ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/301577956

Overload mitigation mechanism for VoIP networks: A transport layer approach based on resource management

Article *in* Transactions on Emerging Telecommunications Technologies · June 2016

DOI: 10.1002/ett.3038

CITATIONS
3

READS
4,583

4 authors:

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

Seyed Amin Hosseini Seno
Ferdowsi University Of Mashhad
117 PUBLICATIONS **853** CITATIONS
SEE PROFILE

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

Farzad Tashtarian
Alpen-Adria-Universität Klagenfurt, Austria
67 PUBLICATIONS **623** CITATIONS
SEE PROFILE

All content following this page was uploaded by Ahmadreza Montazerolghaem on 15 January 2018.

The user has requested enhancement of the downloaded file.

TRANSACTIONS ON EMERGING TELECOMMUNICATIONS TECHNOLOGIES
Trans. Emerging Tel. Tech. (2016)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/ett.3038

RESEARCH ARTICLE

Overload mitigation mechanism for VoIP networks: a transport layer approach based on resource management

Ahmadreza Montazerolghaem[1], Seyed Amin Hosseini Seno[1], Mohammad Hossein Yaghmaee[1] and Farzad Tashtarian[2]

[1] Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
[2] Department of Information Technology, Mashhad Branch, Islamic Azad University, Mashhad, Iran

ABSTRACT

Session initiation protocol (SIP) is an application layer signalling protocol for the set-up, management and termination of multimedia networks such as voice over IP standardised by Internet Engineering Task Force. This protocol has also been recognised by International Telecommunication Union as a main core of next-generation networks. When this network is under overload for reasons such as improper design, instantaneous swarm, components error and sudden reduction of processing capacity, its efficiency is considerably reduced. Overload occurs when the SIP proxy lacks sufficient CPU and memory resources to process all messages. Because the overload cannot be prevented completely, it is important to equip SIP proxies with an effective overload mitigation mechanism. In this paper, capabilities of transmission control protocol are used in transport layer to reduce SIP proxy overload through the proper allocation of proxy resources. Major activities in this field assume SIP on user datagram protocol, which practically does not result in optimal throughput. To evaluate this approach, an Asterisk open source proxy is used. Our implementation results in the real testbed indicate the efficiency improvement of Asterisk proxy under overload. Copyright © 2016 John Wiley & Sons, Ltd.

***Correspondence**
S. A. Hosseini Seno, Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran.
E-mail: hosseini@um.ac.ir

Received 10 September 2015; Revised 21 February 2016; Accepted 2 March 2016

1. INTRODUCTION

The extent and diversity of systems provided by IP networks have made various technologies integrate various types of access networks and convert them into next-generation networks. Session initiation protocol (SIP) with respect to facilities, such as being in text form, end-to-end connection, independence from type of transmitted data and supporting various forms of transmission, is an appropriate choice for signalling protocol in order to make a connection between two IP network users. SIP is the Internet Engineering Task Force protocol for voice over IP and other text and multimedia sessions, like instant messaging, video, online games and other services. These advantages have made SIP to be regarded as a signalling protocol in IP multimedia subsystem, a proposed signalling platform for next-generation networks by the 3rd Generation Partnership Project. With the 3G (3rd Generation) wireless technology adopted by more and more carriers, nearly, all cellular phones and other mobile devices will have to support SIP as their signalling protocol for multimedia sessions in the near future [1]. Despite having all these advantages, the SIP protocol lacks an appropriate mechanism for addressing overload.Overload can cause serious problems for SIP servers. The SIP overload occurs when this server does not have enough resources to process messages. The influence of the most important resource on overload is processing power of CPU and memory of SIP server [1]. It is a proven fact that the performance of SIP servers is largely degraded during overload periods because of the built-in message retransmission mechanism of SIP. The redundant retransmissions increase CPU and memory occupation and worsen the server overload.

1.1. Session initiation protocol

Figures 1 and 2 show the SIP messages procedure, which manages an SIP call session. Main elements of SIP network include: (1) user agents which establish communication as user agent client (UAC) and user agent server (UAS);

Copyright © 2016 John Wiley & Sons, Ltd.

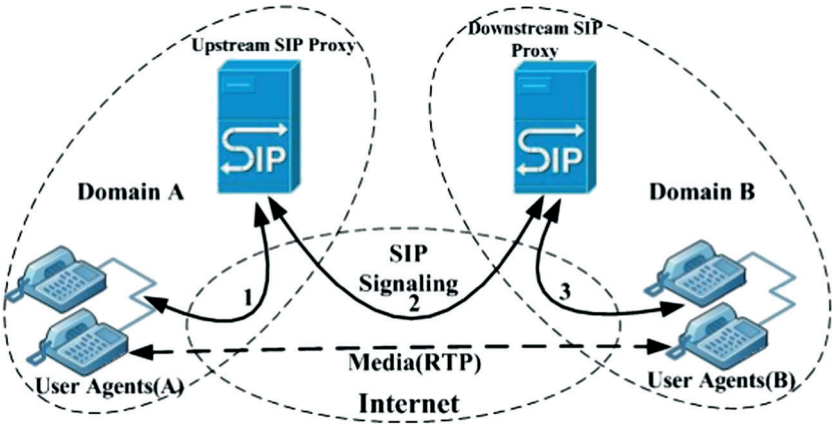


Figure 1. Elements of session initiation protocol network.

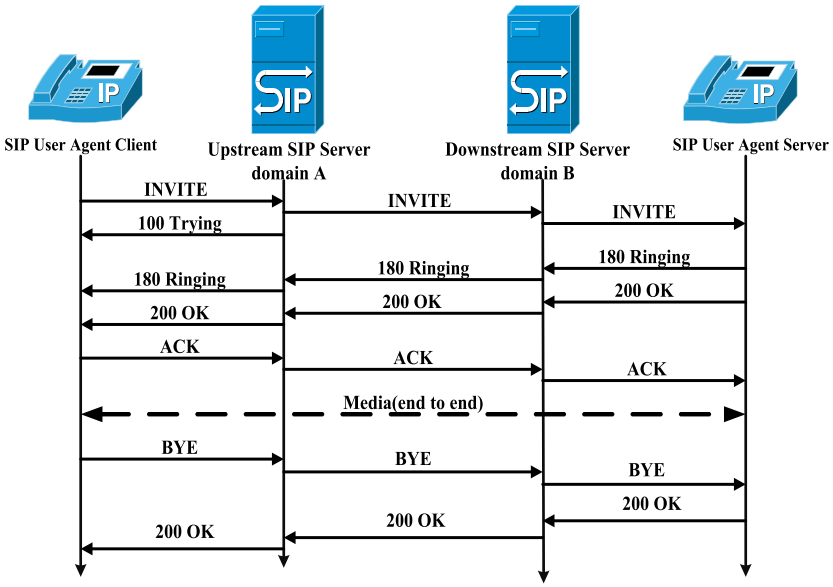


Figure 2. Session initiation protocol message flow.

(2) registration servers which are responsible for the registration and storage of users' information; and (3) proxy servers which are responsible for the routing of signalling messages among the user agents. The UAC sends an '`INVITE`' message through proxies inside the route to the UAS. Each proxy announces the acceptance of the request by sending '100 Trying' response message to the previous step so that the timer stops its retransmission. This action is performed according to the stateful configuration of proxies. When the UAS receives '`INVITE`' request, it sends '180 Ringing' response message to the UAC from the route which requests is sent. After the call acceptance, it sends a '200 Ok' response to the UAC. In the end, the UAC sends an 'ACK' response to the UAS. After 'ACK' is sent to the UAC, the session is initiated. Now, multimedia data are exchanged among the users by real-time transfer protocol without the interference of proxies. Finally, one of the users asks to stop communication by sending 'BYE' request. Another party ends the session by sending '200 Ok' message [2].

1.2. Overload problem in session initiation protocol proxies

Overload occurs when request entry rate into one of the SIP servers is higher than the capacity of its resources. These resources can include processing capacity of processor, memory, network bandwidth, input/output or disc sources which are most notably the processing power of CPU and memory of the SIP server. When the server is put under overload, its efficiency will be dropped so considerably that its useful throughput reaches zero [2, 3]. This condition is worse when user datagram protocol (UDP) is used as the transport layer protocol, because SIP uses several

retransmission timers to ensure communication. For instance, the response of messages is not transmitted in a timely manner; requests will be retransmitted; as a result, the server load will be intensified, because the redundant retransmissions and manipulation of the mentioned timers increase CPU and memory occupation and worsen the server overload. One of the key properties of SIP server overload is that the cost of rejecting a session is usually comparable with the cost of serving a session. Consequently, when an SIP server has to reject a large number of incoming sessions, it ends up spending all its processing cycles for rejection and causes its throughput to collapse [3].

At the time of overload, messages will be highly delayed or will overflow because of buffer filling. The messages, which have been highly delayed, will lead to missed call; this is because if the call is not established within 32 s, and the call will be ignored. Even if we assume that there is no retransmission, buffer overflow will lead to the loss of messages relating to a call which is being established and, finally, entail missed call.

Figure 3 shows the efficiency of the server under overload. This server is not equipped with any mechanism for confronting the overload. Horizontal axis shows the entry load rate, and vertical axis shows the server throughput. Both axes have been normalised on server capacity. As shown later, throughput will reach zero with a sudden drop after entry load is higher than processing load value.

As shown in Figure 4, at the time of overload in a server, many requests are retransmitted by its upstreams. Under such conditions, not only is load intensified on the server under overload but also it causes its upstreams to be put under overload. In this case, overload can be expanded to the entire network and disable the entire network.

