Journal of Network and Computer Applications 41 (2014) 114–125

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Probe-based end-to-end overload control for networks of SIP servers

Jinzhu Wang[a,*], Jianxin Liao[b], Tonghong Li[c], Jing Wang[b], Jingyu Wang[b], Qi Qi[b]

[a] China Mobile, Beijing 100053, China
[b] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, P.O. Box 296, Beijing 100876, China
[c] Department of Computer Science, Technical University of Madrid, Madrid 28660, Spain

ARTICLE INFO

Article history:
Received 13 April 2013
Received in revised form
28 September 2013
Accepted 21 October 2013
Available online 20 November 2013

Keywords:
Overload control
End-to-end
Probing
SIP
VoIP

ABSTRACT

The Session Initiation Protocol (SIP) has been adopted by the IETF as the control protocol for creating, modifying and terminating multimedia sessions. Overload occurs in SIP networks when SIP servers have insufficient resources to handle received messages. Under overload, SIP networks may suffer from congestion collapse due to current ineffective SIP overload control mechanisms. This paper introduces a probe-based end-to-end overload control (PEOC) mechanism, which is deployed at the edge servers of SIP networks and is easy to implement. By probing the SIP network with SIP messages, PEOC estimates the network load and controls the traffic admitted to the network according to the estimated load. Theoretic analysis and extensive simulations verify that PEOC can keep high throughput for SIP networks even when the offered load exceeds the capacity of the network. Besides, it can respond quickly to the sudden variations of the offered load and achieve good fairness.

1. Introduction

The Session Initiation Protocol (SIP) (Rosenberg et al., 2002) is an IETF-defined application-layer control protocol widely used for creating, modifying and terminating multimedia sessions. Typical SIP applications include Voice over IP (VoIP), multimedia distributions, video conferencing, instant messaging and presence service (Liao et al., 2011). The SIP has been adopted by the 3GPP as the basis for the IP Multimedia Subsystem (IMS) architecture.

SIP is a request/response-based protocol. Each end user is represented by a user agent (UA), which takes the role of a user agent client (UAC) or a user agent server (UAS) for a request/response pair. A UAC creates a SIP request and sends it to a UAS. The request traverses through one or more SIP servers (also called SIP proxies) in a SIP network. The main purpose of a SIP server is to route a request to its destination. The response traces back the path the request has taken. Fig. 1 shows an example of SIP call flow. A SIP call is initialized by an INVITE request and terminated by a BYE request. SIP is call-oriented and the SIP server can only reject/drop the INVITE requests if it is unwilling or unable to forward requests. There is no reason to reject/drop messages of an on-going call such as 200 response, ACK request and BYE request. Two typical SIP networks consisting of edge servers and core servers are shown in Fig. 2. Each UA is connected to the network via an edge server located closest to it. When a SIP call between two UAs goes through the network, the first server the call (i.e., the INVITE request of the call) arrives at is denoted as the ingress server, and the last server the call arrives at is denoted as the target server. Clearly, both ingress server and target server are edge servers.

The widespread popularity of SIP has raised attention to its readiness of handling overload (Rosenberg, 2008). Overload of a SIP server occurs if the message arrival rate to the server exceeds its message processing capacity. A SIP server can be overloaded for many reasons, such as emergency-induced call volume, flash crowds generated by the popular TV show, simultaneous registrations of many users due to recovery after a large power outage, or even denial of service attacks. Under overload, the throughput of a SIP server can drop significantly and can even reach zero. Besides, the call setup delay becomes unacceptable for a real-time media call. Furthermore, a number of retransmission timers are defined in SIP to cope with message losses, especially when the unreliable UDP transport is used. Thus, both the SIP server and the UA retransmit a request if a response has not been received in time. Under overload, the SIP server becomes significantly less responsive, which causes a large number of requests to be retransmitted by its neighbors. This aspect not only aggravates the load on the overloaded server, but also leads to overload in its neighbors. In this way, overload can spread in a network of SIP servers and eventually bring down the entire network.

Several design approaches have been proposed in the literatures (Hilt and Widjaja, 2008; Hilt et al., 2011; Gurbani et al., 2013)

* Corresponding author.
E-mail addresses: wangjinzhu8494@gmail.com (J. Wang), liaojx@bupt.edu.cn (J. Liao), tonghong@fi.upm.es (T. Li), wangjing@bupt.edu.cn (J. Wang), wangjingyu@bupt.edu.cn (J. Wang), qiqi8266@bupt.edu.cn (Q. Qi).

1084-8045/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jnca.2013.10.017

in order to manage the overload in the SIP networks, which can be classified into local, hop-by-hop and end-to-end overload control. In local overload control, the SIP server monitors its load and starts to reject requests locally by using 503 (Service Unavailable) responses (Rosenberg, 2008; Hilt and Widjaja, 2008) when it detects overload. In hop-by-hop overload control, the overloaded SIP server can provide feedback to its direct upstream neighbors, which then adjust the amount of traffic forwarded to this SIP server to eliminate overload. The feedback can be conveyed in a SIP response header (Gurbani et al., 2013). In end-to-end overload control, the edge servers, which are considered as the closest servers to the sources of traffic in a SIP network, are responsible for adjusting the amount of traffic forwarded to the overloaded server to eliminate overload. The research in Hilt and Widjaja (2008) indicates that end-to-end overload control achieves the best performance although it is the most complex among all types of overload control approaches.

In this paper, we propose and design PEOC, a probe-based end-to-end overload control mechanism, which is deployed at edge servers and is easy to implement. By probing the SIP network with SIP messages, PEOC estimates the network load and controls the traffic admitted to the network based on the estimated load. The remainder of this paper is organized as follows: Section 2 surveys related work. Section 3 proposes the design of PEOC and Section 4 analyzes the Probe-based Rate Adaption (PRA) algorithm, which can dynamically adjust the rate of calls admitted according to the overload feedback received from the network and the estimated network load obtained by probing the network. In Section 5, the

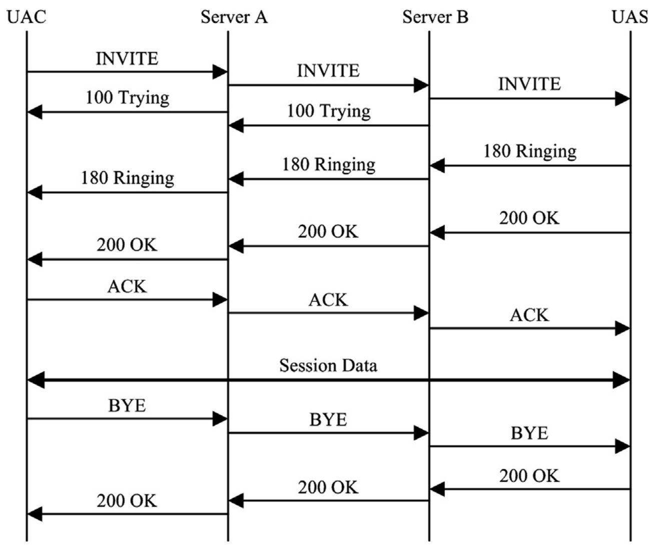

Fig. 1. Sample SIP call flow.

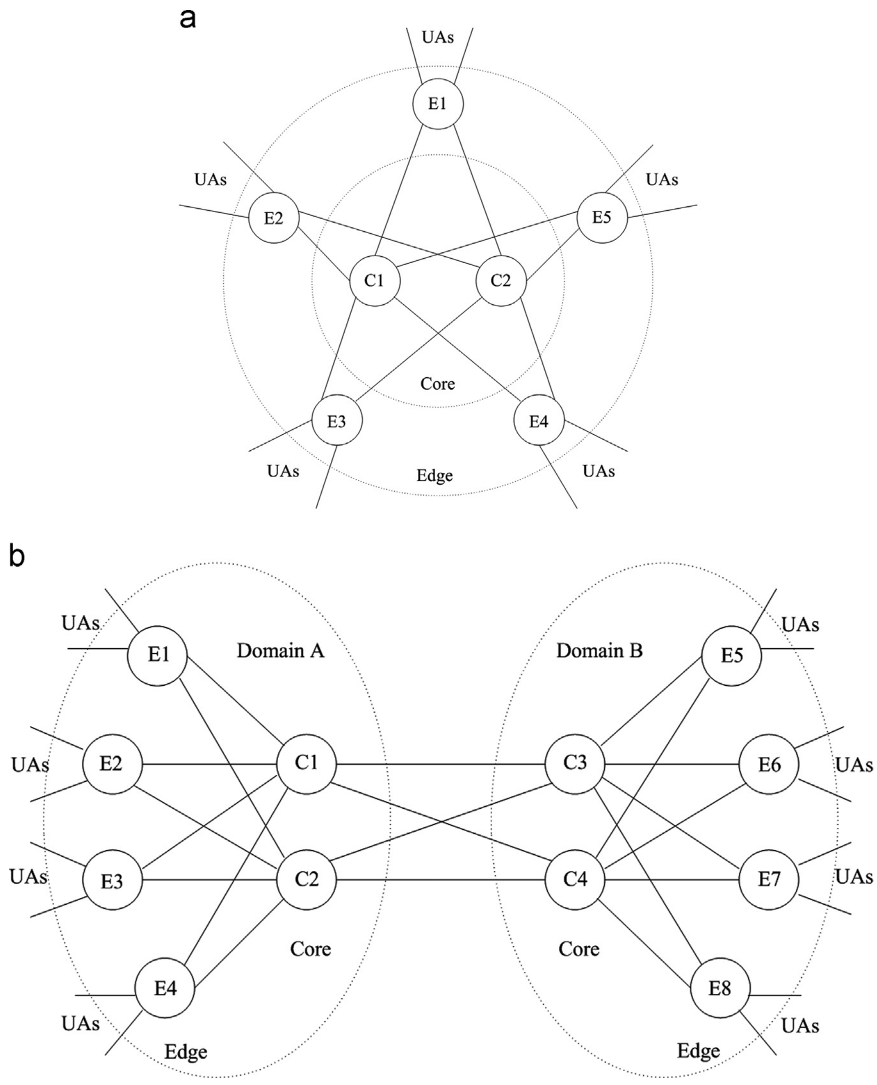

Fig. 2. Typical SIP network topologies. (a) Topology 1 and (b) topology 2.

probing mechanism in PEOC is discussed in detail. Section 6 describes the simulation models and presents the performance results. Finally, conclusions and possible future work are presented in Section 7.

