RESEARCH ARTICLE OPEN ACCESS

Implementation of a Load Balancer for Instant Messaging over SIP Server Clusters with Improved Response time

P. Lekha Chandra [1] A.Rama Satish [2]
STUDENT, D V R & Dr H S MIC College of Technology, Kanchikacherla, Krishna (Dt).
Associate Professor, D V R & Dr H S MIC College of Technology, Kanchikacherla, Krishna (Dt).

Abstract—
This paper introduces an implementation of a Load balancer in a cluster of SIP servers which supports instant messages. The implementation uses TLWL algorithm which provides significantly better response time by distributing requests across the cluster more evenly, thus minimizing occupancy and the corresponding amount of time a particular request waits behind others for service. Resulting in this algorithm improves throughput and response-time of servers. Load balancer maintains sessions in which requests corresponding to the same session are sent by the load balancer to the same server. Load balancer improves both throughput and response time versus a single node while exposing a single interface to external clients.

Index Terms— SIP servers, Load balancer, Least-Work-Left algorithm

I. INTRODUCTION

The Session Initiation Protocol (SIP) is a signaling protocol used for controlling communication sessions such as voice and video calls SIP is an application layer protocol which is independent of underlying layer. SIP [5],[6],[7] is a protocol of growing importance with uses in Voice over IP (VoIP). In large scale ISP's need to provide support to millions of users. Hence, a central component is required to distribute worked across multiple server clusters. The mechanism is known as the load balancing and the device that does the load balancing is called the Load Balancer [4]. It fulfils the demands included in Integrated Services Digital Networks (ISDN) decades ago because it achieves real service integration and offers proven interoperability. The SIP easily integrates the existing technologies of the Internet with instant messaging, presence services, voicemail and email, and network games. Work on SIP began back in 1995 and the first mature SIP RFC that fixed the shortcomings of the previous version (2543) was published in 2002. One of the fundamental characteristics of VoIP with SIP is that signaling with SIP usually takes a completely different path through the Internet than the media of a running call. It can happen that a call between two participants cannot be established only because one of the servers in the signaling chain is not reachable.
We present here a solution to protect a SIP service against all error types except client failures. We join the forces of several geographically distributed SIP servers to a community which we call a federation. But if their primary server is not reachable for one of the clients, it can simply switch over to one of the other servers within the federation. A frequent mechanism to scale a service is to use some form of a load-balancing dispatcher that distributes requests across a cluster of servers. We introduce new algorithms that outperform existing ones. This work is relevant not just to SIP, but also for other systems where it is advantageous for the load balancer to maintain sessions in which requests corresponding to the same session are sent by the load balancer to the same server. SIP is a transaction-based protocol designed to establish and tear down media sessions, frequently referred to as call .

Session-aware request assignment (SARA) is the process where a system assigns requests to servers such that sessions are properly recognized by that server, and subsequent requests corresponding to that same session are assigned to the same server. While SARA can be done in HTTP [2] for *performance* reasons, it is not necessary for *correctness*. HTTP load balancers do not take sessions into account in making load-balancing decisions. One another key aspect of the SIP protocol is that different transaction types, most notably the INVITE and BYE transactions, can incur significantly different overheads. A load balancer can make use of this information to make better load-balancing decisions that improve both response time [10] and throughput.

- Call-Join-Shortest-Queue (CJSQ) [1] tracks the number of Calls allocated to each back-end server and routes new SIP calls to the node with the least number of active calls
- Transaction-Join-Shortest-Queue [1](TJSQ) routes a new *call* to the server that has the fewest active *transactions*, rather than the fewest calls. CJSQ by recognizing that calls in

SIP are composed of the two transactions, INVITE and BYE, and that by tracking their completion separately, finer-grained estimates of server load can be maintained

Transaction-Least-Work-Left (TLWL) [1] routes a new call to the server that has the least *work. It* takes advantage of the observation that INVITE transactions are more expensive than BYE transactions

We implement these algorithms in software by adding them to the OpenSER [3] open-source SIP server configured as a load balancer. Using the open-source SIP_P workload generator driving traffic through the load balancer to a cluster of servers running a commercially available SIP server[8]. Our load balancer can effectively scale SIP server throughput and provide significantly lower response times without becoming a bottleneck. Dramatic response time reductions that we achieve with TLWL and TJSQ suggest that these algorithms should be adapted for other applications, particularly when response time is crucial.