Some factors such as different configurations of servers, size of receptor's memory and the used transport protocol

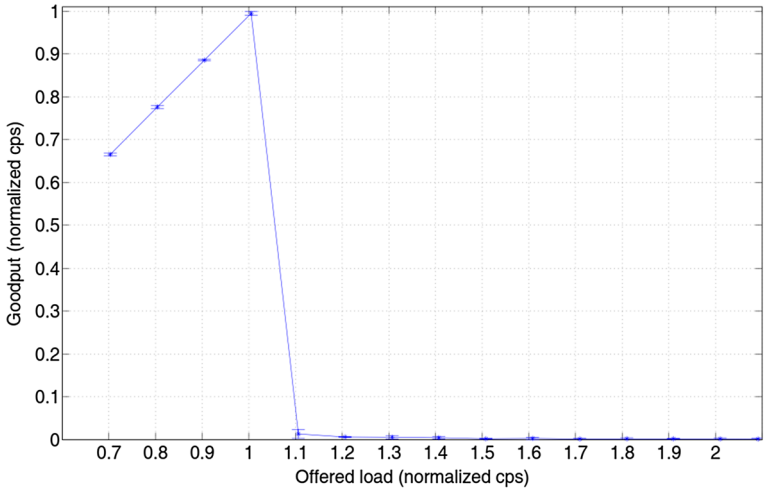


Figure 3. Session initiation protocol server's throughput under different loads [4].

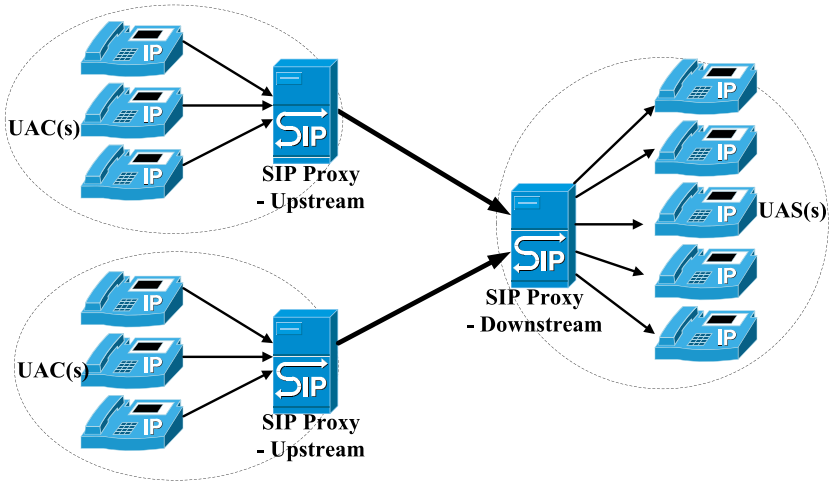


Figure 4. Server to server overload.

are effective for the efficiency of servers under ordinary load and overload conditions [3–6]. The efficiency will be reduced considerably without an overload mitigation method [6].

1.3. Session initiation protocol messages transport medium

Considering the independence of SIP protocol from transport protocols (Figure 5), SIP messages can be transported on all types of transport protocols such as transmission control protocol (TCP), UDP, stream control transmission protocol (SCTP) and transport layer security (TLS). Therefore, the retransmission mechanisms which have been embedded in SIP are able to compensate for the loss of the packets if necessary. These mechanisms are applied when SIP messages are transported on the unreliable transport layer protocol such as UDP. On the contrary, there is no need for the use of the mechanisms which provide reliability in the application layer of SIP by transporting packets on the reliable transport protocols.

The advantage of reliable transport layer protocols such as TCP is the provision of rapid retransmission mechanisms, error correction, congestion control, flow control and packet segmentation in the transport layer. TCP rapid retransmission which retransmits the uncertified packet before the expiry of retransmission timer reduces delay in call only when the loss of packets in the network is possible [7].

Overload mitigation can be implemented in each of the data link, network, transport and application layers [18]. Because SIP is an application layer protocol, it is one of the most frequent overload mitigation methods in this layer. But, the method provided in this paper will utilise TCP procedure to mitigate overload and act in transport layer. Yet, virtually, many of the existing SIP overload control works have been focused on SIP-over-UDP despite the fact that TCP is increasingly seen as the more viable choice of SIP transport [7]. We will show that our method in a real bed and on Asterisk proxy will improve the efficiency. We also study and show the effect of selecting transport layer protocol on call quality parameters in Asterisk proxy. Despite the public belief that the reason for common use of UDP against TCP is its low processing overhead, low efficiency in the use of TCP results shows the proxy configuration manner.

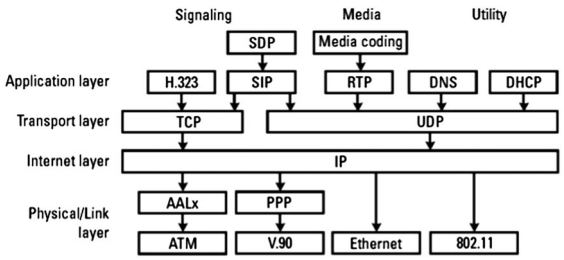


Figure 5. Protocol stack.

In the rest of this paper, we mention a review of the literature in this field in Section 2. Then, we introduce implementation and its components in Section 3. Also in this section, we study effect of transport layer protocol on efficiency of Asterisk proxy. In Section 4, we mention the proposed mechanism, and finally, evaluation of our approach is presented in Section 6.

2. RELATED WORKS

Figure 6 shows the general categories of methods dealing with overload:

2.1. Load balancing methods

One approach for overload prevention is the distribution of new input traffic over SIP servers based on their accessible capacity using a 'load balancer' (Figure 7), which reduces the probability of overload occurrence in a given server. However, the performance of these mechanisms depends on load balancer's efficiency, because the whole signalling traffic of the SIP network passes through it. Failover and

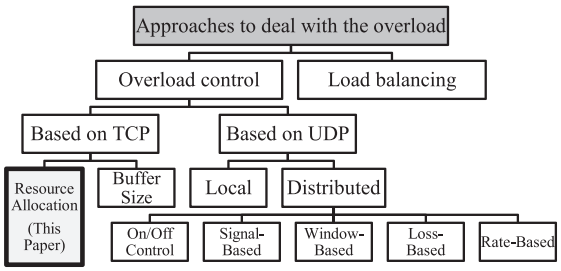


Figure 6. A classification of session initiation protocol overload mitigation mechanisms.

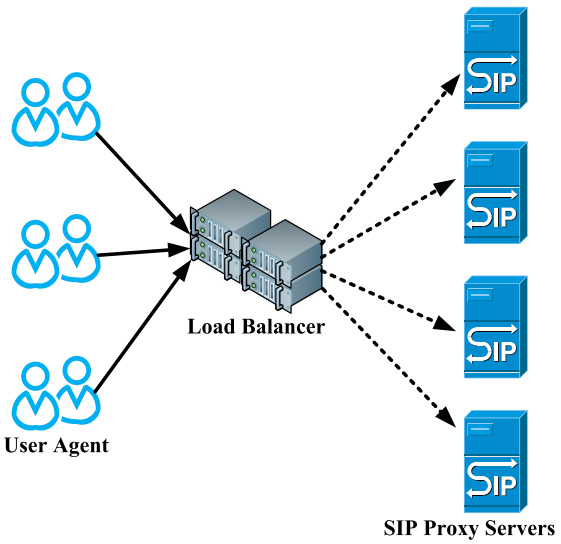


Figure 7. Overall architecture of the load balancing methods.

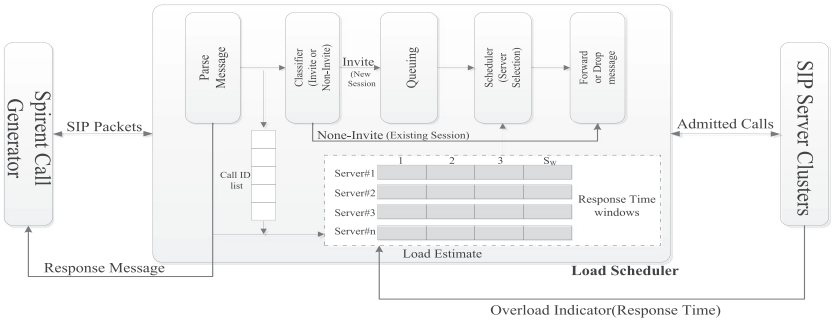


Figure 8. Architecture of the load balancer in [14].

load sharing for web servers are a well-studied problem [8–11]. In [12], various failover and load sharing methods were compared for registration and call routing servers based on SIP. Load sharing techniques are based on DNS, SIP identifiers, network address translators and servers with the same IP addresses.

In [13], a load balancer was featured along with three algorithms, which were based on counting the number of transactions or processed sessions on a server, among which the best algorithm was transaction least-work-left. Transaction least-work-left assigns different weights to different transactions depending on their relative costs. In [14], throughput reduction was prevented by balancing the loads among the available proxy servers by the proper use of an implicit mechanism called history weighted average response time. The innovation of the mechanism was in making use of 'response time history' to approximate the load being currently processed on servers. The internal architecture of history weighted average response is portrayed in Figure 8. The structure of load balancer is such that it should navigate the request messages from users to servers; conversely, it should return the messages from servers to users. Components of the load balancer are parse message, classifier, queuing, scheduler, and forward and drop message.

2.2. Overload control methods

The goal of SIP overload control is to keep server throughput close to its capacity in the presence of overload [15]. These methods mostly assume that the SIP is running on UDP. In general, there are two techniques to control overload: local and distributed.

2.2.1. Based on user datagram protocol.

2.2.1.1. Local. Through the local overload control methods, the overloaded server itself has a consistent control over its resource usage without any need for interplay with other network servers. The criteria for identifying the overload in some of these methods are queue length and CPU usage level [16]. An SIP server starts rejecting extra-requests when becoming close to its capacity limit, which is carried out by sending a '`503 Service Unavailable`' message in response to an '`INVITE`'. The main drawback of this method is that the cost of call reject cannot be ignored. When dealing with heavy overloads, the server must use its entire resources for rejecting the surplus calls. Thus, the server will not be able to respond any services.

