DUNDi, So Easy A Caveman Could Do It!

JR Richardson
Engineering for the Masses
hubguru@gmail.com

General Description

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.

What?

Simply put, DUNDi is an Asterisk specific protocol setup between two or more PBX's whereby a PBX may request extension call route location information from one or more peering PBX's.

Here is a verbal example:

> PBX A: Hello PBX peer B. Do you have extension 201?
>
> PBX B: Hi PBX peer A. Yes I do have extension 201, and here is how you will contact this extension, IAX2/username:password@10.10.14.122/201

Please forgive me if this seems too simplistic, no offense intended. But this is really as simple and as complicated as things need to get with DUNDi. In its purest sense, all DUNDi does is look for extensions on another PBX.

There are things we have to do to setup this peer relationship between the PBX's and some dial plan manipulation to make it work effectively. In this paper, we will step through the basic concepts and configurations needed to get 2 or more Asterisk PBX's setup as DUNDi peers and look at the actual DUNDi messaging while processing DUNDi request.

PBX Test Setup

Assumptions, Asterisk servers are setup and functioning properly, SIP phones are registered to each PBX and normal call flow within each PBX is working fine. For DUNDi to work only 2 Asterisk servers are needed but we will go into examples with 3 servers.

PBX A, ver 1.4, 10.10.14.121, extension 101 through 103
PBX B, ver 1.2, 10.10.14.122, extensions 201 through 203
PBX C, ver 1.2, 10.10.14.123, extensions 301 through 303

DUNDi Configuration:

IAX.CONF

First we have to setup a channel for calls to pass between the PBX's. In this example we will use IAX2. A simple context in iax.conf is all that is needed for the servers to communicate with each other.

```
[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 direct calls across.

The [priv] context is a name, could be any name as long as it is common between the mapping in dundi.conf, more on this later.

The type=friend allows for a 2-way channel between servers.

The dbsecret=dundi/secret is the password used when location information is sent out to requesting servers. The DUNDi protocol generated a new secret password every hour and stores this password in the local Asterisk database.

The context=incomingdundi is the entry point in the dial plan where calls come into. So extensions.conf must have a [incomingdundi] context with a routable extension, like a Goto or include=internal where a new call can be processed properly. This context could be named anything you like or could be any existing context where incoming calls come into your dial plan.

DUNDI.CONF

Now we will look at the dundi.conf specific configuration. This is where we setup our peering relationship between servers and configure DUNDi responses. We will setup the basics requirements in effort to keep explanation simple and easy. DUNDi uses its own protocol on UDP port 4520 to exchange DUNDi query messages. DUNDi does not communicate queries or responses through IAX, just the calls go through the [priv] IAX channel. You can test this by commenting out the iax.conf context [priv], reload chan_iax.so and you can still have DUNDi query's respond for valid extensions.

The [general] section has several parameters that require modification for your specific server to work with others properly. Most of this section is described fairly well in the conf file and the defaults are fine to get up and running.

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

This information is very important when peering with servers outside your organization. At the Asterisk CLI> the command (dundi query {entityID}) will retrieve this information from a remote peer for contact.

```
;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
Cachetime tells the far end PBX how long to cache the query result in its local cache.

ttl=2
For the sake of this paper, we set the ttl (time to live) value to so we don't query past 2 servers, A to B to C. If we allow more hops then a loop could occur, A to B to C to A to B to C etc, not good. Be careful with ttl in a closed loop DUNDi system.

```
autokill=yes
;secretpath=dundi
;storehistory=yes

[mappings]
priv => dundiextens,0,IAX2,priv:${SECRET}@10.10.14.121/${NUMBER},nopartial
```

When a [priv] dundi lookup request comes in, this PBX will advertise whatever extensions are present in the [dundiextens] context in the dialplan (extensions.conf) and if the queried extension exists here, this PBX will send back the contact information in this format:
IAX2/priv:${SECRET}@the ip address of this server/(the extension number requested)
nonpartial tells this mapping to respond only to an exact matching extension request (no pattern matching)

The peer section identifies who this PBX peers with:

```
[00:14:22:23:26:2E]  ;(this is the MAC address of the DUNDi peer PBX B)
model = symmetric
host = 10.10.14.122  ;(this is the IP address of the DUNDi peer B)
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary
```

We use the same inkey/outkey for PBX A, B & C. Sharing the same key across the test servers is much easier than keeping track of a set for each server. To generate the keys:
cd /var/lib/asterisk/keys
#astgenkey -n priv
Do not put a password on the keys. This will generate 2 files (priv.key and priv.pub) Copy the new priv keys to PBX B and C to the /var/lib/asterisk/keys directories. You must have openssl package installed on the system to generate the keys.

