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 framework allowing two or more parties to establish, alter, and terminate various types of media sessions. An open issue is the proper handling of overload situations that in SIP servers occur when the incoming flow of requests overcomes the processing resources. Due to overload, call establishment times increase and retransmissions are triggered causing an uncontrolled increase of the total input rate. In this paper we present a local overload control system, made of two PI controllers, aiming at regulating both the queue length and the CPU load of the SIP proxy to provide high goodput and low call establishment delays. The proposed control system has been implemented in Kamailio (OpenSER) and the controllers parameters have been tuned by employing the Extremum Seeking algorithm to minimize a cost function. A performance evaluation and comparison with Ohta and Occupancy local overload controllers has shown the following main results: 1) the proposed controller is able to counteract overload situations providing a goodput which is close to the optimal while maintaining low call establishment delays and retransmission ratios; 2) the proposed control system significantly outperforms Ohta and OCC both in terms of goodput, call establishment delays and retransmission ratios.**

I. Introduction

The Session Initiation Protocol (SIP) [1] is a signaling framework that allows two or more parties to establish, alter, and terminate various types of media sessions. Today, SIP is the main signaling protocol for multimedia sessions such as Voice over IP, instant messaging, and video conferencing.

A key open issue is the proper handling of overload situations in SIP networks. Overload occurs when the incoming request rate to a SIP server exceeds its processing capacity. Possible causes for overload include poor capacity planning, component failures, avalanche restart, flash crowds and denial of service attacks [2]. It has been shown that overload episodes are exacerbated when SIP is used with the UDP due to the retransmission mechanism employed by SIP to cope with packet losses. In fact, during overload episodes, retransmissions occur and, as a consequence, the total incoming load increases, potentially leading the entire SIP network to collapse [2], [3]. The 503 response code *"Service Unavailable"* can be sent by the overloaded SIP servers to the user agent (UA) to reject messages [1], thus preventing messages retransmissions. Unfortunately, it is well-known that this mechanism is not able to effectively avoid overload [2], [3].

The authors are with the Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Orabona 4, Bari, Italy. emails: l.decicco@poliba.it, g.cofano@poliba.it, mascolo@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).

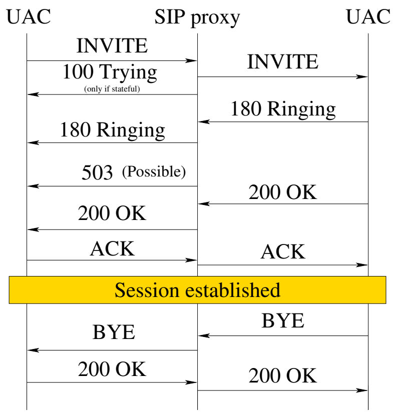


Fig. 1: Time sequence graph of a SIP call

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 based on the control architecture (see [3] for a comparison): *local overload control*, *hop-by-hop*, and *end-to-end*.

In this paper we focus on local overload control which does not require any modification to the SIP protocol. The main contributions of this 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 control loops with two PI controllers; 3) a functional is proposed as a performance metric for local overload control algorithms; 4) based on the proposed performance metric, the parameters of the controllers have been tuned by means of the Extremum Seeking (ES) algorithm [4]; 5) a performance evaluation and comparison with two local overload controllers is performed.

II. SIP Overview

SIP is a client-server message-based protocol for managing media 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 registration servers, for UA registration.

Fig. 1 shows the time sequence graph of the establishment of a SIP call session, which is originated by the *INVITE* message.

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 Unavailabl*e" in order to avoid the start of retransmissions [2]. In this case the incoming message is said to be *rejected.*

III. Related Work

Several local overload control algorithms have been proposed in the literature. The first local overload control mechanism specifically designed for SIP has been proposed by Ohta in [5]: 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) [6], [3]. The algorithm rejects a fraction of *INVITE*s according to a simple control law in order to drive the CPU load to a target utilization.

Distributed overload control algorithms have attracted a great deal of attention due to the promise of providing better performance. The first paper exploring the hop-by-hop approach has been [6]. Authors have defined two overload categories: (i) server to server overload and (ii) client to server overload. In [6] only the first case has been considered and three hop-by-hop window-based feedback algorithms have been proposed. In [7] a feedback-based algorithm has been proposed to regulate retransmission ratios during overload. In [8] 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.

