Hindawi
Security and Communication Networks
Volume 2017, Article ID 7821487, 14 pages
https://doi.org/10.1155/2017/7821487

Research Article

A Secured Load Mitigation and Distribution Scheme for Securing SIP Server

Vennila Ganesan and Manikandan MSK

Department of Electronic and Communication Engineering, Thiagarajar College of Engineering, Madurai, Tamilnadu, India

Correspondence should be addressed to Vennila Ganesan; vennilatg@tce.edu

Received 26 June 2016; Revised 2 February 2017; Accepted 19 February 2017; Published 15 March 2017

Academic Editor: Barbara Masucci

Managing the performance of the Session Initiation Protocol (SIP) server under heavy load conditions is a critical task in a Voice over Internet Protocol (VoIP) network. In this paper, a two-tier model is proposed for the security, load mitigation, and distribution issues of the SIP server. In the first tier, the proposed handler segregates and drops the malicious traffic. The second tier provides a uniform load of distribution, using the least session termination time (LSTT) algorithm. Besides, the mean session termination time is minimized by reducing the waiting time of the SIP messages. Efficiency of the LSTT algorithm is evaluated through the experimental test bed by considering with and without a handler. The experimental results establish that the proposed two-tier model improves the throughput and the CPU utilization. It also reduces the response time and error rate while preserving the quality of multimedia session delivery. This two-tier model provides robust security, dynamic load distribution, appropriate server selection, and session synchronization.

1. Introduction

The enormous growth of VoIP plays an active role in the IP-based business applications that introduce numerous real-time media services. Due to the lower cost and greater flexibility, customers are using various VoIP services such as voice calls, instance message, and video conference. In these services, VoIP adopts a SIP as a signaling protocol to create, manage, and terminate a multimedia session. The Real-time Transport Protocol (RTP) transmits media streams through a User Datagram Protocol (UDP) [1]. Figure 1 shows an example for SIP-based media conversation. An SIP transaction is carried out by the transmission of text-based request and response messages. The SIP transaction consists of two phases, namely, session establishment and session termination phases. In the session establishment phase, an *INVITE* message is followed by the corresponding *ACK* message whereas, in the session termination phase, a *BYE* message is followed by the corresponding *ACK* response message. The SIP server plays two vital roles; one is the utilization of location server to identify the location of the callee and the other is responsible for message routing. The SIP request messages are traversed via several proxies that are present either on the same or on different VoIP domains. Similarly, the related responses are forwarded to the same proxy servers in the reverse order.

The SIP-based IP telephony experiences a bottleneck problem because the signaling part deals with a huge amount of messages and the media part processes media streams. Therefore, both signaling and the media part directly influence the scalability of the VoIP network. In addition, the literature survey provides the causes of the SIP overload [2–6]. First, in order to provide reliable transmission, requests are retransmitted over the UDP for a specific period of time [2, 3]. Then, the SIP messages are transmitted among the clients via several proxy servers and adjacent nodes in the same or another network. Afterwards, the SIP messages are used as real-time session messages; consequently they are highly sensitive [4]. Finally, as voice calls are transmitted over the unsecured Internet, an attacker can easily inject attack packets as normal packets [5]. As a result, the server increases the unsuccessful call completion rate, reduces the

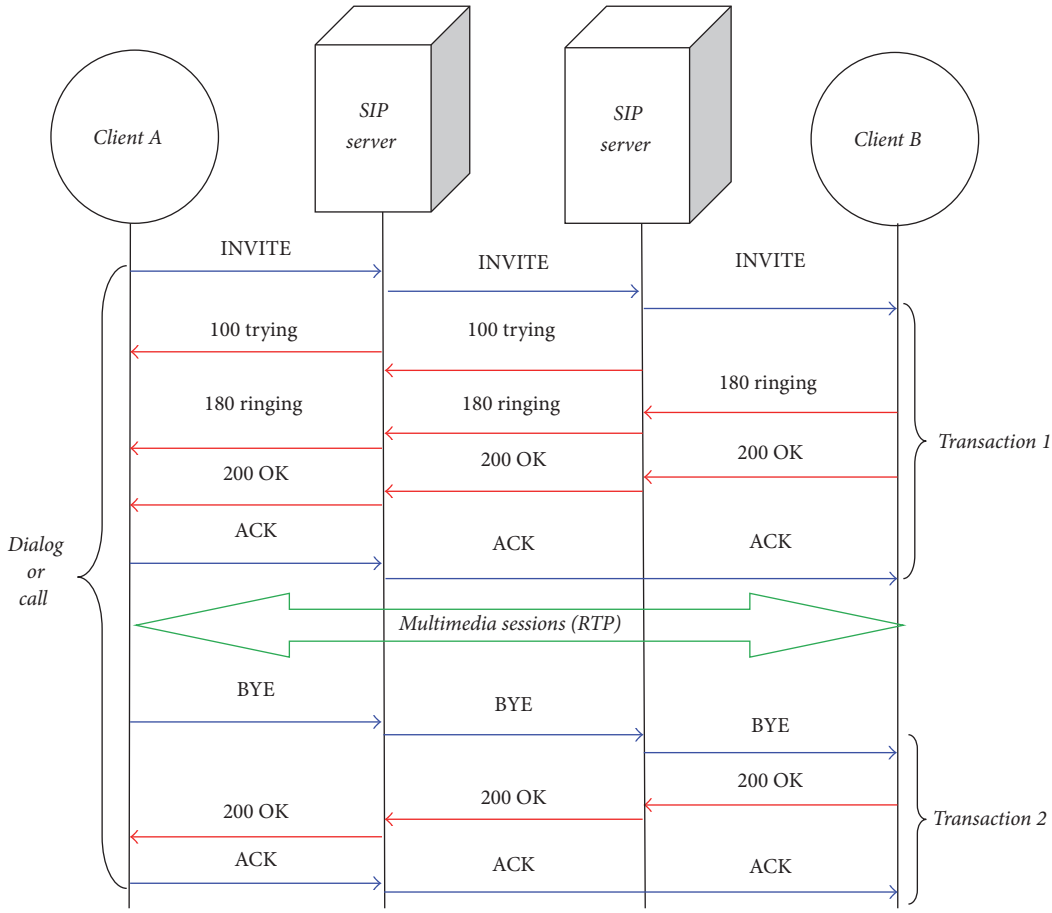

Figure 1: The SIP message flow.

throughput, and increases the call setup delay and spends most of the time to reject the request [6].

The main objective of the proposed work is to remove the overload state by detecting and eliminating the unwanted messages. The work also deals with the load distribution algorithm that provides a uniform load distribution among the servers. This work also maximizes CPU utilization, throughput while minimizing the response time, and execution time of the server. To attain these objectives, a handler and LSTT algorithm is introduced and it provides a uniform load distribution by selecting suitable server based on the least session termination time.

2. Related Works

The key question when supporting end-to-end media stream security is which layer should provide media security, either the network layer or some higher layer. The currently available foremost alternatives for the TCP-based data communication are Internet Protocol Security (IPSec) [7] (network layer) or Transport Layer Security (TLS) (transport/application layer) [8]. The standard SIP authentication protocol is the HTTP Digest Authentication which uses a trusted shared secret key to perform a cryptographic hash function. The experimental results of Salsano et al. [9] demonstrated that even the HTTP Digest Authentication is causing a considerable overhead on the SIP protocol. In the real-time UDP traffic, the main alternatives are Secure Real-time Transport Protocol (SRTP) [10] (transport/application) and IPSec. Chen et al. [11] designed a key exchange protocol which secures media stream and moderates the impact of Spam over IP Telephony (SPIT) using mutual authentication. Yoon et al. [12] established that a few SIP authentication schemes are insecure against the attacks such as offline password guessing attacks, Denning-Sacco attacks, and stolen-verifier attacks.

