# Using DUNDi with a Cluster of Asterisk Servers!

# General Description and Scope

DUNDi™ is a peer-to-peer system for locating Internet gateways to telephony services. Unlike traditional centralized services (such as the remarkably simple and concise ENUM standard), DUNDi is fully-distributed with no centralized authority whatsoever.

Clustering Asterisk gives us the ability to distribute the load of PBX/Soft Switch functions across a lot of Asterisk servers in a common infrastructure. This allows the cluster to look and feel like one huge soft switch. This can also give the soft switch function built in failover protection and high availability for the SIP Agents.

Put DUNDi and an Asterisk Cluster together and you have the foundation of a core VoIP switching network that is scalable without the need for statically addressing SIP peers to a specific registration server then scripting static dial-plan routes to those registration servers. Dynamic peer location and immediate contact is the goal.

This serves a couple of fundamental needs when designing and building an IP Telephony network.

1. We design in an environment that is self healing across the core soft switch.
2. We design architecture with scalability and growth in mind, organic growth so incremental equipment costs are low.

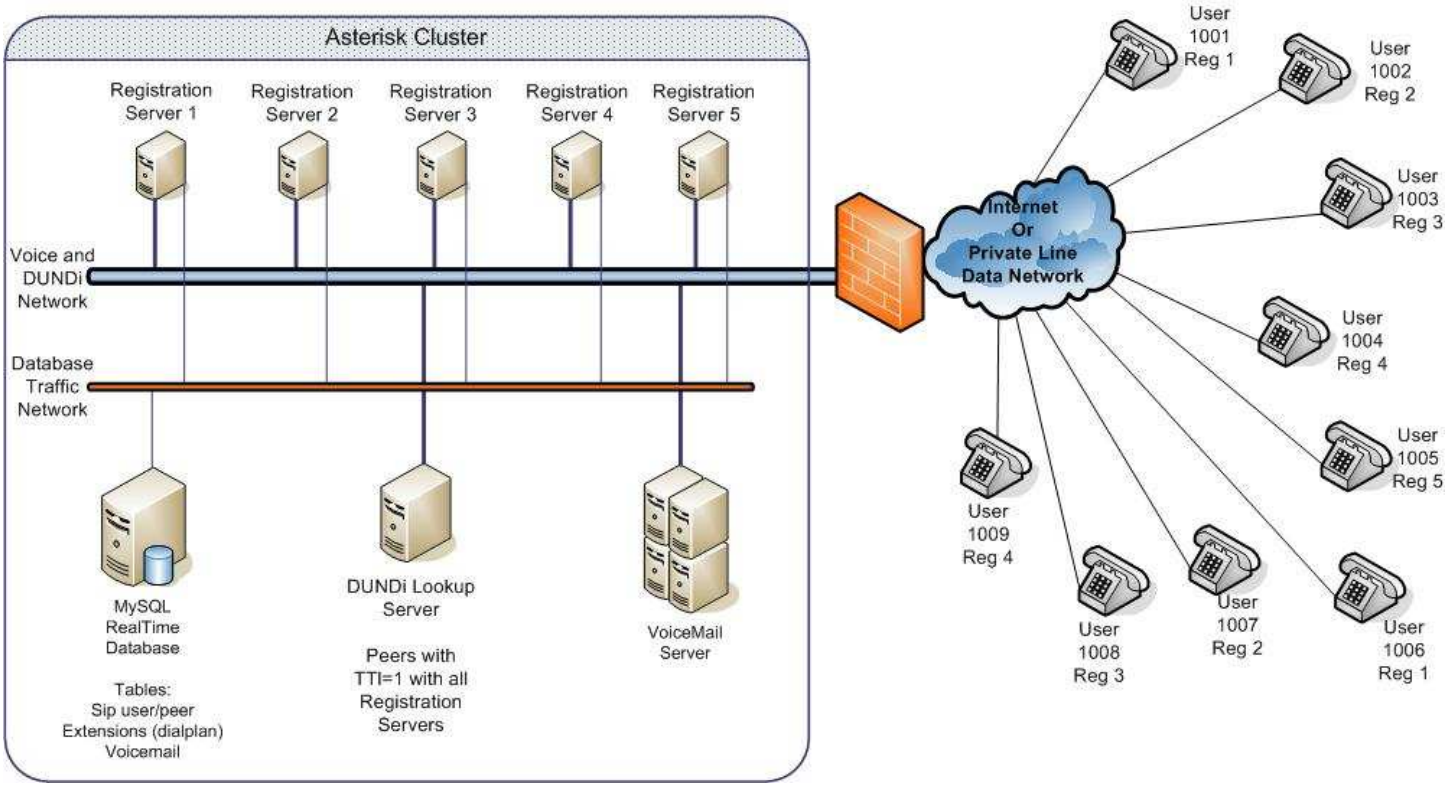

**Figure 1 Asterisk Cluster**

## Functional Requirements

We have to address several individual needs to achieve dynamic peer location and a method of contacting them directly or indirectly during a PBX failure.

To accomplish dynamic peer location across the cluster, we will first implement a method whereby an individual PBX is aware when a specific SIP Agent peer has an active registration on itself. When a SIP Agent registers with a PBX, it is the duty of the local PBX to let the other PBX's know where this SIP agent is and how it can be contacted.

We also need the ability for the cluster to share a common pool of SIP Agent registration information. This will be accomplished through the Asterisk RealTime Architecture using sip users/peer information pulled from a MySQL Database. This database will be a stand-alone server for the purpose of this paper but could also be a cluster of servers in a high availability or load balanced arrangement.

Having a common database where all servers pull the same information eliminates the need to provision each SIP registration server in the cluster.

