



Why Cluster?

Exploring some of the decentralizing
tools in Asterisk 1.4 and beyond

Overview

- We'll look at some of the currently available tools in Asterisk we can use for clustering, in addition to making you aware of the tools that can further be used in our quest for a distributed federation of servers in future versions of Asterisk -- right around the corner!



Overview

- A high level overview of topologies and technologies we can use right now to start building distributed systems, but we don't have time to look at every nut and bolt



Overview

- Will be from the view of my personal experience in building a distributed virtual PBX system for an Internet Telephony Service Provider, and any tools I am familiar with that may be of a benefit to you (and me), in the future
- Clustering comes in many flavours, and a "drop in" solution is not yet possible



Why do I want to cluster?

- Lets you scale up the number of calls simply by adding another node (server)
- If a server dies, the rest of the cluster will take over the load

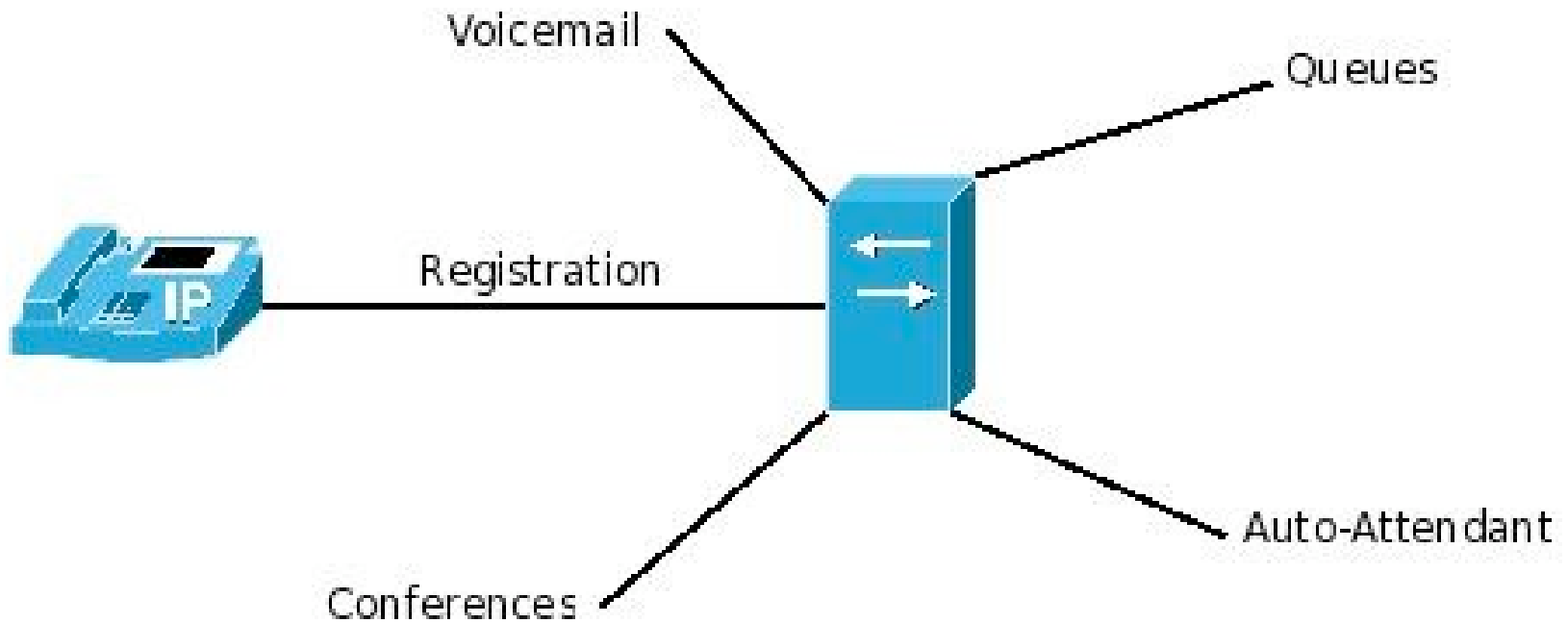


Technologies Utilized

- PostgreSQL (database)
- UnixODBC (database abstraction layer)
- DUNDi (information query system)
- Static and Dynamic Realtime (centralized configuration)
- func_odbc ("Just in time" data from the DB into the dialplan)
- And more!