In general, the load balancing algorithms are classified into static and dynamic. In the static methods, the load is assigned during compilation time according to the prior decision of knowledge. The main drawback of these methods is that the load balancing decision is previously known and hence static methods are less efficient than the dynamic algorithms. Conversely, the load is allocated dynamically based on the dynamic algorithm implemented in the server and offers a good performance. Numerous research works are carried out to control the SIP overload dynamically by

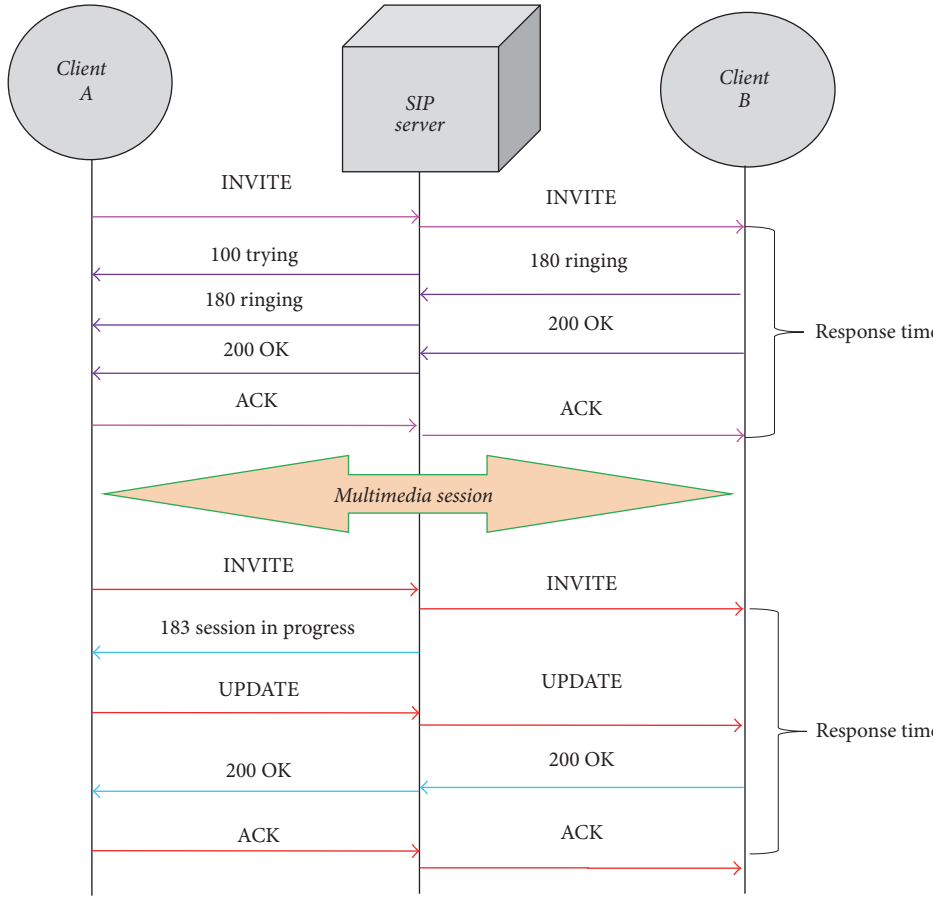

Figure 2: Existing system response time measurement.

considering various parameters such as feedback [4, 13–17], call rejection [18–21], session aware [22, 23], response time [24, 25], and priority [26, 27].

In the feedback-based overload control schemes [4, 13–17], the overloaded server computes constraints on its generated load according to current load and distributes such constraints to its immediate neighborhood which is placed before the overload server by a feedback mechanism. Based on this constraint, the neighborhood server decides whether the traffic has to be forwarded to the overload server or not. The limitation of these works is that the additional overhead problem is added to the overload server when the load constraints are computed and distributed. In the call rejection schemes [18–21], the overloaded server uses a local overload control algorithm to generate error responses (503 Service Unavailable) under a heavy load condition. Depending on the error response rate, incoming traffic is restricted. Though these algorithms result in better solution, in real time it is restricted by a service provider. In addition, high volume of rejection can lead a way for the SIP network failures. Therefore, this approach can be applied efficiently in lightly overloaded servers only.

The session aware method [22, 23] comprises three algorithms, namely, least call, least transaction, and weighted least transaction. In this method, the load is distributed according to the least number of active calls in the server. It is not a feasible solution because each call consumes different time duration. The next option is the least transaction algorithm which chooses the server, based on the least number of active transactions that the server currently handles. The main limitation of this algorithm is that the *INVITE* request takes a longer period than *BYE* request. The third algorithm is the weighted least transaction that assigns a weight of 0.75 for *BYE* and 1 for *INVITE*. In that algorithm, every server has to maintain the session and transaction states for individual user. The weight assignment method is useful only when the server in the cluster does not have the same capacity.

In [24, 25], the average response time of each server is maintained in a separate window and the load is distributed to the server which has the least response time. The overall response time of the session establishment phase is importantly considered in this research. This scheme becomes ineffective when the queuing delay and packet loss rates are increased. Moreover, the SIP transmits a maximum of 8 retransmissions for each original request. Hence, the overall response time of the session establishment will be increased. Besides, signaling messages are interrupted in order to enhance the performance of the SIP network during the media conversation period [28–30]. For example, the signaling messages *INVITE*, *183 Session in progress*, *UPDATE*, *200 OK,* and *ACK* are interrupted during the media conversation to improve the SIP security as shown in Figure 2.

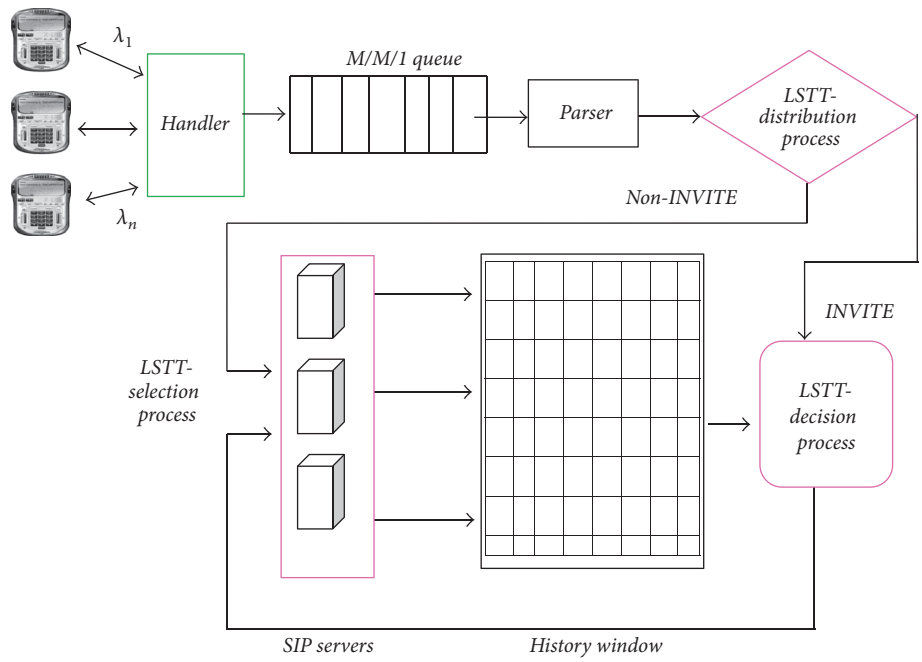

Figure 3: Two-tier system architecture.

As a result, computing response time between *INVITE* and *ACK* is not a feasible solution. The existing weight-based scheduler [22, 23] leads to starvation, since it does not provide a way for a high-weighted request and so the retransmission occurs over a certain period. Therefore, an additional overload problem is applied to the load balancer. A few research works [26, 27] proposed a method which reduces the retransmission rate instead of reducing original sending rate. This solution offers a lesser amount of blocking calls and more revenue for carriers. The performance of the SIP server is increased in the priority queue [26], in which *INVITE* requests are assigned to low priority and all other requests are given high priority. The authors in [27] use the error message *(503 Service Unavailable)* that stops the retransmission request. But this method creates an additional overhead problem to the server. Researchers pay more attention on security threats introduced by the load balancer [31–33]. Kim et al. [31] investigated the performance of SSL protocol for providing secure service in a cluster-based application server and proposed a backend forwarding method for improving server performance through better load balance. The authors [32] implemented a Distributed Denial of Service (DDoS) detection scheme in the load balancer. Here, the dispatcher module of Kamailio server acts as load balancer. This scheme detects low-rate and multiattribute DDoS attacks. However, there are no enough practical results. The authors in [33] implemented authentication layer for load balancing architecture. Thus, only the authenticated users can send jobs and no overload occurs within the cloud platform and the load balancer pays minimum resource wastage. In the previous work [34], the honeywall is a major contribution for reducing the load towards the load balancer. The SIP server is configured to make a decision to select the server with the least number of active calls and BYE_ACK method. In this work, the SIP message arrival rate, delay, and deadline are used to compute the load in the queue.

