Failover and Load Sharing in SIP Telephony

Kundan Singh and Henning Schulzrinne
Department of Computer Science, Columbia University
{kns10,hgs}@cs.columbia.edu

Abstract

We apply some of the existing web server redundancy techniques for high service availability and scalability to the relatively new IP telephony context. The paper compares various failover and load sharing methods for registration and call routing servers based on the Session Initiation Protocol (SIP). In particular, we consider the SIP server failover techniques based on the clients, DNS (Domain Name Service), database replication and IP address takeover, and the load sharing techniques using DNS, SIP identifiers, network address translators and servers with same IP addresses. Additionally, we present an overview of the failover mechanism we implemented in our test-bed using our SIP proxy and registration server and the open source MySQL database.

Keywords: Availability; scalability; failover; load sharing; SIP

Contents

1 Introduction

The Session Initiation Protocol (SIP) [1] is a distributed signaling protocol for IP telephony. The SIP-based telephony services have been proposed as an alternative to the classical PSTN (public switched telephone network) and offers a number of advantages over the PSTN [2]. Traditionally, telephony service is perceived as more reliable than the Internet-based services such as web and email. To ensure wide acceptance of SIP among carriers, the SIP servers should demonstrate similar quantifiable guarantees on service availability and scalability. For example, PSTN switches have a "5 nines" reliability requirement, i.e., available for 99.999% time, which implies at most 5 minutes outage a year.

The SIP proxy servers are more light-weight compared to PSTN switches because they only route call signaling messages without maintaining any per-call state. The SIP proxy server of a *domain* is responsible for forwarding the incoming requests destined for the logical address of the form *user@domain* to the current transport address of the device used by this logical entity, and forwarding the responses back to the request sender. Consider the example shown in Fig. 1. When a user, Bob, starts his SIP phone, it registers his unique identifier *bob@home.com* to the SIP server in the *home.com* domain. The server maintains the mapping between his identifier and his phone's IP address. When another user, Alice, calls *sip:bob@home.com*), her phone does a DNS (Domain Name Service) lookup for the SIP service record of *home.com* and sends the SIP call initiation message to the resolved server IP address. The server "proxies" the call to Bob's currently registered phone. Once Bob picks up the handset, the audio packets can be sent directly between the two phones without going through the server. Further details [2, 3] of the call are skipped for brevity.

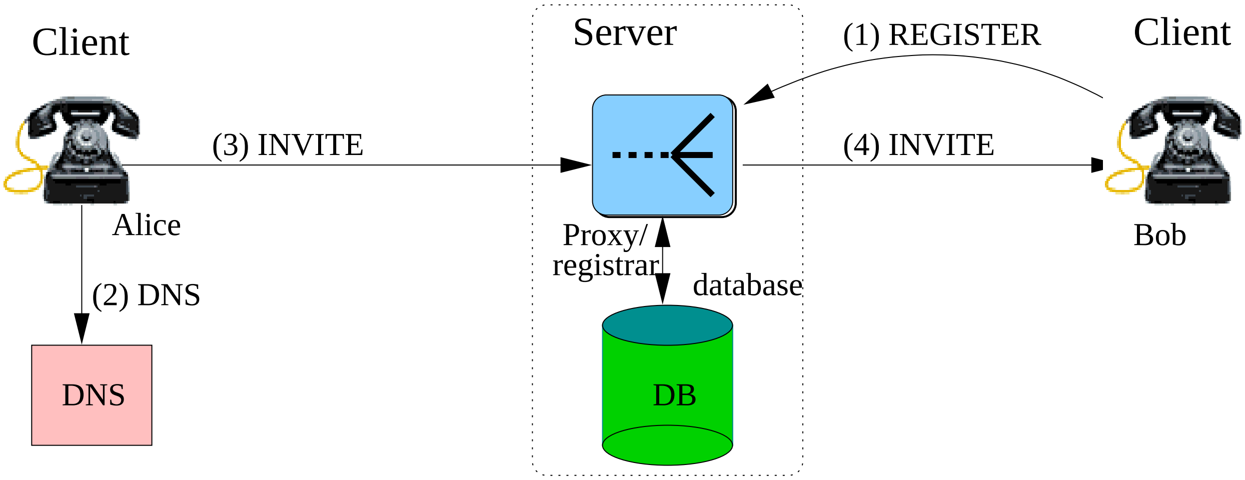


Figure 1: An example SIP call

If the server fails due to some reason, the call initiation or termination messages cannot be proxied correctly. We can improve the service availability by adding a second server that automatically takes over in case of the failure of the first server. Secondly, if there are thousands of registered users and the single server cannot handle the load, then a second server can work along with the first server such that the load is divided between the two. We describe some of these failover and load sharing techniques for SIP servers. These techniques also apply beyond telephony, for example, for SIP-based instant messaging and presence that use the same SIP servers for registration and message routing.

2 Related Work

Failover and load sharing for web servers is a well studied problem [4, 5, 6, 7]. TCP connection migration [8], IP address takeover [9] and MAC address takeover [10] have been proposed for high availability. Load sharing via connection dispatcher [11] and HTTP content or session-based request redirection [12, 13, 10] are available for web servers. Some of these techniques such as DNS-based load sharing [14, 15] also apply to other Internet services like email and SIP. Although SIP is an HTTP like request-response protocol, there are certain fundamental differences that make the problem slightly different. For example, SIP servers can use both TCP and UDP transport, the call requests and responses are usually not bandwidth intensive, caching of responses is not useful, and data update (REGISTER message) and lookup (INVITE message) ratio may be comparable unlike common read-dominated database and web applications.

The IETF's Reliable Server Pooling (Rserpool) working group is developing an architecture for the name resolution of service and aggregate server access that does not require maintaining hard state in the front end load distributor [16, 17]. This can also be used for the SIP server pool.

