Applying Control Theoretic Approach To Mitigate SIP Overload

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*—The Session Initiation Protocol (SIP) retransmission mechanism is designed to maintain reliable transmission over lossy or faulty network conditions. However, the retransmission can amplify the traffic overload faced by the SIP servers. In this paper, by modeling the interaction between an overloaded downstream server and its upstream server as a feedback control system, we propose two Proportional-Integral (PI) control algorithms to mitigate the overload by regulating the retransmission rate in the upstream server. We provide the design guidelines for both overload control algorithms to ensure the system stability. Our OPNET® simulation results demonstrate that: (1) without the control algorithm applied, the overload at a downstream server may propagate to its upstream servers and cause widespread network failure; (2) in case of short-term overload, both proposed feedback control solutions can mitigate the overload effectively without rejecting calls or reducing resource utilization, thus avoiding the disadvantages of existing overload control solutions for SIP networks.**

***Index Terms*—Overload Control, Phase Margin, PI Rate Control, Server Crash, SIP, SIP Retransmission, System Stability**

I. INTRODUCTION

Internet protocol (IP) telephony is experiencing rapidly growing deployment due to its lower-cost telecommunications solutions for both consumer and business services. Session Initiation Protocol (SIP) [50] has become the main signaling protocol to setup, manage and terminate sessions for IP telephony applications such as Voice-over-IP, instant messaging and video conferencing [61]. SIP-based mobility management methods have been proposed to overcome the limitation of mobile IP and provide seamless mobility to the mobile terminals in an all-IP network infrastructure [51]. 3rd Generation Partnership Project (3GPP) [1] has adopted SIP as the basis of its IP Multimedia Subsystem (IMS) architecture [57]. With the

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 [15].

Fig. 1 illustrates a simplified architecture of a SIP network. The two basic elements of a SIP network are: User Agent (UA) and Proxy Server (P-Server) [50]. A P-server not only acts as the contact point with UA for core network service access but also provides routing for the signaling messages. Each P-server is assigned to serve a large number of individual UAs. SIP is responsible for establishing, modifying and terminating sessions for multimedia communication among multiple UAs. A basic SIP operation will be reviewed in Section II.

RFC 5390 [49] identified the various reasons that may cause server overload in a SIP network. These include but not limited to poor capacity planning, dependency failures, component failures, avalanche restart, flash crowds, etc. In general, anything that may trigger a demand burst or a server slowdown can cause server overload and lead to server crash. Existing Differentiated Service (DiffServ) mechanism proposed by Cisco® [10] aims at providing scalable end-to-end Quality of Service (QoS) in the IP layer, but cannot solve SIP overload issues due to the following three aspects: (1) DiffServ mechanism is designed subject to the bandwidth constraint in the IP layer, while SIP overload is subject to the CPU constraint in the application layer; (2) DiffServ mechanism enhances QoS for different types of end-to-end data or media traffic which normally needs to traverse multiple hops before reaching their destination, but SIP overload is mainly caused by hop-by-hop signaling traffic between the neighbouring servers, because hop-by-hop transactions consume most of CPU resources [23, 53]; (3) DiffServ mechanism achieves end-to-end QoS by differentiating different types of data or media traffic, where each source of data or media traffic sends large number of packets to its destination. However, each SIP UA only sends very few signaling messages to its destination for establishing a session [23]. Therefore, DiffServ mechanism cannot mitigate hop-by-hop SIP overload effectively.

There are many published works [2, 3, 11, 12, 16, 18, 20, 22, 23, 26, 36, 37, 39, 40, 42, 43, 45, 46, 48, 52, 53, 56, 58, 59, 60] discussing how to control SIP overload through call rejection or load balancing.

These mechanisms can either increase call rejection rate or cause SIP server underutilized [33]. Rejecting calls can cancel the overload effectively, but it may cause a large amount of revenue loss to carriers [28, 30, 31]. We believe that arbitrarily high volume of call rejection can be avoided in most overload situations by developing overload control algorithms that address the root cause of server overloading and widespread SIP network failures.

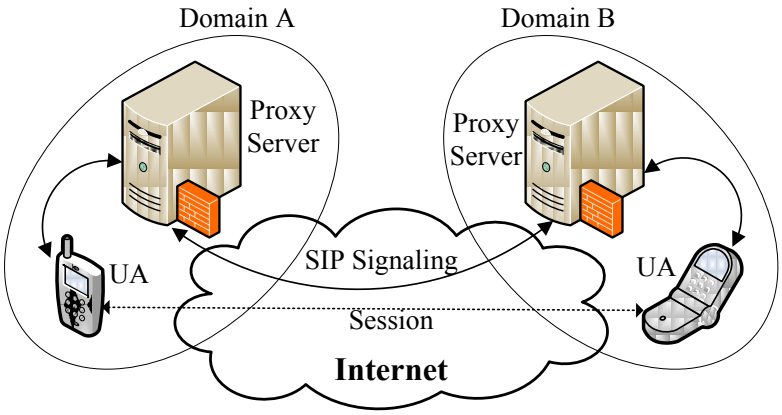


Fig.1. Simplified architecture of a SIP network

The objective of this paper is to develop two novel overload control algorithms through control theoretic approach that allows us to model the interaction between an overloaded downstream server and its upstream server as a feedback control system. Different from existing overload control solutions that try to mitigate the overload through rejecting calls or load balancing, our control approach will address directly unnecessary retransmissions. This is particularly useful when the overload is caused by short term demand bursts or temporary server slow down. For overloading caused by persistent demand surge or server slowdown, our solution can be used together with existing solutions to maximize server utilization.

The contributions of this paper are: (1) Proposing two novel Proportional-Integral (PI) control algorithms to mitigate the server overload at a SIP network. Each algorithm detects the overload implicitly, and then regulates retransmission rate to prevent the overload propagation and the network collapse. Redundant Retransmission Ratio Control (RRRC) algorithm mitigates the overload by clamping redundant retransmission ratio around a target ratio, while Round Trip Delay Control (RTDC) algorithm tries to

clamp round trip delay below a target delay; (2) Extending classical control theory to model the interaction between a downstream server and its upstream server as a feedback control system. We have used the frequency domain technique to establish the stability of the two proposed control algorithms; (3) Providing the guidelines for obtaining PI controller parameters to guarantee the stability of the SIP overload control system. We perform OPNET simulation to validate the efficiency of two proposed SIP overload control solutions.

The paper is organized as follows. Section II reviews SIP protocol briefly. Section III reviews existing SIP overload control solutions. Section IV investigates the impact of the retransmissions on the SIP overload. Section V proposes two different SIP overload control algorithms and applies the control theoretic approach in the controller design. Section VI verifies the validity of the proposed overload control algorithms using OPNET simulation. Conclusions are given in Section VII.

II. SIP PROTOCOL OVERVIEW

Fig. 2 illustrates a basic operation of a SIP network. To set up a call, a User Agent Client (UAC) sends an "Invite" request to a User Agent Server (UAS) via the two proxy servers. The proxy server returns a provisional "100 (Trying)" response to confirm the receipt of the "Invite" request. The user agent server returns a "180 (Ringing)" response after confirming that the parameters are appropriate. It also evicts a "200 (OK)" message to answer the call. The user agent client sends an "ACK" response to the user agent server after receiving the "200 (OK)" message. Finally the call session is established between the user agent client and the user agent server through the SIP session. The "Bye" request is generated to close the session thus terminating the communication [50, 61]. When a SIP proxy server is overloaded, it will send a "503 Service Unavailable" message in response to an "Invite" message. The call will then be rejected. Processing SIP Invite messages and generating 503 Service Unavailable messages still consume a large amount of CPU time. Numerous experiment results (e.g., [23, 42, 52, 53]) have demonstrated that SIP

built-in overload control mechanism based on the 503 Service Unavailable messages cannot prevent overload collapse in a SIP network.

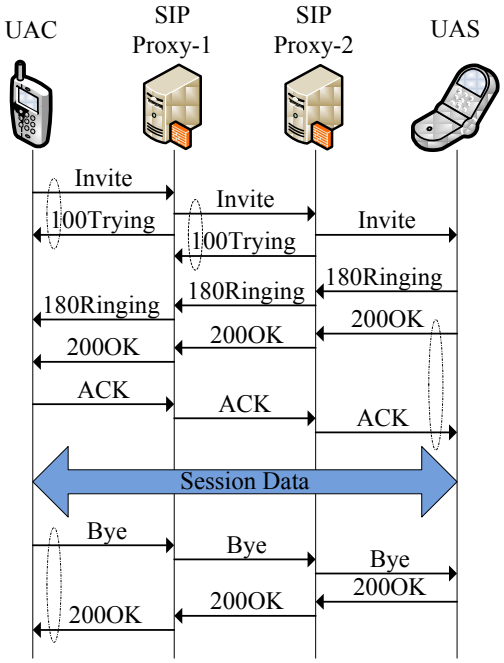


Fig. 2. A typical procedure of session establishment.

SIP introduces a retransmission mechanism to maintain its reliability [19, 27, 50]. 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.

SIP RFC 3261 [50] suggests that the SIP retransmission mechanism should be disabled for hop-by-hop transaction when running SIP over Transmission Control Protocol (TCP) to avoid redundant retransmissions at both SIP and TCP layer [50]. However, recent experimental evaluation on SIP-over-TCP overload behaviour in [52] demonstrates that TCP flow control mechanism cannot prevent SIP overload collapse for time-critical session-based applications due to lack of application context awareness at the transport layer. Other experiments (e.g., [23, 42]) also indicate that the throughput with SIP-over-TCP exhibits similar overload collapse behaviour as that with SIP-over-UDP. Furthermore, IBM research center claims that using TCP to deliver SIP messages degrades server performance from 43% (under stateful

proxy with authentication) to 65% (under stateless proxy without authentication) when compared with using UDP [41]. Therefore most vendors choose to run SIP over UDP instead of TCP [17].

