Local SIP Overload Control: Controller Design and Optimization by Extremum Seeking

Digital Object Identifier 10.1109/TCNS.2015.2401171

Local SIP Overload Control: Controller Design and Optimization by Extremum Seeking

Luca De Cicco, Giuseppe Cofano, Saverio Mascolo

***Abstract*—The Session Initiation Protocol (SIP) is a signaling protocol for managing various types of real time sessions between parties over an Internet Protocol network. An open issue is the control of overload situations that occur when the incoming flow of requests to a SIP server overcomes the server processing capacity. In particular, call establishment times increase due to overload, which triggers retransmissions and causes a further increase of the total incoming flow of requests. This paper proposes an overload control system for regulating both the queue length and the CPU load of the SIP server. The proposed control system is made of two PI controllers tuned by minimizing a proper cost function using the Extremum Seeking algorithm. A real working implementation of the SIP overload controller has been made in the open source SIP server Kamailio. A performance evaluation and comparison of the proposed controller with the main proposals existing in literature has been carried out. Results show that the proposed control system counteracts overload situations and provides a goodput close to the optimal while maintaining low call establishment delays and retransmission ratios.**

I. Introduction

The Session Initiation Protocol (SIP) [1] is an application-layer control (signaling) protocol for establishing, modifying and terminating various types of real time sessions between communication end-points over an Internet Protocol network. In order to establish a real time session, two peers (or user agents) exchange SIP messages which are forwarded by intermediate SIP servers. It is worth noting that today SIP is the major signaling protocol used by Voice over IP (VoIP), instant messaging, and video conferencing applications.

A key open issue that requires to be addressed is the proper handling of overload situations that in a SIP server occurs when the incoming rate of SIP messages, originated by user agents (UA), exceeds the processing capacity of the SIP server. Overload may be caused by poor capacity planning, component failures, avalanche restart, flash crowds and denial of service attacks [2]. SIP sessions are usually transported over the UDP protocol [3], which is a best effort transport protocol without congestion control and retransmission of packets. For this reason lost packets are recovered by the SIP by means of a retransmissions mechanism. During an overload episode, retransmissions occur and, as a consequence, the total incoming load increases, potentially leading the entire SIP network to collapse [2], [4]. To prevent the uncontrolled increase of message retransmissions, an overloaded SIP server can *reject* a message by sending to the originating user agent (UA) a response message with the *503 "Service Unavailable"* code. When an UA receives a *503* response message it should not retransmit the rejected message [1]. Unfortunately, it is well-known that this simple control mechanism is not able to effectively avoid overload [2], [4].

To address this important issue, several overload control algorithms have been proposed and the IETF has established the working group *SIP Overload Control*. Overload control algorithms can be clustered into three categories (see [4] for a comparison): 1) *local overload control*: the algorithm is executed locally on the SIP server and it is based only on local measurements; 2) *hop-by-hop*: the algorithm receives feedback information from the next-hop SIP server; 3) *end-to-end*: the algorithm is executed at the UA and the control loop is closed between the UA and the final SIP server, thus each server on the routing path can reject a message before arriving to the destination based on feedback information received from any server on the routing path.

In the last two decades, a significant effort has been deployed to set in a mathematical control framework the analysis and design of control algorithms for computer networks in a wide range of applicative domains such as congestion control [5], [6], [7], [8], [9], active queue management [10], [11], [12], load balancing [13], multimedia content delivery [14], [15], admission control [16], [17].

In this paper we focus on the design of a feedback control system for local SIP overload control. In particular, the main contributions of the paper are:

1) a mathematical model for an ideal rate-based overload control algorithm which gives an upper bound to the goodput achievable by such algorithms;
2) a local SIP overload control system, based on two PI control loops; the first PI controller drives the queue length to a set-point to guarantee a target response time, while the second one drives the CPU utilization to a target;
3) a functional is proposed as a performance metric for local overload control algorithms;
4) a real implementation of the proposed control system has been made in the open source SIP server Kamailio (OpenSER);
5) based on this implementation and on the proposed performance metric, the controllers parameters have been tuned using the Extremum Seeking algorithm [18];
6) finally, a performance evaluation and comparison with two local overload controllers proposed in literature is

The authors are with the Dipartimento di Ingegneria Elettrica e dell'Informazione, Politecnico di Bari, Via Orabona 4, Bari, Italy. emails: name.surname@poliba.it

This work has been partially supported by the project "Platform for Innovative services in the Future Internet" (PLATINO-PON01 01007) funded by Italian Ministry of Education, Universities and Research (MIUR).

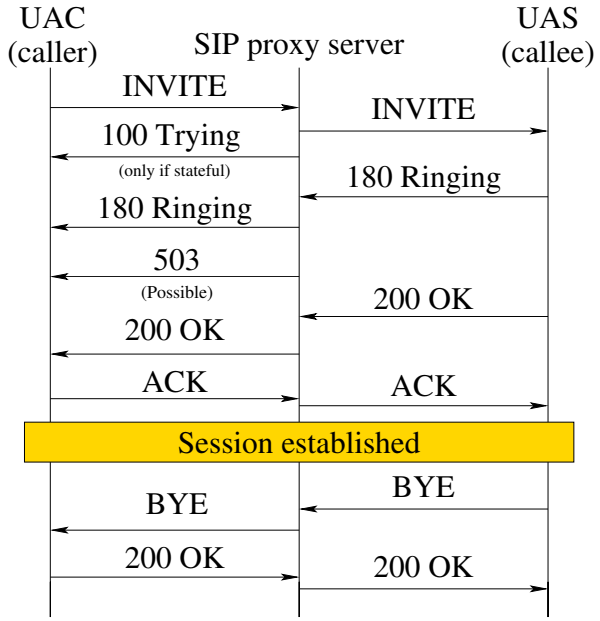


Figure 1. Time sequence graph of a SIP call

carried out.

The rest of the paper is organized as follows: in Section II a brief overview of the SIP protocol is given; Section III provides the state of the art of the proposed overload control mechanisms; Section IV presents the proposed feedback overload control algorithm; in Section V the implementations details of the proposed controller are given; Section VI is devoted to the tuning of the two PI controllers by using ES; Section VII shows the results of the experimental evaluation and, finally, Section VIII concludes the paper.

II. SIP Overview

SIP is a client-server message-based protocol for managing real time sessions. Two logical entities participate in SIP communications: SIP User Agents (UAs) and SIP servers. SIP servers can be further classified as: proxy servers, for session routing, and registrar servers, for UAs registration.

Fig. 1 shows the time sequence graph of the establishment of a SIP call session. The caller, the originating UA Client, sends an *INVITE* request to the callee, the terminating UA Server, through one or more proxy servers. In the case the proxy is stateful, the proxy server returns a provisional *100 Trying* response for confirmation. The terminating UA returns a *180 ringing* response after confirming that the parameters are appropriate. It also sends a *200 OK* message to answer the call. Then, after the *200 OK* message has been received, the originating UA sends an *ACK* response to the terminating UA and the call is said to be established. Finally, one of the two peers sends a *BYE* request to close the session.

In the case the SIP messages are sent over UDP, which today is the most common choice, SIP employs a retransmission mechanism to cope with packet losses. If an *INVITE* message is sent, and a suitable[1] reply message is not received before the *Timer A* expires, the *INVITE* message is retransmitted [1]. The duration of *Timer A* gets doubled each time the same *INVITE* is retransmitted, i.e. $Timer\ A = T_1 \cdot 2^i$, where T_1 is equal to 0.5s by default and i is the number of times the *INVITE* has been retransmitted.

In [1] it is specified that retransmissions are stopped either when a provisional response is received or when the timeout value exceeds *Timer B,* that by default is equal to 32s. When an overload situation is detected, SIP servers send a *503 "Service Unavailable*" to prevent retransmissions [2]. In this case the incoming message is said to be *rejected*.

[1]In the case of a stateless SIP proxy the reception of a 1XX, 2XX, or 5XX reply message prevents *Timer A* from firing.

III. Related Work

SIP overload control techniques are discussed in detail in [19]. In particular, a taxonomy of the overload control schemes is given according to: 1) how overload is detected; 2) the control actuation strategy and 3) the employed control architecture. Overload can be detected either explicitly, when servers signal an overload episode, or implicitly, when the other servers of the network detect it by means of measurements. The control actuation can be rate-based, loss-based, signal-based or window-based. The control architecture can be local, hop-by-hop or end-to-end. Moreover, the following performance metrics are proposed to evaluate a SIP overload control technique: 1) the goodput and the response time of the SIP server and 2) the responsiveness and the stability of the proposed algorithm.

