IJICTR International Journal of Information & Communication Technology Research ITRC

Volume 6- Number 4- Autumn 2014 (13-26)

An Overload Window Control Method Based on Fuzzy Logic to Improve SIP Performance

Ahmad Reza Montazerolghaem
Department of Computer Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
Ahmadreza.montazerolghaem@stu.um.ac.ir

Mohammad Hossein Yaghmaee Moghaddam
Department of Computer Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
hyaghmae@um.ac.ir

Received: March 3, 2013-Accepted: November 7, 2014

***Abstract*—Having facilities such as being in text form, having end-to-end connection establishment and being independent from type of transmitted data, SIP protocol is a good choice for signaling protocol in order to set up a connection between two users of an IP network. However, utilization of SIP protocol in a wide range of applications has made various vulnerabilities in this protocol, amongst which overload could make serious problems in SIP servers. An SIP is overloaded when it does not have sufficient resources (majorly CPU processing power and memory) to process all messages. In this paper, attempts were made to improve window-based overload control in RFC 6537. In window-based overload control method, a window is used to limit the number of messages that are sent to an overloaded SIP proxy simultaneously. In this paper, first, fuzzy logic was used to regulate accurate size of window and then it was developed, implemented and evaluated on an Asterisk open-source proxy. Implementation results showed that this method could practically maintain throughput under overload conditions, dynamically change the maximum window size, and also fairly treat among various upstream servers.**

Keywords- SIP; Overload control; Window based; Asterisk proxy; Fuzzy logic

I. INTRODUCTION

SIP protocol is the signaling protocol in application layer which is used to start, manage and finish the meeting between two or more applications. Major components of a SIP network are user agents, server proxies and registrars. User agent is the terminal component in SIP session. Figure 1 illustrates connection establishment between two user agents in a case in which middle proxies are statefully configured. Before establishing a session between callers (User Agents A in Figure 1) and callee (User Agents B in Figure 1), the information required for establishing a session through SIP signaling is exchanged. SIP signaling is performed by sending requests and responses via SIP proxy servers. The routes of requests and responses are independent from routes of the established sessions. Signaling of SIP takes place between the neighbors, as shown by 1, 2 and 3 in Figure 1. Resolving the SIP URI, each SIP proxy server performs routing of SIP requests and responses. The proxy task is to route and redeploy signaling between user agents.

SIP server is an application one. The overload problem in SIP server is distinguished from ones in other HTTP servers for at least three reasons: first, the messages of SIP meeting pass several SIP proxy servers to reach the destination, which could itself make overload between two SIP proxy servers. Second, SIP has several retransmit timers which are used for dealing with packet loss, especially when the packet is sent via UDP transmission protocol, and this could lead to overload on SIP proxy server. Third, SIP requests are used as real time session signaling; so, they have high sensitivity. Overload in SIP-based networks occurs when the server does not have necessary sources (for instance, CPU processing

power and memory [1]) for answering every received call. Reviews conducted in overloaded SIP proxy server have shown that increasing request rate results in sudden increase in delay in establishing connection and dropping proxy throughput and therefore increase in unsuccessful call rates. Therefore, the aim in overload control in SIP is to maintain the throughput of overloaded server near its capacity. Generally, there are two local and distributed methods for overload control. 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 [2]. But, request rejection mechanism, in order to finish meeting, imposes cost itself and, when server is overloaded, it is compelled to allocate fraction of sources to reject requests, which in turn decreases efficiency in SIP proxy server. In distributed method, upstream servers control load of downstream servers through rejecting requests and try to maintain it under their capacity.

In 2011, design considerations for a SIP overload control mechanism were discussed in the SOC workgroup. The resulted design, named RFC 6537, was standardized in the workgroup [31]. Five ways of distributed overload control were described in this standard. These methods used explicit feedback between SIP proxy servers:

- Rate-based overload control method
- Loss-based overload control method
- Window-based overload control method
- Signal-based overload control method
- On/Off overload control method

This study focused on window-based overload control method. The main idea of this method is to limit the number of output messages by controlling the window size. In other words, in window-based overload control approaches, a limit is applied on the maximum requests waiting for response in the proxy. The main issue of such methods is window size. In this paper, fuzzy logic was proposed to determine window size as accurately and dynamically as possible and then it was implemented and evaluated on Asterisk open-source proxy. Simulation results showed that the proposed method reached a higher throughput than a traditional overload control algorithm proposed in [29].

The rest of this paper is organized as follows: section II includes an overview of the SIP protocol and existing overload control methods. Section III includes SIP overload problems.

[1] In section IV we analyzed the effect of processing and memory resources on the operation of an overloaded SIP proxy.

[2] In order to understand the problem of overload in SIP servers more appropriately, in section III we implemented a simple local control mechanism and compared it with no-control case.

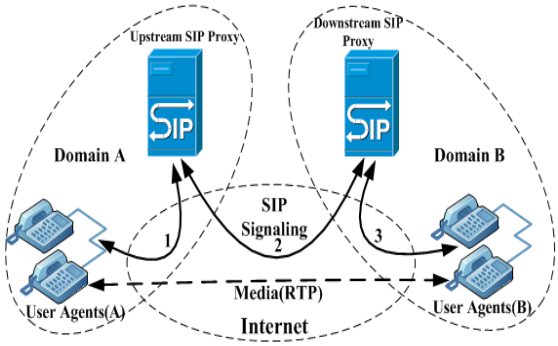

Fig. 1. Network configuration for SIP

To continue the discussion of last section, in Section IV, effect of processing as well as memory resources on the operation of an overloaded SIP proxy is studied. The results lead to the fuzzy approach. In Section V, details of the proposed overload control algorithm which is developed in this open source software is presented. In Section VI, the network topologies and configurations are presented. Section VII contains performance evaluation and experimental results. Finally, Section VIII concludes the paper and outlines future works.

II. BACKGROUND

A. *SIP Overview*

Figure 2 illustrates the typical SIP trapezoid topology and standard SIP voice call signaling 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 of a session starts. Returning a "100 Trying" response to the previous hop in the path confirms 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 reception of "200 OK", an "ACK" request is sent to the callee. After this three way handshake, the media session is independently established between the two parties. The session is then terminated when one party sends a "BYE" request and another responds with a "200 OK".

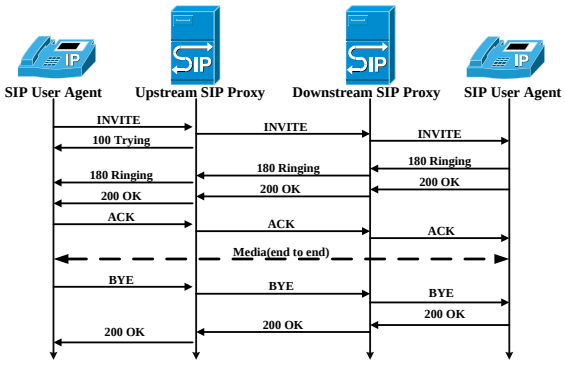

Fig. 2. Exchanged messages for establishing connection in SIP

B. SIP Proxy Server

SIP servers are applications that accept SIP requests and respond to them. An SIP server should not be confused with a user agent server or the client-server nature of the protocol, which is described in terms of clients (originators of requests) and servers (originators of responses to requests). An SIP server is a different type of entity; the types of SIP servers discussed in this section are logical entities. Actual SIP server implementations may contain a number of server types or may operate as a different type of server under different conditions. Because servers provide services and features to user agents, they must support both TCP and UDP for transport.

An SIP proxy server receives a SIP request from a user agent or another proxy and acts on behalf of the user agent in forwarding or responding to the request.

