Implementing State Distribution Model in Asterisk Server

Ahmadreza Montazerolghaem[1], Mohammad Hossien Yaghmaee moghaddam[2], Sohrab Effati[3]
[1,2]Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
Ahmadreza.montazerolghaem@stu-mail.um.ac.ir, hyaghmae@um.ac.ir
[3]Department of Mathematical Sciences, Ferdowsi University of Mashhad, Mashhad, Iran
s-effati@um.ac.ir

***Abstract*—Requests in SIP (Session Initiation Protocol) usually pass through several servers. Since resources are considerably consumed to keep the state, state distribution problem is raised among several servers where the goal is to increase throughput, accessibility of servers and reduce overload. In this paper, the optimisation problem was formulated and implemented as a distributed algorithm on Asterisk proxy server which is a commercial proxy server and evaluated with Spirent traffic production device. It results in a more scalable SIP server which dynamically determines the number of SIP requests for which server is stateful and assigns state management to the server which is inferior to it for the remaining demands. The current SIP servers are statically configured such that they are stateless or stateful and don't give optimal efficiency.**

Keywords—Overload control; linear programming; Asterisk proxy server; state; optimisation problem.

I. INTRODUCTION

This Session Initiation Protocol (SIP) [1] is an application layer protocol which is used for varied class of applications such as VoIP ،IM and Presence [2] and 3GPP [3] for establishing and ending contact. In SIP, communication demands are exchanged by applying a mass of servers. Each of these servers performs some of the contact operations including host detection, direction, state maintenance and authentication.

We worked on Asterisk [4] which is an open source SIP server in low contact loads to measure the spent processing sources experimentally when a VoIP contact is performed. Our measurements indicate that the spent sources considerably change dependent on the provided function. Particularly creation, retention and deletion of contact state are of the considerable consumers of CPU resources. The server which retains the state is called stateful server and the server which doesn't retain the state is called stateless server. This state is used by server to retain information of a set of messages. The retained state is used for supply of a varied class of function including reduction of unnecessary retransmission and provision of accounting service. We also measured CPU usage in Asterisk server for retaining the state when the contact load increases. Our measurements show again that the maximum load which Asterisks can support in stateful condition is lower than the load which it can support in stateless condition.

Static configuration of a set of servers leads to non-optimal throughput in order to be stateful or stateless relative to all contacts (what is performed today).

By distributing state among the servers, throughput can be increased considerably. In this case, each server retains the state only for a fraction of demands and remains stateless for others. If one can ensure that state of each contact demand is managed only in one of the system servers, then, this system will be completely stateful for set of the demands which pass through it.

In this paper, the following cases are studied:

- ✓ Studying "consumption of CPU resources" in an Asterisk server.
- ✓ Introducing "state management" as a main source of CPU.
- ✓ "State distribution optimization"
- ✓ "Implementation and evaluation" on Asterisk server which distributes contact state to improve total throughput in the system.

Rest of this paper is organized as follows: in Section 2, we review the previous literature. In Section 3, a detailed evaluation of SIP server is given. In Section 4, state dynamic distribution formulation is mentioned. In Section 5, we evaluate its performance. In Section 6, conclusion of this paper is given.

II. BACKGROUND

Paper [5] deals with overload control methods in SIP proxy server and uses OPNET software for measuring throughput. In papers [6] and [7] SIP is practically implemented along with TCP and UDP transmission protocol and OpenSER is used to obtain efficiency results. SIPstone [8] is series of benchmark in which various criterions are proposed for evaluating proxy server powers, redirect server and registrar in answering SIP requests. In [9] another benchmark is presented for measuring the effect of operating system, hardware configuration, database, and selected transmission layer on SIP efficiency. In [9] practical experiments are accomplished on four types of proxy implementation which are different in both thread management and memory allocation method. The results of these experiments show that the effective parameters in proxy efficiency could be classified in two parts: parameters related

