		Stateless on 3000 cps	
Method	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
History weighted average response time	2470	4.08	511	4.54	800	9.20	1000	8.16	50	2.44	45	2.16	90	4.96	50	2.82
Fixed-sized window weighted average response time-20	2455	4.89	486	5.09	N/A	N/A	N/A	N/A	80	3.26	80	4.24	N/A	N/A	N/A	N/A
Fixed-sized window average response time-20	2440	6.53	466	7.07	N/A	N/A	N/A	N/A	1000	8.16	90	5.65	N/A	N/A	N/A	N/A
Transaction least-work-left -1.75	2430	5.71	491	4.96	70	16.32	100	20.41	10000	12.04	75	6.16	10000	28.28	10000	42.43
Round robin	2130	24.49	323	28.99	N/A	N/A	N/A	N/A	10000	81.64	700	35.59	N/A	N/A	N/A	N/A

NA, not applicable.
μ: Mean. σ: Standard deviation.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

potential scenario can be occurred, such that most of the heavy requests will be assigned to the same server, which results in a bottleneck. TLWL, on the other hand, monitors the status of the servers by counting the number of active requests being processed by them. This approach may result in preventing that scenario from happening. Therefore, the mean value is improved and standard deviation is decreased significantly. Finally, HWAR also monitors the status of the servers by considering a history of response time values for each server. This leads to better mean and standard deviation values compared with TLWL.

5. CONCLUSION AND FURTHER WORKS

This paper examined the problem of load balancing in the SIP protocol of VoIP connections to prevent an overload status. Few similar works in this area have tried to solve this problem by counting the number of messages, transactions and so on such as TLWL [35]. This paper attempts to prevent throughput reduction by balancing the loads among available proxy servers properly using an implicit mechanism called *HWAR*.

In the proposed architecture, each server has a corresponding window in the load scheduler. The contents of each window are the history of the server's response time over time. Although the number of slots of the window maybe just two, it has an infinite memory capable of recording an immense number of response time values. This strategy, in conjunction with using average response time as the detection factor, is the key advantage of the proposed method. Besides, choosing appropriate weights in HWAR leads to the convergence of the average response time, which was proved earlier.

The proposed method does not need heavy computations for making decisions; it is easier than most of the state-of-the-art works in terms of implementation and application. In addition, no extra feedback is transmitted between proxy servers and the load balancer, and no modification is needed in SIP protocol. The implementation of the proposed method was performed on real SIP servers. The evaluations showed approximately a 5–15% performance enhancement in average response time compared with the best similar works and a 20% enhancement compared with the round-robin algorithm.

We are currently working on extending the current work to model different proxy server statuses mathematically. Further extension can include an auxiliary window for `BYE` messages.

ACKNOWLEDGEMENTS

The authors would like to thank the IP-PBX Laboratory of Ferdowsi University of Mashhad for its financial support of this scientific project.

REFERENCES

1. Xie Q. A new authenticated key agreement for session initiation protocol. *International Journal of Communication Systems* 2012; **25**: 47–54.
2. Christian Callegari RGG, Giordano S, Pagano M. Security and delay issues in SIP systems. *International Journal of Communication Systems* 2009; **22**: 1023–1044.
3. Rosenberg J, Schulzrinne H, Camarillo G, Johnston A, Peterson J, Sparks R, Handley M, Schooler E. *SIP: Session Initiation Protocol*, RFC 3261, 2002.
4. Shen C, Nahum EM, Schulzrinne H, Wright CP. The impact of TLS on SIP server performance: measurement and modeling. *IEEE/ACM Transactions on Networking* 2012; **20**: 217–1230.
5. Mishra G, Dharmaraja S, Kar S. Reducing session establishment delay using timed out packets in SIP signaling network. *International Journal of Communication Systems* 29 JUN 2014 2014. http://onlinelibrary.wiley.com/doi/10.1002/dac.2824/abstract
6. Uzunalioglu H, Houck DJ, Wang YT. Call admission control for voice over IP. *International Journal of Communication Systems* 2006; **19**:363–380.
7. Molina M, Quittek J, Brunner M, Melia T. Scalable and efficient QoS support for SIP-signalled voice calls. *International Journal of Communication Systems* 2006; **19**: 407–424.
8. Ccofano G, De Cicco L, Mascolo S. Local SIP overload control: controller design and optimization by extremum seeking. presented at the 2013 IEEE 52nd Annual Conference on Decision and Control (CDC), Firenze, 2013.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

