DESIGN A SCHEDULER FOR A SIP SIGNALING NETWORK

Reza Gaemi[1,*], Ahamd Reza Montazerolghaem[2], S.-Kazem Shekofteh[3]
[1]Department of Computer Engineering, Quchan Branch, Islamic Azad University, Quchan, Iran
[2,3]Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

ABSTRACT: *The increasing use of SIP in Next Generation Networks necessitates that SIP networks provide adequate control mechanisms to optimize transaction throughput and prevent congestion collapse during traffic overloads. SIP throughput can severely be degraded when an overload situation happens in the proxy servers due to several retransmissions from user agents. In this paper we try to prevent throughput reduction by properly distributing the loads over available proxy servers. The proposed scheme utilizes response time of the servers as the main decision factor. The algorithm is implemented in a real environment using Spirent and Asterisk servers as call generator and load balancer respectively. The results of comparing the proposed method with some well-known algorithms indicate considerable throughput improvement up to 15% with a Round-Robin algorithm.*

Keywords—Scheduler; Session Initiation Protocol; Asterisk.

1. INTRODUCTION

Nowadays, Voice over IP (VoIP) networks are widely used spanning different levels of users such as organizations, academic and home users, since the underlying network can be an ordinary IP network which is a best-effort one. One of the most important elements of these networks is the signalling protocol. SIP is the most efficient used signalling protocol[1], because it's text-based and also end to end, supports mobility, and is independent from transferred data type. Despite these advantages, in networks connecting millions of users it would not perform its functionality well. This protocol is responsible for 1) establishing 2) managing and 3) terminating a call session. More details can be found in RFC3261 [1]. A session can be one of the four types: voice, video, text, or a combination of these. In protocol stack, SIP is an application protocol and SIP messages can be carried by both TCP and UDP. In these networks there are three components: User Agent (UA), proxy server, and other messages. The overview of the order of messages which are transferred between a user agent and a proxy server is represented in Figure 1. arrows show the message communications between network hops and after a call is established, media stream which are shown in black arrows will be transferred end to end. In Overload situation, INVITE messages are dropped and not processed because its queue is full. This case happens when the proxy does not have sufficient processing resources. A 503 Service Unavailable response message is then sent from proxy back to the user agent and the user agent will start to retransmit its `INVITE` message immediately. This trend will lead to congest not only the proxy server but also the whole SIP network.

Two categories of solutions to overcome the problem of load balancing are 1) overload control and 2) message distribution mechanisms. In the former, there are two types of decisions to make to prevent from overload situation, local: each proxy will choose its strategy about overload independently, and distributed: proxy servers cooperate about the decision. In the latter, there exists a third party (so called a balancer) entity that is responsible for well distribution of incoming messages among proxy servers.

In this paper we propose a method for message distribution mechanism. The balancer is in charge of scheduling incoming messages to be transmitted to a proxy server. The most important profit of the proposed load balancer is its message scheduler component which decides the best destination proxy server based on the history of the response times.

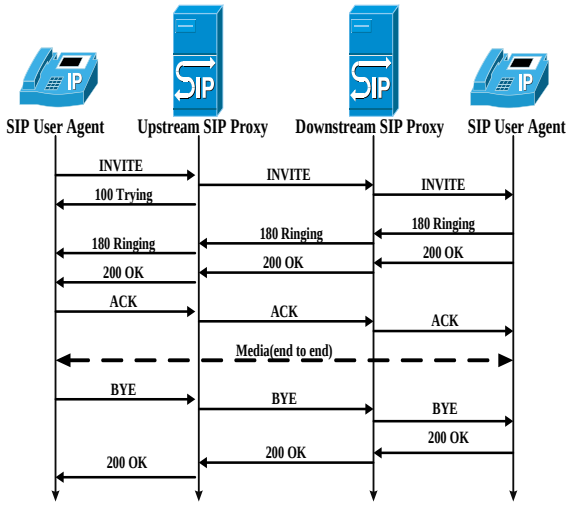

Fig. 1. Establishing a session using SIP protocol

The remainder of this paper is structured as follows. Section 2 describes related work. Section 3 provides some background on Load Scheduling and overload control. Section 4 describes the Proposed Algorithm. Section 5 explains the experimental testbed used for our experiments.