In this paper, two strategies are used to reduce the cost of the call rejection:

- Rejection in the transport layer before the allocation of resources
- Attention to the most important server resources to overload detection

In [3, 17], occupancy-based algorithms, namely OCC, have been proposed using CPU utilisation as a trigger for rejecting calls ([17] is employed in Section 6 as a competitor, with which this paper's proposed algorithm is compared).

Another well-known example of local overload control mechanism proposed by Ohta [18] focused on the algorithm deciding either reject or accept a new SIP session based on the queue length (that is also employed and compared).

2.2.1.2. Distributed. In distributed mechanisms, the upstream servers control the load of the downstream SIP server, keeping it as close as possible to its capacity. Commonly, the overloaded server monitors its resources and sends a feedback to all upstream servers, which inform them to be overloaded. The upstream servers decrease their forwarding rate accordingly. The distributed methods are categorised into two explicit and implicit ones [3]. Based on the information exchanged among the servers, these methods are also classified into the rate-based, loss-based, signal-based, window-based and on/off control techniques [19].

Within the rate-based methods, downstream server has a consistent supervision over the rate of messages delivered from total upstream servers and informs its admissible rate to them [5, 20]. In loss-based methods, downstream server repeatedly measures its current load and, accordingly, requests the upstream servers to reduce their transmitted load [21]. In window-based mechanisms, the load is not transmitted to the downstream server unless there are empty slots in upstream server window. The basic matter of these methods is window size regulated through using the feedback of the downstream

server [1, 2, 15]. The signal-based methods are those in which the upstream server reduces its transmission rate when receiving '`503 Service Unavailable`' message in order to forbid further transmission of '`503 Service Unavailable`' message from the downstream server [22]. The on/off control method, a given server can either hold off or on its received load by transmission of retry-after feedback. In [3], Shen proposed three window-based distributed overload control methods, in which the downstream server dynamically approximations its capacity and generates a feedback indicating the number of currently accessible window slots. The authors of [23] presented two other explicit feedback-based algorithms using CPU occupancy and queuing delay and a combination of window-based and signal-based methods. In [15], a distributed end-to-end adaptive window-based overload control algorithm was proposed, in which upstream servers use call establishment delay to evolve the load on their downstream servers so that no explicit feedback is needed from the downstream servers. In [24], a distributed end-to-end overload control mechanism was introduced, and the main idea is to extend the controllers at the edge servers to implement a call admission control to the SIP network. The controller calculates the call admission rate based on a finite state machine.

The main deficiencies of current distributed overload control schemes are overload detection and the feedback generation costs CPU time and impacts throughput if performed at the overloaded server. Then, only discrete event simulators have carried out the performance evaluation of almost all the proposed mechanisms when no experimental evaluation is provided [25].

2.2.2. Based on transmission control protocol.

Selection of transport layer protocol is effective for the efficiency of SIP signalling. In [6], different options for the selection of transport layer protocol were qualitatively studied for SIP. In [26], the effect of applying different transport layer protocols on the delay of establishing call was studied. Behaviour of proxy with SIP transactions can be stateful or stateless. The SIP transaction means a request and all the related responses, which are exchanged between two SIP adjacent elements. Stateful proxy should maintain the state information of each transaction for some operations such as potential retransmission of signalling packets. SIP proxies have been designed so that they can register the record of transaction operations for each request (stateful) or does not register the record (stateless) [20]. One of the possible optimisations is that proxy creates transaction considering type of request, CPU load and quality of link only for some SIP messages (for example, '`INVITE`' and '`BYE`') and acts statefully [20]. On the contrary, when packet loss rate is negligible or SIP signalling is carried out through TCP which is a guaranteed transport service, the state is not kept. In [26], a scenario of SIP network on TCP was simulated in ns-2 simulator. Transport of SIP packets on TCP is shown in Figure 9. When n-th packet is transmitted from a TCP transmitter, the transmitter stores that packet in the buffer.

On the side of the receiver, if SIP queue is not filled, the message will be immediately transported to SIP queue. Then, the receiver clears its receipt buffer by sending a response message with the size of the proposed window 1, and transmitter also clears packet n from its transmitted window once it receives the response packet, puts n+1-th packet in its transmission window and transmits the new packet to receiver. If SIP queue is filled, messages cannot be transported to SIP queue, and the receiver transmits the size of the proposed window 0 to packet transmitter. Therefore, SIP packet remains in the receiving TCP buffer. Then, TCP transmitter waits for the transmission of the next packet to receive a response from the receiver regarding size of the proposed window 1. In [19] and [21], SIP simulation resulting from UDP and TCP has been compared in OpenSER and SIPd proxies, respectively. In [27], measurement results were shown to explain the impact of TCP on SIP server's scalability and performance. The authors measured the impact of choosing stream control transmission protocol (SCTP) on the server's scalability. The authors of [28] also measured the scalability and performance of two servers, that is, an echo server and a simplified SIP server, on Linux for both SCTP and TCP. Their measurements clarified that using SCTP does not

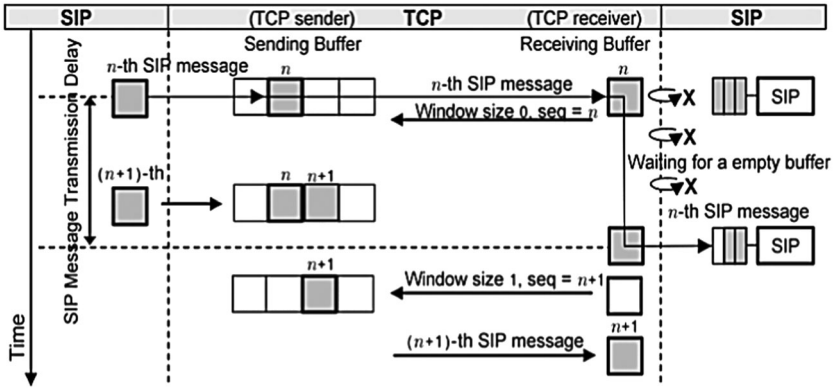


Figure 9. Relationship between transmission control protocol (TCP) and session initiation protocol (SIP) in [26].

substantially affect data transfer latency. In [29], a comparative evaluation of UDP, TCP and SCTP was made based on a model simulating a scenario most likely to profit from congestion control and error correction methods associated with the reliable transport protocols. In [30], the impact of using TLS as a transport on SIP server performance versus the mechanism of SIP over UDP was evaluated and analysed. Using an empirical testbed in OpenSIPS server, OpenSSL, Linux and an Intel-based server, authors showed that TLS can diminish SIP server performance. They used application, library and kernel profiling to illustrate where different costs were incurred. The authors of [7] overload control propose approaches namely connection split and buffer minimisation which split the TCP connection into two, one for '`INVITE`' requests, and the other for all another requests (Figure 10). In order to limit the number of '`INVITEs`' in the server and minimise delay, they minimised the total system's buffer size. Yet, the resulting throughput did not vary greatly from that of the default configuration. The reason was that the number of 'INVITEs' server received from UAC was the same.

During the past decade, most of the overload mitigation mechanisms which have been designed are applied to UDP given SIP, because UDP is the permanent and usual choice for session environments. However, SIP on TCP or on a transport protocol relating to signalling such as SCTP is discussed for several reasons:

(1) Increased demand for having secure signalling with SIP protocol on TLS which is also acceptable by SIP organisation. TLS is a secure transport protocol which can be implemented on reliable transport protocol such as TCP.
(2) Development of SIP in the next-generation networks requires support from the transportability of packets which are larger than MTU but is not possible in UDP.
(3) Connection-oriented transport protocols such as TCP have abundant advantages which can be utilised in SIP networks such as easier firewall.

So, SIP over TCP is becoming more and more popular and will be as a more viable SIP transport choice in the near future.

The SIP over TCP overload control problem possesses two different perspectives when compared with the SIP over UDP overload control moot point. One is TCP's integrated flow control mechanism which equips an intrinsic, existing channel for feedback-based overload control.

The other is the elimination of many application layer retransmission timers that worsen the overload condition in SIP over UDP.

By studying data on the level of Asterisk proxy operating system (Linux), we found that it is possible to use a connection-oriented transport protocol for the communication of this proxy and '`INVITE`' request could be rejected at the stage of establishing communication before the allocation of resources. Before this, requests are rejected at application level and after recognising '`INVITE`' request. Implementation of this mechanism reduces the cost of rejecting call and increasing throughput in overload conditions. In the next sections, we will provide more details.

3. PROBLEM DESCRIPTION

3.1. System set-up

The general structure of the applied test medium is shown in Figure 11. To implement the SIP proxy, the Asterisk open source software [31] is used and installed on PC with Intel Dual Core 2.8 GHz processor and 2 GB memory. For the user agents responsible for the production of SIP traffic, powerful device Spirent Abacus 5000 is used. CentOS Linux PC operating system, version 6.3, is also applied. To study the transmitted and received messages, the reports produced by Spirent and OProfile [32] software are utilised.

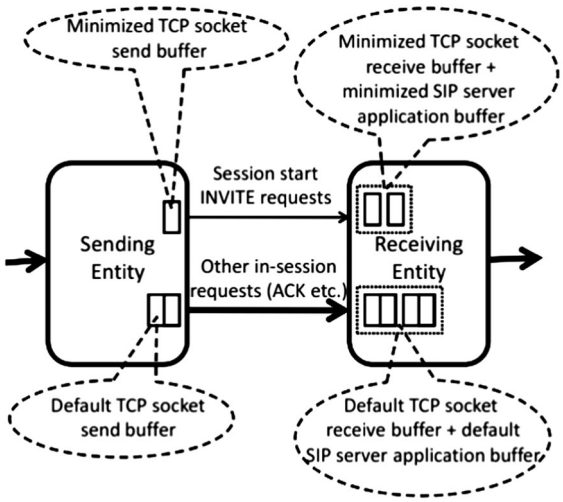


Figure 10. Connection split and buffer minimisation [7].

