A distributed end-to-end overload control mechanism for networks of SIP servers

Jianxin Liao [a], Jinzhu Wang [a], Tonghong Li [b], Jing Wang [a], Jingyu Wang [a], Xiaomin Zhu [a]

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

A B S T R A C T

The Session Initiation Protocol (SIP) is an application-layer control protocol standardized by the IETF for creating, modifying and terminating multimedia sessions. With the increasing use of SIP in large deployments, the current SIP design cannot handle overload effectively, which may cause SIP networks to suffer from congestion collapse under heavy offered load. This paper introduces a distributed end-to-end overload control (DEOC) mechanism, which is deployed at the edge servers of SIP networks and is easy to implement. By applying overload control closest to the source of traffic, DEOC can keep high throughput for SIP networks 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. Theoretic analysis and extensive simulations verify that DEOC is effective in controlling overload of SIP networks.

1. Introduction

The Session Initiation Protocol (SIP) [1] is an application-layer control protocol standardized by the IETF for creating, modifying, and terminating sessions for various types of media, including voice, video and text. It serves as a foundation for many of today's session-oriented applications, such as Voice over IP (VoIP), multimedia distributions, video conferencing, instant messaging and presence service [2]. In addition, the SIP has been adopted by 3GPP as the core control protocol 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. It is clear that both ingress server and target server are edge servers.

The widespread popularity and rapidly growing deployments of SIP require that SIP servers provide adequate control mechanisms to handle overload. Overload of a SIP

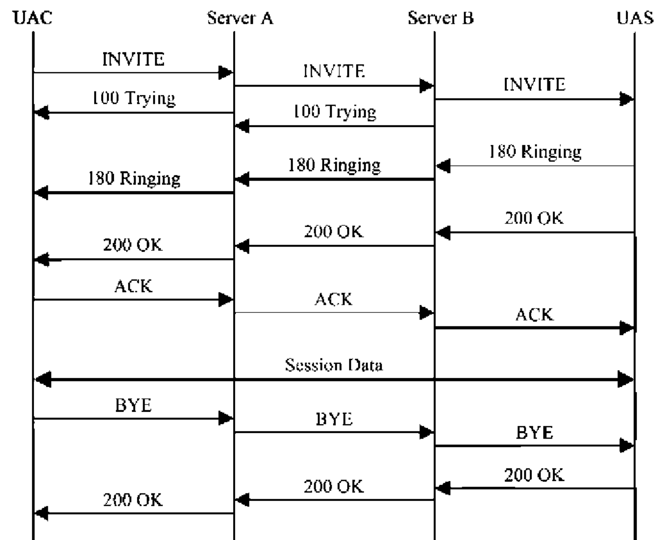
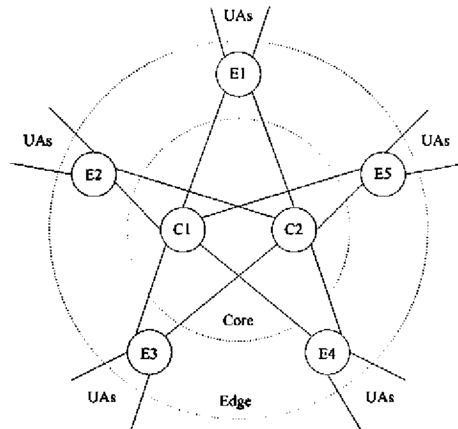


Fig. 1. Sample SIP call flow.



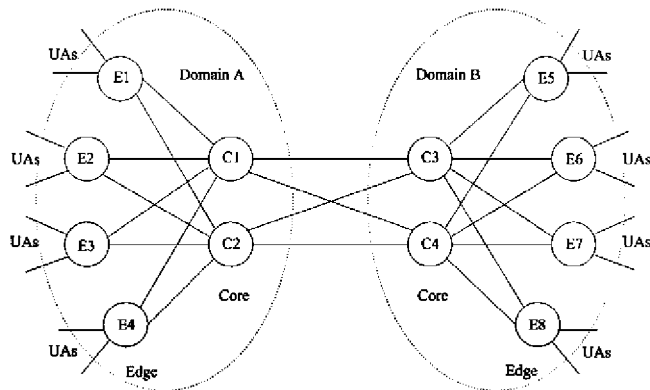


Fig. 2. Typical SIP network topologies.

server occurs if the message arrival rate to the server exceeds its message processing capacity. 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. In this case, the server enters into a congestion collapse.

Unfortunately, the overload problem tends to compound itself. When a SIP server goes into overload, its overall capacity goes down since most of its processing resources are devoted to rejecting or treating load that it cannot actually process. Furthermore, due to the retransmission mechanism in SIP, 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.[1] 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.

The network of SIP servers is typically designed and engineered to user demands. However, it is not economical and often not possible to dimension a SIP network for extreme load peaks, which can lead to overload in SIP servers. These sudden increases of load can be caused by various reasons, e.g., a swarm of users trying to initiate a SIP call around the same time when voting for a TV show or in an emergency situation, or simultaneous registrations of many users due to recovery after a large power outage or a misconfiguration by the service provider [3].

Since overload in SIP servers cannot fully be avoided, it is crucial to equip the SIP with a mechanism that can effectively manage overload [3]. A really simple method, that a SIP server drops requests silently when it is overloaded, is defined in the current SIP specification [1]. Since requests will be retransmitted if dropped, the load of the overloaded server is not relieved but amplified. Simulation study shows that this simple method can cause congestion collapse and the server has difficulty in recovering from that congestion collapse even if the offered load later gets reduced below its capacity [4]. Thus we need to explicitly reject requests rather than silently discard them in order to control the load in SIP servers. Furthermore, the SIP specification provides support for handling overload through 503 (Service Unavailable) responses [1]. SIP servers that are unable to forward a request due to temporary overload can explicitly reject the request with a 503 response. The overloaded server can insert a Retry-After header into the 503 response, which defines the time during which this server does not want to receive any further request from the upstream neighbor so that it can process its backlog of work. With the introduction of the Retry-After header, the server receiving a 503 response from a downstream neighbor stops forwarding requests to this neighbor for a period of time specified in the Retry-After header. As the server alternates between not forwarding and forwarding requests, the forwarded traffic may have an ON/OFF pattern, which can deteriorate the performance of the SIP networks. During the period when the upstream server is unable to forward requests to the overloaded server, it may attempt to forward these requests to an alternative server if available. Retrying at alternate servers can cause traffic oscillation among the servers [3,4]. On the other hand, without the Retry-After header, a 503 response only affects the current request and all other requests can still be forwarded to the downstream neighbor. Since each request is rejected individually, the ON/OFF pattern is avoided. Simulation study indicates that it provides better performance to adopt 503 responses without Retry-After header [4].

In order to fully manage the overload in SIP networks, many approaches, which are classified into local, hop-by-hop and end-to-end overload control, are proposed in the literature [5]. The basic idea of local overload control is that a SIP server starts to reject requests locally by using 503 responses without Retry-After header when it detects overload. Since the requests are explicitly rejected, they will not be retransmitted and thus the load of the overloaded server will not be amplified. Furthermore, local overload control does not require the cooperation between servers. However, this approach only performs well in light overload. Under heavy offered load, an overloaded SIP server with local overload control will eventually spend most of its processing resources on rejecting requests, which leads to poor throughput and unacceptable call setup delay.

Hop-by-hop overload control enables the overloaded SIP server to 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 [6]. Since the overload control loop is within one hop, hop-by-hop overload control is simple and scalable. On the other hand, this type of overload control performs better than local overload control as it makes sure the upstream neighbors only send the amount of traffic that the downstream neighbor can handle at all times. In this ideal situation, there is no message retransmission due to timeout and no extra processing cost in the overloaded server due to rejection. However, the overload control works best only if applied closest to the source of traffic because in this way minimum resources of SIP networks are wasted on processing a request that will finally be rejected. Thus hop-by-hop overload control is still inefficient as overload is resolved near the overloaded sever rather than close to the source of traffic.

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. Since there are often multiple SIP hops between the edge server and the overloaded server, the challenges of end-to-end control are how to inform the edge server which server is overloaded and how the edge server could figure out whether a specific request will be routed through the overloaded server. Thus this type of overload control is more complex than hop-by-hop overload control. On the other hand, end-to-end overload control provides best performance as it throttles the traffic at the edge of the network, minimizing the resources wasted on processing a request that will finally be rejected.

[1] Note that the retransmissions can also happen for SIP over TCP, in which many SIP retransmission timers are not activated. In this case, the UA retransmits the 200 OK response when the corresponding ACK message is not received in time.

In this paper, we propose and design DEOC, a distributed end-to-end overload control mechanism. It is easy to implement. By applying overload control closest to the source of traffic, DEOC eliminates unnecessary resource consumption and keeps high throughput for SIP networks under heavy offered load. The remainder of this paper is organized as follows: Section 2 surveys related work. Section 3 proposes the design of DEOC, and Section 4 analyzes the Rate Adaption (RA) algorithm, which can dynamically adjust the rate of calls admitted and is the essence of the DEOC. In Section 5, simulation models are described, and performance results are presented. Finally, conclusions and possible future work are presented in Section 6.

2. Related work

The research on SIP overload control has attracted much attention recently. Ejzak et al. [7] articulate the need for overload control in SIP and qualitatively discuss the similarities and differences between ISUP/SS7 and SIP networks. Nahum et al. [8] report the experimental results on SIP servers with different configurations, showing that overload has a bad effect on the performance of a SIP server. In [3,4], the overload problem of SIP and the ineffectiveness of its built-in overload control mechanisms are studied in detail.

2.1. Local overload control

As the first step to solve overload in SIP networks, several local overload control mechanisms have been proposed. In [9], a simple bang-bang control (BBC) algorithm is proposed to control overload locally, which defines two thresholds: the high threshold and the low threshold. The requests are rejected by a SIP server when the message queue length exceeds the high threshold, and are admitted to a SIP server when the message queue length falls below the low threshold. Hilt and Widjaja [4] further adopt an occupancy (OCC) algorithm proposed by [10] for local overload control in a SIP server. The OCC is based on processor occupancy and has the objective of dynamically adjusting call acceptance probability to maintain the processor occupancy not exceeding a given target threshold. Besides, [4] also suggests other possible algorithms (e.g., SRED and ARO) proposed in [11] for local overload control. In [12], an approach of using a priority queuing instead of a single queue is proposed, in which the INVITE messages are assigned to low priority while the other messages are assigned to high priority. This approach can protect the SIP server from performance degradation under overload. In [13], the performance of priority queuing methods is analyzed and compared with other queuing structures and service disciplines.

