

## Effective SIP Servers Load Balancing

P. Harikrishna<sup>1</sup>, K. Vasanth Kumar<sup>2</sup> and P. Suresh Babu<sup>2</sup>

1. M.Tech Scholar, Department of CSE, Kaushik College of Engineering, Visakhapatnam, AP, India.

2. Associate Professor, Department of CSE, Kaushik College of Engineering, Visakhapatnam, AP, India

**Abstract:** This paper introduces novel load balancing algorithms for distributing Session Initiation Protocol (SIP) requests to a cluster of SIP servers. Our load balancer improves both throughput and response time versus a single node, while exposing a single interface to external clients. We present the design, implementation and evaluation of our system using a cluster of Intel x86 machines running Linux. Our best algorithm, Transaction Least-Work-Left (TLWL), achieves its performance by integrating several features: knowledge of the SIP protocol; dynamic estimates of back-end server load; distinguishing transactions from calls; recognizing variability in call length; and exploiting differences in processing costs for different SIP transactions. By combining these features, our algorithm provides finer-grained load balancing than standard approaches, resulting in throughput improvements of up to 24 percent and response time improvements of up to two orders of magnitude. We present a detailed analysis of occupancy to show how our algorithms significantly reduce response time.

**Keywords:** Dispatcher, Load Balancing, Performance, Session Initiation Protocol (SIP).

### I. INTRODUCTION

The Session Initiation Protocol (SIP) is a general-purpose Signaling protocol used to control various types of media sessions. SIP is a protocol of growing importance, with uses in Voice over IP, Instant Messaging, IPTV, Voice Conferencing, and Video Conferencing. Wireless providers are standardizing on SIP as the basis for the IP Multimedia System (IMS) standard for the Third Generation Partnership Project (3GPP). Third-party VoIP providers use SIP (e.g., Vonage, Gizmo), as do digital voice offerings from existing legacy Telcos (e.g., AT&T, Verizon) as well as their cable competitors (e.g., Comcast, Time-Warner).

While individual servers may be able to support hundreds or even thousands of users, large-scale ISPs need to support customers in the millions. A central component to providing any large-scale service is the ability to scale that service with increasing load and customer demands. 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. However, almost all research in this space has been in the context of either the Web (e.g., HTTP [24]) or file service (e.g., NFS [1]). This paper presents and evaluates several algorithms for balancing load across multiple SIP servers. We introduce new algorithms which outperform existing ones. Our 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 is sent by the load balancer to the same server.

SIP has a number of features which distinguish it from protocols such as HTTP. SIP is a transaction-based protocol designed to establish and tear down media sessions, frequently referred to as calls. Two types of state exist in SIP. The first, session state, is created by the INVITE transaction and is destroyed by the BYE transaction. Each SIP transaction also

creates state that exists for the duration of that transaction. SIP thus has overheads that are associated both with sessions and with transactions, and taking advantage of this fact can result in more optimized SIP load balancing.

The session-oriented nature of SIP has important implications for load balancing. Transactions corresponding to the same call must be routed to the same server; otherwise, the server will not recognize the 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. In contrast, sessions are less significant in HTTP. While SARA can be done in HTTP for performance reasons (e.g., routing SSL sessions to the same back end to encourage session reuse and minimize key exchange), it is not necessary for correctness. Many HTTP load balancers do not take sessions into account in making load balancing decisions.

Another key aspect of the SIP protocol is that different transaction types, most notably the INVITE and BYE transactions, can incur significantly different overheads: On our systems, INVITE transactions are about 75 percent more expensive than BYE transactions. A load balancer can make use of this information to make better load balancing decisions which improve both response time and throughput. Our work is the first to demonstrate how load balancing can be improved by combining SARA with estimates of relative overhead for different requests.

This paper introduces and evaluates several novel algorithms for balancing load across SIP servers. Each algorithm combines knowledge of the SIP protocol, dynamic estimates of server load, and

Session-Aware Request Assignment (SARA). In addition, the best-performing algorithm takes into account the variability of call lengths, distinguishing transactions from calls, and the difference in relative processing costs for different SIP transactions.

1) Call-Join-Shortest-Queue (CJSQ) tracks the number of calls (in this paper, we use the terms *call* and *session* interchangeably) allocated to each back-end server and routes new SIP calls to the node with the least number of active calls.

2) Transaction-Join-Shortest-Queue (TJSQ) routes a new *call* to the server that has the fewest active *transactions*, rather than the fewest calls. This algorithm improves on 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. This leads to better load balancing, particularly since calls have variable length and thus do not have a unit cost.