Several local overload control algorithms have been proposed in the literature. A literature review of the existing SIP overload control solutions is provided in [20]. The first local overload control mechanism specifically designed for SIP has been proposed by Ohta in [21]: the algorithm decides to either reject or accept a new SIP session based on the queue length. Another well-known example of local control is Local Occupancy (OCC) [22], [4]. The algorithm rejects a fraction of *INVITE*s according to a simple control law to drive the CPU load to a target utilization.

The first comprehensive study on SIP overload control was the one by Hilt and Widjaja [4] in which local, hop-by-hop, and end-to-end overload control algorithms are compared through simulations.

Distributed overload control algorithms have attracted a great deal of attention due to the promise of providing better performance. In [22] authors have defined two overload categories: (i) server to server overload and (ii) client to server overload. In [22] only the first case has been considered and three hop-by-hop window-based feedback algorithms have been proposed. The upstream server employs a feedback information sent by downstream servers to dynamically set the transmission window size and counteract overload. Three different window adjustment algorithms have been compared to two rate-based algorithms. It has been shown that the window-based algorithms achieve better results. However, the algorithms proposed in [22] require an exchange of feedback information between two adjacent servers.

A feedback-based algorithm has been proposed in [23] to regulate retransmission ratios during overload. In the paper

retransmissions have been classified into two categories: *redundant*, in the case they are caused by delay due to overload, and *non-redundant,* if they are due to message loss recovery. A PI controller has been designed to regulate the retransmissions rate at the upstream server to track a set-point of redundant messages rate, thus mitigating overload of the downstream server without requiring an explicit feedback. In [24] an overload mechanism combining local and remote control has been proposed, the former using a priority FIFO queue at the SIP proxy during overload episodes, the latter based on a prediction technique placed at the remote control loop.

In [25] an end-to-end overload control has been designed which does not require any modification of the SIP protocol. A backpressure-based technique, originally conceived for multi-hop radio networks, has been adapted to the SIP networks context.

In [26] a distributed end-to-end adaptive window-based overload control technique is proposed. Upstream servers use call establishment delay to infer the load on their downstream servers so that an explicit feedback from the downstream servers is not needed. In particular, average call establishment delay and its standard deviation are employed to implement a threshold-based overload detection mechanism. This mechanism triggers the transitions between the additive increase and the multiplicative decrease phases (AIMD) of the transmission window updating process.

In [27] a distributed end-to-end overload control mechanism is presented. The key idea is to deploy the controllers at the edge servers to implement a call admission control to the SIP network. The controller computes the call admission rate based on a finite state machine that employs the rate of *503 "Service Unavailable*" responses, which are received from the core servers, as an indication of overload. The call admission rate is actuated using the call gapping technique.

A protocol for communicating overload information between SIP servers and clients has been proposed in [28]. Four *Via* header parameters have been introduced in the SIP protocol to allow SIP servers and clients to communicate traffic reduction requests, to support multiple classes (rate-based, loss-based, etc) of hop-by-hop overload control algorithms and to dynamically change the duration of the overload control action. Thanks to the hop-by-hop approach it is not necessary that each server on the end-to-end path supports the protocol, therefore the backward compatibility is ensured.

Finally, it is worth noting that performance evaluation of the proposed controllers in [4], [21], [22], [29], [24], [27] has been carried out only by using discrete events simulators whereas an experimental evaluation is not provided. The only exception is [26], which evaluates the performance mostly through simulations but also considers a brief experimental evaluation.

IV. The Proposed Control System

In this Section we propose a local overload control algorithm. Such an approach does not require to modify the SIP protocol and can be rapidly deployed in SIP proxies. The proposed algorithm is intended to guarantee that the downstream servers of the controlled server do not receive a calls rate higher than the capacity of the controlled server. This makes it suitable for use in edge servers of a SIP network in order to absorb excessive incoming load and protect the core servers. Moreover, it can be employed in conjunction with a distributed strategy to protect the server in the case the distributed controllers do not work properly. The interaction between the two strategies is, however, outside the scope of this work.

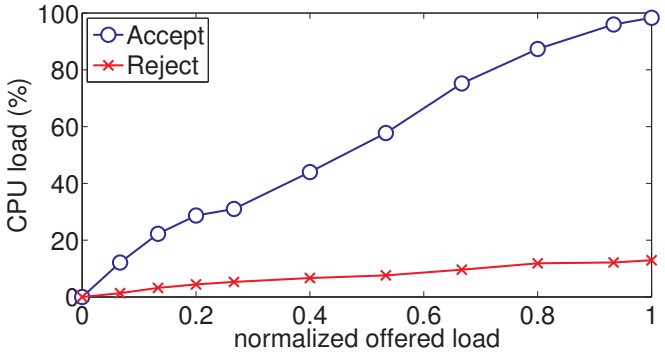


Figure 2. CPU load as a function of the normalized incoming INVITEs rate in the cases of either forwarding or rejecting them

A. The optimal goodput function

In this paper we focus only on *INVITE* transactions since they are the most CPU-expensive messages to be handled by a SIP server [30]. Moreover, we make the basic assumption that the ratio β between the cost of forwarding and the cost of rejecting an *INVITE* is constant and larger than 1 ($\beta > 1$). This assumption has been experimentally validated by running a series of load tests on a SIP proxy server[2]. We have measured the average CPU load of the proxy server at different incoming rates in the case of forwarding and rejecting *INVITE* messages. Fig. 2 shows the CPU load as a function of the normalized incoming *INVITE*s rate. Both curves can be approximated by linear functions, whose ratio is the constant β.

For convenience of notation we normalize by the forwarding cost of one *INVITE* message, obtaining unitary forwarding cost and rejecting cost equal to $1/\beta < 1$. Let $\rho(t)$ denote the incoming load of *INVITE* messages measured in calls per second (cps) and $C(t) \in [0, 1]$ the instantaneous CPU load. As $\rho(t)$ increases, $C(t)$ will increase until the point it reaches its maximum value 1 and overload occurs. We denote with ρ_M the maximum offered load the SIP server can manage without suffering overload, and we define the normalized incoming *INVITEs* rate as $r(t) = \rho(t)/\rho_M$. Finally, the normalized goodput $g(t)$ is the rate of successfully established calls divided by ρ_M. From now on, we will consider only the normalized load $r(t)$ and goodput $g(t)$.

Let us now consider the overall system composed of a generic rate-based local overload controller and the proposed CPU model. The controller computes the fraction $\alpha(t)$ of

[2]We have employed the experimental testbed which is described in detail in Section V.

incoming *INVITE* rate $r(t)$ to be rejected. The CPU load can be modelled as follows:

$$C(t) = (1 - \alpha(t))r(t) + \frac{1}{\beta}\alpha(t)r(t) + d(t) \tag{1}$$

where the first additive term is the CPU load due to the accepted rate, the second one is due to the rejected rate, and the third one models the CPU load due to other processes in execution, which is considered as a disturbance[3].

The following proposition gives an upper bound to the goodput obtainable by any local SIP overload controller based on the model (1):

Proposition 1: Let β denote the ratio between the cost of accepting and the cost of rejecting an *INVITE* message, and let C_T be the target CPU load set for the SIP server. Then the goodput $g(r)$, function of the normalized incoming rate r, obtainable by any local SIP overload controller is bounded by:

$$g_{opt}(r) = \begin{cases} r & r < C_T, \\ -\frac{1}{\beta-1}r + \frac{C_T\beta}{\beta-1} & C_T \leq r < \beta C_T \ , \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Proof: Since we are interested in characterizing the optimal goodput, we consider $d(t) = 0$. When $r < C_T$ the server does not need to drop any INVITE, thus the goodput g is equal to the normalized incoming rate r. For input rates greater than C_T we can use (1) to evaluate $g_{opt}(r)$ and obtain:

$$\alpha(r) = \frac{\beta}{\beta - 1}\left(1 - \frac{C_T}{r}\right). \tag{3}$$

Then, the normalized optimal goodput in the overload zone can be computed as $(1 - \alpha(t)) \cdot r$, that is the fraction of messages forwarded by the SIP server:

$$g_{opt}(r) = -\frac{1}{\beta - 1}r + \frac{C_T\beta}{\beta - 1}. \tag{4}$$

Eq. (3), and consequently (4), is valid until the point all requests are rejected, i.e. $\alpha = 1$. By imposing $\alpha = 1$ in (3), we obtain the maximum rate r_{max} that the SIP server can handle while keeping the CPU utilization equal to C_T:

$$r_{max} = C_T\beta. \tag{5}$$

Thus, when the input rate $r > C_T\beta$ the goodput is zero and the server is not able to avoid overload. ■

Remark 1: Eq. (2) expresses the ideal limit of any local overload control algorithm in which the control variable is the reject rate: Eq. (5) shows that the normal behaviour of a SIP server can be extended even when the normalized incoming *INVITEs* rate is greater than 1, that is the limit at which an uncontrolled SIP server gets overloaded.