2. RELATED WORK

In [2] a load balancer is represented for three algorithms based on counting the number of transactions or sessions processed on a server: 1) counting the sessions, 2) counting the transactions, and 3) counting the weighted transactions named Transaction Least-Work-Left (TLWL). The best algorithm among these three algorithms is TLWL that is used in Section IV as a competitor for comparing our proposed algorithm with. Some of the existing web server redundancy techniques are involved to present a load sharing algorithm [3]. A similar problem resides in the field of balancing HTTP requests [4]. The effects of a round-robin DNS on scalability of NCSA's web site is described in [5].

The idea behind [6] is based on intercepting the prerequisite name resolution process in a typical client-server application within the IP network. A weighted hashing random algorithm that supports dialog in the SIP protocol to distribute messages

is presented in [7]. Cheng et al. [8] 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 SIP proxy servers gets down, hence it can balance proxy servers' load and achieve fast failover.

Some load balancing techniques for real web sites – especially high accessed ones – are described in [9] and [10]. There are some other techniques applied at client-side for assigning requests to servers which are presented in [11] and [12]. Ciardo et al. used request size to a web server for load balancing of clustered web servers in [13].

The need to solve this load distribution problem in other related domains has been considered and Balter and Schroeder proposed Least-Work-Left (LWL) and Join-Shortest-Queue (JSQ) respectively to be applied for task assignment to servers [14; 15].

3. LOAD SCHEDULING SCHEME

The proposed load scheduling algorithm is described in this section. The overall architecture of the algorithm is illustrated in Figure 2. 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 between various solutions to this problem. Our attention aims at minimizing the average response time of the SIP servers.

The first role of the load scheduling is classifying the input request messages. There are two classes of request messages: INVITE and Non-INVITE. In the proposed algorithm we focused on INVITE messages and leave Non-INVITE messages unscheduled. The main reasons for this approach are: 1) An INVITE message is a starting message to establish a session, therefore it consumes more computation times than a Non-INVITE message. 2) When a session is established between a client and a SIP server the subsequent messages must be sent to the same server that the corresponding INVITE message was sent.

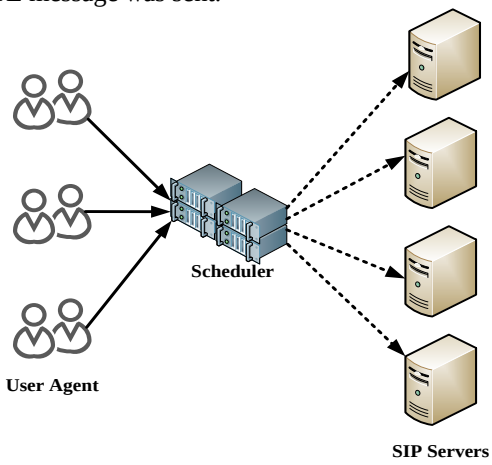

Fig. 2. The overal architecture of the algorithm.

The second role takes care of response messages. The load scheduler plays only the role of a relay in this step, since the destinations of these messages are known. 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 previous request messages sent to each server. Response time for each request is the time period between sending the INVITE message to the selection of the server and reception of corresponding 200 OK message. In the *Selection* phase, the scheduler selects the server which has minimum average response time to forward the input INVITE message.

4. THE PROPOSED ALGORITHM

The major novelty of the current algorithm is making use of average response time factor in order to increase overall throughput. To this end, the load scheduler uses a separate response time window for each server. Each window contains the history of response times of a server over the time. Since the farthest values of response time for a server are less important than the recent values, the size of the window is restricted to a fixed length, S_f.

