國立交通大學

網路工程研究所

碩 士 論 文

適用於可信賴 SIP 代理伺服器群之 高效率負載平衡策略

An Efficient Load Balancing Method for Dependable SIP Proxy Servers

研 究 生：鄭允榕

指導教授：王國禎　教授

中 華 民 國 九 十 六 年 六 月

適用於可信賴 SIP 代理伺服器群之高效率負載平衡策略

An Efficient Load Balancing Method for Dependable SIP Proxy Servers

研 究 生：鄭允榕　　　　Student：Yun-Jung Cheng

指導教授：王國禎　　　　Advisor：Kuochen Wang

國 立 交 通 大 學
網 路 工 程 研 究 所
碩 士 論 文

A Thesis
Submitted to Institute of Network Engineering
College of Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年六月

適用於可信賴 SIP 代理伺服器群之
高效率負載平衡策略

學生：鄭允榕　　　指導教授：王國禎 博士

國立交通大學網路工程研究所

摘 要

因為 VoIP 與多媒體服務的興起， SIP 協定被用來為網路使用者建立雙方對談或是多方視訊會議的連線。在建立連線的過程中， SIP 代理伺服器為雙方使用者轉交訊息，故為極重要的角色。隨著 VoIP 及多媒體服務的要求逐漸增加，採用單一 SIP 代理伺服器會產生效能下降及有單一失敗點的問題。為了解決上述的問題，我們設計一個前端為 $n + 1$ 個 (n 個使用中與一個備用) 分派器，以控制後端為 m 個 使用中 SIP 代理伺服器群的架構。但是分派器要如何將使用者傳遞過來的訊息平均分配到後端 SIP 代理伺服器群，而不會造成單一 SIP 代理伺服器負荷過載或長時間延遲是我們想要解決的問題。在本論文中，我們設計及實作對於 VoIP / 視訊會議等

應用提供一個可靠性的 SIP 群組架構，同時也提出一個基於 OpenAIS 的 SIP 負載平衡策略 (OSLB)，以平衡 SIP 代理伺服器群組間的負載，且當其中一個分配器或是 SIP 伺服器當機時，能將造成失敗的連線數減低。實驗的結果顯示 OSLB 比目前的 SIP load balancer 方法有相近的負載平衡值 (1.05 與 1.04)，但是因為 SIP 代理伺服器當機所造成的失敗連線數較之減少 82%。

關鍵詞：可信賴, 負載平衡, 分派器, SIP, SIP 代理伺服器。

An Efficient Load Balancing Method for Dependable SIP Proxy Servers

Student：Yun-Jung Cheng Advisor：Dr. Kuochen Wang

Department of Computer Science
National Chiao Tung University

Abstract

Because of the arising of VoIP and multimedia services, the Session Initiation Protocol (SIP) has been used to establish multimedia sessions which could be a simple two-way phone call or a collaborative video conference session between users on the Internet. In the procedure of establishing these sessions, a SIP proxy server plays an important role by forwarding SIP messages between users. Continued growth in VoIP and multimedia usages, using only one SIP proxy server may cause performance degradation, and has a single point of failure issue. In order to solve these problems, we design *m active* SIP proxy servers as a cluster in the backend, which are controlled by *n + 1* dispatchers (*n* active dispatchers plus one backup dispatcher) in the front end. But how to make a dispatcher distributes requests from users to one of the back-end SIP proxy servers without causing overloading or long delay is the load balancing issue that we also want to resolve. In this thesis, we have designed and implemented a dependable SIP-based clustered architecture for VoIP/Video conferencing applications, and also have proposed an efficient *OpenAIS-based SIP Load Balancing* strategy (OSLB) that can balance the proxy servers' load and reduce the number of failed calls when one of the dispatchers or one of the SIP proxy servers crashes Experimental results show that our OSLB is comparable to an existing work, SIP load balancer, in terms of load balance metric (1.05 vs. 1.04). However, our OSLB reduces the number of failed calls when a proxy server failed by 82% compared to the SIP load balancer.

Index Terms — dependable, load balancing, dispatcher, SIP (session initiation protocol), SIP proxy server.

Acknowledgements

Many people have helped me with this thesis. I deeply appreciate my thesis advisor, Dr. Kuochen Wang, for his intensive advice and instruction. I would like to thank all the classmates in the *Mobile Computing and Broadband Networking Laboratory* for their invaluable assistance and suggestions. The support by the National Science Council under NSC94-2219-E-009-023 and NSC96-2219-E-009-012 is also grateful acknowledged.

Finally, my family deserves special mention. This thesis is dedicated to them for their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays people not only get information by surfing on the Internet, but also obtain multimedia services through the Internet. The Session Initiation Protocol (SIP) is a signaling protocol developed by IETF to set up, modify, and tear down sessions between end-users over the Internet [1].

Because of continued growth of VoIP and multimedia services, using only one SIP proxy server, which is responsible for multimedia session establishments between network end-points, to handle many session set up requests at the same time may cause the overloading problem and the setup procedure might take too long and result in failures.

Several SIP proxy servers can form a cluster in the back-end controlled by a dispatchers in the front-end can ease these problems. In the clustered architecture, there are three design issues need to be concerned:

- Single point of failure problem of the dispatcher.
- Health condition monitoring of back-end SIP proxy servers.
- Load balancing strategy for SIP proxy servers.

To resolve these issues, we propose a dependable system architecture and an efficient SIP load balancing strategy based on OpenAIS [2] to prevent performance degradation and failure problems.

The thesis is organized as follows. Chapter 2 gives overview of SIP, and related software including OpenAIS and OpenSER [3]. Chapter 3 reviews several existing clustered load balancing methods. Chapter 4 presents the proposed system architecture and load balancing strategy in detail. Then experimental results are evaluated in Chapter 5. Finally, Chapter 6 concludes the thesis with concluding remarks and future work.

Chapter 2

Preliminaries

2.1 Session Initiation Protocol

The SIP is an application layer signaling protocol over IP networks. Fig. 1 shows a typical SIP-based network configuration [4]. The SIP-based network is made up by the following basic components [5][6].

1. *SIP user agent*: A network end-point which initiates or terminates multimedia sessions.

2. *SIP registrar*: A SIP user agent registers its SIP URI (uniform resource identifier), contact information and dynamically updates these data via SIP messages to the SIP registrar.

3. *SIP proxy server*: It routes the SIP requests from one SIP user agent to another.

Fig. 2 is a SIP call setup procedure. The caller issues an INVITE message to initiate the call setup procedure. The SIP proxy server would forward message to the callee, and returns the response back to the caller. Until the caller receives an OK message and responds an ACK message to the callee, a session is established. And issuing a BYE message would end the session.

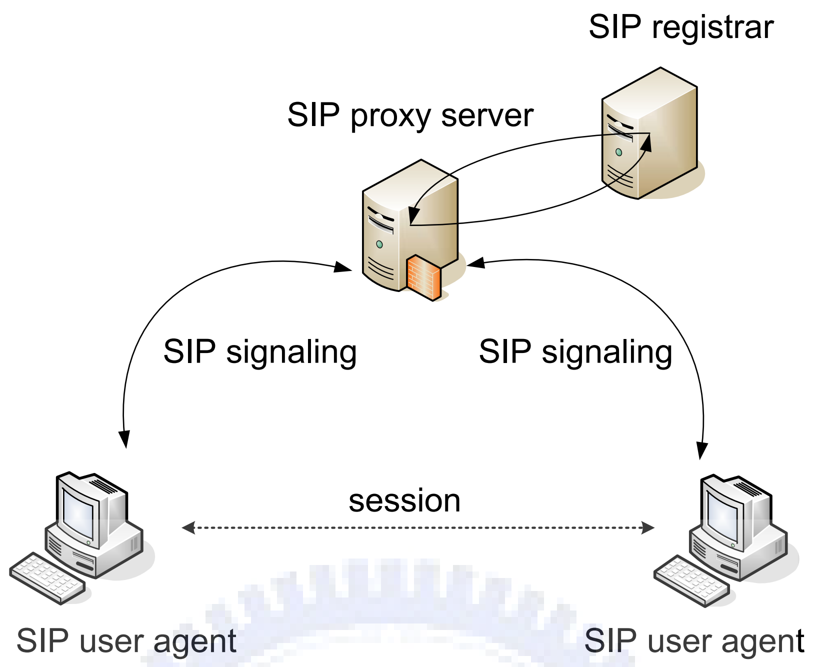


Fig. 1. Typical SIP-based network configuration [4].