SIP has two types of message retransmission:

- Hop-by-hop retransmission: a sender starts the first retransmission of an original message at T_1 seconds, the time interval doubling after every retransmission (exponential backoff), if the corresponding reply message is not received. The last retransmission is sent out at the maximum time interval 64xT_1 seconds. Default value of T_1 is 0.5s, thus there is a maximum of 6 retransmissions. The hop-by-hop "Invite"–"Trying" transaction shown in Fig. 1 follows this rule [50];
- End-to-end retransmission: a sender starts the first retransmission of an original message at T_1 seconds, the time interval doubling after every retransmission but capping off at T_2 seconds, if the corresponding reply message is not received. The last retransmission is sent out at the maximum time interval 64xT_1 seconds. Default value of T_2 is 4s, thus there is a maximum of 10 retransmissions. The end-to-end "OK"–"ACK" and "Bye"–"OK" transactions shown in Fig. 1 follow this rule [50].

As the Invite message is the most complex message to be processed by a SIP server and thus the major CPU load contributor [50, 53], we will focus on the Invite-100Trying transaction and ignore other non-Invite transactions in this paper. Given the proportionate nature and the general similarity of the retransmission mechanisms between the "Invite" and "non-Invite" messages in a typical session [50], our two overload control algorithms are applicable for mitigating the overload caused by non-Invite transactions.

III. RELATED WORK

Performance and reliability of a SIP network have been studied in [21, 54]. Experimental evaluation of SIP servers in [29, 34, 41] showed the overload collapse behaviour. The overload may spread in a network of SIP servers and bring a potential SIP network collapse [23, 32], therefore, different types of

strategies have been proposed to address SIP server overload problem (e.g., [2, 3, 11, 12, 16, 18, 20, 22, 23, 26, 36, 37, 39, 40, 42, 43, 45, 46, 48, 52, 53, 56, 58, 59, 60]).

A. Load Balancing

As a default part of numerous operating systems, SIP Express Router (SER) developed a load balancing module to mitigate the overload caused by large subscriber populations or abnormal operational conditions [36]. In order to balance the demands for bandwidth and the call failure rates of a SIP proxy, request batching was combined with parallel execution to improve call throughput and reduce call failure rate significantly in [11]. Three approaches were introduced for load balancing in cluster-based SIP servers [37]. The load balancer performed session-aware request assignment to route SIP transactions to the proper back-end nodes [37].

In an IMS core network, a load balancing scheme was proposed in [59] to re-direct consequent SIP traffic from the over-utilized Serving Call/Session Control Function (S-CSCF) server to the other under-utilized ones. This can effectively drive the traffic to avoid further overload at potentially over-utilized S-CSCF while not inducing severe extra load [59]. By providing efficient SIP message routing in the high volume Interrogating CSCF (I-CSCF) servers, a SIP message overload transfer scheme can leverage redundant servers to reduce the message disruption in cases of server failures [18]. A user equipment registration scheme not only balanced the workload over multiple Proxy-CSCF (P-CSCF) nodes, .but also reduced the required P-CSCF nodes up to 40% from the standard session initialization procedure of IMS [39].

However, load balancing tries to avoid SIP network failures by reducing the utilization of those servers that may become overloaded. When the total message arrival rate exceeds the aggregated processing capacity of all local servers, load balancing schemes cannot prevent the overload collapse.

B. Priority-Based Overload Control

A priority enqueuing scheme provided differentiate service for different types of SIP messages in every SIP proxy server, where “INVITE” messages were placed into low priority queue [46]. Once the proxy server was overloaded, every “INVITE” message was hardly forwarded to its destination, thus forbidding the successive non-INVITE transactions to reduce the traffic load [46]. Another queuing strategy applied a queue threshold for the “INVITE” message queue to detect the overload in [16]. When the overload was anticipated, part of “INVITE” messages would be rejected to mitigate the overload by blocking the corresponding calls [16]. Priority queue was used to overcome the overload problem of IMS system by blocking non-priority calls (e.g., [3, 48]). Similar to the priority scheme, a novel authentication protocol was developed to reduce the load on the centralized authentication database dramatically and improve the overall security of a carrier-scale VoIP network in [12].

C. Push-Back Overload Control

Since the CPU cost of rejecting a session is usually comparable to the CPU cost of serving a session [53], cancelling “INVITE” transaction using priority queuing scheme is not very cost-effective. Therefore, numerous push-back solutions have been proposed to reduce the traffic load of an overloaded receiving server by advertising its upstream sending servers to decrease their sending rates.

Both centralized and distributed overload control mechanisms for SIP were developed in [23]. Retry-after control, processor occupancy control, queue delay control and window based control were proposed to improve goodput and prevent overload collapse in [43]. Three window-based feedback algorithms were proposed to adjust the message sending rate of the upstream SIP servers based on the queue length [53]. Other push-back solutions can be found in [2, 26, 40, 45, 52, 56, 58, 60].

Such pushback control solutions aim at preventing the overload of a server by reducing the sending rate of its upstream servers. This would increase the queuing delays of newly arrival original messages at the upstream servers, which in turn cause overload at the upstream servers. Overload may thus propagate server-by-server to sources. Unlike a source in TCP typically generates large amount of data, a UA in SIP

only generates very few signalling messages [23]. This leads to rejections of a large number of calls which means revenue loss for carriers. However, it may be unnecessary to reject calls when temporary overload only lasts a short period of time.

D. Proposal of Retransmission-Based Overload Control

Our approach proposed in this paper is different from the existing solutions discussed above. When retransmissions are caused by the overload rather than the message loss, they will bring extra overhead instead of reliability to the network and exacerbate the overload as we will demonstrate later through our analytical modeling and simulation. Therefore, when a short-term overload occurs at a downstream server, we propose to reduce the retransmission rate instead of reducing original message sending rate of its upstream servers.

Such retransmission-based solution will mitigate the overload while maintaining original message sending rate, which leads to less blocking calls and more revenue for carriers [28, 30, 31].

Changing the SIP header requires a time-consuming standardization process. Therefore, for the ease of industrial implementation, our control algorithms locate the controller at each upstream server and request each upstream server to guess the overload at its downstream servers implicitly without any modification and extension to the SIP header. This is quite different from most existing push-back solutions which require some modification to the SIP header.

When an overload lasts for a long period, our retransmission-based solution can be combined with pushback solution to reject some calls by reducing the original message rates of SIP sources. Under this kind of situation, our solution can help maximize server utilization and therefore serve as a complementary mechanism to the existing solutions – On the other hand, RTDC algorithm proposed in this paper can also be used to reduce both retransmission rate and original message rate for mitigating a long-term overload, if SIP service providers are reluctant to modify the SIP header due to the time-consuming standardization process and expensive implementation cost.

IV. QUEUING DYNAMICS OF OVERLOADED SERVER

In order to provide effective overload control solutions, it is necessary to understand the impact of retransmission mechanism on the SIP overload. To achieve this goal, we need to analyze the queuing dynamics of an overloaded server and its upstream server when a transient overload occurs.

The network topology of a real SIP system can be quite complex. Fig. 3 depicts a typical SIP network topology [28, 29, 30, 31, 53]. 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 queuing delay caused by the overload, which may trigger premature retransmissions.

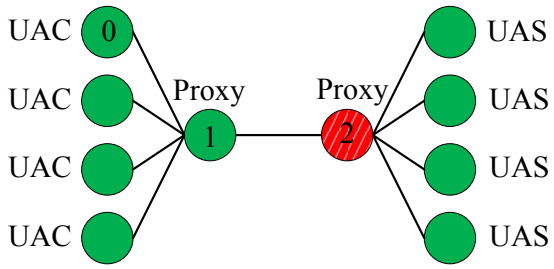


Fig. 3. SIP network topology with an overloaded downstream receiving Server 2 (which is marked with diagonal lines) and its upstream sending Server 1.

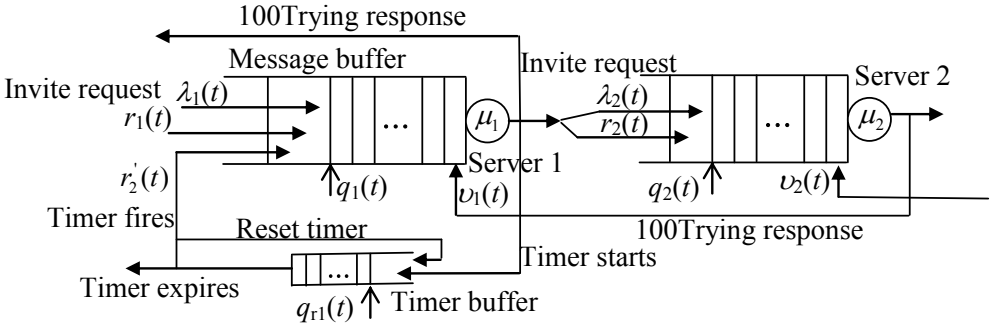


Fig. 4. Queuing dynamics of an overloaded server and its upstream server.

Fig. 4 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 [23, 53]. 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.

Like Eq. (1), we can obtain the 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 aggregated arrival rate of original requests, which can be arbitrary stochastic process (e.g., Poisson, Pareto, Gamma or Lognormal distribution [13, 55]); $r_1(t)$ 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.

Note that the mean arrival rate $\overline{\lambda_1}$ is equal to the inverse of the mean message inter-arrival time $\overline{\psi_1}$, i.e., $\overline{\lambda_1} = 1/\overline{\psi_1}$. When the inter-arrival time of requests is used to describe the characteristics of a queuing process, a probability is typically used to express the relationship between the inter-arrival time and the arrival rate, i.e., by picking an arbitrary message arrival time as the starting point, and letting ψ_1 be the time until the next message arrival, we can obtain a probability $P(\psi_1 > t) = e^{-\overline{\lambda_1} t}$, where $\overline{\lambda_1}$ is the mean arrival rate. Such probability describes the statistical nature of a stable queuing process, but cannot reflect dynamical behaviour of an unstable SIP queuing process in case of an overload. In order to demonstrate the root cause of the overload collapse, we describe the dynamical behaviour of a SIP queuing process in the continuous-time domain using Eqs. (1) and (2).

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 bring the overload to Server 1. After a 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.

In summary, we use Eqs. (1) and (2) to show how an overloaded downstream Server 2 propagates its overload to its upstream Server 1 under SIP retransmission mechanism. When the message arrival rate exceeds the message processing capacity at a SIP server, overload occurs and the queue builds up, which may result in a long queuing delay and trigger unnecessary message retransmissions from its upstream servers. The redundant retransmissions increase the CPU loads of both the overloaded server and its upstream servers. Such overload propagation may bring potential SIP network collapse.