1) Introducing Asterisk Proxy

Asterisk is the most popular open source VOIP telephone system in the world, based on which many available IPPBXs are currently produced. Asterisk is based on C programming language and could be loaded in various operating systems such as Linux NetBSD, UNIX, Solaris and Mac OSX. In addition, it is observed that some versions of Asterisk are installable and operable in Windows platform. Although Asterisk services could be operated using common computers and servers and through calculating power of system (CPU/RAM) on the basis of users multiplicity, popularity of Asterisk and diversity of its services have made producers utilize most of combined platforms of Linux and Asterisk in producing integrated connection equipment at various scales. The minimum system requirements for installing Asterisk are a 500 MHz Pentium computer with 512 MB RAM and 20 GB empty hard space [5]. This software uses UDP and TCP transmission protocols to receive and send SIP messages and, while receiving SIP messages, it first intercepts the message and then decides whether to reply to it or forward it to the next destination [5, 6]. In this paper, UDP transmission protocol was used to receive and send SIP messages. Asterisk uses several Worker Processes to receive and send SIP messages and every Worker Process receives messages individually and makes decisions about it. In order to process a SIP message, Worker Process should make a connection between the message and transaction; the message could be related to a transaction which already exists or it may be a new message for which a transaction is created; these transactions are saved in shared memory of Worker Processes. There is no guarantee that a Worker Process manages every message related to the same transaction and it is probable for one transaction to be managed by several Worker Processes. When a message is sent, a new timer is created and added to the list. A process manages this list, checks timers and, until the timer finishes and no replication is received for that message, resends the message by accessing the appropriate transaction [6].

C. SIP Client Workload Generator

In this work, Spirent Abacus 5000 device was used to create traffic with different transmission rates and various distributions. This device is used for different tests including interoperability, performance, scalability and testing audio and video qualities on IP networks. This production is able to test efficiency and extensibility of the tested proxy by producing hundreds to thousands of calls. Asterisk software and Spirent Abacus 5000 tester device were used for implementing proxy servers and user agents, respectively. According to Figure 1, signaling load was produced by two UAS and UAC user agents, role of both of which was played by Spirent Abacus 5000 device (see Figure 4).

Fig. 3. Spirent Abacus 5000 device

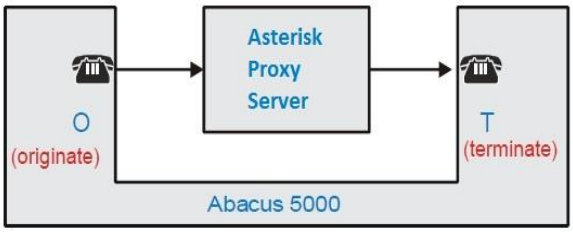

Fig. 4. Role of Spirent Abacus 5000 device

D. Related Works

Many researches about efficiency of SIP proxy server have been done. In [7], overload control methods were dealt with in SIP proxy server and OPNET software was used for measuring throughput. In [10] and [11], SIP was practically implemented along with TCP and UDP transmission protocol and OpenSER was used to obtain efficiency results. [12] and [13] mentioned window-based distributed method and combination of signal and window-based method, respectively. SIPstone [14] is a series of benchmark, in which various criteria are proposed for evaluating proxy server powers and redirecting server and registrar in answering SIP requests. In [15], another benchmark was presented for measuring effect of operating system, hardware configuration, database and selected transmission layer on SIP efficiency. In [16], practical experiments were accomplished on four types of proxy implementation which were different in both thread management and memory allocation methods. The results of these experiments showed that the effective parameters in proxy efficiency could be classified to two parts: parameters related to protocol such as message length, length variability and irregularity of excess load, and parameters related to type of server implementation; e.g. how to allocate sources of operating system to transactions. Also in

[17], similar studies on the effect of operating system and type of proxy implementation on SIP efficiency were done. In [18], efficiency of signaling SIP in establishing VOIP connections was examined using JAIN SIP API and by considering effect of call duration and call rate on the delay of connection establishment between two end-to-end user agents. In [19], effect of delay of user's answer on SIP server's efficiency was analyzed by introducing a tool called SIPperformer. In [20], issues such as importance of security cost in configuration which used authentication and also effect of stateful or stateless proxy and protocol type of transmission layer on proxy's efficiency were studied by placing only one proxy between the user agents. A group of studies has been also concentrated on the evaluation of SIP efficiency under various access technologies. For example in [21], effect of transmission delay and packet loss on W-CDMA link was surveyed using a proxy. In [22], the delay in signaling SIP in establishing IMS meetings was evaluated for different WiMax channels with different speeds. Also, compression techniques for SIP messages were used in order to reduce volume of SIP packets along with their transmission delay. Selection of transmission layer protocol is also influential in the efficiency of signaling SIP. In [23], various options in selection of transmission layer protocol were qualitatively surveyed for SIP. In [24], effect of deploying various transmission layer protocols, especially effect of window control mechanism in TCP on throughput and delay in connection establishment, was evaluated. In [25], it was shown that, despite the general perception in which more common utilization of UDP than TCP was considered on account of the low processing excess load in the former, it was probable that unfavorable efficiency in TCP utilization was due to implementation manner of proxy. A wide variety of local overload control methods differing mainly in the rejection policy and overload detection criteria has been introduced and evaluated in the literature. For instance, Queue-length-based algorithms were proposed in [7, 8, 26 and 27]. Occupancy-based algorithms, namely OCC, which used CPU utilization as a trigger for rejecting calls, were also proposed in [7, 26]. In addition, effect of priority-based queuing and transport protocol on performance of SIP signalling was analyzed in [28, 13], respectively. In [7], it was demonstrated that, for an overload of 100%, system throughput dropped by 25%, from 200 to 160 cps. This throughput degradation could be considered the cost of running the overload control algorithm by CPU. This throughput penalty could be alleviated in many cases where cause of overload condition is upstream SIP servers. This circumstance is called "server-server" overload. Local rejection mechanisms are coupled with "distributed" OC, in which upstream servers control load of the downstream SIP server, keeping its load as close as possible to its capacity. Generally, the overloaded server monitors its resources and sends an explicit feedback to all upstream servers with the purpose of informing them from the overload. It also possibly communicates the amount of load that can accept. Accordingly, the upstream servers lower their forwarding rate. Shen et al. [29] proposed three window-based distributed OC methods in which downstream server dynamically estimated its capacity and generated a feedback, indicating the number of currently available window slots.

While local overload control methods suffer from non-negligible rejection cost, most proposed distributed algorithms increase complexity of the overloaded server by requiring load monitor and calculating an explicit feedback. Another drawback of using explicit feedback is delay of the feedback in reaching upstream servers, which may result in instability or at least performance fluctuations of the algorithm. In the context of Internet congestion control algorithms, this is a well-known phenomenon [30]. Now, the main deficiencies of current overload control schemes could be summed up. First, reliance only on local rejection could lead to throughput degradation. Second, overload detection and possibly feedback generation cost CPU time and impact throughput if accomplished in the overloaded server. An overload condition is a complicated situation which may happen due to many reasons. It is believed that cost of generating feedback information is mainly because of local load estimation.

III. SIP OVERLOAD PROBLEMS

SIP uses its own reliability mechanism, which is using a large set of re-transmission timers, especially when used on an unreliable transport protocol such as UDP. For instance, Timer A is responsible for scheduling INVITE re-transmissions and starts with an initial value of typically T1 = 500 ms and doubles when being expired. After waiting for 64×T1 = 32 s, SIP will stop re-transmission and declare call failure. This mechanism is useful in the case of having unreliable links; but, in overload conditions, it is a major cause of performance degradation. During the overload, messages that arrive at the overloaded server either get dropped or incur large 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 re-transmit some parts of the requests it has already forwarded to the UAS. Therefore, the actual server load increases in a regenerative way so that the call fails. The curve labeled ''without overload control'' in Figure 5 shows the dramatic decrease in server throughput. Here, capacity of SIP server equals 700 calls per second (cps). When load increases beyond this limit, the server becomes overloaded and congestion collapse occurs. A similar behaviour was reported in [7, 8] and many other references. Also, SIP server performance on a real test-bed was evaluated and similar results were obtained, as explained in the following sections.

In order to survey efficiency of SIP proxy when facing overload, the simplest traditional topologies were used [9, 7]. In this topology, platform of which is illustrated in Figure 8, a central proxy inquires every connection to be made between parties. This model is usually used for studying destructive effects of overload on proxy. If the number of users that start to

call in a short duration of time exceeds proxy's capacity, it is faced with overload. The overload with which the proxy in this topology is faced is in the form of client-server and the methods that are presented to get away from it are local inevitable.