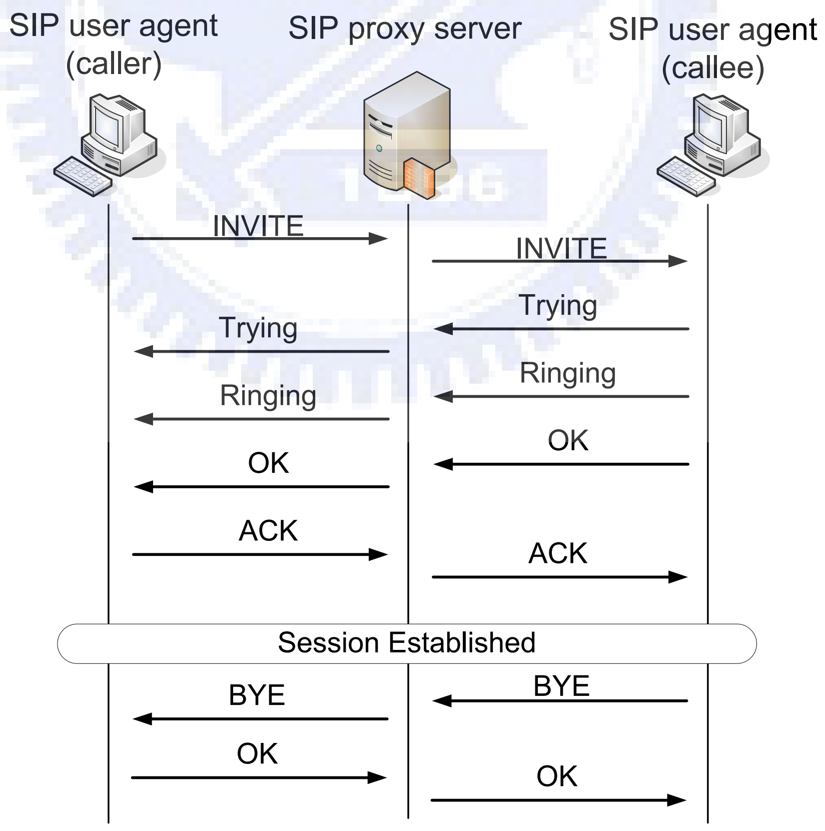


Fig. 2. SIP call setup procedure.

2.2 OpenAIS

OpenAIS (open application interface specification) [2] is an implementation of Service Availability Forums API Specification. The main purpose of OpenAIS is to provide high availability and build a dependable network. The API consists of several parts, including AMF (Availability Management Framework), CKPT (checkpointing) and etc. AMF is the most important component, which is in charge of failover between the active and the backup servers. It monitors the condition of the active server and informs the backup server to become active when the active one fails. CKPT is used to backup information from the active server to the backup server. Fig. 3 shows the architecture of servers using OpenAIS.

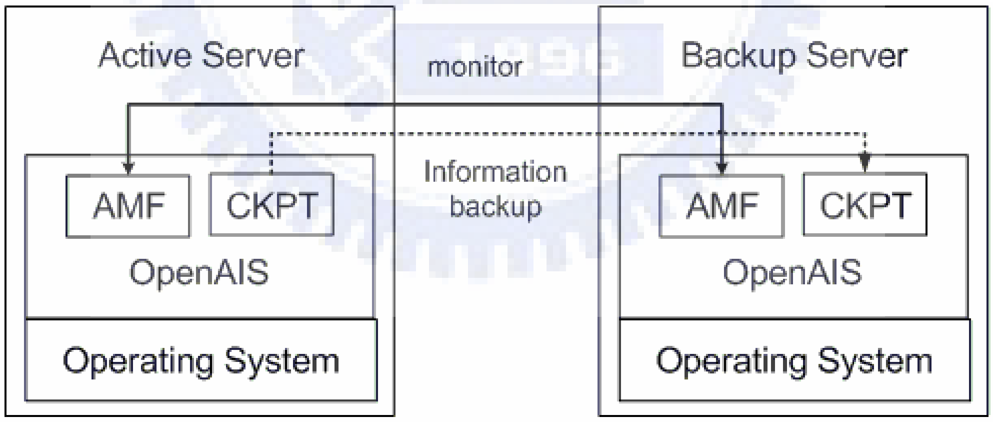


Fig. 3. Architecture of servers using OpenAIS.

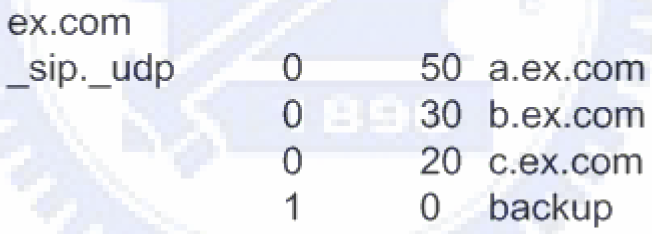
2.3 OpenSER

OpenSER is a mature and flexible open source SIP server. It can be used on systems with limited resources as well as on carrier grade servers, scaling to up to thousands call setups per second. And it aims to be a collaborative project of its users to develop secure and extensible SIP server to provide modern VoIP services [3]. OpenSER can perform not only the function of SIP proxy servers and also include the functions of SIP registrars and SIP redirect servers. It also has modules to support more functions, such as SNMP and high availability functions. The DISPATCHER module is designed to dispatch user agents' requests to SIP proxy servers. That is, it can connect to SIP proxy servers that form as a cluster, and dispatch the requests coming from SIP user agents to selected SIP proxy servers.

Chapter 3

Related Work

Cisco's architecture [5] used DNS SRV (Service) records [7], which associate the name of a service with IP addresses of devices, priority and weight, to support server failover and load balancing. Fig. 4 gives an example DNS SRV records. The selection algorithm for load balancing could be round-robin or weighted random. The system administrator achieves load balancing by configuring the priority and weight of static route or DNS SRV entries.



```
ex.com
_sip._udp      0      50  a.ex.com
               0      30  b.ex.com
               0      20  c.ex.com
               1       0  backup
```

Fig. 4. DNS SRV records

A IP telephony architecture was proposed in [8][9], which is also based on DNS SRV records [7] and NAPTR (Naming Authority Pointer) [10] to support server failover and load sharing [8]. Clients can use weighted randomization to achieve the distribution recorded in DNS SRV records. They used two stages architecture to provide scalability, reliability and load sharing. Fig. 5 shows the two-stage reliable and scalable architecture. First stage servers act like dispatchers. Clients get their IP addresses from DNS SRV records to decide which dispatcher to connect. After dispatchers receive requests from SIP user agents and then forward to one of the second stage server group based on the destination of user identifier.

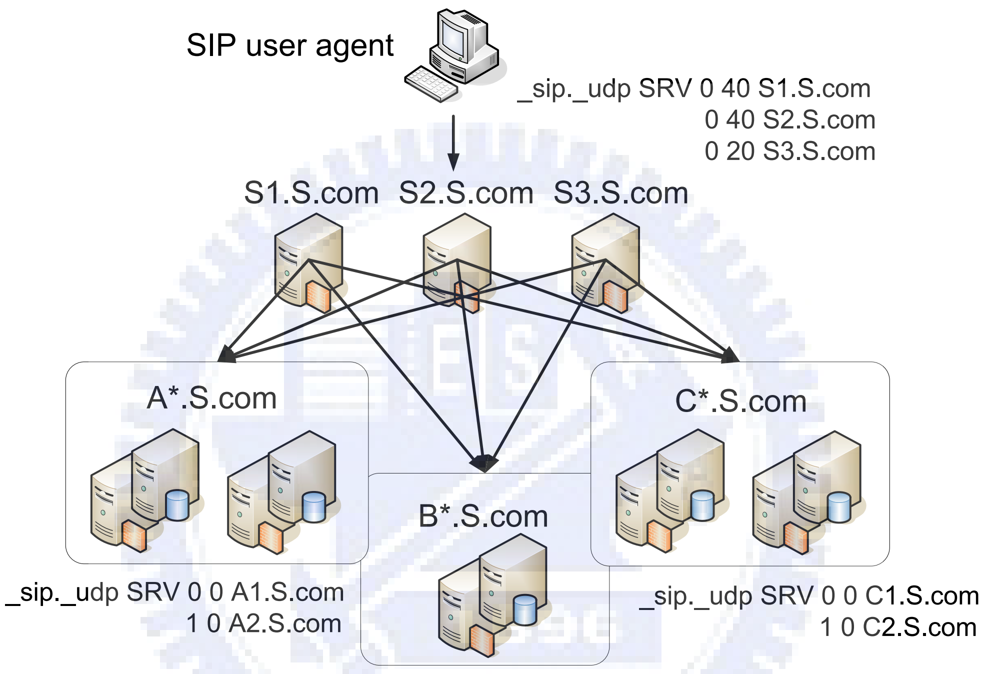


Fig. 5. Two-stage reliable and scalable architecture [8][9].