B. The design of the proposed control system

The overall objective of the control system is to ensure that the CPU does not get overloaded. Towards this end, it is necessary to ensure that the CPU utilization does not get above a fixed threshold. The fixed threshold should be chosen to leave spare CPU cycles to other concurrent system processes that are required to execute management tasks on the server. We start by making the assumption that the SIP proxy server is able to store incoming messages in a queue that will be drained by an asynchronous worker thread. Fig. 3 shows the proposed control system which is made of two feedback loops: the first controller, depicted in the topmost box, steers the queue level $q(t)$, measured in number of *INVITE*s, to a target $q_T(t)$; the other one steers the CPU load $C(t)$ to a desired target $C_T < 1$.

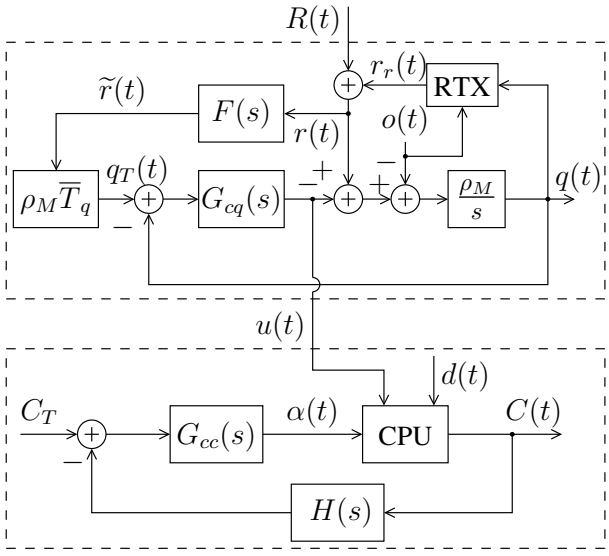


Figure 3. The proposed control system

1) The queue controller: The goal of this control loop is to compute the queue draining rate $u(t)$, i.e. the normalized rate of *INVITE*s to be processed by the CPU, so that the queuing time of incoming *INVITE* messages is well below the first retransmission timeout $T_1 = 0.5$s. In such a way the control system is able to prevent retransmissions. The queue is assumed to be a FIFO drop-tail buffer, with a maximum length q_M of *INVITE* messages, whose length $q(t)$ can be modelled as follows:

$$q(t) = \rho_M \int_0^t (r(\sigma) - u(\sigma) - o(\sigma))d\sigma$$

where $o(t)$ is the queue overflow rate and $r(t)$ is the total incoming rate equal to the sum of $R(t)$, which is the rate of new *INVITE*s, and $r_r(t)$, which is the retransmission rate generated due to the local server. The retransmission rate due to other servers is included in $R(t)$. The overflow rate $o(t)$ can be modelled as follows [6]:

$$o(t) = \begin{cases} r(t) - u(t) & q(t) = q_M, r(t) > u(t) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

which is the rate of dropped messages when the queue is full and the input rate exceeds the output rate. The retransmission rate, which in Fig. 3 is the output of the block "RTX", is caused by the expiration of a complex mechanism of timeouts of variable duration which can be caused by either

[3]The computational cost of the control action has to be considered as a component of the disturbance. Hence, the simplicity of the control system is a key design requirement.

large queuing time $T_q(t)$ or queue overflow rate $o(t) > 0$. Unfortunately, the fluid model of the retransmission rate $r_r(t)$, which would be helpful in the design of the control system, is complex and intractable, indeed it is still not available in the literature.

To keep $r_r(t)$ close to zero, the controller computes the draining rate $u(t)$ so that the queuing time $T_q(t)$ is steered to a set-point $\overline{T}_q$ much lower than the first retransmission timeout. A simple way to do it is to exploit the approximation $T_q(t) = q(t)/(\rho_M u(t))$, with $u(t) \neq 0$ [10]. Using this approximation, $T_q(t)$ can be indirectly controlled by controlling the queue length $q(t)$. In particular, in our proposed control loop $q(t)$ tracks the reference signal $q_T(t) = \overline{T}_q \rho_M \widetilde{r}(t)$, where $\widetilde{r}(t)$ is a low-pass filtered version of the measured total incoming rate $r(t)$. We have set $\overline{T}_q = 50$ms, which is 10 times lower than the first retransmission timer T_1. A first order low-pass filter (LPF) $F(s) = 1/(1+\tau s)$, with $\tau = 0.1$s, cuts off the high frequency components of $r(t)$.

The controller is a PI with proportional gain K_{pq} (measured in s^{-1}) and integral gain K_{iq} (measured in s^{-2}):

$$u(t) = \frac{1}{\rho_M}(K_{pq} e_q(t) + K_{iq} \int_0^t e_q(\tau) d\tau) \tag{7}$$

where $e_q(t) = q_T(t) - q(t)$ is the error and ρ_M is a scaling factor due to the assumption that $u(t)$ is dimensionless.

2) The CPU controller: The CPU controller $G_{cc}(s)$ computes the fraction $\alpha(t)$ of messages to reject by using 503 messages. The CPU control goal is to steer the CPU load $C(t)$ to the desired value C_T. It is worth noting that the output of the first controller, which is the queue draining rate $u(t)$, tracks $r(t)$ at steady state and can be considered as a disturbance acting on the control loop.

The reject ratio $\alpha(t)$ is computed based on the error $e_c(t) = \widetilde{C}(t) - C_T$, where C_T is the target load for the SIP server and $\widetilde{C}(t)$ is the CPU load filtered by $H(s) = 1/(1+\tau_c s)$ with $\tau_c = 0.1$ s. Again, we employ a proportional-integrative controller, whose equation is given by

$$\alpha(t) = K_{pc} e_c(t) + K_{ic} \int_0^t e_c(\tau) d\tau. \tag{8}$$

By taking the derivative of (8), and considering that $\dot{\widetilde{C}}(t) = C(t)/\tau_c - \widetilde{C}(t)/\tau_c$ where $C(t)$ is given by (1), the following mathematical model can be easily derived:

$$\begin{cases} \dot{\alpha}(t) = & \frac{K_{pc}}{\tau_c}[u(t)\cdot(1-\gamma\alpha(t)) + d(t) - \widetilde{C}(t)] + \\ & + K_{ic}(\widetilde{C}(t) - C_T) \\ \dot{\widetilde{C}}(t) = & \frac{1}{\tau_c} u(t)(1-\gamma\alpha(t)) + \frac{1}{\tau_c} d(t) - \frac{1}{\tau_c}\widetilde{C}(t) \end{cases} \tag{9}$$

where we have posed $\gamma = (\beta - 1)/\beta$ for brevity of notation.

Proposition 2: Considered the equilibrium inputs $(C_T, \overline{u}, \overline{d})$ in the positive orthant of $\mathbb{R}^3$, the system (9) has a unique equilibrium point:

$$\begin{aligned} \overline{\alpha} &= \frac{\beta}{\beta - 1}\left(1 - \frac{C_T}{\overline{r}} + \frac{\overline{d}}{\overline{r}}\right) \\ \overline{C} &= C_T \end{aligned} \tag{10}$$

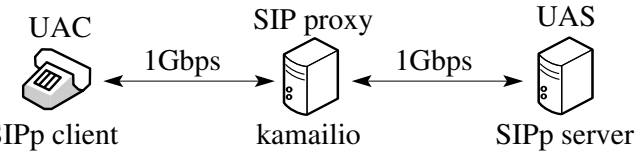


Figure 4. The testbed employed for the experimental evaluation

which is locally asymptotically stable provided K_{ic}, K_{pc}, τ_c are positive.