2.2. Hop-by-hop overload control

As local overload control only performs well in light overload, recent research focuses on various 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 overload 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. Hilt and Widjaja [4] present a loss-based feedback to control overload. The overloaded server calculates the call acceptance probability based on its processor occupancy and takes this probability as the feedback. Noel and Johnson [14] adopt a rate-based feedback where the overloaded server calculates call admission rate based on its estimated message queuing delay and takes this rate as the feedback. In [15], a window-based feedback is proposed where the overloaded server calculates the window value, which represents a certain number of admitted requests that have not been confirmed by responses, based on its estimated message queuing delay and takes the window value as the feedback. Using some special attributes of SIP, Garroppo et al. [16] further extend the mechanisms in [14,15] to estimate the message queuing delay more accurately. Besides, they propose an algorithm based on the prediction of feedback values in order to control overload more efficiently. On the other hand, in sender-based overload 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 [17] propose a sender-based overload control mechanism, where the upstream neighbor multiplicatively reduces the traffic forwarded to the overloaded server after receiving a 503 response, and additively increases the forwarded traffic when not receiving 503 responses for a while. The forwarded traffic is increased until another 503 response is received.

Note that normally receiver-based overload control responds to input traffic variations more quickly than sender-based overload control. This is because in receiver-based overload control, the overloaded server calculates restrictions directly based on its load; while in sender-based overload control, the upstream neighbors calculate restrictions indirectly based on the load of the overload server, which is estimated from the received 503 responses. However, receiver-based overload control is more complex than sender-based overload control as receiver-based overload control adds extra burden on the overloaded server to calculate restrictions and then distribute these restrictions to its direct upstream neighbors.

2.3. End-to-end overload control

Since hop-by-hop overload control is still inefficient, end-to-end overload control begins to attract research attention recently. Hilt and Widjaja [4] propose an end-to-end overload control mechanism: each server in the SIP network calculates and maintains restriction on its offered load. Furthermore, each core server also maintains a list of restrictions of all target servers. The target server sends its restriction to direct upstream neighbors. When a core server receives a restriction related to a specific target

server, the corresponding item in the list of restrictions is updated. Besides, the core server calculates a new restriction based on the list and its own restriction, and then forwards the new one to its direct upstream neighbors. In this way, the restriction for each target server is eventually propagated to all ingress servers. An ingress server makes the decision on whether or not to accept new requests for a specific target server based on its restriction. Simulation result shows that this approach is more efficient than hop-by-hop overload control. However, as the authors point out, the main disadvantage of this mechanism is its great complexity: each core server needs to manage restrictions related to all target servers and cooperate with other servers in the SIP network to propagate restrictions. Therefore, this approach is impractical. In [18], a backpressure-based overload control (Bassoon) mechanism inspired by the theoretic works in stochastic network optimization is proposed. The authors design the operation of the SIP network including scheduling, load balancing and overload control. When a server is overloaded, the scheduling in Bassoon propagates the overload to the direct upstream neighbors and eventually the overload arrives at all ingress servers, which then start to restrict the forwarded traffic. Simulation result shows that using Bassoon, the load in the SIP network is well balanced and the overload can be efficiently controlled. However, this approach still suffers from several drawbacks. Firstly, when a server is overloaded, all servers between the ingress server and the overloaded server will be overloaded before the overload eventually arrives at the ingress server. This aspect causes unnecessary overload in the SIP network and increases the call setup delay of the admitted request. Secondly, Bassoon needs each server to exchange queue backlog with its neighbors in a short time, which results in remarkable overhead in the SIP network. Finally, in order to control overload, Bassoon requires all SIP servers in a network to adopt a specific scheduling and load balancing policy. This is impossible since a mass of SIP servers have already been deployed. Therefore, Bassoon is also impractical.

In fact, similar to hop-by-hop overload control, end-to-end overload control can also be classified into receiver-based and sender-based overload control. Obviously, both [4,18] are receiver-based. 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 complex cooperation (e.g., calculation of restriction in [4] and specific scheduling in [18]) among them. Thus receiver-based end-to-end overload control is even more complex than receiver-based hop-by-hop overload control. Therefore, in order to design a simple end-to-end overload control mechanism, we should adopt sender-based overload control. Our work is motivated by this conclusion. The main contribution of this paper is to design and implement a simple and practical end-to-end overload control mechanism, where a core server only implements local overload control that rejects requests by using 503 responses. It is the edge server's responsibility to calculate and then follow the restriction on forwarded traffic according to received 503 responses.

3. End-to-end overload control

3.1. End-to-end overload control design principles

There are some fundamental principles proposed in the previous works [4,18] for designing end-to-end SIP overload control. We follow these principles in our proposed distributed end-to-end overload control mechanism (DEOC), which are described as follows:

- Overload control only deals with INVITE requests. As SIP is call-oriented, we care about how many calls are successfully established and it makes no sense to reject/drop messages of an on-going call.
- Overload is controlled at ingress servers. That is, arriving calls from UAs are throttled at ingress servers. Overload control works best if applied at the servers closest to the source of traffic because in this way minimum resources of SIP networks are wasted on processing a request that will finally be rejected.
- Overload is controlled on a per-target basis. That is, each ingress server throttles the arriving call from UAs based on its target server. Without the per-target basis, an ingress server should identify which server is overloaded and throttle arriving calls that will be routed through the overloaded server. On the other hand, with the per-target basis, an ingress server only needs to identify which target server the call passing through the overloaded server is related to, and throttle arriving calls that will be forwarded to this target server. Thus per-target basis makes it much easier for ingress servers to control overload for the SIP network.[2]

Note that end-to-end overload control is essentially edge-to-edge overload control. That is, each ingress server (i.e., edge server) controls overload based on per-target server (i.e., per-edge server).

3.2. DEOC design

We deploy a set of DEOCs at each ingress server to control overload for the SIP network. At an ingress server, each

[2] The per-target basis is necessary because the paths starting from the same ingress server and ending at different target servers are different. For example, in the topology 2 of Fig. 2, the path from E1 to E2 is different from the path from E1 to E5. Therefore, when overload occurs at C3 and C4, E1 detects overload in the path from E1 to E5 and throttles the traffic forwarded to E5. However, it should not throttle traffic forwarded to E2. Therefore, the ingress server should throttle traffic on a per-target basis. Note that there may be multiple paths between an ingress server and a target server. In this case, it is more efficient to perform overload control on a per-path basis. However, the per-path control requires the ingress server to identify which path each arriving call from UAs will pass through and it is difficult for the ingress server to identify such information [5], because multiple paths are caused by the local policies (such as failover and load balancing) of servers in SIP networks and these policies can be highly varied [19,20]. Thus, in this case, overload control is still performed on a per-target basis.

DEOC 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 DEOCs at all ingress servers. Note that our approach is completely distributed: there is no centralized entity to control DEOCs and each DEOC is functionally identical and operates independently. Besides, there is no communication between DEOCs. Finally, our approach can be deployed incrementally, by installing DEOCs 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 DEOC can be split into three separate parts: measurement, restriction and control decision. Fig. 3 shows the functional modules in DEOC. The solid line and dashed line represent the data flow and the control flow, respectively. The arriving call at the DEOC 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.

3.2.1. 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 as follows:

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

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 DEOC 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 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.

In the restrictor module, we adopt call gapping [21] 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 arriving process of calls conforms to Poisson distribution. Referring to [21], 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.

3.2.2. Control decision

The main function of DEOC is to decide r. Since each DEOC manages calls related to a specific ingress server and a specific target server, the admitted traffic of the SIP network is determined by the r of all DEOCs. The network is underutilized if r is too small, and is overloaded if r is too large. Therefore, how to design an algorithm in DEOC to calculate proper r is essential for the SIP network. In the following, we design a Rate Adaption (RA) algorithm that can dynamically adjust the call admission rate r.

The RA consists of an increasing rule and a decreasing rule. When there is no overload feedback, RA increases r according to the increasing rule. When receiving the overload feedback, RA decreases r according to the decreasing rule. We will elaborate on RA in Section 4. The controller module periodically executes RA (with interval T) and takes the number of received 503 responses during each T as the overload feedback to RA.

The rate of receiving 503 responses is a good overload feedback because it reflects the load of each server in the route that the calls controlled by the DEOC pass through. As the load of one or more servers in that route increases, the local overload controls of these servers reject more calls by using 503 responses, and thus the DEOC receives more 503 responses as well. This is because the server receiving a 503 response will forward this response to the upstream neighbor, from which the INVITE request related to this response has been received. In this way, when a call is rejected in a SIP network, its 503 response will be finally forwarded to its ingress server. On the other hand, as SIP allows the overloaded server to reject a call by dropping the request and not generating a 503 response, DEOC should also consider the lack of response as an implicit signal of overload. That is, the DEOC should start a timer each time an INVITE request is sent and cancel the timer if the response is received. The expiration of the timer should

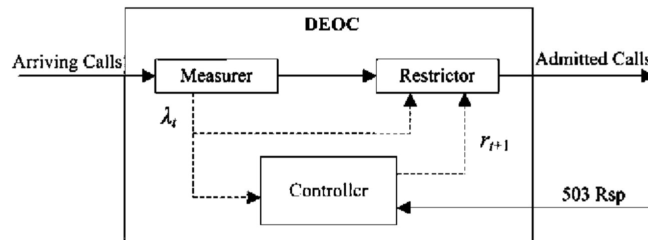


Fig. 3. The functional modules in DEOC.

be considered as the signal of overload. This function will be realized in our future work.

The controller module is implemented based on a finite state machine as shown in Fig. 4. 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 in Fig. 4, the RA only uses a binary feedback signal. That is, it only needs the information about whether 503 responses are received in each T (as receiving 503 responses means that call rejections have occurred in the network, we abbreviate it as rej in Fig. 4). The binary feedback indicates whether the network is currently overloaded or underutilized. A very good reason for adopting a binary feedback is that it makes the RA as simple and efficient as possible. Besides, it also minimizes the overhead of generating the feedback in the network.

There are three states in the state machine: underload, overload prevention and overload recovery. We detail them as follows:

Underload. It is the initial state of the controller. When the current call arrival rate is acceptable for the network, the controller is in underload state, where it does not throttle arriving calls. At the end of each T, if $\lambda_t \leqslant r_t$ and no call rejection is received during the current T, the state of controller transits into underload and r_{t+1} is set to λ_t, which is the real call input rate to the network. Also the gap interval is set as 0 to admit all arriving calls. No call rejection received means that the network is not overloaded and $\lambda_t \leqslant r_t$ indicates that call arrival rate is not greater than the call admission rate that does not lead to overload in the network. Thus all arriving calls are admitted into the network.