Conversation production rate starts from the amount of as low as 100 calls per second and goes up to as high as 1200 calls per second, having Poisson distribution. In this case, only one proxy considers call routing.

Figure 6 shows average call establishment time versus call request rate; call establishment time is defined as the interval between sending the first "INVITE" from meeting starter to time of receiving OK message. As can be seen in this figure, the average call establishment time before received call rate reaching about 700 calls per second was inconsiderable and lower than ten ms. So, capacity of the proxy was 700 calls per second. As the received call rate approached to proxy's capacity (700 cps), the average delay went beyond 10 s.

The reason why call process procedure gets slow in proxy is that the amount of received call rate is beyond the processing capacity of the proxy and therefore proxy's sources are conjugated for analyzing and dissecting new call requests. Besides, the flow of requests of new calls eventuates to overflowing of received queue and losing the packets related to ongoing calls. As a result, users whose requests have remained unanswered or the progression of whose call establishment has remained incomplete proceed to resend their messages. On the other hand, the proxy itself spends a part of its capacity on resending requests which have been already sent because it has not seen some of the received reply packets which are missed through overflowing of queue or have not been checked yet because the proxy is busy. This procedure continues until proxy's throughput falls to near zero. The diagram of proxy's throughput in terms of call request rate which is shown in Figure 5 clearly proves this issue. For example, for rates of higher than 800 cps, the magnitude of throughput is practically negligible. Note that, this diagram shows rate of calls that have started successfully in unit of time.

Figure 7 shows retransmission rate for "INVITE" requests made on the user side. As expected, no request is resent before the received call rate reaches proxy's capacity. But, upon reaching the received call rate to proxy's capacity, retransmission rate abruptly increases and considerably intensifies the load imposed to proxy.

The amount of imposed load is calculated as the ratio of processing cost of each call to its expected processing cost in the case of non-overload. The cost of each call, which includes the required time for dissection of its related packets, regulating timers and creating and eradicating state for its related transactions, increases as load rate increases because, in the case of overload, as a result of resending phenomena, the number of packets which is created and dissected per call along with the number of timers that are regulated and reset increases.

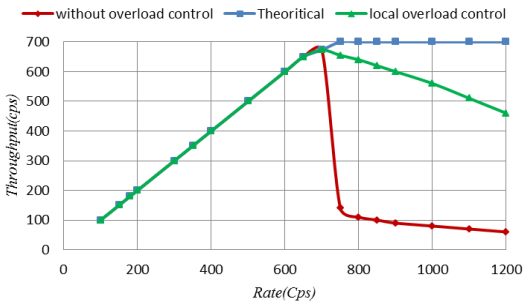

Fig. 5. Proxy's throughput in the case of single proxy

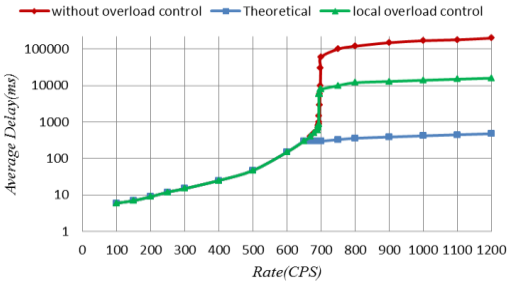

Fig. 6. Delay of call establishment in the case of single proxy

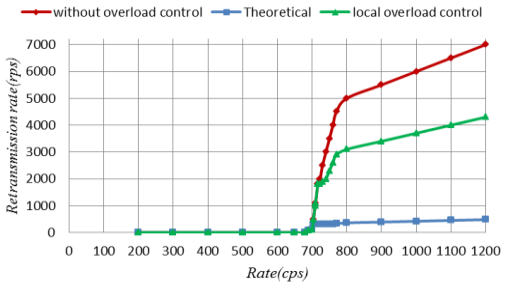

Fig. 7. INVITE retransmission rate in the case of single proxy

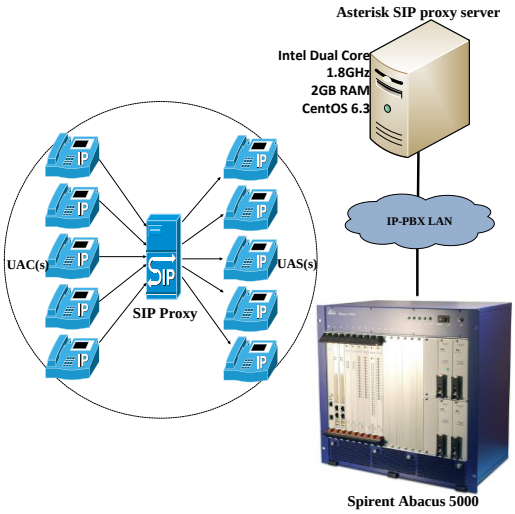

Fig. 8. Single-proxy topology and testbed setup

Although the number of packets that are received and dissected is related to size of queue as well, if the size of queue is small, many packets are missed and there is no need for them to be processed in proxy. Therefore, as cost of each call increases, the amount of load which should be processed each proxy is much more than the load rate that is imposed directly by excess calls. The reason of sudden fall in proxy's throughput in load rates of a little more than 700 cps is this amount, too. For example, if the cost in non-overload case for each call is 5 ms, in the case of overload, it may increase to 6 ms. So, imposed load to proxy is like being 1.2 times of the rate of its received calls.

The major aim of SIP overload control (OC) is to keep server throughput as close as possible to its capacity in the presence of overload. As can be seen in the above figures, the curve labeled ‘‘Theoretical’’ shows how an ideal OC scheme would work when server throughput is 700 cps. As was mentioned before, there are two ways to control overload: local and distributed. In the former, the control loop is internally implemented on the overloaded server; therefore, a SIP server starts rejecting additional requests whenever it gets close to its capacity limit. This is accomplished by sending a “503 Service Unavailable” message in response to an “INVITE” [4]. Under heavy overload conditions, the overloaded server will spend most of its resources on rejecting extra requests, which leads to throughput degradation. This can be clearly observed in the curve labeled ‘‘Local Overload control’’ in Figure 5. The local overload mechanism in this paper will be shown below.

A queuing structure of the SIP server could be seen in Figure 9. As shown in this figure, the queue is a simple single queue. Every time a SIP message is arrived, it is placed in the queue and served with the first in first out (FIFO) procedure. Here, two different states may occur for the server: overload and underload. In order to detect an overload, two thresholds can be introduced: TH_{Low} and TH_{High}. If the occupied number of buffers in the queue exceeds the threshold THHigh, the SIP proxy server recognizes a congestion condition. After that, if the occupied number of buffers becomes less than THLow, the SIP proxy server recognizes that the congestion is removed. Whenever a packet arrives at the queue, first, average queue length of the overloaded server (Qavg) is calculated by function (1) and then the rejection probability of service, Prej, is calculated by function (2) such that the proxy will randomly reject messages when length of the occupied buffering queue reaches a certain threshold. When a message arrives at the proxy, it compares current Qavg with the aforementioned thresholds: if Qavg < THLow, the proxy accepts the message; if Qavg > THHigh, it rejects the message; if THLow < Qavg < THHigh, then Prej calculated by function (2) will determine whether to refuse the message and response by a 503 message.

$$Q_{avg}(n) = (1-W_q)\, Q_{avg}(n-1) + W_q\, Q(n) \quad (1)$$

$$P_{rej} = ((Q_{avg} - TH_{Low}) / (TH_{High} - TH_{Low})) \quad (2)$$

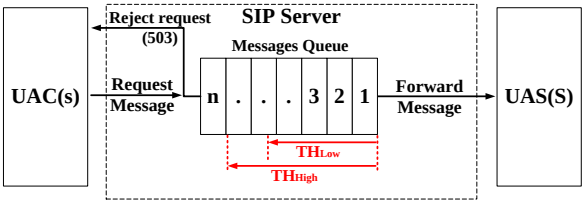

Fig. 9. Queuing structure and thresholds

where Wq is the queue weight. This allows for tuning contribution of the current queue size (Q (n)). The maximum and minimum thresholds for buffer length are set as THLow = 400, THHigh = 1000 messages. The average queue weight Wq is 0.1.