978-1-4799-5359-2/14/$31.00 ©2014 IEEE

to protocol such as message length, length variability, and irregularity of excess load, and parameters related to the type of server implementation e.g. how to allocate sources of operating system to transactions. Also in [10] similar studies about the effect of operating system and type of proxy implementation on SIP efficiency are done. In [11] it is studied about the efficiency of signaling of SIP in establishing VOIP connections using JAIN SIP API and by considering the effect of call duration and call rate on the delay of connection establishment between two end-to-end user agents. In [12] the effect of delay of user's answer on SIP server's efficiency is analyzed by introducing a tool called SIPperformer.

III. EVALUATING CPU CONSUMPTION OF ASTERISK SERVER

In this Section, we evaluate Asterisk in low contact loads to obtain CPU consumption rate for different functions which it provides. Our experiment consists of a set of users (Spirent Abacus 5000 device) which sends their requests to another class of users through an Asterisk proxy. We observed function of this proxy with OProfile software [13] and used Spirent Call Generator both as originate and as terminate. This set allows us to generate an approximate rate of 10,000 contacts per second without conversion of Spirent into a bottleneck. As shown in Figure 1, Spirent traffic generator can be used in this scenario to imitate both originate and terminate. In this experiment, we put Asterisk between its two sides for test.

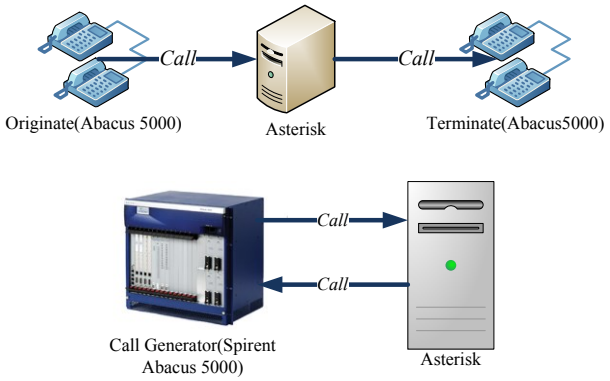

Fig. 1. Place of Asterisk and Spirent Abacus 5000 in test scenario

In these tests, Asterisk server was configured into five conditions as follows. Each condition indicates a service which server provides for contact. In addition, Asterisk server looks up in its dictionary to translate from URI to end point IP address.

Different conditions are as follows:

1: Stateless with No Lookup: as a result of dealing with contact message, no state relating to a contact is maintained. This message also contains enough information such as IP address in end point SIP URI; therefore, there is no need for Lookup.

2: Stateless with Lookup: here, no contact state is kept as the previous case but a lookup is performed to map URI into IP address.

3: Transaction Stateful with Lookup: in this state , a lookup is performed to map URI of message into an IP address. In addition, the state is kept only for each transaction separately.

4: Dialog Stateful with Lookup: here, IP address is looked up. In addition, state is kept for the entire contact term which includes several transactions.

5: Dialog Stateful with Authentication: this condition is like the previous case in addition that it authenticates proxy permissions of the users.

Asterisk server is configured in one of the states mentioned above with any execution and two users of Spirent device start and stop contact through this server for 10 min with rate of one contact per second . During this term, OProfile profiles different parts of Asterisk function. Results of these executions are shown in Figure 2.

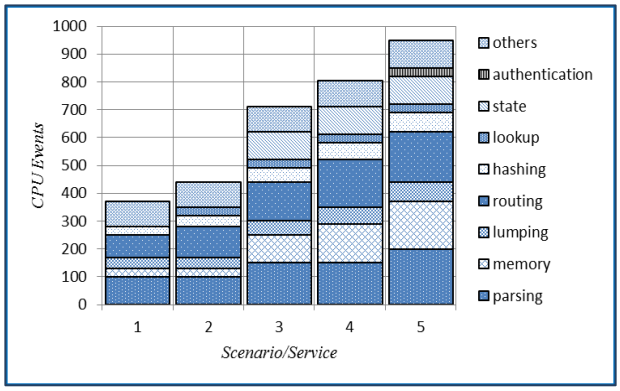

Fig. 2. Cost of different functions of Asterisk server

In the most basic configuration where server is stateless and doesn't perform any lookup, the consumed CPU is almost one third of the state in which a server looks up, keeps transaction state and performs authentication operations. Considering this issue, it is evident that costs of a simple contact depends on the complexity of service provided by SIP server and is considerably variable.

