

A Scalable Architecture for SIP Infrastructure using Content Addressable Networks

Ramrajprabu Balasubramanian

Injong Rhee

Jaewoo Kang

Department of Computer Science
North Carolina State University
Raleigh, NC 27695

Abstract---Session initiation protocol (SIP) provides call establishment functions for VoIP. These functions are highly CPU-intensive and hence need to be distributed over multiple servers in order to support the peak workload. Existing approaches to this problem are unfortunately not scalable or fault-tolerant, incurring high maintenance overheads or introducing a single point of failure. This paper presents a proof-of-concept design and analysis of a scalable, robust architecture for SIP infrastructures using a content addressable network (CAN) model, called CASIP (CAN-based SIP). The performance study of CASIP using real implementation of a SIP stack and NS-2 shows that the proposed system distributes the SIP processing (both update and lookup) load over multiple nodes very effectively without incurring much routing and maintenance overhead.

I. INTRODUCTION

Session Initiation Protocol (SIP) [1] is a popular protocol for session establishment in many Internet applications (e.g., push-to-talk [2], Internet telephony, presence [9], instant messaging [9]). SIP endpoints are known as SIP User Agents or SIP UAs. SIP client loosely denotes SIP end points where SIP UAs run, such as SIP phones and soft phones. SIP (proxy) servers are intermediate network elements between the end points and engage in the routing of SIP messages from one SIP UA to another UA based on a logical SIP address. SIP servers also perform functions of authentication, authorization, media trans-coding, signaling compression, and billing, and also acts as a gateway to other types of telephony networks such as SS7. A logical SIP address (or SIP URI) consists of a username and a domain and identifies a SIP UA.

SIP UAs belonging to a particular domain register their locations with the SIP Registrar of that domain by means of a REGISTER message. The Registrar saves the location information of that particular UA in a location database. SIP servers refer to that location database for address resolution between logical SIP URI destinations and physical locations (or IP addresses). A single domain ranges from several thousands to several hundreds of millions of subscribers. For example, many wireless carriers are considering push-to-talk applications in their network. A typical wireless carrier network has a huge subscriber base: currently, several US wireless carriers have tens of millions of subscribers while

several Chinese wireless carriers serve more than hundreds of millions of subscribers.

To see the scalability issue of SIP VoIP networks, let us briefly review the SIP transaction load. Telephony networks are designed to sustain their performance under the peak load, typically, of an arrival rate of 1-2 calls per hour per subscriber on a busy occasion like Mother's day. Each call consists of a call setup transaction and a call tear down transaction, accounting for a total of 5 messages per call. Also SIP UAs register their location with the SIP Registrar as frequently as once every 5 minutes. The IP addresses of clients are typically dynamically assigned and abrupt IP address changes are common in the wireless Internet. So this registration interval must be kept small so that the location information maintained by the SIP network is consistent.

Based on the above figures, we can estimate that the peak load from the SIP registrations and call setups amounts to about 100,000 SIP messages per second (about 50,000 SIP transactions) for a 10 million subscriber base. Such high call origination rates exact high load on SIP servers due to the inherent text based nature of SIP. Commercially available SIP stacks can only process up to a few hundred transactions per second on a single server node, thus requiring the SIP processing load to be distributed over multiple nodes. Furthermore, the location updates cause about 30,000 updates per second to the location database for a 10 million subscriber base. This update rate cannot be handled even with a completely memory resident database (commercially available memory resident database can support up to only several thousands of updates per second [3]). The capacity of the location database is often restricted by the capacity of CPU and memory available in the location database server. As the size of the network grows further, these loads are proportionally increasing.

SIP vendors have attempted to increase the scalability of the SIP network by distributing the SIP processing loads over multiple servers. However, each distributed SIP server still needs to access the location database in order to process call transactions. Such a location service is handled typically with either a shared location database [10, 11] where a central location server handles all the location queries and updates, or a replicated location database [12] where each database is replicated to multiple locations to speed up the access. These approaches, however, fail to address the

problem adequately – the replicated database approach does not solve the high update problem as updates have to be reflected to all servers for consistency, and the shared database approach introduces a single point of failure.