### For Dynamic Peer Awareness We Use "regcontext" in sip.conf

regcontext=<context>

If regcontext is specified, Asterisk will dynamically create and destroy a NoOp priority 1 extension for a given peer who registers with the server. If the context is not specified in extensions.conf, then it will be dynamically created when a sip agent registers.

Example:

In the [general] section of sip.conf:

```
[general]
regcontext=sipregistration
```

Once the phones, in this example 1001 and 1006 register with REGPBX1, a context of [sipregistration] appears and the "show dialplan" command at the asterisk CLI> will produce:

```
REGPBX1*CLI> show dialplan
[ Context 'sipregistration' created by 'SIP' ]
  '1001' =>       1. Noop(1001)                   [SIP]
  '1006' =>       1. Noop(1006)                   [SIP]
```

This gives this PBX a dedicated context that we can map DUNDi lookup request to. When a DUNDi lookup requests location information for extension 1001, this PBX will reply, “Yes, the extension is active here and this is the contact address”. When a DUNDi lookup requests location information for extension 1002, this PBX will reply, “No, that extension is not active here”.

We do not insert a [sipregistration] context in the extensions.conf file because there is no need. The PBX dynamically creates the context in the dialplan as soon as 1 sip agent registers.

As we insert regcontext=sipregistration in the sip.conf file in each registration server, we begin to see how the cluster, with DUNDi lookups, intuitively knows where SIP Agents are registered.

**DUNDi Lookup Server**

With a server in the cluster dedicated to processing DUNDi lookup requests, we eliminate 2 headaches associated with scaling and growth of the core soft switch function.

1. When adding a new registration or PSTN gateway server to the cluster, we just need to insert the new peer information to the DUNDi lookup server instead of adding the new peers to all the registration servers and PSTN gateway servers. With a handful of servers, not much of a big deal, but with 50 or more servers, now you have a maintenance event that could span several hours or over the course of several evenings.
2. As we bring on more trunk access between other markets, say you are turning up a Dedicated T1 to Lake Charles, LA; you just have to make a route/translation in the DUNDi lookup server instead of across all of the registration servers.