3) Transaction-Least-Work-Left (TLWL) routes a new call to the server that has the least *work*, where work (i.e., load) is based on relative estimates of transaction costs. TLWL takes advantage of the observation that INVITE transactions are more expensive than BYE transactions. We have found that a 1.75:1 cost ratio between INVITE and BYE results in the best performance.

We implement these algorithms in software by adding them to the Open SER open-source SIP server configured as a load balancer. Our evaluation is done using the SIPP open-source workload generator driving traffic through the load balancer to a cluster of servers running a commercially available SIP server. The experiments are conducted on a dedicated test bed

of Intel x86-based servers connected via Gigabit Ethernet.

This paper makes the following contributions:

- We show that two of our new algorithms, TLWL and TJSQ, scale better, provide higher throughputs and exhibit lower response times than any of the other approaches we tested. The differences in response times are particularly significant. For low to moderate workloads, TLWL and TJSQ provide response times for INVITE transactions that are an order of magnitude lower than that of any of the other approaches. Under high loads, the improvement increases to two orders of magnitude.
- We present the design and implementation of a load balancer for SIP servers, and demonstrate throughput of up to 5500 calls per second and scalability of up to 10 nodes. Our measurements show that the dispatcher introduces minimal overhead to a SIP request. We extensively evaluate several approaches for balancing SIP load across servers including the three novel algorithms described above as well as standard distribution policies such as round-robin or hashing based on the SIP Call-ID.
- We present a detailed analysis of why TLWL and TJSQ provide substantially better response times than the other algorithms. *Occupancy* has a significant effect on response times, where the occupancy for a transaction  $T$  assigned to a server  $S$  is the number of transactions already being handled by  $S$  when  $T$  is assigned to it. As described in detail in Section V, by allocating load more evenly across nodes, the distributions of occupancy across the cluster are balanced, resulting in greatly improved response times. The naive approaches, in contrast, lead to imbalances in load. These imbalances result in the distributions of occupancy that exhibit large tails, which contribute significantly to response

time as seen by that request. To our knowledge, we are the first to observe this phenomenon experimentally.

These results show that our load balancer can effectively scale SIP server throughput and provide significantly lower response times without becoming a bottleneck. The 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.

We believe these results are general for load balancers, which should keep track of the number of uncompleted requests assigned to each server in order to make better load balancing decisions. If the load balancer can reliably estimate the relative overhead for requests that it receives, this can further improve performance.

## II. BACKGROUND

This section presents a brief description of SIP.

### A. Overview of the Protocol

SIP is a control-plane protocol designed to establish, alter, and terminate media sessions between two or more parties. The core IETF SIP specification is given in RFC 3261 [26], although there are many additional RFCs that enhance and refine the protocol. SIP uses HTTP-like request/response transactions. A transaction consists of a request to perform a particular method (e.g., INVITE, BYE, CANCEL, etc.) and at least one response to that request. Fig. A below illustrates a typical SIP VoIP scenario, known as the “SIP Trapezoid.”

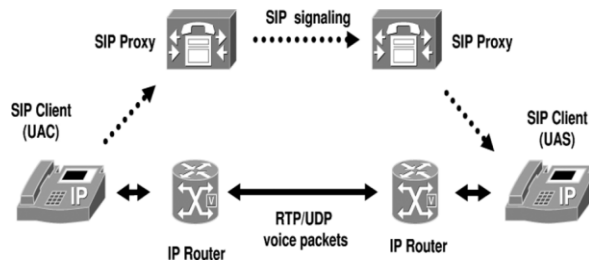


Fig. A. SIP Trapezoid

Note the separation between control and data paths: SIP messages traverse the SIP overlay network, routed by proxies, to find the eventual destinations. Once endpoints are found, communication is typically performed directly in a peer-to-peer fashion. In this example, each endpoint is an IP phone. However, an endpoint can also be a server providing services such as voicemail, firewalling, voice conferencing, etc. This paper focuses on scaling the server (in SIP terms, the UAS, described below), rather than the proxy. The separation of the data plane from the control plane is one of the key features of SIP and contributes to its flexibility. SIP was designed with extensibility in mind; for example, the SIP protocol requires that proxies forward and preserve headers that they do not understand. As another example, SIP can run over many protocols such as UDP, TCP, TLS, SCTP, IPv4, and IPv6.