Overload prevention. When the current call arrival rate is greater than the current call admission rate and there is no overload in the network under this call admission rate, the controller is in overload prevention state, where it gradually increases the call admission rate. At the end of each T, if $\lambda_t > r_t$ and no call rejection is received during the current T, the state of the controller transits into overload prevention and r_{t+1} is increased according to the increasing rule of RA. No call rejection received means that the current call admission rate is acceptable for the network and $\lambda_t > r_t$ indicates that some arriving calls are rejected by the restrictor. In this case, the call admission rate in the next T should be increased to make full use of network resources and avoid unnecessary call rejections at the restrictor.

Overload recovery. When the overload occurs in the network, the controller is in overload recovery state, where it decreases the call admission rate to eliminate overload. At the end of each T, if call rejections are received during the current T, the state of controller transits into overload recovery and r_{t+1} is decreased according to the decreasing rule of RA. Call rejections received mean that the network is overloaded under current call admission rate and thus the call admission rate in the next T should be decreased so that the network can recover from overload. Since decreasing of call admission rate is fast according to the characteristic of RA (we show it in Section 4.4), the call admission rate will be decreased excessively if it is decreased by RA at every T during which call rejections are received, which further leads to network resource being underutilized. Thus when the controller is in overload recovery state, we propose the following method to avoid over decreasing: we define overload factor as the ratio between the number of received call rejections and the number of admitted calls. At the end of each T, we calculate overload factor in the current T and maintain the latest one. If the overload factor in the current T is smaller than the latest one, the decreasing rule of RA is not executed. Otherwise, the decreasing rule of RA is executed. In both cases, the latest overload factor is updated to the one in the current T. This method is reasonable because if the

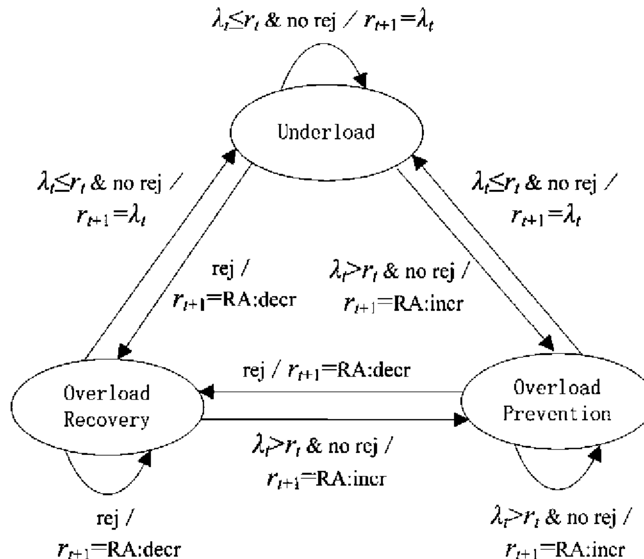


Fig. 4. The finite state machine of the controller module.

overload factor in the current T is smaller than the latest one, the degree of the overload in the network is reduced. In this case, the current call admission rate will eliminate overload after some T intervals. There is no need to further decrease call admission rate and thus the decreasing rule of RA is not executed.

4. RA design and analysis

In this section, we present RA and analyze its performance in detail. The overload of the SIP network is managed by a lot of DEOCs located on the edge of the network. These DEOCs are distributed and each DEOC executes RA to calculate the call admission rate based on the feedback received from the network. The feedback should be designed to be as simple as possible in order to make DEOC simple and practical.

RA has similar design requirements as the congestion control of TCP[3] [22,23] in the following aspects: (1) both RA and TCP congestion control algorithm should be distributed; (2) both algorithms should be deployed at the ends of the network (the end represents the host in TCP and the edge server in DEOC, respectively), which control the load of the network based on the feedback received; and (3) the feedback in both algorithms should be as simple as possible. Thus the design of RA can be inspired by the TCP congestion control algorithm.

In the last 20 years, a large number of TCP congestion control algorithms have been proposed and investigated: [23–25] propose the basic congestion control algorithms, which prevent the network from congestion collapse. Based on RTT measurements, [26,27] propose proactive congestion control algorithms to improve the throughput of a TCP flow. [28] proposes an equation-based congestion control algorithm, which can achieve high smoothness (minimizing the variations of packet transmission rates) as well as being TCP-friendly (competing fairly for bandwidth with conformant TCP flows). [29–31] propose TCP-friendly window-based congestion control algorithms that can achieve high smoothness. Besides, many congestion control algorithms are proposed to address the problems of effectively using network resources in wireless networks [32,33] and high-speed/long-delay networks [34–37].

Note that the TCP congestion control algorithms proposed have several essential differences from RA: (1) TCP congestion control algorithms process load at the packet level, while RA processes load at the call level; (2) as the dominating transport protocol, TCP supports various applications and thus its congestion control algorithms in different scenarios need to meet different requirements. For example, in a stationary environment, high throughput (maximizing the packet transmission rate) and high smoothness are required by bulk data transfer and streaming multimedia respectively. On the other hand, SIP is an application layer protocol and RA only needs to meet the requirements for SIP overload control, e.g., only high throughput (maximizing the rate of calls established) is needed in a stationary environment; and (3) since some TCP congestion control algorithms have been deployed into the network, the design of a new TCP congestion control algorithm should be concerned with both the intra-fairness (the fairness between flows running the same congestion control algorithm) and the inter-fairness (the fairness between flows running different congestion control algorithms). However, RA should only be concerned with the intra-fairness as there is no such an algorithm already in the network.

In the following, we take the basic TCP congestion control algorithm [24] as the basis to design RA. We apply the algorithm in [24] for SIP overload control as follows: 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. Otherwise, it is decreased. The authors [24] define one control rule for rate increase, and another rule for rate decrease. They evaluate all linear control rules based on efficiency, fairness and distribution requirements, and finally obtain the additive increase and multiplicative decrease (AIMD) algorithm as follows:

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

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

where r_t is the call admission rate at time t (in T). α and β are constant factors. That is, if no call rejection is received, the call admission rate is increased additively. Otherwise, it is decreased multiplicatively.

4.1. Aggressiveness, responsiveness and throughput

Before presenting the RA, we first consider important properties of call admission rate control algorithms in SIP networks including aggressiveness, responsiveness and throughput. The network is underutilized when the call admission rate is below the capacity of the network. In this case, DEOC 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 DEOC makes use of network resources as they are available. We define aggressiveness as the inverse of the time needed for the DEOC 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, DEOC needs to decrease the call admission rate as fast as possible in order to eliminate overload. Responsiveness measures how fast a DEOC decreases the call admission rate in response to overload. We define responsiveness as the inverse of the time needed for the DEOC to achieve the decrement of a certain amount of call admission rate, in response to a step increase of network overload. Obviously,

[3] TCP congestion control consists of slow start, congestion avoidance, fast retransmit and fast recovery. Only congestion avoidance can be applied to RA and thus we are only concerned with the congestion avoidance of the TCP congestion control in this paper.

high responsiveness, which allows DEOC 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 to DEOC 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. To simplify the performance analysis of different call admission rate control algorithms when the network is fully utilized, we use an ideal model similar to the one in [30,31]. In the ideal model, we assume that the call admission rate cannot exceed the system capacity (i.e., if the call admission rate obtained from the specific control strategy is greater than the system capacity, we set it as the system capacity). In the ideal model, if the call admission rate is below the network capacity, it is increased. If the call admission rate is equal to the network capacity, it is decreased. Under such an ideal model, a control epoch is defined as a sequence of call admission rate increments followed by one call admission rate decrement. Obviously, under such an ideal model, a control epoch will not include two or more decrements. This is because after one decrement, the call admission rate is below the network capacity and it will begin to be increased.

We study throughput according to the control epoch. 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 control epoch. High effective rate, which implies high throughput of the SIP network, is desirable. We define stable duration as the number of periods in one control epoch. 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, we analyze the performance of AIMD. As for the increasing rule, due to the constant increment α, the aggressiveness is low when α is small, which leads to rejecting a lot of arriving calls unnecessarily at DEOC 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, a constant increment α in AIMD cannot satisfy the requirements for SIP overload control. It is necessary to adopt a more flexible increment instead of a constant one for the increasing rule of RA. 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. Therefore, in order to design RA, we propose a more flexible increasing rule and adopt the same decreasing rule as AIMD. Note that several TCP congestion control algorithms [31,34,36,37] have been proposed, which can achieve high throughput and high aggressiveness for TCP flows. However, these algorithms are not specific to the SIP overload control.

4.2. The non-linear increasing rule of RA

Several non-linear increasing rules have been proposed in the recent research [30,31,34–37]. Compared to the linear increasing rule, these non-linear increasing rules are more flexible although they are more complex. Thus, we adopt non-linear increasing rule in RA.

The linear increasing rule (3) is first extended to the general non-linear increasing rule as follows:

$$\text{increasing}: \ r_{t+1} = r_t + \alpha r_t^k, \quad \alpha > 0 \tag{5}$$

Let $r(t)$ be the continuous approximation of the call admission rate at time t (in T), assuming that t is 0 at the beginning of a control epoch. Using linear interpolation and continuous approximation [30,31], we transform the general non-linear increasing rule as:

$$r_{t+1} - r_t = \alpha r_t^k \tag{6}$$

$$\frac{dr(t)}{dt} = \alpha r^k(t) \tag{7}$$

$$r(t) = \begin{cases} [r^{1-k}(0) + (1-k)\alpha t]^{\frac{1}{1-k}}, & k \neq 1 \\ r(0)e^{\alpha t}, & k = 1 \end{cases} \tag{8}$$

where $r(0)$ is the initial call admission rate in a control epoch. Referring to (8), we can see that $r(t)$ is mainly determined by parameter k. In the following, we investigate the effects of k on $r(t)$.

- When $k > 1$, we can deduce from (8) that at time $t = r^{1-k}(0)/\alpha(k-1)$, $r(t) \to \infty$. Thus the increasing rule is unstable.
- When $0 < k < 1$, we detail the effect based on the derivative of $r(t)$. The derivatives are calculated as follows:

$$\frac{dr(t)}{dt} = \alpha[r^{1-k}(0) + (1-k)\alpha t]^{\frac{k}{1-k}}, \quad k \neq 1 \tag{9}$$

$$\frac{d^2r(t)}{dt^2} = k\alpha^2[r^{1-k}(0) + (1-k)\alpha t]^{\frac{2k-1}{1-k}}, \quad k \neq 1 \tag{10}$$

From (9) and (10), we can see that $r'(t) > 0$ and $r''(t) > 0$. That is, the call admission rate increases with time, and its increasing rate increases with time. In this case, the call admission rate increases cautiously at the beginning, and its increasing rate becomes more and more aggressive when no call rejection is received. This increasing rule achieves high aggressiveness and maintains long stable duration, thus it is suitable for SIP overload control.

