Mitigating SIP Overload Using a Control-Theoretic Approach

Yang Hong, Changcheng Huang, James Yan
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada
E-mail: {yanghong, huang}@sce.carleton.ca, jim.yan@sympatico.ca

***Abstract*—Retransmission mechanism helps SIP maintain its reliability, but it can also make an overload worse. Recent server collapses due to emergency-induced call volume in carrier networks indicate that the built-in overload control mechanism cannot handle overload conditions effectively. Since the retransmissions caused by the overload are redundant, we suggest mitigating the overload by controlling redundant message ratio to an acceptable level. Using control-theoretic approach, we model the interaction of an overloaded downstream server with its upstream server as a feedback control system. Then we develop an adaptive PI control algorithm to mitigate the overload at the downstream server by controlling the retransmission message rate of its upstream servers. By performing OPNET simulations on two typical overload scenarios, we demonstrate that: (1) without overload control algorithm applied, the overload at the downstream server may propagate to its upstream servers; (2) our control-theoretic solution not only mitigate the overload effectively, but also achieve a satisfactory target redundant message ratio.**

***Index Terms*—SIP, Overload Control, Retransmission Rate Control, Redundant Message Ratio, Control System Stability, Phase Margin**

1. INTRODUCTION

SIP (Session Initiation Protocol) [1] is becoming the dominant signaling protocol for Internet-based communication services such as Voice-over-IP, instant messaging, and video conferencing. 3GPP (3rd Generation Partnership Project) has adopted SIP as the basis of its IMS (IP Multimedia Subsystem) architecture [2-4]. With the 3G (3rd Generation) wireless technology being adopted by more and more carriers, most cellular phones and other mobile devices are starting to use or are in the process of supporting SIP for multimedia session establishment [3].

SIP introduces a retransmission mechanism to maintain its reliability [1, 5]. In practice, a SIP sender uses timeout to detect message losses. One or more retransmissions would be triggered if the corresponding reply message is not received in predetermined time intervals. When the message arrival rate exceeds the message processing capacity at a SIP server, overload occurs and the queue increases, which may result in a long queuing delay and trigger unnecessary message retransmissions from its upstream servers. Such redundant retransmissions increase the CPU loads of both the overloaded server and its upstream servers. This may propagate the overload and bring potential network collapse [4, 6-18].

SIP RFC 3261 [1] suggests that the SIP retransmission mechanism should be disabled for hop-by-hop transaction when running SIP over TCP to avoid redundant retransmissions at both SIP and TCP layer [1]. However, nearly all vendors choose to run SIP over UDP instead of TCP for the following reasons [4, 6-21]: (1) The reliability function provided by TCP does not consider real time application which is a critical requirement for SIP protocol; (2) SIP works at application layer while TCP works at transport layer. Even TCP can provide reliability at transport layer, SIP messages can still be dropped or corrupted while being processed at application layer; (3) Designed for preventing congestion caused by bandwidth exhaustion, the complex TCP congestion control mechanism provides little help for SIP overload which is caused by CPU constraint.

RFC 5390 [19] identified the various reasons that may cause server overload in a SIP network. These include poor capacity planning, component failures, flash crowds, denial of service attacks, etc. Recent collapses of SIP servers due to "American Idol" flash crowd in real carrier networks have motivated several overload control solutions. For example, both centralized and distributed overload control mechanisms for SIP were developed in [9]. Three window-based feedback algorithms were proposed to adjust the message sending rate of the upstream SIP servers based on the queue length [10]. Retry-after control, processor occupancy control, queue delay control and window based control were proposed to improve goodput and prevent overload collapse in [6]. However, these overload control proposals suggested that the overloaded receiving server advertises to its upstream sending servers to reduce their sending rates. Such pushback control solution would produce overload propagation and block a large amount of calls unnecessarily, thus reducing the revenue of the service providers. Since retransmissions caused by the overload bring extra overhead instead of reliability to the network and exacerbate the overload [16], we suggest mitigating the overload by reducing the retransmission rate only.