3. Proposed Two-Tier Model

In many cases, the load balancer alone spends more processing time for malicious packets, retransmission, and signaling traffic and latter these messages could be rejected [5, 35]. Therefore, the processing time together with the load distribution delay gets increased. In order to overcome these problems, in this work, a handler is designed that drops malicious and unwanted traffics before serving the load balancer. The operation of the handler is independent of that of the load balancer as shown in Figure 3. The handler is the first entity implementing prior to the load balancer which receives all incoming and outgoing SIP traffic. The handler provides anomaly traffic detection, prevention, and load mitigation. As a result, the work task of the load balancer gets decreased and the availability of server resource gets increased.

The second tier LSTT algorithm defines three processes, namely, distribution, decision, and selection. The main aspect of the LSTT algorithm is that it decides to select an appropriate server for new and existing SIP messages. Based on the call-ID and call sequence (Cseq) in the message header, a distribution process investigates whether the incoming message is new or existing one. If the incoming message is an existing one, then the previous corresponding request handling server is identified and the load can be distributed. Else, a decision process is applied to new message which selects a suitable server. Here, the LSTT algorithm dynamically calculates the session termination time by measuring the

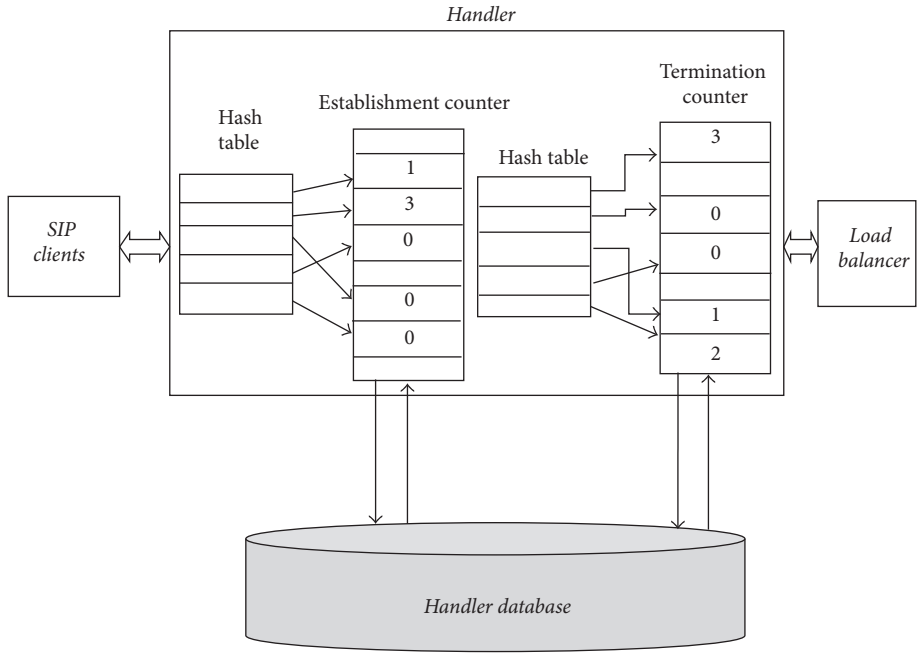

Figure 4: Signaling traffic detection.

timestamp between a *BYE* and its corresponding *ACK*. Then, the selection process chooses an appropriate server which has the least session termination time in the history window. Each server maintains an accurate state and the load balancer updated periodically. At that time, every server will update the message delay time and cumulative session termination time at the predetermined time interval.

3.1. Handler. The SIP has many open choice control messages and it can easily be mounted by using a SIP traffic generator. In the SIP, a few messages *(REGISTER, INVITE, and BYE)* and its message fields like *URI* and *Call-ID* are only protected. All other signaling messages like *100 Trying*, *180 Ringing*, *200 OK*, and *ACK* and message fields like *From*, *To*,and SDP are undefended [36]. Consequently, a signaling traffic increases from 20 to 40% [5, 36–38] in the VoIP network. This signaling traffic increases the processing time of the load balancer and so the call dropping ratio at the victim side is increased. Therefore, an effective signaling traffic detection and prevention scheme is proposed in this work. The signaling traffic and its changes are computed by

$$At_i = At_{i-1} + \Delta t, \tag{1}$$

where At_i is signaling traffic of the ith user and Δt is increasing signaling traffic rate at T period.

The handler is deployed at the edge router of the innocent host and checks the SIP control messages that enter and leave through the edge router. The handler operates at three stages, namely, call rate, control, and drop. The call rate stage uses modified hash tables for two counters (session establishment and session termination) as shown in Figure 4. Each SIP message (*INVITE-ACK* and *BYE-ACK*) is hashed with ⟨source-IP, destination-IP, From-URI, To-URI⟩ and recorded in the handler database. For any valid SIP session there is a unique one-to-one mapping among *INVITE-ACK* and *BYE-ACK*. The abnormal three-way handshake is clearly shown by these two counters.

Initially, the Bloom filter was designed by Bloom [39] and modified to protect from flooding attacks [40, 41]. The Bloom filter has vector V of m bits and set to 0 in the initial condition. Let the element $a \in A$; the bit positions $h_1(a), h_2(a), \ldots, h_k(a)$ in V are set to 1. If any one of the bits is 0, then $a \notin A$. An original Bloom filter is unable to handle the paired three-way handshake messages. Therefore, 0's and 1's are replaced with countable integers in the Bloom filter. For each incoming *INVITE* and *BYE* message, the handler increments the corresponding counter by 1 and decrements by 1 for a pair of *200 OK* and *ACK*. Each paired *INVITE-ACK* and *BYE-ACK* counter remains 0 and the rest of the bit positions in the counter keep a nonzero value. The overflow bit position in the counter shows an unwanted traffic that intends to affect the SIP server. The handler notifies the affected position of the counters by

$$T_{\text{value}} = \mu_{\text{mean}}\left[H_n\right] + \eta \cdot \sigma_{\text{SD}}\left(H_n\right), \tag{2}$$

where H_n is number of signaling messages' *(INVITE, BYE)* rate under normal traffic, μ_{mean} is mean of the signaling message, and σ_{SD} is standard deviation of the signaling messages.