In comparison to no lookup column, all of the other cases (from stateless to authentication) have lookup processing load which includes either query in a database or a cache. Similarly, we also see increase of CPU consumption to keep the state and authenticate permission. Most functions performed by the server increase concurrently with the service. Particularly, we also see considerable increase of costs relating to message parsing, memory and state in case of increase of service. State management requires allocation and deallocation of memory and this increases memory of processor in the last three scenarios. This additional state increases parsing and increases processing in memory of server. Considering the Diagram, it can be found that low demand rate of 1 call per second, "dialog stateful" and "Transaction stateful" are 2 times and 1.75 times as expensive as the stateless case, respectively. Therefore, state management is a good option for distribution and in this paper, we show how stateful and stateless servers behave in case of increaser of contact rate.

Goal of the test which is mentioned in rest of this Section is to obtain behaviour of stateful and stateless servers in case of

increasing load. Asterisk proxy server is configured for execution in the following two cases:

1) Stateless with Lookup and
2) Transaction Stateful with Lookup

To obtain full behaviour of server, contact load was increased until server reached 100% of CPU utilisation. To ensure that server has been saturated only due to utilisation of CPU, we configured Asterisk with enough memory (1024 megabytes) and used Gigabyte Ethernet Interfaces on IP-PBX laboratory network[1].

Abacus 5000 starts load forwarding from 20 contacts per second and increases it with stepwise method (20 contacts per second). Asterisk server is saturated when CPU utilization reaches 100% and throughput doesn't increase with increasing contact. In saturation threshold, 500 Server Busy messages are considerably increased and contact demands are resent by the users. Results are shown in Figure 3.

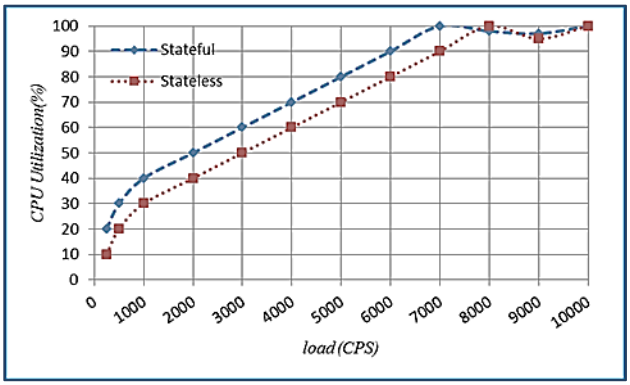

Fig. 3. increase of CPU utilization with increasing contact rate

As shown in Figure 3, CPU utilization increases with statefully configured server faster than the statelessly configured server with increasing contact rate. The stateful server is saturated in approximate rate of 7000 contacts per second but stateless server is saturated in approximate rate of 8000 contacts per second. This difference between the stateful and stateless servers is the basis for obtaining higher throughput through distribution of state functions.

IV. STATE DISTRIBUTION OPTIMIZATION

A state distribution algorithm should meet the following requirements:

Firstly, contact state should be kept by a node in contact demand direction. Other nodes assign this contact in stateless manner; therefore, it doesn't bear processing load relating to state management. Secondly, it should be dynamically determined what node should keep contact state in a route so that the system is able to start that contact as far as there are stateful resources for dealing with an entry contact.

