Local SIP Overload Control

Luca De Cicco, Giuseppe Cofano, and Saverio Mascolo

Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari
Via Orabona n.4, 70125 Bari, Italy
{l.decicco,g.cofano,mascolo}@poliba.it

Abstract. The Session Initiation Protocol (SIP) is a signaling framework that allows two or more parties to establish, alter, and terminate various types of media sessions. The mechanism employed by the standard SIP is not effective in handling overload situations that occur when the incoming flow of requests overcomes the processing resources of the server. In this paper we present a local overload control system based on feedback control theory. The algorithm has been implemented in Kamailio (OpenSER) and a performance comparison with Ohta and Occupancy (OCC) overload control algorithms has been performed. The proposed control system efficiently counteracts overload situations providing a goodput which is close to the optimal while maintaining low call establishment delays and retransmission ratios. On the other hand, Ohta and OCC algorithms provide higher call establishment delays and retransmission ratio and lower goodputs.

1 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. Nowadays, SIP is the main signaling protocol for multimedia sessions such as Voice over IP, instant messaging and video conferencing in the Internet and IP telephony.

A key open issue of SIP is the proper handling of overload situations. Overload typically 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 the overload gets worse when SIP is used with UDP due to the presence of a retransmissions mechanism which is employed to cope with packet losses. During overload episodes retransmissions occur and the total incoming load increases, potentially leading the entire network to collapse [2][3]. The 503 response code *"Service Unavailable"* is sent by the overloaded SIP servers to the user agent (UA) to reject messages, thus preventing messages retransmissions. Unfortunately, it is well-known that this mechanism is not able to effectively mitigate overload situations [2][3].

With the purpose of addressing this issue, researchers have recently proposed several overload control algorithms and the SIP Overload Control IETF working group has been established . Overload control algorithms can be designed following three different approaches (see [3] for a comparison): 1) *local overload control*: the algorithm is executed locally on the SIP server and it does not receive any feedback from other SIP servers; 2) *hop-by-hop*: the control loop is closed between two connected SIP servers; 3) *end-to-end*: the overload 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.

In this paper we propose a local SIP overload controller by employing a feedback control approach: the controller drives the queue length and the CPU utilization to opportune targets to both avoid overload and ensure a target response time.

The rest of the paper is organized as follows: in Section 2 a brief overview of the SIP protocol is given along with the state of the art of overload control methods proposed in the literature; Section 3 presents the proposed feedback overload control algorithm; Section 4 shows the results of an experimental evaluation; Section 5 draws the conclusion of the paper.

2 SIP Overview and Related Work

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.

Figure 1 shows the time sequence diagram of the establishment of a SIP call session. An originating UA sends an *INVITE* request to a terminating UA via a proxy server. The proxy server returns a provisional *100 Trying* response to confirm. In the case the proxy is stateful, the terminating UA returns a *180*

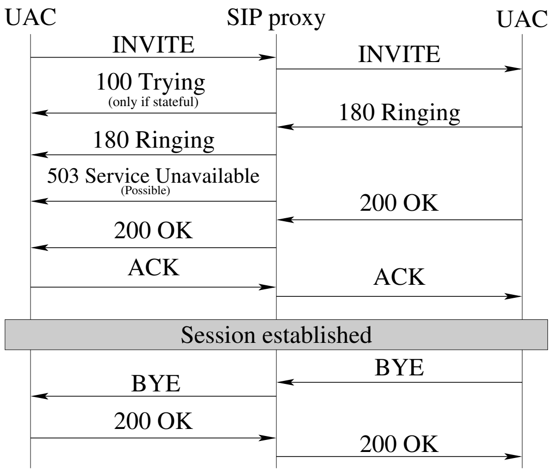


Fig. 1. Time sequence diagram of the establishment of a SIP call

ringing response after confirming that the parameters are appropriate. It also sends a *200 OK* message to answer the call. At that point, after receiving the *200 OK* message, the originating UA sends an *ACK* response to the terminating UA and the call is established. Finally, the *BYE* request is sent to close the session.