The threshold value can be set according to the number of users and their generated signaling traffic. In this work, a maximum of $\alpha = 0.08$ is to be set for the maximum of 8000 calls per second (cps). The next step is to identity the offended SIP messages and eradicate them to provide a secure environment. Particularly, each detection cycle that deviates from the interarrival time of the message is considered to be

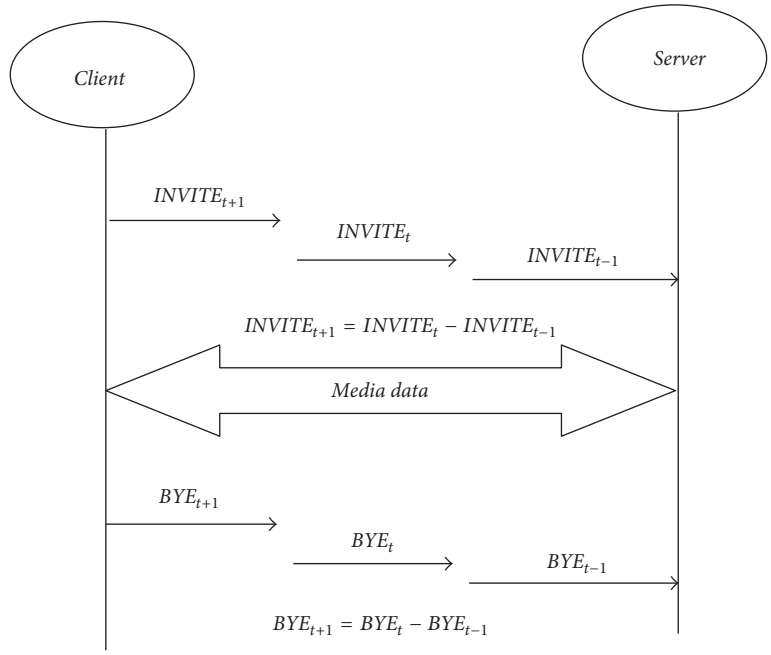

Figure 5: SIP message distribution.

an unwanted message. For every call, the signaling messages occur with a minimum round trip time as stated in

$$T_{INVITE} < T_{100\,\text{Trying}} < T_{180\,\text{Ringing}} < T_{200\,OK} < T_{ACK} < T_{BYE} < T_{\text{after BYE 200 OK}} < T_{\text{after } BYE\ ACK}. \quad (3)$$

First, different types of signaling traffic that contain various forms of the SIP messages from legitimate users are observed. Then, the distribution of time intervals between the first and previous message of the same user is characterized as shown in Figure 5.

The signaling traffic of each message of the same type is identified by

$$\Delta\text{SIP message}_{ik} = \text{SIP messages}_{it} - \text{SIP messages}_{i(t-1)}, \quad (4)$$

where SIP message$_{it}$ is ith observation of k type signaling message and SIP message$_{i(t-1)}$ is $(i-1)$th observation of k type signaling message.

For instance, signaling traffic (*REGISTER*, *INVITE*, and *BYE*) and malformed packets *(fake BYE, fake BUSY, fake BYE drop)* are identified by

$$\text{SIP message}_{i(INVITE)} = INVITE_t - INVITE_{t-1}, \quad (5)$$

where $INVITE_t$ is first *INVITE* packet at time t and $INVITE_{t-1}$ is next *INVITE* packet at time $t-1$ of the same user.

Then, the average time interarrival of each SIP message is computed by

$$\overline{\Delta\text{Sum}_{ik}} = \frac{\sum_{a=1}^{n_i} \Delta\text{SIP message}_{ik}}{n_i}, \quad (6)$$

where n_i is number of messages from ith observation.

The SIP message interarrival time dispersion is computed by

$$\sigma\left(\Delta\text{SIP message}_{ik}\right) = \sqrt{\frac{\sum_{a=1}^{n_i}\left(\Delta\text{SIP message}_{ik} - \overline{\Delta\text{Sum}_{ik}}\right)^2}{n_i - 1}}. \quad (7)$$

At last, the control stage compares the offended messages with threshold and removes it by drop rule. Thus, only controlled and needed requests are transmitted to the load balancer and all other requests are eliminated. Finally, our architecture is added with the second stage called LSTT load balancer.

3.2. LSTT Algorithm for Load Distribution. The main objective of the second tier architecture is that none of the servers should be in an idle and heavily loaded state. The second tier measures the least session termination time which is the round trip time (RTT) between *BYE* and *ACK* as shown in Figure 6.

This work uses a load balancer based on M/M/1 queuing model that consists of an infinite number of clients in which each server maintains a queue with Poisson arrival rate λ and service time μ which is exponentially distributed with mean $1/\mu$. To identify the status of the server, it is considered that $N(t)$ is the number of clients currently in the system, which would follow a Markov chain with states $\{0, 1\}$. When the client arrives, the server state is 0 or 1 and it is said to be in an idle or *BUSY* state, respectively. The given random process $N(t)$ is a birth-death process and hence the rates are

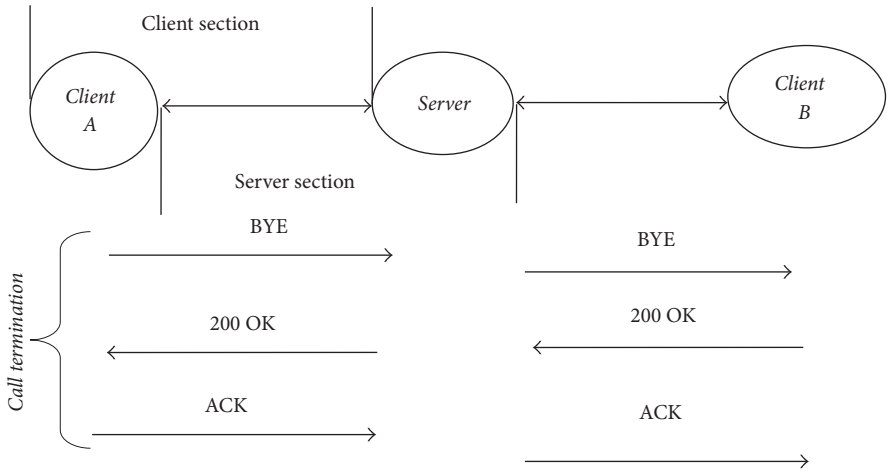

Figure 6: Cumulative session termination.

$$\lambda_k = \lambda$$
$$\mu_k = \mu \quad (8)$$
$$k = 0, 1, 2, 3, \ldots$$

Let $\rho = \lambda/\mu$ be the average number of utilizations or loads in a server. If $\rho > 1$, the SIP server is heavily loaded; otherwise the server is lightly loaded. The distribution of signaling load for different types of K calls can be characterized by empirical measurement as given in

$$P(k_n) = \frac{\text{Total session received}}{\text{Total session}}. \quad (9)$$

Let $T_s = \{BYE, 200\,OK, ACK\}$ be the set of messages used to terminate the session. Each call is assumed to be a probabilistic model and let p_{ij} be the probability of process i to process j for K types of calls, where $i, j \in T_S$. The probability of mean arrival rate λ_i^K for K types of calls is identified by

$$\lambda_i^K = \sum_j \lambda_j^K p_{ji}^K \quad i, j \in T_s, \quad (10)$$

where λ_i^K is mean arrival rate of ith messages for K types of call, λ_j^K is mean arrival rate of jth messages for K types of call, and p_{ji}^K is probability of ith and jth message for K types of call.

The server utilization ρ_i for K calls is computed by

$$\rho_i = \sum_K \frac{\lambda_i^K}{\mu_i^K}. \quad (11)$$

The probability of messages entering the queue for services is equal to the probability that all servers are *BUSY*. Thus, the mean for each message (L_i) in the set T_s is computed by

$$L_i = \frac{\rho_i}{1 - \rho_i}. \quad (12)$$

The total session completion time for terminating a call may be defined as the addition of message entering time and waiting time which is multiplied with service time. The session termination time (T) is computed by

$$T = \frac{1}{E(s)} \left(L_{BYE} + L_{200\,\text{OK}} + L_{ACK}\right), \quad (13)$$

where $E(s)$ is mean service time $1/\mu_i$, $i \varepsilon T_s$.

The distributed messages wait in the queue for a certain period of time before processing and so this waiting time affects the response time-based scheduler. The proposed work reduces the message waiting time; thus, the mean response time gets minimized.

3.2.1. Minimizing Mean Response Time. Figure 7 shows the load balancer which receives a continuous stream of messages to be processed in the message set as M where $M = 1$, 2, and 3.

These messages require an amount of processing time that varies with several orders of magnitude (size, cost, etc.). From Figure 7, the clients $(c_1, c_2, c_3, \ldots, c_9)$ and their generated messages are shaded by different colors. The client1 call is terminated after serving the *INVITE* messages of c_2 and c_3. The call termination varies according to the cost of the message, which means that the *INVITE* message takes a longer time to complete a call than the *BYE* message [23]. Hence, any response time-based scheduling algorithm has to consider delay which decides the efficiency of the algorithm. It is assumed that the server has a capacity S and the sum of the message size in the set must be less than or equal to S. Utilized Parameters shows the notations used in this work.