The contributions of this paper are: (1) Using a control-theoretic approach to model an overloaded downstream server and its upstream server as a feedback control system (as shown in Fig. 4 later on); (2) Proposing a novel PI control algorithm to mitigate the overload and achieve a satisfactory target redundant message ratio by controlling retransmission rate; (3) Performing OPNET simulations under two typical overload scenarios to validate our overload control algorithm. Experimental results will demonstrate that our control-theoretic solution can help the overloaded downstream server to mitigate the overload and prevent the overload from propagating to its upstream servers.

2. SIP RETRANSMISSION MECHANISM OVERVIEW

Fig. 1 describes a basic SIP operation among originating UA (User Agent), SIP P-server (Proxy-server) and terminating UA. To set up a call, an originating UA sends an "Invite" request to

a terminating UA via two P-servers. The P-server returns a provisional "100(Trying)" response to confirm the receipt of the "Invite" request. The terminating UA returns a "180(Ringing)" response after confirming that the parameters are appropriate. It also evicts a "200(OK)" message to answer the call. The originating UA sends an "ACK" response to the terminating UA after receiving the "200(OK)" message and the call session is established. The "Bye" request is generated to close the session thus terminating the communication.

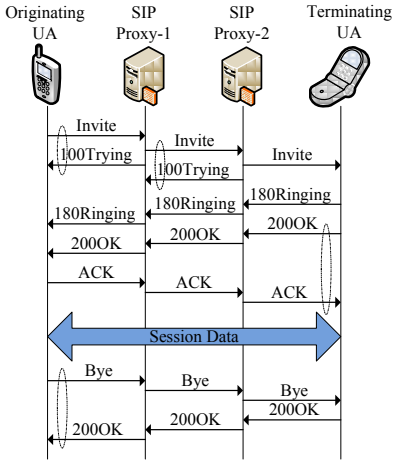


Fig. 1. A typical procedure of session establishment.

The hop-by-hop Invite-100(Trying) transaction is the major workload contributor in case of overload [9]. Therefore, given the proportionate nature and the general similarity of the retransmission mechanisms between the "Invite" and "non-Invite" messages in a typical session [1], we will focus on the hop-by-hop Invite-100(Trying) transaction in this paper. For each hop, the sender starts the first retransmission of the original message at T_1 seconds, and the time interval doubles after every retransmission (exponential back-off), if the corresponding reply message is not received. There is a maximum of 6 retransmissions [1].

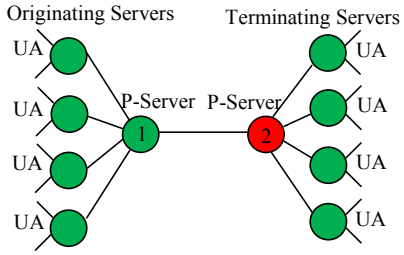


Fig. 2. SIP network topology with an overloaded downstream receiving Server 2 (which is marked in red color) and its upstream sending Server 1.

3. Control-Theoretic Approach To Mitigate SIP Overload

The topology of a real SIP network can be quite complex. Fig. 2 depicts a typical SIP network topology [9]. To focus our study on the interactions between overloaded receiving Server 2 and its upstream sending Server 1, we assume the upstream servers of Server 1 and the downstream servers of Server 2 have sufficient capacity to process all requests, retransmissions, and response messages immediately without any delay. Practical buffer sizes vary with the actual service rates and system configuration plans. With the memory becoming cheaper and cheaper, typical buffer sizes are likely to become larger and larger. We assume that the buffer sizes for all servers are large enough to avoid message loss. Instead, we focus on the delay caused by the overloaded server, which may trigger premature retransmissions.

3.1 Queuing Dynamics of Overloaded Server

Fig. 3 depicts the queuing dynamics of Server 1 and Server 2. There are two queues at each server: one to store the messages and the other to store the retransmission timers [9, 12]. We can obtain the queuing dynamics for the message queue of Server 2 as

$$\dot{q}_2(t) = \lambda_2(t) + r_2(t) + \upsilon_2(t) - \mu_2(t), \quad (1)$$

where $q_2(t)$ denotes the queue size and $q_2(t) \geq 0$; $\lambda_2(t)$ denotes original message rate; $r_2(t)$ denotes retransmission message rate; $\upsilon_2(t)$ denotes response message rate; $\mu_2(t)$ denotes the message service rate.