In [11], it proposed a strategy for load balancing in SIP networks. It uses a directory server to maintain a table of the status of the backend media servers. Table 1 shows the table it proposed. The first column shows the URL for the media server, and the second column maps the URL to a real IP address. Occupation indicates the percentage of the capability being occupied and distance is used to express the communication cost.

Table 1. Table maintained in the directory server.

URL of backend media servers	IP address	Occupation	Distance	Score
TTS@net	10.0.1.1	40	1	40
TTS@net	10.0.1.2	20	10	200

Score = Occupation * distance

When a soft switch, which acts as a SIP proxy server, receives a request from a SIP user agent, the soft switch would look up the table first, finds the matching virtual name of the server, calculates the score, and then forwards the request to the server which has the minimum score. Fig. 6 shows the SIP call center system with a directory server.

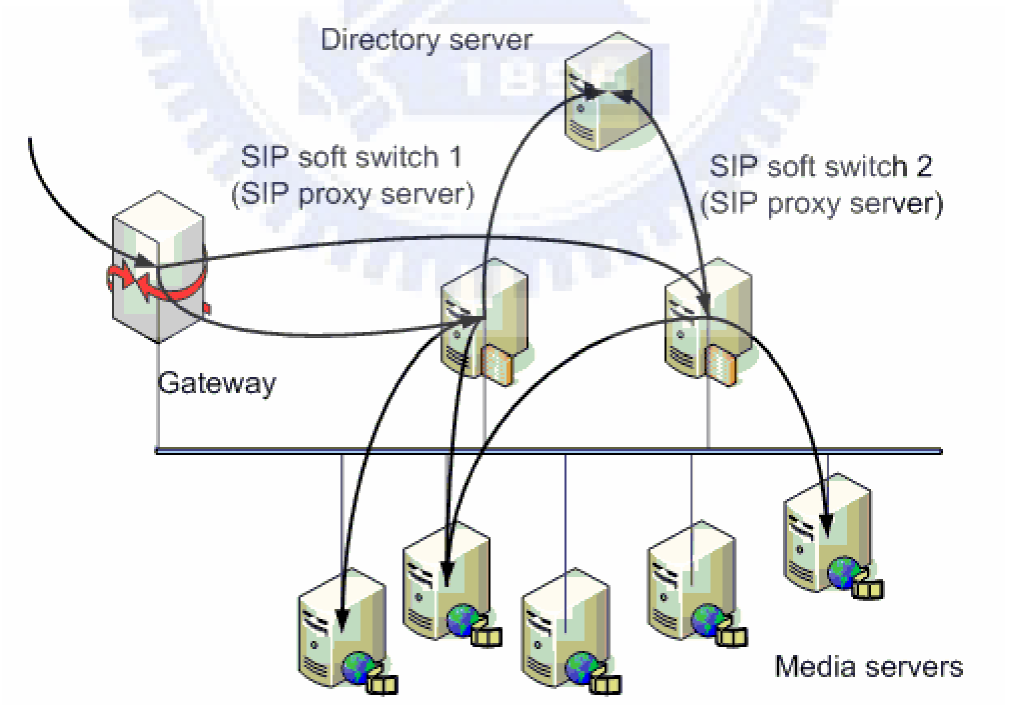


Fig. 6. A SIP call center system with directory server [11].

The SIP load balancer [12][13] receives requests on one port, then assigns an ingress SIP proxy server dynamically to each transaction. And the traffic load can be balanced over a pool of SIP proxy servers based on the real-time demand for services. It uses "*Stickyness*", which is computed by hashing one of either *CallID, To, From* or *SIP URI*. Therefore, the same entry of "*Stickyness*" would be handled by the same SIP proxy server. Fig. 7 shows the architecture of SIP load balancer.

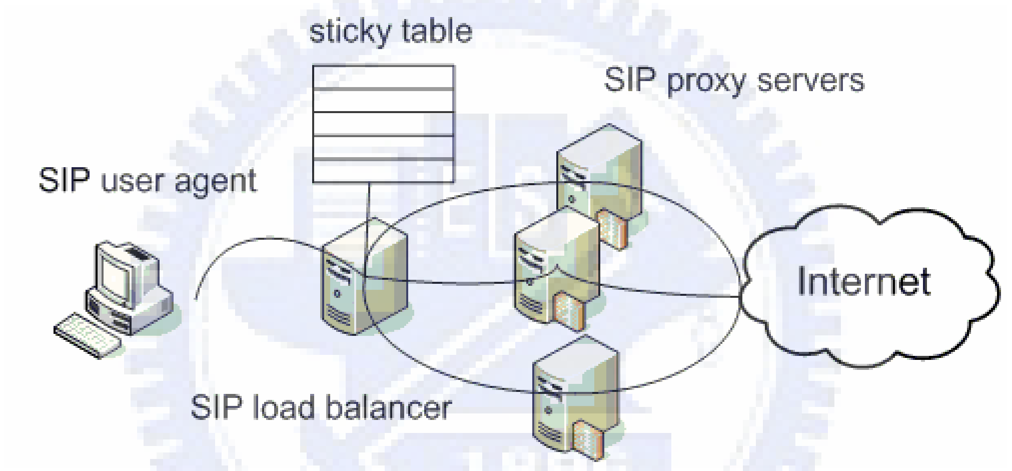


Fig. 7. Architecture of SIP load balancer.

We summarize the above existing approaches and make a qualitative comparison with our proposed method (OSLB), as shown in Table 2. Both Cisco's architecture [5] and two-stage architecture [8][9] are based on DNS SRV. Clients need to obtain these records to handle the dispatcher failover problem and achieve the load balancing of proxy servers. The directory server [11], SIP load balancer [12][13] and proposed OSLB do not involved the DNS SRV records and the client side. The directory server updates the table (like Table 1) based on the status information of media servers, which is announced every five minutes. This periodical announcement can also be used to determine if the corresponding server is still alive or not. The SIP load balancer regularly (around five seconds) issues an OPTION message to the SIP proxy servers to check if they are alive or not. The proposed OSLB, which will be introduced in detail in the next chapter, gets the health condition and CPU loading information of the SIP proxy servers every one second. Since that the SIP load balancer and the OSLB are both not client-involved and doing regular check every few seconds, which is much less than that of the directory server, so we will evaluate these two methods in Chapter 5.

Table 2. Qualitative comparison of existing approaches.

Approach	Cisco's [5] architecture	Two-stage architecture [8][9]	Directory server [11]	SIP Load balancer [12][13]	OSLB (proposed)
DNS and Client involved	Yes	Yes	No	No	No
Server failover mechanism	DNS SRV or VRRP	DNS SRV	Periodically announcement	OPTION message	OpenAIS
Server failover time	Medium to long	Medium to long	Long (5 min)	Medium (10-15 sec)	Short (< 5 sec)
Load balancing mechanism	DNS SRV or Round-Robin	Hash (user identifier)	Score	Stickyness	OpenAIS and OpenSER

Chapter 4

Design Approach

4.1 Proposed Dependable System Architecture

As mentioned in Chapter 1, a clustered architecture needs to address three issues, including the single point of failure problem of front-end dispatcher, health condition and CPU loading of back-end SIP proxy servers. The proposed dependable system architecture is as shown in Fig. 8. In order to provide reliability and scalability, dispatchers adopt an n + 1 redundancy model, which means one backup dispatcher for n active dispatchers. Each active dispatcher handles requests from regional SIP user agents. SIP proxy servers adopt an m + 0 redundancy model. We use the modules provided by OpenSER [3] to set up dispatchers and SIP proxy servers. Using at least two dispatchers can prevent the single point of failure problem. The frond-end dispatcher plays an important role as a monitor and a controller of the back-end proxy servers. It needs to monitor the health condition and CPU loading of each SIP proxy server. We use an open source called OpenAIS [2], which is a middleware for high availability, to do these jobs.