In this paper we will setup PBX A to peer with B, PBX B to peer with A and C, and PBX C to peer with B. Here are the dundi.conf files for each PBX:

PBX A:

```
lab1:/etc/asterisk# more dundi.conf
[general]
department=Your Department
organization=Your Company, Inc.
locality=Your City
stateprov=ST
country=US
email=your@email.com
phone=+12565551212
;bindaddr=0.0.0.0
;port=4520
entityid=00:B0:D0:CB:80:AA
cachetime=5
ttl=2
autokill=yes
;secretpath=dundi
;storehistory=yes

[mappings]
priv => dundiextens,0,IAX2,priv:${SECRET}@10.10.14.121/${NUMBER},nopartial

[00:B0:D0:CB:80:7A]     ;PBX B
model = symmetric
host = 10.10.14.122
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary
```

PBX B:

```
 [general]
department=Your Department
organization=Your Company, Inc.
locality=Your City
stateprov=ST
country=US
email=your@email.com
phone=+12565551212
;bindaddr=0.0.0.0
;port=4520
entityid=00:B0:D0:CB:80:7A
cachetime=5
ttl=2
autokill=yes
;secretpath=dundi
;storehistory=yes

[mappings]
priv => dundiextens,0,IAX2,priv:${SECRET}@10.10.14.122/${NUMBER},nopartial

[00:B0:D0:CB:80:AA]     ;PBX A
model = symmetric
host = 10.10.14.121
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary

[00:B0:D0:CC:5C:E8]     ;PBX C
model = symmetric
host = 10.10.14.123
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary
```

PBX C:

```
 [general]
department=Your Department
organization=Your Company, Inc.
locality=Your City
stateprov=ST
country=US
email=your@email.com
phone=+12565551212
;bindaddr=0.0.0.0
;port=4520
entityid=00:B0:D0:CC:5C:E8
cachetime=5
ttl=2
autokill=yes
;secretpath=dundi
;storehistory=yes

[mappings]
priv => dundiextens,0,IAX2,priv:${SECRET}@10.10.14.123/${NUMBER},nopartial

[00:B0:D0:CB:80:7A]     ;PBX B
model = symmetric
host = 10.10.14.122
inkey = dundi
outkey = dundi
include = priv
permit = priv
qualify = yes
order = primary
```

Once the config files are updated, restart Asterisk or at the CLI>reload pbx_dundi.so

Use the “dundi show peers” command at the Asterisk CLI> to see connection state with the peers identified in dundi.conf:

```
PBXA*CLI> dundi show peers
EID              Host            Model      AvgTime  Status
00:b0:d0:cb:80:7a    10.10.14.122    (S) Symmetric  Unavail  OK (1 ms)
1 dundi peers [1 online, 0 offline, 0 unmonitored]


PBXB*CLI> dundi show peers
EID              Host            Model      AvgTime  Status
00:b0:d0:cc:5c:e8    10.10.14.123    (S) Symmetric  Unavail  OK (1 ms)
00:b0:d0:cb:80:aa    10.10.14.121    (S) Symmetric  Unavail  OK (1 ms)
2 dundi peers [2 online, 0 offline, 0 unmonitored]

PBXC*CLI> dundi show peers
EID              Host            Model      AvgTime  Status
00:b0:d0:cb:80:7a    10.10.14.122    (S) Symmetric  Unavail  OK (1 ms)
1 dundi peers [1 online, 0 offline, 0 unmonitored]
```

EXTENSIONS.CONF

This is where we announce DUNDi extensions, route the calls and query other servers.

For the sake of this paper, we have a very basic extensions.conf file so we can focus on DUNDi. Here are the PBX extensions.conf files for each PBX.

PBX A extensions.conf

```
[general]
static=yes
writeprotect=no
autofallthrough=yes
clearglobalvars=no
priorityjumping=no

[lookupdundi]
switch => DUNDi/priv

[dundiextens]
exten => 101,1,NoOp
exten => 102,1,NoOp
exten => 103,1,NoOp

[incomingdundi]
exten => 101,1,Goto(internal|101|1)
exten => 102,1,Goto(internal|102|1)
exten => 103,1,Goto(internal|103|1)

[internal]
exten => 101,1,Answer
exten => 101,2,Dial(SIP/101|30|tr)
exten => 101,3,Hangup

exten => 102,1,Answer
exten => 102,2,Dial(SIP/102|30|tr)
exten => 102,3,Hangup

exten => 103,1,Answer
exten => 103,2,Dial(SIP/103|30|tr)
exten => 103,3,Hangup

exten => _2XX,1,Goto(lookupdundi|${EXTEN}|1)
exten => _3XX,1,Goto(lookupdundi|${EXTEN}|1)
```