The SIP-based telephony services exhibit three bottlenecks to scalability: signaling, data and gateway. The signaling part deals with higher request processing rate for the SIP servers. The data part is the real-time media interaction between the endpoints and the gateway part deals with the optimal placement of media gateways and switching components [18]. This paper focuses on the signaling part only. SIP allows redirecting a request to a less loaded server using the 302 response, or transferring an existing call dialog to a less loaded endpoint or gateway [1, 19]. Sparks [20] proposes signaling the upstream senders to reduce the request rate when the downstream servers are overloaded. The SIP server can utilize the failover and load sharing research done in databases. For example, MySQL allows replication and clustering [21, 22].

We describe and compare some of these techniques in the context of SIP. We also present an overview of our implementation of failover and describe some practical issues.

3 High Availability: Failover

High availability is achieved by adding a backup component such as the SIP server or user record database. Depending on where the failure is detected and who does the failover, there are various design choices: client-based, DNS-based, database failover and IP takeover.

3.1 Client-based failover

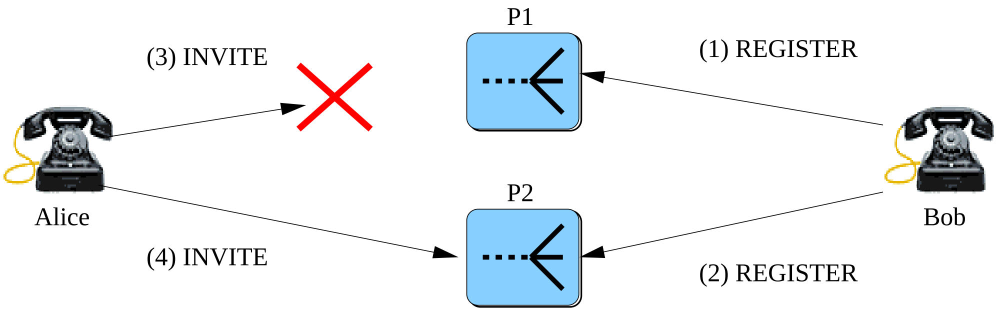
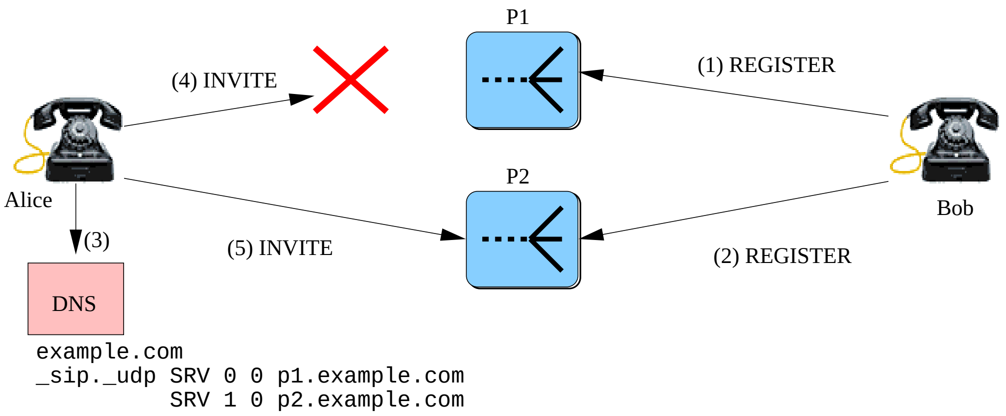


Figure 2: Client-based failover

In the client-based failover (Fig. 2), Bob's phone knows the IP addresses of the primary and the backup servers, P_1 and P_2. It registers with both, so that either server can be used to reach Bob. Similarly, Alice's phone also knows about the two servers. It first tries P_1, and if that fails it switches to P_2.

All failover logic is built into the client. The servers operate independent of each other. This method is used by the Cisco IP phones [23]. Configuring the phones with the two server addresses works well within a domain. However, the phone or the server needs to use DNS to locate the backup server when sending the call invitation to a user in another domain.

3.2 DNS-based failover



```
example.com
_sip._udp SRV 0 0 p1.example.com
          SRV 1 0 p2.example.com
```

Figure 3: DNS SRV-based failover

DNS SRV-based failover is the most clean and hence, preferred way, to failover. In this method, Alice's phone can retrieve the DNS SRV [14] record for *_sip._udp.home.com* to get the two server addresses (Fig. 3). In the example, P_1 will be preferred over P_2 by assigning a lower numeric priority value to P_1.

Alternatively, dynamic DNS can be used to update the A-record for *home.com* from the IP address of P_1 to P_2, when P_1 fails. P_2 can periodically monitor P_1 and update the record when P_1 is dead. Setting a low time-to-live (TTL) for the A-record bindings can reduce the failover latency due to DNS caching [24].

3.3 Failover based on database replication

Not all the SIP phones are capable of registering with multiple servers. Moreover, to keep the server failover architecture independent of the client configuration, the client can register with only P_1, which can then propagate the registration to P_2. If a database is used to store the user records, then replication can be used as shown in Fig. 4. Bob's phone registers with the primary server, P_1, which stores the mapping in the database D_1. The secondary server, P_2, uses the database D_2. Any change in D_1 is propagated to D_2. When P_1 fails, P_2 can take over and use D_2 to proxy the call to Bob. There could be small delay before D_2 gets the updated record from D_1.