Yet, another approach is to partition the database into multiple segments and distribute these segments over multiple nodes in some hierarchical fashion. Although this architecture solves load distribution and failure locality, it can incur too much overhead for reconfiguration. As new nodes are added, the entire database must be reconfigured and reloaded, which could cause an unacceptable disruption of service. We may also try to partition the database based on the geographic proximity of the location entries, or using some naming hierarchies in a similar way that is done in DNS. These schemes, however, still introduce critical points of failure and do not completely distribute the load.

Observing that the location database being a bottleneck, one important question arises: can we adapt existing large-scale distributed database systems to address the scalability problem of the SIP network? In fact, a large body of research exists in the database community introducing highly available, sophisticated distributed databases and replication techniques (e.g., see R*[18], SDD-1[17] for distributed databases and [19] for replication techniques). However, these systems are developed for the different types of applications such as banking and airline reservations where transaction consistency and recoverability are critical. Such systems devote substantial resources to the transaction management and recovery (e.g., ACID transactions [14], two-phase locking [15] and three-phase commit [16] etc.) that are largely irrelevant to the requirements of our SIP architecture. As a result, these systems cannot scale to the level of the subscriber base in a typical real-world deployment scenario.

In this paper we present the design and evaluation of a scalable and robust SIP architecture using a Content Addressable Network (CAN) model [5]. In this architecture, an overlay network of SIP servers forms a CAN network where SIP messages are routed via distributed hash tables based on SIP addresses. Each SIP server is responsible for a unique sub-region of the hash space and maintains a separate location database corresponding to its assigned region. Updates and lookups to the location information of a user are handled by the SIP server mapped to the region that contains the hash value of the SIP address of the user. Each server maintains a small list of neighbors in the overlay network where a SIP message can be routed to reach the final destination. Since each server maintains local state information about its neighbors, a failure of a server does not affect the entire network. In addition, when a server crashes, its neighbors dynamically take over the hash space assigned to that failed server and handle future requests to that space. We call our architecture CASIP (CAN-based SIP).

II. CASIP DESIGN

In this section, we describe our CASIP architecture. Our choice of CAN among DHT techniques is only incidental and can be replaced by other DHT networks such as Chord [7] and Pastry [8]. We use structured P2P networks as opposed to unstructured ones such as Gnutella [4] because they guarantee to serve a query as long as the queried item is contained in the network. In a carrier-grade VoIP network, this feature is essential.

A. CASIP architecture and basic operations

In CASIP, the content is a (key, value) pair where key is a SIP URI and value is the location information (e.g., IP addresses or telephone numbers) of the SIP UA with that URI. For simplicity we chose a 2-dimensional torus, and a hash function $H:k \rightarrow (x,y)$ where k is the SIP URI for a query and (x,y) is the Cartesian coordinate of the location in the hash space for the content corresponding to key k (any simple hash function that gives b bits as an output for $H(k)$ can be split into $b/2$ bits for x and the $b/2$ for y). The hash space is then divided into zones with a two dimensional boundary. Each CASIP node is mapped to a unique zone and stores the location information of SIP UAs whose SIP URIs are mapped to that zone by the hash function. Each CASIP node “owns” the zone, stores the content database associated with the zone, and performs CAN routing functions as well as the functions of a stateless SIP proxy server.