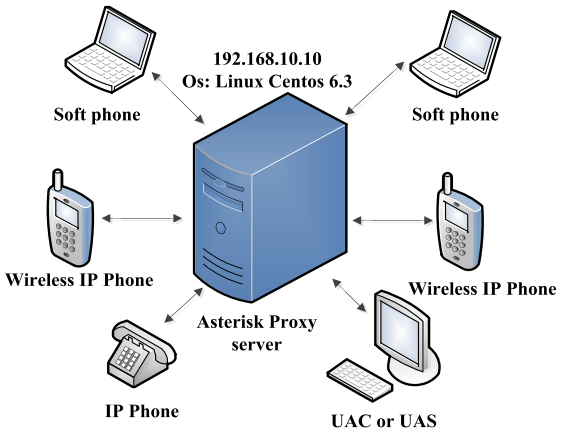


Figure 11. General scheme of testbed.

3.1.1. Asterisk server proxy.

It is one of the most popular open code telephone proxies and a framework for establishing communication. In simple terms, this system converts ordinary computer into a communication server and exchanges sound and image with the data network. Asterisk is supported by Digium Company.

We changed kernel of this proxy according to our desirable algorithm and measured its feedback and performance in C programming language. As shown in Figure 12, it is possible to manage calls or reject calls in transport layer at kernel level. Access to kernel level codes of operating system and changes in kernel space require the set-up and installation of a new kernel on Linux operating system; therefore, a new version of Linux kernel was set up and installed on Asterisk [34, 35].

Internal parts of Asterisk are defined as modules. Application interface module comprises different pluggable commands. It is the normal way of integrating external programmes in Asterisk. The Asterisk configuration module is mapped onto the abstract database schema. The supported databases are ODBC and MySQL. Configuring Asterisk involves the configuration of some important files as *asterisk.conf*, *sip.conf*, *extensions.conf* and *logger.conf*. Codec module determines coding or decoding capabilities. Channel module handles protocols for communication and is responsible for call state representation.

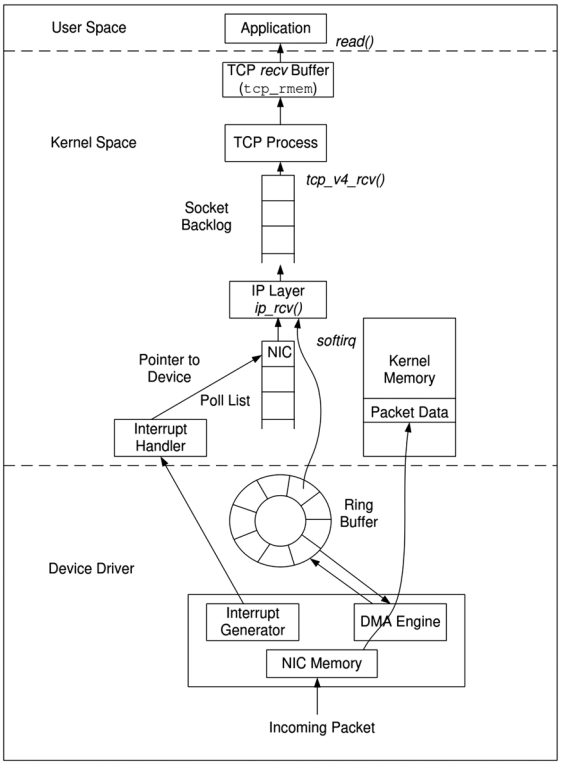


Figure 12. Flow of receiving packet from network card to application layer [33].

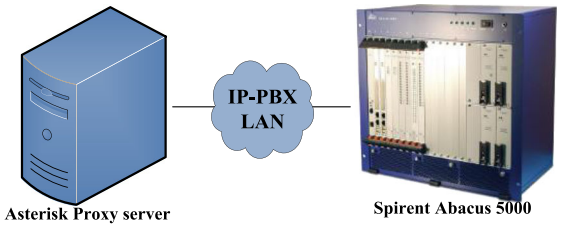


Figure 13. Testbed provided in IP-PBX laboratory [36].

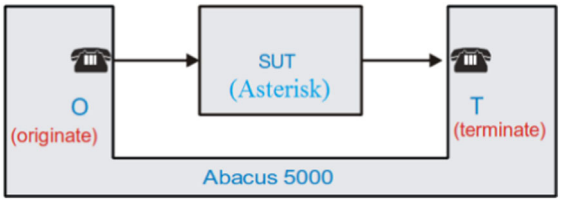


Figure 14. Role of Spirent Abacus 5000 device.

3.1.2. Spirent traffic producer.

To implement user agents, Spirent Abacus 5000 tester was used for the creation of traffic with different transfer rates and different distributions like Poisson distribution. This product is able to test proxy of the subject. Position of this device in call between user agents and server proxy is shown in Figure 13. As demonstrated later, this device plays the role of originator and terminator concurrently. According to Figure 2, signalling load is produced by two UAS and UAC user agents, the role of both of them is played by Spirent Abacus 5000 device (Figure 14).

3.2. Studying effect of transport layer protocol on efficiency of Asterisk proxy

Call production rate starts from the amount as low as 10 calls per second (cps) and goes up to as high as 1000 calls per second, having Poisson distribution. In this case, only one proxy is considered, and no overload control mechanism is contrived on the Asterisk proxy. In Figure 15, processing load is compared in that transport layer protocols are UDP and TCP. In this diagram, the horizontal axis represents the amount of load received by proxy, and vertical axis shows the average CPU utilisation. In this figure, the extent to which Asterisk uses CPU and general load of CPU when the proxy is stateful is demonstrated. According to this figure, the extent of CPU use is maximised at the approximate rate of 800 calls per second. However, when we use UDP protocol, the processor is saturated at a little higher rates (approximate rate of 900 calls per second).

As can be seen in Figure 16, the proxy's memory (here means 'average number of packets in the buffer') is approximately empty before overload occurrence, because every message is drawn out of queue and processed upon reaching the proxy. In overload conditions, many packages would systematically be waiting for the queue to receive

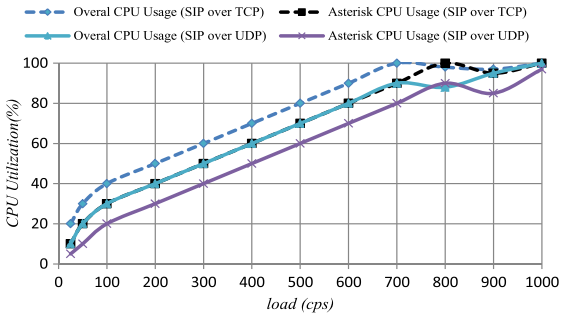


Figure 15. Comparison of processing load (SIP over TCP or UDP).

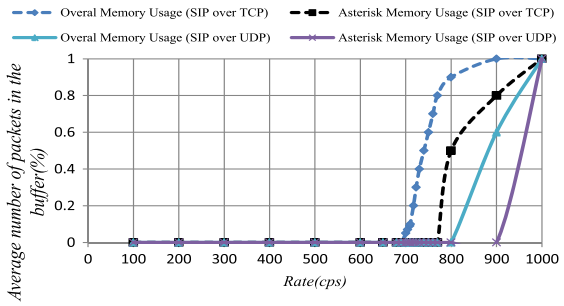


Figure 16. Comparison of memory used (SIP over TCP or UDP).

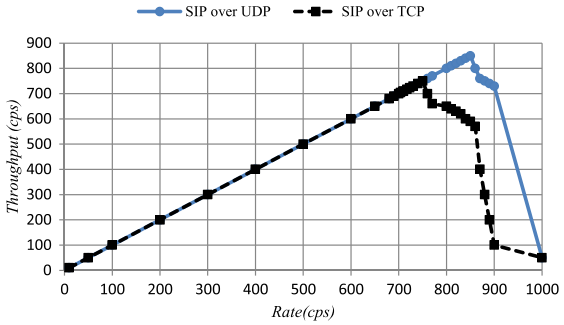


Figure 17. Throughput.

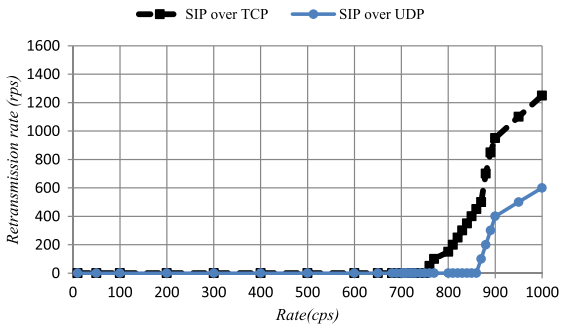


Figure 18. Retransmission rate of 'INVITE' messages.

service, if SIP over TCP increased use of memory a little more.

Although the use of TCP causes faster proxy throughput drop with respect to rate, it does not impose incremental processing and memory cost (Figures 15 and 16) on the proxy compared with UDP. Figure 17 shows throughput (the number of successful calls per unit of time) and compares the cases of UDP and TCP. As demonstrated in this figure, by using UDP for transport layer, the Asterisk is able to maintain its throughput up to about 900 cps, whereas if TCP were used, server's throughput would be approximately equal to the 800 cps. Figure 18 illustrates retransmission rate for 'INVITE' requests made in the user agent. As expected, no message is retransmitted before the received call rate reaches server's capacity (in the SIP over UDP case, approximately 900 cps and, in the SIP over TCP case, approximately 800 cps). But, upon obtaining the received call rate to server's capacity, retransmission rate increases suddenly. Retransmission that occurs in the case of SIP over TCP is due to the normal mechanism of TCP protocol in the transport layer for providing reliability.

