RC24053 (C0609-006) September 15, 2006
Computer Science

IBM Research Report

Flow Management for SIP Application Servers

Jing Sun
Department of Computer Science and Technology
Tsinghua University
Beijing, China

Jinfeng Hu, Ruixiong Tian, Bo Yang
IBM Research Division
China Research Laboratory
HaoHai Building, No. 7, 5th Street
ShangDi, Beijing 100085
China

Flow Management for SIP Application Servers

Jing Sun[1], Jinfeng Hu[2], Ruixiong Tian[2], Bo Yang[2]
[1]Dept. Computer Sci. and Tech., Tsinghua University, Beijing, China
[2]IBM China Research Lab, Beijing, China
E-mail: sunjing05@mails.tsinghua.edu.cn, {hujinf, tianruix, boyang}@cn.ibm.com

Abstract—**Building healthy, long-running SIP application servers is more challenging than WEB servers. Since SIP has been adopted by telecom industry, SIP servers should meet carrier-grade QoS requirement, which is more critical than WEB services. SIP messages are well organized into sessions. When a server experiences an incoming message burst or a relatively long time of overload, many transactions would experience very long response time, even resulting in session failures. This is not acceptable by telecom providers. Unlike http, most SIP messages are transmitted by UDP protocol and retransmissions are employed to improve the transmission reliability. However, these retransmissions would exacerbate the situation when the server is under high load. In this paper, we use front-end flow management to address these challenges simultaneously. Our method integrates concurrency limit, message scheduling, and admission control to achieve overload protection and QoS guarantee. We also devise some novel techniques such as twin-queue scheduling, retransmission removal, and GC detection to improve the effectiveness of the method. Experiments show that the system apparently reduces the average response time and improves the response time distribution.**

Index Terms—**SIP, Flow Management, Session, SLA**

I. Introduction

Session Initiation Protocol (SIP) [1] has been widely accepted as a major signaling protocol to establish and manage sessions in IP networks. Here, a session can be a simple telephone call, a collaborative conference, or any other kind of multimedia session. Moreover, it is also adopted in carrier grade environments, such as the IP Multimedia Subsystem (IMS) of the 3rd Generation Partnership Project (3GPP) in the emerging Universal Mobile Telecommunications System (UMTS) networks [2]. Under this circumstance, leading J2EE middleware providers also embraces SIP and integrates corresponding support (e.g., JSR 116 [10]) into their major products, such as IBM WebSphere and BEA WebLogic.

Compared to web servers, a server running SIP applications has some different characters. First, During a SIP session, messages compose transactions, the failure of which might cause the whole session to fail. A simple call session is shown in Fig. 1, in which there are three transactions: the INVITE one, the ACK one, and the BYE one.

Second, SIP is chosen by telecom industry as the signaling protocol in 3G network and NGN. Therefore, SIP servers should meet more critical QoS requirements than http servers. For a simple telephony call, the response time of a session-establishing request (an non-retransmitted INVITE) on the server end should not exceed 100 milliseconds with a high

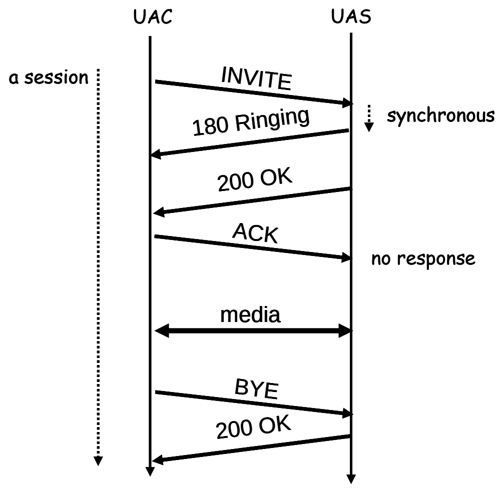


Fig. 1. The simplest SIP session.

possibility [3]. Here “response time” is defined as the time interval from receiving an INVITE to sending out its first synchronous reply, which means the first non-100 response, if the server is a SIP User Agent Server (UAS) or the first corresponding outbound INVITE if the server is a SIP proxy. When the SIP server encounters an incoming message burst, or a long period of overload, many requests would fail to meet the QoS requirement, or even fail to establish the session because of the too long response times. Fig. 2 gives a real server running profile. In this case, the server ran a simplest SIP UAS as shown in Fig. 1 based on a JSR-116-compliant SIP container on top of RedHat ES 3 update 3. The node has an Intel 2.4G hyper-threaded P4 CPU and 4G RAM. A SIPP[4] on a similar configured server in the same network ran as a User Agent Client (UAC) and initializes sessions at a rate of 400 calls per second (CPS). The running lasted 3641 seconds, during which