Finally, it is worth noting that performance evaluation of the proposed controllers in [3], [5], [6], [7], [8] has been carried out by using discrete events simulators and an experimental evaluation is not provided.

IV. The Proposed Control System

In this Section we propose a local overload control algorithm which does not require any modification in SIP protocol and can be rapidly deployed in SIP proxies. Moreover, local control should be always implemented in a large SIP network in order to protect the servers in the case the distributed controllers do not work properly.

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 proxy [9]. Moreover, we make the modelling assumption, which is experimentally validated, that the cost for forwarding one *INVITE* message is unitary, while rejecting one *INVITE* has a cost $\frac{1}{\beta}$ (β>1) which is a fraction of the forward cost. 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 proxy 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 incoming *INVITE* rate $r(t)$ to be rejected. The CPU load can be modelled as follows:

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

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[2].

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 maximum CPU load to be assigned to 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:

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

Proof: The proof is omitted due to space limitations. ■

Remark 1: The normal behaviour of a SIP proxy can be extended even when the normalized incoming *INVITEs* rate is greater than 1, that is the limit at which an uncontrolled SIP proxy gets overloaded.

B. The design of the proposed control system

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. 2 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 $C_T < 1$.

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

[2]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.

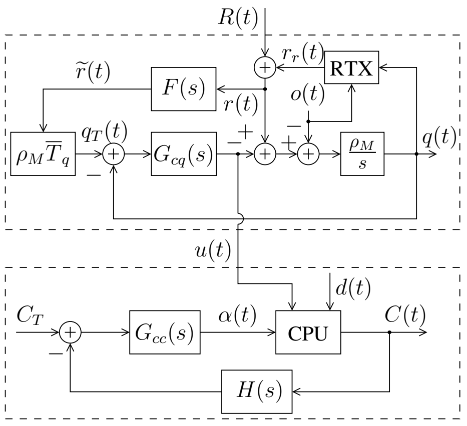


Fig. 2: The proposed control system

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. The queue is assumed to be a FIFO drop-tail buffer, with a maximum length q_M, 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 that is the sum of $R(t)$, the rate of new *INVITE*s, and $r_r(t)$, the retransmission rate. The overflow rate $o(t)$ can be modelled as follows [10], [11]:

$$o(t) = \begin{cases} r(t) - u(t) & q(t) = q_M, r(t) > u(t) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

which is the rate of dropped messages when the queue is full and the input rate exceeds the output rate.

Regarding the retransmission rate $r_r(t)$, which in Fig. 2 is the output of the block "RTX", a fluid model is still not available in the literature. In fact, retransmissions occur due to the expiration of a series of timeouts of variable duration which can be caused by either large queuing times $T_q(t)$ or the queue overflow rate $o(t)$. The modelling of $r_r(t)$ is outside the scope of this work.

Even though, in principle, it would be possible to explicitly control the queuing delay $T_q(t)$, we take a different approach which makes the controller simpler to be implemented. In fact, the model of the $T_q(t)$ is non-linear and it is often approximated in the literature as $T_q(t) = q(t)/(\rho_M u(t))$, with $u(t) \neq 0$ [12]. We are able to indirectly control $T_q(t)$ by using a set-point $q_T(t) = \overline{T}_q \rho_M \tilde{r}(t)$ where $\overline{T}_q$ is the desired queuing time and $\widetilde{r}(t)$ 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, to avoid retransmissions. We employ a first order low-pass filter (LPF), shown in Fig. 2 by the transfer function $F(s) = 1/(1 + \tau s)$ with $\tau = 0.1$s, to filter out 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}):

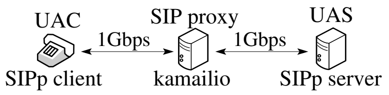


Fig. 3: The testbed employed for the experimental evaluation

$$G_{cq}(s) = \frac{U(s)}{E_q(s)} = \frac{1}{\rho_M}(K_{pq} + \frac{K_{iq}}{s}) \quad (4)$$

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 non dimensional.