It is learnt that the service discipline at each server is FIFO in which scheduling and transmission times are negligible. The arriving SIP message i spends a particular time in the system which is denoted as waiting time W_i and computed by

$$W_i = C_i - \lambda_i \quad \text{for message } i = 1, 2, 3, \ldots, n. \quad (14)$$

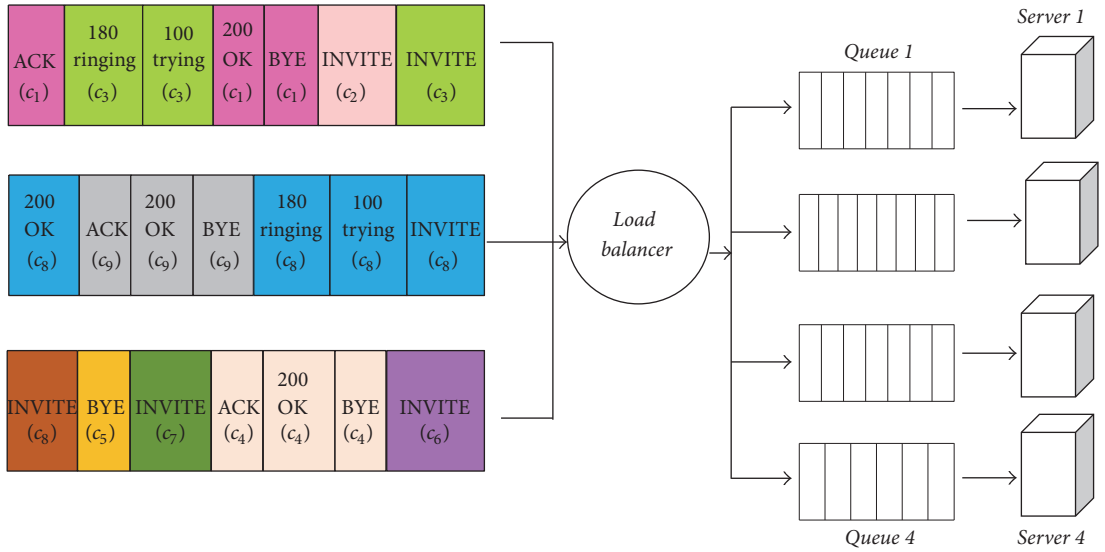

Figure 7: Load distribution in the queue model.

The weighted total response time of the message W_i is given in

$$W_i = \sum_{i=1}^{n} \mathrm{WT}_i \left(C_i - \lambda_i\right). \tag{15}$$

The waiting time weights have a different load index such that

$$\mathrm{WT} = \left[\mathrm{WT}_1, \mathrm{WT}_2, \ldots, \mathrm{WT}_n\right] = \sum_{i=1}^{n} \mathrm{WT}_i = 1. \tag{16}$$

To minimize the waiting time of each message in the message set, the delay factor with known factors of arrival time and a deadline of each SIP message is considered. For example, *INVITE* message waits until $T_1 = 500$ ms and its corresponding response message waits until $T_2 = 64 * T_1$ [1]. Therefore, the proposed system measures how long a message is delayed compared to its deadline. The delay time is measured according to the waiting time of the message in the queue. Thus, the minimization of the weighted mean waiting time is equivalent to the minimization of the weighted total waiting time. To minimize the total waiting time, the existing work takes a maximum flow time and max stretch (delay) [42, 43]. These metrics are desirable for the real-time application, but the maximum flow metrics increases the sequence of message arrival rate which results in starvation. The maximum delay time d_i instead of max flow is applied so as to avoid starvation that is computed by

$$d_i = \max_i \left\{ \frac{C_i - \lambda_i}{D_i - \lambda_i} \right\}, \tag{17}$$

where $S_i = D_i - \lambda_i$.

Equation (17) is rewritten as

$$d_i = \max \left\{ \frac{W_i}{S_i} \right\}. \tag{18}$$

The delay time of a message is its waiting time divided by its processing time as given in

$$d_i = \max \left\{ \frac{W_i}{P_i} \right\}. \tag{19}$$

To reduce the mean response time, decision process assigns the optimal delay time to each message and sorts the message sequences according to the nondecreasing delay time. Then, the load balancer selects the SIP message which has a maximum delay and distributes it to an appropriate server. The LSTT algorithm computes the mean session termination time of each server by

$$\mathrm{ST}_k = \mathrm{CT}_k + \mathrm{NRT}_k \tag{20}$$

$$\sum_{k=1}^{n} \min \left(\mathrm{ST}_k\right), \tag{21}$$

where ST_k is kth server session termination time, CT_k is kth server cumulative termination time, $\mathrm{CT}_k = \sum_{j=1}^{n}$ Session termination time$_j$, where $j = 1, 2, 3, \ldots, n$, Session termination$_j$ is jth session termination time, and NRT_k is kth server newly received session termination time.

Finally, the selection process identifies the server ST_k in the history window by using (21). It can be noted that the messages are the same size with same delay time that are executed in FIFO fashion. The load balancer distributes the messages according to the least cost of the session termination time among the servers.

4. Experimental Setup

Figure 8 depicts the experimental test bed which consists of the various experimental components as listed in Table 1.

4.1. Call Generation. For generating a huge amount of call traffic, SIP$_p$ is used as an open source tool coded with XML for the implementation. When a new call is generated by *INVITE* message, the SIP proxy forwards it to the

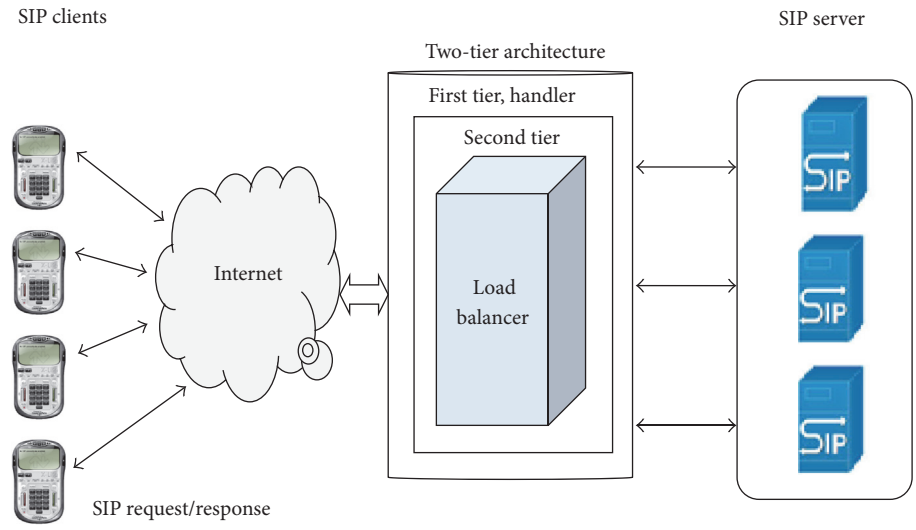

Figure 8: Experimental test bed.

Table 1: Experimental test bed components.

Experimental components	Functions
SIP_p	Client generates SIP signaling traffic
OPenSER	Proxy server handles thousands of VoIP calls/sec
NMON	Provides information about CPU, memory, and network
Wireshark	Network protocol analyzer

corresponding client and the appropriate response will be received. Once a call is initiated successfully, the media data is exchanged; then, the session termination occurs with standard *BYE* message and its corresponding *ACK* message. SIP_p has been configured in such a way that the load is generated continuously without any interruption. The maximum capacity of the proxy and its performance has been measured under heavy load conditions.

4.2. Hardware Implementation. The OpenSIP express version 1.1 *(OpenSER)* is an open source proxy server coded in C and it is considered to have high efficiency with minimal functionality. The database stores OpenSER configuration data and registers client information. An IBM blade server with 4 GB RAM and 100 GB ATA disk drives is used to carry out these experiments. In the study, it is observed that the CPU cycles spend more time in stateful proxy. However, a stateful proxy records the route for future use.