When an incoming message receives to the load scheduler, the load scheduler first classifies it to INVITE or Non-INVITE. Non-INVITE messages are sent to the Forward or Drop module, directly. INVITE messages are queued in order to select the best server to which they will be forwarded. The load scheduler calculates the average response time value for each window and selects the server with minimum average value. The Call ID list contains pairs of $< server_i, session_j >$. A $< server_i, session_j >$ pair indicates that $server_i$ was chosen for processing $session_j$. When the $server_k$ finishes processing $session_l$, it sends a response back to the load scheduler, and the load scheduler plays its second role, as a relay, inserts the pair $< server_k, session_l >$ to the Call ID list and updates the window corresponding to that server.

a) Fixed-Sized Window Average Response Time (FAR) Algorithm:

The first load scheduler is equipped with a fixed-size window and the average response time is calculated over its slots. We chose 5, 10, 15 and 20 for S_f and evaluated the results in Section IV.

b) Fixed-Sized Window Weighted Average Response Time (FWAR) Algorithm:

The most important limitation for *FAR* method is that all of the slots have identical effect on average response time. Since recent response time values indicate the current processing power of the server, we used a weighted average response time (WART) i.e. recent response time values have larger weights than old ones and WART is calculated using Equation (1).

$$\bar{r} = \frac{\sum_{i=1}^{S_f} w_i . r_i}{\sum_{i=1}^{S_f} w_i} \qquad (1$$

$\bar{r}$ indicates average response time, w_i is the weight of the r_i and r_i represents the response time value placed at i^{th} window slot. In our experiments, since the farthest values of response time for a server are less important than the recent ones, we used uniform weights which mean that the weight of the most recent response time is S_f and for the least recent one is 1, Equation (2).

$$w_{S_f} = S_f \ , w_{S_f - 1} = S_f - 1 \ , \dots \ , w_1 = 1 \qquad (2)$$

5. PERFORMANCE EVALUATION

a) Specifications of Environment

In this section we are to describe the details of the implementation of the proposed method and network configurations. Figure 3 illustrates the details of network configuration used to perform the experiments. Spirent Abacus 5000 device is a powerful call generator which is capable of generating 10000 calls per second [16]. It supports different types of load distributions such as Poisson and Trapezoid and acts as user agents. We used Poisson distribution since it has a more normal behaviour than Trapezoid. The load profile configuration shown in this figure represents the timing parameters of calls. This device can act as both a caller and a callee.

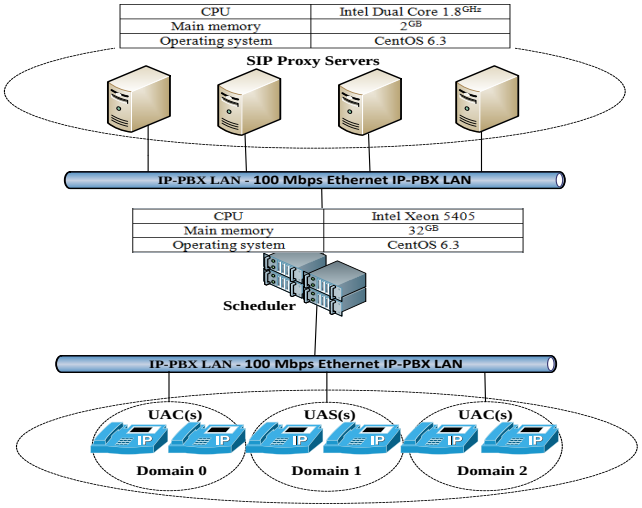

Fig. 3. The test-bed

In order to evaluate the performance of the proposed method in a condition very close to real world, we used commercial Asterisk SIP proxy servers [17, 18]. It can report comprehensive statistics about the status of the server during the experiments. In the following we used reports of both Spirent and Asterisk servers for evaluating the proposed method. The specifications of the load scheduler server and proxy servers are presented in Figure 3.

b) Performance Metrics

- Average Response Time: The average value of response time over various call rates (call per second).
- Throughput: The number of successful sessions established per time.
- Throughput & Average Response Time (heterogeneous back ends): In this sections, functionality of load balancing algorithms is inspected in the case that servers have disparate processing power and capabilities. In most of configurations, the expectation that all of the servers have similar processing power is unrealistic. They are heterogeneous in general. In this experiment, our load balancer will dispatch the requests to different servers.
- Retransmission rate: Users whose requests have remained unanswered, proceeds to resend their messages.