**B. SIP Users, Agents, Transactions, and Messages**

SIP Users, Agents, Transactions, and Messages A SIP Uniform Resource Identifier (URI) uniquely identifies a SIP user, e.g., sip:hongbo@us.ibm.com. This layer of indirection enables features such as location independence and mobility. SIP users employ endpoints known as user agents. These entities initiate and receive sessions. They can be either hardware (e.g., cell phones, pagers, hard VoIP phones) or software (e.g., media mixers, IM clients, soft phones). 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). Most call flows for SIP messages thus display how the UAC and UAS behave for that situation. SIP uses HTTP-like request/response transactions. A transaction consists of a request to perform a particular method (e.g., INVITE, BYE, CANCEL, etc.) and at least one response to that request. Responses may be provisional, namely, that they provide some short-term feedback to the user (e.g., 100 TRYING, 180 RINGING) to indicate progress, or they can be final (e.g., 200 OK, 407 UNAUTHORIZED). The transaction is only completed when a final response is received, not a provisional response. 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. This is called a dialog in SIP and results in state being maintained on the server for the duration of the session. For example, an INVITE message not only creates a transaction (the sequence of messages for completing the INVITE), but also a session if the transactions completes successfully. A BYE message creates a new transaction and, when the transaction completes, ends the session. Fig. B illustrates a typical SIP message flow, where SIP messages are routed through the proxy.

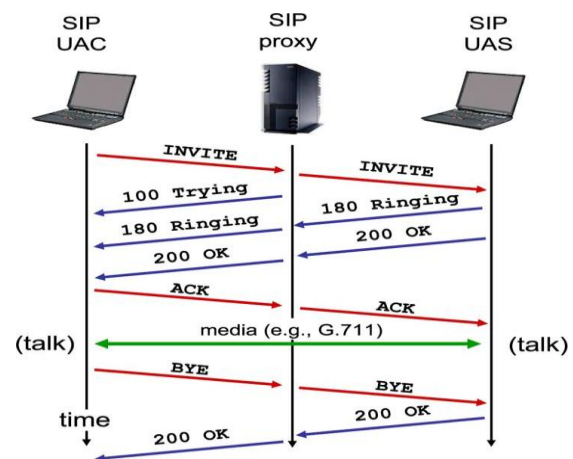


Fig. B. SIP Message Flow

In this example, a call is initiated with the INVITE message and accepted with a 200 OK message. Media is exchanged, and then the call is terminated using the BYE message.

### C. SIP Message Header

SIP is a text-based protocol that derives much of its syntax from HTTP [12]. Messages contain headers and additionally bodies, depending on the type of message. In VoIP, SIP messages contain an additional protocol, the Session Description Protocol (SDP) [30], which negotiates session parameters (e.g., which voice codec to use) between endpoints using an offer/answer model. Once the end-hosts agree to the session characteristics, the Real-time Transport Protocol (RTP)

is typically used to carry voice data [33]. RFC 3261 [31] shows many examples of SIP headers. An important header to notice is the Call-ID: header, which is a globally unique identifier for the session that is to be created. Subsequent SIP messages must refer to that Call-ID to look up the established session state. If a SIP server is provided by a cluster, the initial INVITE request will be routed to one back-end node, which will create the session state. Barring some form of distributed shared memory in the cluster, subsequent packets for that session must also be routed to the same back-end node; otherwise the packet will be erroneously rejected. Thus, many SIP load-balancing approaches use the Call-ID as hashing value in order to route the message to the proper node.

## III. LOAD BALANCING ALGORITHMS

This section presents the design of our load balancing algorithms. Due to space limitations, implementation details are omitted. Figure 1 depicts our overall system. User Agent Clients send

SIP requests (e.g., INVITE, BYE) to our load balancer which then selects a SIP server to handle each request.

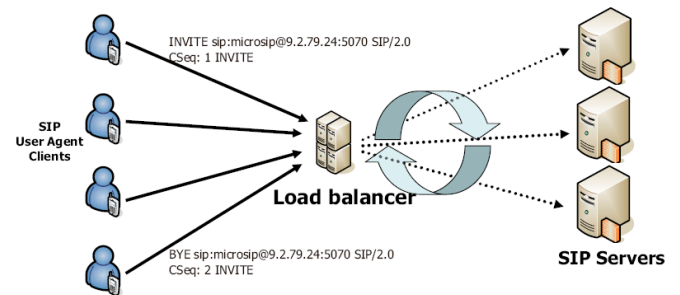


Fig.1. System Architecture

The distinction between the various load balancing algorithms presented in this paper are *how* they choose which SIP server to handle a request. Servers send SIP responses (e.g., 180 TRYING or 200 OK) to the load balancer which then forwards the response to the client.