PBX B extensions.conf

```
[general]
static=yes
writeprotect=no
autofallthrough=yes
clearglobalvars=no
priorityjumping=no

[lookupdundi]
switch => DUNDi/priv

[dundiextens]
exten => 201,1,NoOp
exten => 202,1,NoOp
exten => 203,1,NoOp

[incomingdundi]
exten => 201,1,Goto(internal|201|1)
exten => 202,1,Goto(internal|202|1)
exten => 203,1,Goto(internal|203|1)

[internal]
include => lookupdundi

exten => 201,1,Answer
exten => 201,2,Dial(SIP/201|30|tr)
exten => 201,3,Hangup

exten => 202,1,Answer
exten => 202,2,Dial(SIP/202|30|tr)
exten => 202,3,Hangup

exten => 203,1,Answer
exten => 203,2,Dial(SIP/203|30|tr)
exten => 203,3,Hangup
```

PBX C extensions.conf

```
[general]
static=yes
writeprotect=no
autofallthrough=yes
clearglobalvars=no
priorityjumping=no

[lookupdundi]
switch => DUNDi/priv

[dundiextens]
exten => 301,1,NoOp
exten => 302,1,NoOp
exten => 303,1,NoOp
```

```
[incomingdundi]
exten => 301,1,Goto(internal|301|1)
exten => 302,1,Goto(internal|302|1)
exten => 303,1,Goto(internal|303|1)

[internal]
include => lookupdundi

exten => 301,1,Answer
exten => 301,2,Dial(SIP/301|30|tr)
exten => 301,3,Hangup

exten => 302,1,Answer
exten => 302,2,Dial(SIP/302|30|tr)
exten => 302,3,Hangup

exten => 303,1,Answer
exten => 303,2,Dial(SIP/303|30|tr)
exten => 303,3,Hangup
```

There is one difference between PBX A and B/C and that is in the [internal] context of PBX A you will find these 2 extensions:

```
exten => _2XX,1,Goto(lookupdundi|${EXTEN}|1)
exten => _3XX,1,Goto(lookupdundi|${EXTEN}|1)
```

These are for pattern matching of known extensions on PBX B and C. This is one way to get the call to the DUNDi switch statement context [lookupdundi]. Another way is to use an 'include => lookupdundi' statement in the [internal] context like in PBX B and C.
As we take a closer look into extensions.conf, we find a context called [lookupdundi]. This context has a switch statement 'switch => DUNDi/priv', this is where DUDNi query's the peers and request extension location information. Notice the '/priv' this is the DUNDi mapping we are querying, so if you have a mapping on another server with a name of 'george' and you wanted to search for an extension there, then your switch statement would be 'switch => DUNDi/george'

This becomes very powerful when you need to query PBX's for different things like extensions, voicemail boxes, outbound trunks, you can be very creative with the use of mappings.

The next context is [dundiextens] which has the actual extensions that this PBX responds with location information for. This context corresponds to the priv mapping in dundi.conf. This is easily tested by removing or adding extensions in this context and watching the dundi lookup responses from the CLI> on another server.

The [incomingdundi] context is the entry point at which dundi calls coming to this server hit the dial plan. This corresponds to the 'context=incomingdundi' entry in the [priv] context in iax.conf. All we do here is send the call to the actual extension in [internal] context with a Goto command.

Testing the Setup

Now that we have the PBX's setup for DUNDi, lets take a look at some CLI> outputs from some actual dundi lookups.

Example 1: From PBX A, at the Asterisk CLI> initiate a dundi lookup for exten 201 and then exten 205 (not available on any peer):

```
PBXA*CLI> dundi lookup 201@priv
  1.    0 IAX2/priv:Q4pFZoZInT6ORBgemoDkIA@10.10.14.122/201 (EXISTS)
    from 00:b0:d0:cb:80:7a, expires in 5 s
DUNDi lookup completed in 40 ms

lab1*CLI> dundi lookup 205@priv
DUNDi lookup returned no results.
DUNDi lookup completed in 65 ms
```

Here we see a very useful CLI tool to determine if DUNDi is working and if the extension is available or not. Also the 'dundi debug' command will also give you good detail of the transaction between the requesting PBX and the peering PBX. Enable dundi debug on both PBX A and B, you will see the Tx (transmit) and Rx (receive) messages between the PBX's. Unlike most debug messaging, DUNDi debugs are relatively simple to follow and understand.

Example 2: A call from exten 101 on PBX A to exten 201 on PBX B.