V. PI CONTROLLER FOR MITIGATING SIP OVERLOAD

Through the queuing analysis in the last section, we have demonstrated the impact of retransmissions on the SIP overload, that is, the unnecessary retransmitted messages $r'_2(t)$ triggered by the queuing delay would increase queue sizes at both Server 1 and Server 2, and thus bring the overload to both servers. Therefore, we believe that reducing the retransmission rate $r'_2(t)$ can prevent the queue sizes at both Server 1 and Server 2 from increasing continuously, thus mitigating the overload.

To achieve our goal of mitigating the SIP overload, we propose two PI control algorithms (i.e., Redundant Retransmission Ratio Control algorithm and Round Trip Delay Control algorithm) to reduce

unnecessary retransmissions based on two different overload indicators (i.e., redundant retransmission ratio and round trip delay).

The PID controllers are not necessarily optimal, and it is estimated that present sophisticated controllers would not have more economic effect in 80% of control loops [9]. Furthermore, the derivative action would amplify the noise and may deteriorate the system performance, therefore the industry widely employs PI controller instead of PID controller in the system design [5, 7, 14]. In addition, the stochastic nature of SIP traffic [13, 55] would make PI controller exhibit better performance than PID controller, thus we apply PI controller in SIP overload control.

A. Redundant Retransmission Ratio Control Algorithm

Redundant Retransmission Ratio Control (RRRC) algorithm aims at mitigating the overload by clamping the redundant retransmission ratio around a target value.

1) Overload Control Plant: 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 retransmitted messages $r'_2(t)$ are redundant retransmitted messages $r'_{2r}(t)$, i.e., $r'_{2r}(t) \approx r'_{2r}(t)$ [53]. After a round trip delay τ, these redundant retransmitted messages $r'_{2r}(t)$ are acknowledged as the response messages $\upsilon_{1r}(t)$, i.e., $\upsilon_{1r}(t)=r'_{2r}(t-\tau) \approx r'_2(t-\tau)$.

We define the redundant retransmission 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 $N_{\upsilon Ir}$ of the arrival redundant response messages and the number $N_{\upsilon I}$ of the total arrival response messages during a sampling time interval T_s, then we can obtain υ_{1r} and υ_1 as $\upsilon_{Ir}=N_{\upsilon Ir}/T_s$ and $\upsilon_I=N_{\upsilon I}/T_s$. Considering average 10% packet loss in the IP layer

[35], it is necessary to maintain a target redundant retransmission ratio γ_0 for message loss recovery. Our numerous experiments suggest that γ_0=0.1 is a good choice.

We assume that the system is locally stable and therefore the uncontrolled variables τ and $\upsilon_1(t)$ are constant around an operating point. The transfer function between the instantaneous redundant retransmission ratio $\gamma(t)$ and the retransmission rate $r'_2(t)$ is given by

$$P(s)=\gamma(s)/r'_2(s)= \mathcal{L}\{\gamma(t)\}/\mathcal{L}\{r'_2(t)\}=[r'_2(s)e^{-s\tau}/\upsilon_1]/r'_2(s)=e^{-s\tau}/\upsilon_1. \quad (4)$$

Fig. 5 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 located at the upstream server for mitigating the overload to achieve a desirable target redundant retransmission ratio γ_0, when the overload is anticipated at the downstream server.

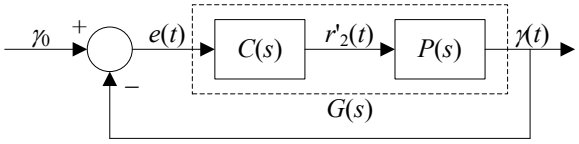


Fig. 5. Feedback SIP overload control system with target redundant retransmission ratio as system input.

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

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

where K_P and K_I denote the proportional gain and integral gain of the PI controller at the upstream server. The goal of PI controller is to clamp the system output (i.e., instantaneous redundant retransmission ratio $\gamma(t)$) around a target value (i.e., target redundant retransmission ratio γ_0) [44]. When the overload triggers excessive redundant retransmissions and makes redundant retransmission ratio $\gamma(t)$ exceed γ_0 continuously, PI controller indicated by Eq. (5) will reduce retransmission rate and clamp $\gamma(t)$ back to γ_0. 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 retransmission 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

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

Nyquist Stability Theorem demonstrates that a positive phase margin ($\phi_m>0$) can guarantee the stability of a control system [44]. A common control engineering practice suggests an interval of phase margin as $30^0 \le \phi_m \le 60^0$ for a good response [24]. From the definition on the phase margin ϕ_m of $G(s)$ [44], we can obtain

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

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

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

$$\omega_g\tau=3\pi/4-\phi_m, \quad (10)$$

$$\sqrt{2}K_I/(\omega_g\upsilon_1)=1. \quad (11)$$

Using the relationship $K_P=K_I/\omega_g$, we can obtain K_P and K_I from Eqs. (10) and (11) as

$$K_P=\upsilon_1/\sqrt{2}, \quad (12)$$

$$K_I=\upsilon_1(3\pi/4-\phi_m)/(\sqrt{2}\tau). \quad (13)$$

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. Theorems 1 and 2 show the impact of the two SIP network parameters on the phase margin or the system stability.

Theorem 1: ***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, $\phi'_m<\phi_m$).

Proof. From the Eqs. (12) and (13), we obtain the PI rate controller $C(s)=K_P+K_I/s$ for the SIP overload control system based on the round trip delay τ.

For the SIP overload control system with the round trip delay τ, the open-loop transfer function is $(K_P+K_I/s)e^{-s\tau}/\upsilon_1$. Then from the definition of the phase margin [44], we can obtain Eqs. (8) and (9). To make the comparison between the phase margins ϕ'_m and ϕ_m, we simplify Eq. (8) as

$$\phi_m = \frac{\pi}{2} + \arctan\left[\frac{K_P\omega_g}{K_I}\right] - \omega_g\tau . \tag{14}$$

Then we can obtain the gain crossover frequency ω_g of the SIP overload control system, which is the smallest positive real root of Eq. (9), i.e.,

$$\omega_g = \frac{K_I}{\sqrt{\upsilon_1^2 - K_P^2}} . \tag{15}$$

Similarly, for the SIP overload control system with round trip delay τ', we can obtain its phase margin as

$$\phi'_m = \frac{\pi}{2} + \arctan\left[\frac{K_P\omega'_g}{K_I}\right] - \omega'_g\tau' , \tag{16}$$

$$\frac{\sqrt{K_P^2\omega_g'^2 + K_I^2}}{\omega'_g\upsilon_1} = 1 . \tag{17}$$

We can also obtain the gain crossover frequency ω'_g, which is the smallest positive real root of Eq. (17), i.e.,

$$\omega_g^{'} = \frac{K_I}{\sqrt{\upsilon_1^2 - K_P^2}}. \tag{18}$$

Eqs. (15) and (18) indicate that $\omega'_g=\omega_g$. Therefore, based on Eqs. (14) and (16), we can obtain

$$\phi_m^{'} - \phi_m = \omega_g \tau - \omega_g^{'} \tau^{'} = \omega_g \left(\tau - \tau^{'}\right) < 0 .\square \tag{19}$$

Theorem 2: ***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, $\phi'_m<\phi_m$).

Proof. As discussed in Theorem 1, for the SIP overload control system with the response message rate υ_1, from the definition of the phase margin [44], we can obtain Eqs. (9) and (14). Then we can obtain the gain crossover frequency ω_g of the SIP overload control system, as described by Eq. (15).

From the definition of the gain margin [44], we can obtain

$$-\frac{\pi}{2} + \arctan\left(\frac{K_P \omega_p}{K_I}\right) - \omega_p \tau = -\pi , \tag{20}$$

where ω_p is the phase crossover frequency of the SIP overload control system.

Similarly, for the SIP overload control system with the response message rate υ'_1, we can obtain its phase margin as

$$\phi_m^{'} = \frac{\pi}{2} + \arctan\left[\frac{K_P \omega_g^{'}}{K_I}\right] - \omega_g^{'} \tau , \tag{21}$$

$$\frac{\sqrt{K_P^2 \omega_g^{'2} + K_I^2}}{\omega_g^{'} \upsilon_1^{'}} = 1 . \tag{22}$$

We can also obtain the gain crossover frequency ω'_g, which is the smallest positive real root of Eq. (22), i.e.,

$$\omega_g^{'} = \frac{K_I}{\sqrt{\upsilon_1^{'2} - K_P^2}}. \tag{23}$$

From the definition of the gain margin [44], we can also obtain

$$-\frac{\pi}{2} + \arctan\left(\frac{K_P \omega_p^{'}}{K_I}\right) - \omega_p^{'}\tau = -\pi. \tag{24}$$

From Eqs. (20) and (24), we can find that $\omega'_p=\omega_p$.

By drawing the Bode plot [44] of the open-loop transfer function $G(s)=C(s)P(s)=(K_P+K_I/s)e^{-s\tau}/\upsilon_1$, we can find that the phase is a decreasing function of frequency between the gain crossover frequency ω_g and the phase crossover frequency ω_p, as described by the following useful Relationship 1.

Relationship 1: ***If*** there exists the frequencies $\omega_g \le \omega_1 < \omega_2 \le \omega_p$, ***then*** we will have

$-\pi=\arg(G(j\omega_p)) \le \arg(G(j\omega_2)) < \arg(G(j\omega_1)) \le \arg(G(j\omega_g)) = -\pi+\phi_m$, and vice versa.

In order to make the comparison between the phase margins ϕ'_m and ϕ_m, we let $\omega_2=\omega'_g<\omega'_p=\omega_p$ and $\omega_1=\omega_g$ as two different frequency points. Then from Equations (15) and (23), we can show that $\omega_g=\omega_1<\omega_2=\omega'_g<\omega'_p=\omega_p$. Applying the Relationship 1, we can obtain

$-\pi=\arg(G(j\omega_p))<\arg(G(j\omega'_g))=\arg(G(j\omega_2)) <\arg(G(j\omega_1))=\arg(G(j\omega_g))$,

$-\pi+\phi'_m=\arg(G(j\omega'_g))<\arg(G(j\omega_g))=-\pi+\phi_m$.

That is, $\phi'_m<\phi_m$.□ (25)

To maintain the stability of the system, we self-tune PI controller when dramatic change of system parameters (υ_1 and τ) drifts the heuristic parameter $\zeta=\tau/\upsilon_1$ out of its interval. Numerous simulation results suggest $0.5\zeta_0 \le \zeta \le 1.5\zeta_0$ is a good interval for monitoring ζ.