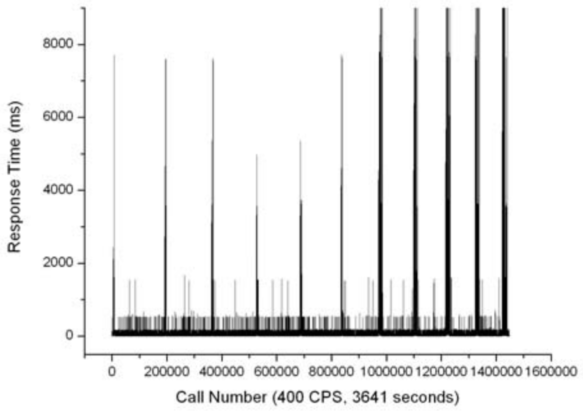


Fig. 2. The server’s response time under 400 CPS

This work was done when Jing is working as intern in IBM research.

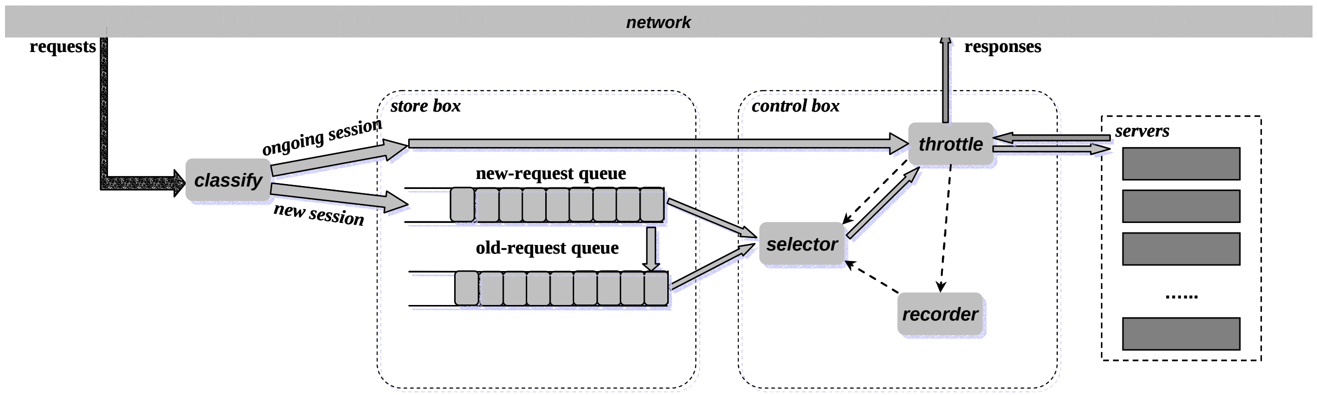


Fig. 3. The Frond-end Flow Management System Architecture

152 calls failed and 95634 messages were retransmitted. As shown in Fig. 2, once the server experienced overload, not only many calls failed, but also the response times of a mass of calls were heavily prolonged, exceeding the requirement of the service level agreement (SLA).

Third, because SIP messages are usually transmitted by UDP, when the server does not respond timely, the client will retransmit the message, maybe for many times. This further exacerbates the overload situation. Moreover, SIP servers embedded in a Java-based middleware have to experience long no-service pause times due to the Java garbage collections (GC), which is the reason why spikes appear periodically in Fig. 2.

Consequently, to keep a SIP server running smoothly in a long term, we should address the following challenges:

1. A server should be always protected from overload;
2. A server should keep response times within the limit specified by SLA for as many requests as possible;
3. The session integrity should be honored. Only session-establishing requests could be rejected if some admission control method is applied.
4. GC's affection should be taken into account.

In this paper, we propose a front-end flow management system (FEFM) to do overload protection and QoS improvement for SIP servers. FEFM uses concurrency limit to control the rate of new session creation on the server, preventing it from overload. To adjust the response times to meet the SLA requirement as much as possible, the INVITE messages are scheduled according to the prediction of the next request's response time. If a request is predicted to be able to have its response time meet the SLA, it would be prior to send to the server. To eliminate the prediction error caused by GC, a GC detector judges whether a request has been affected by GC; if so, its response time would not be used in the predictor. FEFM also adopts a twin-queue structure to improve the server's throughput, which can also be used to compromise a suitable trade-off between the SLA guarantee and the throughput. Elaborated experiments are made to evaluate FEFM from many sides and show its effectiveness for overload protection and response time management.