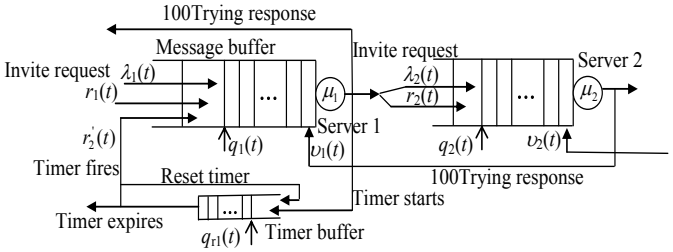


Fig. 3. Queuing dynamics of an overloaded server and its upstream server.

Like Eq. (1), we can obtain queuing dynamics for the message queue of Server 1 as

$$\dot{q}_1(t) = \lambda_1(t) + r_1(t) + r_2'(t) + \upsilon_1(t) - \mu_1(t), \quad (2)$$

where $q_1(t)$ denotes the queue size and $q_1(t) \geq 0$; $\lambda_1(t)$ denotes original message rate; $r_1(t)$ denotes retransmission message rate corresponding to $\lambda_1(t)$; $r_2'(t)$ denotes retransmission message rate generated by Server 1 for $\lambda_2(t)$; $\upsilon_1(t)$ denotes response message rate corresponding to $\lambda_1(t)$, and the response messages will remove the corresponding retransmission timers from timer queue q_{r1}; $\mu_1(t)$ denotes the message service rate.

When Server 2 performs its routine maintenance and reduces its service capacity for signaling messages, the original message rate $\lambda_2(t)$ is larger than the service rate $\mu_2(t)$, the queue size $q_2(t)$ tends to increase according to Eq. (1) (i.e., $\dot{q}_2(t) > 0$). After a short period, the queuing delay of Server 2 is long enough to trigger the retransmissions $r'_2(t)$ which enter the queue of Server 1. If the total new message arrival rate of $\lambda_1(t)$, $\upsilon_1(t)$ and $r'_2(t)$ is larger than the service rate $\mu_1(t)$, the queue size q_1 would increase (i.e., $\dot{q}_1(t) > 0$, as indicated by Eq. (2)) and may trigger the retransmissions $r_1(t)$ to introduce overload to Server 1. After queuing and processing delay at Server 1, the retransmitted messages $r'_2(t)$ enter Server 2 as $r_2(t)$ to increase the queue size $q_2(t)$ more quickly (as described by Eq. (1)), thus making the overload at Server 2 worse.

3.2 Overload Controller Design Using a Control-Theoretic Approach

As the retransmitted messages $r'_2(t)$ may increase queue sizes at both Server 1 and Server 2 and bring the overload to both servers, our goal for mitigating the overload is to control the retransmission rate $r'_2(t)$ using a control-theoretic approach,

thus preventing the queue sizes at both Server 1 and Server 2 from increasing continuously.

Only a retransmitted message for message loss recovery is a non-redundant request message as well as an original message, while a retransmission caused by the overload delay is redundant. Thus a response message corresponding to a redundant retransmitted message is redundant. When overload happens, most of retransmitted messages $r'_2(t)$ are redundant [9] and acknowledged as the response messages $\upsilon_{1r}(t)$ after a round trip delay τ, i.e., $\upsilon_{1r}(t) \approx r'_2(t-\tau)$. We define the redundant message ratio γ as the ratio between the redundant response message rate υ_{1r} and the total response message rate υ_1, i.e.,

$$\gamma(t)=\upsilon_{1r}(t)/\upsilon_1(t) \approx r'_2(t-\tau)/\upsilon_1(t). \quad (3)$$

In the real-time implementation, we count the number $N_{\upsilon 1r}$ of the arrival redundant response messages and the number $N_{\upsilon 1}$ of the total arrival response messages during a sampling time interval T_s, then we can obtain υ_{1r} and υ_1 as $\upsilon_{1r}=N_{\upsilon 1r}/T_s$ and $\upsilon_1=N_{\upsilon 1}/T_s$. Considering average 8% packet loss in the Internet [22], it is necessary to maintain a target redundant message ratio γ_0 for message loss recovery. Our numerous experiments suggest that γ_0=0.1 is a good choice.