We use the round trip delay τ to detect the overload, thus determining whether to activate PI control algorithm. When the overload causes the round trip delay τ longer than the 1st retransmission timer T_1, the unnecessary retransmissions would be triggered. Therefore, we let the 1st retransmission timer T_1 be the threshold set for the indicator of overloading condition: 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. 6.

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 retransmission ratio γ_0 and phase margin ϕ_m; Set the initial values for τ, υ_1 and ζ; Obtain PI controller gains using Eqs. (12) and (13).

(2) Measure τ, υ_1, υ_{1r}, and calculate 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 Eqs. (12) and (13) 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
υ_{1r} : Redundant response message rate
γ : Redundant retransmission ratio $\gamma=\upsilon_{1r}/\upsilon_1$
K_P : Proportional gain of PI controller
K_I : Integral gain of PI controller
r'_2 : Message retransmission rate
ζ : Monitoring parameter $\zeta=\tau/\upsilon_1$

Fixed parameter:

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

Fig. 6. Redundant retransmission ratio control algorithm for mitigating SIP overload.

According to SIP RFC3261 [50], SIP header contains a "Timestamp" field for estimating the round trip delay τ. When the upstream Server 1 sends every request to its downstream Server 2, it records the

sending time t_1 into the "Timestamp" field of each request. After receiving every corresponding response message, Server 1 checks the "Timestamp" field, and calculates the difference between the response receiving time t_2 and the request sending the time t_1, i.e., (t_2-t_1). Then we can use the moving average method to obtain the round trip delay τ as $\tau=(1-w_\tau)\tau+w_\tau(t_2-t_1)$, where w_τ is the averaging weight. Based on numerous experiment results, we recommend a good choice of w_τ as $w_\tau=1/C$, and C is the server service capacity.

B. Round Trip Delay Control Algorithm

Round Trip Delay Control (RTDC) algorithm aims at mitigating the overload by clamping the round trip delay below a target value.

1) Overload Control Plant: In order to present our overload control mechanism more clearly, we assume that the upstream Server 1 can process all the messages without any delay before the overload is propagated from its downstream Server 2, i.e., $\lambda_2(t)=\lambda_1(t)$ and $r_2(t)=r'_2(t)$. Therefore, we can update the queuing dynamics for the message queue of Server 2 as

$$\dot{q}_2(t)=\lambda_1(t)+r_2'(t)+\upsilon_2(t)-\mu_2(t). \tag{26}$$

Then the corresponding message queuing delay of Server 2 becomes

$$\dot{\tau}_2(t)=[r_2'(t)+\lambda_1(t)+\upsilon_2(t)-\mu_2(t)]/\mu_2(t). \tag{27}$$

Each request message corresponds to a response message, and the time to process a response message is typically much smaller than a request message [50, 53]. Thus we can use the request message service rate (i.e., the response message rate $\upsilon_1(t)$) to approximate the total service rate $\mu_2(t)$. Then we can approximate the queuing delay of Server 2 as

$$\dot{\tau}_2(t)\approx[r_2'(t)+\lambda_1(t)+\upsilon_2(t)-\upsilon_1(t)]/\upsilon_1(t), \tag{28}$$

We assume that the system is locally stable and therefore the uncontrolled variables $\lambda_1(t)$, $\upsilon_2(t)$ and $\upsilon_1(t)$ are constant around an operating point. The transfer function between the instantaneous queuing delay $\tau_2(t)$ and the retransmission rate $r'_2(t)$ can be approximated as

$$P(s)=\frac{\tau_2(s)}{r_2'(s)}=\frac{1}{\upsilon_1 s}+\frac{\lambda_1/s+\upsilon_2/s-\upsilon_1/s}{r_2'(s)\upsilon_1 s}. \tag{29}$$

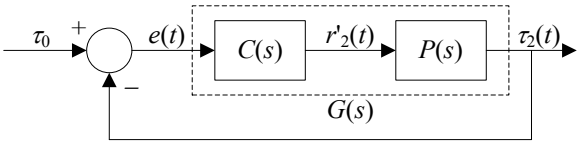


Fig. 7. Feedback SIP overload control system with target round trip delay as system input.

Since the signaling messages are typically CPU capacity constrained rather than bandwidth constrained, for the round trip delay between an upstream server and its overloaded downstream server, the queuing delay is dominant, while transmission and propagation delay are negligible [53]. Therefore, the queuing delay τ_2 of the overloaded downstream Server 2 can approximate the round trip delay between the upstream Server 1 and the downstream Server 2, when the overload happens.

Fig. 7 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 rate controller $C(s)$ is located at the upstream server for mitigating the overload to clamp the round trip delay of the upstream server (i.e., the approximated queuing delay of the overloaded downstream server) below a target value, when the overload is anticipated at its downstream server.

2) The PI Rate Controller Design: Based on the instantaneous round trip delay of the upstream server (which approximates the queuing delay of the overloaded downstream server) $\tau_2(t)$, the retransmission rate $r'_2(t)$ can be obtained by the following PI control algorithm expressed via

$$r_2'(t)=K_P e(t)+K_I\int_0^t e(\eta)\mathrm{d}\eta=K_P(\tau_0-\tau_2(t))+K_I\int_0^t(\tau_0-\tau_2(\eta))\mathrm{d}\eta. \tag{30}$$

where K_P and K_I denote the proportional gain and integral gain of the PI controller located at the upstream server, and τ_0 denotes the target round trip delay. Since the uncontrolled original message rate is maintained to keep the revenue and achieve the user satisfaction in case of short-term overload, PI controller aims at clamping the round trip delay of the upstream server below a desirable target delay τ_0

rather than reaching the target delay τ_0. When the overload makes the round trip delay $\tau_2(t)$ exceed τ_0 continuously, PI controller indicated by Eq. (30) will reduce retransmission rate and clamp $\tau_2(t)$ back to τ_0.

It can be easy to obtain the transfer function between the retransmission rate $r'_2(t)$ and the round trip delay deviation $e(t)$ as

$$C(s)=K_P+K_I/s. \quad (31)$$

Since it is not easy to design the PI controller based on Eq. (29) directly using Nyquist stability criterion, the second term in Eq. (26) is dropped/truncated so that the control plant can be approximated as

$$P(s)=\tau_2(s)/r'_2(s)\approx 1/(\upsilon_1 s). \quad (32)$$

Such control plant approximation has been widely adopted in industrial process control system design, e.g., [4, 6, 8, 24, 25], which has recently found its application in network traffic control, e.g., [38]. The validity of this approximation has also been confirmed by Theorem 3 and verified by our performance evaluation in Section VI later on.

Theorem 3: The truncated component of the overload control plant $P(s)$ described by Eq. (29) only has an impact on the zeros of the closed-loop overload control system, and thus will not reduce the system stability margin.

Proof. From the viewpoint of the control theory and practice, the stability of a closed-loop feedback control system is determined by its poles which are the roots of its characteristic equation [44]. Therefore, we needs to prove that the characteristic equation of the close-loop control system with the truncated control plant $P(s)$ described by Eq. (32) is the same as that of the close-loop control system with the non-truncated control plant $P(s)$ described by Eq. (29).

Based on the truncated control plant $P(s)$ in Eq. (32), we can obtain the open-loop transfer function of the overload control system as

$$G(s)=C(s)P(s)=(K_P+K_I/s)/(\upsilon_1 s), \quad (33)$$

which leads to the close-loop transfer function with truncated control plant as

$$G_{c1}(s)=\frac{G(\mathrm{s})}{1+G(s)}=\frac{\left(K_P+K_I/s\right)/\upsilon_1 s}{1+\left(K_P+K_I/s\right)/\upsilon_1 s}=\frac{K_P s+K_I}{\upsilon_1 \mathrm{s}^2+K_P s+K_I}. \tag{34}$$

By substituting (30) into (27), we can obtain the dynamic of the closed-loop overload control system with non-truncated control plant in (29) as

$$\dot{\tau}_2(t)\approx[K_P(\tau_0-\tau_2(t))+K_I\int_0^t(\tau_0-\tau_2(\eta))\mathrm{d}\eta+\lambda_1(t)+\upsilon_2(t)-\upsilon_1(t)]/\upsilon_1(t). \tag{35}$$

Considering the assumption that the uncontrolled variables $\lambda_1(t)$, $\upsilon_2(t)$ and $\upsilon_1(t)$ are constant around an operating point, we can obtain the Laplace transform of round trip delay as

$$s\tau_2(s)=\left[\left(\mathrm{K_P}+\frac{\mathrm{K_I}}{s}\right)\left(\frac{\tau_0}{s}-\tau_2(s)\right)+\frac{\lambda_1+\upsilon_2-\upsilon_1}{s}\right]\frac{1}{\upsilon_1}.$$

Thus we can obtain the closed-loop transfer function of overload control system with non-truncated plant as

$$G_{c2}(s)=\frac{\tau_2(\mathrm{s})}{\tau_0/s}=\frac{\left(K_P+K_I/s\right)+(\lambda_1+\upsilon_2-\upsilon_1)/\tau_0}{\upsilon_1 \mathrm{s}+\left(K_P+K_I/s\right)}=\frac{K_P s+K_I+(\lambda_1+\upsilon_2-\upsilon_1)s/\tau_0}{\upsilon_1 \mathrm{s}^2+K_P s+K_I}. \tag{36}$$

Comparing Eq. (34) and Eq. (36), we can find that the characteristic equations of both closed-loop overload control system $G_{c1}(s)$ and $G_{c2}(s)$ are the same, i.e.,

$$s^2+K_P s/\upsilon_1+K_I/\upsilon_1=0. \tag{37}$$

In this regard, the truncated component $(\lambda_1+\upsilon_2-\upsilon_1)$ only has an impact on the zeros of the closed-loop overload control system, and thus will not reduce the system stability margin.□

From the definition on the phase margin ϕ_m of $G(s)$ described by Eq. (33) [44], we can obtain

$$\arctan\left[\frac{K_P\omega_g}{K_I}\right]-\frac{\pi}{2}-\frac{\pi}{2}=-\pi+\phi_m, \tag{38}$$

$$\frac{\sqrt{K_P^2\omega_g^2+K_I^2}}{\omega_g}\frac{1}{\upsilon_1\omega_g}=\frac{\sqrt{K_P^2\omega_g^2+K_I^2}}{\upsilon_1\omega_g^2}=1, \tag{39}$$

where ω_g is the gain crossover frequency of the overload control system. To simplify our controller design, we set ω_g=1 based on numerous simulation results. Thus we can obtain K_P and K_I as

$$K_P=\frac{\upsilon_1\tan(\phi_m)}{\sqrt{1+\tan^2(\phi_m)}}, \tag{40}$$

$$K_I = \frac{\upsilon_1}{\sqrt{1+\tan^2(\phi_m)}}. \tag{41}$$

So far we have assumed υ_1 be constant. In reality, this is not necessarily true. If the PI controller parameters K_P and K_I remain unchanged, the varying service rate of the overloaded server may drive the phase margin out of its desirable interval. Theorem 4 shows the impact of the response message rate on the phase margin of the overload control system.