SIP servers can be placed as any type of network topology and this set of nodes can be expressed as a graph. Contact demands enter system through one of the nodes of this graph, pass through a set of nodes with links inside the graph and exist from a proxy node. The last node assigns this demand to internet infrastructure or a contacting user. A topology can have many entry nodes. This is equivalent to having a virtual source node o which generates contacts and sends it to the entry nodes. Similarly, there may be several exit nodes in the system and this is equivalent to a z pit node to which all exit nodes direct their contacts. Advantage of this method is that the network can be expressed as a single- source, single-pit topology and a clearer formulation can be achieved without loss of generality.

In a system, consider i server and a special contact demand which pass through node i. There is an upstream node from which this contact has been directed to node i. We name this node u. this contact indicates that all of the contacts have been directed from u to i or C_{ui}. For the rest of this discussion, all traffic sentences indicate contact (load) rate. Therefore, C_{ui} means contact load from u to i. C_{ui} has two components:

One of the contacts for which state has been kept before (C_{ui}^{BSF}) and another one is the contact for which the state should be kept ($C_{ui}^{\widetilde{SF}}$). Therefore, we have $C_{ui} = C_{ui}^{BSF} + C_{ui}^{\widetilde{SF}}$. Now, sum of entry flow to node i or C_i means sum of all C_{ui} flows ($C_i = \sum_{u \in US_i} C_{ui}$). Similarly , for sum of contact loads for which state has been kept before and sum of the contact loads for which the state should be kept , we will have the expressions which are $C_i^{BSF} = \sum_{u \in US_i} C_{ui}^{BSF}$ an $C_i^{\widetilde{SF}} = \sum_{u \in US_i} C_{ui}^{\widetilde{SF}}$, respectively.

US_i means sum of all possible upstream servers for node i. In node i, entry load of C_i will be distributed among all possible downstream servers. Then, $C_i = \sum_{d \in DS_i} C_{id}$ and C_{id} include three components according to flow conservation law :

- ✓ C_{id}^{FSF}: contact flow from i to d for which state has been kept before by a node before i.
- ✓ C_{id}^{SF}: contact flow from i to d for which node i has kept state.
- ✓ $C_{id}^{\widetilde{SF}}$: contact flow from i to d for which state should be kept.

"Flow cojnstraints" require the following Equations:

$$C_i^{BFS} = \sum_{d \in DS_i} C_{id}^{FSF} \text{ (i)}$$

$$C_i^{\widetilde{SF}} = \sum_{d \in DS_i} C_{id}^{SF} + \sum_{d \in DS_i} C_{id}^{\widetilde{SF}} \text{ (ii)}$$

For the downstream node d, the flow rate for which state has been kept before from node i is equal to $C_{id}^{BSF} = C_{id}^{FSF} + C_{id}^{SF}$. For traffic from upstream node u to i, a distribution have occurred similarly. Hence, we have $C_{ui}^{BSF} = C_{ui}^{FSF} + C_{ui}^{SF}$. This guarantees that decision variables will be C_{id}^{FSF} ، C_i^{SF} and $C_i^{\widetilde{SF}}$ in each node. The constraints which were mentioned above guarantee that the state is exclusively dealt with from both sides. However, it should be guaranteed that state should be kept at least in one of the nodes until demands exit from the system. Therefore, the number of contacts for which state should be kept for each flow from exitnode k to virtual pit node is equal to zero ($C_{kz}^{\widetilde{SF}} = 0$).

[1] http://ippbx-lab.um.ac.ir/

From Section 3, we know that CPU utilization rate for stateful inspection of demands is different from that for stateless inspection of demands. The number of the contacts which is inspected in stateful and stateless manner in node i includes $\sum_{d\in DS_i} C_{id}^{SF}$ and $\sum_{d\in DS_i}(C_{id}^{\widetilde{SF}} + C_{id}^{FSF})$, respectively. Therefore, if it is assumed that CPU utilization rate results from assignment of demand in node i which we show with U_i^{SF} depends on the number of demands which are inspected in stateful manner, then, we will have $U_i^{SF} = f^{SF}(\sum_{d\in DS_i} C_{id}^{SF})$.