4.2 Efficient SIP Load Balancing Strategy

The load balancing strategy we propose mainly to prevent performance degradation or long connection delay due to proxy server failure or overloading. Since each SIP proxy server periodically transmits its heartbeat count and CPU loading to a dispatcher through OpenAIS [2], the dispatcher can stop sending connection requests to a failed or heavy loading SIP proxy server.

The basic requests distribution method is round-robin. We adjust the distribution method by CPU loadings of SIP proxy servers. We set up a fixed upper bound α of CPU loading as a threshold. When the CPU loading of the SIP proxy server exceeds α, the dispatcher would setup a lower bound β for that SIP proxy server and stop forwarding any request to it until its CPU loading is below β. β is computed as follows:

$$\beta = \left(\frac{\sum_{i=1}^{n} current\_CPU\_loading_i}{n} \right)$$

where *n* is the number of SIP proxy servers

If the loadings of all SIP proxy servers are above α, the dispatcher starts to drop incoming INVITE messages until the CPU loading of at least one SIP proxy server is below α. The reason to drop INVITE messages is that users can usually tolerate call cancellation due to system busy, but cannot stand that a call session has been canceled in the middle of the call setup procedure.

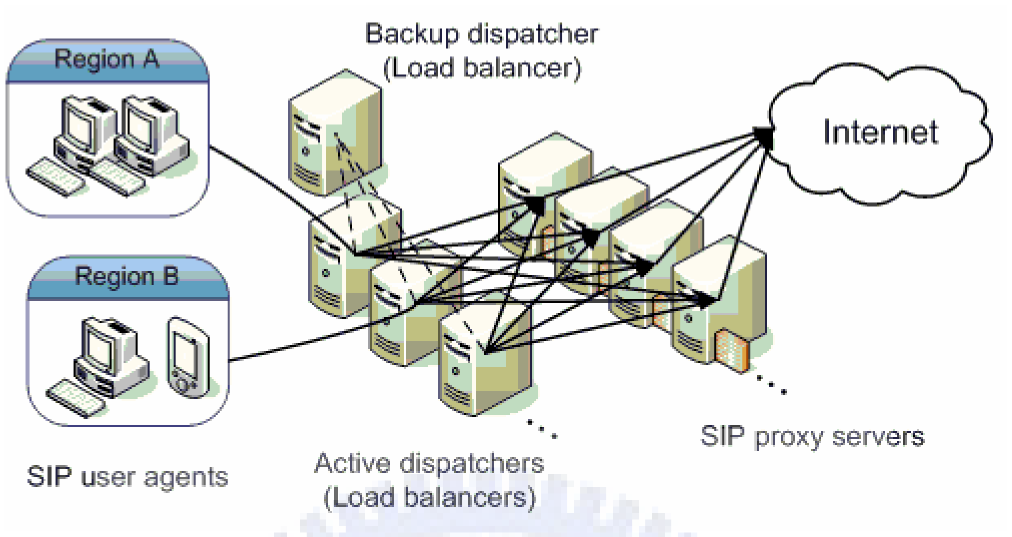

Region A
Backup dispatcher (Load balancer)
Internet
Region B
SIP user agents
Active dispatchers (Load balancers)
SIP proxy servers

Fig. 8. Proposed dependable SIP-based clustered architecture.

4.3 Implementation

In this part, we describe how to implement the above strategy with OpenAIS. As mentioned in Chapter 2, AMF is in charge of failover between the active and backup servers. When the active dispatcher fails, AMF would notify the standby dispatcher to take over the jobs and IP address of the active dispatcher. Therefore, the SIP user agents can keep issuing the requests to the IP address they knew, and the service will not be interrupted since the backup dispatcher takes over the jobs. Between dispatcher and SIP proxy servers, SIP proxy servers use CKPT to transmit heartbeat count and CPU loading to the dispatcher. The dispatcher checks the data every time it gets, and makes decision of keep or stop forwarding data to the SIP proxy servers. Fig. 9 shows the flow of a dispatcher in OSLB.

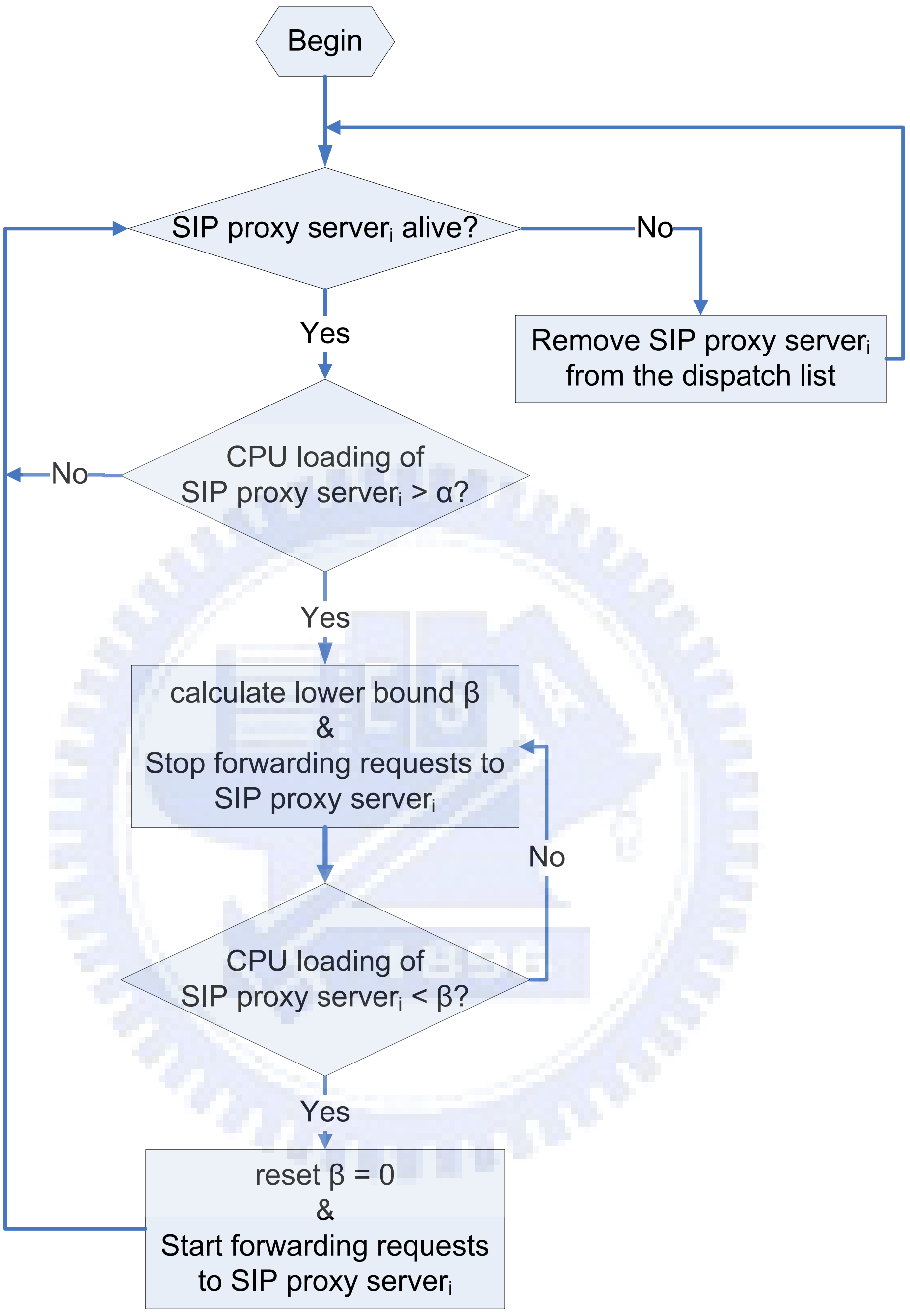

Begin
SIP proxy server$_i$ alive?
No
Remove SIP proxy server$_i$ from the dispatch list
Yes
CPU loading of SIP proxy server$_i$ > α?
No
Yes
calculate lower bound β & Stop forwarding requests to SIP proxy server$_i$
No
CPU loading of SIP proxy server$_i$ < β?
Yes
reset β = 0 & Start forwarding requests to SIP proxy server$_i$

Fig. 9. The operation flow of a dispatcher in OSLB.