II. BACKGROUND

Overview of Protocol:

SIP is a signaling protocol designed to establish, terminate and modify media sessions between two or more parties. Several kinds of sessions can be used. The SIP does not allocate and manage network bandwidth as does a network resource reservation protocol that is considered outside the scope of the protocol. "SIP Trapezoid" a typical SIP VoIP scenario as shown in the fig.1 [1].

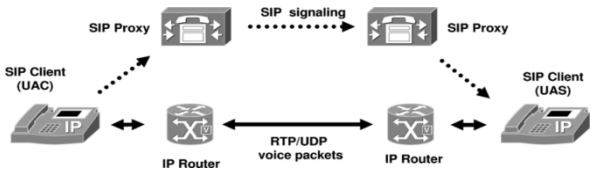


Fig. 1. SIP Trapezoid

Once endpoints are found, communication is typically performed directly in a peer-to-peer fashion. The separation of the data plane from the control plane is one of the key features of SIP and contributes to its flexibility. The SIP protocol requires that proxies forward and preserve headers.

SIP Users, Agents, Transactions, and Messages:

A SIP Uniform Resource Identifier (URI) uniquely identifies a SIP user. This layer of indirection enables features such as location independence and mobility. The SIP users employ endpoints known as *user agents.* These entities initiate and receive sessions. User agents are further decomposed into *User Agent Clients* (UAC) and *User Agent Servers* (UAS), depending on whether they act as a client in a transaction (UAC) or a server (UAS). SIP uses HTTP-like request/response *transactions*. The consists of a request to perform a particular method and at least one response to that request. The responses may be *provisional* that they provide some short-term feedback to the user to indicate progress or they can be final. A SIP session is a relationship in SIP between two user agents that lasts for some time period; in VoIP, a session corresponds to a phone call, which is called a *dialog* in SIP and results in state being maintained on the server for the duration of the session. A BYE message creates a new transaction and when the transaction completes, ends the session. A typical message flow where SIP messages are routed through the proxy [9] as illustrated in fig.2.

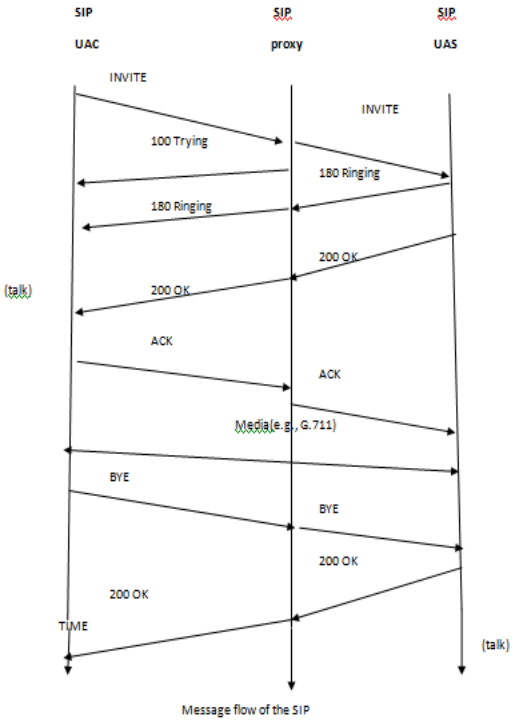


Fig.2. Message flow of the SIP

Message Header of SIP:

SIP is a text-based protocol that derives much of its syntax from HTTP. Messages contain headers and additionally bodies, depending on the type of message. SIP messages contain an additional protocol in VoIP, the Session Description Protocol (SDP), which negotiates session parameters between endpoints using an offer/answer model.

III. EXISTING SYSTEM

User Agent Clients send SIP requests (e.g., INVITE, BYE) to our load balancer that then selects a SIP server to handle each request. The system is depicted in the fig.3 [1]. The SIP responses send by the servers to the load balancer is then forward to the client.

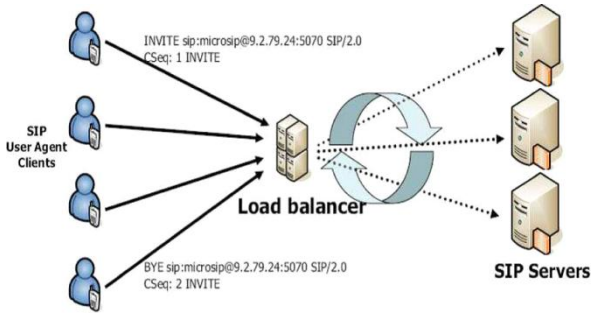


Fig. 3. System architecture

Once a session has been established, then the parties participating in the session would typically communicate directly with each other using a different protocol for the media transfer.