Note that SIP is used to establish, alter, or terminate media sessions. Once a session has been established, the parties participating in the session would typically communicate directly with each other using a different protocol for the media transfer which would not go through our SIP load balancer.

### A. Novel Algorithms

A key aspect of our load balancer is that requests corresponding to the same call are routed to the same server. The load balancer has the freedom to pick a server only on the first request of a call. All subsequent requests corresponding to the call must go to the same server. This allows all requests corresponding to the same session to efficiently access state corresponding to the session.

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. While the concept of assigning work to servers with the least amount of work left to do has been applied in other contexts [13], [27], the specifics of how to do this efficiently for a real application are often not at all obvious. The system needs some method to reliably estimate the amount of work that a server has left to do at the time load balancing decisions are made.

In our system, 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. All responses from servers to clients first go through the load balancer which forwards the responses to the appropriate clients. By monitoring these responses, 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.

**1) Call-Join-Shortest-Queue:** The Call-Join-Shortest-Queue (CJSQ) algorithm estimates the amount of work a server has left to do based on the number of calls (sessions) assigned to the server. Counters are maintained by the load balancer indicating the number of calls assigned to each server. When a new INVITE request is received (which corresponds to a new call), the request is assigned to the server with the lowest counter, and the counter for the server is incremented by one. When the load balancer receives a 200 OK response to the BYE corresponding to the call, it knows that the server has finished processing the call and increments the counter for the 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. There may be long idle periods between the transactions in a call. In addition, different calls may consist of different

numbers of transactions and may consume different amounts of server resources. An advantage of CJSQ is that it 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.

**2) Transaction-Join-Shortest-Queue:** An alternative method is to estimate server load based on the number of transactions (requests) assigned to the servers. The Transaction-Join-Shortest-Queue (TJSQ) algorithm estimates the amount of work a server has left to do is based on the number of transactions (requests) assigned to the server. Counters are maintained by the load balancer indicating the number of transactions assigned to each server. New calls are assigned to servers with the lowest counter.

A limitation of this approach is that all transactions are weighted equally. In the SIP protocol, INVITE requests are more expensive than BYE requests, since the INVITE transaction state machine is more complex than the one for non-INVITE transactions (such as BYE). This difference in processing cost should ideally be taken into account in making load balancing decisions.

**3) Transaction-Least-Work-Left:** The Transaction-Least-Work-Left (TLWL) algorithm addresses this issue by assigning different weights to different transactions depending on their relative costs. It is similar to TJSQ with the enhancement that transactions are weighted by relative overhead; in the special case that all transactions have the same expected overhead, TLWL and TJSQ are the same. Counters are maintained by the load balancer indicating the weighted number of transactions assigned to each server. New calls are assigned to the server with the lowest counter. A ratio is defined in terms of relative cost of INVITE to BYE

transactions. Thus far, our presentation of the load balancing algorithms assumes that the servers have similar processing capacities. However, this may not always be the case. Some servers may be more powerful than others; other servers may have substantial background jobs that consume cycles. In these situations, the load balancer could assign a new call to the server with the lowest value of estimated work left to do (as determined by the counters) divided by the capacity of the server; this applies to CJSQ, TJSQ, and TLWL.

In some cases, though, the load balancer might not know the capacity of the servers. For these situations, our new algorithms have the robustness to automatically adapt to heterogeneous back-end servers with over 60% higher through puts than the previous algorithms we tested.

CJSQ, TJSQ, and TLWL are all novel load balancing algorithms. In addition, we are not aware of any previous work which has successfully adapted least work left algorithms for load balancing with SARA.

#### IV. EXPERIMENTAL ENVIRONMENT

We describe here the hardware and software that we use, our experimental methodology, and the metrics we measure.

**SIP Software:** For client-side workload generation, we use the open source SIPp and Asterisk SIP load generator tool, SIPp is the defacto standard for generating SIP load. SIPp is a configurable packet generator, extensible via a simple XML configuration language. It uses an efficient event-driven architecture but is not fully RFC compliant (e.g., it does not do full packet parsing). It can thus emulate either a client (UAC) or server (UAS), but at many times the capacity of a standard SIP end

host. For the back-end server, we use a commercially available SIP server.

**Hardware and System Software:** We conduct experiments using two different types of machines, both of which are IBM x-Series rack-mounted servers. Table I summarizes the hardware and software configuration for our test bed.