Chapter 5

Performance Evaluation

To evaluate the performance of our approach, we set up a test architecture, as shown in Fig. 10, which include 1 + 1 dispatchers and two active SIP proxy servers (2 + 0). We also used SIPp [14], which is a free open source test tool, as a SIP traffic generator and analyzer. It includes some SIP user agent test scenarios, which were provided by SIPStone [15], a benchmark tool for evaluating SIP proxy server performance. It establishes and releases multiple calls using INVITE and BYE messages. We used default UAC (user agent client) and UAS (user agent server) scenarios which have already been implemented in SIPp to test the proposed design approach. The test scenario for call setup has been shown in Fig. 2. And in order to obtain the number of failed calls, we turned off the retransmission mechanism in SIPp while doing the following tests.

The flowing five test cases are evaluated:

Case A	All dispatchers and SIP proxy servers work well.
Case B	One dispatcher fails.
Case C	One SIP proxy server fails.
Case D	One dispatcher and one proxy server fail.
Case E	Run a background application with high CPU loading in a selected SIP proxy server.

The first four cases are used to evaluate if dispatcher and/or proxy server failures affect load balancing. And the last case is to see if the OSLB can handle a surging load.

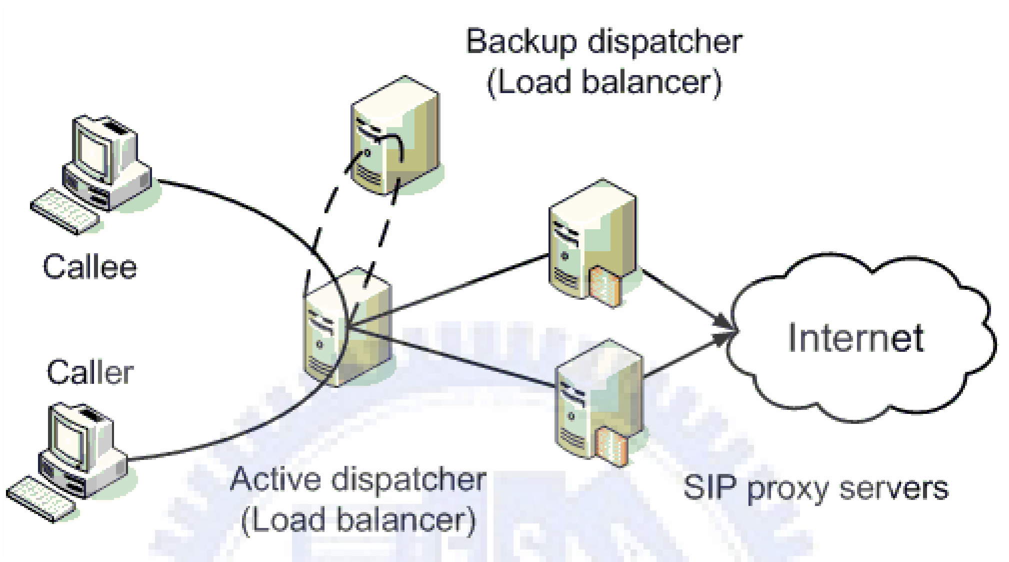


Fig. 10. Test architecture for OSLB.

We used a metric, *Load Balance Metric* (LBM) [16], as follows, to analyze the performance of our load balancing method:

$$LBM = \frac{\sum_{j=1}^{m}\left(\frac{peak\_load_j}{\left(\sum_{i=1}^{n} load_{i,j}\right)/n} \times \frac{\left(\sum_{i=1}^{n} load_{i,j}\right)}{n}\right)}{\sum_{j=1}^{M}\sum_{i=1}^{n} load_{i,j}/n} = \frac{\sum_{j=1}^{m} peak\_load_j}{\sum_{j=1}^{M}\sum_{i=1}^{n} load_{i,j}/n}$$

where

$load_{i,j}$ is the load of server i at the jth sampling point;

$peak\_load_j$ is the highest load on any server at the jth sampling point;

n is the number of SIP proxy servers.

We compare the proposed OSLB with the SIP load balancer [12][13], which is also not client and DNS involved, and its failover detection time is much less than that of the directory server [11]. Table 3 shows the test environment and Table 4 is related test parameters.

Table 3. Test environment.

Environment	**Value**
CPU (of Virtual machines)	AMD Athlon 64 3000+
Memory (of Virtual machines)	2 Gigabyte
CPU (of SIPp)	Intel Core 2 T5500
Memory (of SIPp)	512 Megabyte

Table 4. Test parameters.

Parameter	**Value**
Call rate (default)	100 calls per second
Call rate (failover)	50, 100, 150, 200 calls per second
Call duration	3 minutes
Call limit	10000 calls
Parameters in SIP load balancer	**Value**
Sticky table size	100000
Sticky expire time	32 seconds

In the following, the experimental results of each case are discussed:

Case A: All dispatchers and SIP proxy servers work well.

Fig. 11 shows CPU loading of both SIP proxy servers of OSLB for case A, and Fig. 12 shows CPU loading of both SIP proxy servers of the SIP load balancer for case A. We can see that CPU loading of SIP proxy servers of the OSLB is a little higher than the SIP load balancer. The reason is that AMF, provided by OpenAIS, periodical transmits messages between each server to see if any server is down. And we also used CKPT to transmit data to the dispatcher. However, LBM of OSLB and SIP load balancer are pretty close, LBM of OSLB is 1.05 and LBM of SIP load balancer is 1.04.

Case B: One dispatcher fails

Fig. 13 shows the CPU loading of both SIP proxy servers of OSLB for case B. The active dispatcher failed at the 60th second of the testing period and both SIP proxy servers are not affected by the failure of the active dispatcher. LBM of OSLB remains 1.05 as case A. We cannot evaluate the SIP load balancer in this case since it does not support dispatcher redundancy, and it needs a router or gateway to resolve this problem.

Case C: One SIP proxy server fails.

Fig. 14 shows the CPU loading of both SIP proxy servers of OSLB for case C, and Fig. 15 shows the CPU loading of both SIP proxy servers of the

SIP load balancer for case C as well. SIP proxy server 1 failed at the 120th second of the testing duration. From Fig. 14, we found that the CPU loading of SIP proxy server 2 has sudden decreasing after SIP proxy server 1 failed. The reason is that after the dispatcher detects the failure of SIP proxy server 1, it takes few seconds to remove the failed SIP proxy server from the dispatch list. During this short duration, some requests are still forwarded to the failed SIP proxy server 1 and causing call blocking and call failure problems. After the dispatcher removes failed SIP proxy server from the list, CPU loading of SIP proxy server 2 would arise and successful processing incoming requests. In Fig. 15, after SIP proxy server 1 failed, the CPU loading of SIP proxy server 2 has higher variation than before. In Fig. 16, we also evaluated the number of failed calls with different call rates when one of the SIP proxy servers fails for case C. We found that our approach encountered much fewer failed calls than the SIP load balancer, and have 82 % improvement. This is because that the SIP proxy server failover time of our approach is lower.

Case D: One dispatcher and one proxy server fail.

The active dispatcher failed at the 60th second, and SIP proxy server 1 failed at the 120th second of the testing duration. Fig. 17 shows the CPU loading of both SIP proxy servers of OSLB for *Case D*. Its like the combination of *Case B* and *Case C,* the CPU loading of SIP proxy servers are not affected by the failure of the dispatcher but would be affected by the failure of SIP proxy server.

Case E: Run a background application with high CPU loading in a selected SIP proxy server.

In this case, we ran a background application with high CPU loading in SIP proxy server 1 for one minute during the test period. From Fig. 18 and Fig. 19, we observed that CPU loading of SIP proxy server 2 of both OSLB and the SIP load balancer arising while SIP proxy server 1 has a high CPU loading background application. And Fig. 20 shows the number of failed calls in this test case for case E. The OSLB stopped sending requests to the overloading SIP proxy server before the call-blocking problem became serious. Therefore the proposed OSLB has smaller number of failed calls than the SIP load balancer, and it has 44 % improvement.