2) The CPU controller: The second controller, i.e. $G_{cc}(s)$, computes the fraction of messages to reject $\alpha(t)$, using 503 messages, to steer the CPU load $C(t)$ to the desired value C_T. It is worth noting that the queue draining rate $u(t)$ tracks $r(t)$ at steady state and can be considered as a disturbance acting on such 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 desired CPU load to be allocated to the SIP proxy and $\widetilde{C}(t)$ is the low pass filtered CPU load. We have used a first order filter $H(s) = 1/(1 + \tau_c s)$ with $\tau_c = 0.1$ s. Again, we employ a proportional-integrative controller, whose equation is given by

$$\alpha(t) = K_{pc}e_c(t) + K_{ic}\int_0^t e_c(\tau)d\tau. \quad (5)$$

By taking the derivative of (5), 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:

$$\begin{cases} \dot{\alpha}(t) = & \frac{K_{pc}}{\tau_c}[u(t) \cdot (1 - \gamma\alpha(t)) + d(t) - \widetilde{C}(t)] + \\ & + K_{ic}(\widetilde{C}(t) - C_T) \\ \dot{\widetilde{C}}(t) = & \frac{1}{\tau_c}u(t)(1 - \gamma\alpha(t)) + \frac{1}{\tau_c}d(t) - \frac{1}{\tau_c}\widetilde{C}(t) \end{cases} \quad (6)$$

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

Proposition 2: Considered the equilibrium inputs $(C_T, \overline{u}, \overline{d})$ in the positive orthant of $\mathbb{R}^3$, the system (6) has a unique equilibrium point:

$$\overline{\alpha} = \frac{\beta - 1}{\beta}(1 - \frac{C_T}{\overline{r}} + \frac{\overline{d}}{\overline{r}}); \; \overline{C} = C_T \quad (7)$$

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

Proof: The proof is carried out by using the indirect Lyapunov method and it is omitted due to space constraints. ■

V. Implementation Details

Fig. 3 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[3] to generate a configurable *INVITE* rate. SIPp has been also used to emulate the upstream SIP server. The SIPp client and server ran over an Intel

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

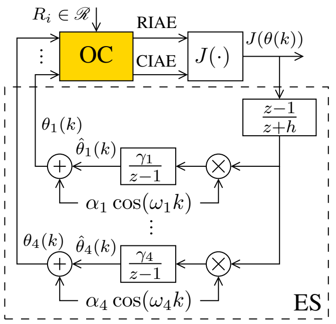


Fig. 4: Block diagram of the discrete extremum seeking algorithm

Pentium IV 3.60 GHz with 2 GB of RAM. The modified Kamailio SIP server, configured in the transaction-stateless mode with no authentication, ran as proxy server over an Intel Pentium III 1 GHz with 756 MB of RAM.

Kamailio is a widely employed open source SIP proxy server. Its architecture is made of a *core,* providing basic SIP server functionalities, and several pluggable modules which extend the core. We have implemented the controller proposed in Section IV, Ohta [5], and OCC [3] overload controllers in the Kamailio module named `ratelimit` in order to perform an experimental performance evaluation.

Both the PI controllers of the proposed controllers have been discretized and provided with an anti-wind up scheme to cope with saturation of the actuation variables. A timer function ensures a sampling time of $T_c = 10$ms, while another timer samples the CPU load every $T_m = 10$ms. Finally, the maximum queue length q_M has been set equal to 800 *INVITE*s.

VI. Controller Optimization

The proposed control system is made by two PI controllers, resulting in four parameters to tune. 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 algorithm Extremum Seeking (ES) [13], [4] to tune the controllers without requiring the knowledge of the mathematical model of the system. ES has been applied successfully in different applicative fields [14], [15]. 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. 4 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 [4] and shown in Fig. 4, which can be described by the following equations:

$$\zeta(k) = -h\zeta(k-1) + J(\theta(k-1)) \tag{8}$$

$$\hat{\theta}_i(k+1) = \hat{\theta}_i(k) - \gamma_i\alpha_i\cos(\omega_i k)(J(\theta(k)) - (1+h)\zeta(k)) \tag{9}$$

$$\theta_i(k+1) = \hat{\theta}_i(k+1) + \alpha_i\cos(\omega_i\cdot(k+1)) \tag{10}$$

Eq. (10) gives the output of the ES algorithm and, evaluated for $i = 1,2,3,4$, computes the new parameters vector.

B. The cost function $J(\theta)$

The cost function $J(\theta)$ is measured at the conclusion of a series of step-response experiments, whose duration has been set to 120s, with input rates $R(t) = R_i \cdot 1(t)$ with $R_i \in \mathscr{R} = \{1.58, 2.10, 2.37, 2.63\}$, where $1(t)$ is the step function. For each $R_i \in \mathscr{R}$ we have repeated every experiment 6 times and considered the best $m = 4$ values to rule out possible outliers due to experimental testbed issues.