Feature	Machine Type-A	Machine Type-B
Quantity	8	3
CPU	3.06 GHz	2.8 GHz
RAM	4 GB	2 GB
Kernel	3.15.4-x86_64-linode45	3.15.4-x86_64-linode45
Distro	CentOS release 6.5 (Final)	CentOS release 6.5 (Final)
Roles	Back-End Server, Load Balancer	Workload generation

TABLE I: HARDWARE TESTBED CHARACTERISTICS

Eight of the servers have two processors; however, for our experiments, we use only one processor. All machines are interconnected using a gigabit Ethernet switch.

**Workload:** The workload we use is SIPp's simple SIP UAC call model consisting of an INVITE, which the server responds to with 100 TRYING, 180 RINGING, and 200 OK responses. The client then sends an ACK request which creates the session. After a variable pause to model call hold times, the client closes the session using a BYE which the server responds to with a 200 OK response. Calls may or may not have pause times associated with them, intended to capture the variable call duration of SIP sessions. In our experiments, pause times are normally distributed with a mean of one minute and a variance of 30 seconds. While simple, this is a common configuration used in SIP performance testing.

Currently no standard SIP workload model exists, although SPEC is attempting to define one [30].

**Methodology:** Each run lasts for 3 minutes after a warm-up period of 10 minutes. There is also a ramp-up phase until the experimental rate is reached. The request rate starts at 1 cps and increases by  $x$  cps every second, where  $x$  is the number of back-end nodes. Thus, if there are 8 servers, after 5 seconds, the request rate will be 41 cps. If load is evenly distributed, each node will see an increase in the rate of received calls of one additional cps until the experimental rate is reached. After the experimental rate is reached, it is sustained. SIPP is used in open-loop mode; calls are generated at the configured rate regardless of whether the other end responds to them.

**Metrics:** We measure both throughput and response time. We define throughput as the number of completed requests per second. The peak throughput is defined as the maximum throughput which can be sustained while successfully handling more than 99.99% of all requests. Response time is defined as the length of time between when a request (INVITE or BYE) is sent and the successful 200 OK is received.

**Component Performance:** We have measured the throughput of a single SIPP node in our system to be 2925 calls per second (cps) without pause times and 2098 cps with pause times. The peak throughput for the back-end SIP server is about 300 cps in our system; this figure varies slightly depending on the workload. Surprisingly, the peak throughput is not affected much by pause times. While we have observed that some servers can be adversely affected by pause times, we believe other overheads dominate and obscure this effect in the server we use.

## V. RESULTS

In this section, we present in detail the experimental results of the load balancing algorithms defined in Section III.

### A. Response Time

We observe significant differences in the response times of the different load balancing algorithms. Figure-2 shows the average response time for each algorithm versus offered load measured for the INVITE transaction.

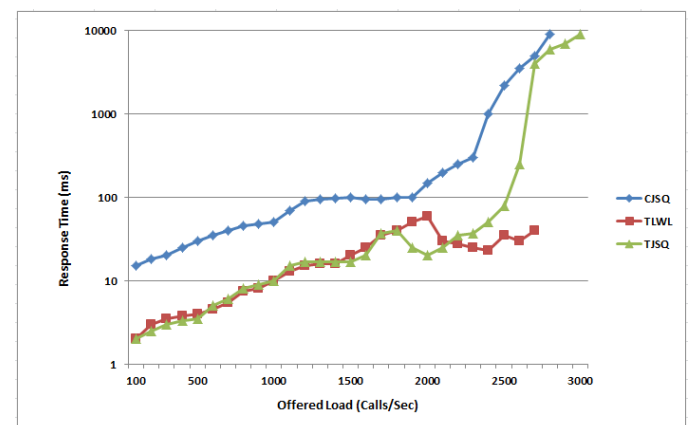


Fig.2: Average Response Time for INVITE

Note especially that the Y axis is in logarithmic scale. In this experiment, the load balancer distributes requests across 8 back-end SIP server nodes.

The algorithms cluster into three groups: TLWL offer the best performance; TJSQ offers moderate performance; and CJSQ results in the worst performance.

As the system approaches peak throughput, the performance advantage of the first group of algorithms increases to two orders of magnitude.

Similar trends are seen in Figure 3, which shows average response time for each algorithm vs.



offered load for BYE transactions, again using 8 back-end SIP server nodes. BYE transactions consume fewer resources than INVITE transactions resulting in lower average response times. TLW Land TJSQ provide the lowest average response times.