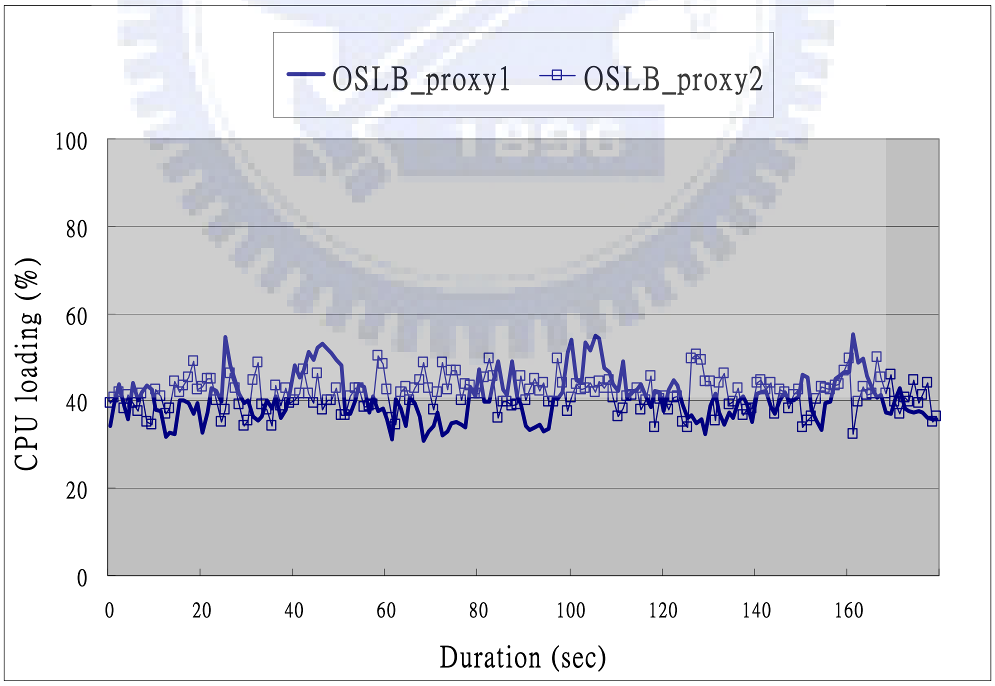


Fig. 11. Case A: CPU loading of OSLB with all servers work well.

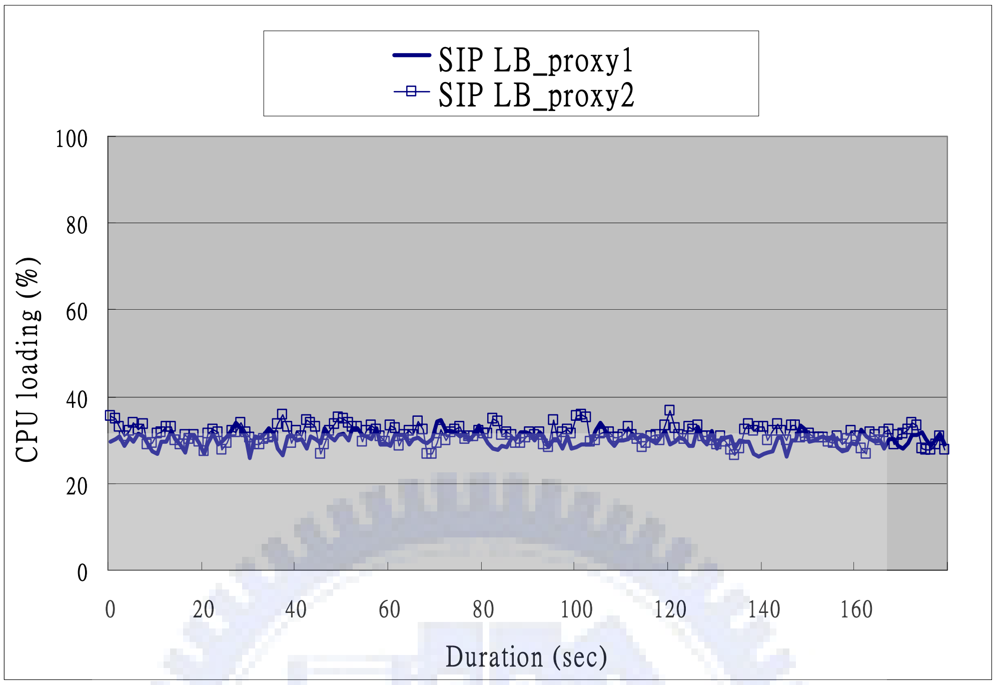


Fig. 12. Case A: CPU loading of SIP load balancer with all servers work well.

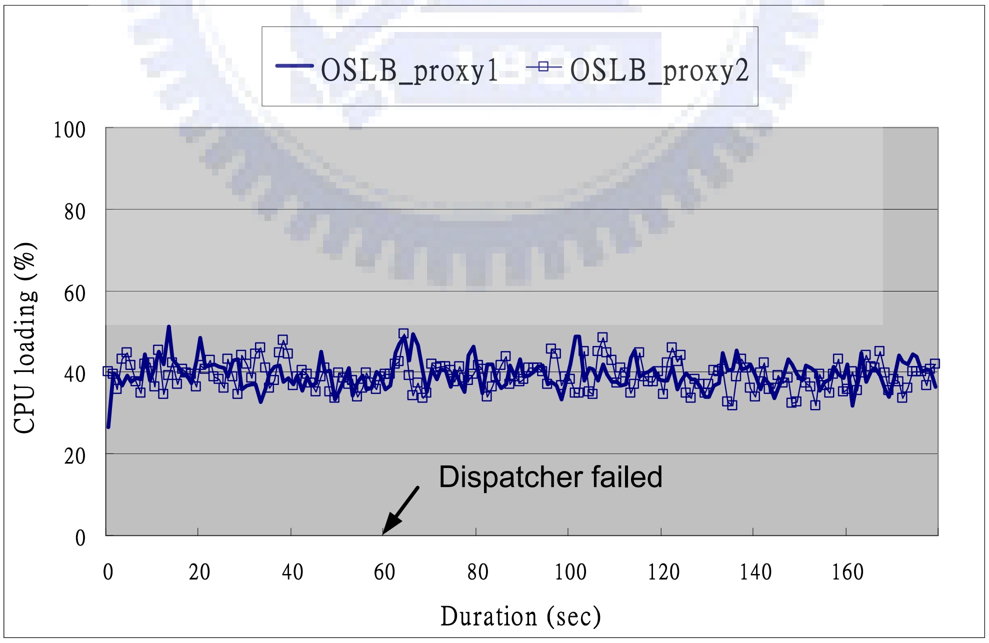


Fig. 13. Case B: CPU loading of OSLB with the failure of one dispatcher.

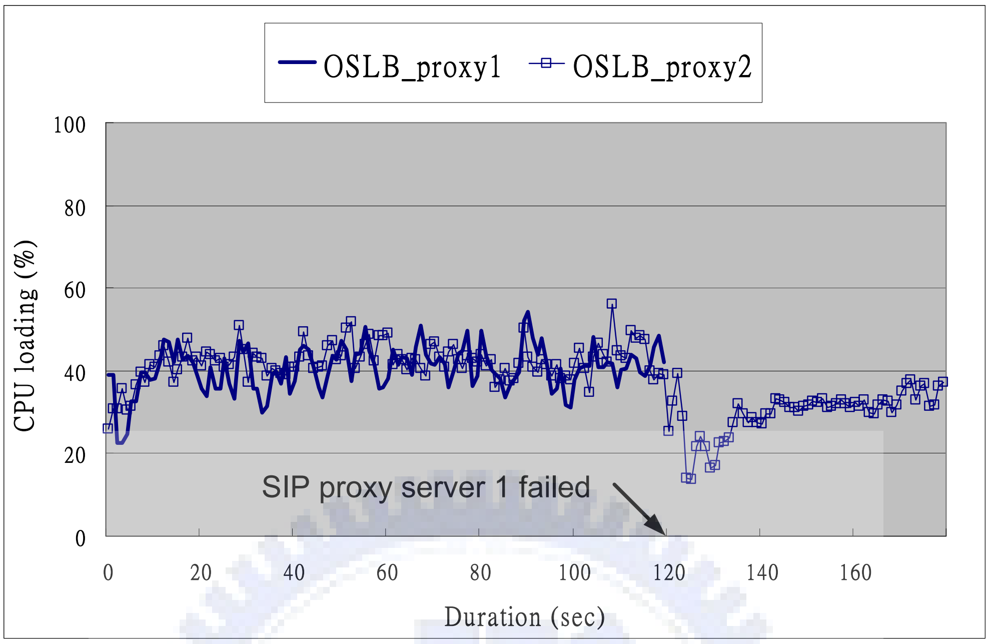


Fig. 14. Case C: CPU loading of OSLB with the failure of one SIP proxy server.