Similarly, CPU utilization rate in non-stateful condition is equal to $U_i^{SL} = f^{SL}(\sum_{d\in DS_i} C_{id}^{\widetilde{SF}} + C_{id}^{FSF})$. Sum of CPU utilization in node i is equal to sum of these two expressions and this sum should not exceed 100% and if they are normalized, then, it means that we have $U_i^{SF} + U_i^{SL} \le 1$.

The above Equations form "system constraints". The goal is to obtain a state distribution so that throughputis maximized. Throughput mainly means contact flow from virtual source node o to all entry nodes. In this pint, state should be kept for all contacts. $C_{oa}^{FSF} = 0$ and $C_{oa}^{SF} = 0$ for entry node a. then, we try to maximize sum of flows from virtual node o to entry nodes or $\sum_{a\in entry\ node} C_{oa}^{\widetilde{SF}}$. Linear optimization problem is as follows:

$$\text{Maximize} \sum_{a\in entry\ node} C_{oa}^{\widetilde{SF}} \quad (I)$$

Subject to

$$C_{oa}^{SF} = 0, C_{oa}^{FSF} = 0, \forall\ a \in \text{entry node}$$

$$\sum_{u\in US_i} (C_{ui}^{FSF} + C_{ui}^{SF}) = \sum_{d\in DS_i} C_{id}^{FSF} \quad (II)$$

$$\sum_{u\in US_i} C_{ui}^{\widetilde{SF}} = \sum_{d\in DS_i} C_{id}^{SF} + \sum_{d\in DS_i} C_{id}^{\widetilde{SF}} \quad (III)$$

$$C_{kz}^{\widetilde{SF}} = 0, \forall\ k \in \text{exit node}$$

$$f^{SF}\left(\sum_{d\in DS_i} C_{id}^{SF}\right) + f^{SL}\left(\sum_{d\in DS_i} C_{id}^{\widetilde{SF}} + C_{id}^{FSF}\right) \le 1 \quad (IV)$$

Now, solve these optimization formulas will help us determine to what extent state should be kept for maximization of throughput. Figure 3 shows that stateful utilization and stateless utilization are linear. Therefore, these functions can be estimated as $f^{SF}(x) = \frac{x}{TH\ Stateful}$ and $f^{SL}(x) = \frac{x}{TH\ Stateless}$, where $TH_{Stateful}$ and $TH_{Stateless}$ are saturation limits of stateful and stateless server.

In our tests, $\approx 7000\ TH_{Stateful}$ and $\approx 8000\ TH_{Stateless}$ were obtained. In this case, optimisation formulation means a linear programming (LP) problem which can be solved effectively. Now, this formulation can be used for calculation of optimal throughput for scenario of two series servers in case of having threshold values. Optimal answer is that each server keeps 3940 contacts per second as stateful and the remaining keeps 3940 contacts per second as stateless. In this case, total throughput is equal to 7880 contacts per second which is higher than throughput in static configuration (approximately 7000 contacts per second). The reason for this increase of throughput is that both servers are equally utilised, hence, higher throughput is obtained. Therefore, solving optimisation problem will result in an optimal answer for throughput. Answer of the linear programming problem includes values of C_{id}^{FSF} ، C_{id}^{SF} and $C_{id}^{\widetilde{SF}}$ for each node i. given utilization constraints, constraint (IV) can be rewritten as follows:

$$\left(\left[\left(\sum_{d\in DS_i} C_{id}^{SF}\right)/TH_{Stateful}\right] + \left[\left(\sum (C_{id}^{\widetilde{SF}} + C_{id}^{FSF})\right)/TH_{Stateless}\right]\right) \le 1 \quad (V)$$