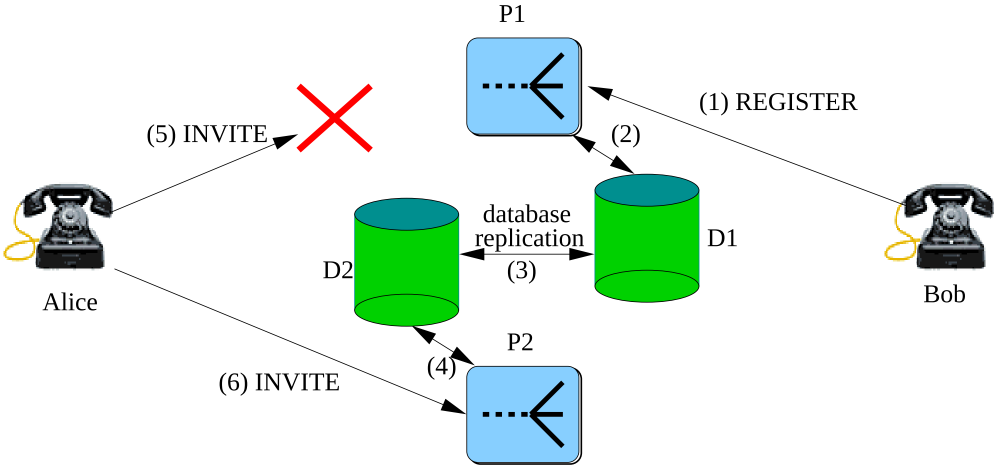


Figure 4: Failover based on database replication

3.4 Failover using IP address takeover

If DNS-based failover cannot be used due to some reason (e.g., not implemented in the client), then IP takeover [9] can also be used (Fig. 5). Both P_1 and P_2 have identical configuration but run on different hosts on the same Ethernet. Both servers are configured to use the external master database, D_1. The slave D_2 is replicated from D_1. The clients know the server IP address as P_1's 10.1.1.1 in this example.

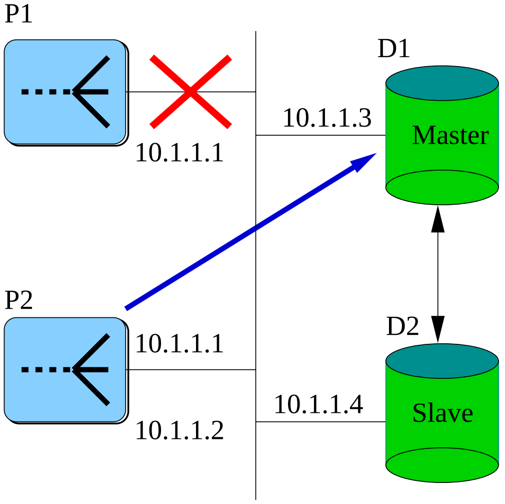


Figure 5: When the primary server fails

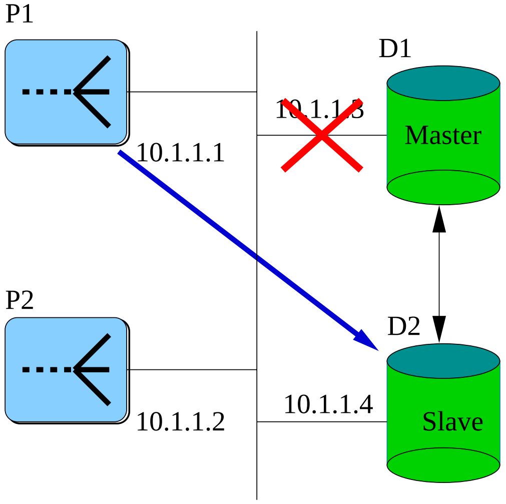


Figure 6: When the master database fails

P_2 periodically monitors the activity of P_1. When P_1 fails, P_2 takes over the IP address 10.1.1.1. Now, all requests sent to the server address will be received and processed by P_2. When D_1 fails, P_1 detects and switches to D_2 (Fig. 6). IP takeover is not used by D_2 since the the SIP servers can be modified to switchover when D_1 fails. There can be a small failover latency due to the ARP cache.

The architecture is transparent to the rest of the network (clients and DNS) and can be implemented without external assumptions. However, if the replication is only from the master to the slave, it requires modification in the SIP server software to first try D_1, and if that fails use D_2 so that all the updates are done to the master server. To avoid replicating the database, P_1 can propagate the REGISTER message also to P_2.

Alternatively, to avoid the server modification, the server and the associated database can be co-located on the same host as shown in Fig. 7. If the primary host fails, both P_2 and D_2 take over. P_1 always uses D_1, whereas P_2 always uses D_2.

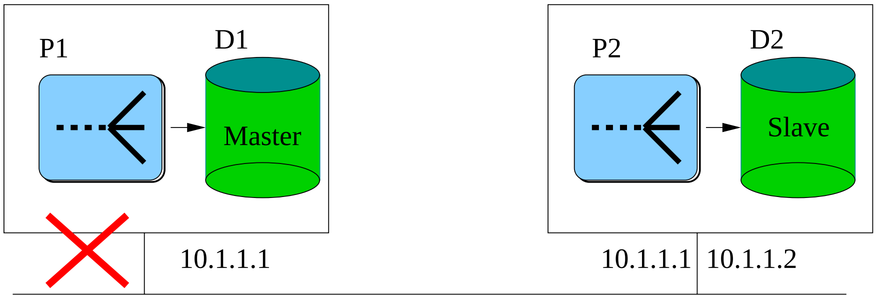


Figure 7: IP takeover: co-located database and proxy server

3.5 Reliable server pooling

In the context of IETF's Reliable Server Pooling architecture [25], Fig. 8 shows the client phone as the pool user(PU), P_1 and P_2 as the pool elements (PE) in the "SIP server pool", and D_1 and D_2 as PEs in the "Database pool". P_1 and P_2 register with their home name server, NS_2, which supervises them, and informs the other name servers (NS) about these PEs. Similarly, D_1 and D_2 also register with the NS. The SIP servers are the pool users of the "Database pool". A pool element is removed from the pool, if it is out of service.

When the client wants to contact the "SIP server pool", it queries one of the name servers, NS_1, to get the list of P_1 and P_2 with relative priority for failover and load sharing. The client chooses to connect to P_1 and sends the call invitation. If P_1 fails, the client detects this and sends the message to P_2. For stateful services, P_1 can exchange state information with another server, P_2, and return the backup server, P_2, to the client in the initial message exchange. This way the client knows which backup server to use in the case of failure. P_1 can also give a signed cookie similar to HTTP cookie to the client, which sends it to the new failover server, P_2, in the initial message exchange. This is needed for call stateful services such as conferencing, but not for SIP proxy server failover.