9. Ggaidamaka YV, Abaev PO, Samouylov KE, Shorgin SYa. Design and software architecture of SIP server for overload control simulation, presented at the 27th european conference on modelling and simulation, 2014.
10. Schulzrinne H, Shen C, Nahum E, "Session initiation protocol (SIP) server overload control: design and evaluation," presented at the iptcomm, 2008.
11. Yaghmaee MH, Montazerolghaem AR. "SIP overload control testbed: design, building and evaluation," *International Journal of Ambient Systems and Applications* 2013; 17–26.
12. Yaghmaee MH, Montazerolghaem AR. "The overload reduction in SIP servers through exact regulation of the retransmission timer of the invite message " *Journal of Computer and Communications* 2013; 7–16.
13. Tang S, Zhang L, Cai Z. "Efficient and flexible password authenticated key agreement for voice over internet protocol session initiation protocol using smart card," *International Journal of Communication Systems* 2013.
14. Ma J, Qi J, Tian Y, "Cryptanalysis of smart-card-based password authenticated key agreement protocol for session initiation protocol of Zhang et al.," *International Journal of Communication Systems* 2014.
15. MS Farash, "An improved password-based authentication scheme for session initiation protocol using smart cards without verification table," *International Journal of Communication Systems* 2014.
16. Leong C-W, Yang S-R. "A conformance test tool for next generation network applications and systems," *International Journal of Communication Systems* 2010; **23**: 708–731.
17. Qi Q, Liao J, Li T, Cao Y, Zhu X, Wang J. "An optimized QoS scheme for IMS-NEMO in heterogeneous networks," *International Journal of Communication Systems* 2011; **25**: 185–204.
18. Chen Y-S, Chiu K-L, Hwang R-H. Seamless session mobility scheme in heterogeneous wireless networks," *International Journal of Communication Systems* 2011; **24**: 789–809.
19. Hsu C-S, Chen Y-S, Cheng C-H. "Network mobility protocol for vehicular ad hoc networks," *International Journal of Communication Systems* 2013.
20. Shen C, Schulzrinne HG, Nahum E. "Application layer feedback-based SIP server overload control," Technical reports2008.
21. V Hilt and I Widjaja, "Controlling overload in networks of SIP servers," in *Network Protocols*, *2008. ICNP 2008. IEEE International Conference on* 2008; 83–93.
22. S Charles and S Henning, "On TCP-based SIP server overload control," in *Principles, Systems and Applications of IP Telecommunications* (ed.) Munich, Germany: ACM, 2010; 71–83.
23. Nahum EM, Tracey J, Wright CP. Evaluating SIP server performance. *SIGM ETRICS Performance Evaluation Revew* 2007; **35**: 349–350.
24. Stephan W, Michael S, Sebastian K, Stefan W, Measurement of the SIP parsing performance in the SIP express router, in *Proceedings of the 13th open European summer school and IFIP TC6.6 conference on Dependable and adaptable networks and services*, (ed.) Enschede, The Netherlands: Springer-Verlag, 2007, 103–110.
25. Cortes M. On SIP performance. *Bell Labs Technical Journal* 2004; **9**: 155.
26. Montagna S, Pignolo M. Performance evaluation of load control techniques in SIP signaling servers, in *Systems*, *2008. ICONS 08. Third International Conference on*, 2008: 51–56.
27. Lindholm H, "A control plane benchmark for telecommunications signalling applications," presented at the LinuxConf Europe. Cambridge, England, 2007.
28. Schulzrinne HG, Narayanan S, *SIPstone: Benchmarking SIP Server Performance*(ed.). New York: Department of Computer Science, Columbia University, 2002.
29. Yun-Jung C, Kuochen W, Rong-Hong J, Chien C, Chia-Yuan H. Efficient failover and load balancing for dependable SIP proxy servers. In *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on*, 2008; 1153–1158.
30. Harchol-Balter M, Schroeder B. Evaluation of task assignment policies for supercomputing servers: the case for load unbalancing and fairness. *Cluster Computing* 2004; **7**: 151–161.
31. Crovella ME, Harchol-Baltera M, Murtac CD. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing* 1999; **59**: 204–228.
32. Arun I, Jim C, Daniel D, Paul D. High-performance Web site design techniques. *IEEE Internet Computing* 2000; **4**: 17–26.
33. Gianfranco C, Alma R, Evgenia S. EQUILOAD: a load balancing policy for clustered web servers. *Performance Evaluation* 2001; **46**: 101–124.
34. H S Singh K, Failover, load sharing and server architecture in SIP telephony. *Computer Communications* 2007; **30**:927.
35. Hongbo J, Iyengar A, Nahum E, Segmuller W, Tantawi AN, Wright CP. Design, implementation, and performance of a load balancer for SIP server clusters. *Networking, IEEE/ACM Transactions on* 2012; **20**: 1190–1202.
36. Acharya A, Singh A, Mahadevan P, Shae Z-Y. SPLAT: a unified SIP services platform for VoIP applications. *International Journal of Communication Systems* 2006; **19**: 425–444.
37. Naghibzadeh M. (2nd edn). *Operating System: Concepts and Techniques*, iUniverse Publisher: New York, 2011.

Copyright © 2015 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

View publication stats