**DUNDi Configuration:**

First we have to setup a channel for internal DUNDi to talk across and also a channel to pass calls between the PBX’s

In this example we will use IAX2. Now there are many ways to set this up and quite a few parameters we could specify for security between all the registration servers and DUNDi lookup server, each having its own context, peer/users relationships and security keys or passwords. Because this is a cluster, protected from the outside world and only used to route calls and DUNDi request internally, just use a simple context in iax.conf that is common to all the servers.

Example in iax.conf:

```
[priv]
type=friend
```

dbsecret=dundi/secret
context=incomingdundi

Add this context to each server. That's it, done with IAX2 setup. Now all the PBX's have a channel to exchange DUNDi lookup request/responses and also a channel to direct calls across within the cluster.

dundi.conf

In the [mappings] section, this is where we specify what [context] in extensions.conf we want to allow DUNDi request access to. This is how the cluster sees any available SIP Agents in the [sipregistration] context on this PBX.

[mappings]
priv => sipregistration,0,IAX2,priv:${SECRET}@10.10.10.121/${NUMBER},nopartial

When a DUNDi lookup request comes to this regpbx1, Asterisk will report back whatever SIP Agents appear in the context [sipregistration]. If the request is for an extension not listed in context [sipregistration], this PBX will reply to the DUNDi lookup with "not found"

**How to configure DUNDi to work in an Asterisk Cluster**

We have 5 registration servers we call regpbx1, regpbx2, regpbx3, regpbx4 & regpbx5. SIP Agent information in pulled dynamically from a MySQL database using the Asterisk RealTime engine. When a sip agent attempts to register to a registration server, Asterisk looks for the agent's information in the MySQL database, if present; the agent is allowed to register.

We have 1 Asterisk PBX setup to process DUNDi lookup request and pass calls from the PSTN Gateways to the SIP registration cluster, we call this server dunpbx1. This PBX is DUNDi peering with all 5 registration servers. All 5 registration servers only peer with this DUNDi lookup server, dunpbx1.

Each registration server is configured as such in dundi.conf:

Regpbx1 dundi.conf

[general]
department=dept
organization=company
locality=city
stateprov=state
country=US
email=engineer@company.com
phone=contact phone number

;bindaddr=0.0.0.0
;port=4520
entityid=02:03:AF:B7:FF:37 (this defaults to the first NIC MAC address, but it's a good idea to specify it)
**********
cachetime=5
**(we reduce the cachetime to 5 seconds, this allows us to perform a DUNDi lookup request every time we dial. If a registration server should fail, and this value is set to the default, 1 hour, it will take up to an hour before the extensions that were registered to this registration server are known elsewhere in the cluster even if they re-register to another registration server.)**
**********
ttl=2
**(we don't want to lookup past any local registration server so limit the hops to 2. [see diagram DUNDi Hop Count])**
**********
autokill=yes
;secretpath=dundi
;storehistory=yes
**********
[mappings]
priv => sipregistration,0,IAX2,priv:${SECRET}@10.10.10.121/${NUMBER},nopartial
**(when a [priv] dundi lookup request comes in, this PBX will advertise whatever extensions are present in the [sipregistration] context in the dialplan and if the exten exists here, the PBX will send back the contact info [IAX2/priv:${SECRET}@the ip address of this server/the extension number requested)**
**********

**This is the peer section, who this PBX peers with:**

[00:14:22:23:26:2E] **;(this is the MAC address of the DUNDi server dunpbx1)**
model = symmetric
host = 10.10.10.120 **;(this is the IP address of the DUNDi lookup server dunpbx1)**
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary

**Note:** I use the same inkey/outkey for regpbx1, 2, 3, 4, 5 and dunpbx1. Since all DUNDi lookups stay within this cluster, provisioning 1 set of keys and sharing across the cluster is much easier than keeping track of a set for each server. This also helps when adding new servers to the cluster.