- When $k = 0$, we deduce from (8) that the call admission rate increases linearly. It is the increasing rule in AIMD and the increasing rate of call admission rate does not change with time.
- When $k < 0$, we can calculate that $r'(t) > 0$ and $r''(t) < 0$ by using (9) and (10). That is, the call admission rate increases with time, but its increasing rate decreases with time. In this case, the call admission rate increases aggressively at the beginning, but its increasing rate

becomes less and less aggressive. This increasing rule cannot achieve high aggressiveness. Obviously, it is not suitable for SIP overload control.

- When $k = 1$, the increasing rule is multiplicative, which is similar to the increasing rule of the slow start in the TCP congestion control. In the multiplicative increasing rule, the call admission rate increases with time and its increasing rate increases with time. This property is similar to that of the increasing rule with $0 < k < 1$. However, according to (8), we can see that the call admission rate increases exponentially in the multiplicative increasing rule. The exponential increase (when $k = 1$) causes the call admission rate to be increased much faster than the polynomial increase (when $0 < k < 1$). Thus, the multiplicative increasing rule increases the call admission rate too fast, which results in low throughput. Therefore, the multiplicative increasing rule is not suitable for SIP overload control.

To sum up, we should adopt $0 < k < 1$ to design a flexible and proper increasing rule, which satisfies the requirements for SIP overload control. Besides, we include $k = 0$ in our further discussions in order to achieve a more comprehensive analysis.

4.3. Fairness

Fairness is an important performance metric in the distributed rate control algorithms [18,24]. In this section, we are concerned with designing RA so as to achieve fairness, which requires that DEOCs with different initial call admission rates finally achieve the same call admission rates. From (9), we can see that in the general non-linear increasing rule with $0 < k < 1$, larger $r(0)$ leads to larger $r'(t)$. That is, the DEOC with larger initial call admission rate can increase the call admission rate more aggressively. On the other hand, in the linear increasing rule (i.e., $k = 0$), $r'(t)$ is independent of $r(0)$. That is, DEOCs with different initial call admission rates can increase the call admission rates at the same speed. Obviously, the linear increasing rule is better than the non-linear one in terms of fairness.

In order to improve fairness, we modify (5) to eliminate $r(0)$ in the expression of $r'(t)$ as follows:

$$r_{t+1} = r_t + \alpha(r_t - r_0)^k, \quad \alpha > 0, 0 \leqslant k < 1 \tag{11}$$

$$r(t) = r(0) + [(1-k)\alpha t]^{\frac{1}{1-k}} \tag{12}$$

$$\frac{dr(t)}{dt} = \alpha[(1-k)\alpha t]^{\frac{k}{1-k}} \tag{13}$$

$$\frac{d^2r(t)}{dt^2} = k\alpha^2[(1-k)\alpha t]^{\frac{2k-1}{1-k}} \tag{14}$$

By definition, $r_0 = r(0)$. From (13), we can see that $r'(t)$ is independent of $r(0)$ in the non-linear increasing rule and thus the fairness is improved by using (11). In the following, we adopt the same network model as in [24,30,31] to demonstrate that multiple users with synchronized feedbacks using the increasing rule (11) can converge to fairness. We analyze a two-user case, which is easy to extend to the multiple-user case.

As shown in Fig. 5, any two-user resource allocation method can be represented by a point X (x_1, x_2), where x_i is the resource allocation (normalized by total capacity) for the ith user ($i = 1, 2$), i.e., x_i represents the call admission rate of the ith DEOC. The fairness index is defined as $max(x_1/x_2, x_2/x_1)$ [31]. If the fairness index is closer to 1, the resource allocation method has better fairness. The line ($x_1 = x_2$) is "equi-fairness line" and the line ($x_1 + x_2 = 1$) is "max-utilization line". The closer to the intersection of the equi-fairness and max-utilization lines, the better the resource allocation method. When the system is under-utilized (we assume $x_1 < x_2$ without loss of generality), AIMD increases the resource allocation of both users by a constant value and thus the increase is parallel to the equi-fairness line as shown in Fig. 5. This movement improves fairness, i.e., reduces the fairness index. When the total resource allocation exceeds max-utilization, both users use multiplicative decreases and the fairness index is not changed. Hence, as the system evolves, AIMD enables the resource allocation point to move towards the equi-fairness line, while oscillating around the max-utilization line.

Since RA adopts the same decreasing rule as AIMD, we are now only concerned with the non-linear increasing rule of RA. Fig. 6 shows an increase procedure, where the curve marked as "org" corresponds to the original non-linear increasing rule as (5) and the curve marked as "imp" corresponds to the improved non-linear increasing rule as (11). For the original rule, when $k = 0$, the increasing rule is additive and the increase is parallel to the equi-fairness line, which improves the fairness. When $k = 1$, the increasing rule is multiplicative and the increase moves along the line opposite to the origin without altering fairness. When $0 < k < 1$, the increase curves lie between the line ($k = 1$) and the line ($k = 0$), which is explained as follows: the value of x_i at point X can be considered as the initial call admission rate of the ith DEOC. Since the value of x_1 is smaller than that of x_2 at point X, the increasing rate of x_1 is smaller than that of x_2 according to (9), which leads to a larger amount of increase in x_2. Thus the curves lie above the line ($k = 0$). On the other hand, (5) indicates that during each period, the proportion between the amount of increase in x_2 and that in x_1 is smaller than the proportion between the value of x_2 and that of x_1 at the beginning of the period. Since the total amount of increase in x_i is composed by the increases during multiple periods, the proportion between the total amount of increase in x_2 and that in x_1 is smaller than the proportion between the value of x_2 and that of x_1 at point X. Thus the curves lie below the line ($k = 1$), where the above two proportions are the same. Therefore, the improvement of fairness when $0 < k < 1$ is less significant than that when $k = 0$. On the other hand, for the improved rule, when $0 \leqslant k < 1$, the increase is parallel to the equi-fairness line as the increasing rate of x_i is independent of the value of x_i at point X according to (13) and thus the resource allocation of each user has the same amount of increase. Therefore, we conclude that (11) outperforms (5) in terms of fairness when $0 < k < 1$. Similar to AIMD, (11) can make both users converge to fairness. Thus we adopt (11) as the increasing rule of RA.

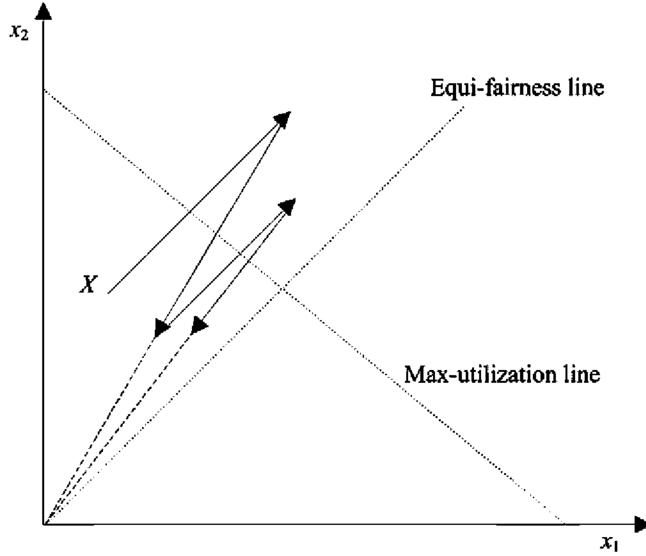


Fig. 5. Sample path showing the convergence to fairness for AIMD.

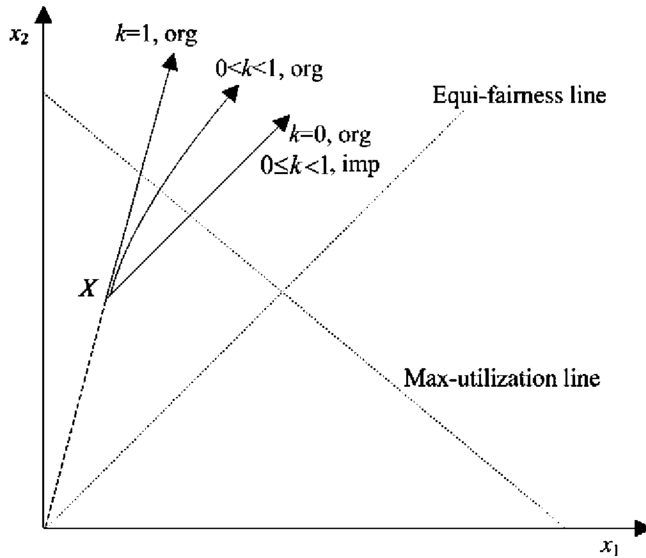


Fig. 6. Improvement in the convergence to fairness for RA.

4.4. The effects of different parameters on RA

Based on the analysis in previous sections, RA adopts (11) and (4) as its increasing rule and decreasing rule, respectively. There are three parameters in RA: α, k and β. We study their effects on the performance of RA in this section. In order to evaluate the effects quantitatively, we assume that the available network resources are increased suddenly by a factor of m to permit us to derive an expression for aggressiveness. Similarly, we decrease the available network resources suddenly by a factor of m to permit derivation of an expression for responsiveness. Note that aggressiveness can also be evaluated by using a step increase of call arrival rate when there are available resources in the network. Since both the increase of available network resources and the increase of call arrival rate have the same effects on RA, only the increase of available network resources is considered in our following analytical calculation. Let r_m denote the maximum call admission rate in a control epoch, thus it is increased from r_m to $m \times r_m$ to test aggressiveness and decreased from r_m to r_m/m to test responsiveness.

First, we consider the decreasing rule and the effect of β. By using (4), we can calculate that it needs $\log_{(1-\beta)} \frac{1}{m}$ periods to decrease the maximum call admission rate from r_m to r_m/m. Thus the responsiveness is determined by β. Since the call admission rate is decreased exponentially, a small β can even achieve high responsiveness. Besides,

the call admission rate would be over decreased if β is large, which leads to network resources being underutilized under overload. Therefore, we adopt β to be a small value for RA.

