Telecommun Syst
DOI 10.1007/s11235-017-0337-9

Design, implementation and performance evaluation of a proactive overload control mechanism for networks of SIP servers

Ahmadreza Montazerolghaem[1] · **M. H. Yaghmaee Moghaddam**[1]

© Springer Science+Business Media New York 2017

Abstract The extent and diversity of systems, provided by IP networks, have made various technologies approach integrating different types of access networks and convert to the next generation network (NGN). The session initiation protocol (SIP) with respect to facilities such as being in text form, end-to-end connection, independence from the type of transmitted data, and support various forms of transmission, is an appropriate choice for signalling protocol in order to make connection between two IP network users. These advantages have made SIP be considered as a signalling protocol in IP multimedia subsystem (IMS), a proposed signalling platform for NGNs. Despite having all these advantages, SIP protocol lacks appropriate mechanism for addressing overload causing serious problems for SIP servers. SIP overload occurs when a SIP server does not have enough resources to process messages. The fact is that the performance of SIP servers is largely degraded during overload periods because of the retransmission mechanism of SIP. In this paper, we propose an advanced mechanism, which is an improved method of the windows based overload control in RFC 6357. In the windows based overload control method, the window is used to limit the amount of message generated by SIP proxy server. A distributed adaptive window-based overload control algorithm, which does not use explicit feedback from the downstream server, is proposed. The number of confirmation messages is used as a measure of the downstream server load. Thus, the proposed algorithm does not impose any additional complexity or processing on the downstream server, which is overloaded, making it a robust approach. Our proposed algorithm is developed and implemented based on an open source proxy. The results of evaluation show that proposed method could maintain the throughput close to the theoretical throughput, practically and fairly. As we know, this is the only SIP overload control mechanism, which is implemented on a real platform without using explicit feedback.

Keywords Overload control (OC) · Voice over IP (VoIP) · Session initiation protocol (SIP) · SIP server

1 Introduction

SIP is the signalling protocol in application layer, which is used to initiate, modify, and tear down the session between two or more applications [1]. The major components of a SIP network are user agents and server proxies [2]. User agent is the terminal component in SIP session. Figure 1 illustrates the typical SIP trapezoid topology and the standard SIP voice call signalling consisting of the `INVITE-BYE` message sequence. When the caller (User Agent Client: UAC) sends an `INVITE` request to the callee (User Agent Server: UAS), which is routed through SIP proxies in the path between them, setting up a session starts. Returning a `100 Trying` response to the previous hop on the path confirms the reception of this request in each proxy. As the UAS receives the `INVITE` request, it sends back a `180 Ringing` response to the caller. It later also sends back a `200 OK` response when the application accepts the call in charge of taking the call. Finally, in order to acknowledge the reception of `200 OK`, an `ACK` request is sent to the callee. After this three-way handshake, the media is independently established between the two parties. The session is then terminated when one

✉ M. H. Yaghmaee Moghaddam
hyaghmae@um.ac.ir

Ahmadreza Montazerolghaem
ahmadreza.montazerolghaem@stu.um.ac.ir

[1] Ferdowsi University of Mashhad (FUM) Campus, Azadi Sq., Mashhad, Khorasan Razavi, Iran

Published online: 30 May 2017

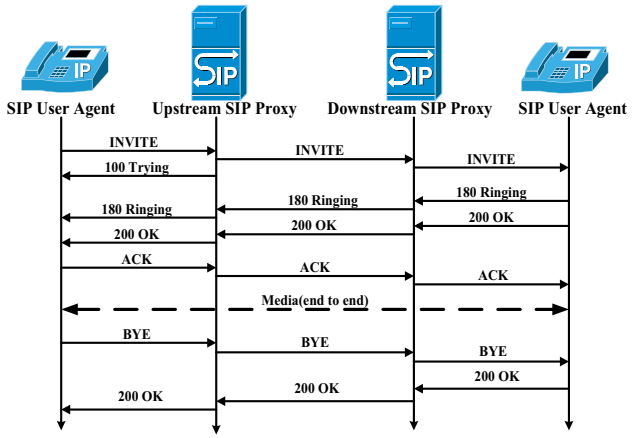

Fig. 1 Exchanged messages for establishing SIP connection

party sends a `BYE` request and the other responds with a `200 OK` [3].

SIP server is an application one. The overload problem in SIP server is distinguished with ones in other HTTP servers for at least three reasons. Firstly, the messages of SIP pass several proxy to reach destination, which could make overload between two SIP proxy servers. Secondly, SIP has several retransmit timers which are used for dealing with packet loss, especially when the packet is sent via User Datagram Protocol (UDP). This could lead to overload on SIP proxy server. Thirdly, SIP requests are used as real time session signalling to have a high sensitivity [4].

Overload in SIP networks occurs when the server does not have resources necessary for handling every received call [5]. Reviews accomplished in overloaded SIP proxy server show that increasing request rate results in sudden increase in delay to establish connection and to drop proxy throughput and therefore to increase in unsuccessful call rates. Therefore, the aim in overload control in SIP is maintaining the throughput of overloaded server near its capacity [6].

The rest of this paper is organized as follows: Sects. 2 and 2 include SIP overload problems and existing related works. In Sect. 4, the detail of proposed overload control algorithm, which is developed in an open source software, is presented. In Sect. 5, we present our network topologies and configurations. Section 6 includes performance evaluation and experimental results. This section also considers the effect of fairness of the proposed scheme. Finally Sect. 7 concludes the paper and outlines future work.

2 SIP overload problem

SIP uses its own reliability mechanism, which uses a large set of re-transmission timers, especially when used on an unreliable transport protocol, such as UDP [7,8]. For instance, Timer A is responsible for scheduling `INVITE` re-transmissions. It starts with an initial value of typically $T_1 = 500$ ms and doubles whenever it expires. After waiting $64 \times T_1 = 32$ s, SIP will stop re-transmitting and declare call failure [9]. This mechanism is useful in case of having unreliable links, but in overload conditions, it is a major cause of performance degradation. During overload, messages that arrive at the overloaded server either get dropped or incur in high delay. Hence, the UACs (and also possibly the upstream proxies) start re-transmitting unacknowledged messages. Furthermore, incoming responses from the UAS, before being processed by the server, experience loss or extensive delay. This makes the server itself to re-transmit some parts of the requests it has already forwarded to the UAS [10]. Therefore, the actual server load increases in a regenerative way so that the call fails.

The curve labeled as "No control" in Fig. 7a shows the dramatic decrease in server throughput. Here, the capacity of SIP server equals to 700 calls per second (cps). When the load increases beyond this limit, the server becomes overloaded and congestion collapse occurs. CPU can consider this throughput degradation as the cost of running the overload control algorithm.

3 Related work

Generally, there are local and distributed methods for overload control [11].

In local control, when SIP proxy server reaches its capacity threshold, it starts to reject requests; SIP estimates this threshold by calculating CPU consumption or queue length [12]. For instance, queue length-based algorithms have been proposed in [13–16]. In queue length-based methods, which are called Bang Bang Control (BBC), two thresholds are considered for the number of packets on the proxy queue. As long as the number of packets on the queue is less than the low threshold, new requests are more likely to be admitted. However, in case the number of packets on the queue is increased until a value of more than the upper threshold, extra requests are more probably rejected. Admission or rejection probability of new calls are often calculated by parameters including current queue length. Occupancy-based overload control algorithms, namely OCC, which use CPU utilization as a trigger for rejecting calls, have also been proposed in [13,15]. However, these mechanisms impose cost itself and when server is overloaded, it is compelled to allocate fraction of resources to reject requests, which in turn decreases efficiency in SIP proxy server.