Queries and routing. When a SIP UA has a SIP request (i.e., INVITE, SUBSCRIBE, REGISTER, etc.) to send, it sends that request to a CASIP node in the network. When a CASIP node receives a SIP request, the node first checks if it holds the zone containing the hash value of the SIP URI in the ReqURI of the SIP request. If it holds the zone that contains the hash value, the CASIP performs the regular stateless SIP proxy server functions for that request: the node either forwards the SIP Request to the destination UA after resolving its location from the SIP URI, or sends an unavailability response in case of location information not being present. (In case of call forwarding, the server creates a new request with the forwarded URI and the request is routed again in the same way as the original request.) If the SIP URI does not fall into that zone, the SIP request is forwarded to a neighbor whose zone has the shortest Cartesian distance to the destination coordinate among its neighbors. When forwarding the request, the CASIP node encapsulates the SIP request into a CAN message type which contains the information about the hashed value (i.e., the coordinate). As this message is routed, it does not go through the SIP stack at each routing CASIP node (so no SIP processing overhead incurred). When it gets to the final destination CASIP node, the message is changed into a SIP message. Figure 1 shows the call graphs of the INVITE request in a regular SIP network and also in the CASIP network. The additional overhead due to CASIP comes from CASIP routing, and additional SIP processing at the first CASIP node.

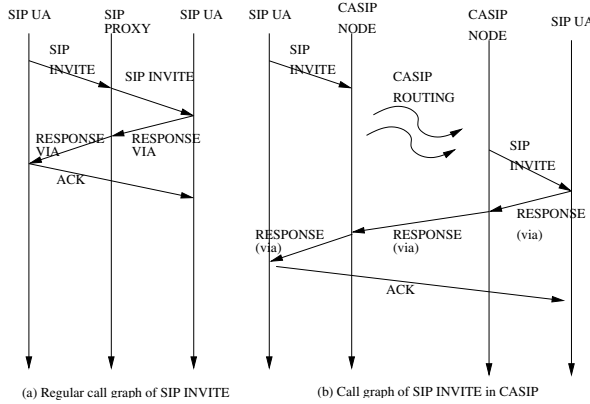


Figure 1: Call graphs of the INVITE request.

Locating a CASIP node. In CASIP, a SIP UA finds a CASIP node in the same way that SIP finds its proxy SIP server (either via multicast or DNS). By the CASIP routing strategy, any node can route the request correctly to the final destination. When a UA discovers a CASIP node, it uses that node for all the future requests. Using the same node to send SIP Requests into the network allows us to use caching to optimize the number of hops the SIP message traverses within the network (more discussion on this later in this section).

Join and leave. CASIP nodes also need a method to be able to join and leave the CASIP network gracefully. We follow the same technique used in CAN: when a node joins the network, it chooses a random point in the hash space and sends a join request to any of the available CASIP nodes. That request is routed to the zone in which that random point belongs to. The CASIP node that owns that zone, then, splits the zone into two halves and transfers one half and the location information stored in that half to the node that is joining. It also informs the joining node of its neighbors and the routing information so that the new node can effectively become a part of the CASIP network. While leaving the CASIP network, a node sends a leave request containing the information about its associated zone to its neighbors. This zone is then merged with one of its neighbors and the location database entries previously stored by the leaving node are transferred to the node that takes over that zone.

Bootstrapping. The CASIP network starts with just one node owning all of the hash space, followed by the other nodes joining the CASIP network. In order for a new node to join the network it needs to know at least one CASIP node in the network. This can be achieved using the same SIP mechanism to discover a SIP proxy. When a new node tries to join, it gets a list of existing CASIP nodes either through SIP multicast address or using a DNS server that holds a list of a few nodes that are already in the CASIP network. Note that this list does not have to be complete and it is sufficient to have some random nodes in the network so the cost of maintaining this list is minimal.