We employ a functional that is the linear combination of the average values of two integral absolute errors over the m experiments. In particular, the functional $J(\theta)$ is:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{R\in\mathscr{R}}(\eta\text{RIAE}_i(\theta,R) + \xi\text{CIAE}_i(\theta,R)). \tag{11}$$

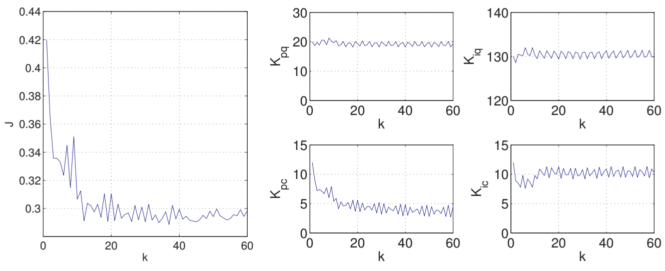
$$\text{RIAE}_i(\theta,R) = \frac{1}{t_f}\int_0^{t_f} |\tilde{r}(t,\theta) - R|dt \tag{12}$$

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

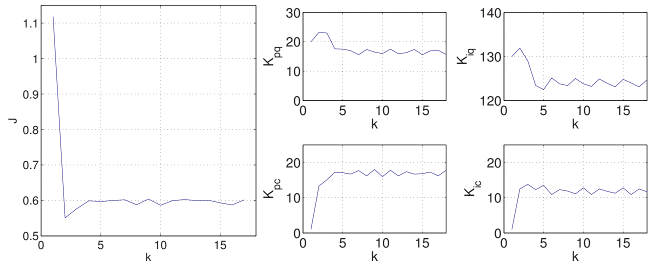
$$\text{CIAE}_i(\theta,R) = \frac{1}{(1-C_T)t_f}\int_0^{t_f} |\widetilde{C}(t,\theta) - C_T|dt \tag{13}$$

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.

Even though a functional depending on the measured goodput would appear as a natural choice for the performance index of the considered system, we argue that such a functional would not work as a proper indicator for overload

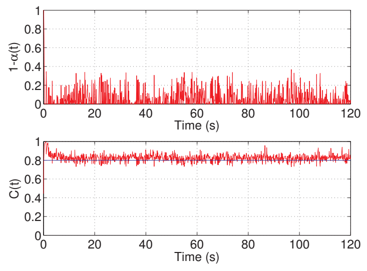


(a) $\theta(0) = [12, 12, 20, 130]$

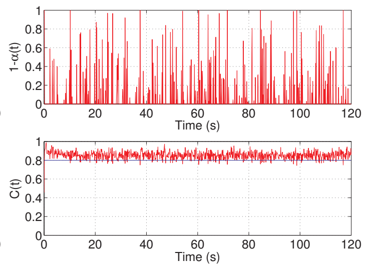


(b) $\theta(0) = [1, 1, 20, 130]$

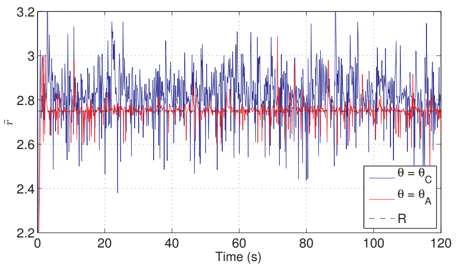
Fig. 5: Evolution of $J(\theta(k))$ and $\theta(k)$



(a) CPU load and accept ratio ($\theta = \overline{\theta}_A$)



(b) CPU load and accept ratio ($\theta = \overline{\theta}_C$)



(c) Arrival rate comparison

Fig. 6: Performance comparison for $\theta = \overline{\theta}_A$ and $\theta = \overline{\theta}_C$ in the case of $R = 2.75$

episodes. In fact, a goodput higher than the ideal maximum given by (2) could be measured, meaning that the SIP proxy is using a CPU load higher than C_T. This situation should be avoided to ensure that the specifics are met.

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.