In distributed method, upstream servers control the load of downstream servers through rejecting requests and try to maintain it under their capacity [17,18]. Generally, the overloaded server monitors its resources and sends an explicit

feedback to all upstream servers with purpose of informing them that it is overloaded. It also possibly communicates the amount of load it can accept. Accordingly, the upstream servers lower their forwarding rate. Design considerations for a SIP overload control mechanism were discussed in SIP overload control (SOC) workgroup in 2011. The resulted design namely RFC 6357 [19] was standardized in the workgroup. Five ways of distributed overload control are described in this standard:

- Rate-based overload control
- Loss-based overload control
- Window-based overload control
- Signal-based overload control
- On/off overload control

Among these methods, window-based algorithms due to more effective, has been more attention recently. Therefore, this paper focused on window-based overload control. Low complexity and simple application are among the characteristics of the proposed method. Other advantages of the proposed method are that this method does not require any cooperation between proxies nor any change in the main mechanism of SIP protocol. We refer to several prominent paper in this context, in the following.

Shen et al. [20] propose three distributed window-based methods in which the downstream server dynamically estimates its capacity and generates a feedback indicating the number of currently available window slots.

In the overload control method introduced in [21], first, transmitting load rate to downstream proxy is determined by considering a window with a constant size. Second, the criterion for rejecting new calls is the difference between predicted latency for making each call and latency stated in Service Level Agreement (SLA) for that call. Furthermore, the rate of transmitting calls by the upstream proxy is accomplished by frequently monitoring CPU application level of the overloaded proxy by implicit feedback. On the other hand, in the proposed method of the present article, an active window controls load rate transmitted by the upstream proxy to each of downstream proxies and window size is constantly adjusted considering the responsiveness of the downstream proxy.

A definite fluid flow model to examine a tandem SIP servers in overload situation is developed by [22]. Also, a fluid flow model to specify the behaviour of the overloaded SIP server is presented in [23]. Overload control problem for a set of SIP proxies with limited resources is proved NP-hard in [24,25]. Khazaei et al. investigate a window based end-to-end overload mechanism with dynamic holonic multi-agent approach [26].

In [27], a window composed of the calls to the downstream proxy is stored in the upstream proxy and also receiving `100 Trying` and `503` messages is regarded a positive and negative feedback, respectively. Also, a counter is used to count the number of positive feedbacks. When the counter reaches a specific limit, one-tenth of the unit is added to the window size. On the other hand, when one transmitting call is expired, the counter is reset and half a unit decreases the window size. Also, to reject new calls in the downstream server, BBC method is applied and queue length of the proxy is adjusted so that the queue latency for the received calls is always less than the values of T_1 counter (the counter related to re-transmission). This issue causes the conversion of increased latency and packet loss or call expiration. However, in the present approach, it is assumed that the target proxy does not apply any call rejection mechanism or overload control. The important point regarding the proposed method in [27] is that receiving the `100 Trying` message as positive feedback for increasing the transmitting load to the overloaded proxy does not have the required efficiency. Because receiving this response message does not mean termination of the signalling related to a transaction and, as long as the transaction signalling is not terminated, the resources allotted for that call in the proxy would not be released. In other words, the imposed load of the related call on the overloaded proxy is not decreased.[1] Therefore, increasing its transmission load under this condition does not seem to be a correct decision. Also, applying the expiration of a call in [27] as the only stimulus for decreasing transmission load rate to the loaded server might impose so much overload on the downstream server that makes its return to the normal condition impossible. Besides, in this algorithm, many parameters must be adjusted and in the case of increased link latency, capacity of the input queue must be reduced to prevent the activation of re-transmission mechanism.

In sum, while local overload control methods suffer from non-negligible rejection cost, most proposed distributed algorithms increase the complexity of the overloaded server by requiring load monitor and calculation of an explicit feedback. Another drawback of using explicit feedback is the delay of the feedback takes to reach upstream servers, which may result in the instability or at least performance fluctuations of the algorithm. In the context of Internet congestion control algorithms, this is a well-known phenomenon [28].

[1] SIP provides a wide variety of response messages regarding request messages. Nonetheless, most of them cannot be used as a confirmation message. For instance, `100 Trying` response message does not show that a message has been successfully delivered. As demonstrated in Fig. 1, when the proxy receives `INVITE` message, in case of being unsure about sending `200 Ok` message within 200 ms to the sender of `INVITE` message, it uses `100 Trying` message to prevent the re-transmission of `INVITE` message. Therefore, `100 Trying` response message does not provide any confirmation for the successful processing of a message. In this article, `200 OK` response message which shows whether a message is successfully processed and removed from the buffer was used as conformation. Therefore, `200 OK` response message was chosen as the confirmation message.

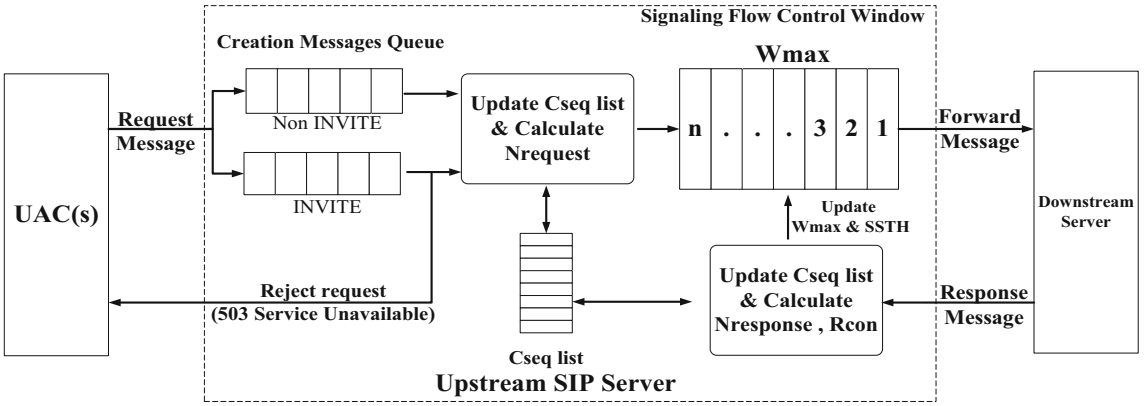

Fig. 2 The basic architecture of proposed flow controller

4 Proposed overload control method

The key issue in window-based algorithms is the window lengths, which should be regulated through feedback in downstream proxy. Therefore, it could be concluded that we can prevent overload through limiting window length. In this algorithms, the upstream server keeps a window of active transactions. If the window has any empty space, new calls will be accepted. However, some vulnerabilities are first, immediate reflectance of current conditions of destination proxy in sender proxy is nerve racking, since the maximum window size is changed via received feedback from the destination proxy. In other words, proxy tolerable rate for each moment is calculated by the proxy itself, and forwarded as a feedback and in the form of current window size to all upstream proxies. This increases the delay too. Second, the processing resource for forwarding feedback is supplied by the proxy processing resources and has an effect on OC during overload periods.

In this section, we propose an effective implicit feedback based method for window-based OC where the number of confirmation messages is considered as a solution for such problems. Also, we limit various request messages to only 5 types (`INVITE`, `BYE`, `CANCEL`, `REGISTER`, and `OPTIONS`). These five requests play an important role as SIP signalling between proxy servers. Furthermore, the main reasons of overload in SIP proxies are these five types of request message.

In traditional window-based overload control, feedback is used to recalculate the maximum window size, while the proposed method in this paper does not use feedback with the aim to conserve server's resources.

Figure 2 shows a diagram of the upstream server where our proposed OC algorithm is employed. The upstream server maintains a signalling flow control window of length n, the number of outstanding transactions that have been started with a destination server. In other words, there is one window per destination SIP server. Note that the downstream (i.e., destination) server is a plain SIP proxy which, does not have any modification.

In this method, priority servicing to received packets is proposed, and two separate queues are considered for `INVITE` and `non-INVITE` messages. Higher priority is allocated to service the `non-INVITE` ones. When the server is faced with overload, as long as the server is in the overload state and no free slot is created in the window, no new call would be admitted (`INVITE` messages are rejected by `503` message). Therefore, the current messages of the system are eliminated by lower probability and the SIP server also prohibits re-transmission. To highlight the logic behind such a strategy, this point must be mentioned that `INVITE` messages generally refer to new sessions, while `non-INVITE` messages indicate transactions or sessions, which have been previously accepted by the system.