4.3. Load Balancer Design. It has also been observed that, without handler mechanism, a server drops the messages which results in retransmission. On the other hand, the SIP server sends *503 Service Unavailable* responses to avoid unwanted retransmissions to the corresponding clients [30]. This response scenario cannot provide a better performance owing to the relatively high processing cost. Initially, a set of experiments is conducted with handler prior to the load balancer and this combination can be used to check the test call sessions. The full session is captured and analyzed in the two-tier model to ensure the total number of sessions for a call from a client to a server and vice versa. The load balancer maintains an active table, to track the incoming load rate and the number of active sessions in the server. The timer is set for 1 sec to update the active table regularly. The proposed algorithm gives a pause time with a mean of 1 min and a variance of 30 sec.

5. Performance Comparison

The load which is generated in this experiment varies up to a maximum of 8000 calls. Then, the detection rate of handler, server throughput, response time, CPU utilization, and error rate are measured. The obtained results are compared with the existing algorithms (response time [25], transaction least work 1.75 [22]). In order to identify the performance of the two-tier model, two different implementations are executed. The first one is load balancer with handler and the other one is without handler. Each implementation is measured for an average of 10 min and every event is measured for 120 sec and 5 sec warm-up period. Every test runs around 10 to 20 times to validate the results and the average values of various parameters are analyzed.

5.1. Attack Packet Distribution. Figure 9 shows the observed SIP signaling attack packets before serving the handler. It can be observed that the fake and flooding traffics give significant changes in the VoIP network. For the duration of one week, 1821 flooding and 632 fake SIP messages are observed. In addition to that, 97 timeouts and 117 retransmission packets are also obtained. Therefore, a strong protection mechanism is required to drop the unwanted traffic.

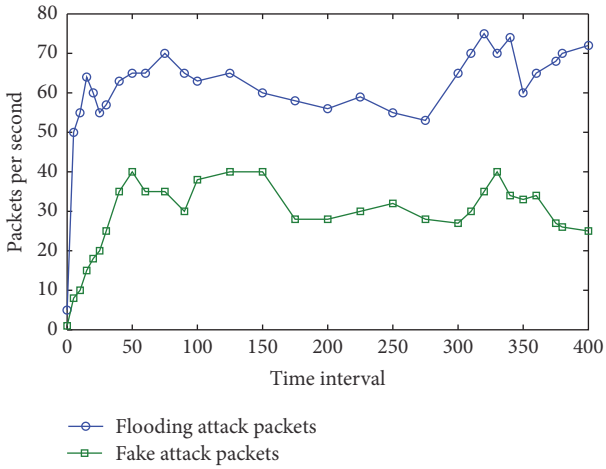

Figure 9: Signaling traffic behavior.

Table 2: Load versus response time.

Generated load	Response time (ms)	
	Response time based scheduler [25]	Proposed two-tier model
200	0.5	0.1
400	1.5	0.8
600	2	1.2
800	2.5	1.3
1000	2.8	1.41

5.2. Handler Detection Rates. The SIP_p generates a normal traffic and is mixed with attack packets from 100 to 400 packets per second (pps). The experiment is repeated for several runs up to 20 times. The proposed process on the handler is capable of identifying lower arrival rate of 50 pps. The duration of the detection rate varies from 5 to 10 sec for increasing the Poisson rate from 100 to 400. The detection rates are calculated from the relative proportions of the SIP attributes during the same period of time. It is found to have certain great fluctuations as shown in Figures 10(a) and 10(b). Existing Hellinger Distance [36] based detection methods require a maximum threshold value of $\alpha = 0.4$. The proposed handler utilizes only $\alpha = 0.08$ for flooding and fake message consumes a maximum of $\alpha = 0.05$.

5.3. False Alarm Rate: Fake Messages. Figure 11 shows the false alarm rate for fake signaling packets. The proposed handler is found to have a good effect on correctly identifying the fake signaling packets. As the call volume increases, the false alarm rate of misclassified fake signal packet reaches less than 0.2 for the proposed handler and 0.5 for the existing system.

5.4. Response Time. Tables 2, 3, and 4 depict the average response time of the proposed two-tier model and existing algorithm. A maximum of 3 servers are used in this experiment and the server capacity is varied from 300 to 500 cps.

Table 3: Arrival rate versus response time.

Arrival rate	Response time (ms)	
	Response time based scheduler	Proposed two-tier model
0.2	2.3	1
0.4	3.1	1.4
0.6	3.5	1.6
0.8	4.7	1.9

Table 4: Utilization versus mean response time.

Utilization	Response time (ms)	
	Response time based scheduler	Proposed two-tier model
0.2	2.4	0.9
0.4	1.8	0.5
0.6	1.2	0.4
0.8	0.9	0.3

Table 5: Load versus throughput.

Generated calls	Throughput (%)		
	TLWL 1.75 [22]	Response time based scheduler	Proposed two-tier model
1000	78.2%	78%	93.3%
2000	81%	82.66	95.7%
3000	86.6%	86.6%	99.45
4000	85%	87.5%	99.28%
5000	88.8%	86.32%	98.43
6000	91.6%	90.325%	99.37%
7000	67.32	71.32	81.32%
8000	61.25%	60.13%	74.84%

The average response time is also varied while increasing the load to the maximum of 1000 cps. The existing algorithm [25] takes a maximum of 2.8 ms whereas the LSTT maintains less than 1.41 ms for 1000 cps. Similarly, service rate of 1 ms is applied and λ and ρ are varied accordingly, for which a LSTT algorithm results in a maximum of 1.9 ms and 0.3 ms response time, respectively, as given in Tables 3 and 4.

5.5. Load Balancer Throughput. The throughput of existing algorithms [22, 25] is compared with the proposed algorithm as given in Tables 5 and 6. The throughput results are obtained for generating a load of 8000 cps without pause duration. From Table 5, it is observed that the LSTT achieves the highest throughput of 99%. However, the existing response time [25] attains only 90% and TLWL 1.75 [22] reaches a maximum of 91.6%.

The number of clients is increased up to 10 for a maximum of 3 servers. The obtained throughput results are furnished in

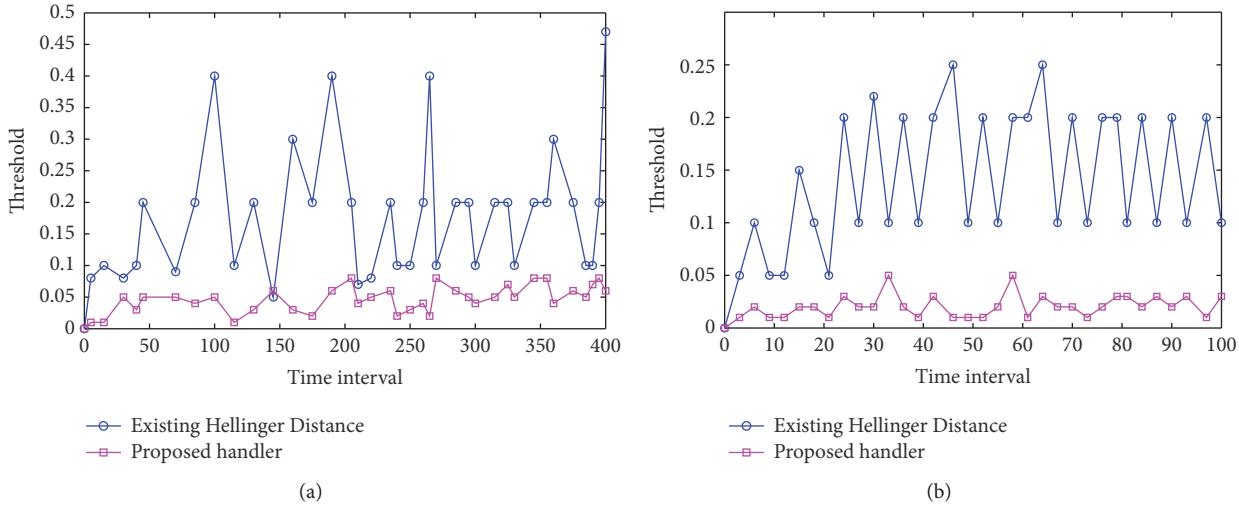

Figure 10: (a) Threshold value for flooding attack. (b) Threshold value for fake signal.