2. Related work

The research on SIP overload control has attracted much attention recently. In (Rosenberg, 2008; Hilt and Widjaja, 2008; Hong et al., 2010), the overload problem of SIP and the ineffectiveness of its built-in overload control mechanisms are studied in detail. In Hilt and Widjaja (2008), Ohta (2006a,b), Cyr et al. (1990), Kasera et al. (2001), and Garroppo et al. (2009), several local overload control mechanisms have been proposed. Furthermore, recent research focuses on studying hop-by-hop overload control mechanisms to handle overload more effectively, which are classified into receiver-based and sender-based overload control. In receiver-based control, the overloaded server calculates restrictions on its offered load according to current load and distributes these restrictions to its direct upstream neighbors as the feedback. Its direct upstream neighbors only follow the received restrictions to throttle traffic forwarded to the overloaded server. This type of overload control is adopted by Hilt and Widjaja (2008), Noel and Johnson (2007), Shen et al. (2008), and Garroppo et al. (2011). On the other hand, in sender-based control, the overloaded server only implements local overload control which rejects requests by using 503 responses. Based on the received 503 responses, its direct upstream neighbors calculate and then follow the restrictions on the traffic forwarded to the overloaded server. Abdelal and Matragi (2010) and Azhari et al. (2012) are typical implementations of sender-based overload control. Note that receiver-based control is more complex than sender-based control as receiver-based control adds extra burden on the overloaded server to calculate restrictions and then distribute these restrictions to its direct upstream neighbors.

Similar to hop-by-hop overload control, end-to-end overload control can also be classified into receiver-based and sender-based overload control. In receiver-based control, the SIP server sends its overload/restriction to the upstream neighbors, which then forward the overload/restriction to their upstream neighbors. In this way, the overload/restriction is eventually propagated to all ingress servers. The ingress server adjusts the amount of traffic forwarded to the overloaded server based on the received overload/restriction. Hilt and Widjaja (2008) and Wang (2010) belong to this type of overload control.

Note that in receiver-based hop-by-hop overload control, the overload/restriction is propagated just through one hop and only the overloaded server takes charge of propagating. However, in receiver-based end-to-end overload control, it is propagated through multiple hops and SIP servers located between the edge server and the overloaded server are all involved in propagating, which needs the complex cooperation among them. Thus, receiver-based end-to-end overload control is even more complex than receiver-based hop-by-hop overload control. On the other hand, sender-based end-to-end overload control is simple and practical. In Liao et al. (2012), our previous work proposed a distributed end-to-end overload control (DEOC) mechanism, which is a sender-based overload control, where the core servers only implement local overload control that rejects requests by using 503 responses. Based on the received 503 responses, the edge servers calculate and then follow the restrictions on the traffic admitted to the SIP network. Simulation result shows that this approach can keep high throughput for SIP networks even when the offered load exceeds the capacity of the network and can respond quickly to the sudden variations of the offered load.

DEOC is a reactive mechanism, because the edge servers in DEOC calculate restrictions only based on a simple binary overload feedback received from the network (i.e., whether or not 503 responses are received), indicating whether the network is currently overloaded or underutilized. In order to further improve the throughput for SIP networks and respond more quickly to the sudden variations of the offered load, in this paper we propose a probe-based end-to-end overload control (PEOC) mechanism. PEOC is also a sender-based overload control. Besides, it is a proactive mechanism because the edge servers in PEOC firstly estimate the network load by probing the SIP network with SIP messages and then calculate restrictions based on both the estimated load and the binary overload feedback received from the network.

3. Probe-based end-to-end overload control[1]

In this section we develop PEOC, a probe-based end-to-end overload control mechanism for SIP networks. In PEOC, the core servers only implement local overload control that rejects requests by using 503 responses. When receiving a 503 response from a downstream neighbor, the server forwards this response to the upstream neighbor, from which the INVITE request related to this response has been received. In this way, the 503 response will finally be forwarded to the edge server. The edge servers estimate the network load by probing the SIP network with SIP messages. Based on the received 503 responses and the estimated load, they calculate and then follow the restrictions on the traffic admitted to the SIP network.

We deploy a set of PEOCs at each ingress server to control overload for the SIP network. At an ingress server, each PEOC is related to a specific target server and controls the arriving calls from UAs that take this ingress server as first-hop and the target server as last-hop in the network. Thus, the load of the network is controlled by PEOCs at all ingress servers. Note that our approach is completely distributed: there is no centralized entity to control PEOCs and each PEOC is functionally identical and operates independently. Besides, there is no communication between PEOCs. Finally, our approach can be deployed incrementally, by installing PEOCs on ingress servers, with no need to alter other servers in the SIP network. Therefore, our approach is easy to implement.

The task of a PEOC can be split into four separate parts: measurement, restriction, probing and control decision. Fig. 3 shows the functional modules in PEOC. The solid line and dashed line represent the data flow and the control flow, respectively. The arriving call at the PEOC firstly goes through measurer module, which is used to measure the inter-arrival time of calls. Afterwards, it goes through restrictor module, which determines whether or not to throttle the received calls in order to avoid overload in network.

3.1. Measurement and restriction

In the measurer module, the inter-arrival time of calls is measured and a standard EWMA (exponentially-weighted moving average) filter is applied to smooth out short-term fluctuations

[1] This section presents the architecture of PEOC. Since PEOC is based on DEOC (Liao et al., 2012) and further adopts a proactive control mechanism to achieve a more efficient overload control, the basic functional modules (Measurer and Restrictor) and the state machine of Controller are the same as those in Liao et al. (2012). We introduce these functional modules in this paper in order to make the paper more readable.

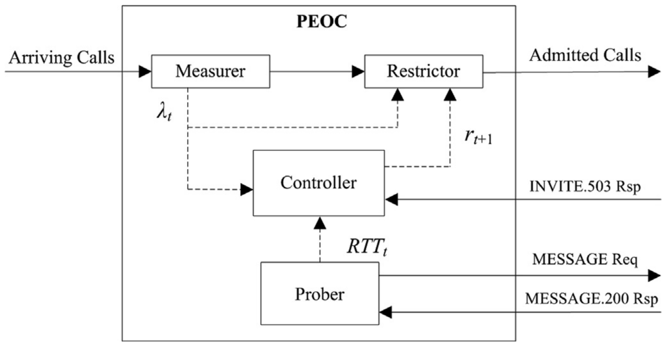

Fig. 3. The functional modules in PEOC.

as follows:

$$\Delta I_{avg} = (1-w) \times \Delta I_{avg} + w \times \Delta i, \quad 0 < w \le 1 \tag{1}$$

where ΔI_{avg} is the average inter-arrival time of calls, Δi is the measured inter-arrival time of calls, and w is the EWMA smoothing weight. We define call arrival rate λ as the number of arriving calls at the PEOC per unit time and it is calculated as $\lambda = 1/\Delta I_{avg}$.

The controller module calculates the call admission rate (denoted as r) periodically and the calculation interval is T. At the end of each interval T, the controller module obtains from the measurer module the current call arrival rate λ_t, which denotes the call arrival rate at time t (in T). Note that the time t is measured in T since r is calculated periodically. Then the controller module obtains from the prober module the current application-layer round trip time RTT_t, which represents the SIP network load at time t (in T). Finally the controller module calculates the call admission rate r_{t+1} for the next T and sends it to the restrictor module to throttle arriving calls according to this threshold.

The prober module periodically probes the SIP network with SIP messages and passes the measured application-layer RTT to the controller module. We will detail the probing mechanism of the prober module in Section 5.