As mentioned before, we have $C_{id} = C_{id}^{FSF} + C_{id}^{SF} + C_{id}^{\widetilde{SF}}$. By substituting $C_{id}^{FSF} + C_{id}^{\widetilde{SF}}$ in V and after sorting sentences, we have:

$$\sum_{d\in DS_i} C_{id}^{SF} \le \left(1 - \frac{\sum_{d\in DS_i} C_{id}}{TH_{Stateless}}\right) \Big/ \left(\frac{1}{TH_{Stateful}} - \frac{1}{TH_{Stateless}}\right) \quad (VI)$$

Equation (VI) shows that sum of the states which should be kept is an entry load function. As a result, server can keep All demands which are not stateful yet as stateful as far as the entry flow is less than stateful saturation limit ($TH_{Stateful}$). Once entry flow passes through $TH_{Stateful}$ ،the server will start leaving the state. Once the entry flow passes through $TH_{Stateful}$, each of the servers should assign state management to the downstream servers so that sum of the kept states in that server doesn't exceed $TH_{Stateful}$. Therefore, the upstream servers can assign state into the downstream servers until we reach exit nodes. Since these nodes don't have any downstream route to assign state to them, they continue keeping state by increasing the number of contacts to get close to maximum utilization. In this point, they dispatch a message indicating "overload condition" to the upstream servers and when all downstream routes are saturated and the server is saturated, this server dispatches a massage of overload condition to the upstream servers and finally the system reaches total saturation.

V. Configuration and Evaluation of Results

For evaluation, we utilized configuration of "two series servers". This main topology is a subset of topology of "several series proxies" which firstly provides structural blocks for more complex topologies and also can achieve an insight into topology of larger network. In addition, it also indicates topologies of server in real world. According to Figure 4, a class of users sends Spirent of their demands to the first proxy in this test and this proxy assigns these demands to the second proxy. The second proxy delivers contact demands to Spirent again and rest of the contact route follows this pattern as well. Scenario was executed for two times. In the first execution, proxies are statically configured and in the second execution, proxies execute dynamic distribution algorithm.

Results of throughput are shown in Figure 5. Static configuration is saturated in approximate rate of 6700 call per second and dynamic distribution is saturated in approximate rate of 8150 call per second. In other words, it improves performance by about 15%. For configuration including "three series servers", a similar trend was also observed. When dynamic distribution is executed, we see that we can expect

improvement of performance by 15 to 20% compared with static configuration for most of these main configurations.

Results of average response time measurement are shown in Figure 6. Effects of saturation on increase of response time can be observed at time of using static configuration in approximate rate of 6500 contacts per second above. Dynamic configuration is able to keep responses times low to higher rate. Once it passes through this rate, response times will be elongated considerably. The reason is that all of the lost demands should be issued again in the entire route and this increases mean response time.

VI. CONCLUSION

We experimentally evaluated performance of a SIP server (Asterisk) under several scenarios. Based on this study, we defined state management problem and studied a mathematical model for inferring an optimal answer. This case gives us an insight for development of a more scalable server through state dynamic distribution among a class of servers. We evaluated this algorithm compared with preconfigured static algorithms which are available today and showed that the maximum contact throughput could be increased by 15 to 20%.