The rest of the paper is organized as follow. Section III describes the system architecture and major algorithms. Section IV gives experiment results. Section IV discusses related works. Section V concludes the paper.

II. System Architecture and Algorithms

The Front-end Flow Management system (FEFM) runs as a SIP stateless proxy [1] on a node in front of a SIP server (or a set of SIP servers). The client sends SIP messages to the proxy who decides which of them are delivered to the server and which are rejected. The system's primary work is to provide overload protection for the SIP server. As mentioned in Section I, it should also shape the response times to maximize the successful rate of sessions that conform to the SLA requirement. However, the SLA is not the only target. We could not reject too many sessions just in order to ensure a small part of requests' response times. The system should also maintain the SIP server's throughput as much as possible, while considering the SLA constraint.

The FEFM's architecture is shown in Fig. 3. When a new request comes, FEFM firstly classifies it. If it belongs an ongoing session, it will be directly sent to the server. Otherwise, it will be put into a *new-request queue* that buffers the session-establishing requests. Meanwhile, a selector picks up a proper session-establishing request from the header of the new-request queue or the header of the *old-request queue* to deliver it to the server when required by the throttle. (The scheduling between the two queues will be described in subsection C.) The throttle limits the number of the concurrent requests that are being executed on the server to guard against server overload. The timestamps of any request and its reply when they pass through the throttle will be written down by a recorder. (The reply of a non-INVITE request is defined as its response if the server acts as a SIP UAS or the corresponding outbound request if the server acts as a SIP proxy.) The timestamps are used to help the selector make its decision. The parameters used in FEFM are defined in Table I.

A. Overload protection

To control the server load, FEFM limits the number of the requests being concurrently executed on the server. By setting a threshold *N*, the throttle can prevent the server from overload. The throttle observes the number of the concurrent requests on the server at runtime, noted as *n*. After a request is sent to the server, whatever it is a session-establishing request or not, the

TABLE I
THE PARAMETERS DEFINITION

Parameter	Definition
N	Max number of concurrent requests on a server
n	Number of current requests running on a server
D	Response time required by SLA
D_1	The threshold for the new-request queue
D_2	The threshold for the old-request queue
T_e	Last request's real execution time
T_w	Waiting time of a request in the new-request queue
T_m	The predicted execution time of the mth request
α	Weight of the exponentially moving average

throttle increases n by 1. When the reply of a request is received, it decreases n by 1. For a request with no response, such as ACK, a timer is set to be triggered periodically (in our system, it is set to $D/2$, where D is the response time requirement documented in SLA) to decrease n by the number of ACKs that are sent to the server in the last period. When a request that should have a response but does not get a response for a long time ($2\cdot D$ in our system), it expires and n is then decreased by 1. Every time when n is decreased, the throttle will check if $n<N$ holds. If so, it would notify the selector to select $N-n$ new session-establishing requests to send to the server.

It should be noted here that setting N does not mean that the number of concurrent executed requests would be never beyond N because all the non-INVITE requests are admitted into the server immediately without checking n. However, when n is over N, no session-establishing request will be admitted into the server any more. Such a situation will persist until some requests get their replies or expire, which cause n to drop under N.

B. *Retransmission removal*

As mentioned above, a SIP client would retransmit a message if it does not receive any expected response in time. The retransmission interval commonly starts at 500ms and doubles after every retransmission. By default, a message will not be retransmitted more than 7 times. However, the retransmissions that originally designed to improve the message reliability will hurt the performance, especially when the load is already high.

As listed in section I, during one-hour running, the UAC retransmitted 95634 messages. Unfortunately, most of the retransmissions occurred when the server was under high load (after GC), which further exacerbated the situation: the load imposed on the server gets even higher, the request queue on the server gets even longer, and consequently the response times gets even larger (more than 8 seconds at the most). As a matter of course, if we eliminate unnecessary retransmissions that are sent to the server, the server's load will be alleviated, especially during the high-load or overload period.