Then, we are concerned with the increasing rule and the effects of α and k. For aggressiveness, referring to (12), the time (which is the number of periods and denoted as T_{agg}) needed to increase the maximum call admission rate from r_m to $m \times r_m$ is calculated as:

$$T_{agg} = \frac{[(m-1)r_m]^{(1-k)}}{(1-k)\alpha} \tag{15}$$

The approximate expressions of stable duration and effective rate are shown in (16) and (17) respectively, which are deduced in Appendix A.

$$T_s = \frac{(\beta r_m)^{(1-k)}}{(1-k)\alpha} \tag{16}$$

$$\mathrm{E}[r(t)] = r_m - \frac{1}{2-k}\beta r_m \tag{17}$$

where T_s denotes the stable duration and $\mathrm{E}[r(t)]$ denotes the effective rate.

Based on (15)–(17), numerical results in Fig. 7 and Table 1 show the tradeoffs among aggressiveness, stable duration and effective rate. We set $r_m = 200$, $\beta = 1/8$ and $m = 2$. Fig. 7 shows the inverse of aggressiveness of RA with parameter $k = 0$, $k = 0.2$, $k = 0.5$ and $k = 0.8$ as the stable duration varies. Their special cases, RA0($k = 0$, $\alpha = 2$), RA1($k = 0$, $\alpha = 5$), RA2($k = 0.5$, $\alpha = 2$) and RA3($k = 0.8$, $\alpha = 2$) are also shown by points. Table 1 shows the effective rate of RA with different parameter k.

From Fig. 7 and Table 1, we can see that the aggressiveness becomes better as k increases, meanwhile the effective rate decreases. In detail, we can see that RA1, RA2 and RA3 have comparable stable durations. Besides, RA3 has the best aggressiveness and the minimum effective rate, which leads to the minimum throughput of the SIP network in RA3 since the stable durations of RA1, RA2 and RA3 are almost identical. In the same way, RA1 has the worst aggressiveness and the maximum throughput. For SIP overload control, both aggressiveness and throughput are important. The low throughput results in poor performance of the SIP network, and the low aggressiveness leads to a lot of unnecessary call rejections at DEOC. Therefore, we choose k to be the moderate value among all possible values ($0 < k < 1$), which is 1/2, to provide high throughput as well as high aggressiveness. In the following, we set k as 1/2 in DEOC. Note that when $k = 1/2$, the call admission rate increases quadratically according to (12), and this increasing can ensure high aggressiveness.

When k is selected, we can see from (15)–(17) that α does not affect the effective rate and as α increases, aggressiveness becomes higher and stable duration decreases, which causes throughput to become lower. We discuss how to choose the value of α in practice in Section 5.3.2.

Table 1
Effective rate of RA with different k.

	$k = 0$	$k = 0.2$	$k = 0.5$	$k = 0.8$
Effective rate (calls per second)	187.5	186.1	183.3	179.2

5. Performance evaluation

5.1. Simulation environment

The simulation platform used is the NS-2 simulator [38]. The proposed DEOC is implemented based on Prior's NS-2 SIP module [39]. 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 [40]). Besides, they are commonly used in recent research [4,18]. 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).

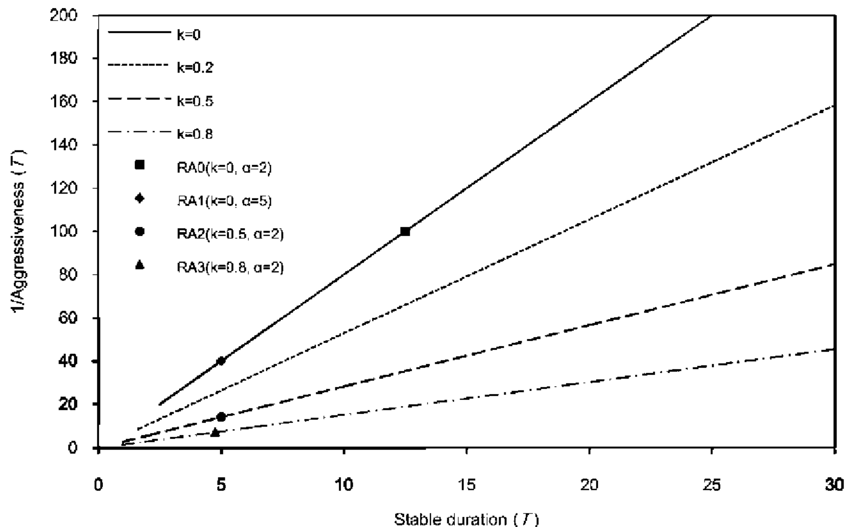


Fig. 7. Inverse of aggressiveness of RA with $k = 0$, $k = 0.2$, $k = 0.5$ and $k = 0.8$ as stable duration varies.

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 [1]. 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. We use 503 responses without Retry-After header recommended by [5,6] 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. Besides, we adopt the early rejection method in SIP servers as suggested by [4] 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 [4,18], we compare the performance of DEOC with no overload control (No Control), local occupancy-based control (OCC-Local) and hop-by-hop occupancy-based control (OCC-Hop). 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 based on the acceptance probability, which is described in [10]. The parameters of the OCC algorithms are set the same as those in [4]. 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 [6]. The direct upstream neighbors then execute the rejection in place of the overloaded server. In DEOC, 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 [41]. The increasing parameter α is set to 0.2 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 DEOC are set to 1 s, i.e., T in DEOC is 1 s. 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.

5.2. Performance results

5.2.1. 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.

In our first experiment, the network topology 1 is used. The results of goodput and call setup delay are shown in Figs. 8 and 9, respectively. As Fig. 8 shows, 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, DEOC outperforms other mechanisms and 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) Both OCC-Hop and DEOC keep constant goodput as the offered load increases. It is because the overload occurs on the core servers, and they are only one-hop away from all edge servers as shown in topology 1. In this case, both hop-by-hop and end-to-end overload control reject excessive requests at the edge servers. Therefore, the overloaded server has enough capacity to serve calls. Besides, no extra resources in the network are wasted on processing requests that will finally be rejected since they are rejected at the edge of the network. Note that the goodput of DEOC is higher than that of OCC-Hop, which is explained as follows: OCC based algorithms have the objective of dynamically adjusting call acceptance probability to maintain the processor occupancy at or below a given target threshold. Obviously in OCC algorithms, higher threshold leads to higher goodput. However, if the threshold is set too high (i.e., higher than 95%), the goodput performance tends to degrade under heavy offered load [15], because the instantaneous server occupancy could still exceed the healthy region and cause longer delays, which result in the expiration of SIP timers and message retransmissions. Similar to the previous research [4,18], a tradeoff threshold achieving high and

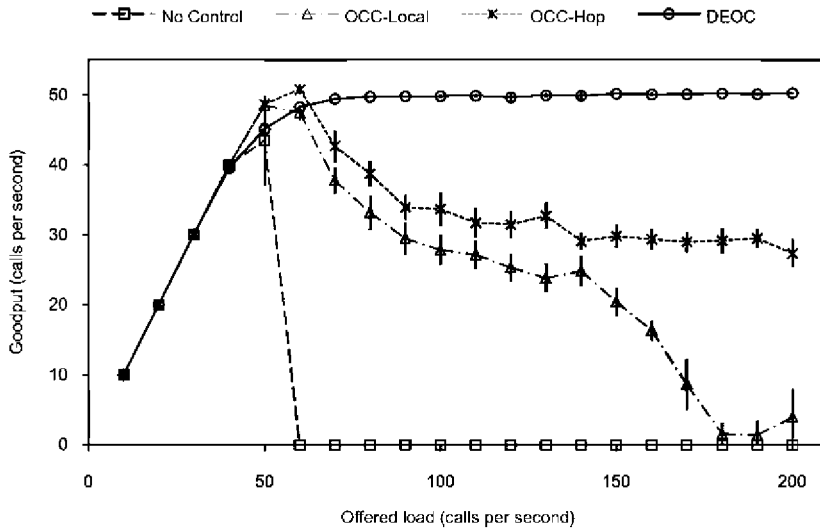


Fig. 8. Goodput comparison with varying offered load among different overload control mechanisms based on topology 1.

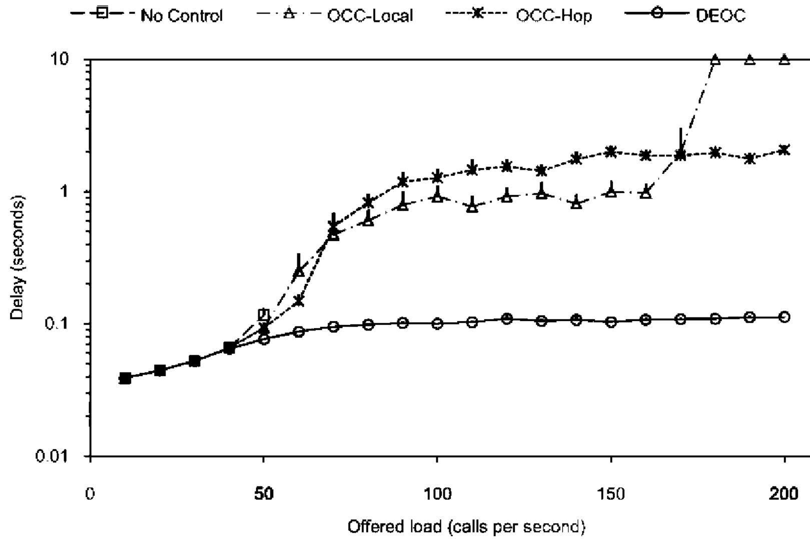


Fig. 9. Delay comparison with varying offered load among different overload control mechanisms based on topology 1.

stable performances across all load conditions is needed and is set to 90% in our experiments. Thus, the OCC-Hop is sub-optimal in terms of goodput as it could not fully utilize the capacity of the processor.

Fig. 9 shows the results of call setup delay. Note that for No Control case, the delay is plotted only up to 50 cps 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, DEOC performs up to an order of magnitude better than other mechanisms. Besides, its call setup delay is always at a low value (i.e., 0.1 s). In this case, the call setup delay does not have a significant increase when the network is under heavy offered load and thus the user experience is always assured.