A description of message process procedure for upstream server is shown in Fig. 3. `INVITE` messages, which indicate request for setting up new calls and are arrived from UACs (or other servers) are placed in the queue of transaction-make messages.[2] If there are any empty slots in the window [decision box (1) in Fig. 3], server accepts new `INVITE` requests. Otherwise, new messages are rejected by sending a `503 Service Unavailable` message. W_{max} indicates the maximum window size for limiting number of messages. $N_{request}$ and $N_{response}$ show the number of messages and replies, respectively, in a sender proxy. $R_{confirmation}$ (R_{con}) results by dividing $N_{response}$ to $N_{request}$ ($N_{response}/N_{request}$). In first step, W_{max} is initiated at pre-defined value of W_{init}. The amount of load sent to destination proxy could be controlled through regulating maximum window size. After that W_{max} is initiated, sender proxy peruses message type repeat-

[2] Limitation of admitting and transmitting new transactions is only applied to `INVITE` transactions, which is because other transactions are basically related to a working session and maintaining active sessions is preferred to initiating new sessions. Also, by rejecting a response for making a new session (`INVITE` message) by `503` response message, all the signalling related to a single call would be rejected.

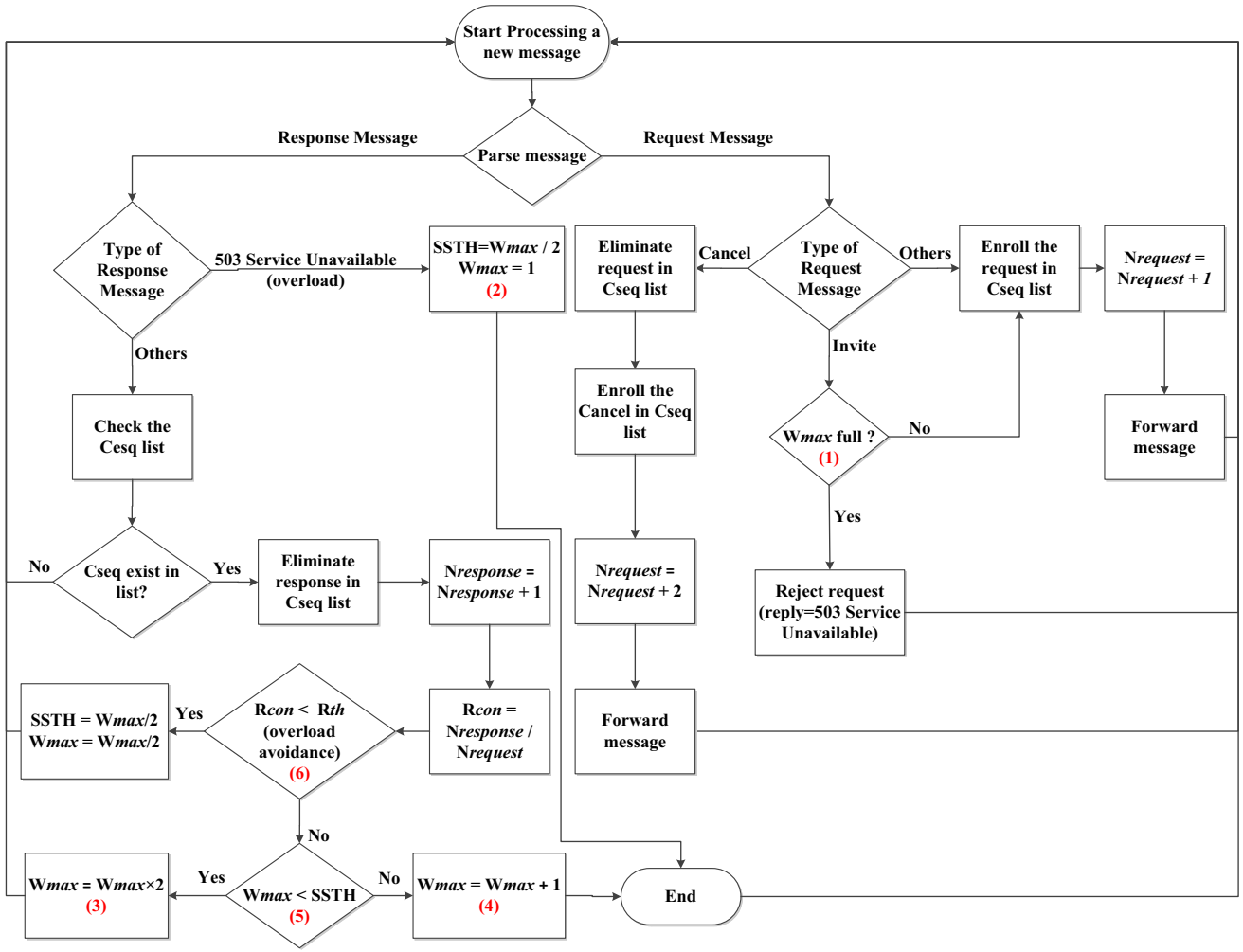

Fig. 3 Proposed flowchart

edly. Receiving a `503` response message by sender proxy means that downstream proxy is overloaded. Therefore, if a message is reviewed and understood as a `503` response, then the value of W_{max} is updated to one and server's overload could be alleviated [box (2)]. In normal conditions, the value of W_{max} increases after it is initiated. While a downstream-overload condition is not triggered, the upstream server will increase its window size as shown in boxes (3) and (4). Box (4) results only in conditions of additive increase in the window size. Also in order to increase window size more rapidly and achieve a higher throughput, for each call setup, window size is multiplied by two until it reaches a specified threshold SSTH [decision box (5)]. Low amount of R_{con} means there is a high load in server. For this reason, whenever R_{con} lowers than a threshold, W_{max} decreases to its half. This threshold is indicated by the symbol R_{th}. This process is shown in the decision box (6). Therefore, as soon as an imminent overload is predicted to occur [decision box (6)], the upstream server halves its window size and also modifies SSTH threshold to half of the current window size. This decreases signalling load imposed on downstream server. Halving SSTH threshold allows window size to reach to half of its previous amount instantly, after which additive increase phase occurs consequently. This is an Additive Increase and Multiplicative Decrease (AIMD) flow current mechanism. It is proved that this mechanism converges to maximum throughput [29].

Therefore, we study about how the values $N_{response}$ and $N_{request}$ are calculated and adapted (to evolve R_{con}) by sender proxy. All SIP messages include *Cseq* header. *Cseq* consists of a sequence number and a method type. The former facilitates recognition and arrangement of transactions. We use *Cseq* header as a means of mapping a request on a response message. For this reason a table consisting of *Cseq* values is made in load sender proxy. Therefore, considering the ratio between response and request messages, dynamic control of server load is feasible. Whenever a request message enters the sender proxy, its type is investigated firstly. In this investigation, there is a distinction between `CANCEL` message and others. General messages except `CANCEL` are recorded in

Cseq table. Then, the new amount of $N_{request}$ ($N_{request} + 1$) is calculated.

In addition to message sending procedure that is described in Fig. 1, a session could be cancelled when the caller wants to cancel a request message sent to the other party or when the callee intends to reject a request message sent by a caller. In other words, `CANCEL` message is used to cancel either a session or a request message sent by the client previously. However a `CANCEL` message should not be used to cancel `non-INVITE` requests. It is feasible to treat reply messages using counting mechanism which was described before, but due to *Cseq* repetition problem `CANCEL` messages may not be applied in an equivalent procedure. If a `CANCEL` message reaches destination proxy before receiving the confirmation message for related request, *Cseq* is recorded in the table repeatedly. Destination proxy generates a confirmation message just for `CANCEL` message, leading to omission of one of two *Cseq* duplications. Accumulation of duplicate values in the table influences R_{con}. To solve this problem, as a `CANCEL` request message enters a load sender proxy, a *Cseq* of target message that had to be cancelled is removed from *Cseq* table. Then a *Cseq* related to `CANCEL` message is recorded in the table and the new amount for $N_{request}$ is calculated ($N_{request} + 2$). This procedure for `CANCEL` message leads to a more accurate value of R_{con}.