Novel Algorithm:

A key aspect of our load balancer is that requests corresponding to the same call are routed to the same server. Load balancer has the freedom to pick a server *only* on the *first* request of a call. Therefore, all subsequent requests corresponding to the call must go to the same server. Our new load-balancing algorithms are based on assigning calls to servers by picking the server with the (estimated) least amount of work assigned but not yet completed. The concept of assigning work to servers with the least amount of work left to do have been applied in other contexts, and then the specifics of how to do this efficiently for a real application are often not at all obvious. The system needs another method to reliably estimate the amount of work that a server [8] has left to do at the time load-balancing decisions are made. The load balancer can estimate the work assigned to a server based on the requests it has assigned to the server and the responses it has received from the server. The responses from servers to clients first go through the load balancer that forwards the responses to the appropriate clients. The load balancer can determine when a server has finished processing a request or call and update the estimates it is maintaining for the work assigned to the server.

The *Call-Join-Shortest-Queue* algorithm estimates the amount of work a server has left to do based on the number of *calls* assigned to the server. The counters are maintained by the load balancer indicating the number of calls assigned to each server. A limitation of this approach is that the number of calls assigned to a server is not always an accurate measure of the load on a server. In addition, different calls may consist of different numbers of transactions and may consume different amounts of server resources. Advantages of CJSQ can be used in environments in which the load balancer is aware of the calls assigned to servers but does not have an accurate estimate of the transactions assigned to servers.

An alternative method is to estimate server load based on the number of *transactions* (requests) assigned to the servers. *Transaction-Join-Shortest-Queue* algorithm estimates the amount of work a server has left to do base on the number of transactions (requests) assigned to the server. The counters are maintained by the load balancer indicating the number of transactions assigned to each server. A limitation of this approach is that all transactions are weighted equally. New calls are assigned to servers with the lowest counter. INVITE requests are more expensive than BYE requests since the INVITE transaction state machine is more complex than the one for non-INVITE transactions in the SIP protocol.

The *Transaction-Least-Work-Left* algorithm addresses this issue by assigning different *weights* to different transactions depending on their relative costs. It is quite similar by relative overhead; in the special case that all transactions have the same expected overhead. Counters are maintained by the load balancer indicating the waiting number of transactions assigned to each server. A ratio is defined in terms of relative cost of INVITE to BYE transactions. New calls are assigned to the server with the lowest counter. TLWL estimates server load based on the weighted number of transactions a server is currently handling. TLWL can be adapted to workloads with other transaction types by using different weights based on the overheads of the transaction types.

Implementation:

The rectangles represent key functional modules of the load balancer, while the irregular shaped boxes represent state information that is maintained. The structure of the load balancing factor is illustrated in fig.4. The arrows represent communication flows. *Receiver* receives requests that are then parsed by the *Parser*. *Session Recognition* module determines if the request corresponds to an already existing session by querying the *Session State*. If so, the request is forwarded to the server to which the session was previously assigned; else the *Server Selection* module assigns the new session to a server using one of the algorithms described earlier. The *Sender* forwards requests to servers and updates *Load Estimates* and *Session State* as needed. *Receiver* also receives responses sent by servers. The client to receive the response is identified by the *Session Recognition* module. The *Sender* then sends the response to the client and updates *Load Estimates* and *Session State* as needed. *Trigger* module updates *Session State* and

Load Estimates after a session has expired. Our load balancer selects the appropriate server to handle the first request of a call. When a new transaction corresponding to the call is received, it will be routed to the correct server.

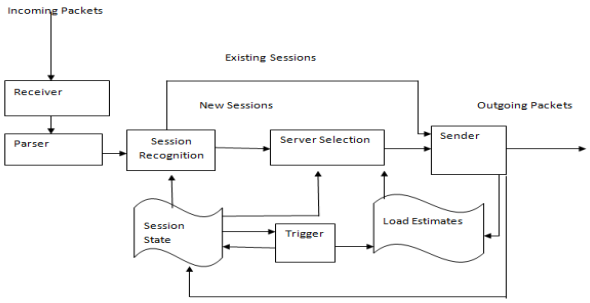


Fig. 4.Load balancer architecture

When an INVITE request arrives corresponding to a new call, the call is assigned to a server using one of the algorithms [1]. Subsequent requests corresponding to the call are always sent to the same machine to where the original INVITE was assigned. The load balancer stops sending requests to the server if a server fails. The load balancer can be notified to start sending requests to the server again if the failed server is later revived.