The retransmission removal is implemented as follows. If a request has already been sent to the server but has neither got its reply nor expired, it will be recorded in the recorder. The INVITE requests that have not been sent to the server will be buffered in the new-request queue or the old-request queue. When FEFM receives a new request, it first checks if it has already been in the recorder, the new-request queue, or the old-request queue. If so, it will be judged as a retransmission and discarded immediately. If a request gets its reply from the server or expires, it will be removed from the recorder. After that, its retransmission will be treated as a non-retransmitted request and not be removed. Therefore, the retransmission removal mechanism of FEFM will not hurt the message reliability.

C. *Response time management*

Generally, a SLA imposes response time requirements on many kinds of requests. In FEFM, we focus our response time management on session-establishing requests. Other requests are admitted into the server as soon as possible without being queued, so they must have better response times than session-establishing requests. Such a design is because 1) the response time of session-establishing request is a part of the end-to-end "post-dialing delay" (a.k.a., "post-selection delay"), which is a highly important QoS metric in telecom standards [11][12]; and 2) queuing or even rejecting other requests takes the risk of cutting off an existing session, which means call failure and should be intensively avoided in telecom services.

When the server load is high, admitting all sessions implies long response times that might be far beyond the SLA requirement. Therefore, FEFM would rather only admit those session-establishing requests that could meet the SLA requirement and reject others. To achieve this, FEFM needs to estimate each session-establishing request's response time, which is composed by T_w (the time that a session-establishing request has been waiting in the queues before it is sent to the server) and T_m (its execution time on the server). T_w is measured by the recorder (according to the time when the request is received by the proxy and when it is about to admitted into the server). T_m is estimated by an exponentially moving average:

$$T_m = \alpha \cdot T_e + (1-\alpha) \cdot T_m \qquad (1)$$

Every time a session-establishing request gets its reply, T_m is updated using the equation (1), in which T_e is the request's execution time measured by the recorder and α is a weight. α is larger, T_m is more responsive. When the load is light, α should be set relatively small to prevent temporarily fluctuation of the response time from causing the estimated T_m deviating from the stable value too much. On the contrary, when the load is high, α should be set larger to reflect the current runtime status rapidly.

The selector checks the header of the new-request queue whenever it is asked by the throttle to admit a new session-establishing request. It predicts the request's response time as (T_w.+ T_m). If it is less than D_1, a preset threshold, the request will be admitted into the server immediately. Otherwise it will be moved to the old-request queue and the selector will turn back to check the new header of the new-request queue. Such a process continues until the selector admits a request from the new-request queue or the new-request queue turns

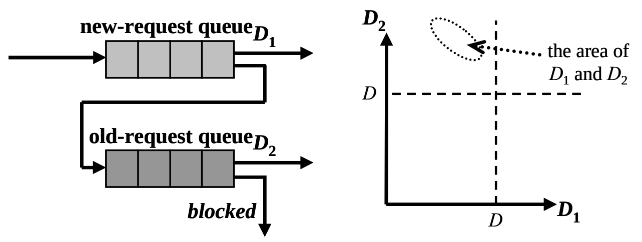


Fig. 4. Twin-queue schedule

empty. If the latter happens, the selector will then turn to check the header of the old-request queue. If its predicted response time is less than D_2, another preset threshold, it will be admitted into the server immediately. Otherwise, it will be rejected (by sending back a "503 Service Unavailable" response) and the selector will turn back to check the new header of the old-request queue. Such a process continues until a request is admitted into the server from the old-request queue or the old-request queue turns empty. If the latter happens, this round of selection terminates. When the selector is asked by the throttle to admit $N-n$ new session-establishing requests, it runs $N-n$ rounds of selection.

D. The twin-queue scheduling

The above twin-queue scheduling is illustrated in Fig. 4. FEFM chooses this strategy, other than a simple FIFO queue, in order to improve the server's throughput, while simultaneously considering the SLA requirement. Any request not being able to meet the response time requirement with a high possibility is not directly rejected, but is put into the old-request queue, which gives it another opportunity to be admitted. When the incoming rate of session-establishing requests declines, the new-request queue would be empty in some time. If FEFM has no request hold in the old-request queue, the server will be idle, which would cause the server's throughput to decrease. In this situation, the requests in the old-request queue will be valuable to maintain the server's throughput, even though they do not conform to the SLA requirement. Moreover, since the requests do not come evenly, (it is more reasonable to assume that the incoming rate of session-establishing requests conforms to a Poisson distribution), the requests accumulated in the old-request queue during a peak will be processed at the off-peak time. Therefore, it not only maintains the server's stable throughput, but also decreases the rejection rate of the session. Regarding the parameters of the twin-queue scheduling, D_1 maximizes the rate of admitted sessions that successfully conforms to the SLA requirement, while D_2 minimizes the rejection rate of the sessions.