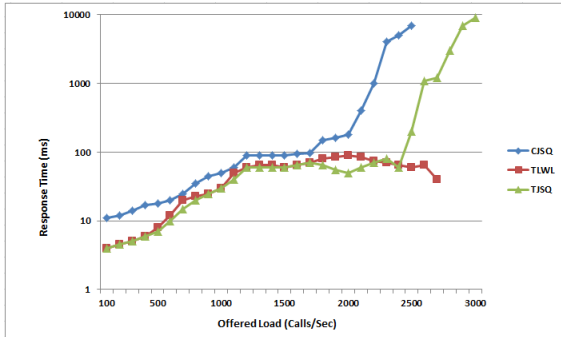


Fig.3: Average Response Time for BYE

However, the differences in response times for the various algorithms are smaller than is the case with INVITE transactions. This is largely because of SARA. The load balancer has freedom to pick the least loaded server for the first INVITE transaction of a call. However, a BYE transaction must be sent to the server which is already handling the call. The significant improvements in response time that TLW Land TJSQ provide present a compelling reason for systems such as these to use our algorithms.

**B. Throughput**

We now examine how our load balancing algorithms perform in terms of how well throughput scales with increasing numbers of back-end servers. In the ideal case, we would hope to see 8 nodes provide 8 times the single-node performance. Recall that the peak throughput is the maximum throughput which can be sustained while successfully handling more than 99.99% of all requests and is approximately 300 cps for a back-end SIP server node. Therefore, linear scalability

suggests a maximum possible throughput of about 2400 cps for 8 nodes. Figure 4 shows the peak throughputs for the various algorithms using 8 back-end nodes. Several interesting results are illustrated in this graph.

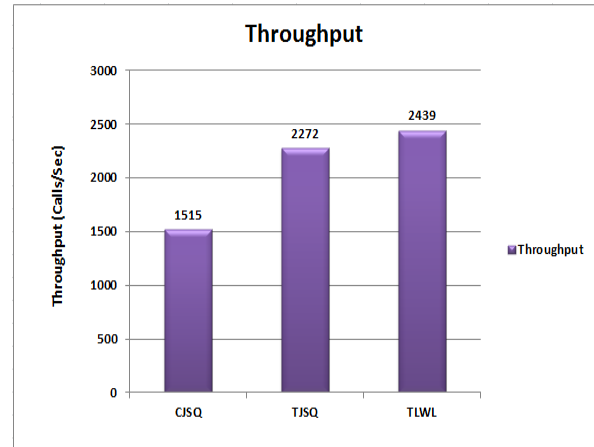


Fig.4: Peak Throughput of 3 Algorithms

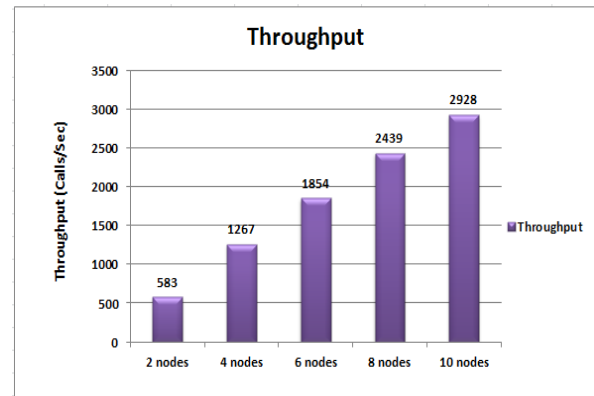


Fig.5: Peak throughput vs. # of nodes (TLWL)

TLWL achieves linear scalability and results in the highest peak throughput of 2439 cps. The same three algorithms resulted in the best response times and peak throughput. Response times may not be the most reliable measure of load on the servers. If the load balancer weights the most recent response time(s) too heavily, this might not provide enough information to determine the least loaded server. On the other hand, if the load balancer gives significant weight to response times in the past, this makes the algorithm too slow to respond to

changing load conditions. A server having the lowest weighted average response time might have several new calls assigned to it resulting in too much load on the server before the load balancer determines that it is no longer the least loaded server. In contrast, when a call is assigned to a server using TLWL or TJSQ, the load balancer takes this information immediately into account when making future load balancing decisions. Therefore, TLWL and TJSQ would not encounter this problem.