After sending more than one request, load sender proxy waits response messages that indicate confirmation. As a response message is received, *Cseq* header field of the message is investigated. If the number of *Cseq* is available in *Cseq* table that was made before, load sender proxy recognizes the response message. In other words, only if the load sender proxy receives reply messages related to request messages sent from it, the response messages are recognized as confirmations, i.e., other messages are treated as castaway. Also, counting `503` response messages is prevented, since as mentioned before such reply messages indicate a retroactive overload in SIP server. In this phase, the message is removed from *Cseq* table, and $N_{response}$ is increased by one. In order to ensure correct and accurate receiving and count of messages, better management, and consequently accurate estimate of the load of downstream servers, a time parameter, T_n, is used in our algorithm and tests. Figure 4 is a descriptive pseudo-code for the process procedure of message in the proposed algorithm for the upstream proxy. Figure 5 schematically shows the manner of maximum variation of the window size, which was discussed.

Overload Control Algorithm

Upstream SIP server behaviour with request message:

```
1. Wmax = 1, SSTH, Rth
2. If INVITE Then
3.        /* new session */
4.        If Wmax = full Then
5.            Reject (request)
6.            /*503 Service Unavailable*/
7.        Else
8.            Update Cseq list
9.            Nrequest = Nrequest + 1
10            Forward (Message)
11. Else If Cancel Then
12.         Update Cseq list
13.         Nrequest = Nrequest + 2
14.         Forward (Message)
15. Else
16.         Update Cseq list
17.         Nrequest = Nrequest + 1
18.         Forward (Message)
```

Upstream SIP server behaviour with response message: Inspect the kind of response in every Tn,

```
19. If 503 Service Unavailable Then
20.          /* overloaded */
21.         SSTH = (Wmax / 2)
22.         Wmax = 1
23. Else
24.          Check the Cseq list
25.          /*this is an existing session*/
26.          Update Cseq list
27.          Nresponce = Nresponce + 1
28.          Rcon = Nresponse / Nrequest
29. If Rcon < Rth Then
30.          SSTH = (Wmax / 2)
31.          Wmax = (Wmax / 2)
32. Else
33.          If Wmax < SSTH Then
34.          /*Slow Start*/
35.              Wmax = (Wmax * 2)
36.          Else
37.          /*Overload Avoidance*/
38.              Wmax = (Wmax + 1)
```

Parameters:

```
Wmax: Maximum Windows Size
Nrequest: The Number of Request
Nresponse: The Number of Response
Rcon: Nresponse/Nrequest
SSTH: Slow Start Threshold
Tn: Measurement Interval to deduct Rcon
```

Fig. 4 Overview of the pseudo-code of the proposed algorithm

5 Network topologies, configurations and practical considerations

The SIP trapezoid topology shown in Fig. 6a, is used as the basic network topology. In this topology, two proxies (the upstream and downstream) are used for handling outgoing and incoming calls, respectively. In order to easily study OC performance, the upstream proxy is made faster than the downstream. All calls are originated from the clients of the upstream proxy and are destined to those of the downstream proxy. In this topology, it is assumed that M transmitter or upstream proxies (e.g. $M = 1$) make an overload in a destination (downstream) proxy by sending many call-making requests. The overload, which the proxy in this topology faces with, is in the form of server-server. In this form of overload,

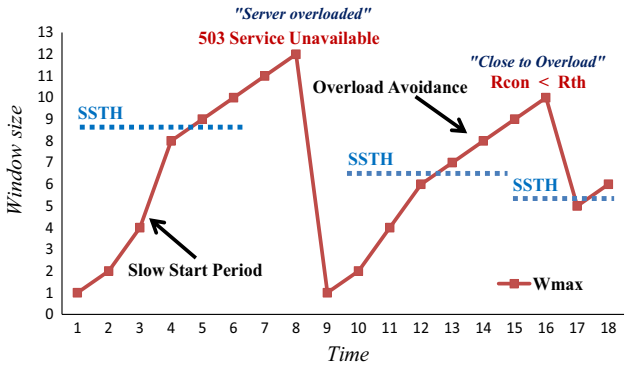

Fig. 5 General scheme of W_{max} variations

a limited number of upstream proxies send a huge volume of traffic to a downstream proxy and leads it to face with overload.

The next network topology (Edge-Core) is depicted in Fig. 6b. This topology consists of a number of edge servers that communicate signalling messages through one core server, which will be overloaded in our experiments. This is a representative of the topology used in the IMS proposed by 3rd Generation Partnership Project (3GPP). All edge servers are assumed and configured to be fast enough so that they are not overloaded. We show how proposed overload control extends to the multiple upstream case while parameters such as fairness as well as throughput are considered.

Our testbed setup, which is composed of two Linux PCs connected over a 100BaseT Ethernet LAN is shown in Fig. 6c. The faster PC functions as the upstream proxy while the slower one is considered as the downstream proxy with a nominal capacity of approximately 700 cps. Asterisk software [30] and Spirent Abacus 5000 tester device are used for implementing proxy servers and user agents, respectively. Asterisk is the most popular open source VoIP telephone system in the world and currently many available IP-PBXs are produced on its basis. Asterisk is based on C programming language and could be loaded in various operating systems such as Linux, Windows, UNIX, Solaris, and Mac OSX. This software uses UDP and TCP transmission protocols to send and receive SIP messages. By receiving SIP messages, it first intercepts the message and then decides whether to reply it or forward it to next destination. In this paper, UDP transmission protocol is used to receive and send SIP messages. Asterisk uses several Worker Processes to receive and send SIP messages. Every Worker Process receives message individually and makes decision about it. In order to process a SIP message, Worker Process should make a connection between the message and the transaction; the message could relate to a transaction, which already existed or it may be new message for which a transaction is created; these transactions are saved in shared memory of Worker Processes [31].

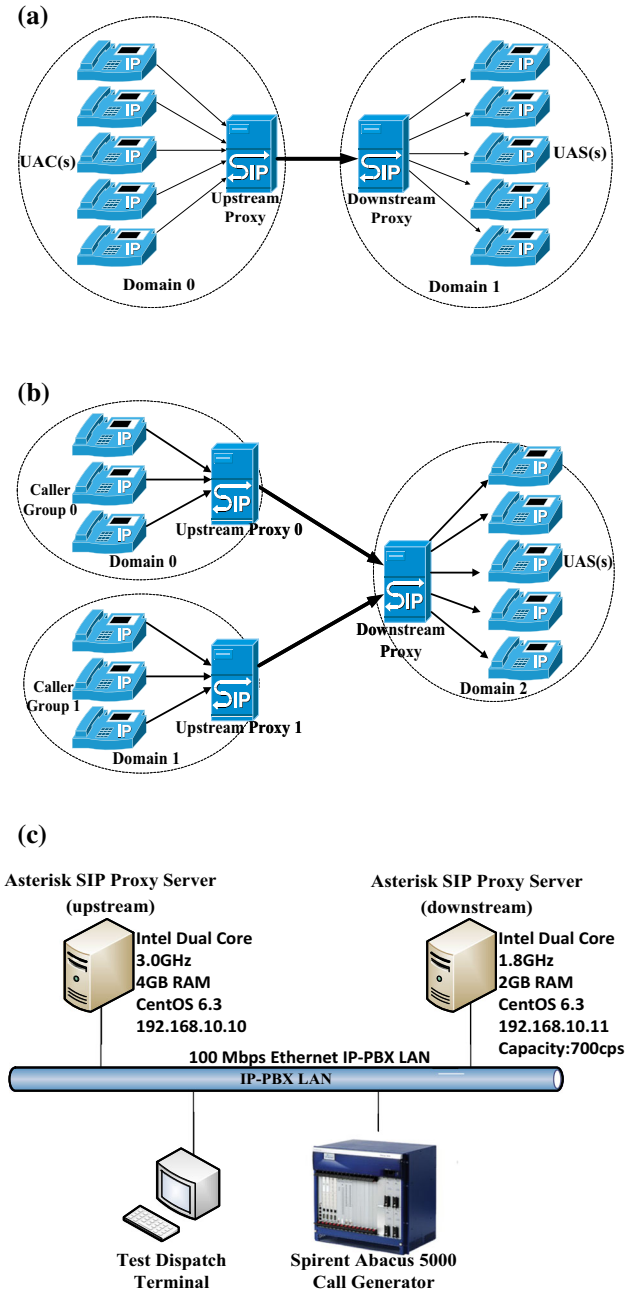

Fig. 6 Topology and testbed setup. **a** Trapezoid topology. **b** Edge-Core topology. **c** Testbed setup