Proof: In order to find the equilibrium point, we impose that the derivatives of (9) are equal to zero. Recalling that $\overline{u} = \overline{r}$ and solving the system we obtain the equilibrium (9). The state matrix A is equal to:

$$A = \begin{bmatrix} -\frac{K_{pc}}{\tau_c}\gamma\overline{u} & K_i - \frac{K_{pc}}{\tau_c} \\ -\frac{1}{\tau_c}\gamma\overline{u} & -\frac{1}{\tau_c} \end{bmatrix}$$

whose characteristic equation is:

$$p(\lambda) = \lambda^2 + \frac{1}{\tau_c}(1 + \gamma K_{pc}\overline{u})\lambda + \gamma\frac{K_{ic}}{\tau_c}\overline{u} = 0$$

The proposition is proved by observing that the characteristic equation has all its roots in the left half-plane provided that K_{ic}, K_{pc}, and τ_c are positive. ■

The equilibrium depends only on the inputs $\overline{u}$, $\overline{d}$, C_T, and on β which is a characteristic parameter of the considered SIP server. It is worth to notice that the controller gains do not affect the equilibrium.

V. Real System Implementation

The proposed overload control system described in Section IV has been implemented in a module of the open source Kamailio SIP proxy server[4] to optimize the controller using the Extremum Seeking algorithm and to compare its performance to Ohta [21] and OCC [4] overload controllers.

Fig. 4 shows the testbed employed for the experimental evaluation. Two Linux PCs are connected through a 1000 BaseT Ethernet LAN. We have adopted a point-to-point topology by using SIPp[5] to generate a configurable *INVITE* rate. SIPp has been also used to emulate the upstream SIP server. The SIPp client and server ran over the faster PC, a Intel Pentium 4 with 3.60 GHz clock speed and 2 GB of RAM. The modified Kamailio SIP server, configured in the transaction-stateless mode with no authentication, ran as proxy server over the slower PC, a Intel Pentium III with CPU clock speed of 1 GHz and 756 MB of RAM. The SIP server PC employs Ubuntu Server 11.10.

Kamailio is an open source SIP proxy server that is widely employed both for scientific and commercial purposes. Its architecture is made of a *core,* providing basic SIP server functionalities, and several pluggable modules which extend the core. The Kamailio core processes the incoming messages synchronously with their arrival and thus it does not implement a queuing structure. Since both Ohta and the proposed control system require the incoming messages to be enqueued before

[4]http://www.kamailio.org/

[5]http://sipp.sourceforge.net/

being processed, we have implemented a queue in the core where the *INVITE* messages are stored before being processed. The three overload controllers have been implemented in the Kamailio module named `ratelimit`.

Both PI controllers of the proposed control system have been discretized and provided with an anti-wind up scheme to cope with saturation of the actuation variables. Two asynchronous timer functions are responsible for, respectively, CPU load sampling and controller actuation. Their sampling times T_c and T_m have been set to 10ms. Finally, the maximum queue length q_M has been set equal to 800 *INVITE*s.

VI. Controller Optimization

The proposed control system is made of two PI controllers, resulting in four parameters to be tuned. Many methods and PID tuning rules have been proposed in the literature so far. However, optimal tuning of the considered system is made complex due to the following reasons: 1) the parameters vector $\theta = [K_{pc}, K_{ic}, K_{pq}, K_{iq}]$ has four components; thus, using standard performance mapping over the whole parameters space would require a very large number of experiments; 2) a complete model of the considered system is not available due to the lack of a mathematically tractable model of the retransmission rate $r_r(t)$.

In the following we use the iterative Extremum Seeking algorithm (ES) [31], [18] to tune the controllers, which does not require the knowledge of the mathematical model of the system. ES has been applied successfully in different applicative fields ranging from automotive applications to sensor networks [32], [33], [34], [35]. In the following we describe the ES version we have employed (Section VI-A), the cost function which we shall minimize (Section VI-B), and finally we show the experimental results of the optimization (Section VI-C).

A. The Extremum Seeking algorithm

ES is an optimization algorithm which iteratively modifies the controller parameters θ to minimize a cost function $J(\theta)$. $J(\cdot)$ is a static map which establishes a steady-state relationship between the parameters vector θ and the obtained performance. ES does not use an explicit mathematical relation of $J(\theta)$, which is considered unknown, but it makes the assumption that, once θ is fixed, it is possible to experimentally measure $J(\theta)$. Fig. 5 shows the iterative optimization technique: 1) the controller is configured with the parameters vector $\theta(k)$; 2) a series of step-response experiments is carried out and $J(\theta(k))$ is measured (see Section VI-B); 3) the ES algorithm computes the new value of the parameters vector $\theta(k+1)$.

In this paper, we employ the implementation of ES presented in [18], modified by adding an anti-windup scheme, as suggested in [36], to cope with the choice of unfeasible negative parameters. The resulting ES algorithm, whose block diagram is shown in Fig. 5, is described by the following

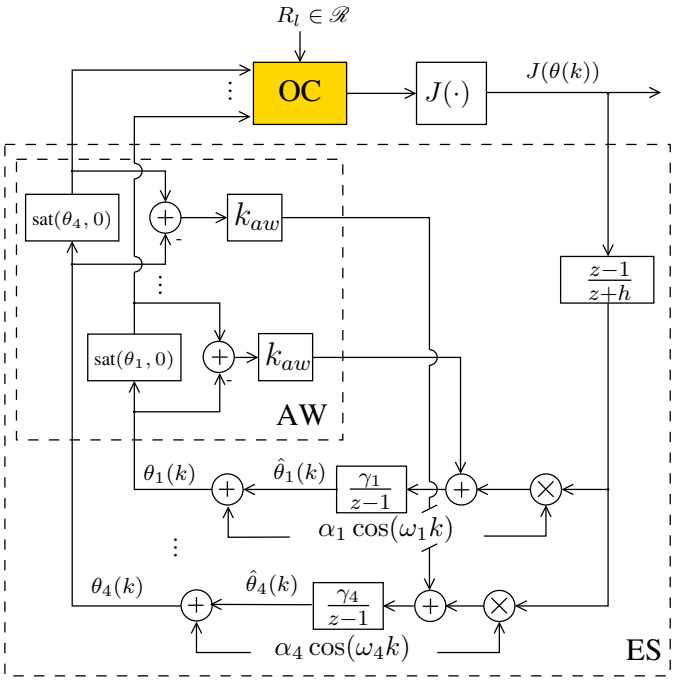


Figure 5. Block diagram of the discrete extremum seeking algorithm

equations:

$$\zeta(k) = -h\zeta(k-1) + J(\theta(k-1)) \tag{11}$$

$$\hat{\theta}_i(k+1) = k_{aw}(\text{sat}(\theta_i(k), 0) - \theta_i(k))) + \hat{\theta}_i(k) + \\ - \gamma_i(\alpha_i \cos(\omega_i k)(J(\theta(k)) - (1+h)\zeta(k)) \tag{12}$$

$$\theta_i(k+1) = \hat{\theta}_i(k+1) + \alpha_i \cos(\omega_i \cdot (k+1)) \tag{13}$$

where the saturation function sat$(\theta_i, 0)$ is given by:

$$\text{sat}(\theta_i, 0) = \begin{cases} 0 & \theta_i < 0 \\ \theta_i & \text{otherwise} \end{cases}$$

Eq. (13) computes the new parameters vector based on the output of the ES algorithm for $i = 1, 2, 3, 4$. In particular, ES obtains an estimate of the gradient $\nabla J(\theta(k))$ by extracting the portion of $J(\theta(k))$ that is due to the perturbation of the parameters estimate $\hat{\theta}(k)$. Equation (13) shows that ES sinusoidally perturbs the input parameters $\hat{\theta}(k+1)$ to give the new parameters vector $\theta(k+1)$. At the end of the k-th experiment, the measured value of $J(\theta(k))$ is high-pass filtered in order to remove its DC portion and demodulated by multiplication with a discrete-time sinusoid with the same frequency of the perturbation signal. Finally, the output, corresponding to the gradient estimate, is low-pass filtered by the integrator with a step size γ, giving the new parameters estimate $\hat{\theta}(k+1)$. Finally, the term $k_{aw} \cdot (\text{sat}(\theta_i(k), 0) - \theta_i(k))$, that is zero unless $\theta_i(k) < 0$, implements a static anti-windup back-calculation scheme that is provided as an input to the integrator block.