Theorem 4: ***If*** current response message rate υ'_1 is larger 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, $\phi'_m<\phi_m$).

Proof. From the Eqs. (40) and (41), we obtain the PI rate controller $C(s)=K_P+K_I/s$ for the SIP overload control system based on the response message rate υ_1.

Then from the definition of the phase margin [44], we can obtain Eqs. (38) and (39). To make the comparison between the phase margins ϕ'_m and ϕ_m, we simplify Eq. (38) as

$$\phi_m = \arctan\left[\frac{K_P\omega_g}{K_I}\right]. \tag{42}$$

Then we can obtain the gain crossover frequency ω_g of the SIP overload control system, which is the smallest positive real root of Eq. (39), i.e.,

$$\omega_g = \sqrt{\frac{K_P^2}{2\upsilon_1^2}+\frac{1}{2}\sqrt{\frac{K_P^4}{\upsilon_1^4}+\frac{4K_I^4}{\upsilon_1^2}}}\ . \tag{43}$$

Similarly, for the SIP overload control system with the response message rate υ'_1, we can obtain its phase margin as

$$\phi'_m = \arctan\left[\frac{K_P\omega'_g}{K_I}\right], \tag{44}$$

$$\frac{\sqrt{K_P^2\omega_g'^2+K_I^2}}{\upsilon_1\omega_g'^2}=1, \tag{45}$$

We can also obtain the gain crossover frequency ω'_g, which is the smallest positive real root of Eq. (45), i.e.,

$$\omega'_g = \sqrt{\frac{K_P^2}{2\upsilon_1'^2} + \frac{1}{2}\sqrt{\frac{K_P^4}{\upsilon_1'^4} + \frac{4K_I^4}{\upsilon_1'^2}}} \ . \tag{46}$$

Eqs. (43) and (46) indicate that $\omega'_g<\omega_g$. Therefore, based on Eqs. (42) and (44), we can obtain

$$\phi'_m - \phi_m = \arctan\left[\frac{K_P\omega'_g}{K_I}\right] - \arctan\left[\frac{K_P\omega_g}{K_I}\right] < 0 \ .\square \tag{47}$$

To achieve a satisfactory performance, we self-tune PI controller when dramatic change of message response rate υ_1 exceeds a specified interval. Numerous simulation results suggest $0.5\upsilon_{10}\le\upsilon_1\le1.5\upsilon_{10}$ is a good interval for monitoring υ_1. Summary of our overload control algorithm is shown in Fig. 8.

Overload Control Algorithm

When each retransmission timer fires or expires
Retransmit the message with a retransmission probability corresponding to a retransmission rate r'_2 calculated by a PI rate controller

Adaptive PI rate control algorithm:

(1) Specify target round trip delay τ_0 and phase margin ϕ_m; Set the initial values for υ_1; Obtain PI controller gains using Eqs. (40) and (41).

(2) Measure τ_2 and υ_1 upon response message arrivals.

(3) If $\upsilon_1>1.5\upsilon_{10}$ or $\upsilon_1<0.5\upsilon_{10}$, self-tune PI controller gains using Eqs. (40) and (41), then update $\upsilon_{10}=\upsilon_1$; Otherwise, PI controller remains unchanged.

(4) Calculate the retransmission rate r'_2 using Eq. (30); Go to Step (2).

Varying parameter:

τ_2 : Instantaneous round trip delay
υ_1 : Response message rate
K_P : Proportional gain of PI controller
K_I : Integral gain of PI controller
r'_2 : Message retransmission rate

Fixed parameter:

τ_0 : Target round trip delay
ϕ_m : Phase margin

Fig. 8. Round trip delay control algorithm for mitigating SIP overload.

VI. PERFORMANCE EVALUATION AND SIMULATION

To verify and compare our two overload control algorithms, we conducted OPNET® [47] event-driven simulations to observe the transient behaviour of the overloaded server and its upstream server. Event-driven simulations have been widely adopted by most researchers (e.g., [23, 42, 43, 53]). During our experiment, four user agent clients generated original request messages with equal rate, and then sent them to four user agent servers via two proxy servers, as shown in Fig. 3. No matter what kind of random distribution for message generation rate and service time, SIP overload means that the total message arrival rate exceeds the service capacity of a SIP server [43, 53]. Currently there is no measurement result for the workload in the real SIP networks. Poisson distributed message arrival rate and exponentially distributed service time are widely adopted by most existing research work (e.g., [23]). Therefore, in this paper, the message generation rates are Poisson distributed. The message service time of each server follows exponential distribution.

Since processing a response message takes much less time than processing a request message, we set the ratio α of the mean processing time of a response message to that of a request message as 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 phase margin is set as 45^0. Since Internet Traffic Report indicates that current global packet loss statistics averaged 10% packet loss [35], average message loss probability σ is set as 10%. For RRRC algorithm, the target redundant retransmission ratio γ_0 is set as 0.1; For RTDC algorithm, the target round trip delay τ_0 is set as 0.5s.

To demonstrate the effectiveness of our overload control solution, 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 [50]. 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 "NOLC" to indicate that overload control algorithm was not applied to all servers in a SIP network, while using "RRRC" or "RTDC" to denote the specific overload control algorithm which was applied to all servers in a SIP network.

A. Overload at Server 1

In this scenario, the mean message generation rate for each user agent client 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.

Figs. 9 and 10 show the dynamic behaviour of overloaded Server 1 and one of its upstream originating servers. The queuing delay is approximately equal to the queue size divided by the server capacity, i.e., $\tau_{q1} \approx q_1/C_1$ and $\tau_{q2} \approx q_2/C_2$. A larger queue size corresponds to a longer queuing delay. Therefore, we did not show the queuing delay (i.e., the approximated round trip delay) here.

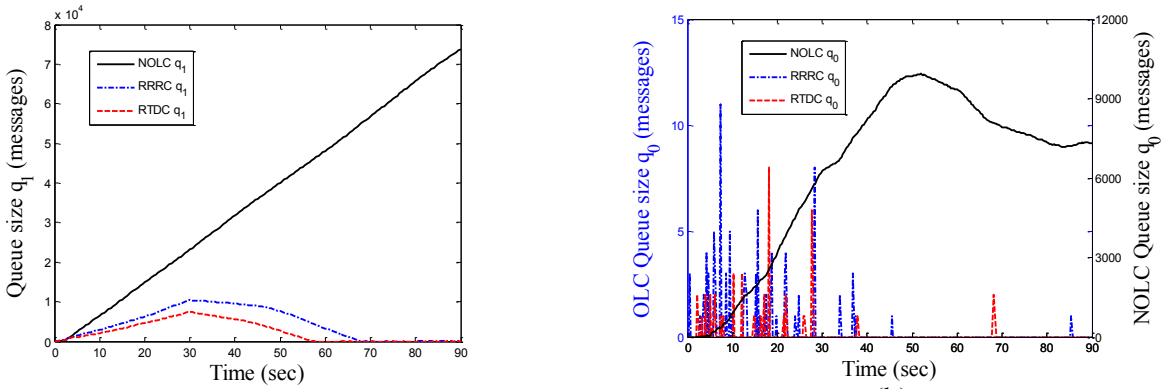
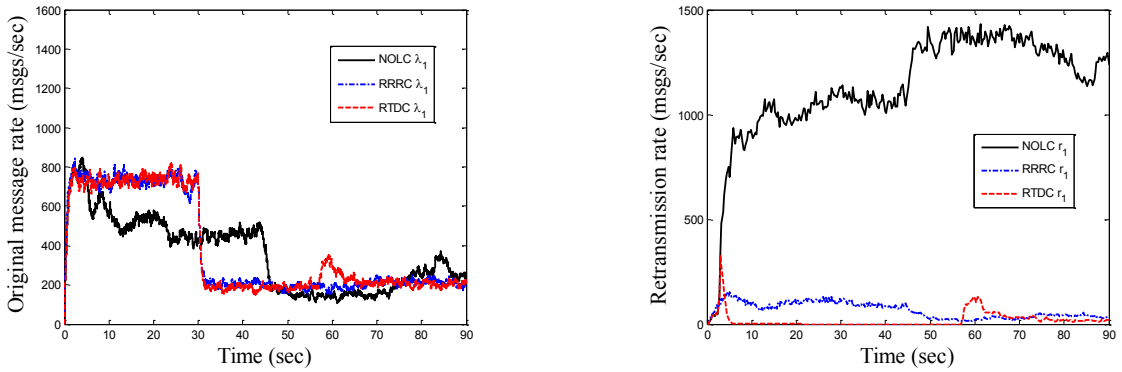


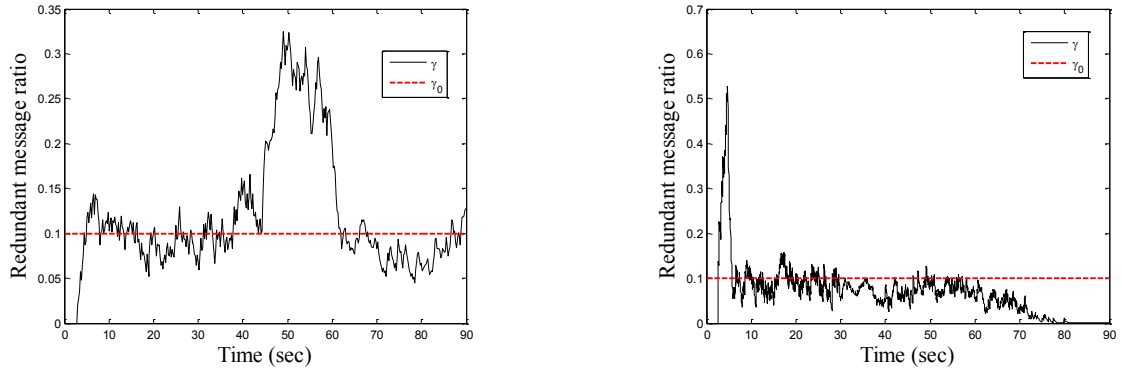
Fig. 9. (a) Queue size q_1 (messages) of Server 1 versus time. (b) Queue size q_0 (messages) of an originating server versus time.