A primary load balancer could be configured with a secondary load balancer that would take over in the event that the primary fails. The primary load balancer would periodically checkpoint its state, either to the secondary load balancer over the network or to a shared disk. Future area of research is to implement this fail over scheme in a manner that both optimizes performance and minimizes lost information in the event that the primary load balancer fails.

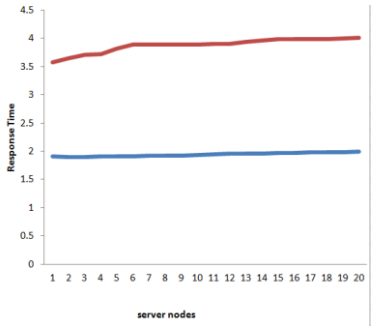
IV.PROPOSED SYSTEM

While conceptually and technically TLWL, TJSQ use similar principles for assigning sessions to servers, there are considerable differences between them in terms of performance polling and estimations of server stress conditions. Although their performance is validated through a complex data (VOIP) oriented SIP mechanisms, we would like to extend them to other areas of communication technologies. Propose to evaluate our algorithms on larger clusters to further test their scalability, adding a fail-over mechanism using a load balancer which overcomes the single point of failure problems, and will implement this scalability on other SIP workloads such as instant messaging.

Instant messaging (IM) is a type of online chat which offers real time text transmission over the internet. Short messages are typically transmitted bi-directionally between two parties, when each user chooses to complete a thought and select "send". Instant messaging systems tend to facilitate connections between specified known users .Instant messaging has proven to be similar to personal computers, email, and the WWW, in that its adoption for use as a business communications medium was driven primarily by individual employees using consumer software at work, rather than by formal mandate or provisioning by corporate information technology departments. Tens of millions of the consumer IM accounts in use are being used for business purposes by employees of companies and other organizations.

V. PERFORMANCE ANALYSIS

TLWL algorithm [1] helps to select a server from cluster of servers which has very least work left for messing sending. Graph shows the relationship between response time and server nodes for both voice call and instant messaging. In voice call number of server nodes increases response time also increases and in instant messing even though the server nodes increases there is no drastic increment in response time and it maintain almost equal response for this proposed system . Server limit is not controls overall system performance.



Comparison between number of server nodes and response time

Several opportunities exist for potential future work. This include design and implementation of secondary load balancer to handle overall session in failure of primary load balancer without effecting the data transformation between clients.

REFERENCES

[1] Hongbo Jiang, Erich Nahum, Wolfgang Segmuller, Asser N. Tantawi, and Charles P.Wright, "Design, Implementation, and Performance of a Load Balancer for SIP Server Clusters," in *Proc.* IEEE/ACM transactions on networking, Oct. 2009.

[2] R.Fielding ,JGettvs, J.Mogul,H.frvstyk and t.Berners-Lee, "Hypertext Trasfer Protocol-HTTP/1.1," Internet Engineering Task Force, RFC 2068, Jan 1997.

[3] M. Aron, P. Druschel, and W. Zwaenepoel, "Efficient support for P-HTTP in cluster-

based Web servers," in *Proc. USENIX Annu. Tech. Conf.*, Monterey, CA, Jun. 1999, pp. 185–198.

[4] Hongbo Jiang, Erich Nahum, Wolfgang Segmuller, Asser N. Tantawi, and Charles P.Wright, "Design, Implementation, and Performance of a Load Balancer for SIP Server Clusters," in *Proc.* IEEE/ACM transactions on networking, Oct. 2009.

[5] V.Hilt and I.Widjaja, "Controlling overload n networks of SIP servers," in Proc.IEEEICNP.Orlandp.FL. Oct 2008,pp.83-93.

[6] E. Nahum, J. Tracey, and C. P. Wright, "Evaluating SIP proxy server performance," in Proc. 17th NOSSDAV, Urbanachampaign,IL, Jun 2007,pp.79-85.

[7] Foundry Networks," ServerIron switches support SIP load balancing VOIP/SIP traffic Available:http://www.foundrynet.com/solutions/sol-app-switch/sol- voip-sip/

[8] M. Aron and P. Druschel, "TCP implementation enhancements for improving Webserver performance," Computer Science Department, Rice University, Houston, TX, Tech. Rep. TR99-335, Jul. 1999.

[9] E. Nahum, J. Tracey, and C. P. Wright, "Evaluating SIP proxy server performance," in Proc. 17th NOSSDAV, Urbana champaign, IL, Jun 2007,pp.79-85.

[10] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar, "A novel server selection technique for improving the response time of a replicated ser vice," in *Proc. IEEE INFOCOM*, 1998, vol. 2, pp. 783–791.