However, the retransmission that occurs in the case of SIP over UDP is related to SIP mechanism and its timers to provide reliability in the application layer. When a protocol, like UDP, which does not provide reliability, is used, the SIP protocol itself takes the responsibility in a different form. The retransmission rate of messages in SIP over UDP is less than that of SIP over TCP. Therefore, when the server load reaches the values near its capacity, the number of remaining messages in the server queue significantly increases, and higher processing and memory costs are required from the server to examine and respond to them (Figures 15 and 16). This issue would also decrease the throughput (Figure 17). On the contrary, when UDP is used for transferring SIP message packets, the number of retransmissions decreases (Figure 18), the number of messages remained in the server queue reduces, and consequently, the memory and processing overheads of the server in the heavy load condition reduce (Figures 15 and 16) but increase the throughput (Figure 17). We intend to improve SIP on TCP operation and, therefore, benefit from many advantages of TCP by identifying and eliminating the messages to which there are not enough resources to respond.

4. PROPOSED MECHANISM

Because 'INVITE' message indicates a request for a call set-up and imposes the greatest processing and memory overheads to the server, we focused on admission or reject of the 'INVITE' request in the case of overload. TCP is able to reject new communication ('INVITE' message) in transport layer and before allocation of resources at the stage of establishing communication (three-way handshaking) while confronting with overload. Figure 19 illustrates handshaking phase in initiation of SIP over TCP session in UA-to-server communication. The 'SYN' packet is sent to indicate that a new connection is to be established. Therefore, we can detect new session ('INVITE' message) with

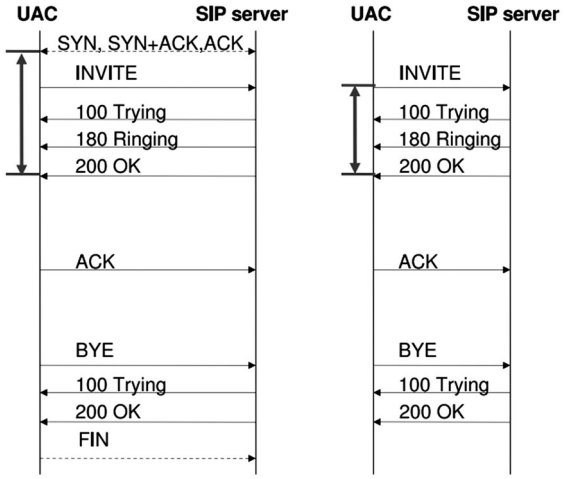


Figure 19. 'TCP 3-way handshake' connection establishment procedure ('SYN, SYN-ACK, ACK') [21].

'SYN' packet at kernel level. This mechanism which is executed in Asterisk proxy Linux kernel reduces cost imposed on the server for the rejection of call in application layer and is imposed on the server with '503 Service Unavailable' response. Other advantages of this approach are as follows: (1) Rejecting 'INVITE' requests using the mentioned methods causes new resources such as socket not to be allocated to the rejected calls and the rejected calls do not have overhead for the system, (2) The number of the exchanged messages is reduced and the overhead imposed on transmission link considerably decreases. In Figure 20 [37], change in the state of each TCP stage is shown. Requests are included in queue to be accepted at the first opportunity. Therefore, 'INVITE' message can be rejected at the stage of processing order not to enter stage of request acceptance by recognising the occurrence of overload on the Asterisk used processing resources. Accuracy of this fact is proved using practical test for several times and by creating real traffic. We observed the server resources to reject the request in the case of an imminent overload and lack of sufficient resources. The fastest reaction would be taken in response to the overload, and the attempt is made not to waste any extra resources under critical overload conditions, because under overload condition, most of 'INVITE' requests would finally be rejected by the basic mechanism of SIP

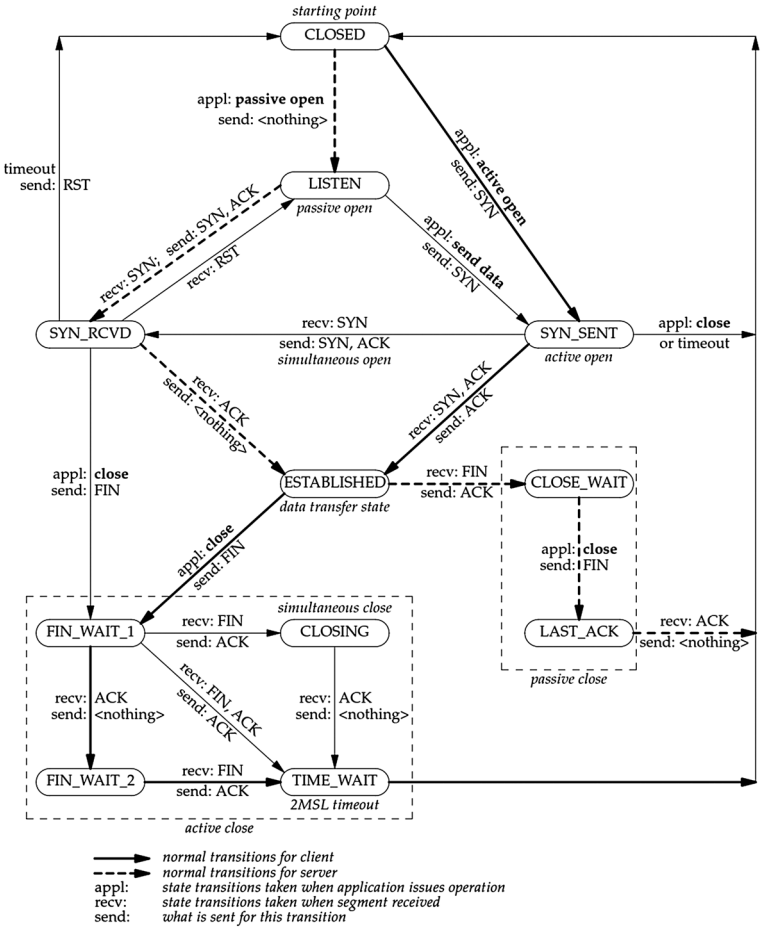


Figure 20. TCP state machine [37].

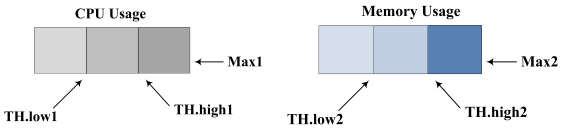
protocol, which causes an extreme drop in the throughput (Figure 3).

As discussed earlier, the most important resources effective for the overload phenomenon are CPU and memory. The following mathematical relations are used to detect the imminent overload such that effects of both of the resources would be appropriately considered. To recognise the occurrence of overload in proxy, two high and low threshold limits were considered for CPU and memory resources (Figure 21), and their optimal values were determined using the tests for several times.

TH.low1 and *TH.high1* thresholds were considered for the server CPU, while *TH.low2* and *TH.high2* thresholds were considered for the server memory. Also, *Max1* and *Max2* indicate the maximum available CPU and Memory, respectively ($TH.low\epsilon[TH.high], TH.high\epsilon[TH.low, Max]$). According to Figure 21(a), p_{cpu} value was calculated using Equation (1) in order to observe CPU condition ($Pcpu\epsilon[0,1]$). I_{cpu} shows the average usage of server CPU ($I_{cpu}\epsilon[0, Max1]$).

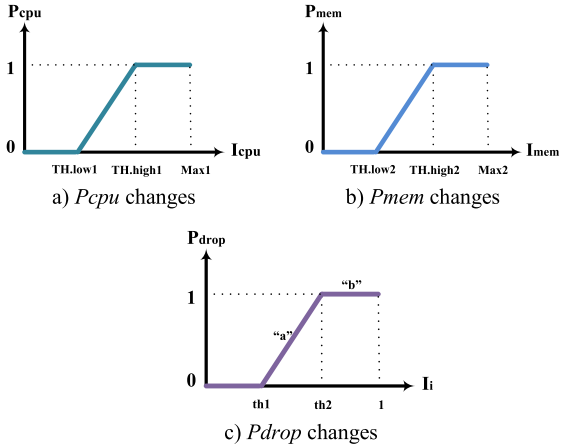
$$p_{cpu} = \begin{cases} 0, & I_{cpu} < TH.low1 \\ \frac{I_{cpu} - TH.low1}{TH.high1 - TH.low1}, & TH.low1 < I_{cpu} \leqslant TH.high1 \\ 1, & I_{cpu} \geqslant TH.high1 \end{cases} \quad (1)$$

If the average usage of server CPU (I_{cpu}) is less than *TH.low1*, the value of p_{cpu} would be zero; however, if this value is higher than *TH.high1* threshold, the value of p_{cpu} would be equal to one. If the average usage of server CPU lies within *TH.low1* and *TH.high1* thresholds, then according to Equation (1), p_{cpu} value would also be a number between 0 and 1. Changes in the p_{cpu} are shown in Figure 22(a). To observe memory condition, a similar process was used which is shown in Equation (2) and Figure 22(b).



a) CPU usage of server b) Memory usage of server

Figure 21. Thresholds considered for the usable resources.



a) *Pcpu* changes b) *Pmem* changes

c) *Pdrop* changes

Figure 22. Change of resources considering variations of thresholds.

$$p_{mem} = \begin{cases} 0, & I_{mem} < TH.lw2 \\ \frac{I_{mem} - TH.low2}{TH.high2 - TH.low2}, & TH.low2 < I_{mem} \leqslant TH.high2 \\ 1, & I_{mem} \geqslant TH.high2 \end{cases} \quad (2)$$

Now, according to the conditions of the resources, the probability of rejecting '`INVITE`' messages is found. Because we want the probability of rejecting the messages to be dependent on both CPU and memory, the following equation was used:

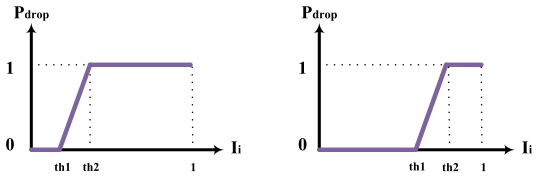
$$I_i = \alpha p_{cpu} + (1 - \alpha)\, p_{mem} I_i \epsilon\, [0, 1] \quad (3)$$

Equation (3) relates p_{cpu} and p_{mem} values to each other via factors α and 1- α (I_i). Factor α indicates weight or importance of CPU or memory in making the decision about rejecting the messages. Importance CPU and memory in decision-making can be changed by varying factor α, because CPU and memory resources are not equally involved in the overload phenomenon. As observed in Figures 15 and 16, by increasing the transmitted load towards the server (axis *x*), the changes in CPU and memory would not be the same. By increasing the load up to the overload point, memory usually would be empty; however, the CPU would increase at a fixed rate. In the overload condition, memory suddenly increases and is filled. Then, CPU goes through the same process. According to Figures 15 and 16 and based on the aforementioned discussion, it can be maintained that, by observing CPU (p_{cpu}) condition, it is possible to more precisely predict the overload, because Figure 15 shows by increasing the overload, CPU usually shows an increasing behaviour at a constant acceleration, whereas variations of memory (p_{mem}) would be more sudden. For this reason, p_{cpu} factor would be greater in Equation (3) ($1 - \alpha < \alpha$). Because $\alpha\epsilon[0,1]$, we can conclude from 1- $\alpha < \alpha$ that: $\alpha > 0.5$. In conclusion, CPU changes are more effective in identifying the overload; however, by considering factor 1- α, we would not ignore memory changes in identifying the overload. Finally, using Equation (4), the probability of rejecting '`INVITE`' messages would be obtained according to conditions of memory and CPU.

$$p_{drop} = \begin{cases} 0, & I_i < th1 \\ \frac{Ii - th1}{th2 - th1}, & th1 \leqslant I_i < th2 \\ 1, & I_i \geqslant th2 \end{cases} \quad (4)$$

In Equation (4), two thresholds *th1* and *th2* were considered ($th2\epsilon\,[th1, 1]$ and $th1\epsilon\,[0, th2]$). If I_i value, which is obtained from Equation (3) is less than *th1*, there is zero probability for rejecting '`INVITE`' messages (p_{drop}).

It means that server is in a desirable condition according to the conditions of resources and there is no overload. If I_i value is more than *th2*, then there would be an overload in the server and '`INVITE`' messages will be rejected with 100% probability until conditions of the resources return to the normal condition again. When I_i is between two *th1* and *th2* thresholds, '`INVITE`' messages will be rejected with $(Ii - th1)/(th2 - th1)$ probability to prevent the overload ($p_{drop} \epsilon [0, 1]$). This mechanism is a method of proactive management of resources to prevent the overload, because by rejecting the messages in excess of the server's capacity, resources would be managed for responding to the available calls. This mechanism provides a situation in which the resource usage leading to the lack of productivity of resource is not so low and leading to overload at the time of call requests is not so high. Figure 22(c) shows the variations of p_{drop} with changes in I_i, in which phase 'a' is a phase preventing the overload and phase 'b' is the overload control phase. Values of two *th1* and *th2* thresholds indicate how strict the overload is encountered. In other words, if we consider a small value for them (Figure 23(a)), the usage of resources would decrease and lead to lower productivity; however, by quickly rejecting the messages, the overload would never take place. But, if we consider larger values for *th1* and *th2* (Figure 23(b)), more resources would be utilised and increase the probability of the overload because of lack of resources. Therefore, precisely setting the threshold values for maintaining trade-off between productivity and resources would be important. In its simplest and one of the most practical forms, productivity can be defined as a total throughput of different flows passing through a server. Consequently, we aim to provide a balance between usage of server resources and increase in the throughput with the condition of no overload. Values of *th1* and *th2* thresholds which were obtained by several tests are presented in 'Performance evaluation' section.



a) Small values for the thresholds b) Large values for the thresholds

Figure 23. Probability diagram of dropping '`INVITE`' requests with respect to *th1* and *th2* threshold values.

The flowchart representing the proposed mechanism is shown in Figure 24. As observed in this figure, TCP determines the type of message before processing the messages. If the message were something other than '`INVITE`' message, it would be accepted. According to Figure 20, it would enter the processing and resource allocation set-up phase. However, '`INVITE`' messages are queued, and the server resources are checked for their acceptance or rejection in each Tn second period. Thus, the probability of p_{drop} is calculated according to Equation (4), which helps make the decision about the acceptance or rejection of

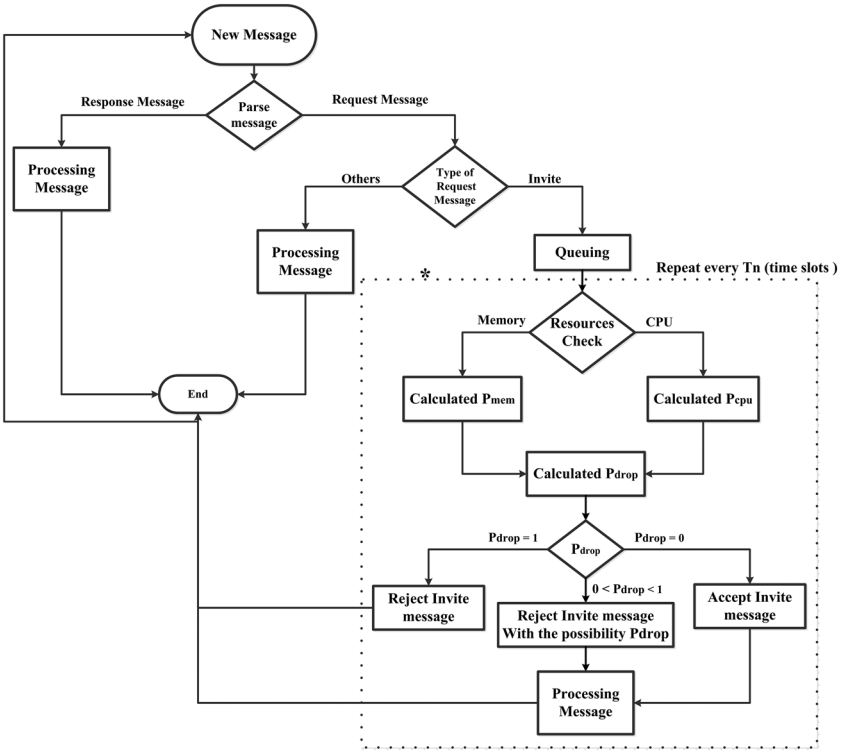


Figure 24. Flowchart of the proposed algorithm.

'INVITE' messages. If there were not enough resources, the message would be rejected by TCP before allocating resources for the set-up; in the case of sufficient resources, the 'INVITE' message would be accepted and enter the cycle for the set-up (Figure 20).

Simple calculation with no computational complexity is one of the advantages of the proposed algorithm. Such an approach does not need to modify the SIP protocol and can be rapidly deployed in SIP servers. The proposed method is intended to guarantee that the downstream servers of the controlled server do not receive a call rate higher than the capacity of the upstream server, which makes it convenient for use in edge servers of an SIP network in order to absorb extreme incoming load and keep the core servers safe.

The pseudocode box * in the proposed flowchart is presented below:

For each T_n
Calculate the average CPU usage (Icpu)
Calculate probability Pcpu
{If (Icpu <TH.low1)
Pcpu =0
If (TH.low1<Icpu <TH.high1)
$Pcpu = \frac{I_{cpu} - TH.low1}{TH.high1 - TH.low1}$
If (TH.high1<Icpu)
Pcpu =1}
Calculate the average memory usage (Imem)
Calculate probability Pmem
{If (Imem <TH.low2)
Pmem =0
If (TH.low2<Imem <TH.high2)
$Pmem = \frac{I_{mem} - TH.low2}{TH.high2 - TH.low2}$
If (TH.high2<Imem)
Pmem =1}
Calculate the Ii
$\{I_i = \alpha p_{cpu} + (1-\alpha)\, p_{Mem}\}$
Calculate the probability Pdrop
{If (Ii <th1)
Pdrop =0
If (th1<Ii <th2)
$Pdrop = \frac{Ii - th1}{th2 - th1}$
If (th2<Ii)
Pdrop =1}

5. STUDYING EFFICIENCY OF THE PROPOSED MECHANISM

We aimed to study the performance of the proposed mechanism under overload conditions. Figure 25 shows the SIP server studied in this article. Figure 25(a) [18] indicates the connections between n number of the proxy servers from which the nth proxy server would face overload condition. Each proxy server has m-1 registered users and, eventually, m×n-1 sessions would be established. As requests for session increase, we continuously increase the available proactive sessions in the nth proxy server to reach the overload level. Our goal is to increase the bearable threshold by the nth proxy by increasing the traffic. As observed in Figure 25(b) [18, 26], the number of TCP connections also increases with the increase in the call requests. We aim to examine our proposed mechanism for the condition, under which these connections are increased. Therefore, to transmit the traffic and to increase the number of connections, the traffic production device Spirent that took the responsibilities of 0 to n-1 proxy servers in this scenario was utilised. We also used the Asterisk Open Source software for the nth proxy server illustrated in Figure 25(c).

We compare the proposed mechanism, named 'TMR' (transport layer mechanism based on resource management) later, with Ohta and the OCC algorithms. The TMR has been tuned by using the optimal parameters found with tests for several times, that is $TH.low1 = 3{:}5$, $TH.high1 = 10{:}3$, $TH.low2 = 19{:}2$, and $TH.high2 = 130{:}5$, $Max1 = 100\%$, $Max2 = 512$, $\alpha = 0.8$, $th1 = 8$, $th2 = 9$. Before showing the results of the experimental evaluation, we briefly explain the other two considered mechanisms, that is, Otha and OCC. The details related to the server configuration are those mentioned in Section 3.A.