In the first set of experiments, we compared the results of average response time of three proposed methods with the best proposed algorithm – TLWL-1.75 –[2] versus common Round Robin method over 8 servers. Different volume of loads are generated and sent to the load scheduler starting from 10 cps – low load – to 3000 cps – heavy load. Round robin method is the worst one because it does not pay any attention to the current load on the servers and selects a server unconsciously. TLWL-1.75 dispatches the INVITE message to the server with lowest work. The work of a server is defined as the number of INVITE and BYE transactions currently processed on that server considering a weight of 1 and 0.75 for INVITE and BYE transactions, respectively. Figure 4 presents that average response time is decreased in the proposed method employing FWAR method since it uses WART, and WART is a better reflector of the current status of the server than the number of active transactions. WART can be considered as a cumulative function of response time and expressed in terms of response time value. This function is monotonically increasing in response time value. Therefore reducing response time value can directly affect WART over time. Since FWAR the load scheduler chooses the server with minimum WART, further values of WART are not very larger than current ones.

The second set of experiments cover the system throughput. Recall from previous discussion on average response time, one can express that the more the average response time for a server is, the longer the proxy queue will be. A server with less average response time has a shorter proxy queue length so it is able to process more incoming messages leading to higher throughput. This discussion is shown in Figure 5.

In many deployments, it is not realistic to expect that all nodes of a cluster have the same server capacity. Some servers may be more powerful than others, or may be running background tasks that limit the CPU resources that can be devoted to SIP. Maximum processing power of the first server is about 300 cps and for second and third servers are 150 cps and 75 cps respectively. Ideally, the proposed algorithm is expected to have a rate of 1.75 times of the first server, i.e., 525 cps in this heterogeneous environment.

The throughputs and average response time of four load balancing algorithms are shown in Figures 6, 7 and 8. As it is obvious in these figures, FWAR exhibits maximum throughput 486 cps that is very close to optimal rate.

These results expose that dynamic algorithm FWAR adapt to heterogeneous environments much better than other ones since they observe response times from servers continuously and try to balance them. Because the first server responses to requests two times faster than the second one, the ratio of the calls allocated to it, are also about two times more than the other one and four times than the third one. It should be noted that this is happened while the load balancer have no knowledge about this differences in processing powers of servers.

