The State of the Art in Locally Distributed Web-Server Systems

S.RENGANAYAKI
Assistant Professor/IT
Sri Angalamman College of Technology
Trichirappalli.
smrenganayaki@gmail.com

K.KALYANASUNDARAM
Assistant Professor/IT
Sri Angalamman College of Technology
Trichy.
krishp.kalyan@gmail.com

***Abstract*—**The paper describes a novel algorithms for a load balancer, allocates the work to the clusters of SIP server. The several load balancing algorithms for distributing Session Initiation Protocol (SIP) request to a cluster of SIP servers. This algorithm also supports the following three techniques such as CJSQ, TJSQ and TLWL. It is combine knowledge of the SIP, recognizing variability in call length, dynamic estimates of back-end server load for different SIP transactions. In this paper load balancer improves both throughput and response time. The SIP is a protocol of growing importance, with uses for VOIP, IPTV, audio conferencing, instant messaging. We present a detailed analysis of occupancy to show how our algorithms significantly reduce response time.

***Keywords*—**Load balancing, Performance, Response time, Server, 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. Wireless provides 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 telecommunications companies (telecoms)(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.

This paper presents and evaluates several algorithms for balancing load across multiple SIP servers. We introduce new algorithms that 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 are sent by the load balancer to the same server. The session state is created by the INVITE and 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 transaction, and taking advantage of this fact can result in more optimized SIP load balancing. The first session state is created by the INVITE transaction and is destroyed by the BYE Transaction.

Another key aspect of the SIP protocol is that different transaction types, most notably these INVITE and BYE transactions, can incur significantly different overheads on our systems, INVITE transactions are about 75% more expensive than BYE transactions. A load balancer can make use of this information to make better load-balancing decisions that 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. In addition, the best-performing algorithm takes into account the variability of call lengths, distinguishing 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. On our platform, 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 OpenSER open-source SIP server configured as a load balancer. Our evaluation is done using the 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. We experimentally evaluate SIP proxy server performance using micro-benchmarks meant to capture common SIP proxy server scenarios. We use standard open-source SIP software such as OpenSER and SIPp, running on an IBM Blade Center with Red Hat Enterprise Linux and Gigabit Ethernet connectivity [5]. We then discuss mechanisms and algorithms for controlling overload in these servers.

We found that performing overload control locally at a server provides a simple remedy for light cases of overload however it is ineffective in handling higher amounts of load [2].This paper makes the following contributions.

We evaluate our algorithms in terms of throughput, response time, and scalability, comparing them to several standard "off-the-shelf" distribution policies such as

round-robin or static hashing based on the SIP Call-ID. Our evaluation tests scalability up to 10 nodes.

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 evaluate the capacity of our load balancer in isolation to determine at what point it may become a bottle neck. We demonstrate throughput of up to 5500 calls per second, which in our environment would saturate at about 20 back-end nodes. 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 improve performance even further.

The remainder of this paper is organized as follows. Section II provides a brief background on SIP. Section III presents the design of our load-balancing algorithms, and Section IV describes their implementation. Section V overviews our experimental software and hardware, and shows our results in detail. Section VI discusses related work. Section VII presents our summary and conclusions and briefly mentions plans for future work.

II. RELATED WORK

A load balancer for SIP is presented in this paper requests are routed to servers based on the receiver of the call. A hash function is used to assign receivers of calls to servers. A key problem with this approach is that it is difficult to come up with an assignment of receivers to servers that result in even load balancing. This approach also does not adapt itself well to changing distributions of calls to receivers. Our study considers a wider variety of load-balancing algorithms and shows scalability to a larger number of nodes. The paper also addresses high availability and how to handle failures.

III. BACKGROUND

This section presents a brief overview of SIP. Readers familiar with SIP may prefer to continue to Section IV.

A. Overview of the Protocol

SIP is a signaling (control-plane) protocol designed to establish, modify, and terminate media sessions between two or more parties. The core IETF SIP specification is given in RFC 3261, although there are many additional RFCs that enhance and re ne the protocol. Several kinds of sessions can be used, including voice, text, and video, which are transported over a separate data-plane protocol. SIP does not allocate and manage network bandwidth as does a network resource reservation protocol such as RSVP that is considered outside the scope of the protocol [9]. 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

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 fi*nal (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. 2 illustrates a typical SIP message flow, where SIP messages are routed through the proxy. 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. In this example, a call is initiated with the INVITE message and accepted with the 200 OK messages. Media is exchanged, and then the call is terminated using the BYE message [6].