Now we assume that the system is locally stable and therefore τ and $\upsilon_1(t)$ are constant. The transfer function between the instantaneous redundant message ratio $\gamma(t)$ and the retransmission rate $r'_2(t)$ is given by

$$P(s)=\gamma(s)/r'_2(s)=[r'_2(s)e^{-s\tau}/\upsilon_1]/r'_2(s)=e^{-s\tau}/\upsilon_1. \quad (4)$$

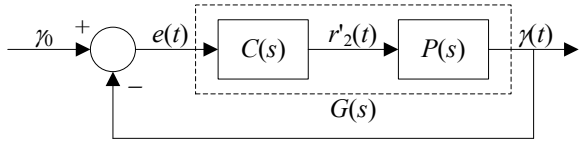


Fig. 4. Feedback SIP overload control system.

Fig. 4 depicts a feedback SIP overload control system, where the overload control plant $P(s)$ represents the interaction between an overloaded downstream receiving server and its upstream sending server, and adaptive PI controller $C(s)$ is designed for mitigating the overload and achieving a desirable target redundant message ratio γ_0, when the overload is anticipated at the downstream server.

Based on the instantaneous redundant message ratio $\gamma(t)$, the retransmission rate $r'_2(t)$ can be obtained by the following PI control algorithm expressed via

$$\begin{aligned} r'_2(t) &= K_P e(t) + K_I \int_0^t e(\eta)\mathrm{d}\eta \\ &= K_P(\gamma_0 - \gamma(t)) + K_I \int_0^t (\gamma_0 - \gamma(\eta))\mathrm{d}\eta \end{aligned}. \quad (5)$$

where K_P and K_I denote the proportional gain and integral gain of the PI controller at the upstream server. In the real-time implementation, a retransmission probability is equal to the ratio between the retransmission rate $r'_2(t)$ and the measured timer expiration rate. It can be easy to obtain the transfer function between the retransmission rate $r'_2(t)$ and the redundant message ratio deviation $e(t)$ as

$$C(s)=K_P+K_I/s. \quad (6)$$

Since the control plant described by Eq. (4) is valid only during the overload period, we only activate the PI control algorithm when overload happens. The open-loop transfer function of overload control system becomes

$$G(s)=C(s)P(s)=(K_P+K_I/s)e^{-s\tau}/\upsilon_1. \quad (7)$$

It is well known that a positive phase margin (ϕ_m>0) can guarantee the stability of the control system in accordance with the Nyquist Stability Theorem [23]. A common control engineering practice suggests an interval of phase margin as $30^0 \le \phi_m \le 60^0$ for a good response [23]. From the definition on the phase margin ϕ_m of $G(s)$ [23], we can obtain

$$\arctan\left[\frac{K_P\omega_g}{K_I}\right] - \frac{\pi}{2} - \omega_g\tau = -\pi + \phi_m, \quad (8)$$

$$\frac{\sqrt{K_P^2\omega_g^2 + K_I^2}}{\omega_g}\frac{\left|e^{-j\omega_g\tau}\right|}{\upsilon_1} = \frac{\sqrt{K_P^2\omega_g^2 + K_I^2}}{\omega_g\upsilon_1} = 1, \quad (9)$$

where ω_g is the gain crossover frequency of the overload control system. To simplify our controller design, we set $K_p\omega_g=K_I$. Thus we can rewrite Eqs. (8) and (9) as

$$\omega_g\tau = \frac{3\pi}{4} - \phi_m, \text{ and } \frac{\sqrt{2}K_I}{\omega_g\upsilon_1} = 1. \quad (10)$$