B. The cost functions

We evaluate the cost function at the end of a series of step-response experiments, each one 120s long, with input rates $R(t) = R_l \cdot 1(t)$, where $R_l \in \mathscr{R} = \{1.58, 2.10, 2.37, 2.63\}$ and

$1(t)$ is the step function. For each $R_l \in \mathscr{R}$ we have repeated the experiment 6 times and considered the best $m = 4$ values to rule out possible outliers due to experimental testbed issues.

The cost function has been carefully chosen. In principle, it would appear natural to employ a goodput-based functional to maximize the average goodput of the server. In particular, a suitable candidate functional based on the goodput $g(\theta, r)$ could be:

$$G(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{R_l \in \mathscr{R}} g(\theta, R_l \cdot 1(t)) \cdot R_l \qquad (14)$$

where the negative sign is due to the fact that ES finds minima of the cost function. For each $R_l \in R$ the goodput is weighted by a factor R_l to ensure that experiments with different loads have the same contribute to the cost $G(\theta)$.

However, we argue that $G(\theta)$ is not a good choice. In fact, a simple yet wrong way to improve the goodput would be by increasing the CPU load over the set-point C_T, which must be chosen to leave a safety margin on full CPU utilization. In other words, a finite tracking error on the CPU set-point could result in a goodput improvement. In Section VI-C we show that in some cases this phenomenon occurs due to actuator saturation. However, the maximization of the goodput at the expense of meeting control specifications is undesirable, since it reduces the safety margin and makes the system less robust to CPU disturbances.

For this reason we have employed a functional $J(\theta)$ based on the integral absolute errors on the retransmissions ratio and the CPU load, which are direct indicators of overload. In particular, $J(\theta)$ is the linear combination of the average values of two integral absolute errors over the m experiments as follows:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{R_l \in \mathscr{R}} (\eta \text{RIAE}_i(\theta, R_l \cdot 1(t)) + \xi \text{CIAE}_i(\theta, R_l \cdot 1(t))). \qquad (15)$$

$$\text{RIAE}_i(\theta, R_l \cdot 1(t)) = \frac{1}{t_f}\int_0^{t_f} |\tilde{r}(t, \theta) - R_l| dt \qquad (16)$$

is the measured absolute integral of the retransmission ratio for the i-th experiment, and

$$\text{CIAE}_i(\theta, R_l \cdot 1(t)) = \frac{1}{(1 - C_T)t_f}\int_0^{t_f} |\widetilde{C}(t, \theta) - C_T| dt \qquad (17)$$

is the measured CPU integral absolute error for the i-th experiment. The error $e_c(t)$ is normalized by its maximum value $1 - C_T$. η and ξ are weighting parameters that can be freely adjusted to emphasize one term over the other. We have found that $\eta = 1$ and $\xi = 2$ ensure a good shaping of the cost function. It is worth noting that, with this cost function, the goodput can be improved by simply employing a larger C_T, since in any case the functional $J(\cdot)$ ensures that the selected optimal parameters are such that the CPU integral absolute error is minimized.

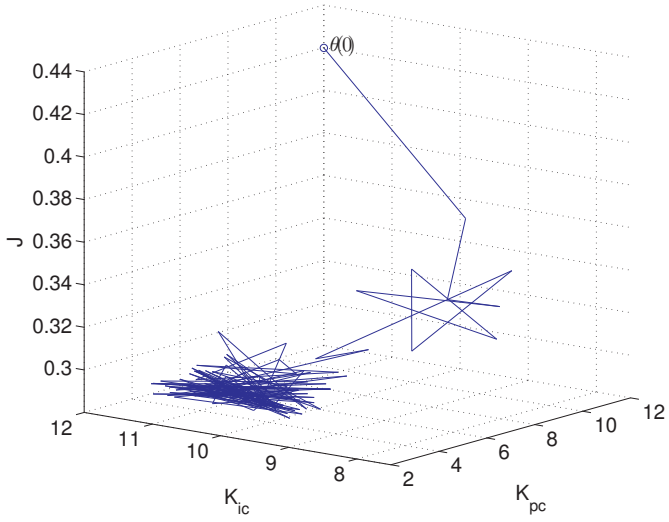


Figure 7. $J(\theta(k))$ function of K_{pc} and K_{ic} for $\theta(0) = \theta_1 = [12, 12, 20, 130]$

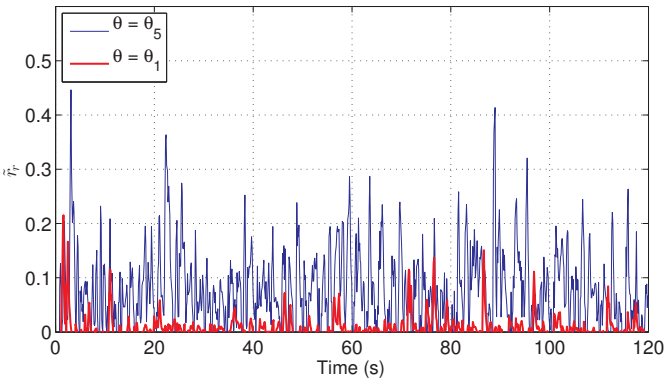


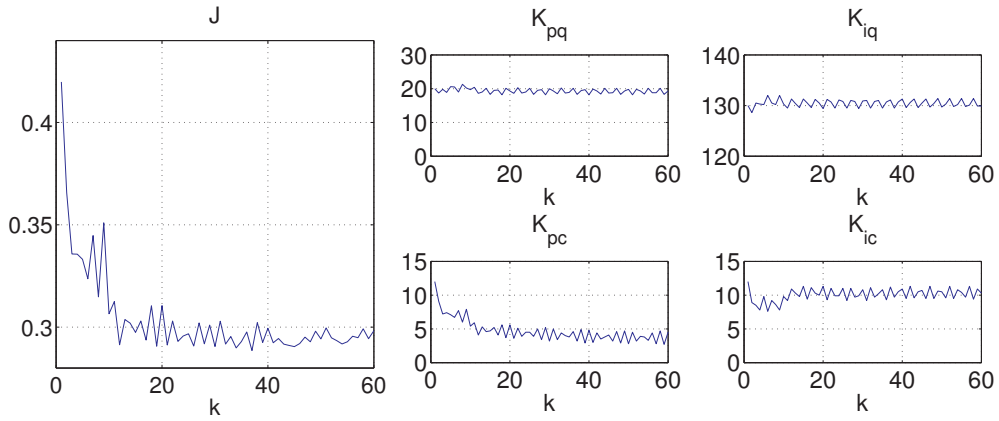
Figure 8. Retransmission rate comparison for $\theta = \overline{\theta}_1$ and $\theta = \overline{\theta}_5$ in the case of $R = 2.75$

C. Experimental results

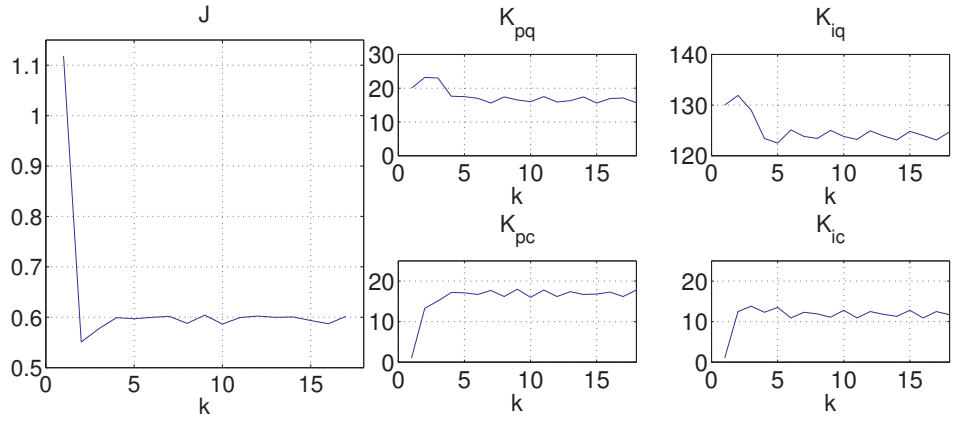
In this section we show the experimental results of the parameters tuning using the ES algorithm described in Section VI-A. We have adopted a step size $\gamma = [10, 10, 10, 10]$, a high-pass filter parameter $h = 0.5$, a sinusoidal amplitude and pulsation equal to $\alpha = [1, 1, 1, 1]$ and $\omega = [2.82, 2.54, 2.28, 2.06]$ respectively. The anti-windup gain k_{aw} has been set to 0.003.