Figure 10 shows a message flow for load regulation purpose. Usually, a SIP proxy server returns the“100 Trying” response for “INVITEs”. As shown in the figure, the SIP proxy server returns “503” when congestion conditions occur. As mentioned before, according to RFC3261, when source SIP UA receives “300-699” response, it must stay in the state of starting Timer, which is called “A” here. In this state, the source SIP UA cannot send any new “INVITE” messages. The period which stays in this state is controlled using Timer A. As was said, the default value of Timer A is chosen as 32 s. Through regulating setting up new calls by Timer A, the offered load to the network can be reduced. It is expected then that the overload is temporarily removed. And, this is the main reason of relative and temporary success of local OC methods.

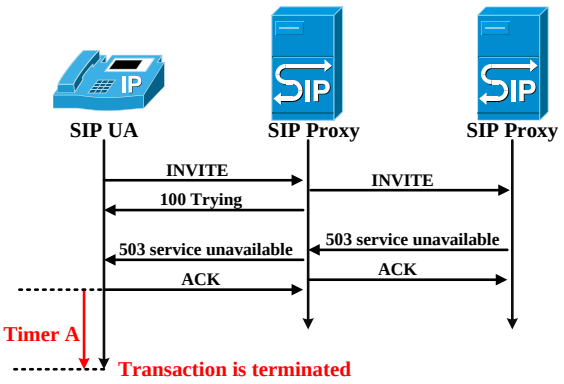

Fig. 10. Message flow for load regulation

IV. PROCESSING AND MEMORY RESOURCES OF A PROXY

As can be seen in Figure 11, proxy's queue is approximately empty before occurring overload since every message is drawn out of queue and processed upon reaching the proxy. Although, in overload conditions, many packets are consistently waiting in the queue to receive service, applying local overload control method could decrease memory usage. The diagram in Figure 12 shows CPU usage in proxy in terms of call request rate. In this diagram, the horizontal axis represents the amount of load received by proxy and vertical axis represents the average CPU utilization. These amounts are achieved by regular sampling (every second) of CPU engagement times and their averages. As was expected, in load rates lower than proxy's capacity, percentage of CPU

utilization is proportional to the amount of load to proxy's capacity. For example, when there is no overload control mechanism, in about 350 cps which is half of the proxy's capacity, utilization of proxy's CPU is approximately 0.5 as well; also, in a rate of 630 cps in which call establishment request rate is 90% of proxy's capacity (700 cps), utilization of proxy's CPU is 0.9. In loads greater than proxy's capacity which lead to overload, CPU utilization reaches 100%. Using local overload control mechanism, CPU utilization could be decreased.

In Figure 13, the processing resource used in proxy in "without overload control" case is illustrated. It could be seen in the figure that, as the rate increases, percentage of CPU utilization by Asterisk and MySQL which are responsible for processing SIP packets and managing users' database, respectively, increases as well. This procedure continues until the rate of about 700 cps, in which CPU efficiency reaches 100 percent. After that, as load increases, no additional processing resources are devoted to either of the procedures. Therefore, in rates of higher than this, call establishment delay increases dramatically.

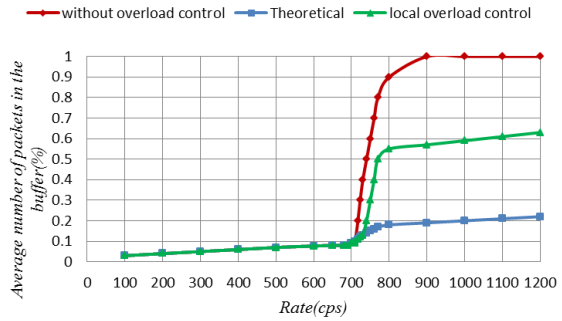

Fig. 11. Average queue packet count in the presence and absence of local overload control mechanism

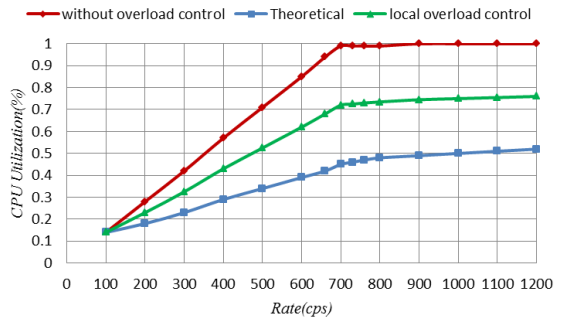

Fig. 12. Average CPU utilization in the presence and absence of local overload control mechanism

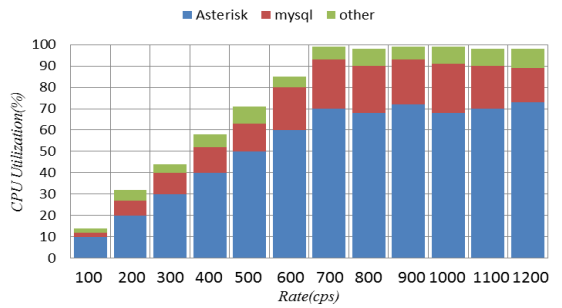

Fig. 13. Utilization percentage of proxy's CPU in case of not using any overload control method

Then, we study effect of limitation of memory and processor on efficiency of another proxy called OpenSER (With the same details). In these tests, 512 MB of memory is allocated to OpenSER and then the used memory has been monitored during test. The number of the sent, received and deleted messages from SIP buffer has been monitored.

In Figure 15, maximum rate of shared memory has been shown which an ascending function of call rate is naturally.

In dotted curve which is the authentication and stateful proxy, the use of memory has reached its highest limit in rate 700 cps. In this state, server parsing many demands for which it creates transaction but call setup time is very long due to shortage of processing sources and many of these contacts will fail. With increasing call establish rate, major part of messages is either "INVITE" or resending it. As a result, proxy is more involved in analysis of calls and operations relating to authentication and communication with database. Therefore, fewer messages are sent to transaction allocation and submission phase which results in reduction of the memory used by proxy.

In the stateful curve without authentication, proxy memory reaches saturation limit in rates of higher than 700 cps. In this case, the number of active transaction considerably increases (exceeding 10000) and because server has no enough memory for new contacts, it prevents new users from persisting on contact requests with error 500 which means internal error of server. When proxy memory is filled, the number of missing packets is also enhanced.

In this paper, we seek to prevent location of the proxy in this situation.

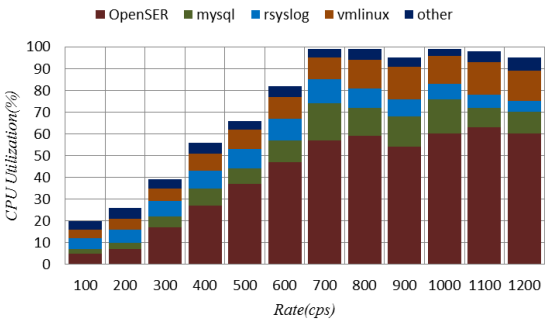

Fig. 14. Average CPU utilization (OpenSER SIP Proxy)

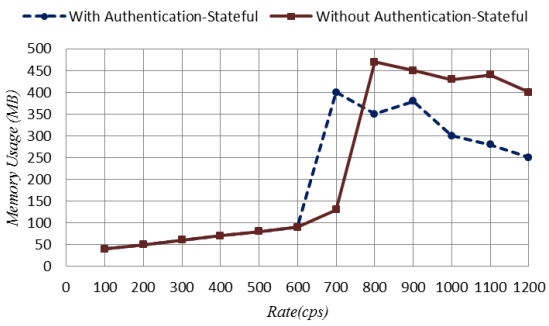

Fig. 15. Maximum memory usage (OpenSER SIP Proxy)

The recent figures show that proxy's efficiency changes due to two factors: the allotted memory and processing power of the processor on which the proxy runs. Both processor saturation and memory deficiency degrade proxy's efficiency dramatically. Note that, it is feasible to prevent proxy from accepting calls which are more than its capacity by limiting its allotted memory. Under these circumstances, proxy's processor never reaches the saturation limit and also, through sending a 500 message by the proxy, additional calls are not made. However, it is notable that this policy improves proxy's efficiency to some limited extent. In other words, as call rate increases, proxy's processor which is needful of parsing received messages in order to know their contents reaches saturation again. But, this happens under heavier loads.