In the second experiment, we use the network topology 2. The results of goodput and call setup delay are shown in Figs. 10 and 11, respectively. Fig. 10 shows the goodput obtained from different overload control algorithms under different offered load. The results are similar to those in Fig. 8. The only difference is that when the offered load goes beyond the capacity of the network, the goodput in OCC-Hop degrades while it remains high in DEOC. We explain it as follows: the overload occurs on the core servers, but the direct upstream neighbors of the overloaded core server include other core servers as shown in topology 2. Therefore, in hop-by-hop overload control, a lot of resources in the network are wasted on processing requests that will finally be rejected, e.g., when C3 is overloaded, its incoming calls from E1 are rejected by C1/C2. However, C1/C2 may also be overloaded and executing the rejection will

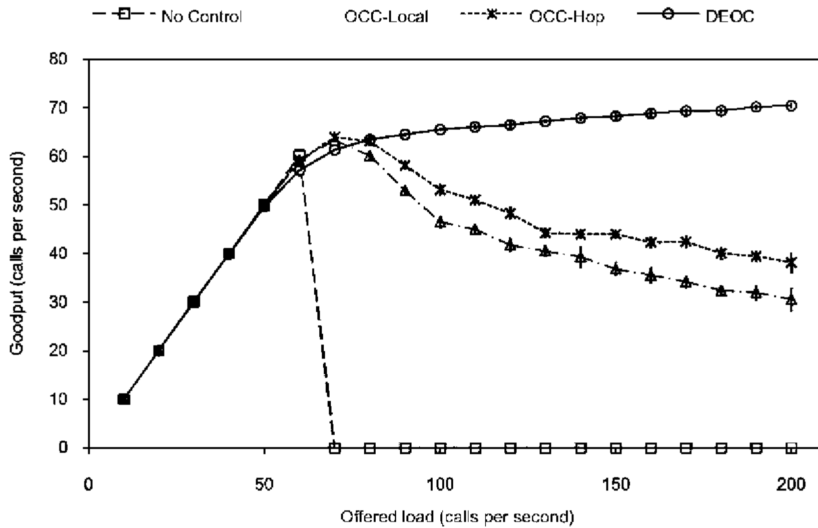


Fig. 10. Goodput comparison with varying offered load among different overload control mechanisms based on topology 2.

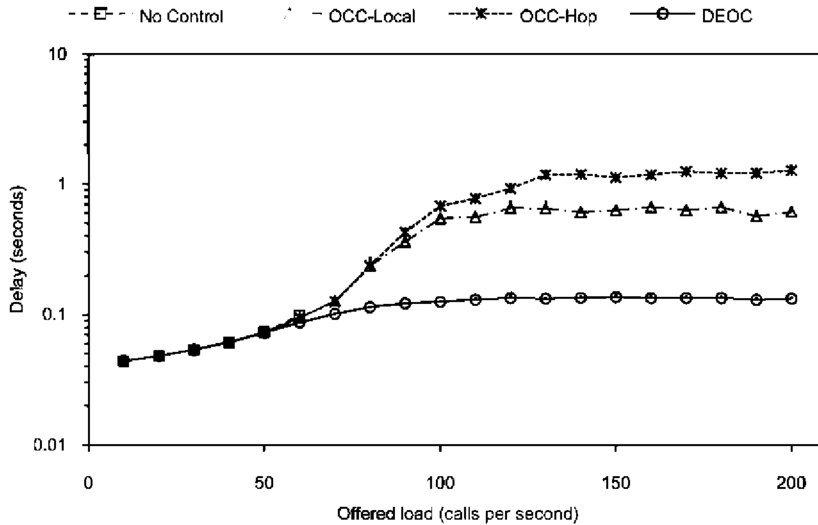


Fig. 11. Delay comparison with varying offered load among different overload control mechanisms based on topology 2.

amplify the overload in C1/C2. Besides, unnecessary resource consumption takes place in E1 since the requests forwarded by E1 towards C3 will probably be rejected by C1/C2. Therefore, as the offered load increases, the goodput of hop-by-hop overload control decreases. Obviously, when C3 is overloaded, its incoming calls from E1 should be rejected by E1 in order to achieve high goodput. This requirement is satisfied by end-to-end overload control and thus DEOC keeps constant goodput as the offered load increases. Fig. 11 shows the results of call setup delay, which are similar to those in Fig. 9.

5.2.2. *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 their results in Figs. 12 and 13, 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 DEOC all respond quickly to the sudden load variations. Besides, DEOC achieves better goodput and call setup delay when the offered load goes beyond the capacity of the network.

In Fig. 12, we observe that there is a decrease just after an increase in the goodput curves of all mechanisms at

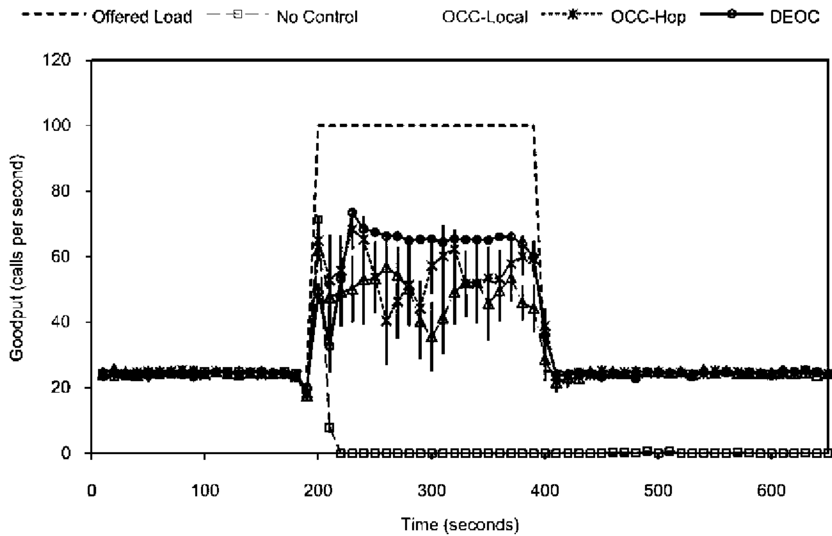


Fig. 12. Goodput comparison among different overload control mechanisms when the offered load is varied suddenly between overload and underutilization.

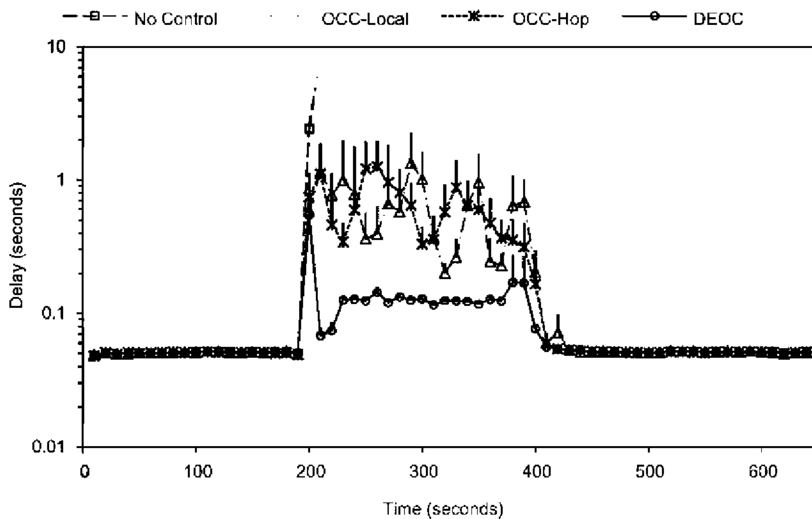


Fig. 13. Delay comparison among different overload control mechanisms when the offered load is varied suddenly between overload and underutilization.

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 DEOC, we observe that after a sudden increase of the offered load (i.e., after 200 s), the goodput drops immediately as shown in Fig. 12. Besides, the call setup delay is limited to a lower value (i.e., less than 1 s) under overload as shown in Fig. 13. This is mainly because in DEOC, 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, DEOC has high responsiveness. After the overload is eliminated, the goodput increases immediately to the capacity of the net-

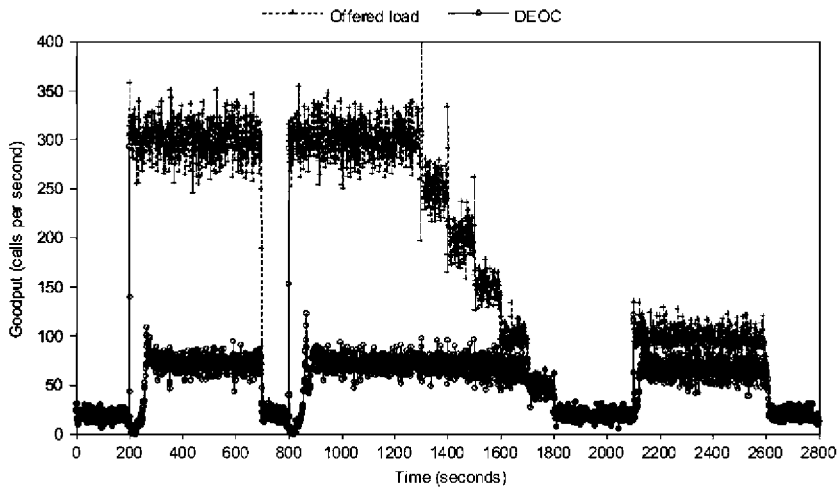


Fig. 14. Goodput of DEOC under variable offered load (the offered load and the goodput of DEOC overlap in the time intervals 0–200, 700–800, 1700–2100 and 2600–2800).

work as shown in Fig. 12. Therefore, DEOC has high aggressiveness.

5.2.3. Performance under VoIP traffic

In this section, we examine the performance of DEOC using a realistic overload profile, which represents the overload events experienced in real VoIP networks [17]. It consists of a severe overload event followed by a slight one. The severe overload event sustains for a long interval, which includes a short dip, and then ends gradually. We measure the instantaneous goodput and call setup delay every 1 s and show a sample path of goodput and call setup delay as a function of time in Figs. 14 and 15, respectively.