In the case the SIP messages are sent over a UDP socket, SIP employs a retransmission mechanism to cope with packet losses. In particular, SIP uses the *Timer A* to trigger an *INVITE* retransmission every time it expires [1]. The first retransmission occurs when the initial default value of $T_1 = 500$ms is reached. After each retransmission *Timer A* is doubled. Retransmissions are stopped when a provisional response is received or when the timeout value exceeds 32s. In case of overload, SIP servers are mandated to send a *503 "Service Unavailable*" that avoids the start of retransmissions. The incoming message is said to be rejected.

Several local overload control algorithms have been proposed so far. The first local overload control mechanism specifically designed for SIP was proposed by Ohta [4]: 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), in the version described in [3]. It is based on CPU utilization: when it becomes higher than the desired target value, the algorithm reacts by rejecting a fraction of *INVITE*s according to a simple control law. Authors proved that a distributed version of the same control mechanism, though more complex, can achieve better performance.

Distributed overload control algorithms have attracted a great deal of attention due to the promise of achieving better performance. The first paper exploring the hop-by-hop approach was [5]. Authors group overload into two categories: (i) server to server overload and (ii) client to server overload. In [5] only the first case was considered and three hop-by-hop window-based feedback algorithms were proposed. The idea is that the sender server employs a feedback information which is sent by downstream servers to dynamically set the transmission window size to counteract overload. The window adjustment is made according to three different policies. The three algorithms were compared to two rate-based algorithms, showing better results. A hop-by-hop strategy is followed in [6], where several algorithms are developed. Metrics such as queue delay, CPU utilization and fraction of successful calls are employed to throttle the traffic in the upstream servers. However, techniques in [5] require the communication of a feedback information between servers. A different approach is proposed in [7] which employs a feedback-based approach to regulate retransmissions ratio when overload episodes occur. Authors distinguish between redundant and non-redundant retransmitted messages, i.e. due to overload delay or due to message loss recovery: a PI controller is employed at the upstream server to regulate the retransmissions rate in order to track the desired value of redundant messages rate, thus preventing overload over the downstream server without requiring explicit feedback. In [8] authors proposed an overload mechanism combining local and remote control, the former based on the appropriate queuing structure and buffer management of the SIP proxy, the latter based on a prediction technique in the remote control loop according to a NLMS algorithm.

An end-to-end overload control proposal was made in [9] which does not require any modification of the SIP protocol. The author adapted to the SIP networks context an algorithm employed for multi-hop radio networks making use of a backpressure-based technique. Finally, it is worth to mention that the performance of the proposed controllers in [4,5,3,7,9] were evaluated by means of discrete events simulators, and an experimental evaluation is not provided.

3 The Proposed Control System

In this Section we propose a local overload control algorithm based on a feedback control approach. We adopt a local control approach since it does not require any modification in SIP protocol and it 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.

Without loss of generality, in this paper we focus only on *INVITE* transactions since they are the most CPU-expensive messages to be handled by a SIP proxy [10]. Furthermore, 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 $\frac{1}{\beta}$ ($\beta>1$) which is a fraction of the forward cost. Let $\rho(t)$ denote the incoming load of *INVITE* messages measured in cps (calls per second) 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 $r(t) = \rho(t)/\rho_M$. 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 model 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:

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

where the first term is the load due to the accepted rate, the second is due to rejected rate, and the third models the CPU load due to other processes in execution.

The overall control system is made of two control loops aiming at controlling both the queue length $q(t)$ of incoming *INVITE* messages and the CPU utilization $C(t)$.