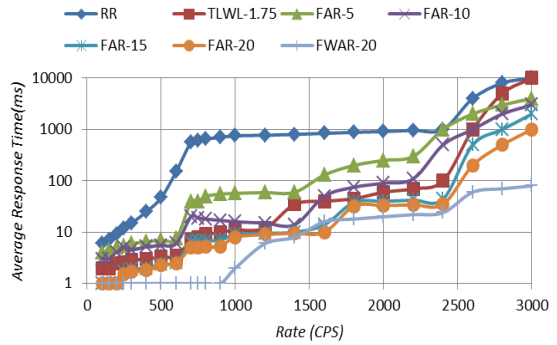

Fig. 4. Evaluating Average Response Time of the proposed method

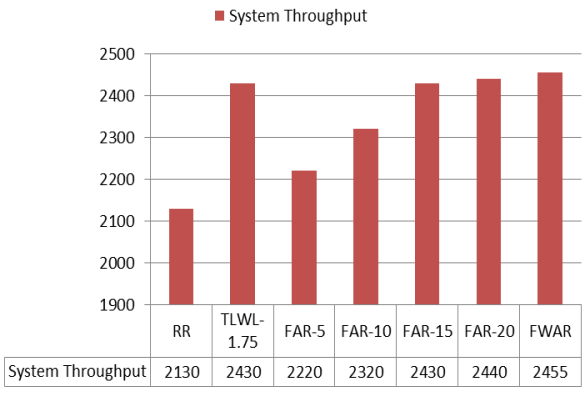

Fig. 5. Throughput

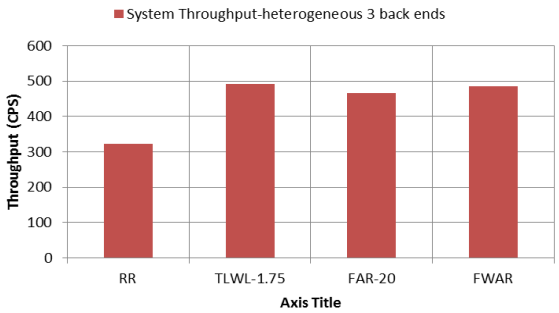

Fig. 6. Throughput (heterogeneous back ends)

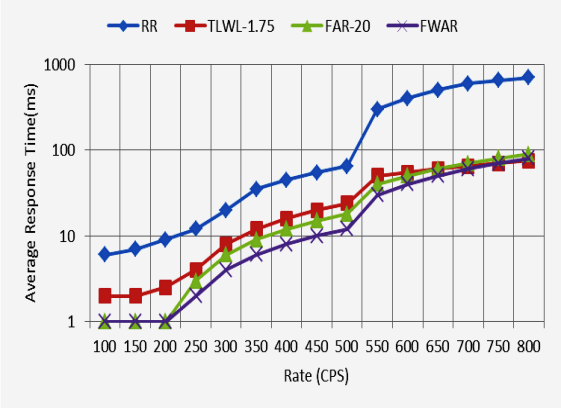

Fig. 7. Average Response Time (heterogeneous back ends)

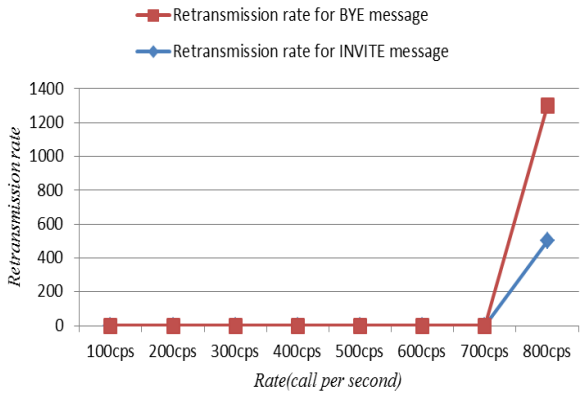

Fig. 8. Summary of concluded results in presence of FWAR mechanism

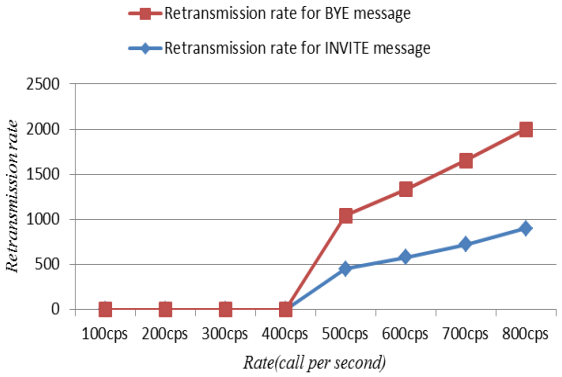

Fig. 9. Summary of concluded results in presence of RR(Round Robin) mechanism

Figures 9 and 10 illustrates retransmission rate for INVITE and BYE requests from user side, individually. As expected, no request is resend before received call rate reaches proxy's capacity. But upon reaching received call rate to proxy's capacity, resend rate increases abruptly and intensifies the load imposed to proxy considerably. As it is expected, when we use FWAR mechanism in SIP load balancer, retransmission rates of messages decrease considerably compared to the Round Robin mechanism. Overload leads to loss of OK packages related to the passed calls. So the proxy is required to resend INVITE requests related to lost packages. In this case, increase of retransmission rate makes proxy spend much of its time on resending requests related to ongoing calls and therefore throughput rate of proxy falls considerably.