Regarding the above mentioned points, we modify the source code of Asterisk as follow, to achieve a satisfactory proxy for implementation of the proposed algorithm. In our experiment, the configuration of Asterisk is performed via a file named *asterisk.cfg*. This configuration file controls the modules to be loaded and their corresponding parameters. All of the SIP flows are also controlled in several routing blocks defined in this file. The main core of this file is responsible for managing SIP messages and basic functions are placed there. So, the configuration

file controls the modules to be loaded besides providing the capability of assignments of relevant parameters to the user.

We use Spirent Abacus 5000 device to create traffic with different transmission rates and various distributions. This production is able to test the efficiency and extensibility of tested proxy by producing hundreds to thousands of calls. According to Fig. 1, user agents, whose role is played by Spirent Abacus 5000 device, produce signalling load. We have implemented our proposed mechanism on the upstream server, by modifying Asterisk code. However the downstream proxy is intact.

6 Evaluation of efficiency

6.1 Methodology of tests and metrics

Reports produced by Spirent Abacus 5000 device are used to check the time and type of sent and received messages by users. A part of the parameters, which can be set by Spirent for the transmission of desirable load by this device to the proxy along, is given in Table 1. Each run lasts for 100 s after a warmup period of 300 s. The request rate starts from low amount (100 cps) and increases to heavy rates of about 1600 cps. After the maximum rate is reached, it is sustained. Spirent Abacus 5000 is used in open loop mode and calls are generated at the configured rate.

Also, the reports of asterisk are used to measure the status of progression of calls and transactions that occurred in proxy; and also oprofile software is used to measure the processing load of proxy.

There are various criteria to determine the efficiency of SIP [32], amongst which in this study we concentrated on the delay of connection establishment (the interval between sending `INVITE` from UAC to receiving `200 OK` from proxy), retransmission rate, and proxy throughput (the number of successful calls per unit of time).

6.2 Throughput

Figure 7a shows the throughput as a function of rate in received call requests in case of existence and non-existence of overload control method. It represents that proxy's throughput could maintain at around maximum capacity in case of existence of overload control mechanism. This figure shows throughput when perfect overload control is done (curve labeled "Theoretical"), which keeps throughput at the maximum downstream server capacity of 700 cps. Under overload conditions, the throughput of proposed mechanism converges to 635 cps and is almost independent of the load. On the other hand, the throughput of the "Local OC" approach ([13]) is lower than that of proposed mechanism and furthermore decreases as load increases. The curve labeled "WIN-DISC" is an explicit feedback window-based approach proposed in [20] by Shen et al. in which the downstream server calculates and sends back a window size at the end of each discrete control interval of $T_c = 200$ ms, determining the number of new sessions it can accept for the next control interval (in experiments parameter values were used that yielded maximum throughput, i.e., DB = 200 ms and $T_m = 100$ ms, exactly as reported in [20]). The throughput of proposed mechanism is constantly higher than that of WIN-DISC. The poor throughput of WIN-DISC may be

Table 1 Some of the properties and parameters that can be set in Spirent Abacus 5000

Parameter	Description	Value
Call duration	Test duration	100 s
Calling profile	Traffic generation method	Exponential
Number of channel	Number of calls	1600
Number of originate	Number of callers	800
Number of terminate	Number of call-receivers	800
IP Address	IP address of the downstream proxy	192.168.10.11
IP Address	IP address of the upstream proxy	192.168.10.10
Gateway address	Gateway address	192.168.10.9
TCP or UDP	Type of transport layer protocols	UDP
DHCP Settings	DHCP settings	Static IP
IPv4 or IPv6	Using IPv4 of IPv6	IPv4
Authentication setting	Using authentication settings	AKAv1-MD5
Port number	Number of used Port	5060
Retry timeout	Timeout	30 s
Timing adjustment	Call timing settings for initiation	0.3 s
SIP proxy capacity	Theoretical capacity	~700 cps

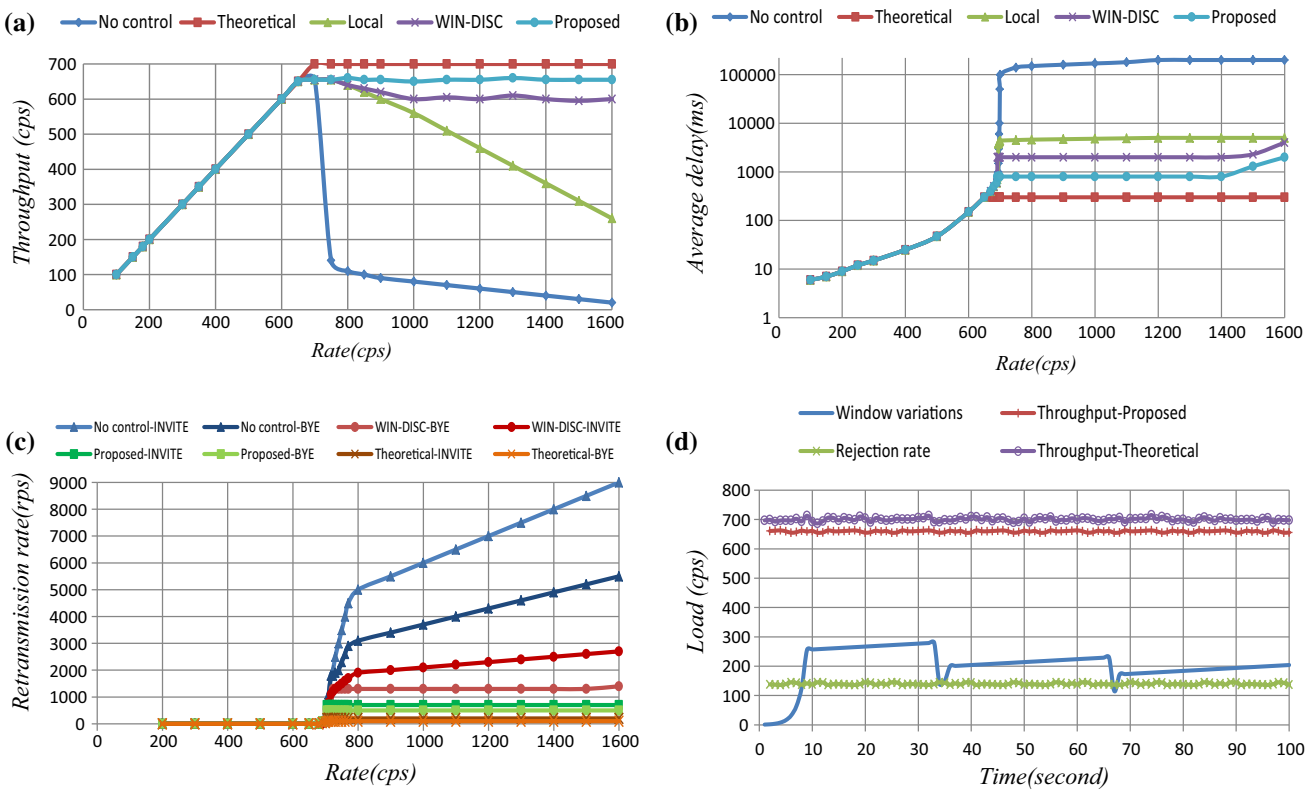

Fig. 7 Result comparison of our mechanism with WIN-DISC and no control methods. **a** Performance. **b** Average delay. **c** Retransmission rate. **d** Windows variations, throughput and rejection rate with rate 800 cps

Table 2 Effect of R_{th} on throughput of proposed method

R_{th}	0.1	0.2	0.3	0.4	0.5
Throughput (load is 680)	677.8	678.3	679.3	679.2	679.2
Throughput (load is 1000)	657	655	650	645	643

attributed to the explicit nature of the feedback used, which is known to result in throughput degradation and stability problems as the feedback loop delay increases.