Calls-Join-Shortest-Queue (CJSQ) is significantly worse than the others, since it does not distinguish call hold times in the way that the transaction-based algorithms do. Experiments we ran that did not include pause times (not shown due to space limitations) showed CJSQ providing very good performance, comparable to TJSQ. This is perhaps not surprising since, when there are no pause times, the algorithms are effectively equivalent. However, the presence of pause times can lead CJSQ to misjudgments about allocation that end up being worse than a static allocation such as Hash. TJSQ does better than most of the other algorithms. This shows that knowledge of SIP transactions and paying attention to the call hold time can make a significant difference, particularly in contrast to CJSQ.

We determined that the load balancer can support up to about 5400 cps before becoming overloaded. Given that the peak throughput of the back-end SIP server that we use is about 300 cps, the prototype should be able to support about 17 servers of this type. The load balancer was not a bottleneck in any of the experiments described in this paper.

In many deployments, it is not realistic to expect that all nodes of a cluster have the same server capacity. Some servers may be more powerful than

others. Other servers maybe running background tasks which limit the CPU resources which can be devoted to SIP. Our new algorithms adapt to heterogeneous back ends much more effectively than the prior art ones. Experiments we ran indicate that TLWL achieves near optimal throughput when the back ends differ in processing power by as much as 50% which is over 60% higher throughput than the prior art algorithms we tested.

## VII. SUMMARY AND CONCLUSIONS

This paper introduces three novel approaches to load balancing in SIP server clusters. We present the design, implementation, and evaluation of a load balancer for cluster based SIP servers. Our load balancer performs session-aware request assignment (SARA) to ensure that SIP transactions are routed to the proper back-end node that contains the appropriate session state. We presented three novel algorithms: Call Join Shortest Queue (CJSQ), Transaction Join Shortest Queue (TJSQ), and Transaction Least-Work-Left (TLWL).

The TLWL algorithms result in the best performance, both in terms of response time and throughput, followed by TJSQ. TJSQ has the advantage that no knowledge is needed of relative overheads of different transaction types. The most significant performance differences were in response time. For SIP applications that require good quality of service, these dramatically lower response times are significant. We showed that these algorithms provide 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. TLWL provides 25% better throughput than other algorithms. TJSQ provides nearly the same level of performance. CJSQ performs poorly since it does

not distinguish transactions from calls and does not consider variable call hold times.

Our results show that by combining knowledge of the SIP protocol, recognizing variability in call lengths, distinguishing transactions from calls, and accounting for the difference in processing costs for different SIP transaction types, load balancing for SIP servers can be significantly improved.

The dramatic reduction in response times achieved by both TLWL and TJSQ, compared to other approaches, suggest that they should be applied to other domains besides SIP, particularly if response time is crucial. Our results are influenced by the fact that SIP requires SARA. However, even where SARA is not needed, variants of TLWL and TJSQ could be deployed and may offer significant benefits over commonly deployed load balancing algorithms based on round robin, hashing, or response times. A key aspect of TJSQ and TLWL is that they track the number of uncompleted requests assigned to each server, in order to make better assignments. This can be applied to load balancing systems in general. In addition, if the load balancer can reliably estimate the relative overhead for requests that it receives, this can further improve performance.

## REFERENCES

1. Darrell C. Anderson, Jeffrey S. Chase, and Amin Vahdat. Interposed request routing for scalable network storage. In *USENIX Operating Systems Design and Implementation (OSDI)*, San Diego, California, USA, October 2000.
2. Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Efficient support for P-HTTP in cluster-based Web servers. In *Proceedings of the USENIX 1999 Annual*

*Technical Conference*, Monterey, CA, June 1999.

3. Mohit Aron, Darren Sanders, Peter Druschel, and Willy Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *Proceedings of the USENIX 2000 Annual Technical Conference*, San Diego, CA, June 2000.
4. Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip S. Yu. The state of the art in locally distributed Web-server systems. *ACM Computing Surveys*, 34(2):263–311, June 2002.
5. Jim Challenger, Paul Dantzig, and Arun Iyengar. A scalable and highly available system for serving dynamic data at frequently accessed Web sites. In *Proceedings of ACM/IEEE SC98*, November 1998.
6. Gianfranco Ciardo, Alma Riska, and Evgenia Smirni. EQUILOAD: A load balancing policy for clustered Web servers. *Performance Evaluation*, 46(2-3):101–124, 2001.
7. Dan Dias, William Kish, Rajat Mukherjee, and Renu Tewari. A scalable and highly available Web server. In *Proceedings of the 1996 IEEE Computer Conference (COMPCON)*, February 1996.
8. F5. F5 introduces intelligent traffic management solution to power service providers' rollout of multimedia services (<http://www.f5.com/news-press-events/press/2007/20070924.html>).