The SIP server, P_1, queries the NS to get the list, D_1 and D_2, for the "Database pool". D_1 and D_2 are backed up and replicated by each other, so they can return this backup server information in the initial message exchange.

The primary limitation is that this requires new protocol support for name resolution and aggregate server access in the clients. A translator can be used to interoperate with the clients that do not support reliable server pooling. However, this makes the translator as a single point of failure between the client and the server, hence limiting the reliability. Secondly, the name space is flat unlike DNS heirarchy, and is designed for a limited scale (e.g., within an enterprise), but may be combined with wide area DNS based name resolution. More work is needed in that context.

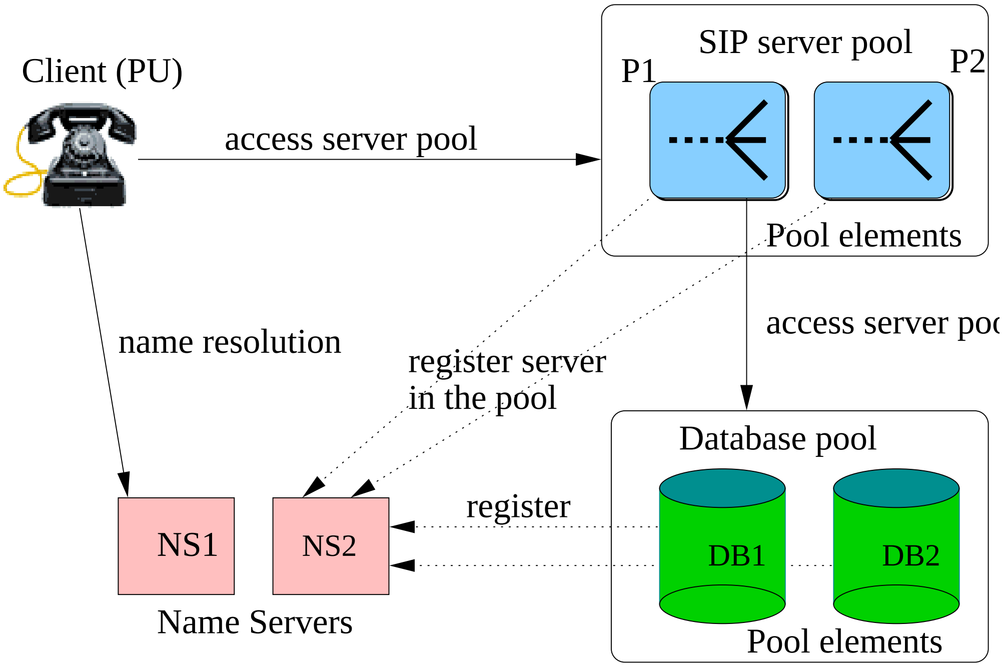


Figure 8: Reliable server pooling for SIP

4 Scalability: Load sharing

In failover, the backup server takes over in case of failure whereas in load sharing all the redundant servers are active and distribute the load among themselves. Some of the failover techniques can also be extended to load sharing.

4.1 DNS-based load sharing

The DNS SRV [14] and NAPTR [15] mechanisms can be used for load sharing using the priority and weight fields.

```
example.com
_sip._udp 0 40 a.example.com
          0 40 b.example.com
          0 20 c.example.com
          1  0 backup.somewhere.com
```

The above DNS entry indicates that the servers a, b, c should be used if possible (priority 0), with backup.somewhere.com as the backup server (priority 1) for failover. Within the three primary servers, a and b are to receive a combined total of 80% of the requests, while c, presumably a slower server, should get the remaining 20%. Clients can use weighted randomization to achieve this distribution.

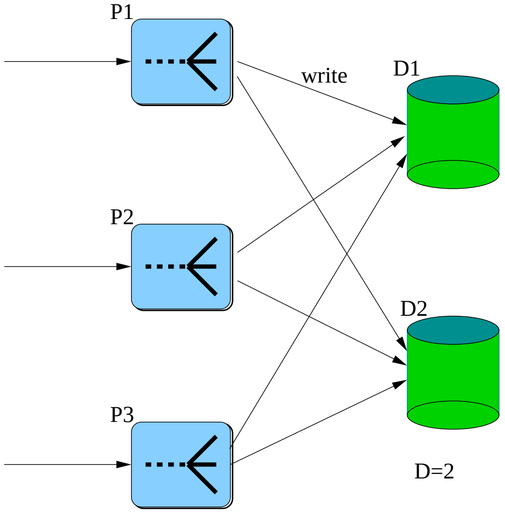


Figure 9: DNS-based

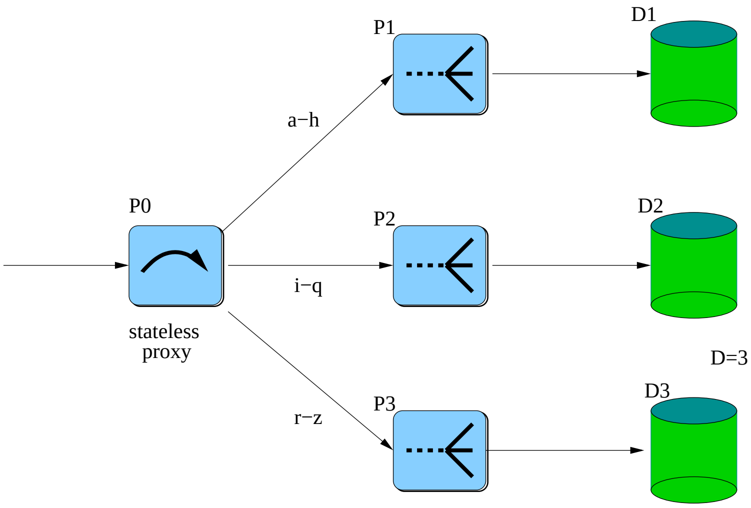


Figure 10: Identifier-based load sharing

However, simple random distribution of requests is not sufficient since the servers need to access to the same registration information. Thus, in the example above, each server would have to replicate incoming REGISTER requests to all other servers or update the common shared/replicated database(s). In either case, the updates triggered by REGISTER quickly become the bottleneck. The SIP phones typically do REGISTER refresh once an hour, thus, for a wireless operator with one million phones, it has to process about 280 updates per second. Our SIP server handles about 300 registration requests per second and about 90 proxy requests per second [26] on a dedicated Sun Netra XI, a 500 MHz UltraSPARC IIe with 128 MB of memory, running Solaris 2.9