PBX A:

```
PBXA*CLI>
   -- Executing [201@internal:1] Goto("SIP/101-081d2f58", "lookupdundi|201|1") in new stack
   -- Goto (lookupdundi,201,1)
   -- Called priv:UvTZKI5ej1fM4XQmPE9+Cg@10.10.14.122/201
   -- Call accepted by 10.10.14.122 (format ulaw)
   -- Format for call is ulaw
   -- IAX2/10.10.14.122:4569-1 answered SIP/101-081d2f58
   -- Hungup 'IAX2/10.10.14.122:4569-1'
  == Spawn extension (lookupdundi, 201, 1) exited non-zero on 'SIP/101-081d2f58'
```

Notice the call is sent to the [lookupdundi] context and then sent over to the 10.10.14.122 PBX B, call accepted, ulaw, answered then hungup. Looks like any other PBX-to-PBX IAX trunk call except that the password in a bit long.

PBX B:

```
PBXB*CLI>
   -- Accepting AUTHENTICATED call from 10.10.14.121:
      > requested format = ulaw,
      > requested prefs = (ulaw),
      > actual format = ulaw,
      > host prefs = (ulaw|gsm),
      > priority = mine
   -- Executing Goto("IAX2/priv-1", "internal|201|1") in new stack
   -- Goto (internal,201,1)
   -- Executing Answer("IAX2/priv-1", "") in new stack
```

```
   -- Executing Dial("IAX2/priv-1", "SIP/201|30|tr") in new stack
   -- Called 201
   -- SIP/201-08181d88 is ringing
  == Spawn extension (internal, 201, 2) exited non-zero on 'IAX2/priv-1'
   -- Hungup 'IAX2/priv-1'
```

Notice the call is accepted, ulaw, hits the [incomingdudni] context then is sent to the [internal] context at exten 201. This is normal call progressing like any other call, internally or externally.

Example 3: From exten 101 on PBX A, we call exten 301 on PBX C. Keep in mind that PBX A can find extensions on PBX C, but only through PBX B. So the query will go from A to B to C, but the actual call will go from A to C directly.

PBX A:

```
PBXA*CLI>
   -- Executing [301@internal:1] Goto("SIP/101-081d2f58", "lookupdundi|301|1") in new stack
   -- Goto (lookupdundi,301,1)
   -- Called priv:qef1LMx6q0cuAf1S9QbgXw@10.10.14.123/301
   -- Call accepted by 10.10.14.123 (format ulaw)
   -- Format for call is ulaw
   -- IAX2/10.10.14.123:4569-1 answered SIP/101-081d2f58
   -- Hungup 'IAX2/10.10.14.123:4569-1'
  == Spawn extension (lookupdundi, 301, 1) exited non-zero on 'SIP/101-081d2f58'
```

PBX B:

```
PBXB*CLI>
```

Notice nothing happens on PBX B, the call does not come through this PBX. You will not see DUNDi transactions unless you enable ‘dundi debug’ at the CLI>. With dundi debug enabled on PBX B, you will see the Rx query from PBX A, then PBX B will re-distribute the query to PBX C, then PBX C will respond to PBX B with good contact information, then PBX B will respond back to PBX A with the contact info form PBC C to contact exten 301 directly. This paper is long enough so I did not want to include debug messaging here.

PBX C:

```
lab3*CLI>
   -- Accepting AUTHENTICATED call from 10.10.14.121:
      > requested format = ulaw,
      > requested prefs = (ulaw),
      > actual format = ulaw,
      > host prefs = (ulaw|gsm),
      > priority = mine
   -- Executing Goto("IAX2/10.10.14.121:4569-1", "internal|301|1") in new stack
   -- Goto (internal,301,1)
   -- Executing Answer("IAX2/10.10.14.121:4569-1", "") in new stack
   -- Executing Dial("IAX2/10.10.14.121:4569-1", "SIP/301|30|tr") in new stack
   -- Called 301
   -- SIP/301-0685 is ringing
   -- SIP/301-0685 is ringing
  == Spawn extension (internal, 301, 2) exited non-zero on 'IAX2/10.10.14.121:4569-1'
   -- Hungup 'IAX2/10.10.14.121:4569-1'
```

Closing Notes

The wiki has some great information on DUNDi and also examples of how to manage e164.

http://www.voip-info.org/wiki-DUNDi

http://www.voip-info.org/wiki-Asterisk+DUNDi+Call+Routing

http://www.astricon.net/files/usa06/Friday-General_Conference/JR_Richardson_Whitepaper.pdf

Also a quick google search for 'asterisk dundi' will pull up some great information on other presentations, explanations, user comments from mailing list, etc.

DUNDi is a great protocol and relatively easy to implement once the basic concepts and configurations are fully understood. So my advice is to start with a very simple setup like the one described above, and layer in complexity only after you get comfortable with using and debugging the protocol.

Hope this helps.

JR