Using the relationship $K_P=K_I/\omega_g$, we can obtain K_P and K_I from Eq. (10) as

$$K_P = \frac{\upsilon_1}{\sqrt{2}}, \text{ and } K_I = \frac{\upsilon_1(3\pi/4 - \phi_m)}{\sqrt{2}\tau}. \quad (11)$$

Overload Control Algorithm

When each retransmission timer fires or expires
  if $\tau < T_1$
    Retransmit the message
  else
    Retransmit the message with a retransmission probability corresponding to a retransmission rate r'_2 calculated by a PI controller

Adaptive PI control algorithm:

(1) Specify target redundant message ratio γ_0 and phase margin ϕ_m; Set the initial values for τ, υ and ζ; Obtain PI controller gains using Eq. (11).

(2) Calculate τ, υ, ζ and γ upon response message arrivals.

(3) If $\zeta > 1.5\zeta_0$ or $\zeta < 0.5\zeta_0$, self-tune PI controller gains using Eq. (11) and update $\zeta_0=\zeta$; Otherwise, PI controller remains unchanged.

(4) Calculate the retransmission rate r'_2 using Eq. (5); Go to Step (2).

Varying parameter:

τ : Round trip delay
υ : Response message rate
γ : Redundant message ratio
K_P : Proportional gain of PI controller
K_I : Integral gain of PI controller
r'_2 : Message retransmission rate
ζ : Monitoring parameter

Fixed parameter:

T_1 : First-time retransmission timer
γ_0 : Target redundant message ratio
ϕ_m : Phase margin

Fig. 5. Overload control algorithm using control-theoretic approach.

So far we have assumed υ_1 and τ be constant. In reality, this is not necessarily true. If the PI controller parameters K_P and

K_I are kept constant, the varying queuing dynamics of the overloaded server may cause the change in two SIP network parameters (υ_1 and τ), thus may drive the phase margin to negative and the overload control system into instability. Lemmas 1 and 2 show the impact of the two SIP network parameters on the phase margin or the system stability. The proofs of the two lemmas are omitted due to space limitation.

Lemma 1: ***If*** current response message rate υ'_1 is lower than previous response message rate υ_1 (that is, $\upsilon'_1<\upsilon_1$) and the PI controller is designed based on υ_1, ***then*** the overload control system with υ'_1 will have less phase margin than that with υ_1 (that is, $\phi'_m<\phi_m$).

Lemma 2: ***If*** the current round trip delay τ' is longer than the previous round trip delay τ (i.e., $\tau'>\tau$) and the PI controller is designed based on τ, ***then*** the overload control system with τ' will have less phase margin than that with τ (that is, $\phi'_m<\phi_m$).

To maintain the stability of the system, we self-tune PI controller when dramatic change of network parameters (υ_1 and τ) drifts the heuristic parameter $\zeta=\tau/\upsilon_1$ out of its interval.

We use the round trip delay τ to detect the overload, thus determining whether to activate PI control algorithm. If $\tau<T_1$, retransmit all the messages whose retransmission timers fire or expire; If $\tau\geq T_1$, overload is anticipated at the downstream server, thus the upstream servers retransmit the messages with the retransmission rate calculated by PI controller. Summary of our overload control algorithm is shown in Fig. 5.

4. Performance Evaluation And Simulation

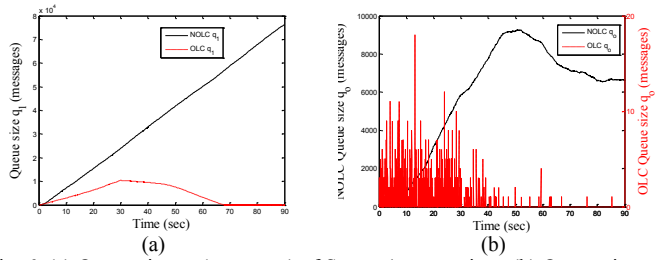
To verify our PI controller, we conducted OPNET simulations to observe the transient behaviour of the overloaded server and its upstream servers based on the network topology in Fig. 2. Four originating servers generated original request messages with equal rate. Both message generation rate and service rate are Poisson distributed[1]. Since processing a response message takes much less time than processing a request message, the time ratio is set to be α=0.5. The mean service capacity of a Proxy server is 1000 messages/sec measured based on the processing time of request message, i.e. $C_1=C_2$=1000 request messages/s. That is, the mean processing times for a request message and a response message are 1ms and 0.5ms respectively. The mean service capacity of an originating server or a termination server is equal to 500 request messages/sec. The total message service rate μ is bounded by the service capacity C at each server, i.e., $\mu\leq C$. The target redundant message ratio γ_0 is set as 0.1. The phase margin is set as 45^0. Average message loss probability is 10%.