We have performed a series of ES runs employing different initial conditions to investigate the presence of multiple local minima. The results are reported in Table I. Two different minima have been found: 1) Runs from 1 to 4 have converged in the neighborhood of different points with the same value of $K_{pc} = 3.5$ and the same optimal value $J(\theta) = 0.29$; 2) Run 5 has converged to the neighborhood of $\overline{\theta}_5 = [17, 12, 16.5, 124]$, where $J(\overline{\theta}_5) = 0.6$. In the following we consider Run 1, which is representative of the case $J(\theta) = 0.29$, and Run 5 to compare performance differences and to justify the preference of the functional $J(\cdot)$ (see (15)) over the goodput-based functional $G(\cdot)$ (see (14)).

Let us start by analyzing Run 1. Fig. 6 (a) shows the evolution of both $J(\theta(k))$ and the parameters vector $\theta(k)$ in



(a) $\theta(0) = \theta_1 = [12, 12, 20, 130]$



(b) $\theta(0) = \theta_5 = [1, 1, 20, 130]$

Figure 6. Evolution of $J(\theta(k))$ and $\theta(k)$

Table I
OPTIMAL PARAMETERS $\overline{\theta}_i$ FOR DIFFERENT INITIAL CONDITIONS θ_i

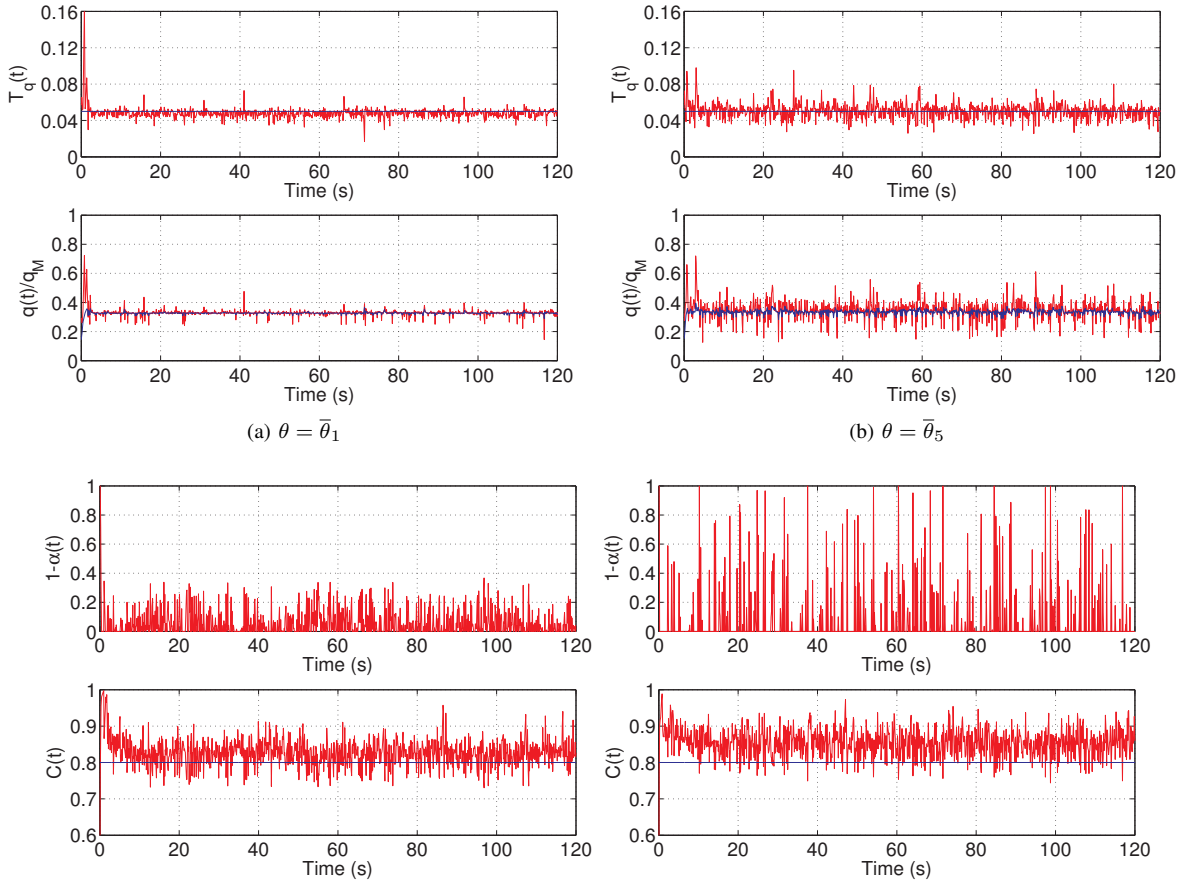
	Initial Conditions θ_i				**Optimal** $\overline{\theta}_i$				
R.	K_{pc}	K_{ic}	K_{pq}	K_{iq}	K_{pc}	K_{ic}	K_{pq}	K_{iq}	$J(\overline{\theta}_i)$
1	12	12	20	130	3.5	10.5	19	130.5	0.29
2	2	2	20	130	3.5	9	20	131.5	0.29
3	-17	12	-16.5	120	3.5	13.5	17	127	0.29
4	0.5	0.5	4	4	3.5	7.5	20	130	0.29
5	1	1	20	130	17	12	16.5	124	0.6

the case $\theta(0) = \theta_1 = [12, 12, 20, 130]$. $J(\theta(k))$ converges in a neighborhood of $\overline{\theta}_1 = [3.5, 10.5, 19, 130.5]$ corresponding to $J(\overline{\theta}_1) = 0.29$. It can be noticed that $J(\theta(k))$ is not very sensitive to variations of K_{pq} and K_{iq}. We argue that these parameters negatively affect the performance of the system only if the settling time of the first control loop is high[6] and, as a consequence, retransmissions are triggered. To get a further insight, let us consider Fig. 7, which shows $J(\theta(k))$ as a function of K_{pc} and K_{ic}. $J(\theta(k))$ is very sensitive to the proportional gain of the CPU loop. In particular, K_{pc} decreases until the point $J(\theta(k))$ eventually reaches a minimum that is obtained for $K_{pc} = 3.5$ and $K_{ic} = 10.3$. Runs 2,3 and 4 have shown a similar behavior. We can conclude that the system performances are highly sensitive to K_{pc} variations.

[6]With $K_{pq}(0) = 20$ and $K_{iq}(0) = 130$ the first loop has damping factor of 0.7 and 2% settling time of 0.1s.

A different behaviour is obtained in Run 5 which converges in a neighborhood of $\overline{\theta}_5 = [17, 12, 16.5, 124]$ corresponding to $J(\overline{\theta}_5) = 0.6$, that is much higher than $J(\overline{\theta}_1)$. We argue that this local minimum of $J(\cdot)$ corresponds to the case of actuator saturation. Fig. 9 compares the dynamics of the CPU load, the accept ratio, the queue length, and the queuing time in the case of $R = 2.75$ when either $\overline{\theta}_1$ or $\overline{\theta}_5$ is employed.

In the case of $\overline{\theta}_5$, Fig. 9 (d) shows that the accept ratio $1 - \alpha(t)$ is close to 0 and, since the anti wind-up disconnects the integral mode when the actuator is saturated, a finite tracking error on the CPU set-point is obtained. On the other hand, in the case of $\overline{\theta}_1$ the CPU set-point is tracked with zero steady-state error, but with a slightly lower goodput. Fig. 8 compares the retransmission rates measured in the case of $R = 2.75$ when either $\overline{\theta}_1$ or $\overline{\theta}_5$ is employed. The figure clearly shows that retransmissions are negligible with $\overline{\theta}_1$, whereas with $\overline{\theta}_5$



(a) $\theta = \overline{\theta}_1$ (b) $\theta = \overline{\theta}_5$

(c) $\theta = \overline{\theta}_1$ (d) $\theta = \overline{\theta}_5$

Figure 9. CPU load, accept ratio, queuing time and queue length comparison for $\theta = \overline{\theta}_1$ and $\theta = \overline{\theta}_5$ in the case of $R = 2.75$