As these figures show, when the offered load is increased nearly 5 times of the network's capacity suddenly at 200 s and 800 s respectively, there is a sudden increase of goodput and overload occurs. (The reason is similar to that in Section 5.2.2.) Under overload, DEOC decreases the call admission rate rapidly to eliminate overload. Specifically, we can see from Fig. 14 that it only takes DEOC few seconds to decrease the call admission rate from approximately 300 cps to less than 50 cps at 200 s and 800 s respectively, which indicates its high responsiveness. Due to the high responsiveness, we see from Fig. 15 that the call setup delay decreases quickly after the call admission rate is decreased by DEOC, and thus the network recovers from overload quickly. After overload is eliminated, DEOC increases the call admission rate to the capacity of the network in a few seconds as shown in Fig. 14, which indicates its high aggressiveness. Then DEOC keeps the goodput at a value that maximizes the utilization of

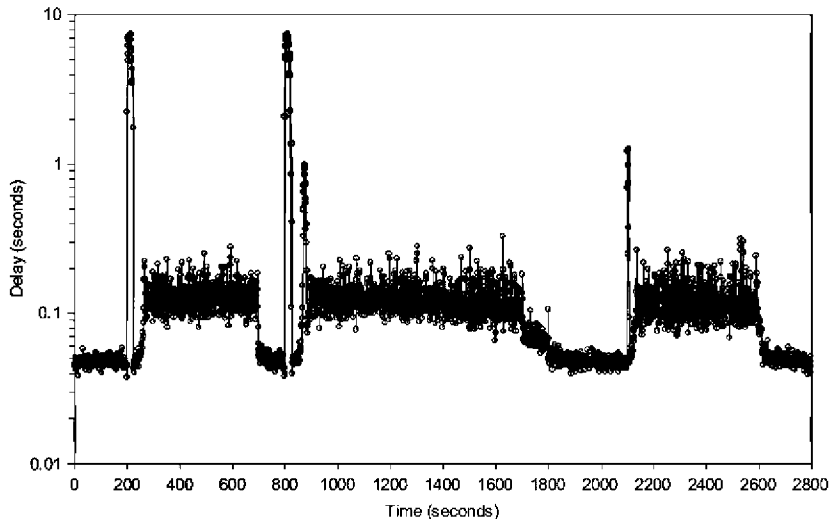


Fig. 15. Delay of DEOC under variable offered load.

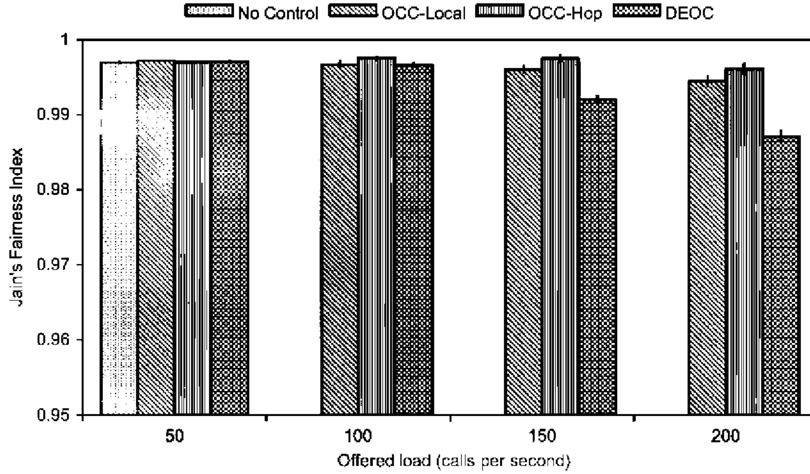


Fig. 16. Jain's fairness index comparison among different overload control mechanisms when the offered load is 50, 100, 150 and 200 cps respectively.

network while limiting the call setup delay. As the offered load gradually decreases, DEOC accordingly adapts to the decrease. Finally, DEOC can also handle light overload event efficiently.

5.2.4. 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 [42] 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 [43], which is defined as follows:

$$f = \frac{\left(\sum_{i=1}^{k} x_i\right)^2}{k\left(\sum_{i=1}^{k} x_i^2\right)} \tag{18}$$

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. 16 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 respectively), 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 DEOC. We explain it as follows: both OCC-Local and OCC-Hop

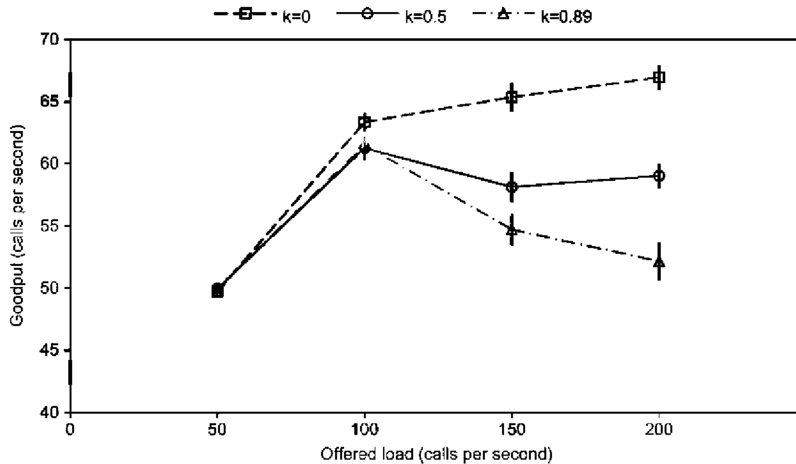


Fig. 17. Goodput comparison with varying offered load among DEOC with $k = 0$, $k = 0.5$ and $k = 0.89$.

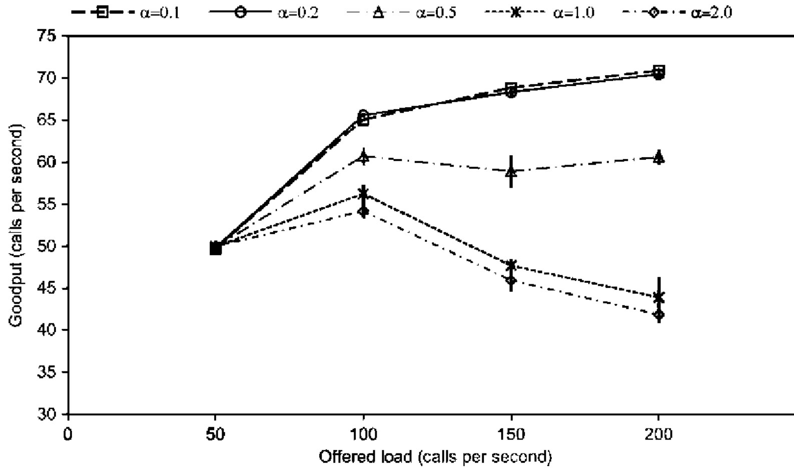


Fig. 18. Goodput comparison with varying offered load among DEOC with $\alpha = 0.1$, $\alpha = 0.2$,$\alpha = 0.5$,$\alpha = 1.0$ and $\alpha = 2.0$.

control overload near the overloaded server. Thus they have better knowledge of the overload. However, DEOC infers 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, DEOC can still achieve good fairness.

5.3. Parameter tuning

In this section, we discuss the effects of parameters k and α on the performance of DEOC. In the following experiments, when comparing response time, we use the same simulation setup as that in Section 5.2.2. 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.

5.3.1. Effect of parameter k

The parameter k determines the call admission rate increasing rule of RA and reflects a trade-off between goodput and aggressiveness. In this section, we adopt different k in DEOC. Similar to the discussion in Section 4.4, for each k, the corresponding α is calculated according to (16) in order to make DEOC with different k have the same stable durations. A larger k implies a more aggressive call admission rate increase and a lower effective rate, which leads to a lower goodput.

Fig. 17 shows the goodput obtained from DEOC with $k = 0$, $k = 0.5$ and $k = 0.89$ 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 respectively), the DEOC with minimum k has the highest goodput and the DEOC with maximum k has the lowest goodput. Table 2 shows the response time of DEOC with different k. We can see that the DEOC with minimum k has the longest response time and the DEOC with maximum k has the shortest response time. Thus the larger k leads to lower goodput and higher aggressiveness, which validates our analysis in Section 4.4.

5.3.2. Effect of parameter α

Fig. 18 shows the goodput obtained from DEOC with various α under different offered load. When the offered load goes beyond the capacity of the network, the DEOC with smaller α has higher goodput. The response time of DEOC with different α is shown in Table 3. We can see that the DEOC with smaller α has longer response time. Therefore, the smaller α, the higher goodput and lower aggressiveness. This conclusion is consistent with the analytical results in Section 4.4.

Note that when $\alpha > 0.2$, goodput drops significantly as the offered load increases. We explain it as follows: Table 3 shows the stable duration calculated according to (16). From Table 3 and Fig. 18, we can see that longer stable duration leads to lower aggressiveness and higher goodput. In fact, when only α varies, the stable duration is directly related to the goodput as discussed in Section 4.4. When $\alpha > 0.2$, the stable duration is too short (less than or little greater than one period), which indicates that DEOC increases call admission rate excessively and is unable to keep high goodput under heavy offered load.

Therefore, in practice, first we limit α to possible values that can guarantee that the goodput is acceptable under heavy offered load. In most cases, we empirically find out

Table 2
Simulated response time comparison among DEOC with $k = 0$, $k = 0.5$ and $k = 0.89$.

	$k = 0$	$k = 0.5$	$k = 0.89$
Simulated response time (s)	10.4	5.8	3.6
(95% Confidence interval)	(8.74, 12.06)	(5.1, 6.5)	(2.87, 4.33)

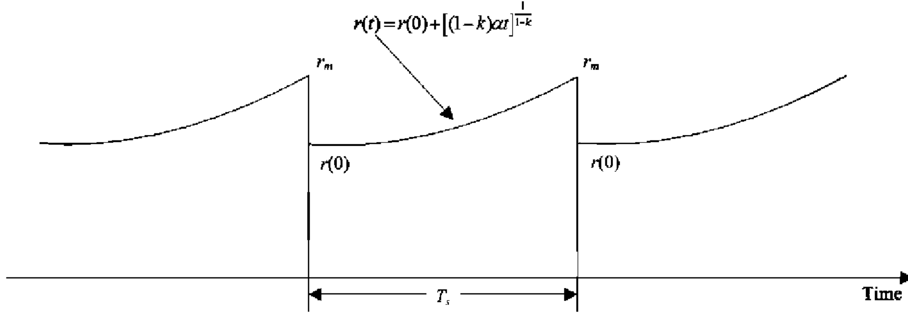


Fig. 19. Control epochs where call admission rate increases and decreases alternately when the network is fully utilized.

Table 3
Simulated response time comparison and calculated stable duration comparison among DEOC with $\alpha = 0.1$, $\alpha = 0.2$, $\alpha = 0.5$, $\alpha = 1.0$ and $\alpha = 2.0$.

	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.5$	$\alpha = 1.0$	$\alpha = 2.0$
Simulated response time (s)	32.6	14.1	5.1	2.4	1.4
(95% Confidence interval)	(30.15, 35.05)	(12.63, 15.57)	(4.16, 6.04)	(1.73, 3.07)	(0.97, 1.83)
Calculated stable duration (s)	7.56	3.78	1.51	0.76	0.38

that keeping stable duration more than 3 periods may be enough to provide acceptable goodput. Then, among all possible values of α meeting the requirements for goodput, the maximum one is selected as the value of α so as to provide better aggressiveness.