Using OCC the CPU utilization rapidly fluctuates and the fluctuations become more severe as overload deteriorates. This is due to the regenerative nature of overload where calls progressively take much more CPU time due to retransmissions. However, once enough calls are rejected, the CPU utilization drops dramatically. This will again cause the OCC algorithm to send a feedback causing incoming call rate to increase, which will put the server into overload repeating the same pattern. Due to using R_{con} ($N_{request}/N_{response}$) as opposed to CPU utilization, this is not the case for our proposed algorithm.

We also study about the effect of R_{th} on the throughput of proposed mechanism. Table 2 shows the throughput sensitivity of our algorithm on R_{th} when overloaded with 1000 cps. For larger values of R_{th}, server reacts to overload more slowly and its throughput roughly drops by 10–15%. However, applying too small values (e.g., 0.1) for increases overload false alarms while the downstream server is actually working under normal load.

Also, Table 2 shows that when the input load equals to 680 cps and R_{th} = 0.1, throughput is slightly less than input load but approaches it as R_{th} increases to 0.5. Therefore, a conservative value should be chosen for R_{th}.

For instance, each proxy or network administrator is aware of conventional R_{th}, by putting this parameter as an algorithm input, we can expect to receive the maximum possible efficiency for the network composed of SIP proxies, even in overload conditions. Also, to generalize the proposed initial algorithm which uses the deviation of R_{con} parameter from a specific threshold as a stimulus for decreasing the transmitting load to the downstream proxy, new methods can be presented in future, in which the stimulus for decreasing rate of transmitted load to the upstream proxy would be determined by actively estimating R_{th} depending on the network status.

6.3 Average delay of call establishment

The size of window increases along with receiving new requests, so does R_{con}. When the R_{con} is lower than R_{th}, the size of window reduces to half. Also, SSTH reduces to half of the size of window and then starts to increase again. As it is shown in Fig. 7b, this makes increases the average time of call establishment in this proxy to about 1500 cps linearly and with a growth rate far much lower than the case in which the overload control mechanism is not used.

6.4 Retransmission rate

Figure 7c illustrates retransmission rate for `INVITE` and `BYE` requests from user side, individually. As it is expected, when we use overload control mechanism in upstream server, retransmission rates of messages decrease considerably. Overload leads to loss of `OK` packets related to the passed calls. Therefore, the proxy is required to resend `INVITE` requests related to lost packets. In this case, the increase of retransmission rate makes proxy spend much of its time on resending requests related to ongoing calls and therefore throughput rate of proxy falls considerably. Processing the abundant packets, which exist in proxy's queue, delays the passes calls more and increases the retransmission rate in caller's side.

6.5 Window size variations

In this scenario, Spirent Abacus 5000 generates load traffic with rate 800 cps for 100 s. The upstream server transfers this traffic to the downstream server, while the latter operates at its maximum capacity. Until R_{con} gets lower that R_{th}, window size increases as new requests are received. At this moment both window size and SSTH decreases to half of the window size, then window size starts to increase again. As shown in Fig. 7d, this leads to retention of throughput at the capacity of downstream proxy and rejection of other incoming requests.

6.6 Effect of T_n

Furthermore, numerous tests were conducted to analyze and determine the optimum value for T_n, some results of which are presented in the following section. As given in Fig. 8, with increasing T_n, high fluctuation throughput and its average are decreased. High T_n size means slow reflection of confirmation rate and consequently slow reaction of the window. Therefore, the more the rate of our proxy sampling from the received messages, the more the awareness from the condition of proxy and network and consequently the faster and more precise the reaction for adjusting the window size, which would help in achieving maximum throughput and proxy stability.

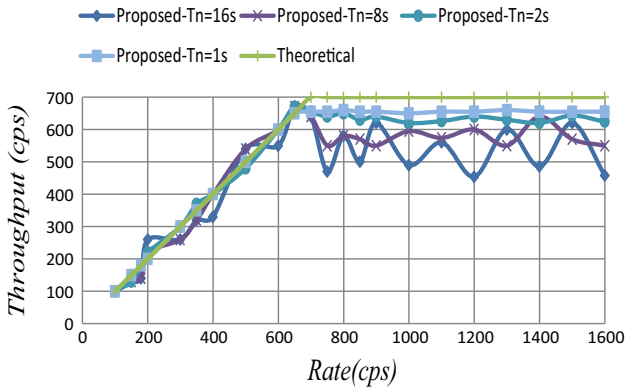

Fig. 8 Effect of different T_ns on the throughput of the proposed method

6.7 Effect of network and link latency

Network and link latency are negligible in previous tests; therefore, in such a state, message loss or fail to the server will not happen. To evaluate the efficiency of our mechanism in the case with a negligible amount of network (link) latency, a series of tests is performed in the networks with link latency, the results of which are presented below.

In order to simulate network latency and packet loss, netem network simulator [33], which is a part of the core of Fedora 12 distribution, is used.

Figure 9a compares throughput of the proposed algorithm with WIN-DISC for different network latencies from zero to 150 ms. In WIN-DISC, the downstream server calculates the window size at the end of each control discrete interval ($T_c = 200$ ms), refers to it, and then determines the number of new sessions which can admit for the following control interval.

As expected, increasing network latency decreases throughput of all the algorithms. This issue is caused by the increase in the number of re-transmissions, which occurs when the process of call-making is prolonged. In other words, more messages per call sequence are resulted and consequently the throughput is decreased. Nonetheless, throughput of the proposed algorithm is decreased from 655 cps to 550 cps, which is higher than that of WIN-DISC. Low throughput of WIN-DISC is related to the explicit nature of the used feedback; it is known that by increasing the latency of feedback loop, throughput is decreased and stability problems are emerged. Noted that, in theory, in spite of the high network latency, maximum throughput can be achieved by removing all the re-transmissions. As shown in Fig. 9a entitled "Theoretical" curve, maximum theoretical throughput in a network with the latency of 150 ms is 690 cps.

Figure 9b shows the sensitivity of the throughput of our algorithms to R_{th} when it is overloaded by 800 cps for different network latencies (D_{link}). Higher values of R_{th} lead to slower reactions and generally decrease the throughput by

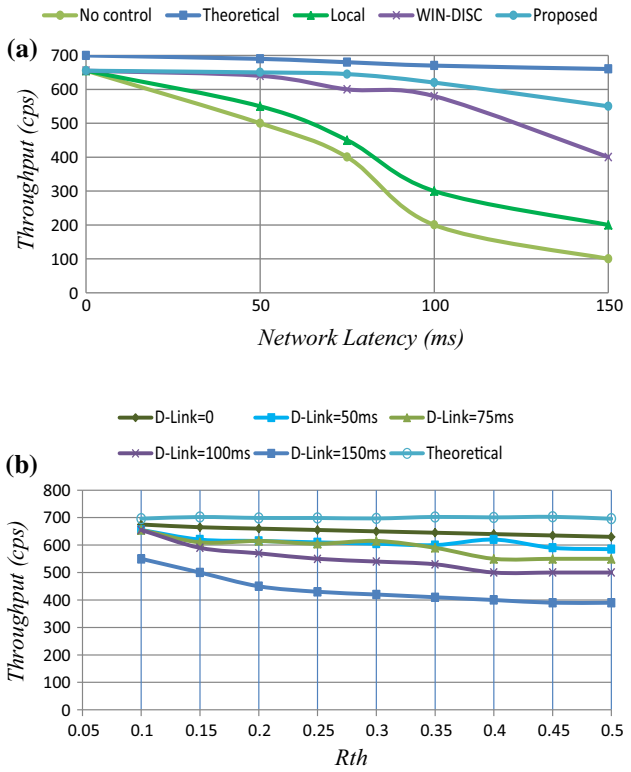

Fig. 9 Analysis of network and link latency. **a** Effect of network latency on throughput. **b** Effect of R_{th} and latency on the throughput of the proposed algorithm

Table 3 Results of throughput with latency for trapezoid testbed using the proposed algorithm (capacity of the downstream proxy is almost 700 cps)

Load (cps)	600	800	900	1400
D-link = 0	600	655	650	645
D-link = 50 ms	600	650	620	620
D-link = 100 ms	600	610	605	590
1%Packet loss	570	630	630	625

5–15%, depending on the network latency. However, considering very low values for R_{th} (e.g., 0.1) results in increased false alarms of overload (when the downstream server is working in a normal condition). Therefore, as mentioned in the article, a conservative value should be selected for R_{th}.