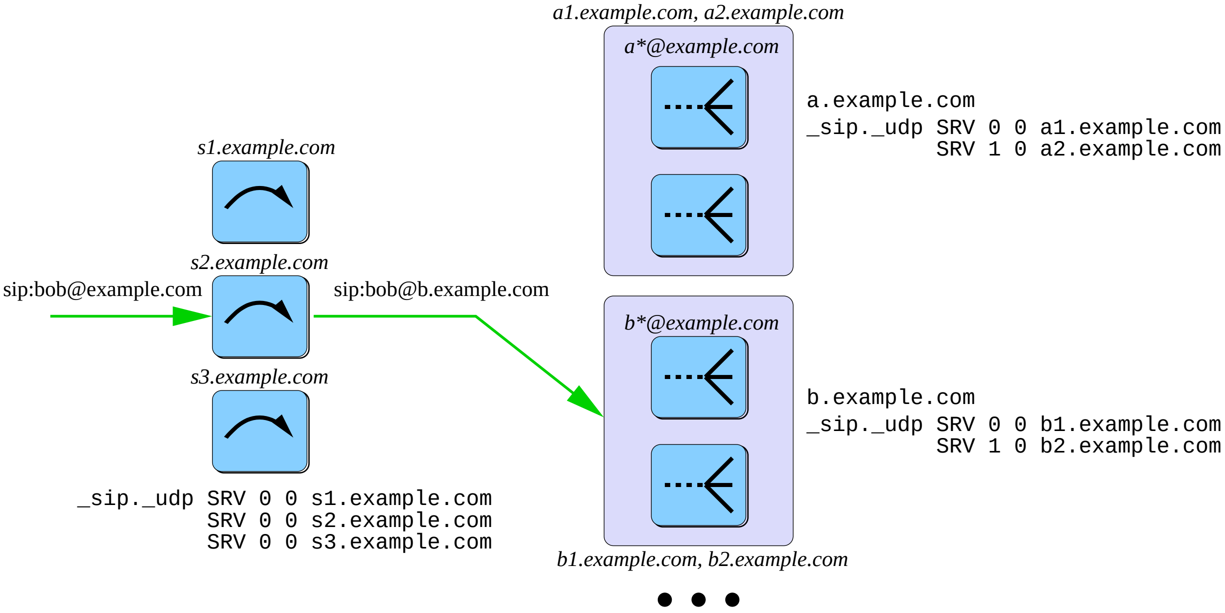
Fig. 9 shows an example with three redundant servers and two redundant databases. For every REGISTER, it performs one read and one write in the database. For every INVITE-based call request, it performs one read from the database. Every write should be propagated to all the D databases, whereas read can be done from any available database. Suppose there are N writes and $r * N$ reads, e.g., if same number of INVITE and REGISTER are processed then $r = 2$. Suppose, the database write takes T units of time, and database read takes $t * T$ units, where typically $t = 1/2$. Total time per database will be $((tr/D) + 1)TN$.

4.2 Identifier-based load sharing

In this method (Fig. 10), the user space is divided into multiple non-overlapping groups, e.g., based on the first letter of the user identifier. For example, P_1 handles `a-h`, P_2 handles `i-q` and P_3 handles `r-z`. A high speed stateless server (P_0), proxies the call request to P_1, P_2 and P_3 based on the identifier. If a call is received for destination *bob@home.com* it goes to P_1, whereas *sam@home.com* goes to P_3. Each server has its own database and does not need to interact with the others.

The only bottleneck may be the stateless proxy. However, since stateless proxies are much faster than stateful proxies, the method works reasonably well. Moreover, the layer-2 approach for load sharing among stateless proxies can be used as described in Section 4.4.

Suppose N, D, T, t and r are as defined in the previous section. Since each read and write operation is limited to one database and assuming uniform distribution of requests to the different servers, total time per database will be $((tr+1)/D)TN$. Since the writes do not have to be propagated to all the databases and the database can be co-located on the same host with the proxy, it reduces the internal network traffic.



```
a.example.com
_sip._udp SRV 0 0 a1.example.com
          SRV 1 0 a2.example.com
```

```
b.example.com
_sip._udp SRV 0 0 b1.example.com
          SRV 1 0 b2.example.com
```

```
_sip._udp SRV 0 0 s1.example.com
          SRV 0 0 s2.example.com
          SRV 0 0 s3.example.com
```

Figure 11: SIP scalability using DNS SRV

We can combine the two methods in a two-stage scaling architecture, shown in Fig. 11. The first set of proxy servers selected via DNS SRV perform stateless request routing to the particular second-stage cluster based on the identifier. The cluster member is again determined via the DNS SRV. The second-stage server performs the actual request processing. This architecture scales to any desired processing load and user population size. Adding an additional stage does not affect the audio delay, since the media path (usually directly between the SIP phones) is independent of the signaling path.