In the restrictor module, we adopt call gapping (Berger, 1991) to throttle arriving calls. Once admitting a call, the restrictor starts a timer of duration τ, which is the gap interval. Then it rejects all subsequent calls arriving before the timer expires. Every time the restrictor module receives r_{t+1} from the controller module, it obtains λ_t from the measurer module and then calculates the gap interval in the next T. Suppose that the arrival process of calls conforms to Poisson distribution. Referring to Berger (1991), the gap interval τ_{t+1} adopted in the next T is calculated as

$$\tau_{t+1} = \max(0, 1/r_{t+1} - 1/\lambda_t) \tag{2}$$

When the gap interval is 0, the restrictor module does not throttle arriving calls and all arriving calls are admitted to the network.

3.2. Control decision

The main function of PEOC is to calculate r. In the following, we design a Probe-based Rate Adaption (PRA) algorithm that can dynamically adjust the call admission rate r. The PRA consists of an increasing rule and a decreasing rule. When there is no overload feedback, PRA increases r according to the increasing rule. When receiving the overload feedback, PRA decreases r according to the decreasing rule. Besides, PRA uses the estimated network load, which is obtained from the prober module, to design the increasing rule. We will elaborate on PRA in Section 4.

The controller module periodically executes PRA (with interval T) and takes the number of received 503 responses during each T as the overload feedback to PRA. The controller module is implemented based on a finite state machine. The state machine is executed at the end of each T, which takes λ_t and the number of received 503 responses in the current T as input and outputs r_{t+1} to control admitted calls in the next T. Note that the state machine only uses a binary overload feedback. That is, it only needs the information about whether 503 responses are received in each T. The binary overload feedback indicates whether the network is currently overloaded or underutilized. A very good reason for adopting a binary feedback is that it makes the PEOC as simple and practical as possible. Besides, it also minimizes the overhead of generating the feedback in the network. We omit the introduction of the finite state machine in this paper, which is similar to that in Liao et al. (2012). The detailed description of the finite state machine can be found in our previous work (Liao et al., 2012).

4. Probe-based Rate Adaption (PRA) design

In this section we design PRA. The overload of the SIP network is managed by a lot of PEOCs located on the edge of the network. These PEOCs are distributed and each PEOC executes PRA to calculate the call admission rate based on the overload feedback received from the network. The feedback should be designed to be as simple as possible in order to make PEOC simple and practical. Considering that these features are similar to those of the TCP congestion avoidance (Allman et al., 2009), the design of PRA can be inspired by the TCP congestion avoidance. The basic TCP congestion avoidance algorithm, which is proposed by Chiu and Jain (1989) and Jacobson (1988) and is based on a simple binary overload feedback received from the network, is applied by Liao et al. (2012) to the SIP overload control as follows:

$$\text{increasing}: \; r_{t+1} = r_t + \alpha, \quad \alpha > 0 \tag{3}$$

$$\text{decreasing}: \; r_{t+1} = r_t - \beta r_t, \quad 0 < \beta < 1 \tag{4}$$

where r_t is the call admission rate at time t (in T). α and β are constant factors. The algorithm is periodically executed (the period is T). If no call rejection is received in the current period, the call admission rate in the next period is increased additively. Otherwise, it is decreased multiplicatively. Therefore, this algorithm is called as AIMD (additive increase and multiplicative decrease) (Chiu and Jain, 1989).

4.1. Aggressiveness, responsiveness and throughput

Before presenting the PRA, we first consider the important properties of call admission rate control algorithms in SIP networks including aggressiveness, responsiveness and throughput (Liao et al., 2012). The network is underutilized when the call admission rate is below the capacity of the network. In this case, a good call admission rate control algorithm needs to increase the call admission rate as fast as possible in order to make full use of network resources and avoid unnecessary call rejections. Aggressiveness measures how fast a call admission rate control algorithm makes use of network resources as they are available. We define aggressiveness as the inverse of the time needed for the call admission rate control algorithm to achieve the increment of a certain amount of call admission rate, in response to: (1) a step increase of available network resources or (2) a step increase of call arrival rate when there are available resources in the network. Obviously, high aggressiveness, implying potentially high utilization, is desirable.

The network is overloaded when the call admission rate exceeds the capacity of the network. In this case, a good call admission rate control algorithm needs to decrease the call admission rate as fast as possible in order to eliminate overload.

Responsiveness measures how fast a call admission rate control algorithm decreases the call admission rate in response to overload. We define responsiveness as the inverse of the time needed for the call admission rate control algorithm to achieve the decrement of a certain amount of call admission rate, in response to a step increase of network overload. Obviously, high responsiveness, which allows call admission rate control algorithm to decrease the call admission rate quickly when overload occurs, is desirable.

The network is fully utilized when the call admission rate is close to the capacity of the network. In this case, since the feedback is binary, the call admission rate oscillates around the network capacity over time and the throughput of the network is determined by the call admission rate control algorithm. The throughput in SIP overload control is determined by effective rate and stable duration. We define effective rate as the average call admission rate during one increasing phase. The increasing phase consists of a sequence of consecutive call admission rate increases, which is followed by call admission rate decreases. High effective rate, which implies high throughput of the SIP network, is desirable. We define stable duration as the number of periods in one increasing phase. A shorter stable duration leads to the occurrence of overload being more frequent. As the network's throughput can be reduced by the occurrence of overload, short stable duration is not desirable.

Based on these properties, Liao et al. (2012) analyzes the performance of AIMD. As for the increasing rule, due to the constant increasing rate of call admission rate, the aggressiveness is low when α is small, which leads to rejecting a lot of arriving calls unnecessarily when the network is underutilized. On the other hand, the stable duration is short when α is large, which causes frequent decrease of call admission rate and thus results in low throughput of SIP network. Therefore, the constant increasing rate cannot satisfy the requirements for SIP overload control. On the other hand, the decreasing rule seems to be suitable for SIP overload control. Since the decreasing is multiplicative, it is possible to achieve high responsiveness even by using a small β, which can keep high throughput of the SIP network.

4.2. Proactive vs. reactive algorithm

Note that the AIMD is reactive. That is, this algorithm calculates the call admission rate only based on the binary overload feedback received from the network, which can be used to decide whether the increasing rule or the decreasing rule is adopted. During the increasing phase, this reactive algorithm only adopts a predefined increasing rule. However, the aggressiveness is low if the call admission rate is increased too slowly (i.e., α is small) and the throughput is low if the call admission rate is increased too fast (i.e., α is large). Thus the increasing rule (3) cannot achieve high aggressiveness and high throughput simultaneously. The research (Liao et al., 2012) further extends the linear increasing rule (3) to the non-linear one in order to achieve a better tradeoff between aggressiveness and throughput. However, it is still a reactive algorithm and cannot achieve high aggressiveness and high throughput simultaneously.

Therefore, in order to achieve high aggressiveness as well as high throughput, we should design a call admission rate control algorithm to adopt different increasing rules adaptively based on the different network load during the increasing phase. That is, when the current network load is low, since the network will not be overloaded, the call admission rate should be increased aggressively in order to achieve high aggressiveness. On the other hand, when the current network load is high, since the network tends to be overloaded, the call admission rate should be increased conservatively in order to avoid overload and achieve high throughput. The current network load can be obtained by probing the SIP network. Obviously, this algorithm is proactive, which can achieve both high aggressiveness and high throughput and thus is more suitable for SIP overload control than the reactive algorithm.

4.3. The increasing rule of the Probe-Based rate Adaption (PRA)

Motivated by the conclusion that the proactive algorithm is more effective than the reactive algorithm, in this paper we propose a probe-based call admission rate adaption algorithm (PRA). PRA also uses the binary overload feedback received from the network to decide whether the increasing rule or the decreasing rule is adopted. Besides, it obtains the estimated network load from the prober module, which probes the SIP network with SIP messages. During the increasing phase, PRA adopts different increasing rules adaptively based on the different network load in order to achieve high aggressiveness as well as high throughput. Thus PRA is proactive and can control the overload of the network efficiently and in a timely manner. Note that the probing mechanism is important to PRA because the PRA uses it to obtain the network load. We will detail the probing mechanism in Section 5.

PRA takes the application-layer *RTT* as the indicator of the network load. At the end of each T (at time t), the controller module obtains the current RTT_t from the prober module, which represents the network load at time t. Based on the current network load RTT_t, PRA adopts different increasing rules during the increasing phase. We compare RTT_t with a predefined threshold T_{th} and consider that the network load is low if $RTT_t < T_{th}$ and is high if $RTT_t \geq T_{th}$. When the network load is low, since the network will not be overloaded, the call admission rate should be increased more aggressively in order to achieve high aggressiveness. Thus in this case, we choose the multiplicative increasing rule to achieve the high increasing rate of call admission rate. On the other hand, when the network load is high, since the network tends to be overloaded, the call admission rate should be increased more conservatively in order to avoid overload and achieve high throughput. Thus in this case, we choose the additive increasing rule to achieve the low increasing rate of call admission rate. To sum up, the increasing rule in PRA is as follows:

$$\text{increasing}: \begin{cases} r_{t+1} = r_t + \delta r_t, & 0 < \delta < 1, \quad \text{if} \quad RTT_t < T_{th} \\ r_{t+1} = r_t + \theta, & \theta > 0, \quad\quad\; \text{if} \quad RTT_t \geq T_{th} \end{cases} \tag{5}$$

where δ and θ are constant factors (we set $0 < \delta < 1$ to prevent the call admission rate from increasing too aggressively). That is, when the network load is low, the call admission rate is increased multiplicatively. Otherwise, it is increased additively. Besides, PRA adopts the decreasing rule (4) as its decreasing rule since (4) is suitable for SIP overload control.