In addition, Table 3 demonstrates throughput under different conditions. The second row is related to lack of network latency. Different input loads (600, 800, 900, and 1400 cps) are tested: one is a normal load (600 cps), one is slightly more than the downstream server capacity (800 cps), next is a fairly extreme overload (900 cps), and the last is an extreme overload (1400 cps). It is obvious that, in the absence of any network latency, implementing the proposed algorithm results in almost maximum throughput. Nonetheless, throughput is decreased by increasing the network latency. Such an important point is in agreement with our results in Fig. 9. Additionally, the rate of packet loss is defined as 1% and the throughput is decreased by 5%; this state is given in the last row of the table and is because packet loss results in message re-transmission and increases the latency of transaction completion, which consequently decreases the window size. Therefore, the proposed algorithm is tested in different conditions including network latency and non-zero rate of packet loss and shows high throughput.

6.8 Fairness analysis

We use the Edge-Core topology, which is illustrated in Fig. 6b to study the performance of our proposed overload control algorithm when used by multiple upstream proxies. In testbed, two upstream edge proxies are considered, and each forwarding calls from a group of UAs to a single core proxy causing it overload. We also investigate the fairness of our overload control algorithm through monitoring the throughput perceived by each upstream edge proxy during overload.

In order to provide fairness during overload, we require that the capacity of the overloaded downstream proxy be equally split between all upstream (edge) proxies. Note that with proposed mechanism, the downstream server does not need to know about the number of upstream servers connected to it and also does not generate any explicit feedback. Our argument is that due to the use of an AIMD flow control regime, our proposed mechanism should be fair.

Indeed, Fig. 10a verifies this claim. In this figure, caller groups generate call requests of rate 650 cps each starting 100 s after the previous group. Network latency is set at zero. It could be seen clearly from the figure that all upstream servers get roughly the same throughput, which obviously decreases as more caller groups start sending requests. Note that the total throughput of the core proxy is maxed out at its capacity (i.e., 700 cps). From 0th to 100th s, only one upstream proxy is sending at 650 cps so it gets the entire capacity. From 100th to 200th s, there are two operating proxies and each receiving roughly equal throughput of 352 and 348 cps. At 200th s, the second proxy stops sending requests and the other regains the extra capacity.

Table 4 also shows average throughput of each caller group within different intervals and calculates Jain's fairness index for each of them. When all values are equal to each other, value of fairness index is unity. By the emergence of more variations between the values, this index tends to zero.

Next, we consider an MIMD (Multiplicative Increase Multiplicative Decrease) regime for window adaptation. This is simply resulted by increasing/decreasing the window size by a constant each time a call is established within/exceeding an acceptable delay bound. Indeed, a similar approach is proposed in [27]. Figure 10b demonstrations the throughput for such an MIMD regime when the increment and decrement

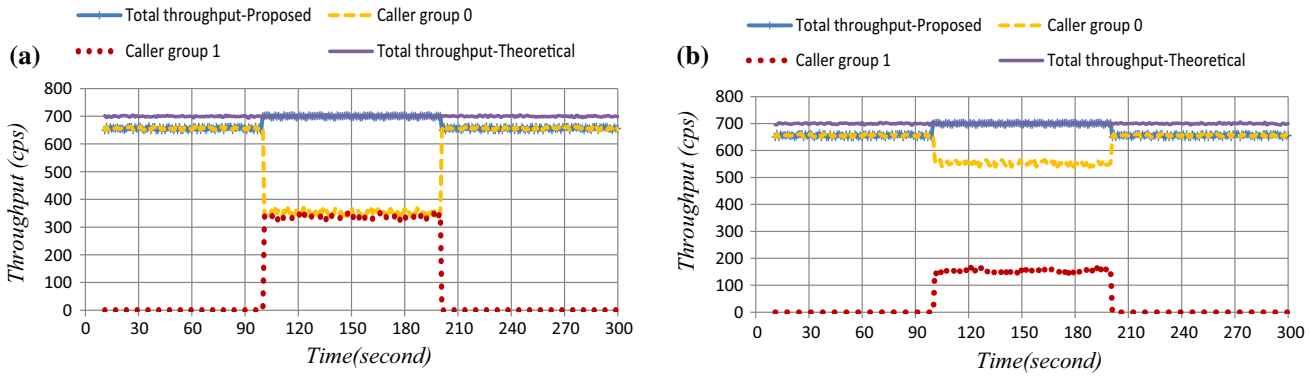

Fig. 10 Analysis of fairness. **a** Fairness analysis of proposed method. **b** Studying fairness under MIMD regime

Table 4 Fairness of our proposed mechanism

Time	0–100	100–200	200–300
Throughput	657	359	659
Jain's fairness index	1	0.9998	1

Table 5 Comparing fairness index

Time	0–100	100–200	200–300
Jain's fairness index (MIMD)	1	0.6784	1
Jain's fairness index (AIMD)	1	0.9998	1

rates are set to 0.1 and 0.5, respectively (as proposed in [27]). MIMD regime when two proxies are in sending mode treats very discriminatory. When two proxies transmit requests, fairness index of MIMD method is very low and equal to 0.6784 (see Table 5).

7 Conclusion and future work

The studies accomplished in this paper show that SIP protocol is inefficient in facing with overload. Therefore, when call request rate increases, the delay of call establishment increases suddenly, proxy's throughput falls, and consequently retransmission rates and unsuccessful calls increase.

In this paper, window-based overload control method is developed, implemented, and tested on a real platform. Also, the efficiency of SIP proxy in case of overload is studied by using such a distributed overload control method, which is developed on Asterisk. The proposed algorithm does not need any explicit feedback and uses number of confirmation messages to detect overload. Also the proposed method can change the maximum window size dynamically. Studying the charts of throughput, delay, and retransmission rate of `INVITE` and `BYE` messages shows that the algorithm maintains the throughput at up and be fair. Also, the proposed algorithm has a self-clocking nature because of using implicit feedback. The benefit of self-clocking nature is increased stability. In addition, the proposed method is simpler than other distributed methods which have been introduced for solving server to server overload problem in terms of implementation and application. Since, it does not cause any interference in the operation of downstream proxy. Even downstream proxy does not require to monitor its resources and detect its imminent overload condition or reject extra calls. Furthermore, no extra feedback exchanges between the adjacent proxies. The proposed approach is a proactive method, because it tries to identify and prevent overload before proxy queue is filled; messages are overflowed, and thus proxy resources are saturated.

As future work, we intend to study more advanced window update strategies and check more sophisticated overload detection functions. We also plan to implement our mechanism using the Kamailio proxy and the extended Edge-Core topology. Moreover, an optimization model for achieve optimal W_{max}, and sensitivity analysis of the SIP proxies are also underway.