4.3 Network address translation

A network address translator (NAT) device can expose a unique public address as the server address and distribute the incoming traffic to one of the several internal private hosts running the SIP servers [27]. Soon the NAT itself becomes the bottleneck making the architecture inefficient. Moreover, the transaction stateful nature of SIP servers require that subsequent re-transmissions should be handled by the same internal server. So the NAT needs to maintain the transaction state for the duration of the transaction, further limiting scalability.

4.4 Servers with the same IP address

In this approach, all the redundant servers in the same broadcast network use the same IP address. The router on the subnet is configured to forward the incoming packets to one of these servers' MAC address.

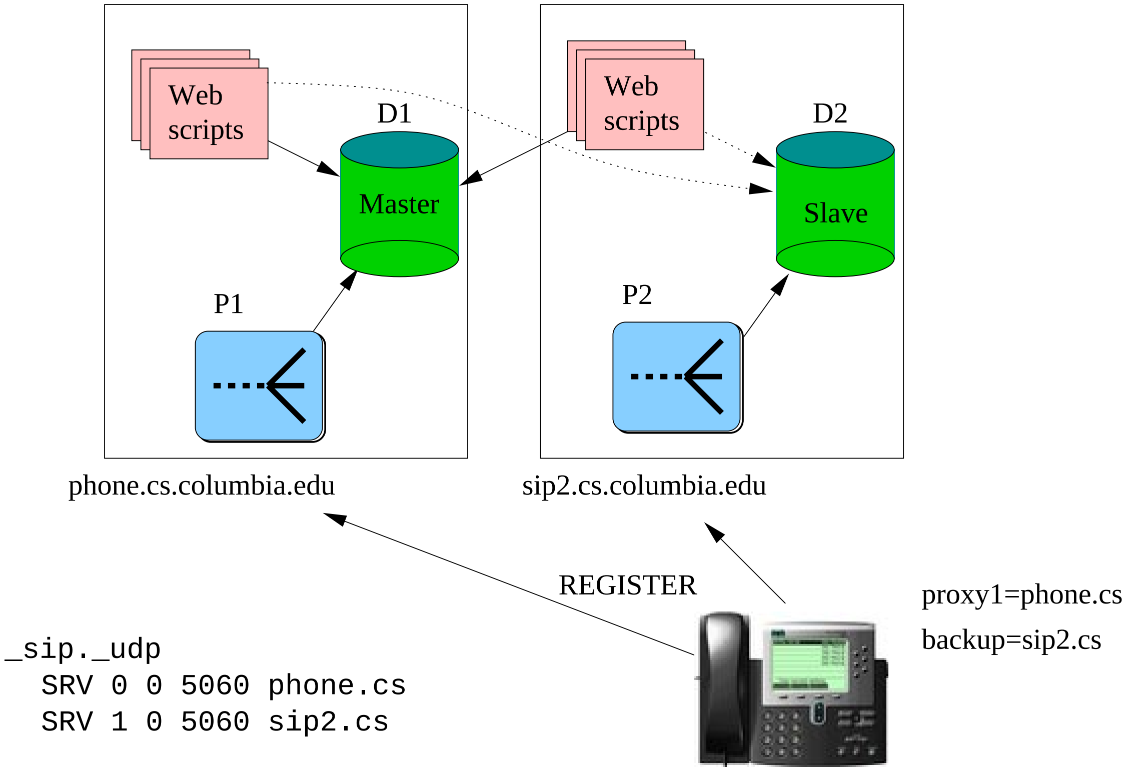
The router can use various algorithms such as “round robin” or “response time from server” to determine the less loaded server to choose.

To avoid storing SIP transaction states in the subnet router, this method is only recommended for stateless SIP proxies that use only UDP transport and treat each request as independent without maintaining any transaction state.

In the absence of DNS SRV, we can use this method for the first stage in Fig. 11. This is less efficient since the network bandwidth of this subnet may limit the number of servers in the cluster. Moreover, this method does not work if the network is dead.

5 Implementation

We have used some of the above techniques in our Columbia InterNet Extensible Multimedia Architecture (CINEMA). The architecture [28, 29] consists of our SIP server, sipd and a MySQL database for user profile and system configuration. Other components such as the PSTN gateway and media servers are outside the scope of this paper. The configuration and management are done via a web interface that accesses various CGI (Common Gateway Interface) scripts written in Tcl on the web server. All the servers may run on a single machine for an enterprise setup.



```
_sip._udp
  SRV 0 0 5060 phone.cs
  SRV 1 0 5060 sip2.cs
```

Figure 12: Failover in CINEMA

For failover, we use two sets of identical servers on two different machines as shown in Fig. 12. The database and SIP server share the same host. The databases are replicated using MySQL 4.0 replication [21] such that both D_1 and D_2 are master and slave of each other. MySQL propagates the binary log of the SQL commands of master to the slave, and the slave runs these commands again to do the replication. Appendix A contains the details of two-way replication in MySQL.

MySQL 4.0 does not support any locking protocol between the master and the slave to guarantee the atomicity of the distributed updates. However, the updates from the SIP server are additive, i.e., each registration from each device is one database record, so having two devices for the same user register with two database replicas does not interfere with the other registration. For example, if *bob@home.com* registers *bob@location1.com* with D_1 and *bob@location2.com* with D_2, both, D_1 and D_2, will propagate the updates to each other such that both will have both of Bob's locations. There is a slight window of vulnerability when one contact is added from D_1 and the same contact is removed in D_2, then after the propagation of updates the two databases will be inconsistent with different contacts for the user. It turns out that this does not occur for the simple failover as we describe next. We can safely use the two-way replication as long as updates are done by only the SIP server.

For a simple failover case, the primary server P_1 is preferred over the secondary server P_2. So all the REGISTER requests go to P_1 and are updated in D_1. The replication happens from D_1 to D_2, not the other way. Only in the case of failure of P_1, will the update happen to D_2 through P_2. But D_1 will not be updated by the server in this case. By making sure that database becomes consistent before the failed server is brought up, we can avoid the database inconsistency problem mentioned above.