I copy the above dundi.conf config file to the other registration servers and modify the server specific entries:

entityid=02:03:AF:B7:FF:37 **(this is server specific, per PBX)**

[mappings]
priv => sipregistration,0,IAX2,priv:${SECRET}@10.10.10.122/${NUMBER},nopartial
**(ip address is server specific, per PBX)**

The DUNDi lookup server, dunpbx1, has a peering section with all 5 registration servers as such:

[01:31:AF:E7:FF:37] **;(regpbx1 entityid (MAC address))**
model = symmetric
host = 10.10.10.121 **(regpbx1 IP Address)**
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary

[03:11:AA:02:B3:1D] **;(regpbx2 entityid (MAC address))**
model = symmetric
host = 10.10.10.122 **;(regpbx2 IP Address)**
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary

[05:31:AD:18:53:0B] **;(regpbx3 entityid (MAC address))**
model = symmetric
host = 10.10.10.123 **;(regpbx3 IP Address)**
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary

[01:19:4A:52:B3:1D] **;(regpbx4 entityid (MAC address))**
model = symmetric
host = 10.10.10.124 **;(regpbx4 IP Address)**
inkey = dundi

outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary

[02:7A:AE:43:52:0C] **;(regpbx5 entityid (MAC address))**
model = symmetric
host = 10.10.10.125 **;(regpbx5 IP Address)**
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary

Use the "dundi show peers" command at the asterisk cli to see connection state with the peers identified in dundi.conf:

```
dunpbx1*CLI> dundi show peers
EID               Host            Model      AvgTime  Status
01:31:af:e7:ff:37     10.10.10.121    (S) Symmetric  Unavail  OK (1 ms)
03:11:aa:02:b3:1d   10.10.10.122    (S) Symmetric  Unavail  OK (1 ms)
05:31:ad:18:53:0b   10.10.10.123    (S) Symmetric  Unavail  OK (1 ms)
01:19:4a:52:b3:1d   10.10.10.124    (S) Symmetric  Unavail  OK (1 ms)
02:7a:ae:43:52:0c   10.10.10.125    (S) Symmetric  Unavail  OK (1 ms)
5 dundi peers [5 online, 0 offline, 0 unmonitored]
ast1*CLI>
```

With these 5 PBX peers listed as such, when a DUNDi lookup request comes from regpbx1, this PBX will request extension location information from regpbx2, 3, 4 & 5. If a DUNDi lookup request comes from regpbx3, this PBX will request extension location information from regpbx1, 2, 4 & 5.

The TTL value in this arrangement is important to keep low. The DUNDi lookup server is restricted to only 1 hop, TTL=1, so requests do not propagate past the registration servers. The registration servers are restricted to 2 hops, TTL=2, for the same reason, but if they were only set to 1, the DUNDi lookup request would not get past the DUNDi lookup server. See figure 2, DUNDi Hop Diagram.