6. 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. However, current existing end-to-end overload control solutions [4,18] are too complex to be practical. In this paper, we proposed DEOC, which is a distributed end-to-end overload control mechanism for SIP networks and is easy to implement. We presented the design of the proposed approach, and evaluated its performance through simulation experiments. Our simulation results demonstrated that DEOC 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 DEOC and evaluate its performance in real networks. In real networks, some edge servers may not deploy DEOC as they are not under the control of the carrier. If only some of the edge servers deploy DEOC, it will lead to the unfairness between the edge servers with DEOC and those without DEOC when overload occurs in the core servers. This is because the edge servers with DEOC decrease the call admission rates in response to overload, while the edge servers without DEOC 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 DEOC. Besides, we will explore an application-layer probing mechanism at the edge servers, which provides a more complex and accurate feedback than the binary one. Based on the feedback, it is expected that DEOC can infer the overload of the network more accurately and in a timely manner and thus overload control can be performed more efficiently. Finally, we will define a SIP header in the 503 response to indicate the reason of sending this 503 response. A SIP server can send a 503 response due to the following two reasons: overload and server maintenance. Both reasons are different in nature. For maintenance, it is useful for the receiver of a 503 response to re-send the request to an alternate server since it is likely that not all servers of a domain will be taken down at the same time. However, in the case of overload, re-sending requests to alternate servers is problematic and may amplify the load on severs [3]. Thus in this case, the receiver of a 503 response should forward this response to the upstream neighbor, from which the INVITE request related to this response has been received. In this way, this 503 response finally arrives at the edge server, which then sends a 500 response to UE to reject the call. Therefore, we should define a SIP header in the 503 response to indicate the reason of sending this 503 response so that the receiver of a 503 response can take different strategies based on the specific reason.

Appendix A

In this appendix, we detail the deduction of (16) and (17). When discussing stable duration or effective rate of RA, we consider control epochs where the call admission rate increases and decreases alternately when SIP network is fully utilized, as shown in Fig. 19. In each control epoch, the call admission rate increases until it achieves the maximum value r_m. Then, the call admission rate decreases and the system enters into the next control epoch. Thus, by using (4), the initial call admission rate in the control epoch is $r(0) = (1-\beta)r_m$. We omit the effect of call admission rate decreasing on the control epoch as shown in Fig. 19 because it only occupies one period in the model and the stable duration and the effective rate are relatively measured.

The stable duration is the time (the number of periods) needed to increase the call admission rate from $r(0)$ to r_m. Referring to (12), it is calculated as:

$$\begin{aligned} r_m &= (1-\beta)r_m + [(1-k)\alpha T_s]^{\frac{1}{1-k}} \\ T_s &= \frac{(\beta r_m)^{(1-k)}}{(1-k)\alpha} \end{aligned} \tag{A.1}$$

To compute effective rate, we assume that $r(t)$ is uniformly distributed in a control epoch. By using (12) and (A.1), the effective rate is calculated as follows:

$$\begin{aligned} \mathrm{E}[r(t)] &= \frac{1}{T_s}\int_0^{T_s} r(t)dt = \frac{1}{T_s}\int_0^{T_s} r(0) + [(1-k)\alpha t]^{\frac{1}{1-k}}dt \\ &= r(0) + \frac{1-k}{2-k}[(1-k)\alpha T_s]^{\frac{1}{1-k}} = r_m - \frac{1}{2-k}\beta r_m \end{aligned} \tag{A.2}$$

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, SIP: Session Initiation Protocol, IETF RFC 3261, June 2002.
[2] J. Liao, J. Wang, T. Li, J. Wang, X. Zhu, A token-bucket based notification traffic control mechanism for IMS presence service, Computer Communications 34 (10) (2011) 1243–1257.
[3] J. Rosenberg, Requirements for Management of Overload in the Session Initiation Protocol, IETF RFC 5390, December 2008.
[4] V. Hilt, I. Widjaja, Controlling overload in networks of SIP servers, in: Proceedings of IEEE ICNP, October 2008, pp. 83–93.
[5] V. Hilt, E. Noel, C. Shen, A. Abdelal, Design considerations for session initiation protocol (SIP) overload control, IETF RFC 6357, August 2011.
[6] V. Gurbani, V. Hilt, H. Schulzrinne, Session initiation protocol (SIP) overload control, draft-ietf-soc-overload-control-08, IETF, Work in Progress, March 2012.
[7] R.P. Ejzak, C.K. Florkey, P.T. Lee, Network overload and congestion: a comparison of ISUP and SIP, Bell Labs Technical Journal 9 (3) (2004) 173–182.
[8] E.M. Nahum, J. Tracey, C.P. Wright, Evaluating SIP proxy server performance, ACM SIGMETRICS Performance Evaluation Review 35 (1) (2007) 349–350.
[9] M. Ohta, Overload control in a SIP signaling network, Enformatika Transactions on Engineering, Computing and Technology (2006) 205–210.
[10] B.L. Cyr, J.S. Kaufman, P.T. Lee, Load balancing and overload control in a distributed processing telecommunication systems, United States Patent No. 4,974,256, 1990.
[11] S. Kasera, J. Pinheiro, C. Loader, M. Karaul, A. Hari, T. LaPorta, Fast and robust signaling overload control, in: Proceedings of IEEE ICNP, November 2001, pp. 323–331.
[12] M. Ohta, Overload protection in a SIP signaling network, in: Proceedings of IEEE ICISP, August 2006.
[13] R.G. Garroppo, S. Giordano, S. Spagna, S. Niccolini, Queueing strategies for local overload control in SIP server, in: Proceedings of IEEE GLOBECOM, December 2009, pp. 1–6.
[14] E.C. Noel, C.R. Johnson, Initial simulation results that analyze SIP based VoIP networks under overload, in: Proceedings of International Teletraffic Congress, June 2007, pp. 54–64.
[15] C. Shen, H. Schulzrinne, E. Nahum, Session initiation protocol (SIP) server overload control: design and evaluation, in: Proceedings of IPTComm, July 2008, pp. 149–173.
[16] R.G. Garroppo, S. Giordano, S. Niccolini, S. Spagna, A prediction-based overload control algorithm for SIP servers, IEEE Transactions on Network and Service Management 8 (1) (2011) 39–51.
[17] A. Abdelal, W. Matragi, Signal-based overload control for SIP servers, in: Proceedings of IEEE CCNC, January 2010, pp. 1–7.
[18] Y. Wang, SIP overload control: a backpressure-based approach, in: Proceedings of ACM SIGCOMM (poster), August 2010, pp. 399–400.
[19] H. Jiang, A. Iyengar, E. Nahum, W. Segmuller, A. Tantawi, C.P. Wright, Load balancing for SIP server clusters, in: Proceedings of IEEE INFOCOM, April 2009, pp. 2286–2294.
[20] S. Kundan, H. Schulzrinne, Failover, load sharing and server architecture in SIP telephony, Computer Communications 30 (5) (2007) 927–942.
[21] A. Berger, Comparison of call gapping and percent blocking for overload control in distributed switching systems and telecommunications networks, IEEE Transactions on Communications 39 (4) (1991) 574–580.
[22] A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock, Host-to-host congestion control for TCP, IEEE Communications Surveys & Tutorials 12 (3) (2010) 304–342.
[23] M. Allman, V. Paxson, E. Blanton, TCP congestion control, IETF RFC 5681, September 2009.
[24] D.-M. Chiu, R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, Computer Networks and ISDN Systems 17 (1) (1989) 1–14.
[25] V. Jacobson, Congestion avoidance and control, in: Proceedings of ACM SIGCOMM, 1988, pp. 314–329.
[26] L. Brakmo, L. Peterson, TCP Vegas: end to end congestion avoidance on a global Internet, IEEE Journal of Selected Areas in Communications 13 (8) (1995) 1465–1480.
[27] G. Hasegawa, K. Kurata, M. Murata, Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet, in: Proceedings of IEEE ICNP, November 2000, pp. 177–186.
[28] M. Handley, S. Floyd, J. Padhye, J. Widmer, TCP Friendly rate control (TFRC): protocol specification, IETF RFC 5348, September 2008.
[29] Y.R. Yang, S.S. Lam, General AIMD congestion control, in: Proceedings of IEEE ICNP, November 2000, pp. 187–198.
[30] D. Bansal, H. Balakrishnan, Binomial congestion control algorithms, in: Proceedings of IEEE INFOCOM, April 2001, pp. 631–640.
[31] S. Jin, L. Guo, I. Matta, A. Bestavros, A spectrum of TCP-Friendly window-based congestion control algorithms, IEEE Transactions on Networking 11 (3) (2003) 341–355.
[32] S. Mascolo, C. Casetti, M. Gerla, M.Y. Sanadidi, R. Wang, TCP Westwood: bandwidth estimation for enhanced transport over wireless links, in: Proceedings of ACM MOBICOM, July 2001, pp. 287–297.
[33] L.A. Grieco, S. Mascolo, Performance evaluation and comparison of Westwood+, New Reno and Vegas TCP congestion control, ACM Computer Communication Review 34 (2) (2004) 25–38.
[34] S. Floyd, HighSpeed TCP for large congestion windows, IETF RFC 3649, December 2003.
[35] T. Kelly, Scalable TCP: improving performance in highspeed wide area networks, Computer Communication Review 32 (2) (2003) 83–91.
[36] D. Leith, R. Shorten, H-TCP: TCP for high-speed and long-distance networks, in: Proceedings of PFLDnet, 2004.
[37] I. Rhee, L. Xu, CUBIC: a new TCP-friendly high-speed TCP variant, SIGOPS Operating Systems Review 42 (5) (2008) 64–74.
[38] NS-2 Network Simulator. <http://www.isi.edu/nsnam/ns/>.

[39] R. Prior, SIP module for NS-2. <http://www.dcc.fc.up.pt/~rprior/ns/index-en.html>.
[40] 3rd Generation Partnership Project. <http://www.3gpp.org>.
[41] R. Jain, K.K. Ramakrishnan, Congestion avoidance in computer networks with a connectionless network layer: concepts, goals and methodology, in: Proceedings of Computer Networking Symposium, 1988, pp. 134–143.
[42] D. Bertsekas, R. Gallager, Data Networks, Prentice Hall, 1987.
[43] R. Jain, D.-M. Chiu, W. Hawe, A quantitative measure of fairness and discrimination for resource allocation in shared systems, Digital Equipment Corporation, Technical Report TR-301, September 1984.