Carefully selecting D_1 and D_2 could balance the SLA guarantee and the server's throughput. Commonly, we can set D_1 as the SLA-specified response time requirement. However, in practice, we recommend setting it a litter smaller than the requirement. This is because the response time prediction could have some error, overestimation or underestimation. Setting D_1 as SLA requirement would move some requests into old-request queue by mistake due to overestimation. Lowering D_1 can help reduce the possibility of this mistake. However, it would increase the possibility of another mistake that is due to underestimation: admitting some requests that eventually fail to meet the SLA requirement. This is not a big problem because we have some room to tolerate it. Most SLAs do not impose a strict requirement upon all the requests. They commonly provide a percentile requirement, e.g., a 95 percentile value.

D_2 should be larger than the SLA requirement. But we don't recommend a very large D_2. Large D_2 indicates long post-dialing delay, which is a bad experience for users. The area of D_1 and D_2 is depicted in Fig. 4, in which the SLA requirement is noted as D.

E. GC detection

For a SIP server implemented in Java, garbage collection (GC) is inevitable in a long run. GC will heavily prolong the execution time, which will hurt the accuracy of the response time prediction. If an execution time during GC period is used in equation (1), T_m will deviate too much from the real value. If it is an overestimation, it will cause all the succeeding session-establishing requests to be pushed into the old-request queue for a long time, which therefore impairs the ability of the system to meet the SLA requirement. FEFM develops a technique to avoid such an overestimation.

The recorder continuously monitors the time interval of two consecutive replies received from the server. Assume at time t, this value suddenly increases and exceeds a threshold (100ms is good enough based on our experiments). Then FEFM presumes that a GC just completed on the server. From then on, all the response times whose related requests are admitted into the server before t will not be used for the response time prediction, more specifically, the estimation of T_m by equation (1).

III. Experiment Results and Discussions

We used a lot of experiments to evaluate the FEFM system.

The test bed includes three nodes, running SIP server, FEFM, and client respectively, connected by 1G Ethernet. The servers have the same configuration: 1* 2.4 G HyperThread Pentium 4 CPU, 4G RAM, Redhat ES 3 update 3, and Linux 2.4.3-smp. The SIP server runs a simple UAS (as illustrated in Fig. 1.) on top of the SIP container within IBM WebSphere 6.1, which is compliant with JSR 116 [10]. FEFM is implemented in Java and runs on a SIP stack compliant with JSR 32 [15]. Both the server and the FEFM use IBM JVM 1.5. The client runs SIPP 1.1 [4] as a UAC. The parameters of SIP server and FEFM are shown in Table II. We don't need more than one SIPP client to generate SIP traffic because SIPP is written in C and can generate enough loads for the SIP server which is Java-based[*].

A. Response time prediction

The system heavily depends on the execution time prediction. Fig. 5 shows predicted execution times vs. real execution times (from recorder). The statistics are presented in Table III. The predicted execution times match the real values well. There are

[*] It is certain that C-based server could provide higher performance than Java. However, Java middleware is more attractive in the market for building application servers because of its ease-to-use.

TABLE II
THE SERVER'S CONFIGURATION

FEFM Parameter	Value	Server Parameter	Value
D	200 ms	Min Heap Size	2G
N	150	Max Heap Size	2G
D_1	D	GC policy	gencon
D_2	8 s		
α	0.5		

The FEFM has the same JVM configuration with SIP server

TABLE III
THE PREDICTON AND REAL MEASUREMENT STATISTICS

Statistic	Prediction	Real Measurement
Mean (ms)	5.6806	6.87468
Standard Deviation	13.59279	22.30121
Standard Error	0.06079	0.09973
Min (ms)	0.61918	0
Max (ms)	487.37405	567
Median	3.12883	3
25% (ms)	2.60938	3
75% (ms)	4.09382	4
95% (ms)	19.59743	21