V. WINDOW-BASED FUZZY OVERLOAD CONTROL MECHANISM

The main issue of window-based algorithms is the window length which is regulated by downstream proxy's feedback. It is feasible to prevent overload by limiting window length. Therefore, new calls are accepted only when there is an empty slot in the window. As mentioned before, it is feasible to efficiently limit the number of messages by using window size.

In this section, an effective fuzzy-based method is introduced for window-based overload control. In this method, fuzzy logic was used to solve the problems related to changes of window size. Fuzzy logic bore characteristics that made it appropriate equipment for solving such problems. In fuzzy logic, unsure data were received and processed although a sure and finite output was generated. Instead of request and response messages, this logic was used to determine overload window size accurately and dynamically. Considering the results achieved from analyzing SIP proxy's processing and memory resources in IV section, in order to initially diagnose occurrence of overload and then change the window size, instead of using request and response messages and comparing them, CPU and memory utilization of the overloaded proxy were used to prevent overload and react on time in the case of overload happening.

In this method, a fuzzy controller was contrived to dynamically change window size in upstream server. The input of this controller was average utilization of CPU and memory in downstream server (I_{cpu} and I_{mem}, respectively) and its output was rate of changes of window size in upstream server (ΔW) (see Figure 17).

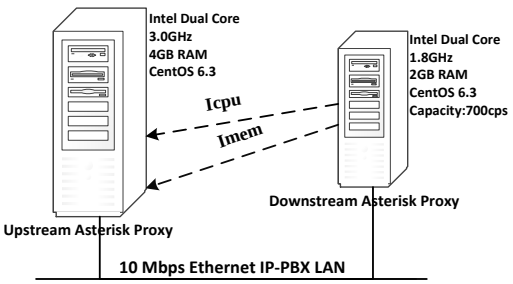

Fig. 16. General scheme

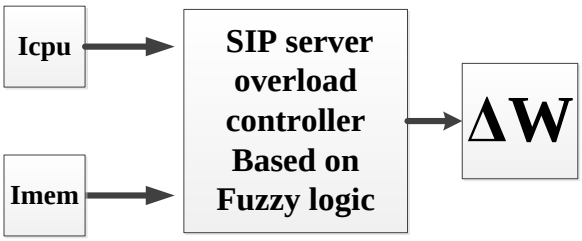

Fig. 17. Fuzzy controller

In this method, the window size control mechanism in upstream proxy was as follows:

1) Wmax = Winit
2) Calculating ΔW by fuzzy controller
3) $W_{max(t+1)}=W_{max(t)}+(\Delta W * W_{max(t)})$

On the basis of the results of performed experiments, the best range of changes for ΔW was [-0.6, 0.4], the membership function of which is stated below.

A. *Fuzzy Derivation System*

Fuzzy derivation is a method which interprets values of input vector and assigns a value to output vector using defined rules. In this paper, Mamdani approach was used as the fuzzy derivation method.

B. *Input and Output Membership Functions*

The proposed fuzzy system in the algorithm included two input and one output variables. Membership functions were determined using experimental experiences and trial-and-error procedure. The improvement of system efficiency dramatically depended on membership functions determined in this phase. Membership functions for input and output variables are as shown in the following figures.

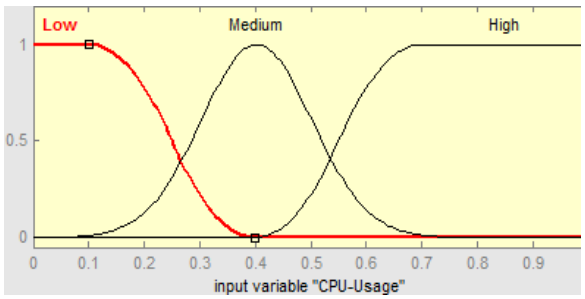

Fig. 18. Membership function of input variable Icpu

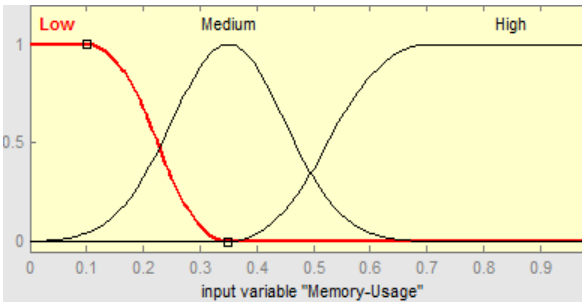

Fig. 19. Membership function of input variable Imem

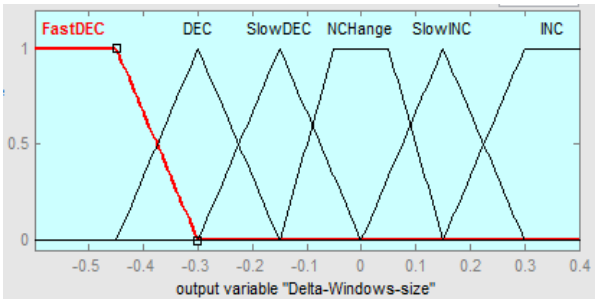

Fig. 20. Membership function of output variable

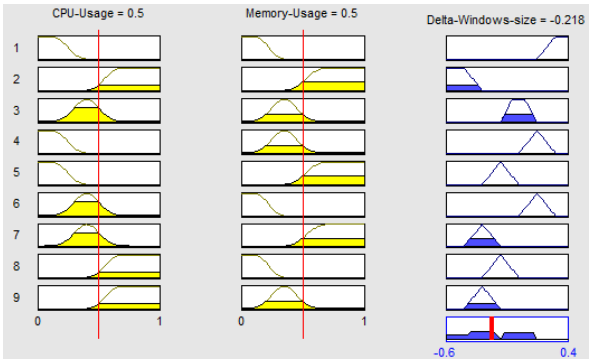

Fig. 21. Fuzzy system output for [0.5, 0.5]

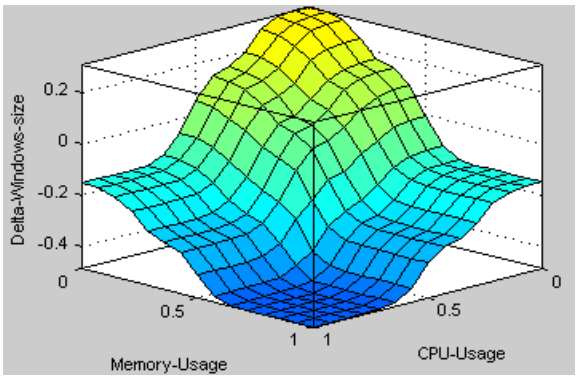

Fig. 22. Three-dimensional diagram of system's output versus both of its inputs

C. *Fuzzy Rules Base*

The next step of designing fuzzy systems is to design fuzzy rules base. Fuzzy rules base is like the core of fuzzy derivation engine. This base is a set of If-Then rules, each responsible for a part of derivation and decision making process. Note that, if rules base is extensive, then fuzzy system is complicated and its speed is degraded. The rules based used in the proposed algorithm is presented as follows.