According to (5), when the network load is low, the call admission rate is increased multiplicatively and δ determines its increasing rate. A larger δ leads to a much higher increasing rate. Obviously, a higher increasing rate achieves higher aggressiveness. On the other hand, it leads to lower throughput because the network tends to be overloaded more frequently when the call admission rate is increased too fast. Since a small δ can achieve high aggressiveness, δ should be set as a small value in order to keep high throughput.

On the other hand, when the network load is high, the call admission rate is increased additively and θ determines its increasing rate. The increasing rate increases as θ increases, thus larger θ leads to higher aggressiveness and lower throughput. Since θ is the increasing step size of call admission rate when the network load is high, θ should be set as a small value in order to avoid overload and achieve a better tradeoff between aggressiveness and throughput.

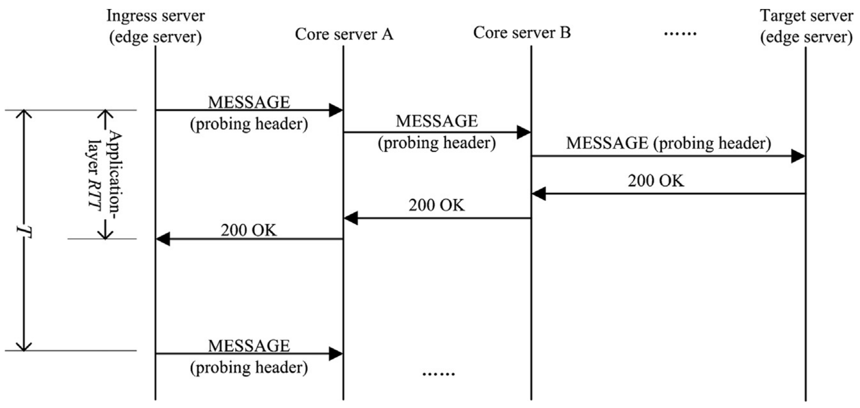

Fig. 4. Sample probing flow.

Note that T_{th} is the predefined threshold value, which can determine whether the network load is high or low. Obviously, a larger T_{th} leads to a larger proportion of multiplicative increase, which results in higher aggressiveness and lower throughput as multiplicative increase enables the call admission rate to increase very fast. On the other hand, a smaller T_{th} leads to a larger proportion of additive increase and thus results in lower aggressiveness and higher throughput. In the real circumstance, the value of T_{th} can be defined according to our requirements of aggressiveness and throughput.

5. Probing mechanism

5.1. Probing mechanism and prediction of application-layer RTT

PEOC uses the prober module to probe the SIP network with SIP messages. We define a Probing header and insert it into the MESSAGE request for probing. The probing flow is shown in Fig. 4. At the end of each *T*, PEOC sends a MESSAGE request with a Probing header to its target server. The MESSAGE request traverses through one or more core servers and finally arrives at the target server. When a target server receives a MESSAGE request with a Probing header, it replies a 200 OK response immediately and the 200 OK response traces back the path the MESSAGE request has taken. After receiving the 200 OK response, PEOC calculates the elapsed time between sending the MESSAGE request and receiving the 200 OK response, and takes it as the measured application-layer *RTT*.

When receiving a probing message (request/response), the core server inserts it into the processing queue. If the load of the core server is higher, more time is spent on forwarding the received probing message. Thus the time spent on forwarding the probing message can indicate the core server's load. Similarly, the target server also inserts the probing request into the processing queue and the time spent on replying the 200 OK can indicate its load. Since the application-layer *RTT* includes the forwarding time in the core servers and the replying time in the target server, the increase of load either in core servers or target server can lead to the increase of application-layer *RTT*. Thus application-layer *RTT* is a good indicator of the SIP network load.

The prober module probes the SIP network periodically. At time *t* (at the end of each *T*), it sends a MESSAGE request with a Probing header in order to probe the SIP network and measure the current *RTT*. In the meanwhile, the prober module passes RTT_t to the controller module, which is the *RTT* at time *t* and reflects the current network load. However, due to the delay between sending the MESSAGE request and receiving its 200 OK response, the measured *RTT* is not available at the prober module at time *t*. Therefore, we should design a prediction algorithm in the prober module in order to predict RTT_t based on the historical measurements of *RTT*.

5.2. Prediction algorithm

The time series prediction is well investigated and many predictors have been proposed in control systems (Garroppo et al., 2011; Hayes, 1996; Haykin, 1991; Mishra et al., 1996; Adas, 1998; Garroppo et al., 2008). Among these predictors, the linear predictors can provide low complexity as well as high accuracy and responsiveness (Garroppo et al., 2008). Therefore, we adopt a linear predictor in PEOC.

The *k*-step linear predictor is concerned with the predication of z_{t+k} using a linear combination of the current and previous values of z_t (Hayes, 1996). A *p*th-order linear predictor has the form

$$\hat{z}_{t+k} = \sum_{l=0}^{p-1} w_l z_{t-l} \tag{6}$$

where w_l (for $l=0, 1, \ldots, p-1$) are the prediction filter coefficients. Let

$$\begin{aligned} &\underline{w} = [w_0, w_1, \ldots, w_{p-1}]^T \\ &\underline{z}_t = [z_t, z_{t-1}, \ldots, z_{t-p+1}]^T \\ &e_t = z_{t+k} - \hat{z}_{t+k} \end{aligned} \tag{7}$$

From (6) and (7),

$$e_t = z_{t+k} - \underline{w}^T \underline{z}_t \tag{8}$$

The filter coefficients can be determined according to arbitrary optimality criteria. One of the most famous and widely adopted prediction algorithms is the Linear Minimum Mean Square Error (LMMSE) predictor (Hayes, 1996; Haykin, 1991), in which the filter coefficients are derived by minimizing the Mean Square Error of prediction:

$$\text{Minimize}: \; E[e_t^2] \tag{9}$$

The problem of this predictor is that the derivation of the LMMSE filter coefficients requires the knowledge of the autocorrelation of $\underline{z}_t$ and the inversion of a $p \times p$ matrix. These facts make LMMSE unsuitable for being used as an on-line technique for predicting (Adas, 1998). In order to solve this problem, we consider the Normalized Least Mean Square (NLMS) predictor (Hayes, 1996; Haykin, 1991), which is based on an adaptive approach. It does not require prior knowledge of the autocorrelation structure of a sequence (Haykin, 1991; Adas, 1998). Therefore, it can be used as an on-line technique for predicting.

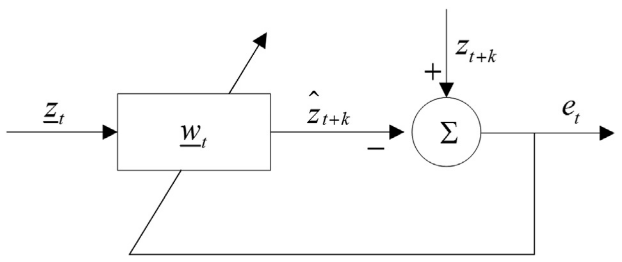

Fig. 5. NLMS algorithm scheme.

The operation of the NLMS predictor is shown in Fig. 5. The filter coefficients $\underline{w}_t$ are time-varying. The errors e_t are fed back and used to adapt the filter coefficients in order to decrease the mean square error. The NLMS operates as follows:

- Initialize the filter coefficients $\underline{w}_0 = 0$.
- For each new data (error e_t), update the filter coefficients $\underline{w}_t$ according to the following recursive equation:

$$\underline{w}_{t+1} = \underline{w}_t + \omega \frac{e_t \underline{z}_t}{\|\underline{z}_t\|^2} \tag{10}$$

where $\|\underline{z}_t\|^2 = \underline{z}_t^T \underline{z}_t$ and ω is a constant factor called step size. According to Haykin (1991), NLMS converges in the mean to LMMSE predictor if $0 < \omega < 2$.