Web scripts are used to manage user profile and system configuration. To maintain database consistency, the web scripts should not be allowed to modify D_2 if D_1 is up. To facilitate this we modified the MySQL-Tcl client interface to accept a list of connection attributes. For example, if D_1 and D_2 are listed then the scripts tries to connect to D_1 first, and if that fails then tries D_2 as shown in Fig. 12. For our web scripts, the short lived TCP connection to MySQL is active as long as the CGI script is running. So the failover at the connection setup is sufficient. In future, for long lived connection it should be modified to provide failover even when the connection breaks.

We use an in-memory cache of user records inside the SIP server to improve its performance [28, 26]. This causes more latency in updating the user registration from P_1 to P_2. If the failover happens before the update is propogated to the P_2, then it may have an old and expired record. However, in practice the phones refresh registrations much before the expiry and the problem is not visible. For example, suppose the record expires every two hours and the refresh happens every 50 minutes. Suppose P_1 receives the registration update from a phone and fails before propagating the update to D_1. At this point, the record in D_2 has 70 minutes to expire so P_2 can still handle the calls to this phone. The next refresh happens in 50 minutes, before expiration of the record in D_2. If a new phone is setup (first time registration) just before failure of P_1, it will be unavailable until the next refresh. Secondly, with the Cisco phone [23] that has the primary and backup proxy address options (Section 3.1), the phone sends registers with both P_1 and P_2. Both D_1 and D_2 propagate the same contact location change to each other. However, since the contact record is keyed on the user identifier and contact location, the second write just overrides the first write without any other side effect. Alternatively, the server can be modified to perform the immediate synchronization between the in-memory cache and external database if the current load is less.

The two-way replication can be extended to more servers by using circular replication such as D_1-D_2-D_3-D_1 using the MySQL master/slave configuration [21]. To provide failover of individual servers (e.g., D_1 fails but not P_1), the SIP server P_1 should switch to D_2, if D_1 is not available.

6 Conclusions and Future Work

We have shown how to apply some of the existing failover and load sharing techniques to SIP servers. The DNS SRV is the most preferred way to do redundancy since it does not require network co-location of the servers. For example, one can place SIP servers on different networks. With IP address takeover and NATs, that is rather difficult. This is less important for enterprise environments, but interesting for voice service providers such as Vonage. DNS itself is replicated, so a single name server outage does not affect operation. Combining DNS with the identifier-based load sharing can scale to large user base.

We have also described the failover implementation in our test bed. We will continue implementing and experimenting with load sharing techniques for our SIP server. Other call stateful services such as voicemail, conferencing and PSTN interworking need more work to do failover and load sharing in the middle of the call without breaking the session.

Detection and recovery of wide area path outages [30] is complementary to the individual server failover. Instead of statically configuring the redundant servers, it will be useful if the servers can automatically discover and configure other available servers on the Internet, e.g., to handle temporary overload [31]. This gives rise to the service model where the provider can sell its SIP services dynamically by becoming part of another customer SIP network. A peer-to-peer approach for the SIP service also seems promising for future work.

7 Acknowledgment

Jonathan Lennox is the primary architect of our SIP server, sipd. Sankaran Narayanan implemented efficient database interaction in sipd. The work is supported by a grant from SIPquest, Inc.

A Two-way replication in MySQL

This section describes the steps needed to setup two-way replication in MySQL. Please refer to Fig. 12 for the following steps:

1. Edit `/etc/my.cnf` to set the unique server-id for D_1 and enable binary logging:

   ```
   [mysqld]
   server-id = 1
   log-bin
   ```

   Restart mysqld.

2. Create a replication user on D_1 with appropriate privileges for D_2's IP address.

   ```
   GRANT SELECT,PROCESS,FILE,SUPER,RELOAD,
    REPLICATION CLIENT,REPLICATION SERVER ON
    *.* TO replication@"sip2.cs.columbia.edu"
    IDENTIFIED BY "somepassword";
   ```

3. Then copy the `data/sip` directory to `snapshot.tar` file

4. Get the master status (file name and position) of the binary log.

   ```
   SHOW MASTER STATUS;
   ```

   Suppose it shows file as `phone-bin.001` and position as 73.

5. Shutdown mysqld and start it again. Make sure no updates are happening in D_1 or D_2 while setting up the replication. Make sure D_2 is dead.

6. Create a replication user on D_2 similar to D_1, but with permissions for IP address of D_1, so that D_1 can access D_2.

7. Copy and uncompress the `snapshot.tar` from D_1 to the D_2 `data` directory. This will ensure the content the sip database of D_2 is same as that of D_1 when for the given master status of D_1. Some fields in the sip database, such as `cinema::sipdhost` should store the actual host name of the machine running the server. These fields can be populated with `sip2` for D_2 using the SQL_SLAVE_SKIP_COUNTER global in MySQL.

8. Edit `/etc/my.cnf` of D_2, similar to D_1, except that the server-id is 2 for D_2. (server-id values are not important as long as they are unique for a given replication setup.)

9. Start mysqld on D_2.

10. Setup D_2 as slave of D_1, by running following command on D_2:

    ```
    CHANGE MASTER TO
     MASTER_HOST='sip2.cs.columbia.edu',
     MASTER_USER='replication',
     MASTER_PASSWORD='somepassword',
     MASTER_LOG_FILE='phone-bin.001',
     MASTER_LOG_POS=73;
    START SLAVE;
    ```

    The log file and position are same as that recorded from D_1. At this point we have D_1 to D_2 replication complete.