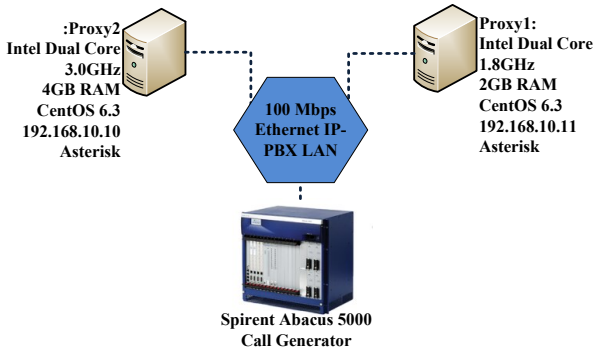

Fig. 4. Testbed of two proxies

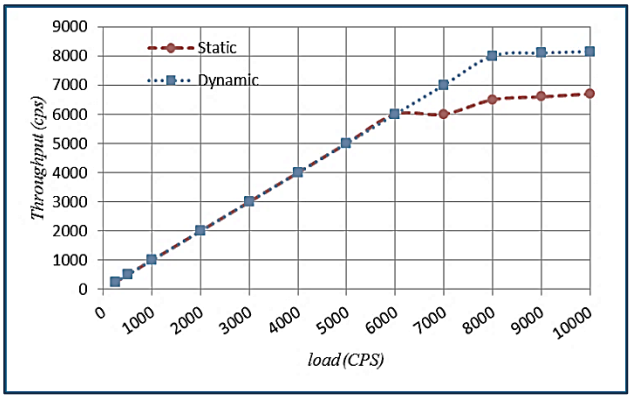

Fig. 5. Throughput in two cases of dynamic and static distribution

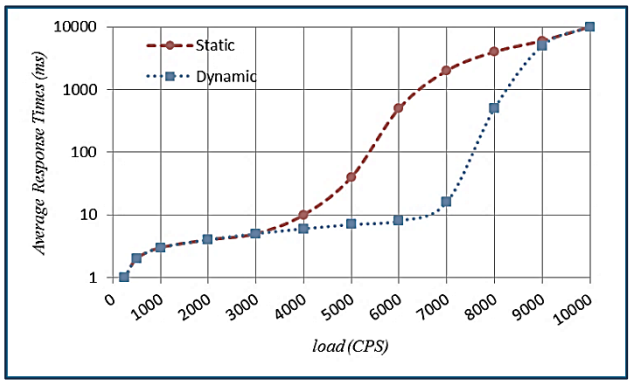

Fig. 6. Mean contact time in two cases of dynamic and static distribution

VII. REFERENCES

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, *et al.*, "SIP: Session Initiation Protocol," ed: RFC Editor, 2002.

[2] J. Rosenberg, "A Presence Event Package for the Session Initiation Protocol (SIP)," in *RFC 3856*, ed, August 2004.

[3] G. Patel and S. Dennett, "The 3GPP and 3GPP2 movements toward an all-IP mobile network," *Personal Communications, IEEE,* vol. 7, pp. 62-64, 2000.

[4] *http://www.asterisk.org/*.

[5] V. Hilt and I. Widjaja, "Controlling overload in networks of SIP servers," in *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*, 2008, pp. 83-93.

[6] S. Charles and S. Henning, "On TCP-based SIP server overload control," in *Principles, Systems and Applications of IP Telecommunications* ed. Munich, Germany: ACM, 2010, pp. 71-83.

[7] M. N. Erich, T. John, and P. W. Charles, "Evaluating SIP server performance," *SIGMETRICS Perform. Eval. Rev.* , vol. 35, pp. 349-350, 2007.

[8] H. Lindholm, "A Control Plane Benchmark for Telecommunications Signalling Applications," presented at the LinuxConf Europe, Cambridge, England, 2007.

[9] H. G. S. S. Narayanan, "SIPstone: Benchmarking SIP Server Performance," ed. New York: Department of Computer Science, Columbia University, 2002.

[10] W. Stephan, S. Michael, K. Sebastian, and W. Stefan, "Measurement of the SIP parsing performance in the SIP express router," in *Proceedings of the 13th open European summer school and IFIP TC6.6 conference on Dependable and adaptable networks and services*, ed. Enschede, The Netherlands: Springer-Verlag, 2007, pp. 103-110.

[11] S. S. Gokhale and L. Jijun, "Signaling performance of SIP based VoIP: a measurement-based approach," in *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, 2005, pp. 5 pp.-765.

[12] S. Montagna and M. Pignolo, "Performance Evaluation of Load Control Techniques in SIP Signaling Servers," in *Systems, 2008. ICONS 08. Third International Conference on*, 2008, pp. 51-56.

[13] *http://oprofile.sourceforge.net/news/*.