Note that at time t, the value of z_{t+k} is not available to compute e_t. Thus e_{t-k} is used instead (Adas, 1998), and the one-step NLMS predictor update equation becomes:

$$\underline{w}_{t+1} = \underline{w}_t + \frac{\omega e_{t-1} \underline{z}_{t-1}}{\|\underline{z}_{t-1}\|^2} \tag{11}$$

In PEOC, we adopt this one-step NLMS predictor to predict RTT_t based on the historical measurements of RTT, in which we set the predictor step size ω and the predictor order p to 0.8 and 20, respectively (Garroppo et al., 2011). Let $\underline{w}_t = [w_{t,0}, w_{t,1}, \ldots, w_{t,19}]^T$ and $\underline{RTT}_t = [RTT_t, RTT_{t-1}, \ldots, RTT_{t-19}]^T$. Assume that $\overline{RTT}_t$ denotes the predicted RTT at time t. We have the following formulas:

$$\overline{RTT}_t = \sum_{l=0}^{19} w_{t,l} RTT_{t-1-l} = \underline{w}_t^T \underline{RTT}_{t-1} \tag{12}$$

$$e_{t-2} = RTT_{t-1} - \overline{RTT}_{t-1} = RTT_{t-1} - \underline{w}_{t-1}^T \underline{RTT}_{t-2} \tag{13}$$

$$\underline{w}_t = \underline{w}_{t-1} + 0.8 \frac{e_{t-2} \underline{RTT}_{t-2}}{\|\underline{RTT}_{t-2}\|^2} \tag{14}$$

The pseudo-algorithm for predicting RTT in PEOC is shown in Algorithm 1. We elaborate it as follows: during an interval T (e.g., the time interval between $t-1$ and t), when receiving the 200 OK response of the probing MESSAGE request sent at time $t-1$, we measure RTT_{t-1} and use (13) to calculate e_{t-2} based on the measured RTT_{t-1} and $\overline{RTT}_{t-1}$. According to (14), we then calculate the predictor filter coefficients $\underline{w}_t$ in order to predict $\overline{RTT}_t$. Finally by using (12), we predict $\overline{RTT}_t$ based on the calculated $\underline{w}_t$ and $\underline{RTT}_{t-1}$. At the end of the interval T (at time t), the prober module passes the predicted value $\overline{RTT}_t$ to the controller module.

Algorithm 1. The pseudo-algorithm for predicting RTT.

```
1. Receive a 200 OK (e.g., at time interval between $t-1$ and $t$);
2. $RTT_{t-1} \leftarrow$ current time-MESSAGE sending time;
3. $e_{t-2} \leftarrow RTT_{t-1} - \overline{RTT}_{t-1} = RTT_{t-1} - \underline{w}_{t-1}^T \underline{RTT}_{t-2}$
4. $\underline{w}_t \leftarrow \underline{w}_{t-1} + 0.8 \frac{e_{t-2} \underline{RTT}_{t-2}}{\|\underline{RTT}_{t-2}\|^2}$
5. $\underline{RTT}_{t-1} \leftarrow$ combine $RTT_{t-1}$ and $\underline{RTT}_{t-2}$
6. $\overline{RTT}_t \leftarrow \underline{w}_t^T \underline{RTT}_{t-1}$
```

6. Performance evaluation

6.1. Simulation environment

The simulation platform used is the NS-2 simulator (NS-2 Network Simulator). The proposed PEOC is implemented based on Prior's NS-2 SIP module (Prior). Two SIP network topologies shown in Fig. 2 are adopted in our simulations. These topologies are representatives of the topologies proposed in standards (e.g., the IMS architecture; 3rd Generation Partnership). Besides, they are commonly used in recent research (Hilt and Widjaja, 2008; Wang, 2010; Liao et al., 2012). A typical example of SIP call flow shown in Fig. 1 is adopted, and all SIP servers have the same processing capacities, which can process 200 messages per second, i.e., ~32 calls per second (cps). They use a round-robin (RR) scheme to balance the load whenever there are multiple next-hop servers available. Both edge servers and core servers receive a large number of calls from different UAs, and may be overloaded. Since a core server receives the traffic from multiple edge servers, it bears more load than edge server and is more likely to be overloaded.

The SIP servers are set to operate in transaction stateful mode (Rosenberg et al., 2002). UAs and SIP servers transmit messages via UDP, thus the reliability of message transmission is achieved by SIP retransmissions. Servers are set to record route, which causes all SIP messages exchanged between two UAs to traverse through these servers. The processing time of SIP servers spent on forwarding a probing message is set as 0.5 ms. We use 503 responses without Retry-After header recommended by Hilt et al. (2011) and Gurbani et al. (2013) to reject INVITE requests when a server's load gets close to its capacity limit. Considering that rejecting INVITE requests by sending 503 responses consumes processing resources, we set the processing time of SIP servers spent on sending a 503 response to be equal to that spent on processing any other message (except probing message). Besides, we adopt the early rejection method in SIP servers as suggested by Hilt and Widjaja (2008) to speed up the rejection process, in which INVITE requests are rejected before queuing them in the message buffer. Thus, in SIP servers, the INVITE request to be rejected and its 503 response are processed with high priority. All other SIP messages are served in a FIFO fashion.

Our experiment uses an infinite number of UAs and each new call is generated by a new UA instance. The calls arrive at each edge server according to Poisson process, and the destination of a call is randomly picked among the other edge servers according to uniform distribution. The holding time for an established call is assumed to be exponentially distributed with an average of 30 s. The offered load to the network is the total number of calls per second initiated by all UAs. In the following results, we use the goodput and call setup delay as performance metrics. The goodput is defined as the number of calls per second successfully established. A call is successfully established if the UA receives a 200 response within 10 s after the INVITE request is sent. The call setup delay is defined as the time between sending the initial INVITE request and receiving a 200 response.

Similar to previous end-to-end overload control research (Hilt and Widjaja, 2008; Wang, 2010; Liao et al., 2012), we compare the performance of PEOC with no overload control (No Control), local occupancy-based control (OCC-Local) and hop-by-hop occupancy-based control (OCC-Hop). Besides, the performance of PEOC is compared with DEOC proposed in Liao et al. (2012), which adopts a reactive algorithm to control overload for SIP networks. In No Control, the SIP server just drops messages if its buffer is full (i.e., drop-tail), and the buffer size is set to 100. In OCC-Local, each SIP server monitors its processor occupancy and calculates the acceptance probability, and then probabilistically rejects arriving calls

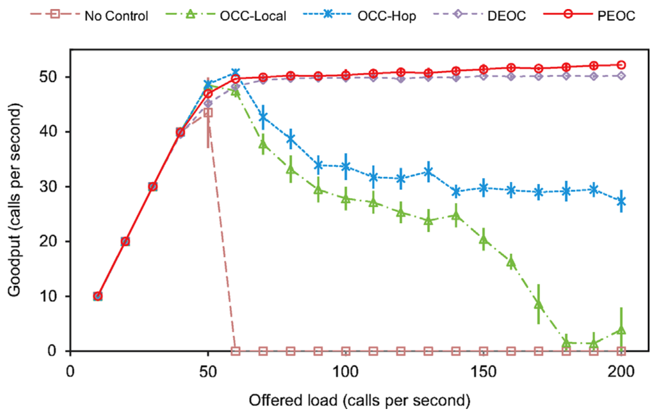

Fig. 6. Goodput comparison with varying offered load among different overload control mechanisms based on topology 1.

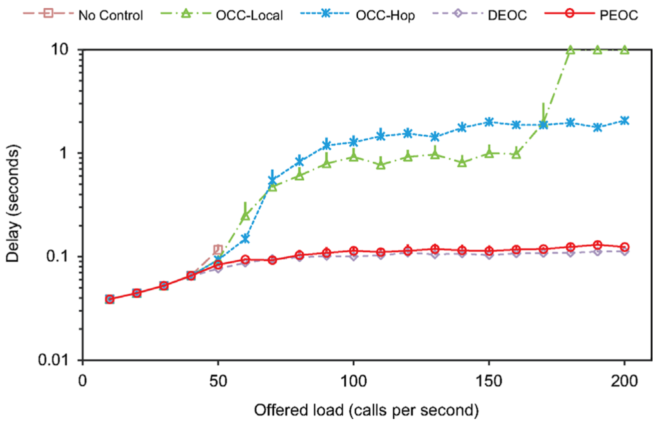

Fig. 7. Delay comparison with varying offered load among different overload control mechanisms based on topology 1.

based on the acceptance probability, which is described in Cyr et al. (1990). The parameters of the OCC algorithms are set the same as those in Hilt and Widjaja (2008). OCC-Hop is extended from OCC-Local, in which each server still monitors its processor occupancy and calculates the acceptance probability. However, the overloaded server sends the acceptance probability as the feedback to its direct upstream neighbors as specified in Gurbani et al. (2013). The direct upstream neighbors then execute the rejection in place of the overloaded server. In PEOC, we adopt OCC-Local as the local overload control on each server. The EWMA smoothing weight w is set to 0.1. The decreasing parameter β is set to 1/8 as used by Jain and Ramakrishnan (1988). The multiplicative increasing parameter δ is set to 0.1, the additive increasing parameter θ is set to 0.1 and the threshold T_{th} is set to 50 ms unless otherwise specified. The overload control mechanism is terminated if there is no call rejection received within 100 s and is restarted as soon as the call rejection is received. The execution periods of all controls including OCC-Local, OCC-Hop and PEOC are set to 1 s, i.e., T in PEOC is 1 s. The parameters of DEOC except the increasing parameters, which are set to the default values as suggested by Liao et al. (2012), have similar values as those of PEOC. The following experiments are performed in topology 2 of Fig. 2 unless otherwise specified. Each experimental value is averaged over 10 independent runs and the 95% confidence interval is calculated unless otherwise specified.