11. Now record the master status on D_2. Suppose it shows file as `sip2-bin.002` and position as 79.

12. Copy all the `*bin.*` (binary logs) from D_2's `data` directory to D_1's `data` directory.

13. Not set D_1 as slave of D_2 by running following command on D_1:

```
CHANGE MASTER TO
 MASTER_HOST='phone.cs.columbia.edu',
 MASTER_USER='replication',
 MASTER_PASSWORD='somepassword',
 MASTER_LOG_FILE='sip2-bin.002',
 MASTER_LOG_POS=79;
START SLAVE;
```

At this point D_2 to D_1 replication is also complete. To allow access from other hosts, it may be required to remove the no-authentication line from the MySQL permissions table.

```
USE mysql;
DELETE FROM user WHERE User='';
FLUSH PRIVILEGES;
```

To bring up D_1 after a failover, tables on D_2 should be read-locked to prevent database inconsistency. In case failover messes up for some reason, the whole procedure can be repeated to setup the failover from scratch without losing the data in D_1.

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: session initiation protocol," RFC 3261, Internet Engineering Task Force, June 2002.

[2] H. Schulzrinne and J. Rosenberg, "Internet telephony: Architecture and protocols – an IETF perspective," *Computer Networks and ISDN Systems*, vol. 31, pp. 237–255, Feb. 1999.

[3] H. Schulzrinne and J. Rosenberg, "The session initiation protocol: Internet-centric signaling," *IEEE Communications Magazine*, vol. 38, Oct. 2000.

[4] H. Bryhni, E. Klovning, and Øivind Kure, "A comparison of load balancing techniques for scalable web servers," *IEEE Network*, vol. 14, July 2000.

[5] K. Suryanarayanan and K. J. Christensen, "Performance evaluation of new methods of automatic redirection for load balancing of apache servers distributed in the Internet," in *IEEE Conference on Local Computer Networks*, (Tampa, Florida, USA), Nov. 2000.

[6] O. Damani, P. Chung, Y. Huang, C. Kintala, and Y. Wang, "ONE-IP: techniques for hosting a service on a cluster of machines," *Computer Networks*, vol. 29, pp. 1019–1027, Sept. 1997.

[7] D. Oppenheimer, A. Ganapathi, and D. Patterson, "Why do internet services fail, and what can be done about it?," in *4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, (Seattle, WA), Mar. 2003.

[8] A. C. Snoeren, D. G. Andersen, and H. Balakrishnan, "Fine-grained failover using connection migration," in *USENIX Symposium on Internet Technologies and Systems*, (San Francisco), Mar. 2001.

[9] High-Availability Linux Project, http://www.linux-ha.org/.

[10] Cisco Systems, Failover configuration for LocalDirector, http://www.cisco.com/warp/public/cc/pd/cxsr/400/tech/loco

[11] G. Hunt, G. Goldszmidt, R. P. King, and R. Mukherjee, "Network dispatcher: a connection router for scalable Internet services," *Computer Networks*, vol. 30, pp. 347–357, Apr. 1998.

[12] C.-L. Yang and M.-Y. Luo, "Efficient support for content-based routing in web server clusters," in *2nd USENIX Symposium on Internet Technologies and Systems*, (Boulder, Colorado, USA), Oct 1999.

[13] Akamai Technologies, Inc. http://www.akamai.com.

[14] A. Gulbrandsen, P. Vixie, and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)," RFC 2782, Internet Engineering Task Force, Feb. 2000.

[15] M. Mealling and R. W. Daniel, "The naming authority pointer (NAPTR) DNS resource record," RFC 2915, Internet Engineering Task Force, Sept. 2000.

[16] P. Conrad, "Services provided by reliable server pooling," Internet Draft draft-ietf-rserpool-service-00, Internet Engineering Task Force, Jan. 2004. Work in progress.

[17] M. Tuexen, Q. Xie, R. J. Stewart, M. Shore, L. Ong, J. Loughney, and M. Stillman, "Requirements for reliable server pooling," RFC 3237, Internet Engineering Task Force, Jan. 2002.

[18] A. Srinivasan, K. G. Ramakrishnan, K. Kumaran, M. Aravamudan, and S. Naqvi, "Optimal design of signaling networks for Internet telephony," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Tel Aviv, Israel), Mar. 2000.

[19] R. Sparks, "The session initiation protocol (SIP) refer method," RFC 3515, Internet Engineering Task Force, Apr. 2003.

[20] R. Sparks, "SIP load management," Internet Draft draft-sparks-sipping-load-00, Internet Engineering Task Force, Oct. 2003. Work in progress.

[21] MySQL, Open Source SQL server, http://www.mysql.com.

[22] Emic Cluster for MySQL, http://www.emicnetworks.com.

[23] Cisco IP phone 7960, Release 2.1, http://www.cisco.com.

[24] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," in *ACM SIGCOMM Internet Measurement Workshop*, (San Francisco, California), Nov. 2001.

[25] M. Tuexen, Q. Xie, and M. B. et. al., "Architecture for reliable server pooling," Internet Draft draft-ietf-rserpool-arch-07, Internet Engineering Task Force, Oct. 2003. Work in progress.

[26] J. Lennox, "Services for internet telephony," PhD. thesis, Department of Computer Science, Columbia University, New York, New York, Jan. 2004. http://www.cs.columbia.edu/~lennox/thesis.pdf.

[27] P. Srisuresh and D. Gan, "Load sharing using IP network address translation (LSNAT)," RFC 2391, Internet Engineering Task Force, Aug. 1998.

[28] K. Singh, W. Jiang, J. Lennox, S. Narayanan, and H. Schulzrinne, "CINEMA: columbia internet extensible multimedia architecture," technical report CUCS-011-02, Department of Computer Science, Columbia University, New York, New York, May 2002.

[29] W. Jiang, J. Lennox, S. Narayanan, H. Schulzrinne, K. Singh, and X. Wu, "Integrating Internet telephony services," *IEEE Internet Computing*, vol. 6, pp. 64–72, May 2002.

[30] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks," in *18th ACM SOSP*, (Banff, Canada), Oct. 2001.

[31] W. Zhao and H. Schulzrinne, "DotSlash: A scalable and efficient rescue system for handling web hotspots," technical report CUCS-007-04, Department of Computer Science, Columbia University, New York, New York, Feb. 2004.