We have performed a series of ES runs employing different initial conditions to investigate the presence of multiple local minima. The first run has been started from $\theta(0) = [12, 12, 20, 130]$. $K_{pq}(0) = 20$ and $K_{iq}(0) = 130$ have been chosen to ensure that the first loop has a damping factor of 0.7 and a 2% settling time equal to approximately 0.1s. Fig. 5 (a) shows the evolution of both $J(\theta(k))$ and the parameters vector $\theta(k)$. $J(\theta(k))$ converges into a neighborhood of $\overline{\theta}_A = [3.5, 10.3, 19.2, 130.5]$ corresponding to $J(\overline{\theta}_A) = 0.29$.

It can be noticed that $J(\theta(k))$ is not very sensitive to the variations of K_{pq} and K_{iq}. In fact, these parameters negatively affect the performance of the system only if the settling time of the first loop is high and, as a consequence, retransmissions are triggered. The figure clearly suggests that $J(\theta(k))$ 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 $K_{ic} = 10.3$. A second run, started from $\theta(0) = [2, 2, 20, 130]$, has confirmed this result.

A third run, started from $\theta(0) = [1, 1, 20, 130]$, has led to a different equilibrium as shown in Fig. 5 (b). The parameters vector converges to the neighborhood of $\overline{\theta}_C = [17, 12, 16.5, 124]$, where $J(\overline{\theta}_C) = 0.6$, which is much higher *wrt* $J(\overline{\theta}_A)$. To get a further insight, let us consider Fig. 6 (a) and (b) that show the CPU load $\tilde{C}(t)$ and the accept ratio $1 - \alpha(t)$ respectively for $\overline{\theta}_A$ and $\overline{\theta}_C$ when the incoming rate R is equal to 2.75. When $\theta = \overline{\theta}_C$, the reject ratio is saturated, i.e. $\alpha = 1$, and a finite CPU load tracking error is present due to the anti-windup scheme. For $\theta = \overline{\theta}_A$, $\alpha(t)$ is not saturated and $C(t)$ tracks C_T with zero steady-state error. Fig. 6 (c) compares the measured retransmission rates when θ is either equal to $\overline{\theta}_A$ or $\overline{\theta}_C$. For $\theta = \overline{\theta}_C$ a moderate retransmission rate is exhibited, whereas for $\theta = \overline{\theta}_A$ retransmissions are negligible. Thus it has been explained the reason why a small K_{pc} provides better performance. However, if K_{pc} is made too small the settling time increases and the retransmissions generated during the transient become significant. This can be verified by considering Fig. 5 (b) which shows that, for $K_{pc} = 1$ and $K_{ic} = 1$, the corresponding value of J is very high ($J(\theta) = 1.1$). In other words, the functional $J(\theta)$ is able to make a trade-off between responsiveness in tracking the target C_T and control effort preventing actuator saturation.

VII. Performance Comparison

In this section we compare the proposed control system, named PI in the following, with Ohta and OCC algorithms. Ohta's algorithm is a simple bang-bang controller that decides to either reject or accept a new SIP session based on the queue length [5]. OCC dynamically adjusts the probability f of accepting an incoming *INVITE* request based on measurements of the CPU load to drive it to a target

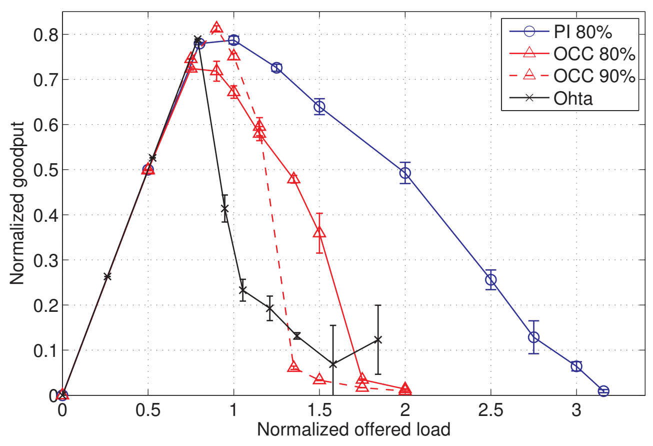
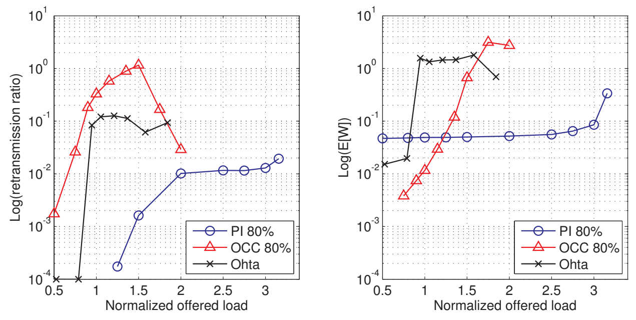