Failure handling. A CASIP node can route messages even if some nodes in its neighbors fail. Failure can be

detected using the ICMP destination unreachable messages or a lack of periodic zone information messages for a timeout period (In CAN, this timeout period is set proportionally to the size of zones [5]). Using the same mechanism described for node leaving, we can also have a neighbor take over the zone of the failed node. However all the location information stored in the zone corresponding to a failed node are lost. As SIP UAs frequently refreshes their location information through the periodic REGISTER requests, the location information lost during the failure can be completely recovered after a short delay (typically, 5 minutes). One way to enhance the availability is to replicate the database to a few locations by maintaining more than one hash function. In this approach, the same content is replicated to multiple nodes using different hash spaces. Since the location database is redundantly replicated, node failures will not lead to loss of location information stored by that node.

Optimization. The major disadvantages of the CASIP approach are the following. (1) SIP messages traverse several overlay hops before reaching the final destination. A d-dimensional CAN network can give about $d/4(n^{1/d})$ hops. Thus, the hop count increases with the number of nodes. (2) As CAN consists of overlay networks, a neighboring node may not be a physical neighbor. Thus, the network incurs high delay overhead. Each hop in the SIP network introduces additional call setup delays. This delay becomes unacceptable under a large number of nodes in the networks. The acceptable call setup delays are in the range of 3 seconds to 8 seconds for Internet telephony [6]. Push-to-talk applications require about 1.5 second call setup delays [3]. Our goal is to limit the call setup delays within these bounds. Our approach to this problem is to use caching techniques to optimize the path length. The first CASIP node encountered by a SIP Request is modified to look up its cache first to resolve the location of a destination SIP UA. A cache miss results in the CASIP node routing the SIP Request using normal CAN routing.

We use a technique called zone caching. This technique caches the coordinates of a zone and the IP address of the CASIP node serving that zone. The d-dimensional coordinates of the zone containing the destination SIP URI is recorded by the serving node in a special 'Via' header parameter inserted by the serving node. The first node caches this zone coordinate and the address (note that the entry is not created per URI). When a request arrives, the first node initially hashes the request URI and then lookups an entry in its cache for the zone containing the hash value of the URI. If the entry is found, it sends the request directly to the IP address corresponding to that matching entry. Even if the zone information is invalid due to addition or deletion of nodes, the correct information can be refreshed after one cache miss. In this approach, the number of entries in the cache is kept to at most the number of zones (or the number of nodes) in the network and also rerouting in the CASIP network solves the cache inconsistency problem.

III. PERFORMANCE RESULTS

In this section, we evaluate the performance of the CASIP architecture through simulation. We build a CASIP network using a network simulation (NS-2) on a topology generated from a topology generator tool called GT-ITM [13]. The topology consists of 100 backbone router nodes and inter-transit delays are set randomly between 40ms and 80ms by the tool and the transit-to-stub delays are set to 10ms. We divide the topology into five geographical regions and attach 10 CASIP nodes to each region through stub nodes. We attach two SIP traffic generators to each region and these traffic generators create an aggregate load equivalent to the one generated by one million subscribers based on the peak load described in the introduction. The peak load consists of approximately 3,000 SIP call setup and tear down messages and 3,000 registration update transactions, all per second. These numbers are taken from a peak estimate of 2 calls per hour per subscriber and one registration update per 5 minute per subscriber. In terms of the number of read and write transactions, they constitute about 600 read transactions and 3,000 update transactions per second. This load will generate approximately 9,000 SIP messages per second (5 messages per call and 2 messages per update). Using this simulation setup we study the SIP processing loads on each node, and the average number of hops traversed by a SIP message, and the database update load on each node. We also study the latency in call setup as the number of nodes in the CASIP network increases, and also the number of messages exchanged as nodes join and leave. The results are compared to those from the shared database approach and the replicated database approach.