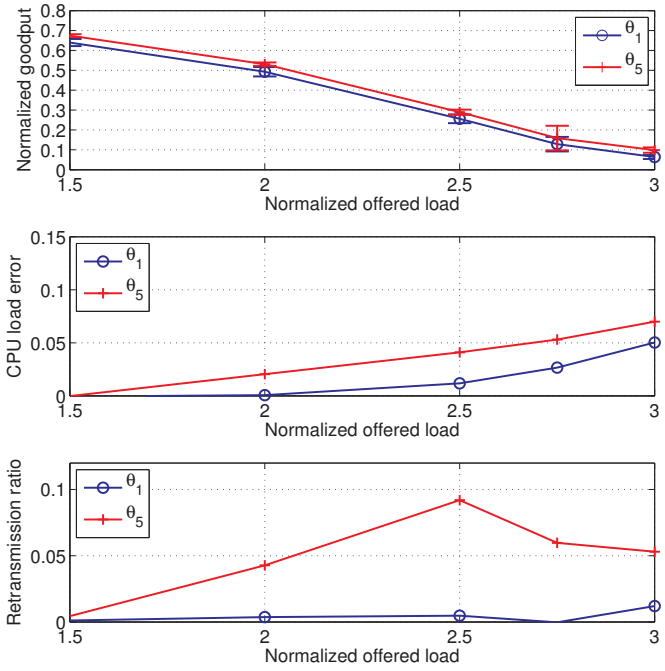


Figure 10. Comparison of goodput, average CPU load error and retransmission ratio for $\overline{\theta}_1$ and $\overline{\theta}_5$

a moderate retransmission rate is measured. Fig. 9 (b) shows when higher retransmission rate are measured ($\theta = \overline{\theta}_5$), larger oscillations of the queuing time $T_q(t)$ and of the normalized queue length $q(t)/q_M$ occur. To summarize, $\overline{\theta}_1$ should be preferred to $\overline{\theta}_5$ since it ensures a perfect tracking of the CPU target, while minimizing queuing times and retransmission rates.

To conclude, we show that the proposed functional $J(\cdot)$ is more suitable than the goodput-based functional $G(\cdot)$ (14) for the considered system. To the purpose, Fig. 10 compares the performance of the system as a function of the normalized load R when either $\overline{\theta}_1$ or $\overline{\theta}_5$ is employed. In particular, with $\overline{\theta}_5$ a slightly higher goodput is achieved at the expense of higher CPU load errors and retransmission rates. It is easy to check that in the case $G(\cdot)$ had been employed, $\overline{\theta}_5$ would have been preferred to $\overline{\theta}_1$ since $G(\overline{\theta}_5) = -6.01 < G(\overline{\theta}_1) = -5.73$. On the other hand, since $J(\overline{\theta}_1) < J(\overline{\theta}_5)$ (see Table I), the proposed functional J chooses $\overline{\theta}_1$.

VII. Performance Evaluation and Comparison

In this section we compare the proposed control system, named *PI* in the following, with the Ohta and the OCC algorithms. The proposed control system has been tuned by using the optimal parameters found in Section VI-C, i.e. $K_{pc} = 3.5$, $K_{ic} = 10.3$, $K_{pq} = 19.2$, and $K_{iq} = 130.5$.

Before showing the results of the experimental evaluation we briefly describe the other two considered control systems, i.e. Otha and OCC.

Ohta's algorithm is a simple queue-based bang-bang controller that differentiates between two different states of the server: *normal* and *congestion*. During normal state the server forwards all the received messages. When the queue length exceeds a high watermark value (`hi_wm`) the server enters into congestion state: in this state it rejects all the requests. The normal state is entered again when the queue length becomes less than the watermark value (`lo_wm`). Since it is based on a queuing structure such as our algorithm, Ohta was implemented in a similar way in the Kamailio's `ratelimit` module. Queue buffer size was set to 1000, `lo_wm` and `hi_wm` to 400 and 800 respectively as suggested in [4].

OCC dynamically adjusts the probability f of accepting an incoming *INVITE* request based on measurements of the CPU load C to drive it to a target utilization C_T. The control law is a discrete time nonlinear controller described by the following equation:

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

where f_k is the acceptance ratio, $\phi_k = \min(C_T/C_k, \phi_{max})$. f_{min} avoids to have zero minimal acceptance ratio, whereas $\phi_{max} > 1$ is the maximum multiplicative increase factor. OCC was implemented in the `ratelimit` module without using a queuing structure, since messages are forwarded synchronously with their arrival. We employed a control interval $T_c = 1$sec, a measurement interval $T_m = 0.1$sec, $f_{min} = 0.02$, and $\phi_{max} = 5$ as suggested in [4].

Fig. 11 shows the normalized goodput as a function of the normalized offered load. In particular, the proposed control system achieves a normalized goodput equal to 0.5 when the normalized offered load is equal to 2, whereas both OCC and Ohta are overloaded. When OCC is used with a target CPU load equal to 0.9 the goodput degrades significantly for input rates greater than 1.3, due to its low responsiveness. OCC 80% better handles overload *wrt* OCC 90%, and it is able to support input rates up to 1.7. For what concerns the Ohta algorithm, Fig. 11 shows that as soon as the input rate gets greater than 1, the goodput suffers a significant step-like drop, indicating that the algorithm is not able to properly handle overload episodes.

Fig. 12 (a) shows that the proposed control system maintains the retransmissions ratio below 0.1 up to a normalized offered load equal to 3. On the other hand, OCC and Ohta are not able to handle overload since retransmissions start to increase for an offered load equal to 1. Fig. 12 (b) shows the call establishment time versus the normalized offered load. The proposed algorithm maintains a call establishment time which tracks the target value $T_q = 0.05$s for up to an offered load equal to 3, which confirms that the first control loop successfully tracks the reference signal $q_T(t)$. On the other hand, OCC and Ohta exhibit a quickly increasing average call establishment time as soon as the offered load approaches the value of 1.

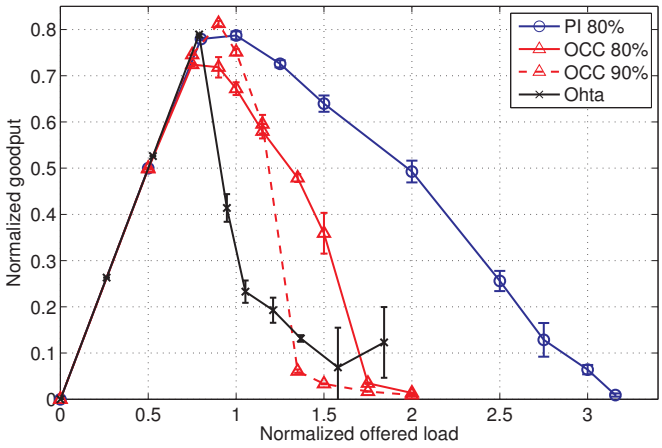
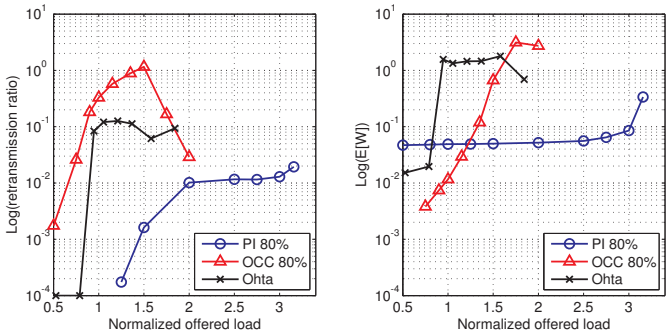


Figure 11. Goodput comparison



(a) Retransmissions ratio (b) Call establishment time

Figure 12. Retransmission ratio and call establishment time

VIII. Conclusions

We have proposed a SIP overload control system that controls both the queue length and the CPU load of a SIP server. We have implemented the proposed overload control system in Kamailio, an open source SIP proxy, and carried out a performance optimization by means of the Extremum Seeking algorithm. A comparison with the well known Occupancy (OCC) and Ohta algorithms has been carried out. Results have shown that the proposed control system significantly outperforms the OCC and Ohta algorithms, providing higher goodput along with lower retransmissions ratio and call establishment time. The proposed control system handles overload up to a maximum normalized input load equal to 3, whereas OCC supports input rates up to 1.7 and Ohta fails to handle overload condition already when the normalized offered load approaches to 1.

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterston, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," *RFC 3261, Internet Engineering Task Force*, June 2002.

[2] J. Rosenberg, "Requirements for Management of Overload in the Session Initiation Protocol," *RFC 5390*, Dec. 2008.

[3] C. Shen and H. Schulzrinne, "On tcp-based sip server overload control," in *Principles, Systems and Applications of IP Telecommunications*, pp. 71–83, ACM, 2010.