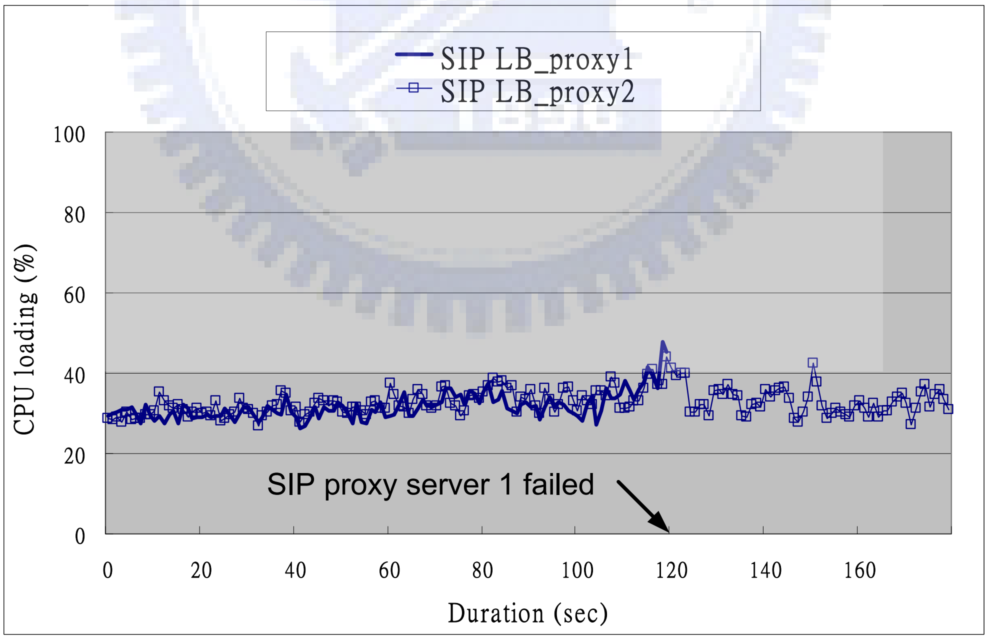


Fig. 15. Case C: CPU loading of SIP load balancer with the failure of one SIP proxy server.

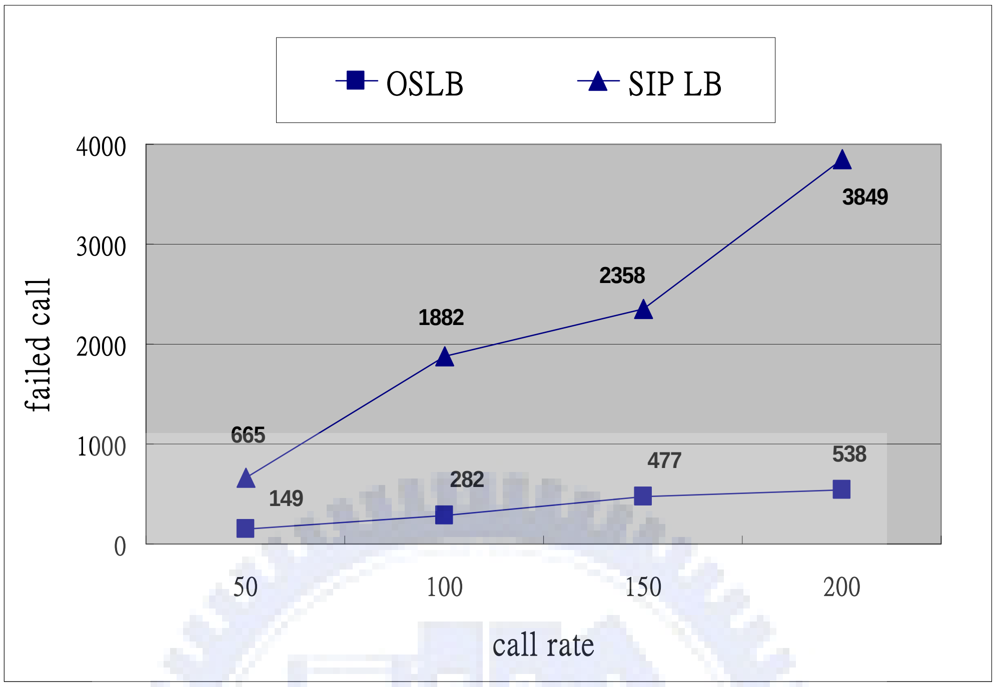


Fig. 16. Case C: Number of failed calls under different call rates with the failure of one SIP proxy server.

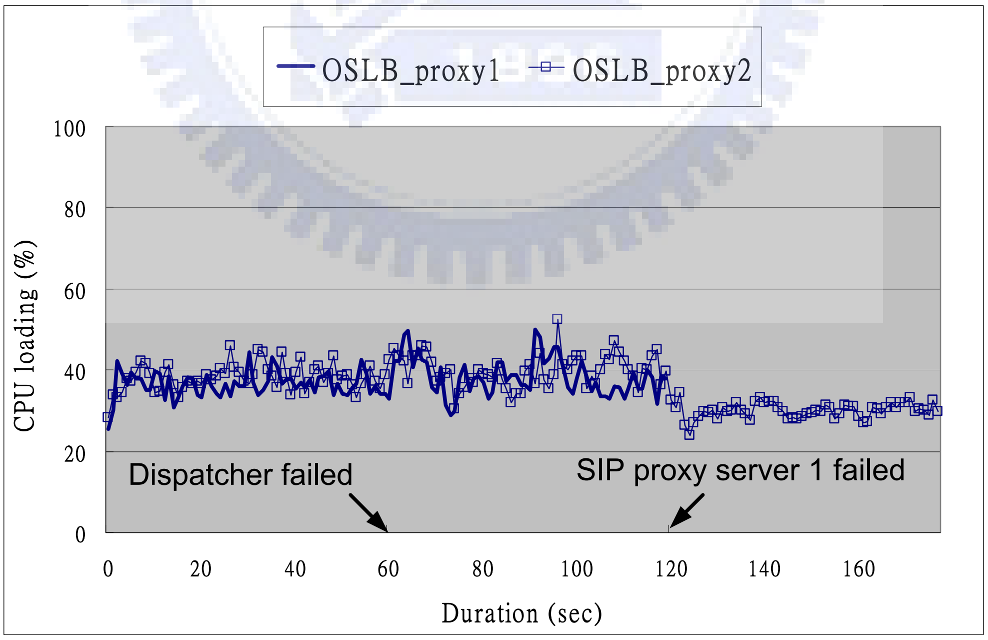


Fig. 17. Case D: CPU loading of OSLB with the failure of one dispatcher and one SIP proxy server.

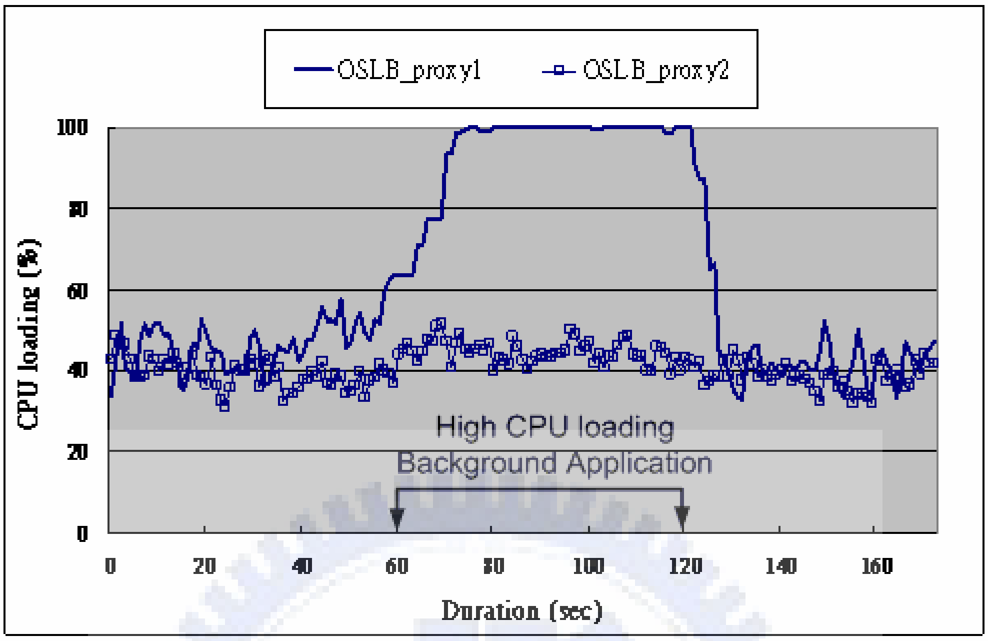


Fig. 18. Case E: CPU loading of OSLB with one SIP proxy server running a high CPU loading background application.