We start by assuming that the SIP proxy server is able to store incoming messages in a queue that will be drained by an asynchronous worker thread. The amount of messages drained and the reject ratio are the output of the two controllers. Figure 2 depicts the proposed control architecture consisting of two

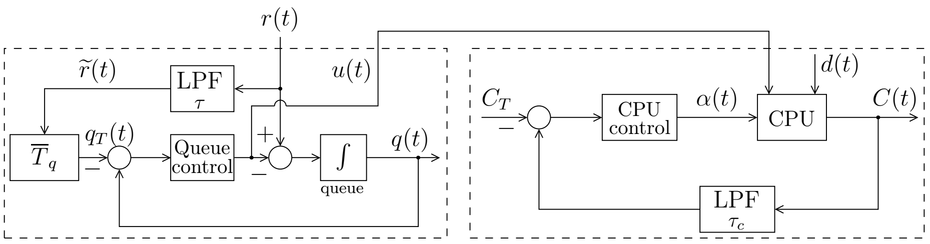


Fig. 2. The proposed control architecture

feedback loops: the goal of the first controller, depicted in the box at the left in Figure 2, is to steer the queue level $q(t)$ to a target $q_T(t)$; the other aims at steering the CPU load $C(t)$ to the desired target C_T.

Let us now focus on the first controller: the goal of this control loop is to compute the queue draining rate $u(t)$, i.e. the rate of *INVITE*s to be processed by the CPU, so that the queuing time of incoming *INVITE* messages is lower than the first retransmission timeout $T_1 = 500$ms. The queue is modelled by the integrator which is drained at the rate $u(t)$ decided by the controller and filled at the incoming rate $r(t)$.

It is possible to steer the queuing delay $T_q(t)$ to a set-point $\overline{T}_q$, the desired queuing delay, at steady state by considering a variable set-point for the queue control loop. We are able to indirectly control $T_q(t)$ by using a set-point $q_T(t) = \overline{T}_q\tilde{r}(t)$ where $\tilde{r}(t)$ is a low-pass filtered version of the measured instantaneous *INVITE*s incoming rate $\tilde{r}(t)$. 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) with a time constant $\tau = 0.4$ s to filter out the high frequency components of the incoming rate $r(t)$.

We employ a proportional-integrative controller so that $q(t)$ can track $q_T(t)$ with zero steady state error [11]:

$$u(t) = K_{pq}e_q(t) + K_{iq}\int_0^t e_q(\tau)d\tau \tag{2}$$

where $e_q(t) = q_T(t) - q(t)$ is the error and K_{pq} and K_{iq} denote the proportional and integral gains of the controller respectively.

The second control loop computes the fraction of messages to reject $\alpha(t)$, using 503 messages, in order to steer the CPU usage $C(t)$ to the desired value C_T. The queue draining rate $u(t)$ can be considered as a disturbance acting on such control loop.

Let us now focus on the proposed controller: it computes the reject ratio $\alpha(t)$ based on the error $e_c(t) = \tilde{C}(t) - C_T$, where C_T is the set-point and $\tilde{C}(t)$ is the CPU load $C(t)$ passed through a first order low-pass filter with time constant $\tau_c = 0.1$ s, to filter out the high frequency components of $C(t)$. Again, we employ a proportional-integrative controller, whose equation is given by $\alpha(t) = K_{pc}e_c(t) + K_{ic}\int_0^t e_c(\tau)d\tau$.

4 Performance Evaluation

The proposed overload control system has been implemented in a module of the open source Kamailio SIP proxy[1] with the purpose of carrying out the controller optimization and a comparison with Ohta and OCC overload controllers. This section is organized as follows: Section 4.1describes the experimental testbed employed in the performance evaluation; in Section 4.2 the implementation details of the considered overload controllers are provided; in Section4.3 a comparison of the proposed control system with Ohta and OCC algorithm is provided.

4.1 Experimental Scenario and Metrics

Figure 3 shows the testbed employed for the experimental evaluation. Two Linux PCs are connected over a 1000 BaseT Ethernet LAN. We adopted a point-to-point topology by using SIPp[2] to generate the *INVITE* stream at a configurable rate. SIPp was also used to emulate the upstream SIP server behavior. 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.