(a) (b)

Fig. 10. (a) Moving average original message rate λ_1 (messages/sec) of Server 1 versus time. (b) Moving average retransmission rate r_1 (messages/sec) for Server 1 versus time.

Without overload control algorithm applied, it is easy to see from Fig. 9(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 user agent clients was larger than that of proxy Server 1, the queue size of each user agent client decreased slowly (see Fig. 9(b)) after new original message generation rates decreased.



(a) (b)

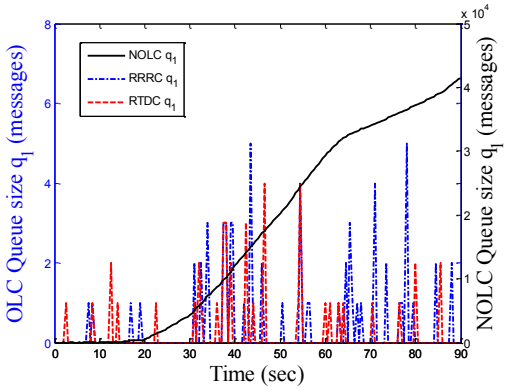
Fig. 11. Redundant retransmission ratio γ versus time. (a) Overload scenario 1. (b) Overload scenario 2.

During the short period of the demand burst, RTDC algorithm or RRRC algorithm made the queue size of Server 1 increase slowly by restricting the retransmission rate r_1 (see Fig. 10(b)). After the new user demand rate reduced at time t=30s, RTDC algorithm cancelled the overload at Server 1 within 27s (11s faster than RRRC algorithm), as shown in Fig. 9(a). In addition, RRRC algorithm clamped the redundant retransmission ratio (depicted in black solid line) around its target value (depicted in red dashed line) during the overload period, as shown in Fig. 11(a).

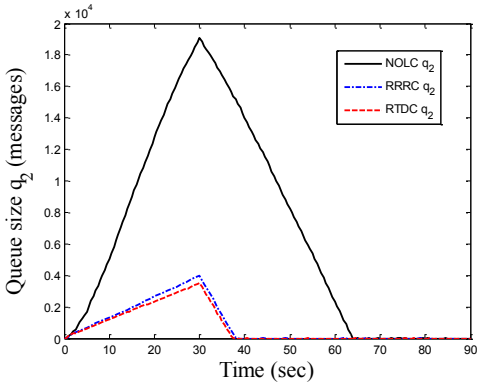
B. Overload at Server 2

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 user agent client was 50 messages/sec.

Without overload control algorithm applied, Figs. 12 and 13 demonstrate that Server 2 became overloaded first, which was followed by a later overload at Server 1. The queue size at Server 1 increased more quickly 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. 12).

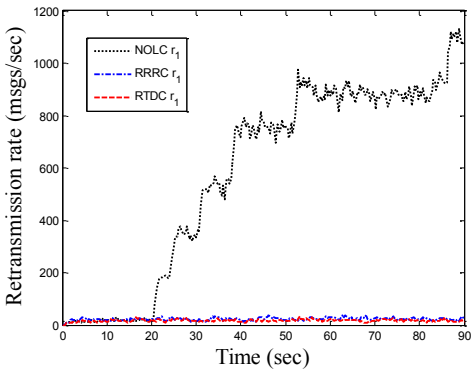


(a)

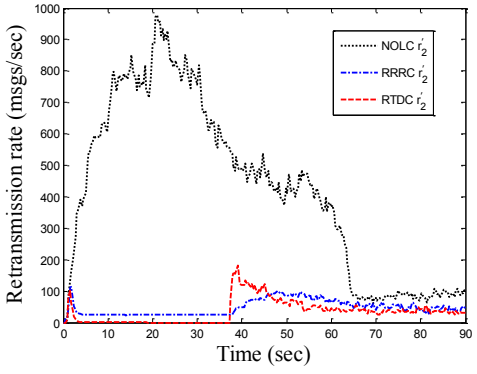


(b)

Fig. 12. (a) Queue size q_1 (messages) versus time. (b) Queue size q_2 (messages) versus time.



(a)



(b)

Fig. 13. (a) Moving average retransmission rate r_1 (messages/sec) for Server 1 versus time. (b) Moving average retransmission rate r'_2 (messages/sec) for Server 2 versus time.

With RRRC algorithm or RTDC algorithm applied, the retransmissions r'_2 were restricted (see Fig. 13(b)), thus the overload at Server 2 was mitigated and the queue size of Server 2 increased relatively slowly. In the mean time, Server 1 had enough capacity to process the limited retransmissions for Server 2, thus maintaining a small queue. After Server 2 resumed its normal service, RTDC only spent 7s (2s faster than RRRC algorithm) to cancel the overload and the buffer became empty at time $t\approx$37s, as shown in Fig. 12(b). In addition, RRRC algorithm clamped the redundant retransmission ratio (depicted in black solid line) around its target value (depicted in red dashed line) during the overload period, as shown in Fig. 11(b).

C. Remarks

1) Time Required to Overcome Overload Failure without Overload Control Algorithms: The queuing analysis and performance evaluation has demonstrated that (1) If the total arrival rate of all types of messages exceeds the service capacity continuously due to unnecessary retransmissions, the server will crash eventually; (2) without overload control algorithm applied, short-term overload may cause server failure eventually in both typical overload scenarios.

To avoid the server failure, an overloaded server should have sufficient capacity to process all the arrival messages after it resumes its normal service. We would like to estimate the time required to overcome the failure under the above two typical overload scenarios, when no overload control algorithm is applied.

Scenario A: an initial overload happens at Server 1 due to a demand burst

In this scenario, no overload happens at Server 2, thus there are no unnecessary retransmissions, i.e., $r'_2=0$. Every original message sent from Server 1 to Server 2 requires a response message, i.e., $\upsilon_1=\lambda_2$. After demand burst to Server 1 is cancelled, in order to prevent the queue size from increasing continuously, according to Eq. (2), we must guarantee that service rate $\mu_1\geq\lambda_1+r_1+\upsilon_1$ (or normalized capacity $C_1\geq\lambda_1+r_1+0.5\lambda_2$, given that the processing time of a request message is twice that of a response message).

From a long-term period, we assume $\lambda_2=\lambda_1$. Then we have $r_1\leq(C_1-1.5\lambda_1)=1000-1.5*200=3.5\lambda_1$, i.e., 4 upstream servers of Server 1 cannot trigger more than 3 retransmissions. This makes the maximum queue size allowed for Server 1 be $q_{1max}=(C_1-1.5\lambda_1)*(T_1+T_2+T_3+T_4)=(1000-1.5*200)*7.5=5250$ messages. Thus the time required to overcome the failure can be estimated as

$\Re=q_{1max}/(C_1-1.5\lambda_1-r_1)=q_{1max}/(C_1-1.5\lambda_1-3\lambda_1)=52.5$s.

Scenario B: an initial overload happens at Server 2 due to server slowdown

From a long-term period, we assume $r_2=r'_2$, $\lambda_2=\lambda_1$, and $\lambda_3=\lambda_2=\lambda_1$. 4 downstream servers of Server 2 have larger aggregated service capacity than Server 2 (i.e., $C_2=1000\leq2000$ messages/sec), and then they can reply Server 2 with response messages without any delay, and no unnecessary retransmissions would be triggered (i.e., $r'_3=0$). Every request message requires a response message, i.e., $\upsilon_1=\lambda_2+r_2=\lambda_1+r'_2$, and $\upsilon_2=\lambda_3=\lambda_2=\lambda_1$. After Server 2 resumes its normal service, in order to prevent the queue size from increasing continuously, according to Eqs. (1) and (2), we must guarantee that service rate $\mu_1\geq\lambda_1+r_1+r'_2+\upsilon_1$ and $\mu_2\geq\lambda_2+r_2+\upsilon_2$ (or normalized capacity $C_1\geq\lambda_1+r_1+r'_2+0.5(\lambda_1+r'_2)$ and $C_2\geq\lambda_1+r'_2+0.5\lambda_1$, given that the processing time of a request message is twice that of a response message). Server 1 and Server 2 have the same mean service capacity, i.e., $C_2=C_1$. Then we have $r_1+1.5r'_2\leq(C_1-1.5\lambda_1)=1000-1.5*200=700=3.5\lambda_1$, i.e., both Server 1 and Server 2 cannot trigger more than one retransmissions, thus the maximum queue size allowed for Server 1 is $q_{1max}=(C_1-1.5\lambda_1)*(T_1+T_2)=(1000-1.5*200)*1.5=1050$ messages. Then the time required to overcome the failure can be estimated as

$\Re=q_{1max}/(C_1-1.5\lambda_1-r_1-r'_2)=q_{1max}/(C_1-1.5\lambda_1-\lambda_1-\lambda_1)=3.5$s.

2) Comparison of Both Overload Control Algorithms: Since existing push-back solutions mitigate the overload by rejecting calls and causing the revenue loss, they cancel the overload much faster than our retransmission-based solutions in this paper. Their respective performance evaluation can be found in the references (e.g., [2, 23, 26, 40, 42, 45, 52, 53, 56, 58, 60]).

Performance comparison between RRRC algorithm and RTDC algorithm demonstrates that RTDC algorithm performs better than RRRC algorithm in both typical overload scenarios. Some people may want to ask a question, "What is the advantage of RRRC algorithm over RTDC algorithm?"

Most existing overload control solutions require the modification of the SIP header and the cooperation between the neighbouring servers which may belong to different carriers in different countries. Time-consuming standardization process and implementation cost may make the carriers reluctant to invest, thus hindering the large scale deployment of those solutions. In order to encourage the carriers implement RRRC algorithm or RTDC algorithm in their SIP servers to avoid the potential network collapse, both algorithms do not need the cooperation between different carriers all over the world by using implicit way to detect the overload from neighbouring servers.