Effect of CASIP on SIP and database load. In this experiment, we measure the CPU load that SIP message processing and database operations have on SIP servers as we increase the number of SIP servers. We measure the CPU load by the number of messages that each server on average has to handle. Figure 2 shows that result. Generally, we observe that the number of messages handled by each node decreases as the number of servers increases, more so for CASIP and shared database and much less so for the replicated approach. This is because the location database size on each of the nodes in the replicated approach remains the same since the whole database is replicated on each of the nodes; therefore, the same number of updates has to go to all servers, while lookups can be handled locally. CASIP shows much better load distribution in terms of the number of messages processed per node with the increase in number of nodes. But the routing overhead causes the number of messages to be about three times of that of the shared database approach. The optimized version of CASIP that uses zone caching can reduce this overhead almost by half, so that its total message count comes very close to twice as many as that of the shared database.

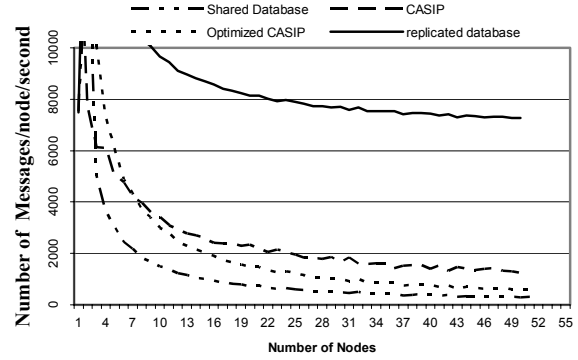


Figure 2. The number of messages processed per node by each scheme.

We also compute the size of subscriber population that can be supported as we increase the number of servers. This is computed assuming that a single server can handle up to 500 SIP transactions per second while processing 5000 updates and 10,000 lookups per second. These numbers are based on our optimistic estimate on the performance of memory resident database and SIP servers. The result is shown in Figure 3. We found that the routing overhead of optimized CASIP drops the capacity of servers by approximately 10 to 15%.

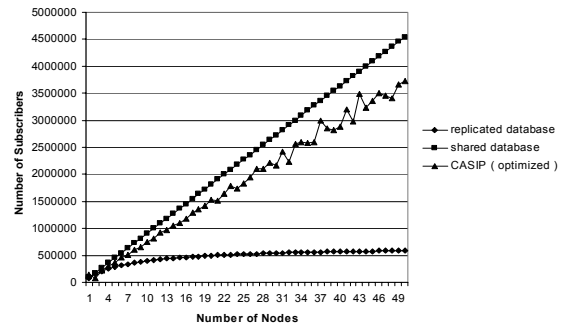


Figure 3. The size of subscriber population supported by each scheme as the number of servers increases.

Path length and call setup delays in CASIP. The path length of a SIP message within the SIP network is the number of CASIP nodes it visits in the network before it gets to the final destination. The average path length in the shared and the replicated database approaches equals zero since the location information of any SIP UA is available to them at that very first node. In CASIP, SIP messages are always forwarded to neighbors with the smallest Cartesian distance to the destination SIP URI's hash. Hence the path length of a SIP message is the Cartesian distance between the first CASIP nodes zone and the zone containing the location information of the destination SIP URI. The average path length in a d -dimensional CAN with n nodes is $O(n^{1/d})$ [5]. However, when we applied zone caching, the path length converged to 1. Moreover, using the link delays in network simulator, we measured the call setup delays in our topology.

As predicted, the increased setup delays of CASIP were due to the increased path length. However, use of zone caching kept the delay always below 250 ms.

Reconfiguration overheads. We measured the number of messages exchanged between CASIP nodes when a new node joined or left. Since addition or deletion only affected the routing tables of the immediate neighbors of that particular node, the observed number of zone information exchanges between the neighbors was very small. The message overhead was also independent of the total size of the network.

Effect of node failure. We intentionally failed a varying number of CASIP nodes in the network and observed the effect of the failure on the call completion ratio. In a test where 50 nodes were used and one node was disabled, the call completion ratio dropped by less than 2% before the node was re-enabled again. Once the hash space of that node was taken over, the SIP registration quickly re-established the location of the SIP URIs in that zone, taking the call completion ratio back to 100%. We also note that even during the failure of a node, calls that originally flowed through that node are successfully routed around the failed node. We also increased the number of failed nodes to 10% and 20% of the network. We observed that the call completion ratios dropped quite rapidly to 88% and 67% respectively. However, these numbers seemed to be reasonable considering the magnitude of failures.