6.2. Performance results

In this section, we evaluate PEOC's performance in terms of goodput, aggressiveness, responsiveness and fairness. The experiment

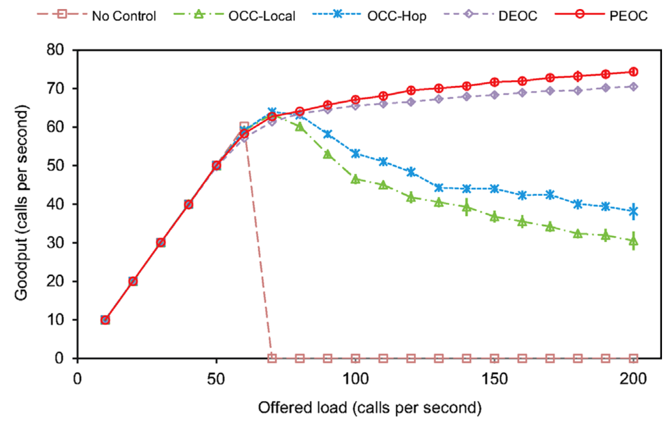

Fig. 8. Goodput comparison with varying offered load among different overload control mechanisms based on topology 2.

results prove that PEOC can keep high throughput even when the offered load exceeds the capacity of the network. Besides, it responds quickly to the sudden variations of the offered load and achieves good fairness.

6.2.1. Goodput and call setup delay

In this section, we evaluate the goodput and the call setup delay obtained from different overload control mechanisms under two network topologies. The network topology 1 and 2 are used in our first and second experiment, respectively and the results in both experiments are similar. The results of goodput and call setup delay in our first experiment are shown in Figs. 6 and 7, respectively and those in the second experiment are shown in Figs. 8 and 9, respectively.

As Figs. 6 and 8 show, when the offered load is within the capacity of the network, all mechanisms achieve comparable goodput. When the offered load goes beyond the capacity of the network, PEOC and DEOC outperform other mechanisms. Besides, PEOC achieves better performance than DEOC. We detail it as follows: (1) in No Control, the goodput rapidly drops to zero with a severe congestion collapse. It is because the simple drop-tail cannot relieve overload. Requests will be retransmitted if they are dropped, which amplifies the load on the overloaded server and eventually leads to congestion collapse of the network. (2) In OCC-Local, the goodput degrades approximately linearly as the offered load increases, which is explained as follows: the overloaded server rejects incoming calls by itself. Since the rejection also consumes the processing resources, the overloaded server spends more and more resources on rejecting incoming calls and thus fewer resources are left for serving calls as the offered load increases. (3) OCC-Hop performs better than OCC-Local under overload, because the task of rejection is shifted to upstream neighbors. Thus the overloaded server would not waste its resources on rejecting calls. However, this approach is still suboptimal. This is because overload is resolved near the overloaded server rather than close to the source of traffic, thus a lot of resources in SIP networks are wasted on processing a request that will finally be rejected. (4) DEOC and PEOC outperform other mechanisms. This is because the end-to-end overload controls throttle traffic at the edge of the network, thus minimum resources of SIP networks are wasted on processing a request that will finally be rejected. Besides, PEOC achieves better performance than DEOC. The reason is explained as follows: the DEOC is a reactive mechanism, which controls the overload only based on a binary overload feedback received from the network. On the other hand, the PEOC is a proactive mechanism, which controls the overload not only based on the received binary overload feedback,

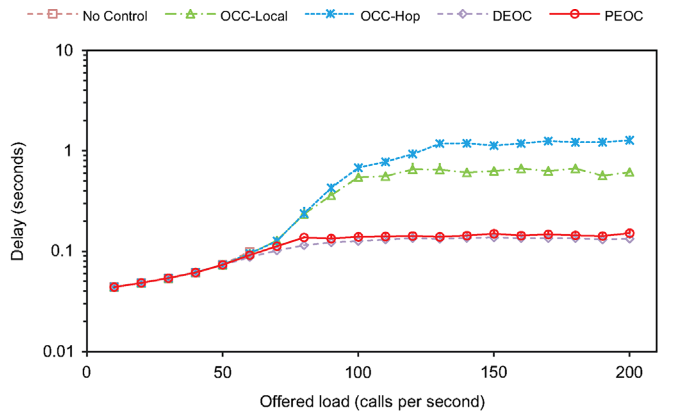

Fig. 9. Delay comparison with varying offered load among different overload control mechanisms based on topology 2.

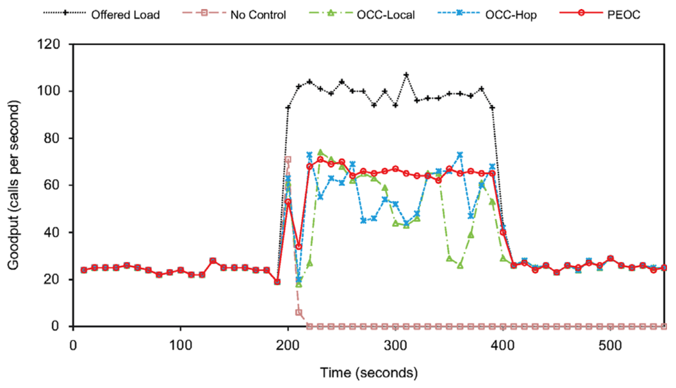

Fig. 10. Goodput comparison among different overload control mechanisms when the offered load is varied suddenly between overload and underutilization.

but also by probing the SIP network proactively. Thus PEOC can infer the network load more accurately and perform the overload control more efficiently.

Figs. 7 and 9 show the results of call setup delay. Note that for No Control case, the delay is plotted only up to 50 cps in the first experiment and 60 cps in the second experiment as no call can get through the network under overload. When the offered load is under the capacity of the network, all mechanisms ensure the same call setup delays. When the offered load increases above the capacity of the network, PEOC and DEOC perform up to an order of magnitude better than other mechanisms. Besides, their call setup delays are always at a low value (i.e., 0.1 s). In this case, the call setup delays do not increase significantly even when the network is under heavy offered load and thus the user experience is always assured.

6.2.2. *Aggressiveness and responsiveness*

In this section, we vary the offered load to the SIP network to study how fast the overload control mechanisms respond to the sudden load variations. Initially, the offered load to the network is 25 cps, which is below the capacity of the network. At 200 s, the offered load is increased to 100 cps suddenly, which is beyond the capacity of the network. At 400 s, the offered load is decreased back to 25 cps. We measure the instantaneous goodput and call setup delay every 10 s and show a sample path of goodput and call setup delay as a function of time in Figs. 10 and 11, respectively.

As expected, in No Control, the network suffers from congestion collapse under overload and can hardly recover from it even if the offered load is decreased below the capacity. OCC-Local, OCC-Hop and PEOC all respond quickly to the sudden load variations.

Table 1
Response time comparison between DEOC and PEOC.

	DEOC	PEOC
Response time (s) (95% confidence interval)	14.1 (12.63, 15.57)	5.3 (4.37, 6.23)

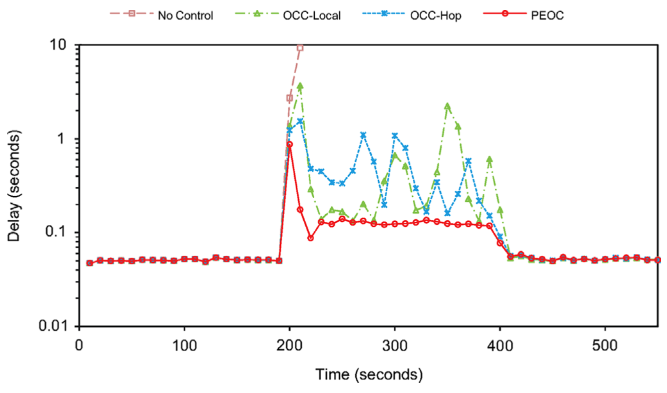

Fig. 11. Delay comparison among different overload control mechanisms when the offered load is varied suddenly between overload and underutilization.

Besides, PEOC achieves better goodput and call setup delay when the offered load goes beyond the capacity of the network.

In Fig. 10, we observe that there is a decrease just after an increase in the goodput curves of all mechanisms at 200 s when the offered load is increased suddenly. This is because in our experiment the control parameters (e.g., processor occupancy, acceptance probability, call gap interval, etc.) are updated once per second, but the offered load is increased immediately at 200 s. Therefore, at 200 s, a large number of new calls are admitted into the network before the control parameters are updated. Due to the call-oriented property of SIP, if the INVITE message is admitted, all subsequent messages belonging to the same call as the INVITE message should also be admitted. Thus, the network is overloaded after 200 s, and the overload controls of all mechanisms begin to decrease the call admission rate to eliminate overload, which causes the decrease of goodput. After the overload is eliminated, the goodput recovers.

As for PEOC, we observe that after a sudden increase of the offered load (i.e., after 200 s), the goodput drops immediately as shown in Fig. 10. Besides, the call setup delay is limited to a lower value (i.e., less than 1 s) under overload as shown in Fig. 11. This is mainly because in PEOC, the call admission rate is decreased more quickly when overload occurs, and thus few calls, which will amplify the overload of the network, are admitted. Therefore, PEOC has high responsiveness. After the overload is eliminated, the goodput increases immediately to the capacity of the network as shown in Fig. 10. Therefore, PEOC has high aggressiveness.