Since RTDC algorithm does not know the queuing delay of an overloaded server, it uses the round trip delay to approximate the queuing delay as the overload indicator. However, the round trip delay in the SIP layer includes the hop-by-hop queuing delay in the IP layer. Any bandwidth congestion in the IP layer would introduce a long queuing delay, thus causing RTDC algorithm at an upstream server to prevent the retransmissions of the actual lost message unnecessarily when its downstream server has a light workload. Given that the network bandwidth has increased dramatically and powerful TCP/RED congestion control algorithm is implemented in the Internet, the probability of generating hop-by-hop queuing delay should be a decreasing function of the queuing delay in the IP layer. That is, the probability of a long hop-by-hop queuing delay in the IP layer is quite low, thus the round trip delay is mainly determined by the queuing delay of the overloaded server in the SIP layer. In addition, in order to minimize the impact of bandwidth congestion in the IP layer on the non-redundant retransmissions in the SIP layer, RTDC algorithm suggests the target hop-by-hop round trip delay to be the minimum time threshold (i.e., 500ms, according to SIP RFC [50]) for triggering a retransmission.

Quite different from RTDC algorithm, RRRC algorithm at each upstream server uses the redundant retransmission ratio to detect the overload at its downstream server, thus making reducing the impact of the bandwidth congestion in the IP layer significantly.

3) Call Blocking Probability of Both Overload Control Algorithms: SIP provides maximum 6 retransmissions for each original "Invite" message, i.e., the aggregated retransmission time is $T_1+\ldots+T_6=63T_1\approx32$s [50]. If the overload lasts more than 32s, overload control algorithms may forbid most or all of retransmissions including those non-redundant retransmissions required for message loss recovery. Consequently the actual message loss would cause the corresponding calls to be blocked.

As current global packet loss statistics indicates 10% packet loss by average in the Internet [35], we assume average message loss probability to be σ=10%.

For RRRC algorithm, the target redundant retransmission ratio is set as $\gamma_0=\sigma=0.1$. When the timer of an original message expires due to the overload or the actual message loss, SIP retransmits the original message with a retransmission probability χ. The retransmission for actual message loss is non-redundant, then redundant retransmission ratio becomes $\gamma_0=(1-\sigma)\chi$, thus the retransmission probability can be obtained as $\chi=\gamma_0/(1-\sigma)$. We can estimate the call blocking probability of RRRC algorithm as $p=\sigma*(1-\chi)=\sigma(1-\gamma_0/(1-\sigma))=0.1*(1-0.1/(1-0.1))\approx8.9\%$.

For RTDC algorithm, when the round trip delay caused by the overload continues to exceed the target delay, the retransmission rate calculated by Eq. (30) would decrease to zero, indicating all the retransmissions are forbidden. Then we can estimate the call blocking probability of RTDC algorithm as $p=\sigma=10\%$.

4) Overload Control Algorithm For SIP over TCP: As most service provider SIP networks choose large-scale deployment of SIP-over-UDP instead of TCP [17], both RRRC and RTDC are proposed to mitigate SIP overload by reducing retransmission rate only to avoid excessive call blocking. However, secure SIP-based services have to run SIP over TCP with Transport Layer Security (TLS) – In the best case,

the baseline UDP performance is about three times better that with TLS (the proxy chain mode); in the worst case, UDP is 17 times better performance than with TLS (the local proxy with TLS and mutual authentication) [62]. Therefore, for securing Voice-over-IP, SIP overload is caused by extensive computation cost of implementing TLS with TCP instead of SIP retransmission – in this secure SIP application scenario, RTDC algorithm can also mitigate the overload of SIP-over-TCP by reducing the original message rate, where a corresponding minor modification is required, e.g., the following Eq. (48) will substitute Eq. (28) to approximate the message queuing delay of the overloaded Server 2 as,

$$\dot{\tau}_2(t) \approx [\lambda_1(t) + \upsilon_2(t) - \upsilon_1(t)]/\upsilon_1(t) . \tag{48}$$

Correspondingly, the original message rate $\lambda_1(t)$ can be obtained by the following PI control algorithm expressed via

$$\lambda_1(t) = K_P e(t) + K_I \int_0^t e(\eta)\mathrm{d}\eta = K_P(\tau_0 - \tau_2(t)) + K_I \int_0^t (\tau_0 - \tau_2(\eta))\mathrm{d}\eta . \tag{49}$$

VII. CONCLUSIONS

We have made a queuing analysis of an overloaded SIP server to demonstrate that unnecessary retransmissions can make overload much worse and therefore should be controlled before any other overload control mechanisms are applied. Two novel PI control algorithms are proposed to mitigate the overload by reducing the retransmission rate only, while maintaining the original message rate to avoid excessive revenue loss.

In order to provide the design guideline of obtaining the PI controller gains, we apply control theoretic approach to model the interaction between an overloaded downstream server and its upstream server as a feedback control system. Nyquist stability criterion is used to ensure the system stability in the controller design. In addition, the advantages, disadvantages and the call blocking probability of both proposed overload control algorithms have been discussed.

Through OPNET simulations, we have demonstrated that: (1) without any overload control algorithm applied, the overload at the downstream server may propagate or migrate to its upstream servers eventually, thus causing potential widespread server crash in a SIP network; (2) both proposed overload control algorithms can mitigate the overload effectively and prevent the overload propagation, thus helping the server cancel the short-term overload quickly. In addition, we also estimate the time required to overcome the overload failure, if no overload control algorithm is activated.

When combined with existing overload control mechanisms, our proposal can improve the overload situation and increase server utilization significantly. Furthermore, since our two overload control algorithms do not require any modification on the SIP header and the cooperation between different carriers, any carrier can freely implement our solution in their SIP servers to avoid potential widespread network failure in the near future.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers whose comments help to improve the quality of this paper. This work was supported by the NSERC grant #CRDPJ 354729-07 and the OCE grant #CA-ST-150764-8. OPNET simulation codes for RRRC and RTDC algorithms in this paper are available for non-commercial research use upon request.

REFERENCES

1. 3GPP (2011). 3rd Generation Partnership Project. http://www.3gpp.org.
2. Abdelal, A., & Matragi, W. (2010). Signal-Based Overload Control for SIP Servers. In *Proceedings of IEEE CCNC*, Las Vegas, NV.
3. Amooee, A.M., & Falahati, A. (2009). Overcoming Overload in IMS by Employment of Multiserver Nodes and Priority Queues. In *Proceedings of International Conference on Signal Processing Systems*, Singapore, pp. 348-352.

4. Astrom, K.J., Goodwin, G.C., & Kumar, P.R. (1995). *Adaptive Control, Filtering, and Signal Processing*, *Vol. 74*, *The IMA Volumes in Mathematics and its Applications*, Springer-Verlag.
5. Astrom, K.J., & Hagglund, T. (1995). *PID Controllers: Theory, Design, and Tuning*, *Second Edition*, Instrument Society of America, North Carolina.
6. Astrom, K.J., Hagglund, T., Hang, C.C., & Ho, W.K. (1993). Automatic Tuning and Adaptation for PID Controllers-A Survey. *IFAC Journal of Control Engineering Practice*, *1*(4), pp. 699-714.
7. Astrom, K.J. & Murray, R.M. (2008). *Feedback Systems: An Introduction for Scientists and Engineers*, Princeton University Press, New Jersey.
8. Astrom, K.J., & Wittenmark, B. (1995). *Adaptive Control*, *Second Edition*, Addison-Wesley Publication Co., Boston, MA.
9. Bristol, E.H. (1986). The EXACT Pattern Recognition Adaptive Controllers, A User-oriented Commercial Success. *Adaptive and learning systems*, New York, Plenum Press, pp. 149-163.
10. Cisco white paper. (2005). DiffServ – The Scalable End-to-End QoS Model.
11. Dacosta, I., Balasubramaniyan, V., Ahamad, M., & Traynor, P. (2009). Improving Authentication Performance of Distributed SIP Proxies. In *Proceedings of IPTComm*, Atlanta, GA, July 2009.
12. Dacosta, I., & Traynor, P. (2010). Proxychain: Developing a Robust and Efficient Authentication Infrastructure for Carrier-Scale VoIP Networks. In *Proceedings of the USENIX Annual Technical Conference*, Boston, MA.
13. Dang, T.D., Sonkoly, B., & Molnar, S. (2004). Fractal Analysis and Modeling of VoIP Traffic. In *Proceedings of International Telecommunication Network Strategy and Planning Symposium*, Vienna, Austria, pp. 123–130.
14. Desbourough, L., & Miller, R. (2002). Increasing customer value of industrial control performance monitoring—Honeywell's experience. In *Proceedings of Sixth International Conference on Chemical Process Control*, AIChE Symposium Series, *98*(326).

15. Faccin, S.M., Lalwaney, P., & Patil, B. (2004). IP Multimedia Services: Analysis of Mobile IP and SIP Interactions in 3G Networks. *IEEE Communications Magazine*, *42*(1), pp. 113-120.
16. Garroppo, R.G., Giordano, S., Spagna, S., & Niccolini, S. (2009). Queuing Strategies for Local Overload Control in SIP Server. In *Proceedings of IEEE Globecom*, Honolulu, Hawaii.
17. Geneiatakis, D., & Lambrinoudakis, C. (2007). A lightweight protection mechanism against signaling attacks in a SIP-based VoIP environment. *Telecommunication Systems*, *36*(4), pp. 153-159.
18. Geng, F., Wang, J., Zhao, L., & Wang, G. (2006). A SIP Message Overload Transfer Scheme. In *Proceedings of ChinaCom*, Beijing, China.
19. Govind, M., Sundaragopalan, S., Binu, K.S., & Saha, S. (2003). Retransmission in SIP over UDP – Traffic Engineering Issues. In *Proceedings of International Conference on Communication and Broadband Networking*, Bangalore, India.
20. Gurbani, V.K., Hilt, V., & Schulzrinne, H. (2011). Session Initiation Protocol (SIP) Overload Control. *IETF Internet-Draft*, draft-ietfsoc-overload-control-05.
21. Gurbani, V.K., Jagadeesan, L.J., & Mendiratta, V.B. (2005). Characterizing Session Initiation Protocol (SIP) Network Performance and Reliability. In *Proceedings of 2nd International Service Availability Symposium*, Berlin, Germany, pp. 196-211.
22. Hilt, V., Noel, E., Shen, C., & Abdelal, A. (2011). Design Considerations for Session Initiation Protocol (SIP) Overload Control, *IETF Internet-Draft*, draft-hilt-soc-overload-design-06.
23. Hilt, V., & Widjaja, I. (2008). Controlling Overload in Networks of SIP Servers. In *Proceedings of IEEE ICNP*, Orlando, Florida, pp. 83-93.
24. Ho, W.K., Hong, Y., Hansson, A., Hjalmarsson, H., & Deng, J.W. (2003). Relay Auto-Tuning of PID Controllers using Iterative Feedback Tuning. *Automatica*, *39*(1), pp. 149-157.
25. Ho, W.K., Lee, T.H., Han, H.P., & Hong, Y. (2001). Self-Tuning IMC-PID Control with Interval Gain and Phase Margin Assignment. *IEEE Transactions on Control Systems Technology*, *9*(3), pp. 535-541.