TABLE IV
THE RESPONSE TIME STATISTICS

Statistic	**300 cps**		**400 cps**		**500 cps**	
	with FEFM	**no FEFM**	**with FEFM**	**no FEFM**	**with FEFM**	**no FEFM**
Mean (ms)	12.04	26.42	12.37	26.35	22.05	84.39
Standard Error	0.186	0.676	0.094	0.466	0.170	1.520
Min (ms)	1	1	1	1	1	1
Max (ms)	1699	9199	1655	9842	2184	19883
95% (ms)	20	23	65	83	119	124
< 150 ms	98.87%	98.50%	98.39%	97.64%	96.82%	96.28%
Total calls	180000	180000	240000	240000	300000	300000
Retrans	4507	7225	12439	13340	41911	46919
Timeout	0	0	0	0	333	796

some underestimations, especially during GC periods, but almost no overestimations are observed. This is because FEFM only detects GCs after they occur, but does not predict GCs before they occur. Therefore, it is possible for a quest admitted into the server whose execution time has been predicted as common finally encounters a GC when being executed on the server. As discussed above, underestimations have no significant impact on the performance management, whilst overestimations have, which are eliminated by our GC detection technique.

B. *GC policy selection*

As shown above, GC has ill impact on response times. We have tested difference kinds of GCs and tried to choose the one with the least impact. IBM JVM has three kinds of GCs: Optthrughput, Optavgpause, and Gencon (SUM JVM has some

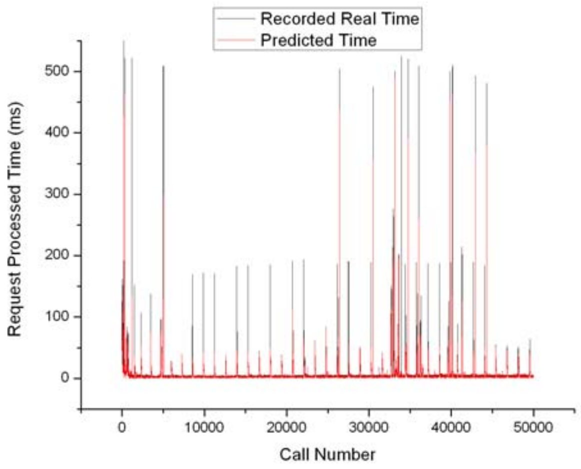


Fig. 5. Predicted execution times vs. real execution times (300CPS)

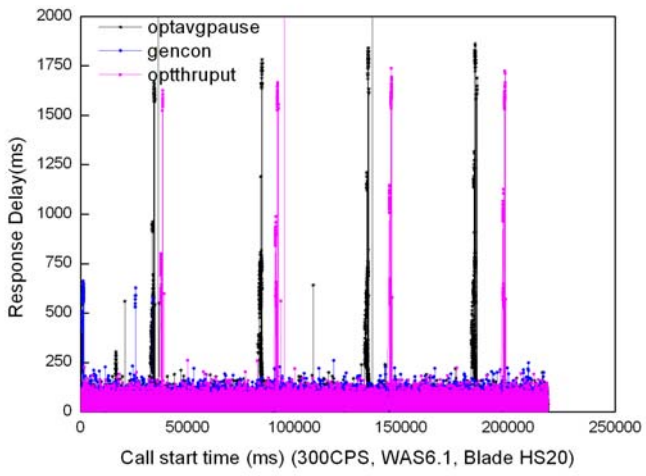
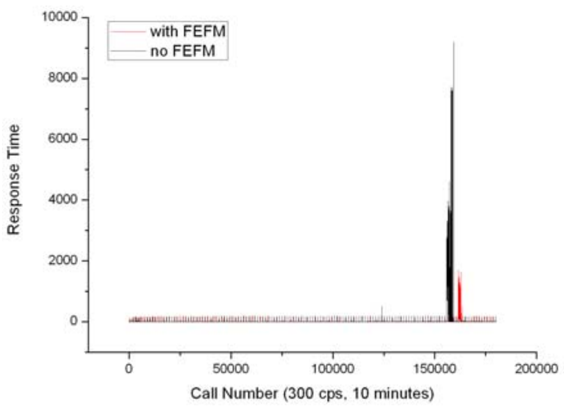
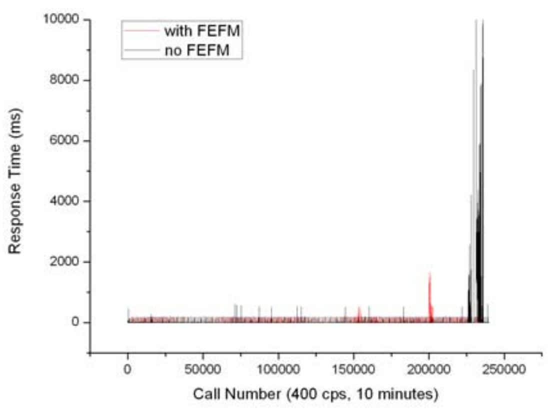