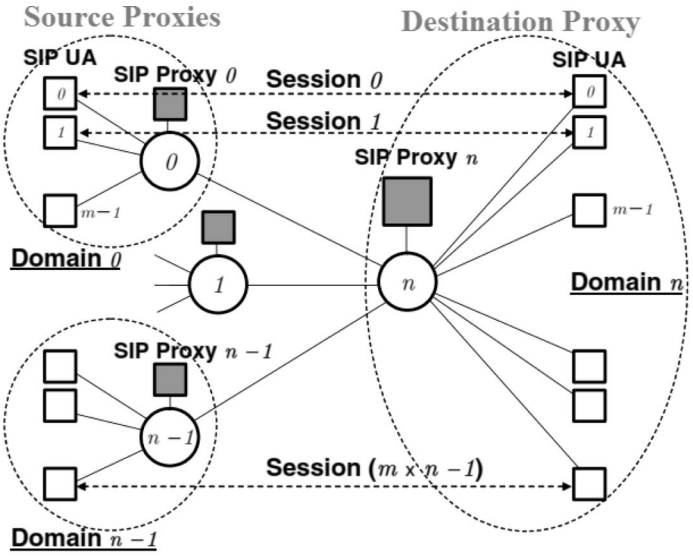
Ohta's method is a simple queue-based algorithm that differentiates between two different states of the SIP server: normal and overload. During the normal state, the SIP server forwards the whole received messages. When the queue length trespasses a high watermark value, the SIP server enters into overload situation. In this condition, it rejects all the requests. The normal condition is entered again when the queue length becomes less than the low threshold value. Queue buffer size is set to 1000 and low, and high threshold values are set 400 and 800, respectively, suggested in [18]. The OCC method regulates the probability f (Equation (5)) of accepting an incoming 'INVITE' message based on measurements of the CPU load C to drive it to a target utilisation C_T. We have tried to obtain the maximum throughput with OCC by setting target CPU utilisation to 90% suggested in [17].

$$f_{k+1} = \begin{cases} f_{min} & \emptyset_k f_k < f_{min} \\ 1 & \emptyset_k f_k > 1 \\ \emptyset_k f_k & \text{otherwise} \end{cases} \tag{5}$$

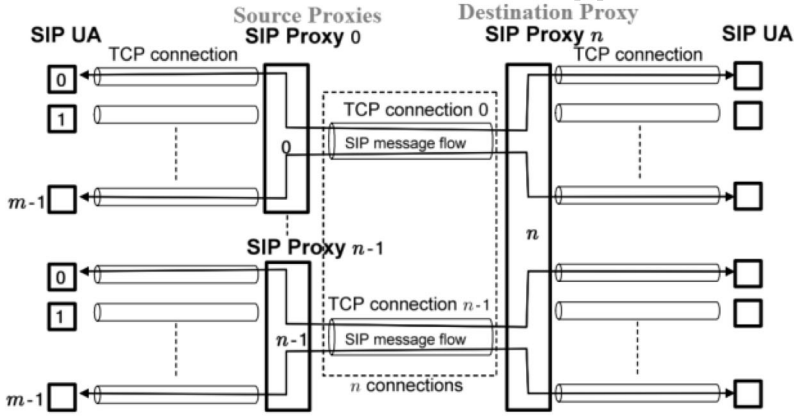
where f_{min} indicates the threshold for the minimum fraction of traffic accepted and $\emptyset_k = min(C_T/C_k \emptyset_{max}).\emptyset_{max}$ explains the maximum possible multiplicative increase in f from one period to the next [17].

5.1. Throughput

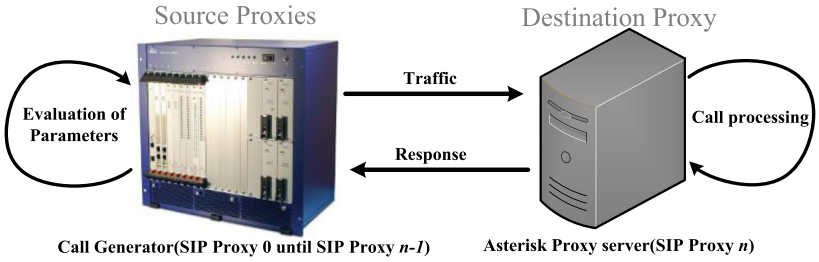
Server throughput is the number of successful calls in time unit. Figure 26 shows SIP server throughput. Also, it demonstrates the number of the established calls in time unit in terms of call request rate for four states: (1) basic mechanism of SIP protocol (rejection of call in application layer); (2) TMR method (our proposed mechanism); (3) Ohta; and (4) OCC methods. Thus, the proxy throughput increases was entry load increases to reach saturation through passing this threshold. Proxy throughput not only increases but also decreases, which means the failure of proxy to establish some calls. The conducted studies have



a) General scheme of the connections and position of the n[th] proxy under overload condition [8]



b) n-1 TCP connections exist between n servers [8, 9]



c) Testbed

Figure 25. Network and testbed model.

clarified that this phenomenon is due to the accumulation of proxy buffers with SIP packets, which lead to overflow and increase of packet loss rate after a while. At the same time, the use of call rejection mechanism in the application layer and transportation of packets with UDP protocol will worsen the situation. As expected, the saturation limit is higher when the proxy is equipped with overload mitigation mechanism in TCP layer. Also, Figure 26 shows throughput when perfect overload control is carried out (curved labelled 'Theoretical'), which keeps throughput at the maximum server capacity of 800 cps. As shown in Figure 26, a greater throughput could be obtained because TMR is a more intelligent mechanism than OCC and Ohta. In TMR method, simultaneous and accurate use of both

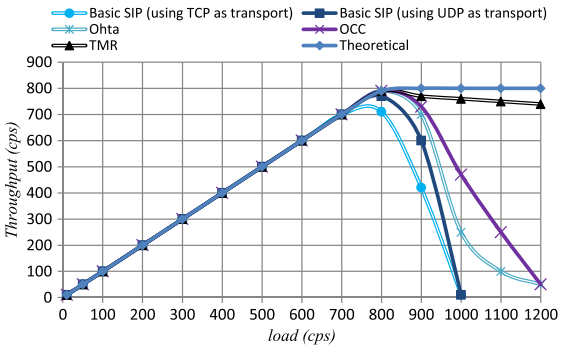


Figure 26. Studying throughput.

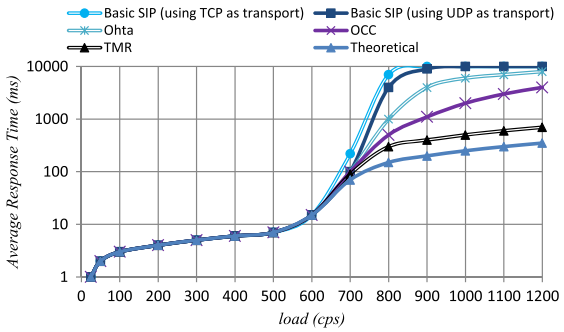


Figure 27. Studying average response time.

CPU and Memory criteria and also precise setting for the parameters introduced in this method cause a less significant drop of throughput than other methods; even under the overload condition to more than the server's offered capacity, this method was able to keep its own throughput near the theoretical throughput as much as possible.

5.2. Average response time

This evaluation criterion is the average time between transmission of '`INVITE`' message by UAC and the time of receiving '200 Ok' from the proxy. The evident phenomenon shown in Figure 27 is sudden delay in call for the loads of above a threshold limit. In the configuration in which a proxy uses the proposed mechanism of this paper, this threshold value is higher than the corresponding saturation limit shown in Figure 26. The reason is that a high amount of the processing power of server is spent for the rejection of calls in contact rejection method with SIP because of the cost of rejecting 'INVITEs' on the application layer and server responds to the received calls later.

Figure 27 shows that the TMR method has shorter average response time than OCC and Ohta, which are because of high speed of TMR method in rejecting the requests for which the server does not have enough available resources to manage. Because the TMR method proactively identifies the messages in excess of the capacity of the server resources and rejects them in TCP layer before allocating resource for them, the number of the retransmissions would reduce, and also the average time for responding to the admitted messages would significantly decrease. As a result, a server has enough time and resources to respond to the available calls.

5.3. Retransmission rate

Retransmission rate of '`INVITE`' messages is presented in Figure 28. In the absence of TMR mechanism, no request will be retransmitted before the input calls reach the capacity level of the proxy; however, as soon as the input rate reaches capacity level of the proxy, the retransmission rate suddenly increases and significantly increases the imposed load on the proxy. Finally, the server memory would be filled with the repeated copies and all the CPU processing capacity would be spent on checking the repeated messages, which also cause a drop in the throughput. The strong point of the TMR method is the timely deletion of copies of the messages to prevent the server from being filled with messages from which the server does not have enough power and resources to respond and is only a waste of the server resources.

5.4. Memory resource

As clearly observed in Figure 29, the used memory by the mechanism proposed in this article does not exceed 300 MB even in the traffic peak time which is less than

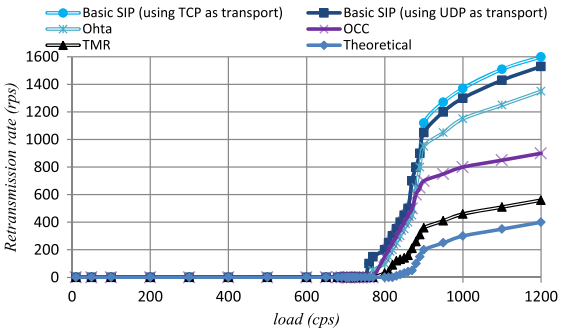


Figure 28. Studying retransmission rate.