26. Homayouni, M., Jahanbakhsh, M., Azhari, V., & Akbari, A. (2010). Controlling Overload in SIP Proxies: An Adaptive Window Based Approach Using no Explicit Feedback. In *Proceedings of IEEE Globecom*, Miami, FL.
27. Hong, Y., Huang, C., & Yan, J. (2010). Analysis of SIP Retransmission Probability Using a Markov-Modulated Poisson Process Model. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium*, Osaka, Japan, pp. 179–186.
28. Hong, Y., Huang, C., & Yan, J. (2010). Mitigating SIP Overload Using a Control-Theoretic Approach. In *Proceedings of IEEE Globecom*, Miami FL.
29. Hong, Y., Huang, C., & Yan, J. (2010). Stability Condition for SIP Retransmission Mechanism: Analysis and Performance Evaluation. In *Proceedings of IEEE SPECTS*, Ottawa, Canada, pp. 387-394.
30. Hong, Y., Huang, C., & Yan, J. (2011). Controlling Retransmission Rate For Mitigating SIP Overload. In *Proceedings of IEEE ICC*, Kyoto, Japan.
31. Hong, Y., Huang, C., & Yan, J. (2011). Design Of A PI Rate Controller For Mitigating SIP Overload. In *Proceedings of IEEE ICC*, Kyoto, Japan.
32. Hong, Y., Huang, C., & Yan, J. (2011). Modeling and Simulation of SIP Tandem Server with Finite Buffer. *ACM Transactions on Modeling and Computer Simulation*, *21*(2).
33. Hong, Y., Huang, C., & Yan, J. (2012). A Comparative Study of SIP Overload Control Algorithms. In *Internet and Distributed Computing Advancements: Theoretical Frameworks and Practical Applications*, J. Abawajy, M. Pathan, M. Rahman, A.K. Pathan, and M.M. Deris, eds., IGI Global, Hershey, PA.
34. Hong, Y., Huang, C., & Yan, J. (2012). Modeling Chaotic Behaviour of SIP Retransmission Mechanism. *International Journal of Parallel, Emergent and Distributed Systems*, in press.
35. Internet Traffic Report. (2010). http://www.internettrafficreport.com/.
36. IPTEL. (2011). SIP Express Router. http://www.iptel.org/ser/.

37. Jiang, H., Iyengar, A., Nahum, E., Segmuller, W., Tantawi, A., & Wright, C. (2009). Load Balancing for SIP Server Clusters. In *Proceedings of IEEE INFOCOM*, Rio de Janeiro, Brazil, pp. 2286-2294.
38. Katabi, D., Handley, M., & Rohrs, C. (2002). Congestion Control for High Bandwidth Delay Product Networks. In *Proceedings of ACM SIGCOMM*, Pittsburgh, PA.
39. Kitatsuji, Y., Noishiki, Y., Itou, M., & Yokota, H. (2010). Service Initiation Procedure with On-demand UE Registration for Scalable IMS Services. In *Proceedings of The Fifth International Conference on Mobile Computing and Ubiquitous Networking*, Seattle, WA.
40. Montagna, S., & Pignolo, M. (2010). Comparison between two approaches to overload control in a Real Server: "local" or "hybrid" solutions? In *Proceedings of IEEE MELECON*, Valletta, Malta, pp. 845-849.
41. Nahm, E.M., Tracey, J., & Wright, C.P. (2007). Evaluating SIP server performance. In *Proceedings of ACM SIGMETRICS*, San Diego, CA, pp. 349–350.
42. Noel, E., & Johnson, C.R. (2007). Initial simulation results that analyze SIP based VoIP networks under overload. In *Proceedings of 20th International Teletraffic Congress*, Ottawa, Canada, pp. 54-64.
43. Noel, E., & Johnson, C.R. (2009). Novel Overload Controls for SIP Networks. In *Proceedings of 21st International Teletraffic Congress*, Paris, France.
44. Ogata, K. (2002). *Modern Control Engineering*, *Fourth Edition*, Prentice Hall, New Jersey.
45. Ohta, M. (2006). Overload Control in a SIP Signaling Network. In *Proceeding of World Academy of Science, Engineering and Technology*, Vienna, Austria, pp. 205-210.
46. Ohta, M. (2006). Overload Protection in a SIP Signaling Network. In *Proceedings of International Conference on Internet Surveillance and Protection*, Cap Esterel, France.
47. OPNET Technologies Inc. (2003). *OPNET Modeler Manuals*, *OPNET Version 10.0*.
48. Rebahi, Y., Pallares, J.J., Vingarzan, D., Onofrei, A.A., Gouveia, F., & Magedanz, T. (2011). A Priority Queuing Model for IMS-based Emergency Services. In *Proceedings of IEEE CCNC*, Las Vegas, NV.
49. Rosenberg, J. (2008). Requirements for Management of Overload in the Session Initiation Protocol. *IETF RFC 5390*.

50. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., & Schooler, E. (2002). SIP: Session Initiation Protocol. *IETF RFC 3261*, June 2002.
51. Sadhukhan, P., Das, P.K., & Saha, S. (2011). Hybrid mobility management schemes integrating mobile IP and SIP for seamless invocation of services in All-IP network. *Telecommunication Systems*, DOI: 10.1007/s11235-011-9483-7 Online First.
52. Shen, C., & Schulzrinne, H. (2010). On TCP-based SIP Server Overload Control. In *Proceedings of IPTComm*, Munich, Germany.
53. Shen, C., Schulzrinne, H., & Nahum, E. (2008). SIP Server Overload Control: Design and Evaluation. In *Proceedings of IPTComm*, Heidelberg, Germany.
54. Subramanian, S.V., & Dutta, R. (2009). Measurements and Analysis of M/M/1 and M/M/c Queuing Models of the SIP Proxy Server. In *Proceedings of IEEE ICCCN*, San Francisco, CA, pp. 397-402.
55. Sukhov, A., Calyam, P., Daly, W., & Illin, A. (2005). Towards an analytical model for characterizing behaviour of high-speed VoIP applications. In *Proceedings of TERENA Networking Conference*, Poznan, Poland.
56. Sun, J., Tian, R., Hu, J., & Yang, B. (2009). Rate-based SIP Flow Management for SLA Satisfaction. In *Proceedings of IEEE/IFIP IM*, New York, NY, pp. 125-128.
57. Usui, T., Kitatsuji, Y., & Yokota, H. (2013). A study on traffic management cooperating with IMS in MPLS networks. *Telecommunication Systems, 52*(2), pp. 671-680.
58. Wang, Y.G. (2010). SIP Overload Control: A Backpressure-based Approach. *ACM SIGCOMM Computer Communications Review*, *40*(4), pp. 399-400.
59. Xu, L., Huang, C., Yan, J., & Drwiega, T. (2009). De-Registration Based S-CSCF Load Balancing in IMS Core Network. In *Proceedings of IEEE ICC*, Dresden, Germany.
60. Yang, J., Huang, F., & Gou, S.Z. (2009). An Optimized Algorithm for Overload Control of SIP Signaling Network. In *Proceedings of 5th International Conference on Wireless Communications, Networking and Mobile Computing*, Beijing, China.

61. Zhang, G., Fischer-Hübner, S., & Ehlert, S. (2010). Blocking attacks on SIP VoIP proxies caused by external processing. *Telecommunication Systems*, *45*(1), pp. 61-76.
62. Shen, C., Nahum, E., Schulzrinne, H., & Wright, C. P. (2012). The Impact of TLS on SIP Server Performance: Measurement and Modeling. *IEEE/ACM Transactions on Networking*, *20*(4), pp. 1217-1230.

Yang Hong received B.S. degree in electronic engineering from Shanghai Jiao Tong University, China, M.E. degree in electrical engineering from National University of Singapore, Singapore, and Ph.D. degree in electrical engineering from University of Ottawa, Canada. His research interests include SIP overload control, Internet congestion control, modeling and performance evaluation of computer networks, and industrial process control.

Dr. Huang received his B. Eng. in 1985 and M. Eng. in 1988 both in Electronic Engineering from Tsinghua University, Beijing, China. He received a Ph.D. degree in Electrical Engineering from Carleton University, Ottawa, Canada in 1997. From 1996 to 1998, he worked for Nortel Networks, Ottawa, Canada where he was a systems engineering specialist. He was a systems engineer and network architect in the Optical Networking Group of Tellabs, Illinois, USA during the period of 1998 to 2000. Since July 2000, he has been with the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada where he is currently an associate professor. Dr. Huang won the CFI new opportunity award for building an optical network laboratory in 2001. He was an associate editor of *IEEE Communications Letters* from 2004 to 2006. Dr. Huang is a senior member of IEEE.

James Yan is currently an adjunct research professor with the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada. Dr. Yan received his B.A.Sc., M.A.Sc., and Ph.D. degrees in electrical engineering from the University of British Columbia, Vancouver, Canada. He was a telecommunications systems engineering senior manager in Bell-Northern Research (1982-1996) and in Nortel (1996 to 2004). In those positions, he was responsible for projects in performance analysis of networks and products, advanced technology research, planning new network services and architectures, development of network design methods and tools, and new product definition. From 1988 to 1990, he participated in an exchange program with the Canadian Federal Government, where he was project prime for the planning of the evolution of the nationwide federal government telecommunications network. Dr. Yan is a member of IEEE and Professional Engineers Ontario.