Table 1 shows the comparison of response time between DEOC and PEOC. In the following experiments, when comparing response time, we use the same simulation setup as in this section. The response time in the simulation is defined as the time (i.e., the number of periods and the period is 1 s in the experiment) that the network spends on increasing goodput to its capacity after overload is eliminated (after 200 s). In our simulation, the beginning of response time is at the time when the processor occupancy of each core server is less than 50% (i.e., the goodput starts to increase after overload is eliminated). The end of response time is at the time when goodput increases to the network's capacity. We can see that our defined response time indicates aggressiveness, in which shorter response time means higher aggressiveness.

From Table 1, we can observe that the response time in DEOC is longer than that in PEOC. This is because PEOC probes the network proactively and thus can infer the network load in a timely manner. Therefore, PEOC has higher aggressiveness than DEOC.

6.2.3. *Fairness*

Fairness is an important performance metric for distributed rate control algorithms. In this section, we define an admitted call flow as the admitted calls that have the same ingress server and the same target server. We investigate the goodput of each admitted call flow, whose ingress server is in domain A and target server is in domain B, to study the performance of different overload control mechanisms in terms of fairness. We choose these admitted call flows to evaluate the fairness as they have the same bottleneck (Bertsekas and Gallager, 1987) in the SIP network. In the experiment, we measure the goodput of each admitted call flow when the offered load is 50, 100, 150 and 200 cps and then calculate Jain's fairness index (Jain et al., 1984), which is defined as follows:

$$f = \frac{(\sum_{i=1}^{k} x_i)^2}{k(\sum_{i=1}^{k} x_i^2)}$$

The fairness index f considers k admitted call flows where the goodput of flow i is x_i. f is between 0 and 1, where 1 is completely fair (all flows share the bottleneck resource equally). Fig. 12 shows the results in terms of the fairness index.

When the offered load is within the capacity of the network (i.e., the offered load is 50 cps), all mechanisms have good fairness since there is no call rejection. When the offered load goes beyond the capacity of the network (i.e., the offered load is 100, 150 and 200 cps), in No Control case, the index is not plotted as no call can get through the network under overload. All other mechanisms have good fairness (all indexes are above 0.98). Note that when the offered load is high (150, 200 cps), OCC-Local and OCC-Hop have slightly better fairness than PEOC and DEOC. We explain it as follows: both OCC-Local and OCC-Hop control overload near the overloaded server. Thus they have better knowledge of the overload. However, both PEOC and DEOC infer the overload at the edge of the network. This inference may not be as accurate or timely as that of OCC-Local and OCC-Hop. Even so, PEOC can still achieve good fairness.

6.3. *Parameter tuning*

6.3.1. *Effect of parameter δ*

The parameter δ determines the increasing rate of call admission rate when the network load is low. According to the analysis in Section 4, a smaller δ leads to a higher goodput and a lower aggressiveness. Fig. 13 shows the goodput obtained from PEOC with $\delta=0.05$, $\delta=0.1$, $\delta=0.2$ and $\delta=0.5$ under different offered load. When the offered load goes beyond the capacity of the network (i.e., the offered load is 100, 150 and 200 cps), the PEOC with minimum δ has the highest goodput and the PEOC with maximum δ has the lowest goodput. Table 2 shows the response time of PEOC with different δ. We can see that the PEOC with minimum δ has the longest response time and the PEOC with maximum δ has the shortest response time. Thus the larger δ leads to lower goodput and higher aggressiveness. This conclusion validates our analysis of δ in Section 4. From Table 2 we can see that even a small δ can achieve high aggressiveness, thus in practice δ should be set as a small value in order to keep high throughput.

6.3.2. *Effect of parameter θ*

The parameter θ is the increasing step size of call admission rate when the network load is high. According to the analysis in Section 4, a smaller θ leads to a higher goodput and a lower aggressiveness. Fig. 14 shows the goodput obtained from PEOC with various θ under different offered load. When the offered load goes beyond the capacity of the network, the PEOC with smaller θ has higher goodput. The response time of PEOC with different θ is

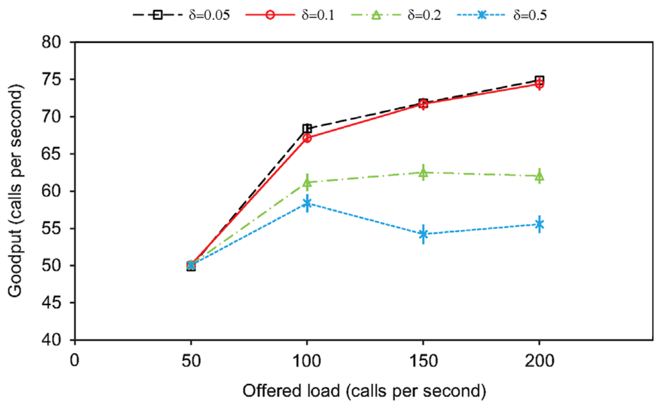

Fig. 13. Goodput comparison with varying offered load among PEOC with $\delta=0.05$, $\delta=0.1$, $\delta=0.2$ and $\delta=0.5$.

Table 2
Response time comparison among PEOC with $\delta=0.05$, $\delta=0.1$, $\delta=0.2$ and $\delta=0.5$.

	$\delta=0.05$	$\delta=0.1$	$\delta=0.2$	$\delta=0.5$
Response time (s)	10.1	5.3	2.4	1.3
(95% confidence interval)	(8.36, 11.84)	(4.37, 6.23)	(2.08, 2.72)	(1.00, 1.60)

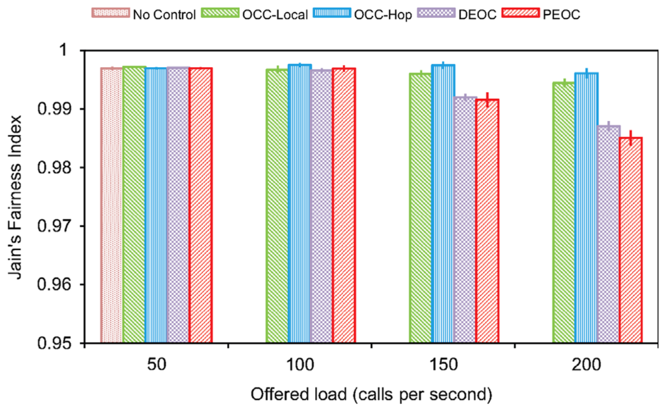

Fig. 12. Jain's fairness index comparison among different overload control mechanisms when the offered load is 50, 100, 150 and 200 cps.

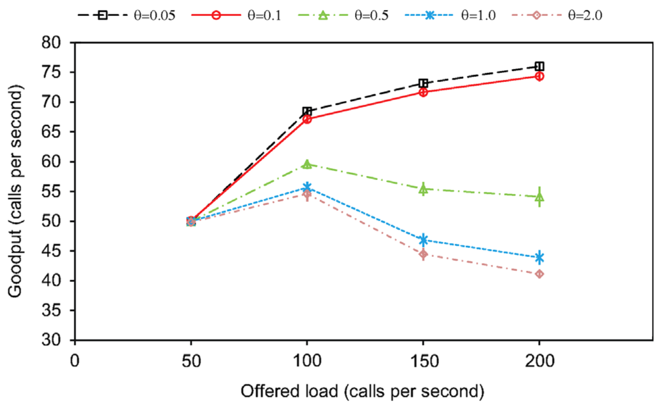

Fig. 14. Goodput comparison with varying offered load among PEOC with $\theta=0.05$, $\theta=0.1$, $\theta=0.5$, $\theta=1.0$ and $\theta=2.0$.

Table 3
Response time comparison among PEOC with $\theta=0.05$, $\theta=0.1$, $\theta=0.5$, $\theta=1.0$ and $\theta=2.0$.

	$\theta=0.05$	$\theta=0.1$	$\theta=0.5$	$\theta=1.0$	$\theta=2.0$
Response time (s) (95% confidence interval)	9.7 (8.27, 11.13)	5.3 (4.37, 6.23)	3.5 (2.90, 4.10)	2.1 (1.56, 2.64)	1.6 (1.17, 2.03)

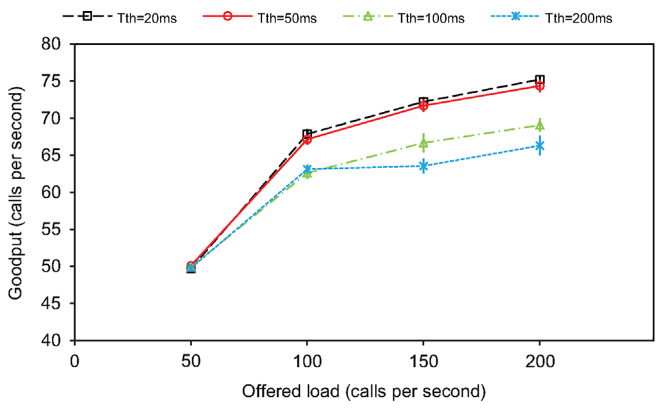

Fig. 15. Goodput comparison with varying offered load among PEOC with $T_{th}=20$ ms, $T_{th}=50$ ms, $T_{th}=100$ ms and $T_{th}=200$ ms.