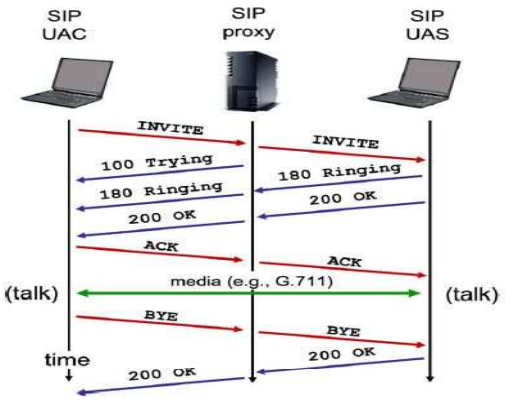

Fig. 1. SIP message flow.

C. SIP Message Header

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. In VoIP, SIP messages contain an additional protocol, the Session Description Protocol (SDP), 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. RFC 3261 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 [11].

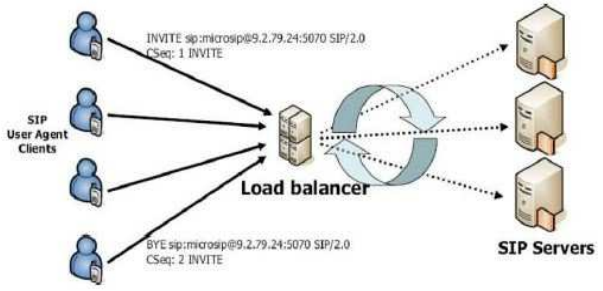

Fig. 2. System architecture.

IV. LOAD-BALANCING ALGORITHMS

This section presents the design of our load-balancing algorithms [1]. Fig. 3 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. The distinction between the various load-balancing algorithms presented in this paper is how they choose which SIP server to handle a request. Servers send SIP responses (e.g. 180 TRYING) to the load balancer, which then forwards the response to the client. we will also describe our workload generator (which is able to generate overload conditions at the server using a few client machines), and our overload model[4].

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

1)Call-Join-Shortest-Queue:

The 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 decrements the counter for the server. 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 TJSQ algorithm estimates the amount of work a server has left to do base 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 TLWL algorithm *ad*dresses 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. We experimented with several values for this ratio of relative cost. TLWL-2 assumes INVITE transactions are twice as expensive as BYE transactions and are indicated in our graphs as TLWL-2. We found the best performing estimate of relative costs was 1.75 these are indicated in our graphs as TLWL-1.75. Note that if it is not feasible to determine the relative overheads of different transaction types, TJSQ can be used, which results in almost as good performance as TLWL-1.75.TLWL estimates server load based on the weighted number of transactions a server is currently handling. For example, if a server is processing an INVITE (relative cost of 1.75) and a BYE transaction (relative cost of 1.0), the server has a load of 2.75.TLWL can be adapted to workloads with other transaction types by using different weights based on the overheads of the transaction types.

In addition, the relative costs used for TLWL could be adaptively varied to improve performance. We did not need to adaptively vary the relative costs because the value of 1.75 was relatively constant.

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

for load balancing Session Initiation Protocol(SIP) with Session Aware Request Assignment(SARA).