1. If (CPU-Usage is Low) and (Memory-Usage is Low) Then (Delta-Windows-size is INC)
2. If (CPU-Usage is High) and (Memory-Usage is High) Then (Delta-Windows-size is FastDEC)
3. If (CPU-Usage is Medium) and (Memory-Usage is Medium) Then (Delta-Windows-size is NoCHange)
4. If (CPU-Usage is Low) and (Memory-Usage is Medium) Then (Delta-Windows-size is SlowINC)
5. If (CPU-Usage is Low) and (Memory-Usage is High) Then (Delta-Windows-size is SlowDEC)
6. If (CPU-Usage is Medium) and (Memory-Usage is Low) Then (Delta-Windows-size is SlowINC)
7. If (CPU-Usage is Medium) and (Memory-Usage is High) Then (Delta-Windows-size is DEC)
8. If (CPU-Usage is High) and (Memory-Usage is Low) Then (Delta-Windows-size is SlowDEC)
9. If (CPU-Usage is High) and (Memory-Usage is Medium) Then (Delta-Windows-size is DEC)

TABLE I. FUZZY RULES BASE

Icpu/Imem	Low	Medium	High
Low	INC	SlowINC	SlowDEC
Medium	SlowINC	NoCHange	DEC
High	SlowDEC	DEC	FastDEC

In order to evaluate rules, first of all, inputs are made fuzzy and then applied to the rules' premier section. In this system, AND fuzzy operator was used to derive a number representing the assessment of the rules' premier section. Then, the derived number was applied to the inferior section. Also, union operator was used to merge the results of applying fuzzy rules. Consequently, a central average de-fuzzier operator was used to derive a real output.

For example, as shown in Figure 21, for input values I_{cpu}=0.5 and I_{mem}=0.5, the proposed fuzzy system considered ΔW as -0.218, which represented that window size should decrease as -0.218*W_{max}.

Figure 22 shows output variation versus inputs as a three-dimensional diagram. It is clear that as Memory and CPU in downstream server were more involved, window size in upstream server decreased.

VI. NETWORK TOPOLOGIES , CONFIGURATIONS AND PRACTICAL CONSIDERATIONS

The SIP trapezoid, shown in Figure 23, was used as the basic network topology. In this topology, two proxies, namely, upstream and downstream, were used for handling outgoing and incoming calls, respectively. In order to easily study OC performance, the upstream proxy was made faster than the downstream. All the calls were originated from the clients of the upstream proxy and 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 with which the proxy in this topology is faced is in the form of server-server. In this form of overload, a limited number of upstream proxies send a huge volume of traffic to a downstream proxy and leads it to be faced with overload. This topology is applicable wherever any user gets service from their local service provider proxy. The capacities of upstream proxies are considered so that they do not face overload during experiments. By the way, for simplicity, only one upstream proxy is used.

Figure 24 shows the present test bed setup, which was composed of two Linux PCs connected over a 100Base-T Ethernet LAN. The faster PC functioned as the upstream proxy while the slower one was considered the downstream proxy with nominal capacity of approximately 700 cps. Upstream and downstream servers were configured in the transaction-stateful mode without any authentication.

Asterisk software and Spirent Abacus 5000 tester device were used for implementing proxy servers and user agents, respectively. The upstream server was a PC with INTEL Dual Core 3 GHZ processor and 4 GB memory and the downstream server was a PC with INTEL 1.8 GHz processor and 2 GB memory. Both servers used version 6.3 of Linux CentOS as their operating system. By modifying Asterisk code, the proposed mechanism on the upstream server was implemented. However, the downstream proxy was intact. Also, using MATLAB, the proposed fuzzy system was simulated.

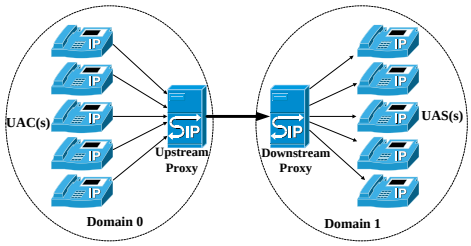

Fig. 23. Dual-proxy topology (trapezoid)

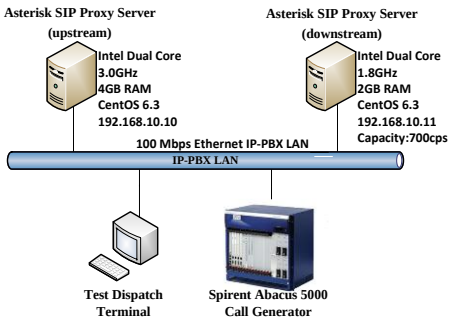

Fig. 24. Test bed setup for the trapezoid topology

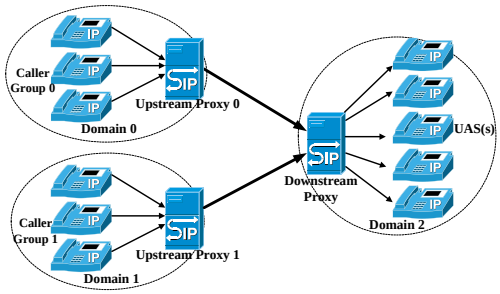

Fig. 25. Edge-core topology

The next network topology (Edge-Core) is depicted in Figure 25. This topology consists of a number of edge servers that communicate signaling messages through one core server, which was overloaded in the present experiments. This is a representative of the topology used in the IP Multimedia Subsystem proposed by 3GPP. In the current experiments, there were two upstream servers and a downstream one. All the edge servers were assumed and configured to be fast enough so that they were not overloaded. Using the Edge-Core topology, it was shown how the proposed overload control scheme could be extended to the multiple upstream cases while parameters such as fairness as well as throughput were considered.

VII. EVALUATION OF EFFICIENCY

In this section, efficiency of overload control algorithm is reviewed and compared with local overload control. Reports produced by Spirent Abacus 5000 device were used to check time and type of sent and received messages by users. Also, reports of Asterisk software were used to measure status of progression of calls and transactions that occurred in proxy; and also Oprofile software was used to measure processing load of proxy. There are various criteria to determine efficiency of SIP [33], amongst which, in this study, the delay of connection establishment (interval between sending "INVITE" from UAC to receiving OK from proxy), retransmission rate and proxy throughput (number of successful calls per unit of time) were concentrated on. Conversation production rate starts from low amount and continues to heavy rates of about 1600 cps.

A. Result for Trapezoid Topology

1) Throughput

Figure 26 shows throughput as a function of rate of received call requests in the case of existence and non-existence of overload control method, which represents that proxy's throughput could be maintained at its around maximum capacity in case of existence of overload control mechanism. As is shown in Figure 26, using fuzzy overload control mechanism, the upstream proxy was able to maintain its throughput up to about 1500 cps, which was about twice the capacity of downstream proxy, whereas, if overload control algorithm was not used, upstream proxy's throughput would be approximately equal to the one related to downstream proxy (700 cps). Also, this figure showed throughput when perfect overload control was done (curved labeled "Theoretical"), which kept throughput at the maximum downstream server capacity of 700 cps. Under overload conditions, throughput of the proposed mechanism converged to 645 cps and was almost independent from the load. On the other hand, throughput of the local OC approach was lower than that of the proposed mechanism and furthermore decreased with the load increase. The curve labeled "WIN-DISC" is an explicit feedback window-based approach proposed in [29] by Shen et Al., in which the downstream server calculated and sent back a window size at the end of each discrete control interval of Tc = 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 Tm = 100 ms, exactly as reported in [29]). Throughput of the proposed mechanism was constantly higher than that of WIN-DISC. The poor throughput of WIN-DISC might be attributed to the explicit nature of the used feedback, which is known to result in throughput degradation and stability problems as the feedback loop delay increases.Using OCC, CPU utilization rapidly fluctuated and became more severe as overload deteriorated. This was due to the regenerative nature of overload where calls progressively took much more CPU time due to retransmissions. However, once enough calls were rejected, the CPU utilization dropped.