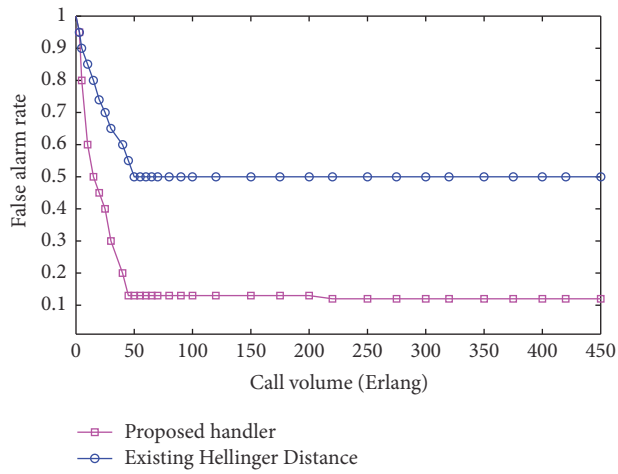

Figure 11: False alarm rate for fake messages.

Table 6: Number of clients versus throughput.

Number of clients	Throughput (Kbs)		
	TLWL 1.75	Response time based scheduler	Proposed two-tier model
2	13	15	18
4	30	33	38
6	41	44	50
8	45	52	72
10	50	55	78

Table 7: Execution time of the LSTT algorithm.

CPU utilization (%)	LSTT algorithm
10	0.034
20	0.033
30	0.033
40	0.0333
50	0.0334
60	0.0334
70	0.0336
80	0.0337
100	0.0339

Table 6 and it is obvious that the proposed algorithm works better in spite of increasing the client rate. Efficiency of the LSTT algorithm is verified by removing the first tier and it is observed that the LSTT linearly increases the throughput for a maximum of 6821 cps without a handler and 7420 cps with a handler as shown in Figure 12.

5.6. CPU Utilization. Table 7 summarizes the execution time of the LSTT algorithm and corresponding CPU utilization. The execution time for the estimation process of the LSTT algorithm is the time stamp between a *BYE* request from the client and its corresponding *ACK* from the server. The two-tier model utilizes lesser time to accomplish a SIP transaction because it spends a minimum amount of time to analyze the message whereas the rest of the algorithms monitor and analyze large amount of messages.

5.7. Error Rate. Figure 13 demonstrates the error rates of different load balancing algorithms. The error rate is the percentage of unsuccessful requests which do not get the appropriate servers since the load balancer spends more time in message analysis.

$$\text{Error rate} = \frac{\text{Number of requests forwarded}}{\text{Total number of requests}} \times 100. \quad (22)$$

From Figure 13, it is observed that the better load balancing algorithm increases the acceptance rate of load and

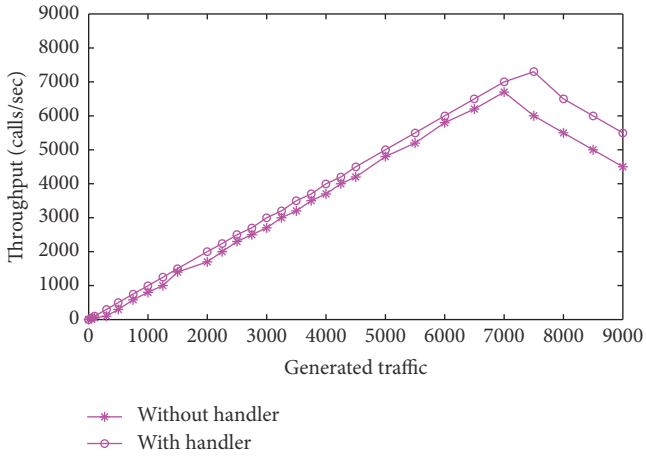

Figure 12: Throughput comparison of two-tier model.

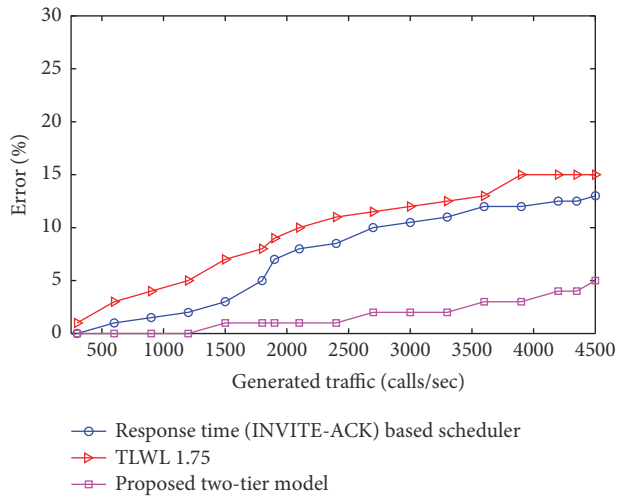

Figure 13: Error rate.

rejects the request that spends more time on consumption of resources. Furthermore, the arrival rate of excessive requests in the load balancer takes more processing time and these requests are not required for services. Spending more time to analyze these requests raises the error rates. However, dropping of these messages could result in a maximum throughput. The error rate result clearly brings out that there are a number of requests that are not effectively distributed due to timeout conditions. The proposed LSTT algorithm achieves 5% of error rate at offered load of 4500 cps and TLWL 1.75 generates 15% of error rate.

6. Conclusion

In this paper, a two-tier model was proposed to eliminate the malicious traffic in the first tier and provide uniform load distribution in the second tier. The proposed LSTT algorithm ensures that all inbound/outbound SIP messages are routed to an appropriate server which has the least session termination time at the moment. The implementation results are phenomenally impressive and the proposed model significantly improves the throughput and 99% of CPU utilization even when the offered load was a maximum of 8000 cps. At the same time, the system responds instantly to the variation of the generated load and achieves a better performance with a reduced error rate of 5%. Thus, it is concluded that the proposed two-tier system provides an enhanced performance with a guaranteed QoS.

Utilized Parameters

i: SIP messages $i = 1, 2, 3, \ldots, n$
λ: Arrival time
P: Processing time
W: Waiting time
D: Deadline
D: Delay
C: Message completion time
S: Slowdown or slack time
WT: Weight.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] S. H. Rosenberg, G. Camarillo, A. Johnston et al., "SIP: session initiation protocol," IETF-RFC 3261, 2002.

[2] C. Shen, E. Nahum, H. Schulzrinne, and C. P. Wright, "The impact of TLS on SIP server performance: measurement and modeling," *IEEE/ACM Transactions on Networking*, vol. 20, no. 4, pp. 1217–1230, 2012.

[3] J. Rosenberg, "Requirements for management of overload in the session initiation protocol," RFC 5390, IETF, 2008.

[4] V. Hilt, I. Widjaja, and B. Labs, "Controlling overload in networks of SIP servers," in *Proceedings of the 16th IEEE International Conference on Network Protocols (ICNP '08)*, pp. 83–93, October 2008.

[5] D. Sisalem, "SIP overload control: where are we today?" in *Proceedings of the Trustworthy Internet*, pp. 273–287, Springer, Milan, Italy, 2011.

[6] A. r. Montazerolghaem and M. H. Yaghmaee, "SIP Overload Control Testbed: Design, Building and Evaluation," *The International Journal of Ambient Systems and Applications*, vol. 1, no. 2, pp. 17–26, 2013.

[7] S. Kent and K. Seo, "Security architecture for the internet protocol," IETF RFC 4301.

[8] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.2," RFC 5246, IETF, 2008.

[9] S. Salsano, L. Veltri, and D. Papalilo, "SIP security issues: the SIP authentication procedure and its processing load," *IEEE Network*, vol. 16, no. 6, pp. 38–44, 2002.