References

1. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., et al. (2002). *Sip: session initiation protocol (rfc 3261)*. Rep: Tech.
2. Jiang, H., Iyengar, A., Nahum, E., Segmuller, W., Tantawi, A. N., & Wright, C. P. (2012). Design, implementation, and performance of a load balancer for sip server clusters. *IEEE/ACM Transactions on Networking (TON)*, *20*(4), 1190–1202.
3. Ohta, M. (2006). Overload protection in a sip signaling network. In *International Conference on Internet Surveillance and Protection (ICISP&# 146; 06)* (pp. 11–11), IEEE.
4. Shen, C., & Schulzrinne, H. (2010). On tcp-based sip server overload control. In *Principles, Systems and Applications of IP Telecommunications* (pp. 71–83), ACM.
5. Hong, Y., Huang, C., & Yan, J. (2010) Mitigating sip overload using a control-theoretic approach. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE* (pp. 1–5), IEEE.
6. Ohta, M. (2008). Performance comparisons of transport protocols for session initiation protocol signaling. In *Telecommunication*

networking workshop on QoS in multiservice IP networks, 2008. IT-NEWS 2008. 4th International (pp. 148–153), IEEE.
7. Abaev, P., Pechinkin, A., & Razumchik, R. (2012). On analytical model for optimal sip server hop-by-hop overload control. In *2012 4th International Congress on ultra modern telecommunications and control systems and workshops (ICUMT)* (pp. 286–291), IEEE.
8. Hong, Y., Huang, C., & Yan, J. (2011). Modeling and simulation of sip tandem server with finite buffer. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, *21*(2), 11.
9. Mishra, G., Dharmaraja, S., & Kar, S. (2016). Reducing session establishment delay using timed out packets in sip signaling network. *International Journal of Communication Systems*, *29*(2), 262–276.
10. Montazerolghaem, A., Shekofteh, S., Yaghmaee, M. H., & Naghibzadeh, M. (2015). A load scheduler for sip proxy servers: Design, implementation and evaluation of a history weighted window approach. *International Journal of Communication Systems*. *30*(3), e2980. doi:10.1002/dac.2980.
11. Garroppo, R. G., Giordano, S., Niccolini, S., & Spagna, S. (2011). A prediction-based overload control algorithm for sip servers. *IEEE Transactions on Network and Service Management*, *8*(1), 39–51.
12. De Cicco, L., Cofano, G., & Mascolo, S. (2015). Local sip overload control: Controller design and optimization by extremum seeking. *IEEE Transactions on Control of Network Systems*, *2*(3), 267–277.
13. Hilt, V., & Widjaja, I. (2008). Controlling overload in networks of sip servers. In *IEEE international conference on network protocols, 2008. ICNP 2008* (pp. 83–93), IEEE.
14. Noel, E. C., & Johnson, C. R. (2007). Initial simulation results that analyze sip based voip networks under overload. In *Managing traffic performance in converged networks* (pp. 54–64). Springer. doi:10.1007/978-3-540-72990-7_9.
15. Montagna, S., & Pignolo, M. (2008). Performance evaluation of load control techniques in sip signaling servers. In *Third international conference on systems, 2008. ICONS 08* (pp. 51–56), IEEE.
16. Ohta, M. (2009). Overload control in a sip signaling network. *International Journal of Electrical and Electronics Engineering*, *3*(2), 87–92.
17. Liao, J., Wang, J., Li, T., Wang, J., Wang, J., & Zhu, X. (2012). A distributed end-to-end overload control mechanism for networks of sip servers. *Computer Networks*, *56*(12), 2847–2868.
18. Wang, J., Liao, J., Li, T., Wang, J., Wang, J., & Qi, Q. (2014). Probe-based end-to-end overload control for networks of sip servers. *Journal of Network and Computer Applications*, *41*, 114–125.
19. Hilt, V., Noel, E., Shen, C., & Abdelal, A. (2011). *Design considerations for session initiation protocol (sip) overload control (rfc6357)*. Rep: Tech.
20. Shen, C., Schulzrinne, H., & Nahum, E. (2008). Session initiation protocol (sip) server overload control: Design and evaluation. In *Principles, systems and applications of IP telecommunications. Services and security for next generation networks* (pp. 149–173). Springer. doi:10.1007/978-3-540-89054-6_8.
21. Sun, J., Yu, H., & Zheng, W. (2008). Flow management with service differentiation for sip application servers. In *The third ChinaGrid annual conference (chinagrid 2008)* (pp. 272–277), IEEE.
22. Jahanbakhsh, M., Azhari, S. V., & Nemati, H. (2017). Lyapunov stability of sip systems and its application to overload control. *Computer Communications, 103*, 1–17. doi:10.1016/j.comcom.2017.01.014.
23. Yavas, D. Y., Hokelek, I., & Gunsel, B. (2016). Modeling of priority-based request scheduling mechanism for finite buffer sip servers. In *Proceedings of the 11th international conference on Queueing theory and network applications* (p. 6), ACM.
24. Montazerolghaem, A., Moghaddam, M. H. Y., & Tashtarian, F. (2015). Overload control in sip networks: A heuristic approach based on mathematical optimization. In *Global Communications Conference (GLOBECOM), 2015 IEEE* (pp. 1–6), IEEE.
25. Montazerolghaem, A., Yaghmaee, M. H., Leon-Garcia, A., Naghibzadeh, M., & Tashtarian, F. (2016). A load-balanced call admission controller for ims cloud computing. *IEEE Transactions on Network and Service Management*, *13*(4), 806–822.
26. Khazaei, M., & Mozayani, N. (2016). A dynamic distributed overload control mechanism in sip networks with holonic multi-agent systems. *Telecommunication Systems*, *63*(3), 437–455.
27. Noel, E. & Johnson, C. R. (2009). Novel overload controls for sip networks. In *Teletraffic Congress, 2009. ITC 21 2009. 21st International* (pp. 1–8), IEEE.
28. Low, S. H., Paganini, F., & Doyle, J. C. (2002). Internet congestion control. *IEEE Control Systems*, *22*(1), 28–43.
29. Chiu, D.-M., & Jain, R. (1989). Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, *17*(1), 1–14.
30. http://www.asterisk.org/.
31. Qin, D. (2011). Research on the performance of asterisk-based media gateway. In *2011 Fourth international symposium on knowledge acquisition and modeling (KAM)* (pp. 347–349), IEEE.
32. Malas, D., & Morton, A. (2011). *Basic telephony sip end-to-end performance metrics (rfc6076)*. Rep: Tech.
33. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem. *Tech. Rep.*

Ahmadreza Montazerolghaem received the B.Sc. degree in Information Technology from the computer department, Sadjad University of Technology and M.Sc. degree in computer engineering from the computer department, Ferdowsi University of Mashhad (FUM), Iran, in 2010 and 2013, respectively. Currently, he is a Ph.D. candidate in computer engineering at computer department, FUM. He is an IEEE Student member and a member of IP-PBX type approval lab in FUM. He is also a member of National Elites Foundation (Society of prominent students of the country). His research interests are in Software Defined Networking, Network Function Virtualization, Voice over IP, and Optimization.

M. H. Yaghmaee Moghaddam received his B.S. degree in communication engineering from Sharif University of Technology, Tehran, Iran in 1993, and M.S. degree in communication engineering from Tehran Polytechnic (Amirkabir) University of Technology in 1995. He received his Ph.D. degree in communication engineering from Tehran Polytechnic (Amirkabir) University of Technology in 2000. He has been a computer network engineer with several networking projects in Iran Telecommunication Research Center (ITRC) since 1992. November 1998 to July1999, he was with Network Technology Group (NTG), C&C Media research labs., NEC corporation, Tokyo, Japan, as visiting research scholar. September 2007 to August 2008, he

was with the Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, USA as the visiting associate professor. July 2015 to September 2016, he was with the electrical and computer engineering department of the University of Toronto (UoT) as the visiting professor. Currently, he is a full professor at the Computer Engineering Department, Ferdowsi University of Mashhad (FUM). His research interests are in Smart Grid, Computer and Communication Networks, Quality of Services (QoS), Software Defined Networking (SDN) and Network Function Virtualization (NFV). He is an IEEE Senior member and head of the IP-PBX type approval lab in the Ferdowsi University of Mashhad. He is the author of some books on Smart Grid, TCP/IP and Smart City in Persian language.