Two typical overload scenarios were simulated: (1) Overload at Server 1 due to a demand burst; (2) Overload at Server 2 due to a server slowdown. The simulation time is 90s, and the 1st-time retransmission timer is T_1=500ms [1]. In each scenario, we performed our simulations with overload control algorithm and without overload control algorithm separately. In all the simulation plots in this paper, we use "OLC"/"NOLC" to indicate that overload control algorithm "was"/"was not" applied to all servers in the SIP network.

[1] Currently there is no measurement result for the workload in the real SIP networks. Poisson distributed message arrival rate and service rate are widely adopted by most existing research work (e.g., [10]).

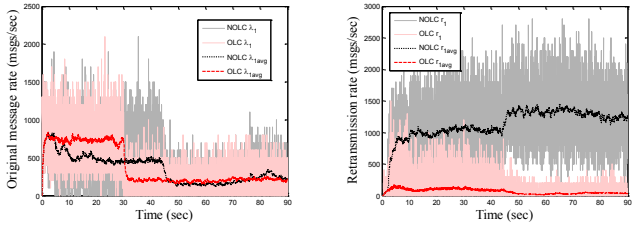
4.1 Overload at Server 1

In this scenario, the mean message generation rate for each original server was 200 messages/sec (i.e., λ_1=800 messages/sec, emulating a short surge of user demands) from time t=0s to t=30s, and 50 messages/sec (i.e., λ_1=200 messages/sec, emulating regular user demands) from time t=30s to t=90s. The mean service capacities of two proxy servers were $C_1=C_2$=1000 messages/sec.



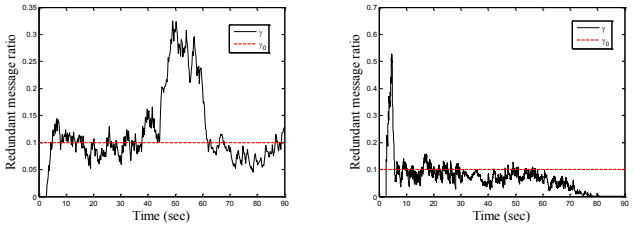
(a) (b)

Fig. 6. (a) Queue size q_1 (messages) of Server 1 versus time. (b) Queue size q_o (messages) of an originating server versus time.



(a) (b)

Fig. 7. (a) Original message rate λ_1 and moving average original message rate λ_{1avg} (messages/sec) of Server 1 versus time. (b) Retransmission rate r_1 and moving average retransmission rate r_{1avg} (messages/sec) for Server 1 versus time.



(a) (b)

Fig. 8. Redundant message ratio γ versus time. (a) Overload scenario 1. (b) Overload scenario 2.

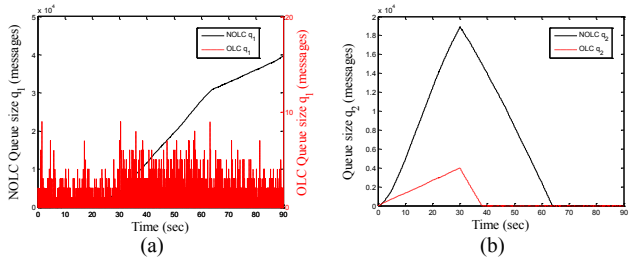
Figs. 6 and 7 show the dynamic behaviour of overloaded Server 1 and one of its upstream originating servers.