IV. CONCLUSION

The purpose of this paper is to show the feasibility of using DHT for SIP processing, especially handling the registration overhead. It is still an early stage to claim that DHT must be adopted and more research effort must be expended to mature the technique. However, we demonstrate in this paper that the main overheads of DHT can be overcome with a simple caching scheme, and also with some minimal overhead of routing, SIP can benefit from effective and low-cost failure handling and reconfiguration. These features prove to be essential for incremental deployment requirements---an important economical issue for carriers. In order to make CASIP a practical solution for SIP, we need further improvement. The most notable one is to incorporate the physical location and topology information in the construction of DHT networks. This has been an area of intense research in P2P community lately and will be likely overcome in a foreseeable future. We want the readers to look at this paper as a starting point for applying P2P concepts to SIP and encourage them for more research in the area.

- [1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "IP: Session Initiation Protocol," RFC 3261prop, June 2002.
- [2] Open Mobile Alliance – Push-to-talk Over Cellular Working Group http://www.openmobilealliance.org/tech/wg_committees/poc.html.
- [3] "In-Memory Tables Performance Benchmark" - <http://www.aidaim.com/articles/dbmementests.php>.

- [4] Y. Chawathe *et al.*, "Making Gnutella-like P2P Systems Scalable," *ACM SIGCOMM 2003*.
- [5] Sylvia Ratnasamy *et al.*, "A Scalable Content-Addressable Network," *ACM SIGCOMM 2001*
- [6] T.Eyers, H Schulzrinne. "Predicting Internet Telephony Call Setup Delay," Technical Report, Columbia University, 1999.
- [7] I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, H.Balakrishnan "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM , 2001*.
- [8] A. Rowston and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, 2001.
- [9] B. Campbell, Ed, J. Rosenberg, H. Schulzrinne, C. Huitema D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging," RFC 3428, December 2002
- [10] Ubiquity Software Corporation, "The Ubiquity SIP Application Server," http://www.ubiquity.net/pdf/SIP_AS_brochure.pdf
- [11] Microsoft Live Communications Server 2003, Architecture, Deployment and Operations. Microsoft IT Showcase. <http://prade.microsoftemea.com/content/Communication%20and%20Collaboration/1>
- [12] The Cisco SIP Proxy Server, Product Overview, http://www.cisco.com/univercd/cc/td/doc/product/voice/sipproxy/adm/ingd/ver1_0/overview.htm
- [13] Ellen W. Zegura, Kenneth Calvert and M. Jeff Donahoo. [A Quantitative Comparison of Graph-based Models for Internet Topology](#). IEEE/ACM Transactions on Networking, December 1997
- [14] J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques," Morgan Kaufmann, 1992
- [15] K. Eswaran, J. Gray, R. Lorie, and I. Traiger, "The Notions of Consistency and Predicat Locks in a Database System," *Communications of the ACM*, 19(11):624-633, 1976
- [16] D. Skeen, "Nonblocking commit Protocols," In Proc. ACM SIGMOD Conf. on the Management of Data, 1981
- [17] J. Rothnie and N. Goodman, "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases," In Proc. Berkeley Workshop on Distributed Data Management and Computer Networks, 1977
- [18] R. Williams , D. Daniels , L. Haas , G. Lapis , Lindsay P. Ng , R. Obermarck , P. Selinger , A. Walker , P. Wilms , R. Yost, "R*: An overview of the architecture," *Readings in database systems*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1988
- [19] Michael J. Carey, Miron Livny, "Conflict Detection Tradeoffs for Replicated Data," *ACM Trans. Database Syst.* 16(4): 703-746(1991)