Fig. 7: Goodput comparison



(a) Retransmissions ratio (b) Call establishment time

Fig. 8: Retransmission ratio and call establishment time

utilization [6], [3]. For both of them we have employed the parameters suggested in [3]. For what concerns the proposed controller we have employed $K_{pc} = 3.5$, $K_{ic} = 10.3$, $K_{pq} = 19.2$, and $K_{iq} = 130.5$ which have been found in Section VI-C.

Fig. 7, Fig. 8 (a), and Fig. 8 (b) show respectively the goodput curves, the retransmission ratio, and the call establishment time W for each of the considered SIP overload controllers. Fig. 7 shows that the proposed control system achieves significantly better performance in terms of goodput, retransmissions ratio, and call establishment time. In particular, the proposed control system achieves a normalized goodput equal to 0.5 when $R = 2$, whereas OCC and Ohta are overloaded. When OCC is used with CPU load target 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. 7 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. 8 (a) shows that the proposed control system maintains the retransmissions ratio below 0.1 for a normalized rate equal to 3, i.e. it prevents the uncontrolled increase of retransmissions which is a symptom of an overload situation. On the other hand, OCC and Ohta are not able to handle overload correctly and retransmissions cannot be controlled for large values of R. Let us consider the Fig. 8 (b) which shows the call establishment time. The proposed algorithm maintains a call establishment time which matches the target value $T_q = 0.05$s for every input load: this confirms that the first control loop tracks the reference signal $q_T(t)$. On the other hand, OCC and Ohta exhibit very high average call establishment time.

VIII. Conclusions

We have proposed a SIP overload control algorithm controlling both the queue length and the CPU load of a SIP proxy. 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 algorithm Extremum Seeking and a comparison with OCC and Ohta, two local overload control algorithms. The results have shown that the proposed control system significantly outperforms OCC and Ohta providing higher goodput and exhibiting low retransmissions ratio and call establishment time. The proposed control system handles overload up to a maximum normalized input load equal to 3, OCC supports input rates lower than 1.7, whereas Ohta fails to properly handle overload.

References

[1] J. Rosenberg, et al, “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, Internet Engineering Task Force*, Dec. 2008.
[3] V. Hilt and I. Widjaja, “Controlling Overload in Networks of SIP Servers,” in *Proc. IEEE ICNP*, pp. 83–93, Oct. 2008.
[4] 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.
[5] M. Ohta, “Overload control in a sip signaling network,” *Proc. of World Academy of Science, Engineering and Technology*, pp. 205–210, 2006.
[6] C. Shen, et al, “Session initiation protocol (sip) server overload control: Design and evaluation,” in *Proc. IPTCOMM ’08*, pp. 149–173, 2008.
[7] Y. Hong, et al, “Mitigating sip overload using a control-theoretic approach,” in *Proc. IEEE GLOBECOM 2010*, pp. 1–5, 2010.
[8] R. Garroppo, et al, “A prediction-based overload control algorithm for sip servers,” *IEEE Transactions on Network and Service Management*, vol. 8, no. 1, pp. 39–51, 2011.
[9] H. Jiang, et al, “Load Balancing for SIP Server Clusters,” in *Proc. IEEE INFOCOM*, pp. 2286 –2294, Apr. 2009.
[10] S. Mascolo, “Congestion control in high-speed communication networks using the Smith principle,” *Automatica*, vol. 35, no. 12, pp. 1921–1935, 1999.
[11] L. De Cicco and S. Mascolo, “A mathematical model of the skype voip congestion control algorithm,” *IEEE Transactions on Automatic Control*, vol. 55, pp. 790 –795, march 2010.
[12] C.V. Hollot, et al, “Analysis and design of controllers for AQM routers supporting TCP flows,” *IEEE Transactions on Automatic Control*, vol. 47, no. 6, pp. 945–959, 2002.
[13] M. Krstic and H.-H. Wang, “Stability of extremum seeking feedback for general nonlinear dynamic systems,” *Automatica*, vol. 36, no. 4, pp. 595 – 601, 2000.
[14] D. Popovic, et al, “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.
[15] D. Carnevale, et al, “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.