V. LOAD BALANCER IMPLEMENTATION

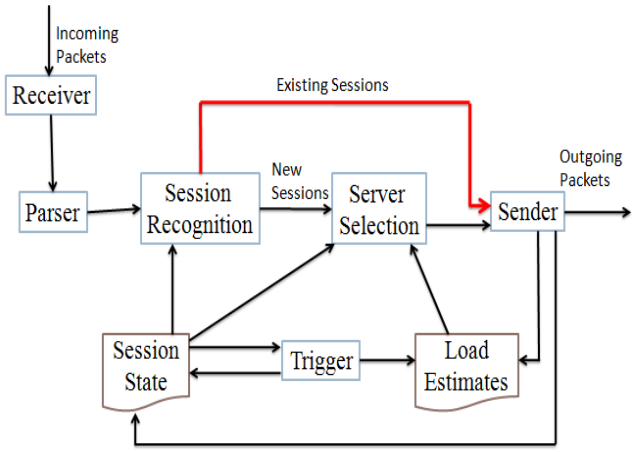

Fig.3. Load Balancer Structure

This section describes our implementation. Fig.3 illustrates the structure of the load balancer. The rectangles represent key functional modules of the load balancer, while the irregular shaped boxes represent state information that is maintained. The arrows represent communication flows. The receiver receives requests that are then parsed by the Parser. The Session Recognition module determines if the request corresponds to an already existing session by querying the Session State, which is implemented as a hash table. The Trigger module updates Session State and Load Estimates after a session has expired.

```
01:  h = hash call-id
02:  look up session in active table
03:  if not found
04:    /* don't know this session */
05:    if INVITE
06:      /* new session */
07:      select one node d using algorithm
08:      (TLWL, TJSQ, RR, Hash, etc)
09:      add entry (s,d,ts) to active table
10:      s = STATUS_INV
11:      node_counter[d] += w_inv
12:    /* non-invites omitted for clarity */
13:  else /* this is an existing session */
14:    if 200 response for INVITE
15:      s = STATUS_INV_200
16:      record response time for INVITE
17:      node_counter[d] -= w_inv
18:    else if ACK request
19:      s = STATUS_ACK
20:    else if BYE request
21:      s = STATUS_BYE
22:      node_counter[d] += w_bye
23:    else if 200 response for BYE
24:      s = STATUS_BYE_200
25:      record response time for BYE
26:      node_counter[d] -= w_bye
27:      move entry to expired table
28:  /* end session lookup check */
29:  if request (INVITE , BYE etc.)
30:    forward to d
31:  else if response (200/100/180/481)
32:    forward to client
```

Fig. 4. Load-balancing pseudo code.

TABLE I
HARDWARE TESTBED CHARACTERISTICS

Feature	Machine Type A	Machine Type B
Quantity	11	3
CPU	3.06 GHz	2.8 GHz
RAM	4 GB	2 GB
Kernel	2.6.9-55.0.6	2.6.9-11
Distro	RedHat AS 4.5	RedHat AS 4.5
Roles	Back-End Server, Load Balancer	Workload Generation

VI. 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. Performance is limited by the CPU processing power of the servers and not by memory. The average response time for each algorithm versus offered load measured for the INVITE transaction. Note especially that the -axis is in logarithmic scale. In this experiment, the load balancer distributes requests across eight back-end SIP server nodes. Two versions of Transaction-Least-Work-Left are used. For the curve labeled TLWL-1.75, INVITE transactions are 1.75 times the weight of BYE transactions. In the curve labeled TLWL-2, the weight is 2:1.The curve labeled Hash uses the standard OpenSER hash function, whereas the curve labeled FNV Hash uses FNV Hash. Round-robin is denoted RR on the graph.

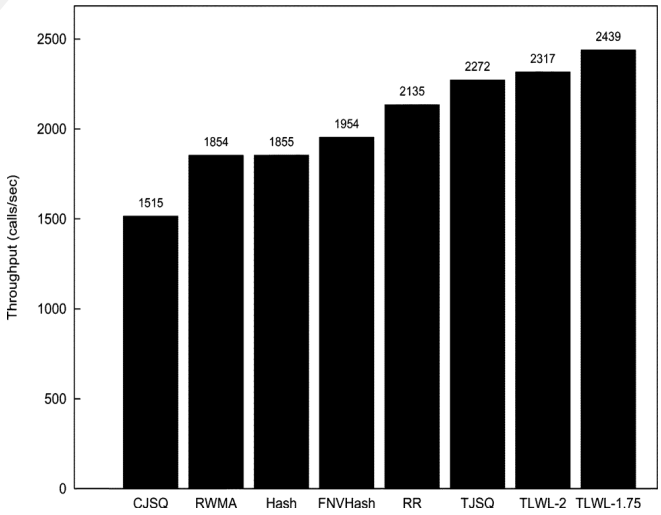

Fig. 5. Peak throughput of various algorithms with eight SIP servers.

b. Throughput

We now examine how our load-balancing algorithms perform in terms o f how well throughput scales with increasing numbers of back-end servers. In the ideal case, we would hope to see eight nodes provide eight times the single node performance. Recall that the peak throughput is the maximum throughput that 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 eight nodes. Fig. 8 shows the peak throughputs for the various algorithms using eight back end nodes. Several interesting results are illustrated in this graph.