Fig. 6. The comparison of three GC policies

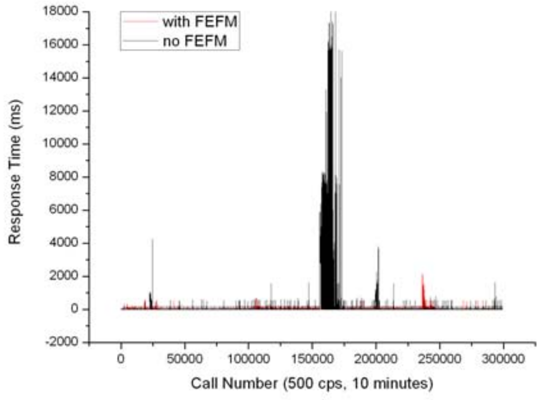
similar ones). Optthrughput tries to maximize the server's throughput. Optavgpause is optimized towards the average response time. Gencon is a generational plus concurrent GC policy. When JVM are specified to use Gencon, the Java heap is split into two areas, a new area (say "nursery") and an old area (say "tenured"). A Java object is first created in the nursery area, and then moved to the tenured area if it lives a long time. JVM only do GC in the nursery area. Only when the available space in the nursery area is not enough for a new object, it will perform a global GC that covers the old area. Since the nursery area is often set rather small, most GCs can be completed very quickly while global GCs happen periodically with long intervals. We compare the three kinds of GCs using 4-minute experiments win which the load is light (300 CPS). Fig. 6 shows the result. We can see that Gencon increases the response times slightly in ordinary running time, but Optthrughput and Optavgpause often perform GC and heavily increase the response times during the GC period. Gencon's global GC also produces response time spikes, but they show up much less frequently than the other two. Therefore, for the time-critical SIP applications, we should choose Gencon.



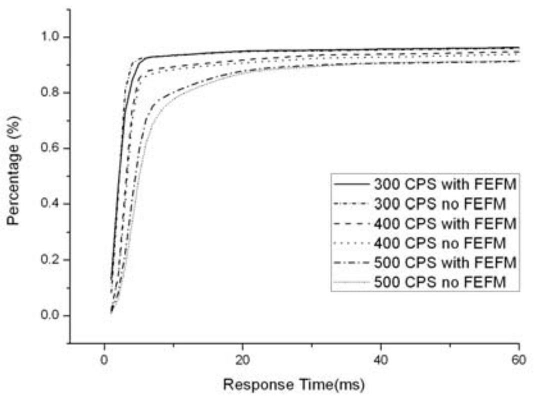
A) Response Time under 300CPS



B) Response Time under 400 CPS



C) Response Time under 500 CPS



D) CDF

Fig.7. The response time under various load and the cumulative distribution function graph

C. *FEFM running results*

Fig. 7 shows the response times of the session-establishing request using FEFM vs. not using FEFM. The loads are 300 CPS (light), 400 CPS (moderate), and 500 CPS (high), respectively. Some other statistics are given in Table IV. When not using FEFM, with 300 CPS, the server's average CPU utilization is 37% and no session fails. With 400 CPS, the server's average utilization is 53% and still no session failures. With 500 CPS the server's average utilization is 66% and 0.27% of the total sessions fail. Though some previous flow management works (for WEB servers) tried to fully utilize the server, with server's CPU utilization almost approaching 100% [7][6], it is not preferred in real cases. Commonly, service providers will not have their servers running at CPU utilization larger than 60%. Or else, they would add more resources [3]. Moreover, the session failure rate of 0.27% is much higher than the telecom service criterion, which is no more than 0.02% [16].

Under light load, the server runs well commonly. However, when global GC occurs, the requests are suspended and cumulated on the server, causing a huge spike, as shown in the Fig. 7 (A). When using FEFM, the spike is cut down greatly. The maximum response time drops nearly 8s. The standard error of all the response times is also reduced from 0.676 to 0.186, which means that the response times are more unified. When the server is under moderate load, FEFM also improves the response time's distribution. More requests' response times decreased than those without FEFM, as shown in Fig. 7 (D). Under high load, the server starts to reject the requests during global GC, which lasts longer than that of light or moderate load. There are 796 session failures. After using FEFM, the global GC's impact is alleviated. Both the session failures and the maximum response time decrease obviously.