6. CONCLUSION AND FURTHER WORKS

In this paper we examined the problem of load balancing in SIP network of VoIP connections. The problem is stated as selecting a server for processing the incoming message in order to prevent from the overload situations. Few similar works in this area and other related areas such as HTTP servers tried to solve this problem by counting the number of messages, transactions, etc. The most important drawback to these methods is using an inappropriate overload detection factor. In the proposed load scheduler each server has a corresponding window in the load scheduler. The content of

each window is the history of response time of the server over time. The slots are weighted monotonically in decreasing order i.e. the most recent response time value has the largest weight for calculating average response time. This strategy in conjunction with using average response time as the detection factor is the key advantage of the proposed method.

Evaluation of implementation of the proposed method on real SIP servers showed 1-2% performance enhancement in average response time related to best similar works and 15% compared to round robin algorithm.

We are working on using an infinite-length window and better method of weighting to this window slots. The efficiency of the extended method is being proved mathematically.

ACKNOWLEDGMENT

We are grateful to Islamic Azad University, Quchan branch authorities, for their useful collaboration.

REFERENCES

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "*SIP: Session initiation protocol"*, 2002.

[2] H. Jiang, A. Iyengar, E. Nahum, W. Segmuller, A.N. Tantawi, and CharlesP.Wright, "*Design, Implementation, and Performance of a Load Balancer for SIP Server Clusters*". IEEE/ACM Transactions on Networking 20 (2012) 1190-1202.

[3] K. Singh, and H. Schulzrinne, "*Failover, load sharing and server architecture in SIP telephony*". Computer Communications 30 (2007) 927–942.

[4] V. Cardellini, E. Casalicchio, M. Colajanni, and P.S. Yu, "*The state of the art in locally distributed Web-server systems*". Comput. Surveys 34 (2002) 263–311.

[5] D. Dias, W. Kish, R. Mukherjee, and R. Tewari. "*A scalable and highly available Web server"*. in *IEEE Compcon*. 1996.

[6] J.-S. Leu, H.-C. Hsieh, Y.-C. Chen, and Y.-P. Chi, "*Design and Implementation of a Low Cost DNS-based Load Balancing Solution for the SIP-based VoIP Service*". Proc. IEEE Asia-Pacific Services Computing Conference2008

[7] C. Shen, H. Schulzrinne, and E. Nahum, "Application Layer Feedback-based SIP Server Overload Control", 2008.

[8] Yun-JungCheng, KuochenWang, Rong-HongJan, ChienChen, and Chia-YuanHuang, "*Efficient Failover and Load Balancing for Dependable SIP Proxy Servers*". Proc. IEEE Symposium on Computers and Communications, Marrakech, 6-9 July 2008 pp. 1153-1158

[9] J. Challenger, P. Dantzig, and A. Iyengar. "A scalable and highly available system for serving dynamic data at frequently accessed Web sites". in ACM/IEEE Conf. Supercomput. 1998.

[10] A. Iyengar, J. Challenger, D. Dias, and P. Dantzig, "*High-performance Web site design techniques*". IEEE Internet Comput. 4 (2000) 17–26.

[11] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar. "A novel server selection technique for improving the response time of a replicated service". in IEEE INFOCOM. 1998.

[12] D. Mosedale, W. Foss, and R. McCool, "*Lessons learned administering Netscape's Internet site*". IEEE Internet Comput. 1 (1997) 28-35.

[13] G. Ciardo, A. Riska, and E. Smirni, "*EQUILOAD: A load balancing policy for clustered Web servers*". Perform. Eval. 46 (2001) 101–124.

[14] M. Harchol-Balter, M. Crovella, and C.D. Murta, "*On choosing a task assignment policy for a distributed server system*". J. Parallel Distrib. Comput. 59 (1999) 204–228.

[15] B. Schroeder, and M. Harchol-Balter, "Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness". Cluster Comput. 7 (2004) 151–161.

[16] www.spirent.com, "Spirent: The New generation of Ethernet testing".

[17] www.asterisk.org.

[18] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M.Handley, E. Schooler, SIP: Session Initiation Protocol, RFC 3261, December, 2002.