9. Zongming Fei, Samrat Bhattacharjee, Ellen Zegura, and Mustapha Ammar. A novel server selection technique for improving the response time of a replicated service. In Proceedings of IEEE INFOCOM, 1998.
  10. Hanhua Feng, Vishal Misra, and Dan Rubenstein. PBS: A unified priority-based scheduler. In Proceedings of ACM Sigmetrics, San Diego, CA, June 2007.
  11. Richard Gayraud and Olivier Jacques. SIPp. <http://sipp.sourceforge.net>.
  12. G. Goldszmidt, G. Hunt, R. King, and R. Mukherjee. Network dispatcher: A connection router for scalable Internet services. In Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, April 1998.
  13. N. Griffiths, "Nmon: A free tool to analyze AIX and Linux performance," 2006 [Online]. Available: [http://www.ibm.com/developerworks/aix/library/au-analyze\\_aix/index.html](http://www.ibm.com/developerworks/aix/library/au-analyze_aix/index.html)
  14. M. Harchol-Balter, M. Crovella, and C. D. Murta, "On choosing a task assignment policy for a distributed server system," J. Parallel Distrib. Comput., vol. 59, no. 2, pp. 204–228, 1999.
  15. V. Hilt and I. Widjaja, "Controlling overload in networks of SIP servers," in Proc. IEEE ICNP, Orlando, FL, Oct. 2008, pp. 83–93. IBM, "Application switching with Nortel Networks Layer 2–7 Gigabit Ethernet switch module for IBM Blade Center," 2006 [Online]. Available: <http://www.ibm.com/webapp/wcs/stores/servlet/ProductDisplay?productId=4611686018425170446&storeId=1&langId=-1&catalogId=-840>
  16. A. Iyengar, J. Challenger, D. Dias, and P. Dantzig, "High-performance Web site design techniques," IEEE Internet Comput., vol. 4, no. 2, pp. 17–26, Mar./Apr. 2000.
  17. T. T. Kwan, R. E. McGrath, and D. A. Reed, "NCSA's World Wide Web server: Design and performance," Computer, vol. 28, no. 11, pp.68–74, Nov. 1995.
  18. D. Mosedale, W. Foss, and R. McCool, "Lessons learned administering Netscape's Internet site," IEEE Internet Comput., vol. 1, no. 2, pp.28–35, Mar./Apr. 1997.
  19. 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.
  20. Foundry Networks, "ServerIron switches support SIP load balancing VoIP/SIP traffic management solutions," Accessed Jul. 2007 [Online]. Available: <http://www.foundry.net.com/solutions/sol-app-switch/solvoip-sip/>
- Nortel Networks, "Layer 2–7 GbE switch module for IBM Blade-Center," Accessed Jul. 2007 [Online]. Available: <http://www-132.ibm.com/webapp/wcs/stores/servlet/ProductDisplay?productId=4611686018425170446&storeId=1&langId=-1&catalogId=-840>

## BIOGRAPHIES



Mr.P.Harikrishna is pursuing his M.Tech in Computer Science and Engineering at Kaushik College of Engineering, Visakhapatnam. His research interest includes VoIP/SIP Server Clustering and Load Balancing.



Sri.K.VasanthKumar, is an excellent teacher and received MS (Information Systems & Applications) from Bharathidasan University, Tiruchirapalli, Tamil Nadu and M.Tech (CSE) from JNTU Kakinada, Andhra Pradesh. He is



working as Associate Professor in Department of CSE & IT, Kaushik college of Engineering. He has 7 years of teaching experience in various

engineering colleges and 4 years of industrial experience. To his credit he has 1 International publication and 4 workshops. His areas of interest include Datawarehousing and Network Security.

Sri.P.SureshBabu, B.Tech, M.E., CSI Associate Professor Department of Computer Science & Engineering Kaushik College of Engineering, Visakhapatnam, Andhra Pradesh. Teaching Experience: 15 Years, Industrial Experience: 4 Years Sri.P.Suresh Babu, well known Author and excellent teacher Received B.Tech(CSE) from Acharya Nagarjuna university, GUNTUR, Andhra Pradesh and M.E(CSE) from Sathyabama university, Chennai is working as Associate Professor Department of Computer Science and Engineering, Kaushik college of Engineering, He is an active member of CSI. To his credit 7 International publications, 2 national publications, 2 International conferences and 4 workshops. His areas of interest include Artificial Intelligence and Fuzzy Logics.