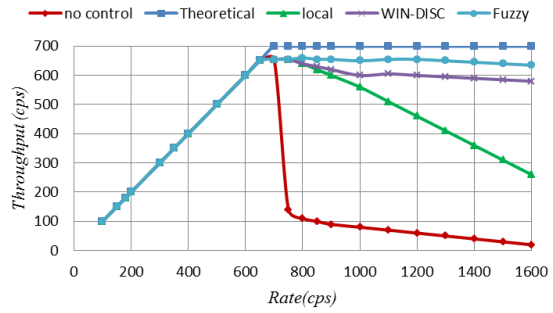

Fig. 26. Performance comparison of the proposed mechanism with that of WIN-DISC and no control

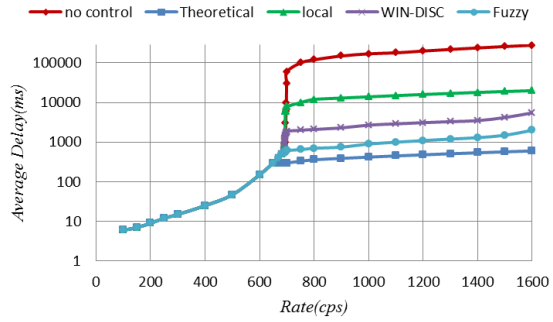

Fig. 27. Average delay comparison of the proposed mechanism with that of WIN-DISC and no control

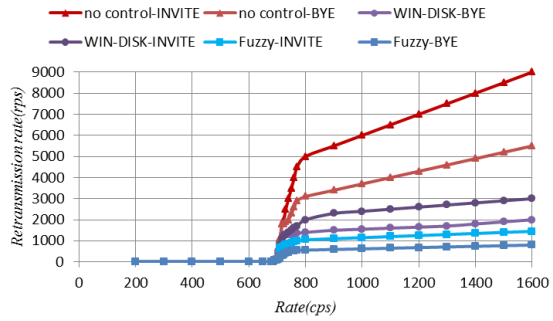

Fig. 28. Retransmission rate comparison of the propsoed mechanism with that of WIN-DISC and no control

2) *Average Delay of Call Establishment*

It was seen that, as new requests were received, window size started to increase and therefore delay increased, too. As shown in Figure 27, this linearly increased the average time of call establishment in this proxy to about 1500 cps with growth rate far much lower than the case in which the overload control mechanism was not used.

3) *Retransmission Rate*

In rates higher than downstream server's capacity (700 cps), the huge amount of received requests stimulated CPU sensor and therefore many calls were rejected. Sudden rejection of calls led many "Ack" packets to reach proxy in a very short interval and therefore fill the queue of proxy so that there was no place in this queue for the packets of answers related to ongoing calls. Missing of answer packets was a stimulation to activate retransmission mechanisms in both server and client which deteriorated the situation. So, calls accepted in proxy in this status were accepted with a very long delay. The diagram shown in Figure 28 individually illustrates retransmission rate for "INVITE" and "BYE" requests from the user side. As expected, when fuzzy overload control mechanism was used in upstream server, resending rates of messages considerably decreased. Overload led to loss of "OK" packets related to the passed calls. Thus, the proxy was required to resend "INVITE" requests related to lost packets. In this case, increase of resending rate made proxy spend much of its time on resending requests related to ongoing calls and therefore throughput rate of proxy considerably fell. Processing the abundant packets which existed in proxy's queue caused more delay in the passes calls and increased resending rate on caller's side. Note that, if the capacity of proxy's queue was high enough, additional requests were rejected before being expired and resent from user's side; on the other hand, not losing "OK" packets related to the passed calls in this case led to decrease in average time of successful call establishment and therefore there was no need for proxy to resend "INVITE" messages; so, throughput would have less decrease.

4) *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. In traditional mechanisms, as new requests are received, window size as well as delay starts to increase. This continues until delay exceeds a threshold value. At this time, window size reduces to its half. However, in fuzzy method, the average window size changes majorly about maximum window size. As shown in Figure 29, this issue leads to retention of throughput at the capacity of downstream proxy and rejection of other incoming requests.

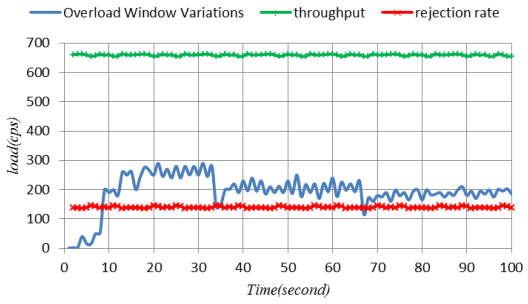

Fig. 29. Windows variations, throughput and rejection rate with rate of 800 cps

B. *Result for Edge-Core Topology*

In this section, the Edge-Core topology was used, as illustrated in Figure 25, to study performance of the proposed overload control algorithm when used by multiple upstream proxies. In the implementation scenario, two upstream edge proxies were considered, each forwarding calls from a group of UAs to a single core proxy causing it to overload. Also, fairness of the proposed overload control algorithm was investigated

through monitoring the throughput perceived by each upstream edge proxy during the overload.

1) Fairness Analysis

In order to provide fairness during the overload, capacity of the overloaded downstream proxy was required to be equally split between all upstream (edge) proxies which communicated with it. Note that, with the proposed mechanism, the downstream server did not need to know about the number of upstream servers connected to it and also did not generate any extra feedback.

The proposed mechanism did not require changes in the SIP protocol since it was implemented in the sending (upstream) servers; consequently, it did not impose processing burden on overloaded servers.

Indeed, Figure 30 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 in 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 0 to 100 s, only one upstream proxy was sent at 650 cps so it got the entire capacity. From 100 to 200 s, there were two operating proxies, each receiving roughly equal throughput of 360 cps and 340 cps. At 200 s, the second proxy stopped sending requests and another one regained the extra capacity.

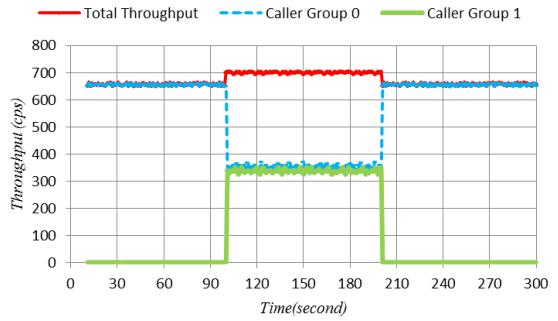

Fig. 30. Fairness analysis of the proposed method

VIII. CONCLUSION

The studies accomplished in this paper showed that SIP protocol was not efficient enough in facing congestion so that, when call request rate increased, delay of call establishment increased suddenly, proxy's throughput fell and consequently retransmission rates and unsuccessful calls increased. In this paper, fuzzy window-based control method was developed, implemented and tested on a real platform. Also, the efficiency of SIP proxy in case of overload was studied using a distributed overload control method, which was developed on Asterisk open source proxy. The proposed overload control algorithm for SIP servers was a window-based approach that required no extra feedback and used fuzzy logic to detect the overload. Moreover, the suggested method could change the maximum window size dynamically. Studying the charts of throughput, delay and retransmission rate of "INVITE" and "BYE" messages demonstrated that the proposed algorithm was able to maintain the throughput at a high level and be fair. As future works, we intend to investigate more sophisticated window update strategies. In addition, an analytical model as well as stability analysis of the SIP network is also underway.

ACKNOWLEDGEMENTS

This work was supported by IP-PBX Type approval laboratory, Ferdowsi University of Mashhad, both morally and materially.

REFERENCES

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. RFC 2068, Internet Engineering Task Force, January 1997.

[2] J. Rosenberg and Henning Schulzrinne. An offer/answer model with session description protocol (SDP). RFC 3264, Internet Engineering Task Force, June 2002.

[3] Henning Schulzrinne, Stephen Casner, Ron Frederick, and Van Jacobson. RTP: a transport protocol for real-time applications. RFC 3550, Internet Engineering Task Force, July 2003.

[4] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. SIP: Session initiation protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[5] P. Montoro, a Comparative Study of VoIP Standards with Asterisk, in: International Conference on Digital Telecommunications, 2009.

[6] D .Qin, Research on the Performance of Asterisk-Based Media Gateway, in: International Symposium on Knowledge Acquisition and Modelling (KAM), 2011.

[7] V. Hilt. I. Widjaja, Controlling overload in networks of sip servers, in: IEEE International Conference on Network Protocols, 2008, pp. 83–93.

[8] E.C. Noel, C.R. Johnson, Initial simulation results that analyze sip based voip networks under overload, International Teletraffic Congress (2007) 54–64.

[9] V. Hilt, E. Noel, C. Shen, A. Abdelal, Design consideration for session initiation protocol (sip) overload control, SOC working group, Internet draft,draft-ietfsoc-overload-design, Work in progress (May 2011).