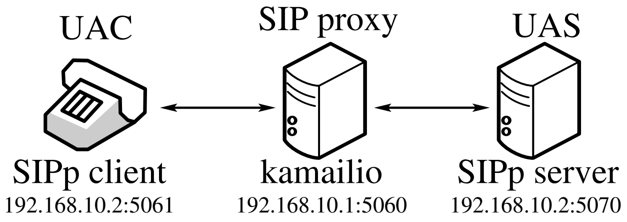


Fig. 3. The testbed employed for the experimental evaluation

We have adopted the goodput as the main metric to evaluate and compare the performance of the three algorithms. We also considered 1) the retransmissions ratio defined as the ratio between the number of retransmissions and the total received calls and 2) the response time, i.e. the time required to establish a call.

For each $r_i \in R_i = \{0.53, 0.79, ..., 2.61\}$ we ran $m = 5$ experiments, whose duration was 120s, and we measured the obtained goodput $g_k(r_i)$ for $k = 1, \ldots, m$. We then evaluated the $g(r_i)$ as the average of the goodputs $g_k(r_i)$. Finally, in the case of the proposed control system and OCC, we have carried out the experiments considering the CPU target equal to 0.8 and 0.9.

[1] `http://www.kamailio.org/`

[2] `http://sipp.sourceforge.net/`

4.2 Implementation Details

Kamailio is an open source SIP proxy server written in C language for Unix-like operating systems. It is distributed under the GPL license and it is widely employed both for scientific and commercial purposes. Kamailio employs an architecture which is made of a *core,* which provides basic SIP server functionalities, and several pluggable modules, extending the core. The Kamailio core does not implement a queuing structure, since it processes the incoming messages synchronously with their arrival. Since both Ohta and the proposed control system require the incoming messages to be enqueued before being processed, we implemented in the core a queue where the *INVITE* messages are stored[3]. The three overload controllers were implemented in a Kamailio module called `ratelimit`. In the following we provide a description of the considered algorithms and their implementation details.

The Proposed Control System. Both the PI controllers were 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 $T_c = 20$ms, while another timer samples the CPU load every $T_m = 10$ms. The maximum queue length was set to 800 calls. The parameters of the controllers described in Section 3.2 have been optimized by means of the iterative algorithm Extremum Seeking [12] with the aim of maximizing the obtained goodput. The optimal parameters obtained were $K_{pq} = 20$, $K_{iq} = 130$ $K_{pc} = 5$, $K_{ic} = 5$.

Ohta. 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 as our algorithm, Ohta was implemented in a similar way in the "ratelimit" module. Queue buffer size was set to 1000, `lo_wm` and `hi_wm` to 400 and 800 respectively as suggested in [3].

OCC. The algorithm dynamically adjusts the probability f of accepting an incoming *INVITE* request based on measurements of the CPU load to drive it to a target utilization ρ_{targ}.If the CPU utilization ρ is larger than the target value ρ_{targ}, the load is reduced by rejecting a higher percentage of incoming requests. The control law is a discrete time nonlinear controller described by the following equation:

$$f^{k+1} = \begin{cases} f_{min} & \phi^k f^k < f_{min} \\ 1 & \phi^k f^k > 1 \\ \phi^k f^k & \text{otherwise} \end{cases}$$

[3] The other messages are forwarded as usual.

where f^k is the acceptance ratio, $\phi^k = \min(\frac{\rho_{targ}}{\rho}, \phi_{max})$. f_{min} avoids to have zero minimal acceptance ratio, whereas $\phi_{max} > 1$ is the maximum multiplicative increase factor. The CPU occupancy is measured every measurement interval T_m , whereas the algorithm actuation variable is update every control interval T_c.

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 = 1sec, a measurement interval T_m = 0.1sec, $f_{min} = 0.02$, and $\phi_{max} = 5$ as suggested in [3].

4.3 Comparison with Ohta and OCC

In this section we compare the proposed control system, named PI in the following, with Ohta and OCC algorithms.

Figure 4, Figure 5 (a), and Figure 5 (b) show respectively the goodput curves, the retransmission ratio, and the response time W for each of the considered SIP overload controllers. Figure 4 shows that the proposed control system achieves significantly better performance in terms of goodput, retransmissions ratio, and response time.