IV. Related Works

There is no too much research work on SIP QoS management. [13] assesses the QoS of SIP-based mobile service, but does provide any method for improvement. In [8], an admission controller is built based on application specific policy information and call authorization status. However, it focuses on the interaction of SIP authorization process and admission controller, rather than how to use admission control to improve QoS. [9] uses virtual SIP links (VSL) to build an overlay, upon which QoS is described and guaranteed for SIP applications. This method demand the application should be developed compliant with the VSL specification.

Our work is more inspired by some techniques in web flow management. [7] first uses session-based admission control to achieve overload protection for web servers. However, it does not manage the response time. The response time prediction method has also been adopted by some previous work [6]. FEFM integrates these techniques together with some other novel techniques such as retransmission removal, twin-queue scheduling, and GC detection, building a well-performing SIP flow management system. There are also some systems managing response time based on runtime feedback [5][14]. We don't choose that because it is hard to be used for session-based flows in which a session-establishing request indicates a number of other requests in the next minutes or even hours.

V. Conclusion

The session-based message flow and strict SLA requirement are critical to a SIP application. To design a general flow management system for the SIP server, we must simultaneously consider the integrity of a session, the response time for a request within SLA limit, and the server's throughput. We propose a Front-end Flow Management system to tackle the challenges. The SIP server is protected from overload by throttling the session-establishing requests and retransmission removal. The new-session requests are scheduled based on the response time prediction to maximize the fraction of sessions conforming to SLA requirement. A twin-queue is used to get a trade off between SLA guarantee and the server's throughput.

From experiment results, we prove the straightforward methods mentioned above have positive effect on the SIP server. Our system decreases the response time on the worst situation, alleviates the impaction of GC, and improves the response time distribution. However, there are some further works remained to handle with, as discussion in the future works.

References

[1] J. Rosenberg, H. Schulzrinne "SIP: Session Initiation Protocol," IETF RFC 3261, June, 2002

[2] 3rd Generation Partnership Project, "Signaling flows for the IP multimedia call control based on SIP and SDP - stage 3 (release 5)," 3GPP TR 24.228 V5.7.0, December 2003.

[3] Communications with IBM customers in telecom industry

[4] SIPP. http://sourceforge.net/projects/sipp/

[5] Abhinav Kamra, Vishal Misra, and Erich M. Nahum "Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites" *In International workshop on Quality of Service (IWQoS)*, Jun. 2004, pp. 47-56.

[6] Josep M. Blanquer, Antoni Batchelli, Klaus Schauser, and Rich Wolski "Quorum: Flexible Quality of Service for Internet Services". *In NSDI 2005*, May 2005, pp. 159—174

[7] L. Cherkasova and P. Phaal "Session based admission control: a mechanism for improving performance of commercial web sites" *Proc. IEEE/IFIP Seventh International Workshop on Quality of Service (IWQoS)*, Jun. 1999

[8] Ana Elisa Goulart, Randal T Abler. "On the interaction of SIP and admission control : An inter-domain call authorization model for internet multimedia applications" *Proc. IEEE IPOM'05,* 0ct. 2005, pp. 9-18

[9] A. A. Kist, R. J. Harris "Using virtual SIP links to enable QoS for signalling" *Proc. IEEE ICON2003*, Sept. 2003, pp. 301-306

[10] JSR (Java Specification Requests) 116. http://jcp.org/en/jsr/detail?id=116

[11] Network grade of service parameters and target values for circuit-switched services in the evolving ISDN. ITU-T Recommendation E.721.

[12] Network grade of service parameters and target values for circuit-switched public land mobile services. ITU-T Recommendation E.771.

[13] Marc Portoles-Comeras, Marc Cardenete-Suriol, Josep Mangues-Bafalluy, Manuel Requena-Esteso, and Loïc Hersant. Experimental Assessment of VoIP Quality in MIPv6 and SIP Mobility Scenarios. ICC 2006.

[14] Matt Welsh and David Culler. Adaptive Overload Control for Busy Internet Servers. USITS 2003.

[15] JSR (Java Specification Requests) 32. http://jcp.org/en/jsr/detail?id=32.

[16] Telecom service criterion of China. http://www.gxca.gov.cn/policy/law_mii36ling.htm (in Chinese).