Table 4
Response time comparison among PEOC with $T_{th}=20$ ms, $T_{th}=50$ ms, $T_{th}=100$ ms and $T_{th}=200$ ms.

	$T_{th}=20$ ms	$T_{th}=50$ ms	$T_{th}=100$ ms	$T_{th}=200$ ms
Response time (s) (95% confidence interval)	7.2 (6.29, 8.11)	5.3 (4.37, 6.23)	2.9 (2.16, 3.64)	1.5 (1.06, 1.94)

shown in Table 3. We can see that the PEOC with smaller θ has longer response time. Therefore, the smaller θ, the higher goodput and the lower aggressiveness. This conclusion validates our analysis of θ in Section 4. From Table 3 we can see that even a small θ can achieve high aggressiveness, thus in practice it is suitable to set θ as a small value in order to keep high throughput.

6.3.3. *Effect of parameter T_{th}*

The parameter T_{th} is the threshold value that determines whether the network load is high or low. According to the analysis in Section 4, a smaller T_{th} leads to a higher goodput and a lower aggressiveness. Fig. 15 shows the goodput obtained from PEOC with various T_{th} under different offered load. When the offered load goes beyond the capacity of the network, the PEOC with smaller T_{th} has higher goodput. Table 4 shows the response time of PEOC with different T_{th}. We can see that the PEOC with smaller T_{th} has longer response time. Therefore, the smaller T_{th}, the higher goodput and the lower aggressiveness. This conclusion validates our analysis about T_{th} in Section 4. In practice, we may choose a proper threshold T_{th} to balance the performance between goodput and aggressiveness according to our needs.

7. Conclusion and future work

With the increasing popularity and rapidly growing deployments of SIP, the issue of overload control in SIP networks becomes more and more important. Compared to the traditional Local and Hop-by-hop overload controls, the end-to-end overload control can better utilize network resources and improve the throughput when the offered load exceeds the capacity of the network. The current existing end-to-end overload control solutions can be classified into receiver-based and sender-based control. The receiver-based mechanisms (Hilt and Widjaja, 2008; Wang, 2010) control the overload based on the complicated cooperation among edge servers and core servers, which are too complex to be practical. On the other hand, the sender-based mechanism DEOC (Liao et al., 2012) controls the overload based on the simple binary overload feedback without the need of the complicated cooperation among servers and thus is practical and easy to implement.

In this paper, we proposed PEOC, which is a probe-based end-to-end overload control mechanism for SIP networks. Similarly to DEOC (Liao et al., 2012), PEOC controls the overload based on the simple binary overload feedback. Besides, it estimates the network load by probing the SIP network with SIP messages and thus can control the overload of the network in a more efficiently and timely manner compared with DEOC. We presented the design of the proposed approach, and evaluated its performance through simulation experiments. Our simulation results demonstrated that PEOC can keep high throughput even when the offered load exceeds the capacity of the network. Besides, it responds quickly to the sudden variations of the offered load and achieves good fairness.

In our future work, we plan to implement PEOC and evaluate its performance in real networks. In real networks, some edge servers may not deploy PEOC as they are not under the control of the carrier. If only some of the edge servers deploy PEOC, it will lead to the unfairness between the edge servers with PEOC and those without PEOC when overload occurs in the core servers. This is because the edge servers with PEOC decrease the call admission rates in response to overload, while the edge servers without PEOC are not cooperative and do not decrease the call admission rates. Therefore, we need to design and implement a mechanism in core servers that can guarantee the fairness between all edge servers, when only some of the edge servers deploy PEOC.

Acknowledgments

This work was jointly supported by: (1) the National Basic Research Program of China (No. 2013CB329102); (2) National Natural Science Foundation of China (Nos. 61372120, 61271019, 61101119, 61121001, 61072057, and 60902051); (3) PCSIRT (No. IRT1049); (4) Spanish Research Council (MICCIN) under Project no. TIN2010-19077; and (5) Madrid Research Foundation (CAM) under Project no. S2009TIC-1692 (cofunded by ERDF and ESF).

References

3rd Generation Partnership Project. ⟨http://www.3gpp.org⟩.
Abdelal A, Matragi W. Signal-based overload control for SIP servers. In: Proceedings of IEEE CCNC; January 2010. p. 1–7.
Adas AM. Using adaptive linear prediction to support real-time VBR video under RCBR network service model. IEEE Trans Networking 1998;6(5):635–44.
Allman M, Paxson V, Blanton E. TCP congestion control, IETF RFC 5681; September 2009.
Azhari SV, Homayouni M, Nemati H, Enayatizadeh J, Akbari A. Overload control in SIP networks using no explicit feedback: a window based approach. Comput Commun 2012;35(12):1472–83.

Berger A. Comparison of call gapping and percent blocking for overload control in distributed switching systems and telecommunications networks. IEEE Trans Commun 1991;39(4):574–80.

Bertsekas D, Gallager R. Data networks. Prentice Hall; 1987.

Chiu D-M, Jain R. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. Comput Networks ISDN Syst 1989;17 (1):1–14.

Cyr BL, Kaufman JS, Lee PT .Load balancing and overload control in a distributed processing telecommunication systems, United States Patent No. 4,974,256; 1990.

Garroppo RG, Giordano S, Pagano M, Procissi G. On traffic prediction for resource allocation: a Chebyshev bound based allocation scheme. Comput Commun 2008;31(16):3741–51.

Garroppo RG, Giordano S, Niccolini S, Spagna S, Prediction-Based A. Overload control algorithm for SIP servers. IEEE Trans Network Serv Manage 2011;8 (1):39–51.

Garroppo RG, Giordano S, Spagna S, Niccolini S. Queueing strategies for local overload control in SIP server. In: Proceedings of IEEE GLOBECOM; December 2009. p. 1–6.

Gurbani V, Hilt V, Schulzrinne H. Session Initiation Protocol (SIP) overload control, draft-ietf-soc-overload-control-13, IETF, Work in Progress; May 2013.

Hayes M. Statistical digital signal processing and modeling. Wiley; 1996.

Haykin S. Adaptive filter theory. Prentice Hall; 1991.

Hilt V, Widjaja I. Controlling overload in networks of SIP servers. In: Proceedings of IEEE ICNP; October 2008. p. 83–93.

Hilt V, Noel E, Shen C, Abdelal A. Design considerations for Session Initiation Protocol (SIP) overload control, IETF RFC 6357; August 2011.

Hong Y, Huang C, Yan James. Mitigating SIP overload using a control-theoretic approach. In: Proceedings of IEEE GLOBECOM; December 2010. p. 1–5.

Jacobson V. Congestion avoidance and control. In: Proceedings of ACM SIGCOMM; 1988. p. 314–329.

Jain R, Ramakrishnan KK. Congestion avoidance in computer networks with a connectionless network layer: concepts, goals and methodology. In: Proceedings of the computer networking symposium; 1988. p. 134–143.

Jain R, Chiu D-M, Hawe W. A quantitative measure of fairness and discrimination for resource allocation in shared systems, digital equipment corporation. Technical report TR-301; September 1984.

Kasera S, Pinheiro J, Loader C, Karaul M, Hari A, LaPorta T. Fast and robust signaling overload control. In: Proceedings of IEEE ICNP; November 2001. p. 323–331.

Liao J, Wang J, Li T, Wang J, Zhu X. A token-bucket based notification traffic control mechanism for IMS presence service. Comput Commun 2011;34(10):1243–57.

Liao J, Wang J, Li T, Wang J, Wang J, Zhu X. A distributed end-to-end overload control mechanism for networks of SIP servers. Comput Networks 2012;56 (12):2847–68.

Mishra PP, Kanakia H, Tripathi SK. On hop-by-hop rate-based congestion control. IEEE Trans Networking 1996;4(2):224–39.

NS-2 Network Simulator. ⟨http://www.isi.edu/nsnam/ns/⟩.

Noel EC, Johnson CR. Initial simulation results that analyze SIP based VoIP networks under overload. In: Proceedings of the international teletraffic congress; June 2007. p. 54–64.

Ohta M. Overload control in a SIP signaling network, enformatika transactions on engineering. Comput Technol March 2006a: 205–210.

Ohta M. Overload protection in a SIP signaling network. In: Proceedings of IEEE ICISP; August 2006b.

Prior R. SIP module for NS-2. ⟨http://www.dcc.fc.up.pt/~rprior/ns/index-en.html⟩.

Rosenberg J. Requirements for management of overload in the session initiation protocol, IETF RFC 5390; December 2008.

Rosenberg J, Schulzrinne H, Camarillo G, Johnston A, Peterson J, Sparks R, et al., SIP: Session Initiation Protocol, IETF RFC 3261; June 2002.

Shen C, Schulzrinne H, Nahum E. Session Initiation Protocol (SIP) server overload control: design and evaluation. In: Proceedings of IPTComm; July 2008. p. 149–173.

Wang Y. SIP overload control: a backpressure-based approach. In: Proceedings of ACM SIGCOMM (poster); August 2010. p. 399–400.