In particular, the proposed control system achieves a normalized goodput equal to 0.4 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% betters handle overload *wrt* OCC 90%, and it is able to support input rates up to

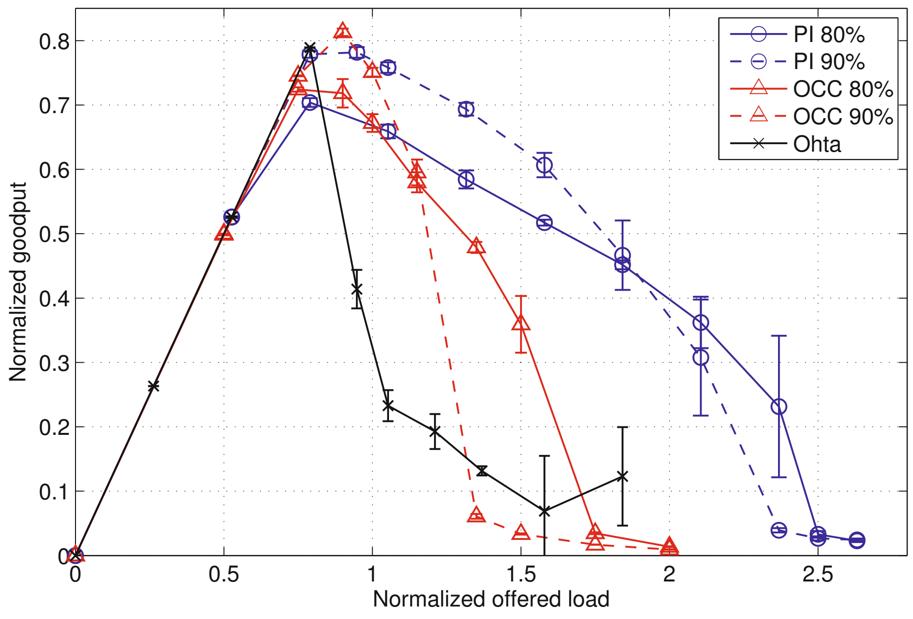
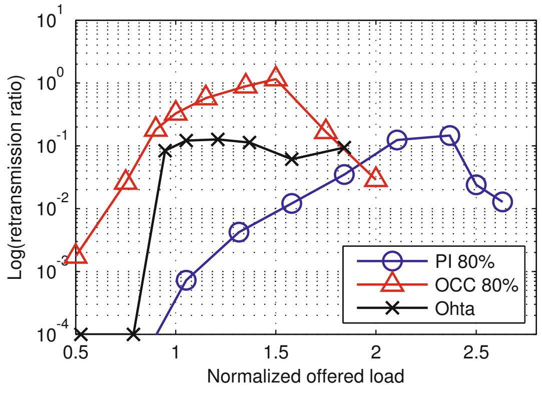
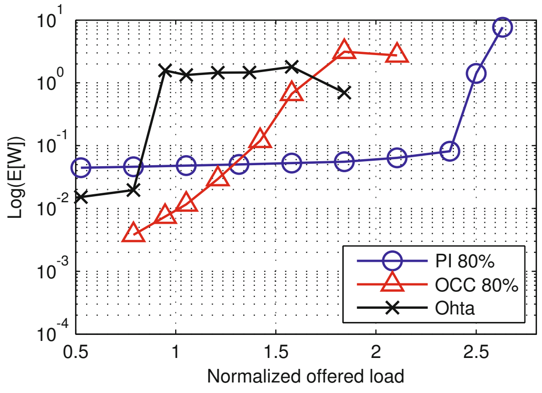