[10] F. Palmieri and U. Fiore, "Providing true end-to-end security in converged voice over IP infrastructures," *Computers & Security*, vol. 28, no. 6, pp. 433–449, 2009.

[11] C.-Y. Chen, T.-Y. Wu, Y.-M. Huang, and H.-C. Chao, "An efficient end-to-end security mechanism for IP multimedia subsystem," *Computer Communications*, vol. 31, no. 18, pp. 4259–4268, 2008.

[12] E.-J. Yoon, K.-Y. Yoo, C. Kim, Y.-S. Hong, M. Jo, and H.-H. Chen, "A secure and efficient SIP authentication scheme for converged VoIP networks," *Computer Communications*, vol. 33, no. 14, pp. 1674–1681, 2010.

[13] C. Shen, H. Schulzrinne, and E. Nahum, "Session initiation protocol (SIP) server overload control: design and evaluation," in *Proceedings of Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, pp. 149–173, Springer, 2008.

[14] E. Noel and C. R. Johnson, "Novel overload controls for SIP networks," in *Proceedings of the 21st International Teletraffic Congress (ITC 21 '09)*, September 2009.

[15] Y. Wang, "SIP overload control: a backpressure-based approach," in *Proceedings of the ACM SIGCOMM Conference (SIGCOMM '10)*, pp. 399–400, New Delhi, India, August 2010.

[16] R. G. Garroppo, S. Giordano, S. Niccolini, and S. Spagna, "A prediction-based overload control algorithm for SIP servers," *IEEE Transactions on Network and Service Management*, vol. 8, no. 1, pp. 39–51, 2011.

[17] Y. Hong, C. Huang, and J. Yan, "Mitigating SIP overload using a control-theoretic approach," in *Proceedings of the 53rd IEEE Global Communications Conference (GLOBECOM '10)*, December 2010.

[18] A. Abdelal and W. Matragi, "Signal-based overload control for SIP servers," in *Proceedings of the 7th IEEE Consumer Communications and Networking Conference (CCNC '10)*, pp. 1–7, IEEE, Las Vegas, Nev, USA, January 2010.

[19] S. V. Azhari, M. Homayouni, H. Nemati, J. Enayatizadeh, and A. Akbari, "Overload control in SIP networks using no explicit feedback: a window based approach," *Computer Communications*, vol. 35, no. 12, pp. 1472–1483, 2012.

[20] J. Liao, J. Wang, T. Li, J. Wang, J. Wang, and X. Zhu, "A distributed end-to-end overload control mechanism for networks of SIP servers," *Computer Networks*, vol. 56, no. 12, pp. 2847–2868, 2012.

[21] J. Wang, J. Liao, T. Li, J. Wang, J. Wang, and Q. Qi, "Probe-based end-to-end overload control for networks of SIP servers," *Journal of Network and Computer Applications*, vol. 41, pp. 114–125, 2014.

[22] H. Jiang, A. Iyengar, E. Nahum, W. Segmuller, A. N. Tantawi, and C. P. Wright, "Design, implementation, and performance of a load balancer for SIP server clusters," *IEEE/ACM Transactions on Networking*, vol. 20, no. 4, pp. 1190–1202, 2012.

[23] H. Jiang, A. Iyengar, E. Nahum, W. Segmuller, A. N. Tantawi, and C. P. Wright, "Load balancing for SIP server clusters," in *Proceedings of the IEEE INFOCOM*, pp. 2286–2294, IEEE, Rio de Janeiro, Brazil, April 2009.

[24] A. Montazerolghaem, S.-K. Shekofteh, G. Khojaste, M. Naghibzadeh, and M.-H. Yaghmaee-M, "A novel load scheduling for session initiation protocol networks," in *Proceedings of the 4th International Conference on Computer and Knowledge Engineering (ICCKE '14)*, pp. 509–514, October 2014.

[25] A. Montazerolghaem, S.-K. Shekofteh, M. H. Yaghmaee, and M. Naghibzadeh, "A load scheduler for SIP proxy servers: design, implementation and evaluation of a history weighted window approach," *International Journal of Communication Systems*, vol. 30, no. 3, Article ID e2980, 2015.

[26] M. Ohta, "Overload protection in a SIP signaling network," in *Proceedings of the International Conference on Internet Surveillance and Protection (ICISP '06)*, August 2006.

[27] R. G. Garroppo, S. Giordano, S. Spagna, and S. Niccolini, "Queueing strategies for local overload control in SIP server," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '09)*, pp. 1–6, IEEE, Honolulu, Hawaii, USA, December 2009.

[28] C.-H. Wang and Y.-S. Liu, "A dependable privacy protection for end-to-end VoIP via Elliptic-Curve Diffie-Hellman and dynamic key changes," *Journal of Network and Computer Applications*, vol. 34, no. 5, pp. 1545–1556, 2011.

[29] M. Hasebe, J. Koshiko, Y. Suzuki, T. Yoshikawa, and P. Kyzivat, "Example call flows of race conditions in the Session Initiation Protocol (SIP)," RFC 5407, IETE Draft, 2008.

[30] G. Camarillo, C. Holmberg, and Y. Gao, "Re-INVITE and target-refresh request handling in the session initiation protocol (SIP)," RFC Editor, 2011.

[31] J.-H. Kim, G. S. Choi, and C. R. Das, "A load balancing scheme for cluster-based secure network servers," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER '05)*, September 2005.

[32] A. Akbar, S. M. Basha, and S. A. Sattar, "Leveraging the SIP load balancer to detect and mitigate DDos attacks," in *Proceedings of the 1st International Conference on Green Computing and Internet of Things (ICGCIoT '15)*, pp. 1204–1208, IEEE, Noida, India, October 2015.

[33] M. Belkhouraf, A. Kartit, H. Ouahmane, H. K. Idrissi, Z. Kartit, and M. El Marraki, "A secured load balancing architecture for cloud computing based on multiple clusters," in *Proceedings of the 4th International Conference on Cloud Computing Technologies and Applications (CloudTech '15)*, Marrakech, Morocco, June 2015.

[34] G. Vennila and M. S. K. Manikandan, "Two stage secure dynamic load balancing architecture for SIP server clusters," *Journal of Engineering Science and Technology Review*, vol. 7, no. 3, pp. 1–6, 2014.

[35] W. Mi, X. Qiu, and C. Zhang, "The analysis of security threats in structured P2P load balancing schemes," in *Proceedings of the International Conference on Cloud and Service Computing (CSC '11)*, pp. 296–301, December 2011.

[36] R. Zhang, X. Wang, X. Yang, and X. Jiang, "On the billing vulnerabilities of SIP-based VoIP systems," *Computer Networks*, vol. 54, no. 11, pp. 1837–1847, 2010.

[37] J. Tang, Y. Cheng, Y. Hao, and W. Song, "SIP flooding attack detection with a multi-dimensional sketch design," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 6, pp. 582–595, 2014.

[38] J. Stanek and L. Kencl, "SIPp-DD: SIP DDoS flood-attack simulation tool," in *Proceedings of the 20th International Conference on Computer Communications and Networks (ICCCN '11)*, August 2011.

[39] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[40] B.-H. Roh, J. W. Kim, K.-Y. Ryu, and J.-T. Ryu, "A whitelist-based countermeasure scheme using a Bloom filter against SIP flooding attacks," *Computers and Security*, vol. 37, pp. 46–61, 2013.

[41] D. Geneiatakis, N. Vrakas, and C. Lambrinoudakis, "Utilizing bloom filters for detecting flooding attacks against SIP based services," *Computers and Security*, vol. 28, no. 7, pp. 578–591, 2009.

[42] D. Xiaotie and Y. Zhang, "Minimizing mean response time in batch processing system," in *Proceedings of the 5th Annual International Computing and Combinatorics Conference (COCOON '99)*, pp. 231–240, Springer, Tokyo, Japan, July 1999.

[43] M. A. Bender, S. Muthukrishnan, and R. Rajaraman, "Improved algorithms for stretch scheduling," in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '02)*, pp. 762–771, Philadelphia, PA, USA, January 2002.