Without overload control algorithm applied, it is easy to see from Fig. 6(a) that Server 1 became CPU overloaded immediately and the overload deteriorated as time evolves, leading to the eventual crash of Server 1. Since the aggregate service capacity of four originating servers was larger than that of proxy Server 1, the queue size of each originating server decreased slowly (see Fig. 6(b)) after new original message generation rates decreased.

Our overload control algorithm made the queue size of Server 1 increase slowly during the period of the demand burst, and cancelled the overload at Server 1 within 38s after the new user demand rate reduced at time t=30s. PI controller helped the redundant message ratio (depicted in black) to clamp its target value (depicted in red) during the overload period, as shown in Fig. 8(a).

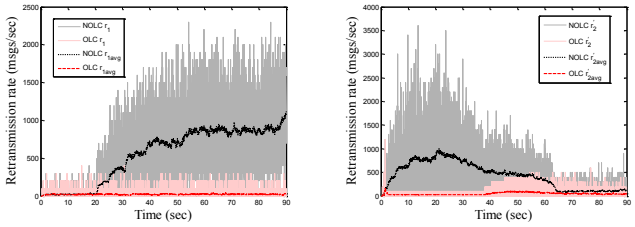
4.2 Overload at Server 2

In this scenario, the mean server capacities of the two proxy servers were C_1=1000 messages/sec from time t=0s to t=90s, C_2=100 messages/sec from time t=0s to t=30s, and C_2=1000 messages/sec from time t=30s to t=90s. The mean message generation rate for each original server was 50 messages/sec.



(a) (b)

Fig. 9. (a) Queue size q_1 (messages) versus time. (b) Queue size q_2 (messages) versus time.



(a) (b)

Fig. 10. (a) Retransmission rate r_1 and moving average retransmission rate r_{1avg} (messages/sec) for Server 1 versus time. (b) Retransmission rate r'_2 and moving average retransmission rate r'_{2avg} (messages/sec) for Server 2 versus time.

Without overload control algorithm applied, Figs. 9 and 10 demonstrate that Server 2 became overloaded first, which was followed by a later overload at Server 1. The queue size at Server 1 increased faster due to the extra work load for handling retransmissions for both Server 1 and Server 2. After Server 2 resumed its normal service at time t=30s, Server 1 and Server 2 had the same service capacity. Because Server 1 had to process part of r_1 which would not enter Server 2, the total arrival rate at Server 2 was less than its service capacity. Eventually the overload at Server 2 was cancelled, while the overload at Server 1 persisted (see Fig. 9).

Our overload control algorithm regulated the retransmission rate r'_2 to mitigate the overload effectively and clamp the redundant message ratio around its target value (see Fig. 8(b)). After Server 2 resumed its normal service, it only spent 9s to cancel the overload and the buffer became empty at time t≈39s.

5. Conclusions

Using a control-theoretic approach, we have modelled an overloaded downstream receiving server and its upstream sending server as a feedback control system. Then we have developed an innovative adaptive PI control algorithm to mitigate the overload by controlling retransmission rate while achieving a desirable target redundant message ratio.

By analyzing the queuing dynamics and performing OPNET simulations, we have demonstrated that without overload control algorithm applied, the overload at the downstream server may propagate or migrate to its upstream servers eventually. Without protocol modification, our overload control algorithm can mitigate the overload effectively and prevent the overload propagation. We will compare the performance of our overload algorithm with other existing overload solutions in different networks under different workload generators (e.g., [15]) in our future work.

Acknowledgment

We appreciate the financial support from the NSERC grant #CRDPJ 354729-07 and the OCE grant #CA-ST-150764-8.