Fig. 4. Goodput comparison



(a) Retransmissions ratio

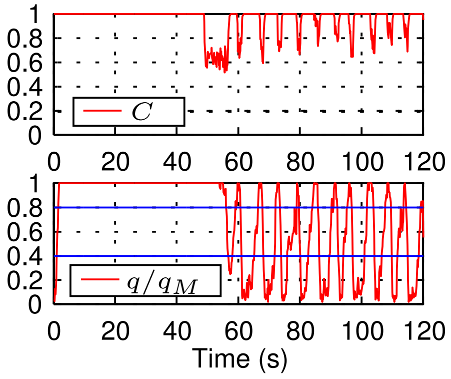


(b) Response time

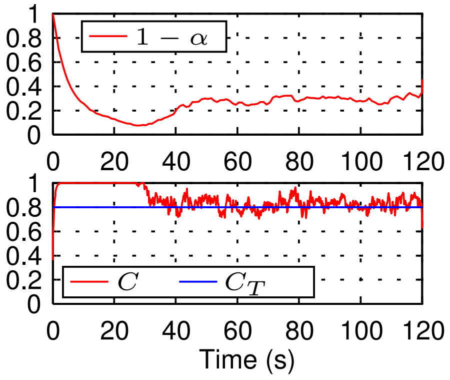
Fig. 5. Retransmission ratio and response time comparison

1.7. For what concerns the Ohta algorithm, Figure 4 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 being able to properly handle overload episodes. Moreover, measurements obtained for input rates which are higher than the server maximum processing capacity are not statistically relevant since the overload situation was so heavy that only the initial calls were accepted by the SIPp server.

Figure 5 (a) shows that the proposed control system maintains the retransmission ratio below 0.1 for a normalized rate equal to 2, 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 are not controlled for large values of r. Let us consider the Figure 5 (b) which shows the response time. The proposed algorithm maintains a response time which matches the target value $T_q = 0.05$s for every



(a) Ohta

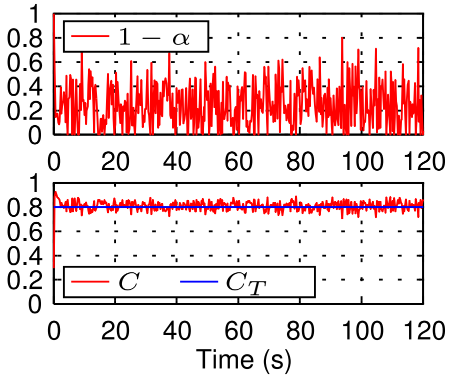


(b) OCC

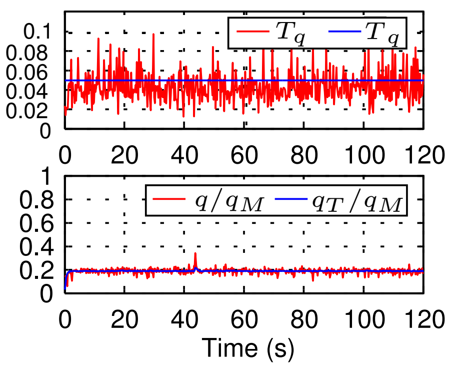
Fig. 6. Ohta ($r = 1.1$) and OCC ($r = 1.57$) dynamic behaviour

input load: this confirms that the first control loop tracks the reference signal $q_T(t)$. On the other hand, OCC exhibits very high average response time.

Figure 7, Figure 8, and Figure 6 show the dynamic behaviour of the considered algorithms. Figure 7 (b) confirms that the queue control loop of the proposed control system tracks the reference signal $q_T(t)$, thus indirectly tracking the target queuing time $\overline{T}_q = 0.05$s. Moreover, Figure 7 (a) shows that the CPU control loop quickly steers the CPU load $C(t)$ to the target C_T effectively avoiding overload. Figure 8 shows the estimated arrival rates, drawn with solid lines, for several input rates, drawn with dashed lines. It is important to notice that the estimated arrival rate $\tilde{r}$ is the sum of the incoming rate and the retransmission rate. The figure shows that retransmissions are very low for rates less



(a) Accept ratio $1-\alpha$ and CPU load C



(b) Queuing time T_q and normalized queue length q/q_M

Fig. 7. The proposed control system dynamic behaviour ($r = 1.57$)