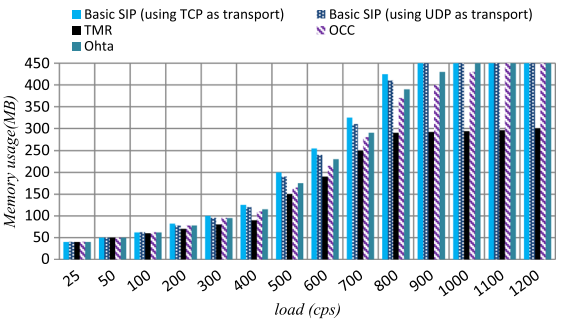


Figure 29. Studying memory used in Asterisk for five methods.

DOI: 10.1002/ett

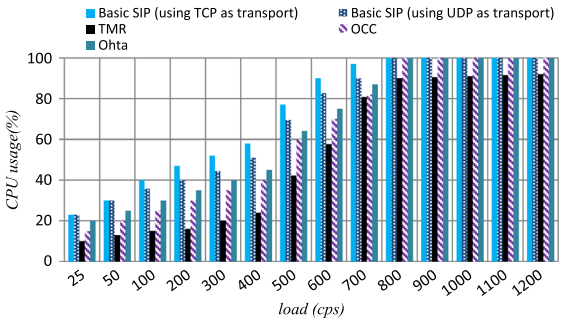


Figure 30. Studying CPU used in Asterisk for five methods.

that in application layer for the call rejections (the total memory considered for this test scenario was 500 MB). If there were no overload, the messages in the proxy queue would be processed upon arrival followed by the appropriate transaction. However, in the case of overload, the proxy queue is continuously filled, but the proxy does not have the necessary time to appropriately process and respond to them. The only fast solution is to reject the new calls in this condition. We know that the cost of rejecting a call in the application layer is not zero; therefore, rejecting new calls also requires processing and spending for the proxy. However, if excessive calls are identified and rejected in the transport layer, if necessary, there is no need to buffer the excessive calls in the application layer and spends further cost; therefore, filling in the proxy buffer is practically prevented.

5.5. CPU resource

Because the TMR method will act cautiously about resources, the CPU usage would not experience sudden changes in this method. The TMR method manages the admitted calls in order to manage the resources and prevents their variations. Figure 30 clearly shows that CPU usage in the OCC and Ohta methods also reache their maximum values (100%) and all the CPU processing capability will be used when the server experiences a load greater than its capacity; however, they would face throughput drop (Figure 26). In contrast, the TMR method manages the admitted requests in the server to not only reach a higher throughput (Figure 26) but also use lower resources (Figures 29 and 30).

6. CONCLUSION

Results of different tests show that high call request rate leads to the sudden increase of delay and drop of SIP proxy throughput. The reason for this inefficient mechanism is the SIP application layer protocol when confronting overload so that this mechanism would try to reject a new call by sending '`503 Service Unavailable`' response message through the application layer. In this paper, a TCP protocol capability-based approach was introduced to reduce the cost of rejecting new call and implemented on a real testbed. Results indicated the improved efficiency of proxy during overload. The reason is that the TMR method was aware of the server resources and its preventable nature. In this method, the SIP overload was identified and controlled in the transport layer, which enabled us to optimally use the server resources as well as to reach a throughput near the theoretical throughput, even at the time of the increased requests to more than the capacity of the proxy. One of the future works might be an accurate study on the parameters introduced in this article (e.g. α) to find optimal values for them. Also, in future works, we will study the effect of these parameters on each other. Furthermore, we will extend the idea to SCTP protocols.

ACKNOWLEDGEMENTS

This work was supported by the Ferdowsi University of Mashhad (No. 30683 date of approval. June 9, 2014). Also, the authors would like to thank the IP-PBX Laboratory of Ferdowsi University of Mashhad for its support of this scientific project.

REFERENCES

1. Hong Y, Huang C, Yan J. Modeling and simulation of SIP tandem server with finite buffer. *ACM Transactions on Modeling and Computer Simulation* 2011; **21**: 1–27.
2. Hilt ENV, Shen C, Abdelal A. Design considerations for session initiation protocol (SIP) overload control, 2011.
3. Charles Shen HS, Nahum Erich. Session initiation protocol (SIP) server overload control: design and evaluation, presented at the IPTComm, 2008.
4. Garroppo RG, Giordano S, Niccolini S, Spagna S. A prediction-based overload control algorithm for SIP servers. *Network and Service Management, IEEE Transactions on* 2011; **8**: 39–51.
5. Camarillo G, Kantola R, Schulzrinne H. Evaluation of transport protocols for the session initiation protocol. *Network, IEEE* 2003; **17**: 40–46.
6. Gurbani VK, Jain R. Transport protocol considerations for session initiation protocol networks. *Bell Labs Technical Journal* 2004; **9**: 83–97, Spring.
7. Charles S, Henning S. On TCP-based SIP server overload control. In *Principles, Systems and Applications of IP Telecommunications*. ACM: Munich, Germany, 2010; 71–83.
8. Bryhni H, Klovning E, Kure I. A comparison of load balancing techniques for scalable web servers. *IEEE Network* 2000; **14**(4): 58–64.
9. Suryanarayanan K, Christensen KJ. Performance evaluation of new methods of automatic redirection for load balancing of apache servers distributed in the Internet. In *IEEE Conference on Local Computer Networks*, Tampa, Florida, USA, 2000; 644–651.

10. Damani O, Chung P, Huang Y, Kintala C, Wang Y. ONE-IP: techniques for hosting a serviceon a cluster of machines. *Computer Networks* 1997; **29**.
11. Oppenheimer D, Ganapathi A, Patterson D. Why do internet services fail, and what can be done about it. In *4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA, 2003; 1–15.
12. Kundan Singh HS. Failover, load sharing and server architecture in SIP telephony. *Computer Communications* 2007; **30**: 927.
13. 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.
14. Montazerolghaem A, Shekofteh S-K, Yaghmaee MH, Naghibzade M. A load scheduler for SIP proxy servers: design, implementation and evaluation of a history weighted window approach. *International Journal of Communication Systems* 2015. DOI: 10.1002/dac.2980
15. Azhari SV, Homayouni M, Nemati H, Enayatizadeh J, Akbari A. Overload control in SIP networks using no explicit feedback: a window based approach. *Computer Communications* 2012; **35**: 1472–1483.
16. De Cicco L. Local SIP Overload Control: Controller Design and Optimization by Extremum Seeking. *IEEE Transactions on Control of Network Systems* 2015; **2**(3): 267–277.
17. Hilt V, Widjaja I. Controlling overload in networks of SIP servers. In *Proceedings of IEEE ICNP*, Orlando, FL, 2008; 83–93.
18. Ohta M. Overload control in a SIP signaling network. *World Academy of Science, Engineering and Technology* 2006; **12**: 205–210.
19. Erich MN, John T, Charles PW. Evaluating SIP server performance. *ACM SIGMETRICS Performance Evaluation Review* 2007; **35**: 349–350.
20. Cortes M, Esteban JO, Jun H. Towards stateless core: improving SIP proxy scalability. In *Global Telecommunications Conference, GLOBECOM*, San Francisco, CA, 2006; 1–6.
21. Kumiko Ono HS. One server per city: using TCP forvery large SIP servers, in IPTComm, 2008.
22. Abdelal A, Matragi W. Signal-based overload control for sip servers. In *7th IEEE Conference on Consumer Communications and Networking Conference, CCNC'10, IEEE Press*, Piscataway, NJ, USA, 2010; 990–996.
23. Noel E, Johnson C. Novel overload controls for SIP networks. In *21st International Teletraffic Congress*, Paris, 2009; 1–8.
24. Liao J, Wang J, Li T, Wang J, Wang J, Zhu X. A distributed end-to-end overload control mechanism for networks of sip servers. *Computer Networks* 2012; **56**(12): 2847–2868.
25. Wang J, Liao J, Li T, Wang J, Wang J, Qi Q. Probe-based end-to-end overload control for networks of SIP servers. *Journal of Network and Computer Applications* 2014; **41**: 114–125.
26. Ohta M. Performance comparisons of transport protocols for session initiation protocol signaling. In *4th International Workshop on QoS in Multiservice IP Networks in Telecommunication Networking*, Venice, 2008; 148–153.
27. Kumiko Ono HS. The impact of SCTP on SIP server scalability and performance. In *IEEE Global Telecommunications Conference (GLOBECOM)*, New Orleans, LO, 2008; 1–5.
28. Rosenberg J, Schulzrinne H, Camarillo G. The stream control transmission protocol (SCTP) as a transport for the session initiation protocol (SIP), RFC 4168, 2005.
29. Nagesha, Manvi SS. Performance analysis of SCTP compared to TCP and UDP. In *Advances in Computing and Information Technology*, Vol. 176. Springer: Berlin Heidelberg, 2012; 515–524.
30. Shen C, Nahum E, Schulzrinne H, Wright CP. The impact of TLS on SIP server performance. *IEEE/ACM Transactions on Networking (TON)* 2012; **20**: 4.
31. (Available from: http://www.asterisk.org/).
32. (Available from: http://oprofile.sourceforge.net/).
33. Bhuiyan H, McGinley M, Li T, Veeraraghavan M. TCP Implementation in Linux: A Brief Tutorial.
34. (Available from: https://www.kernel.org/).
35. Salzman PJ, Burian M, Pomerantz O. The Kernel module programming guide, CreateSpace, 2009.
36. (Available from: http://ippbx-lab.um.ac.ir).
37. Seth S, Venkatesulu M. *TCP/IP Architecture, Design and Implementation in Linux*. Wiley-IEEE Computer Society Press, 2009. (Available from: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470377844.html).

View publication stats