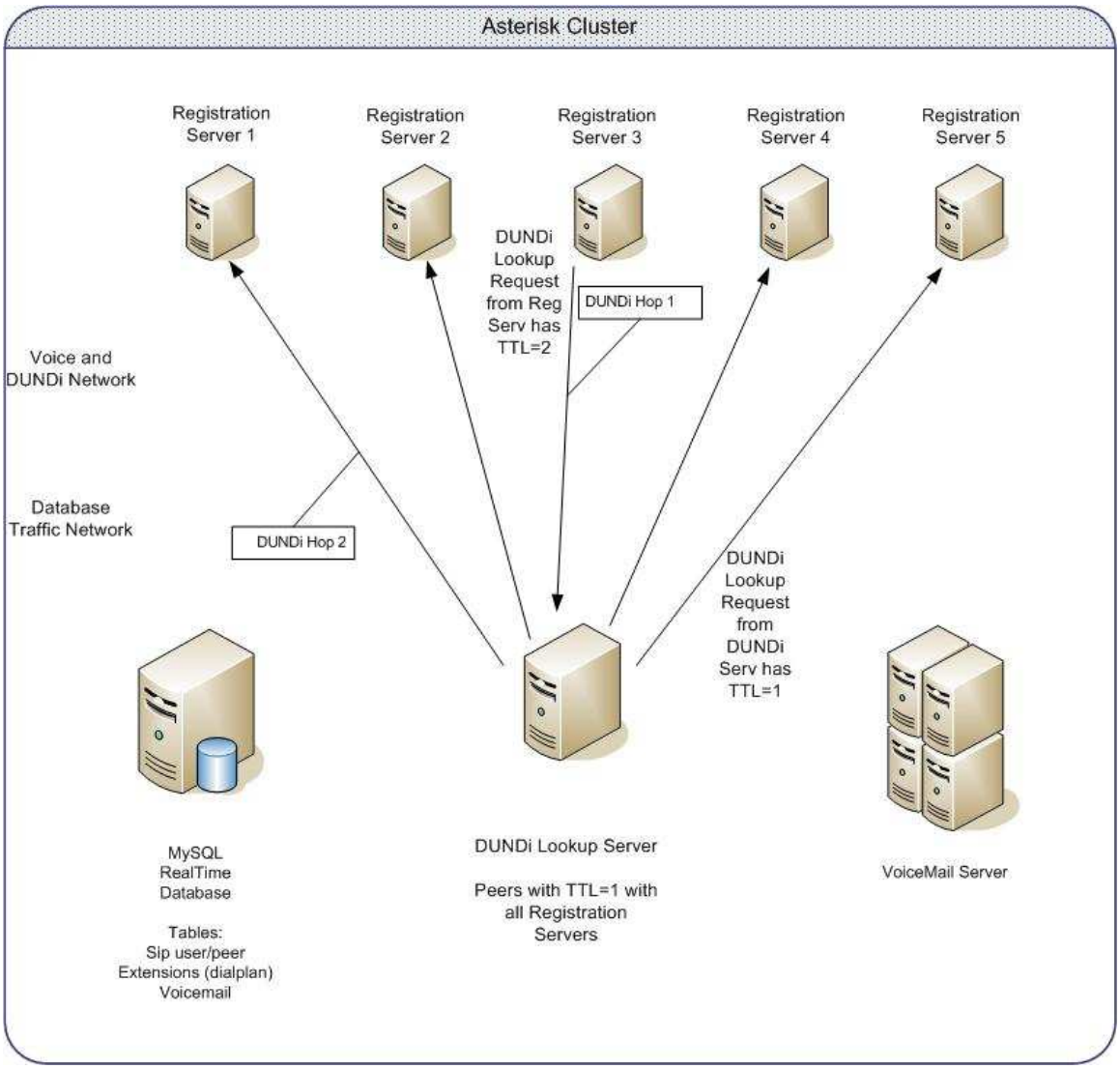


**Figure 2**
**DUNDi Hop Diagram**

# Dial Plan Routing, Efficiency First

So now that we have dynamic SIP Agent location awareness within the cluster, just how do we route calls within the cluster, what does the dial plan look like?

Let us look at a few examples:

**Example 1:** SIP Agent who calls another SIP Agent who happens to be registered to the same registration server.

In this example, extension 1001 calls 1006. We are going to look internally for the extension and if present, we will follow standard PBX function, and connect the call.

In extension.conf, here is the dialplan logic used to perform an internal lookup:

```
[lookuplocal]
exten => _XXXX,1,ChanIsAvail(SIP/${EXTEN}&IAX2/${EXTEN}|sj)
exten => _XXXX,2,Goto(incoming|${EXTEN}|2)
exten => _XXXX,3,Hangup
exten => _XXXX,102,Goto(lookupdundi|${EXTEN}|1)
exten => _XXXX,103,Hangup
```

We check to see is the extension we dialed, 1006, is available on the local PBX using the "ChanIsAvail" command. Then depending on the channel availability, we send the call to the [incoming] context to look for extension 1006 priority 2 or we send the call to the [lookupdundi] context priority 1.