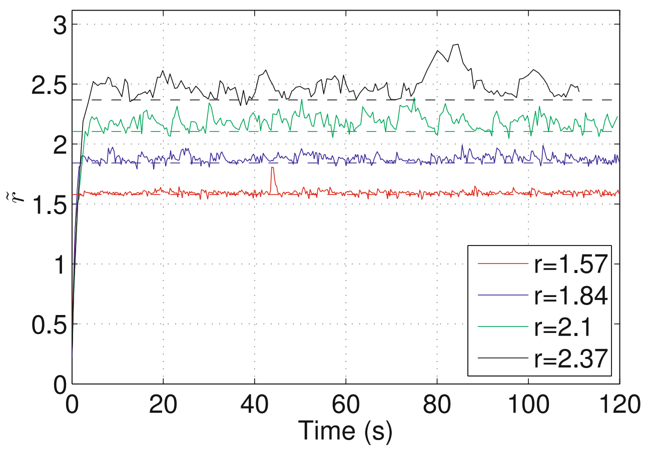


Fig. 8. The estimated arrival rate $\tilde{r}$ for the proposed control system

than $r = 2.01$, whereas for $r \geq 2.01$ larger retransmission rates occur. However, the figure shows that these retransmissions are effectively controlled and they are rejected at steady state.

Figure 6 (b) shows the dynamics of OCC for a normalized input rate $r = 1.57$: the figure shows that the accept ratio dynamics for OCC is slow and, as a consequence,during the first 25s the CPU is overloaded ($C(t) = 1$). Finally, Figure 6 (a) shows the dynamics of Ohta obtained when $r = 1.1$. The queue length exhibits remarkable oscillations and the overload is not avoided.

5 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 evaluation 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 response time. The proposed control system handles overload up to a maximum normalized input load equal to 2.3, OCC supports input rates lower than 1.7, whereas Ohta fails to properly handle overload.

Acknowledgement. This work has been partially supported by the project "Platform for Innovative services in the Future Internet" (PLATINO - PON 2007IT161PO006) funded by Italian Ministry of Education, Universities and Research (MIUR).

References

1. Rosenberg, J., et al.: SIP: Session Initiation Protocol. RFC 3261 (June 2002)
2. Rosenberg, J.: Requirements for Management of Overload in the Session Initiation Protocol. RFC 5390, Internet Engineering Task Force (December 2008)
3. Hilt, V., Widjaja, I.: Controlling Overload in Networks of SIP Servers. In: Proc. IEEE ICNP, pp. 83–93 (October 2008)
4. Ohta, M.: Overload control in a SIP signaling network. Enformatika Transactions in Enginnering, Computing and Technology, 205–210 (2006)
5. Shen, C., Schulzrinne, H., Nahum, E.: Session initiation protocol (SIP) server overload control: Design and evaluation. In: Schulzrinne, H., State, R., Niccolini, S. (eds.) IPTComm 2008. LNCS, vol. 5310, pp. 149–173. Springer, Heidelberg (2008)
6. Noel, E., Johnson, C.: Novel overload controls for SIP networks. In: 21st International Teletraffic Congress, ITC 21 2009, pp. 1–8. IEEE (2009)
7. Hong, Y., Huang, C., Yan, J.: Mitigating sip overload using a control-theoretic approach. In: Proc. of IEEE GLOBECOM 2010, pp. 1–5 (2010)
8. Garroppo, R., Giordano, S., Niccolini, S., Spagna, S.: A prediction-based overload control algorithm for SIP servers. IEEE Transactions on Network and Service Management (99), 1–13 (2011)
9. Wang, Y.: SIP overload control: a backpressure-based approach. In: Proc. of ACM SIGCOMM 2010, pp. 399–400 (2010)
10. Jiang, H., et al.: Load Balancing for SIP Server Clusters. In: Proc. IEEE INFOCOM, pp. 2286–2294 (April 2009)
11. Franklin, G., Powell, J., Emami-Naeini, A.: Feedback control of dynamic systems. Addison-Wesley (1994)
12. Killingsworth, N., Krstic, M.: PID tuning using extremum seeking: online, model-free performance optimization. IEEE Control Systems 26(1), 70–79 (2006)