[4] V. Hilt and I. Widjaja, "Controlling Overload in Networks of SIP Servers," in *Proc. of IEEE ICNP*, pp. 83–93, Oct. 2008.

[5] S. H. Low and D. E. Lapsley, "Optimization flow control - i: basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–874, 1999.

[6] S. Mascolo, "Congestion control in high-speed communication networks using the Smith principle," *Automatica*, vol. 35, no. 12, pp. 1921–1935, 1999.

[7] S. H. Low, F. Paganini, and J. C. Doyle, "Internet congestion control," *IEEE Control Systems*, vol. 22, no. 1, pp. 28–43, 2002.

[8] R. Srikant, *The mathematics of Internet congestion control*. Springer, 2004.

[9] F. Paganini, Z. Wang, J. C. Doyle, and S. H. Low, "Congestion control for high performance, stability, and fairness in general networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 43–56, 2005.

[10] C. V. Hollot, V. Misra, D. Towsley, W. Gong, "Analysis and design of controllers for AQM routers supporting TCP flows," *IEEE Transactions on Automatic Control*, vol. 47, no. 6, pp. 945–959, 2002.

[11] S. Liu, T. Basar, and R. Srikant, "Exponential-red: a stabilizing aqm scheme for low-and high-speed tcp protocols," *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 1068–1081, 2005.

[12] W. Michiels, D. Melchor-Aguilar, and S.-I. Niculescu, "Stability analysis of some classes of tcp/aqm networks," *International Journal of Control*, vol. 79, no. 9, pp. 1136–1144, 2006.

[13] S. Manfredi, F. Oliviero, and S. P. Romano, "A distributed control law for load balancing in content delivery networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 1, pp. 55–68, 2013.

[14] L. De Cicco and S. Mascolo, "A mathematical model of the skype voip congestion control algorithm," *IEEE Transactions on Automatic Control*, vol. 55, no. 3, pp. 790–795, 2010.

[15] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo, "Elastic: a client-side controller for dynamic adaptive streaming over http (dash)," in *Proc. of Packet Video Workshop*, pp. 1–8, 2013.

[16] M. Kihl, A. Robertsson, M. Andersson, and B. Wittenmark, "Control-theoretic analysis of admission control mechanisms for web server systems," *The World Wide Web Journal, Springer*, vol. 11, pp. 93–116, Aug. 2007.

[17] A. Pietrabissa, "An alternative lp formulation of the admission control problem in multiclass networks," *IEEE Transactions on Automatic Control*, vol. 53, no. 3, pp. 839–845, 2008.

[18] N. J. Killingsworth and M. Krstic, "PID tuning using extremum seeking: online, model-free performance optimization," *IEEE Control Systems Magazine*, vol. 26, pp. 70–79, Feb. 2006.

[19] V. Hilt, E. Noel, C. Shen, and A. Abdelal, "Design Considerations for Session Initiation Protocol (SIP) Overload Control," *RFC 6357, Internet Engineering Task Force*, Aug. 2011.

[20] Y. Hong, C. Huang, and J. Yan, "A comparative study of sip overload control algorithms," *Network and Traffic Engineering in Emerging Distributed Computing Applications*, 2012.

[21] M. Ohta, "Overload control in a sip signaling network," *Proc. of World Academy of Science, Engineering and Technology*, pp. 205–210, 2006.

[22] C. Shen, H. Schulzrinne, and E. Nahum, "Session initiation protocol (sip) server overload control: Design and evaluation," in *Proc. of IPTCOMM*, pp. 149–173, 2008.

[23] Y. Hong, C. Huang, and J. Yan, "Applying control theoretic approach to mitigate sip overload," *Telecommunication Systems*, vol. 54, no. 4, pp. 387–404, 2013.

[24] R. Garroppo, S. Giordano, S. Niccolini, and S. Spagna, "A prediction-based overload control algorithm for sip servers," *IEEE Transactions on Network and Service Management*, vol. 8, no. 1, pp. 39–51, 2011.

[25] Y. Wang, "Sip overload control: a backpressure-based approach," in *Proc. of ACM SIGCOMM 2010, poster session*, pp. 399–400, 2010.

[26] S. V. Azhari, M. Homayouni, H. Nemati, J. Enayatizadeh, and A. Akbari, "Overload control in sip networks using no explicit feedback: A window based approach," *Computer Communications*, vol. 35, no. 12, pp. 1472 – 1483, 2012.

[27] J. Liao, J. Wang, T. Li, J. Wang, J. Wang, and X. Zhu, "A distributed end-to-end overload control mechanism for networks of sip servers," *Computer Networks*, vol. 56, no. 12, pp. 2847 – 2868, 2012.

[28] V. Gurbani, V. Hilt, and V. Schulzrinne, " Session Initiation Protocol (SIP) Overload Control," *IETF, Internet-Draft, Work in Progress*, 2013.

[29] Y. Hong, C. Huang, and J. Yan, "Mitigating sip overload using a control-theoretic approach," in *Proc. of IEEE GLOBECOM 2010*, pp. 1–5, 2010.

[30] H. Jiang, A. Iyengar, E. Nahum, W. Segmuller, A. N. Tantawi, and C. P. Wright, "Design, implementation, and performance of a load balancer for sip server clusters," *IEEE/ACM Transactions on Networking*, vol. 20, no. 4, pp. 1190–1202, 2012.

[31] M. Krstic and H. Wang, "Stability of extremum seeking feedback for general nonlinear dynamic systems," *Automatica*, vol. 36, no. 4, pp. 595 – 601, 2000.

[32] D. Popovic, M. Jankovic, S. Magner, and A. R. Teel,, "Extremum seeking methods for optimization of variable cam timing engine operation," *IEEE Transaction on Control Systems Technology*, vol. 14, no. 3, pp. 398–407, 2006.

[33] D. Carnevale, A. Astolfi, C. Centioli, S. Podda, V. Vitale, and L. Zaccarian, "A new extremum seeking technique and its application to maximize rf heating on ftu," *Fusion engineering and design*, vol. 84, no. 2, pp. 554–558, 2009.

[34] K. H. Stankovic, M. Johansson and D. M. Stipanovic, "Distributed seeking of nash equilibria in mobile sensor networks," in *Proc. of IEEE Conference on Decision and Control*, pp. 5598–5603, 2010.

[35] P. Dower, P. Farrell and D. Nesic, "Extremum seeking control of cascaded raman optical amplifiers," *IEEE Transaction on Control Systems Technology*, vol. 16, no. 3, pp. 396–407, 2008.

[36] Y. Tan, Y. Li, and I. Mareels, "Extremum Seeking for Constrained Inputs," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2405–2410, 2013.

Luca De Cicco received a computer science engineering degree cum laude in 2003 and a Ph.D. in July 2008, both from the Technical University of Bari, Italy. He is currently a researcher at Politecnico di Bari. He has held visiting positions at the University of New Mexico (Albuquerque, USA), Ecole Superieure d'Electricité (Paris, France), the Laboratory of Information, Networking and Communication Sciences - LINCS (Paris, France) in 2007, 2012 and 2013. He is co-author of more than 30 papers published in international journals, books or conferences. His main interests focus on the modelling and design of congestion control algorithms for multimedia transport, adaptive video streaming, SIP overload control.

Giuseppe Cofano received the Telecommunications Engineering degree (cum laude) from Politecnico di Bari, Bari, Italy, in 2012. Since 2013 he is a Ph.D. student at Politecnico di Bari. His main interests focus on the modeling and design of control algorithms for multimedia transport and adaptive video streaming.

Saverio Mascolo received the Laurea degree, cum laude, in electronics engineering in 1991 and the Ph.D in 1994, both from the Technical University of Bari, Italy. Since 2001, he has been associate professor in Automatic Control at Politecnico di Bari where is now full professor since 2012. He has been post-doc in 1995 and visiting researcher in 1999 at the University of California Los Angeles, and visiting consultant at University of Uppsala, Sweden from 2002 to 2006. He has authored or co-authored more than 100 papers in international journals, books or conferences. He is author and assignee of 4 US patents and 3 Italian patents. His current research interests focus on the Future Internet, in particular in the topic of real-time communication over the Web. He has worked on congestion control in data networks (TCP and ATM), end-to-end bandwidth estimate, modelling and control. He is senior member of IEEE and ACM. He is Associate Editor of the IEEE Transactions on Automatic Control journal and of Computer Networks Journal, Elsevier.