[10] C. Shen and H. Schulzrinne, On TCP-based SIP server overload control, in: Principles, Systems and Applications of IP Telecommunications, 2010.

[11] E. M. Nahum, J. Tracey, and C. P. Wright, Evaluating SIP server performance, in: SIGMETRICS Perform. Eval. Rev., vol. 35, pp. 349-350, 2007.

[12] D. Y. Hwang, J. H. Park, S. W .Yoo, and K. H. Kim, A window-based overload control considering the number of confirmation Massages for SIP server,in: Ubiquitous and Future Networks (ICUFN),2012.

[13] M. Ohta, Performance comparisons of transport protocols for session initiation protocol signaling, in: Telecommunication Networking Workshop on QoS in Multiservice IP Networks, 2008, pp. 148–153.

[14] H. Schulzrinne, SIPstone – Benchmarking SIP server performance, Tech. rep., Columbia University, Apr. 2002. Available from http://www.columbia.edu/

[15] Raatikainen K., et al., A Control Plane Benchmark for Telecommunications Signalling Applications ,Linux Conf Europe, 5th September 2007.

[16] M. Cortes, J. R. Ensor, and J. O. Esteban, On SIP Performance, Bell Labs Technical Journal, Volume 9, Issue 3, pp. 155-172, 2004.

[17] h S. Wanke, M. Scharf, S. Kiesel, S. Wahl, Measurement of the SIP Parsing Performance in the SIP Express Router, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Berlin / Heidelberg, Volume 4606 LNCS, pp. 103-110, 2007.

[18] SS Gokhale., Signaling performance of SIP based VoIP: A measurement-based approach, In Proc. of IEEE Globecom '05, Vol. 2,pp. 761-765, Nov. 2005.

[19] Caixia Chi, Dong Wang, Ruibing Hao, Wei Zhou, Performance Evaluation of SIP Servers, 2008.

[20] Erich M. Nahum, John Tracey, and Charles P. Wright, Evaluating SIP Proxy Server Performance, 17th International workshop on Network and Operating Systems Support for Digital Audio & Video, 2007.

[21] Vincent Planat, Nadjia Kara, SIP Signaling Retransmission Analysis over 3G network, MoMM, 2006.

[22] A. Munir, Analysis of SIP-based IMS Session Establishment Signaling for WiMax-3G network, Proceedings of the Fourth International Conference on Networking and Services,Volume 00, pp.282-287, 2008.

[23] V. K. Gurbani, R. Jain, Transport Protocol Considerations for Session Initiation Protocol Networks, Bell Labs Technical Journal, Volume 9, Issue 1, pp. 83-97, 2004.

[24] Masataka Ohta, Performance Comparisons of Transport Protocols for Session Initiation Protocol Signaling, 2008.

[25] Kaushik Kumar Ram, Ian C. Fedeli, Alan L. Cox, and Scott Rixner. Explaining the Impact of Network Transport Protocols on SIP Proxy Performance, IEEE International Symposium on Performance Analysis of Systems and software, ISPASS,pp. 75-84, 2008.

[26] S. Montagna, M. Pignolo, Performance evaluation of load control techniques in sip signaling servers, in: ICONS, 2008, pp. 51–56.

[27] M. Ohta, Overload control in a sip signaling network, Transactions on Engineering Computing and Technology 12 (2006) 2006.

[28] M. Ohta, Overload protection in a sip signaling network, in: International Conference on Internet Surveillance and Protection, 2006, pp. 11–11.

[29] C. Shen, H. Schulzrinne, E. M. Nahum, Session initiation protocol (sip) server overload control: design and evaluation, in: IPTComm, 2008, pp. 149–173.

[30] S. Low, F. Paganini, J. Doyle, Internet congestion control, IEEE Control Systems Magazine 22 (1) (2002) 28–43.

[31] V. Hilt, E. Noel, C. Shen, A. Abdelal, Design consideration for session initiation protocol (sip) overload control, IETF,RFC6357,August 2011.

[32] D.-M. Chiu, R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, Computing Network ISDN System 17 (1) (1989) 1–14.

[33] D. Malas, SIP End-to-End Performance Metrics, Internet-Draft, October 31, 2008 (work in progress), http://www.ietf.org/internet-drafts/draft-ietf-pmol-sip-perf-etrics-02.txt.

[34] R.Mahmood, M.A.Azas, SIP messages delay analysis in heterogeneous network, International Conference on Wireless Communication and Sensor Computing, 2010.

[35] E. Noel, C. Johnson, Novel overload controls for sip networks, in: 21st International Teletraffic Congress 2009, 2009, pp. 1–8.

[36] S. Montagna, M. Pignolo, Comparison between two approaches to overload control in a real server: local or hybrid solutions, in: Proceedings of the 15th IEEE Mediterranean Electrotechnical Conference, MELECON 2010, IEEE Press, Piscataway, NJ, USA, 2010, pp. 845–849.

[37] M. Homayouni, S. Azhari, M. Jahanbakhsh, A. Akbari, A. Mansoori, N. Amani, Configuration of a sip signaling network: an experimental analysis, in: Fifth International Joint Conference on INC, IMS and IDC 2009, 2009, pp. 76 – 81.

[38] Y. Hong, C. Huang, J. Yan, Mitigating sip overload using a control-theoretic approach, in: Global Telecommunications Conference (GLOBECOM 2010), IEEE, 2010, pp. 1–5.

[39] A. Abdelal, W. Matragi, Signal-based overload control for sip servers, in: Proceedings of the 7th IEEE Conference on Consumer Communications and Networking Conference, CCNC'10, IEEE Press, Piscataway, NJ, USA, 2010, pp. 990–996.

[40] T.-Y. Chi, C.-H. Chen, H.-C. Chao, S.Y. Kuo, An efficient earthquake early warning message delivery algorithm using an in time control-theoretic approach, in: C.-H. Hsu, L. Yang, J. Ma, C. Zhu (Eds.), Ubiquitous Intelligence and Computing, Lecture Notes in Computer Science, 6905, Springer, Berlin, Heidelberg, 2011, pp. 161–173.

[41] R.G. Garroppo, S. Giordano, S. Niccolini, S. Spagna, A prediction-based overload control algorithm for sip servers, in: IEEE Transactions on Network and Service Management, IEEE Press, Piscataway, NJ, USA, vol. 8, 2011, pp. 39–51.

[42] Abaev, Pavel ,On analytical model for optimal SIP server hop-by-hop overload control, in: 4th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops 2012.

[43] Kuzminykh, I. ,A combined LIFO-Priority algorithm for overload control of SIP server, in:International Conference on Modern Problems of Radio Engineering Telecommunications and Computer Science 2012.

[44] J. Sun, H. Yu, W. Zheng, Flow management with service differentiation for sip application servers, in: CHINAGRID '08: Proceedings of the The Third ChinaGrid Annual Conference, IEEE Computer Society, Washington, DC, USA, 2008, pp. 272–277.

Ahmadreza Montazerolghaem was born on July 1987 in Esfahan, Iran. He received the B.Sc. degree in Information Technology from the computer department, Sadjad University of Technology and his M.Sc. degree in computer engineering from the computer department, Ferdowsi University of Mashhad, Mashhad, Iran, in 2010 and 2013, respectively. He is a Ph.D. candidate in computer engineering at computer department, Ferdowsi University of Mashhad. His research interests are in Next Generation Networks, Voice over IP, Fuzzy Logic Control, Overload Control and Session Initiation Protocol.

Mohammad Hossein Yaghmaee received his B.S. degree in Communications Engineering from Sharif University of Technology, Tehran, Iran in 1993, and his M.Sc. degree in Communications Engineering from Tehran Polytechnic (Amirkabir) University of Technology in 1995. He received his Ph.D. degree in Communications 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. From September 2007 to August 2008, he was with the Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, USA as a visiting associate professor. He is the author of 5 books all in Farsi language. He has published more than 130 international conferences and journal papers in. He is head of IP-PBX type approval lab at Ferdowsi University of Mashhad.