Here is a snippet from the Asterisk CLI>

```
regpbx1*CLI>
   -- Executing Goto("SIP/1001-6c54", "lookuplocal|1006|1")
   -- Goto (lookuplocal,1006,1)
   -- Executing ChanIsAvail("SIP/1001-6c54", "SIP/1006&IAX2/1006|sj") in new stack
   -- Executing Goto("SIP/1001-6c54", "incoming|1006|2") in new stack
   -- Goto (incoming,1006,2)
   -- Executing Answer("SIP/1001-6c54", "")
   -- Executing Dial("SIP/1001-6c54", "Sip/1006|20|tr")
   -- Called 1006
   -- SIP/1006-88c4 is ringing
   -- SIP/1006-88c4 is ringing
   -- SIP/1006-88c4 answered SIP/1001-6c54
  == Spawn extension (incoming, 1006, 3) exited non-zero on 'SIP/1001-6c54'
regpbx1*CLI>
```

**Example 2:** SIP Agent who calls another SIP Agent who is registered on another registration server.

In this example, extension 1001 calls 1002. We are going to look internally for the extension and if present, we will follow standard PBX function, and connect the call. But if the extension is not present we will perform a DUNDi lookup within the cluster and connect the call to the appropriate server that has extension 1002 registered.

In extension.conf, here is the dialplan logic used to perform a DUNDi lookup:

```
[lookupdundi]
exten => _X,1,Goto(${ARG1},1)
switch => DUNDi/priv
```

exten => i,1,Goto(lookupmysql,${INVALID_EXTEN},1)
The ${ARG1} variable is the extension from the previous [lookuplocal] context, in this example it is 1002. We are using the switch statement to tell DUNDi to look for extension 1002 in the "priv" mapping on the receiving DUNDi peers. Recall that all the registration servers have the [sipregistration] context mapped to "priv" in their dundi.conf files. So when this request for extension 1002 goes out to the local peer, which is dunpbx1, the request hits the DUNDi lookup server and forwards the request to the remaining registration servers in the cluster.

Registration server 2 responds to the DUNDi lookup server with connection information, and this is passed back to registration sever 1 to connect the call.

If the DUNDi lookup returns "extension not found", this will be considered an invalid extension within this [lookupdundi] context and the call will be passed to the invalid extension handler, then we send the call to context [lookupmysql] extension 1002 priority1.

Here is a snippet from the Asterisk CLI>

```
regpbx1*CLI>
   -- Executing Goto("SIP/1001-2c5d", "lookuplocal|1002|1")
   -- Goto (lookuplocal,1002,1)
   -- Executing ChanIsAvail("SIP/1001-2c5d", "SIP/1002&IAX2/1002|sj") in new stack
   -- Executing Goto("SIP/1001-2c5d", "lookupdundi|1002|1") in new stack
   -- Goto (lookupdundi,1002,1)
   -- Called priv:wqOGASylTGtBljFtvGjuCw@10.10.10.122/1002
   -- Call accepted by 10.10.10.122 (format ulaw)
   -- Format for call is ulaw
   -- IAX2/10.10.10.122:4569-1 answered SIP/1001-2c5d
   -- Hungup 'IAX2/10.10.10.122:4569-1'
  == Spawn extension (lookupdundi, 1002, 1) exited non-zero on 'SIP/1239-2c5d'
regpbx1*CLI>
```

**Example 3:** In this example, extension 1001 calls 1002. But let us say registration server 2 just failed. We are going to look internally for the extension and if present, we will follow standard PBX function, and connect the call. But if the extension is not present we will perform a DUNDi lookup within the cluster and connect the call to the appropriate server that has extension 1002 registered. If DUNDi returns "extension not found" then we will ask the MySQL server if it knows how to contact the SIP Agent directly. This is possible because the Asterisk RealTime Architecture pulls SIP Agent registration info from the database and also, when a SIP Agent registers to a PBX, Asterisk then writes the "fullcontact" information back to the database for that SIP Agent. We can use this information to call the SIP Agent directly if it is available.