References

[1] J. Rosenberg et al., "SIP: Session Initiation Protocol," *IETF RFC 3261*, June 2002.
[2] "3rd Generation Partnership Project". http://www.3gpp.org.
[3] S.M. Faccin, P. Lalwaney, and B. Patil, "IP Multimedia Services: Analysis of Mobile IP and SIP Interactions in 3G Networks," *IEEE Communications Magazine*, 42(1), January 2004, pp. 113-120.
[4] E. Noel and C.R. Johnson, "Initial simulation results that analyze SIP based VoIP networks under overload," *Proceedings of 20th International Teletraffic Congress*, 2007, pp. 54-64.
[5] M. Govind, S. Sundaragopalan, K. S. Binu, and S. Saha, "Retransmission in SIP over UDP - Traffic Engineering Issues," *Proceedings of International Conference on Communication and Broadband Networking*, Bangalore, India, May 2003.
[6] E. Noel and C.R. Johnson, "Novel Overload Controls for SIP Networks," *Proceedings of 21st International Teletraffic Congress*, 2009.
[7] R.P. Ejzak, C.K. Florkey, and R.W. Hemmeter, "Network Overload and Congestion: A comparison of ISUP and SIP," *Bell Labs Technical Journal*, 9(3), 2004, pp. 173–182.
[8] M. Ohta, "Overload Control in a SIP Signaling Network," *Proceeding of World Academy of Science, Engineering and Technology*, Vienna, Austria, March 2006, pp. 205—210.
[9] V. Hilt and I. Widjaja, "Controlling Overload in Networks of SIP Servers," *Proceedings of IEEE ICNP*, Orlando, Florida, October 2008, pp. 83-93.
[10] C. Shen, H. Schulzrinne, and E. Nahum, "SIP Server Overload Control: Design and Evaluation," *Proceedings of IPTComm*, Heidelberg, Germany, July 2008.
[11] A. Abdelal and W. Matragi, "Signal-Based Overload Control for SIP Servers," *Proceedings of IEEE CCNC*, Las Vegas, NV, January 2010.
[12] "SIP Express Router" http://www.iptel.org/ser/.
[13] T. Warabino, Y. Kishi and H. Yokota, "Session Control Cooperating Core and Overlay Networks for "Minimum Core" Architecture," *Proceedings of IEEE Globecom*, Honolulu, Hawaii, December 2009.
[14] Y. Hong, C. Huang, and J. Yan, "Analysis of SIP Retransmission Probability Using a Markov-Modulated Poisson Process Model," *Proceedings of IEEE/IFIP Network Operations and Management Symposium*, Osaka, Japan, April 2010, pp. 179–186.
[15] E.M. Nahum, J. Tracey, and C.P. Wright, "Evaluating SIP server performance," *Proceedings of international conference on Measurement and modeling of computer systems* (*ACM SIGMETRICS*), San Diego, CA, US, 2007, pp. 349–350.
[16] J. Sun, R.X. Tian, J.F. Hu, and B. Yang, "Rate-based SIP Flow Management for SLA Satisfaction," *Proceedings of* 11th International Symposium on Integrated Network Management (*IEEE/IFIP IM*), New York, USA, June 2009, pp. 125-128.
[17] Y. Hong, C. Huang, and J. Yan, J., "Modeling and Simulation of SIP Tandem Server with Finite Buffer," To appear in *ACM Transactions on Modeling and Computer Simulation*, April 2011.
[18] V. Hilt, I. Widjaja, and H. Schulzrinne, "Session Initiation Protocol (SIP) Overload Control," *IETF Internet-Draft*, draft-hilt-sipping-overload-07, October 2009.
[19] J. Rosenberg, "Requirements for Management of Overload in the Session Initiation Protocol," *IETF RFC 5390*, December 2008.
[20] W. R. Stevens, *TCP/IP Illustrated*, Volume 1, Addison-Wesley, Boston, 1994.
[21] Y. Hong, O. W. W. Yang, and C. C. Huang, "Self-Tuning PI TCP Flow Controller for AQM Routers With Interval Gain and Phase Margin Assignment," *Proceedings of IEEE Globecom*, Dallas, TX, U.S.A, November 2004, pp. 1324-1328.
[22] "Internet Traffic Report", http://www.internettrafficreport.com/, 2010.
[23] W.K. Ho, Y. Hong, A. Hansson, H. Hjalmarsson, and J.W. Deng, "Relay Auto-Tuning of PID Controllers Using Iterative Feedback Tuning," *Automatica*, **39**(1), 2003, pp. 149-157.