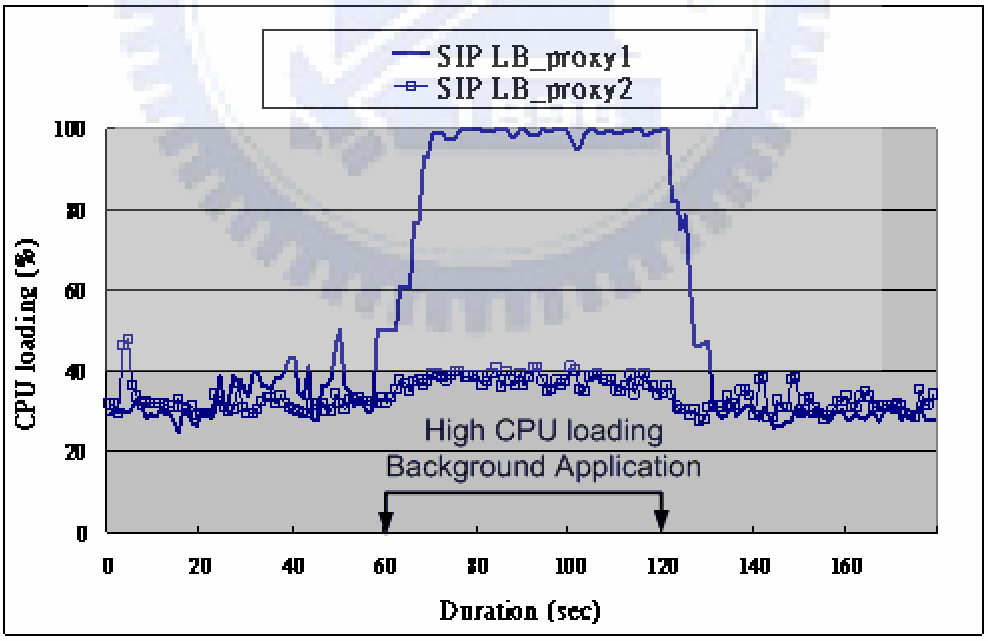


Fig. 19. Case E: CPU loading of SIP load balancer with one SIP proxy server running a high CPU loading background application.

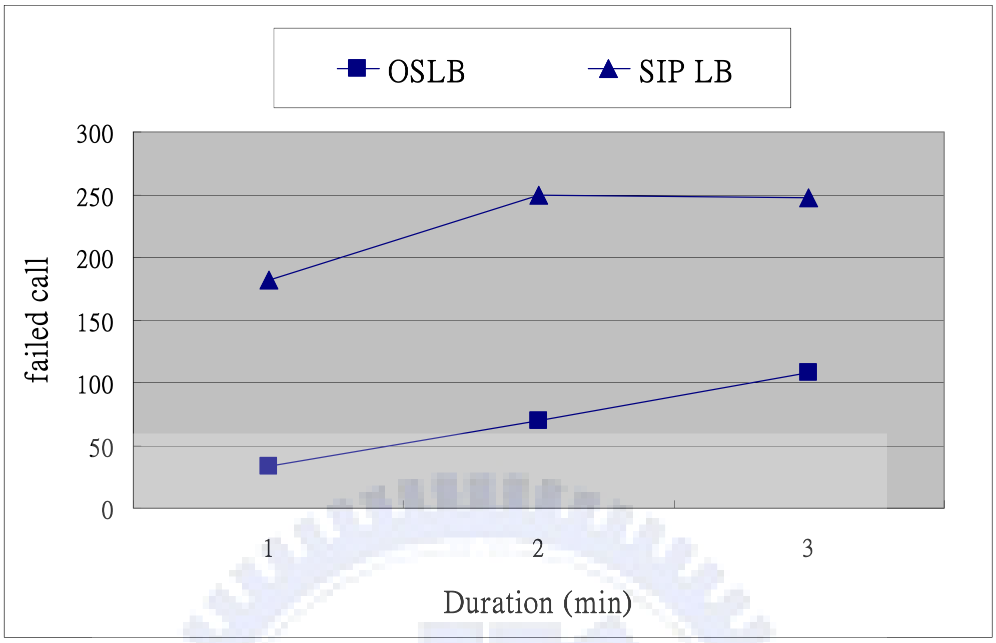


Fig. 20. Case E: Number of failed calls with one SIP proxy server running a high CPU loading background application.

Chapter 6

Conclusions and Future Work

6.1 Concluding Remarks

In the SIP clustered architecture, how fast a dispatcher can respond to the failure of a SIP proxy server and to the heavy CPU loading of SIP proxy servers is a very important issue. If the dispatcher takes long time to respond, the call blocking problem may occur and result in increased number failed calls. In this thesis, we have designed and implemented a dependable SIP-based clustered architecture for VoIP/Video conferencing applications. We have also designed and implemented an efficient Open-AIS based load balancing (OSLB) strategy for this architecture. The experimental results have shown that our OSLB is comparable to an existing work, SIP load balancer, in terms of load balance metric (1.05 vs. 1.04); however, our OSLB reduces number of failed calls when a proxy server failed by 82% compared to the SIP load balancer.

6.2 Future Work

Currently, we set α (upper bound value of overloading) as a fixed value. In the future, we can dynamically adjust α according to the incoming call rate, network condition, and the capacity of the SIP proxy servers. In addition, multimedia services provided by media servers will be demanded by more and more people, and the connections between the SIP user agents and media servers have QoS requirements. Load balancing between clustered media servers, which affects the QoS provided to the SIP user agents, is a significant research issue. In the future, we can extend our load balancing strategy to the clustered back-end media servers controlled by the front-end SIP proxy servers.

Bibliography

[1] A.B. Johnston, *SIP: understanding the Session Initiation Protocol*. 2nd edition, Artech House, 2004.

[2] “OpenAIS-standard-based cluster framework,” http://developer.osdl.org/dev/openais/

[3] “OpenSER-the open source SIP server,” http://www.openser.org/

[4] M. Ohta, “Overload control in a SIP signaling network,” *Transactions on Engineering, Computing and Technology*, Volume 12, March 2006.

[5] Cisco Inc., “High availability solutions for SIP enable voice-over-IP networks,” http://www.cisco.com/en/US/tech/tk652/tk701/technologies_white_paper09186a00800a9818.shtml

[6] Y.B. Lin and A.C. Pang, *Wireless and mobile All-IP networks*. John Wiley, 2005.

[7] A. Gulbrandsen, P. Vixie and L. Esibov, “A DNS RR for specifying the location of services (DNS SRV)," RFC 2782, Internet Engineering Task Force, Feb. 2000.

[8] K. Singh and H. Schulzrinne, “Failover, load sharing and server architecture in SIP telephony,” *Computer Communications*, Volume 30, Issue 5, pp. 927-942, March 2007.

[9] K. Singh and H. Schulzrinne, “Failover and load sharing in SIP telephony,” *Technical Report*, Department of Computer Science, Columbia University.

[10] M. Mealling and R. W. Daniel, “The naming authority pointer (NAPTR) DNS resource record," RFC 2915, Internet Engineering Task Force, Sept. 2000.

[11] M. Liang, Y. Guo, Y. Guo and L. Zhang; “Cluster service for streaming media on application layer,” in *Proceedings of the 7th International Conference on Signal Processing*, Volume 3, pp. 2494-2497, August 2004.

[12] "SIP load balancer," http://www.vovida.org/applications/downloads/loadbalancer/

[13] http://www.xtremenetworks.biz/ip-pbx.htm

[14] "SIPp-a free open source test tool / traffic generator for the SIP protocol," http://sipp.sourceforge.net/

[15] "SIPStone-Benchmarking SIP server performance," http://www.sipstone.org/

[16] R.B. Bunt, D.L. Eager, G.M. Oster, and C.L. Williamson, "Achieving load balance and effective caching in clustered Web servers," in Proc. 4th International Web Caching Workshop, 1999.I.F. Akyildiz and X. Wang, "A survey on wireless mesh networks," in *IEEE Communications Magazine*, Volume 43, pp. 23 – 30, Sept. 2005.