In extension.conf, here is the dialplan logic used to perform a MySQL lookup:

```
[lookupmysql]
include => invalid
exten => _X.,1,MYSQL(Connect connid 10.10.10.110 asteriskdb password db)
exten => _X.,2,MYSQL(Query resultid ${connid} SELECT\ fullcontact\ from\ sip\
where\ name=${EXTEN})
exten => _X.,3,MYSQL(Fetch fetchid ${resultid} var1)
exten => _X.,4,MYSQL(Clear ${resultid})
exten => _X.,5,MYSQL(Disconnect ${connid})
exten => _X.,6,GotoIf($["${var1}" = ""]?invalid,i,1:${EXTEN},8)
exten => _X.,8,Set(direct=${var1:4})
exten => _X.,9,Dial(SIP/${direct},15,r)
exten => _X.,10,Goto(sendtovm,${EXTEN},1)
exten => _X.,11,Hangup
```

Here we first must connect to the MySQL database with the correct authentication information. Then we query the table “sip” for the table field “fullcontact” where the name is 1002. We fetch this info, then clear the “resulted” and disconnect from the MySQL server.

(This is very important, you do not want to leave sql connections open, they will not close until you disconnect or restart, not disconnecting would eventually reach the MySQL connection limitation and not allow any more lookups and possible crash.)

We use the “GotoIf” command, if we pulled data from the “fullcontact” field and put that data into a variable ${direct} but first we strip 4 characters from the front of the data because it is appended with “sip:”, then we call the SIP Agent directly. If there was no data in the “fullcontact” field then the SIP Agent never registered with the cluster, so we send the call to the [invalid] context priority 1.

Since we are contacting the SIP Agent directly, we do not have any dial-plan logic associated with the extension we are calling, so we only ring the device for 15 seconds, then send the call to [sendtovm] context priority 1.

Here is a snippet from the Asterisk CLI>

```
regpbx1*CLI>
   -- Executing Goto("SIP/1001-3a46", "lookuplocal|1002|1")
   -- Goto (lookuplocal,1002,1)
   -- Executing ChanIsAvail("SIP/1001-3a46", "SIP/1002&IAX2/1002|sj") in new stack
   -- Executing Goto("SIP/1001-3a46", "lookupdundi|1002|1") in new stack
   -- Goto (lookupdundi,1002,1)
   -- Sent into invalid extension '1002' in context 'lookupdundi' on SIP/1001-3a46
   -- Executing Goto("SIP/1001-3a46", "lookupmysql|1002|1") in new stack
   -- Goto (lookupmysql,1002,1)
```

```
   -- Executing MYSQL("SIP/1001-3a46", "Connect connid 10.10.10.110 asteriskdb
password db") in new stack
   -- Executing MYSQL("SIP/1001-3a46", "Query resultid 1 SELECT fullcontact from
sip where name=1002") in new stack
   -- Executing MYSQL("SIP/1001-3a46", "Fetch fetchid 2 var1") in new stack
Mar 29 18:29:47 WARNING[31837]: app_addon_sql_mysql.c:318 aMYSQL_fetch:
ast_MYSQL_fetch: numFields=1
   -- Executing MYSQL("SIP/1001-3a46", "Clear 2") in new stack
   -- Executing MYSQL("SIP/1001-3a46", "Disconnect 1") in new stack
   -- Executing GotoIf("SIP/1001-3a46", "0?invalid|i|1:1002|8") in new stack
   -- Goto (lookupmysql,1002,8)
   -- Executing Set("SIP/1001-3a46", "direct=1002@10.10.10.63:5060") in new stack
   -- Executing Dial("SIP/1001-3a46", "SIP/1002@10.10.10.63:5060|15|r") in new stack
   -- Called 1002@10.10.10.63:5060
   -- SIP/10.10.10.63:5060-32f2 is ringing
   -- SIP/10.10.10.63:5060-32f2 answered SIP/1001-3a46
   -- Attempting native bridge of SIP/1001-3a46 and SIP/10.10.10.63:5060-32f2
  == Spawn extension (lookupmysql, 1002, 9) exited non-zero on 'SIP/1001-3a46'
regpbx1*CLI>
```