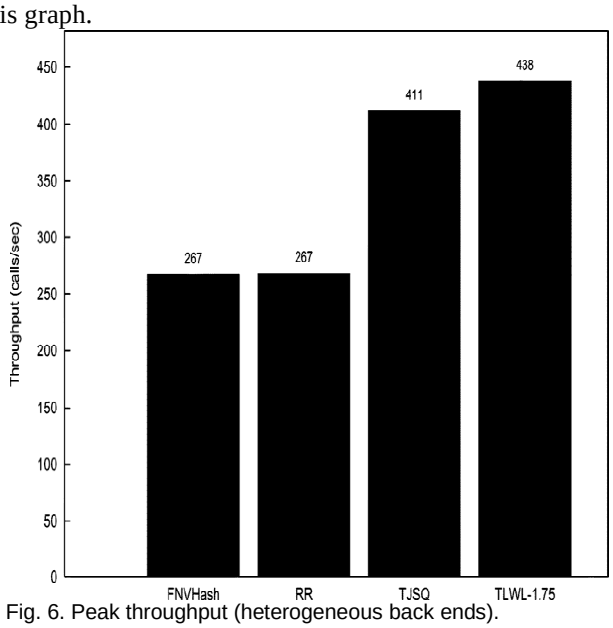

Fig. 6. Peak throughput (heterogeneous back ends).

c. Occupancy and Response Time

Given the substantial improvements in response time shown in graph, we believe it is worth explaining in depth how certain load-balancing algorithms can reduce response time versus others. We show this in two steps. First, we demonstrate how the different algorithms behave in terms of occupancy namely, the number of requests allocated to the system. The occupancy for a transaction assigned to a server is the number of transactions already being handled by when is assigned to it.

d. Heterogeneous Back Ends

In this section, we look at how our load-balancing algorithms perform when the back-end servers have different capabilities. In these experiments, the load balancer is routing requests to two different nodes. One of the nodes is running another task that is consuming about 50% of its CPU capacity. The other node is purely dedicated to handling SIP requests. Recall that the maximum capacity of a single server node is 300 cps. Ideally, the load balancing algorithm in this heterogeneous system should result in a throughput of about one and a half times this rate, or 450 cps.

e. Load Balancer Capacity

In this section, we evaluate the performance of the load balancer itself to see how much load it can support before it becomes a bottleneck for the cluster. We use five nodes as clients and five nodes as servers, which allow us to generate around 10000 cps without becoming a bottleneck.

VII. SUMMARY AND CONCLUSION

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 to ensure that SIP transactions are routed to the proper back end node that contains the appropriate session state. We presented three novel algorithms: CJSQ, TJSQ, and 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. TJSQ has the advantage that no knowledge is needed8 of relative overheads of different transaction types. The most significant performance differences were in response time. Under light to moderate loads, TLWL-1.75, TLWL-2, and TJSQ achieved response times for INVITE transactions that were at least five times smaller than the other algorithms we tested. Under heavy loads, TLWL-1.75, TLWL-2, and TJSQ have response time's two orders of magnitude smaller than the other approaches. For SIP applications that require good quality of service, these dramatically lower response times are significant. 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.

REFERENCES

[1] Hongbo Jiang, *Member, IEEE*, Arun Iyengar, *Fellow, IEEE*, Erich Nahum, *Member, IEEE*, Wolfgang Segmuller, Asser N. Tantawi, *Senior Member, IEEE*, *Member, ACM*, and Charles P.Wright "Design, Implementation, and Performance of a Load Balancer for SIP Server Clusters" IEEE/ACM transactions on networking, June 2012.

[2] V.Hilt and I.Widjaja, "Controlling overload in networks of SIP servers," in Proc. IEEE ICNP, Orlando, FL, Oct. 2008.

[3] Sandeep Sharma, Sarabjit Singh, and Meenakshi Sharma, "Performance Analysis of Load Balancing algorithms" World Academy of Science, Engineering and Technology 2008.

[4] C.Shen, H.Schulzrinne, and E.M.Nahum, "Session initiation protocol (SIP) server overload control Design and evaluation," in *Proc.IPTComm*, Heidelberg, Germany, Jul. 2008, pp. 149–173.

[5] 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.

[6] G.Ciardo, A.Riska, and E.Smirni, "EQUILOAD: load balancing Policy for clustered Web servers," Perform Evaluating, vol. 46, no.2-3, pp. 101–124, 2001.

[7] M.Aron, D.Sanders, P.Druschel, and W.Zwaenepoel, "Scalable Content-aware request distribution in cluster-based network servers," in Proc. USENIX Annu. Tech. Conf., San Diego, CA, Jun. 2000, pp. 323–336.

[8] L.Zhang, S.Deering, D.Estrin, S.Shenker, and D. Zappola, "RSVP: A new resource reservation protocol," *IEEE Commun. Mag.*, vol. 40, no. 5, pp. 116–127, May 2002.

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

[10] http://www.brekeke.com

[11] http://sipp.sourceforge.net