# A Few Cautionary Notes

**“ChanIsAvail” Usage**

Why we don’t perform a “ChanIsAvail” command if we have to dial the SIP Agent directly. Some phones, like Cisco and Polycom, will accept incoming extension request and ring the extension as long as the phone still thinks it is registered to a PBX. Hence if the registration time is set to say, 5 minutes, if the registration server crashes and there is still 4 min 59 sec left on the registration time, then the phone will acknowledge a SIP request as available and accept a direct dial. During this condition, 5 minute window max, if you use “ChanIsAvail’ command, the phone will say, “yes, I am available, send the call.”

But if the server crashed and the phone registration times out, then the phone will realize it is not registered and take that extension out of available status, so the “ChanIsAvail” command will get a false negative and not try to dial the phone, but the phone is still available for direct dial even if it is not registered. So in the dial plan, always dial the phone and have a timeout to do something else after. Even if the phone is off-line totally, you still made an effort to contact, and then send the call to voicemail or wherever. At least the call was not falsely diverted when a call could have been established to the SIP Agent.

**SIP Agents Using Backup Registration Server**

It is possible to have a SIP Agent registered to more than 1 registration server. What happens when a DUNDi request goes out in this situation is the call will be completed to the first responder. So if extension 1002 is registered to registration server 1 and registration server 2 and extension 1003, registered on registration server 3 calls for extension 1002, then the call will complete to the first server that responds to the DUNDi request.

This may seem like an ideal situation but it contains pitfalls, mostly with PBX features like call parking. Ideally you want all extensions within the same office, persons who would be parking calls and asking others to pick up, to be registered to the same PBX, otherwise a situation could arise where extension 1001 registered on registration server 1 parks a call for extension 1002 which is on registration server 2.

To prevent this, use caution with backup registration servers. As with the Cisco phone, you can choose not to have a backup registration server but a proxy server of last resort. The phone will not register to the last resort proxy but will pass outbound calls to it. If the registration server fails, the phone can still complete outbound calls and using [lookupmysql] context as noted in example 3 above, inbound calls can still be completed to the SIP Agent.

**Asterisk RealTime and MySQL**

In diagram 1, I depicted 2 Ethernet segments, 1 for RealTme/MySQL exchanges and 1 for voice/DUNDi traffic. The reason is for scalability and to segment the traffic into more controllable environments. Most server class machines come with 2 or more NIC's so why not use them. MySQL and RealTime is a very chatty environment to operate voice in and several components in the dial plan rely on quick responses from the MySQL database. When voice traffic is prioritized across the switching network, the database traffic can get held in buffers and greatly extend response time. In the testing environment, 10ms lookups is fantastic. Add 10,000 customers to the cluster and you'll be praying for that kind of lookup time, but you'll never see it. You will get your best database response times when segmenting the RealTime/MySQL function and the other PBX and Voice functions.

**Load Balancing Across the Registration Server Cluster**

If the network is hosting a lot of non-contiguous users, unassociated, like residential customers, then a round robin or truer server-load aware method is deployed, then balancing can be arbitrary and fairly straight forward. When you have 10 users in Company A, 50 users in Company B, 25 Users in Company C and 5 users in Company D, then it makes more sense to put Company A, C and D on one server and Company B on another server to have some ability to offer complex PBX features. In the event server B crashes, the users will still have in and out calling until another server can be reconfigured with server B's IP address. For this reason and many others, several hot standby servers should be deployed strategically within the cluster.