Single Server

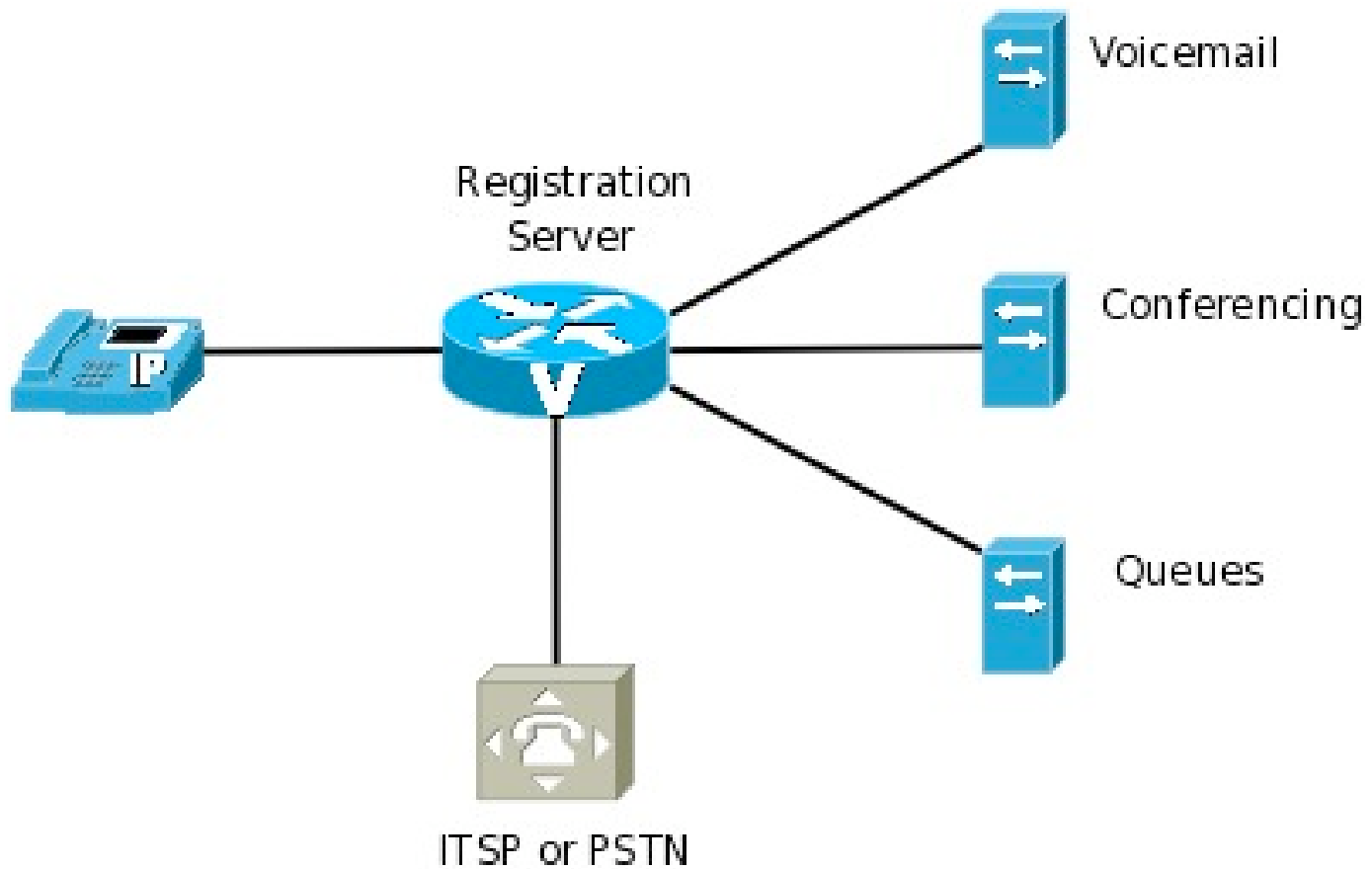


Single Server

- Handles all features of your PBX system
- No redundancy, no fail-over; if the system goes down, your phones are down
- Typical system would handle 20 phones and 2-4 phone lines



Multi-Server



Multi-Server

- Multiple boxes, each one handles one or more services
- Voicemail server, Queue server, Conferencing server, etc...
- Lets you scale a bit better by directing heavy load applications to dedicated machines



Cluster Topologies

- Centralized registration server
- Distributed registration server
- Hybrid registration server

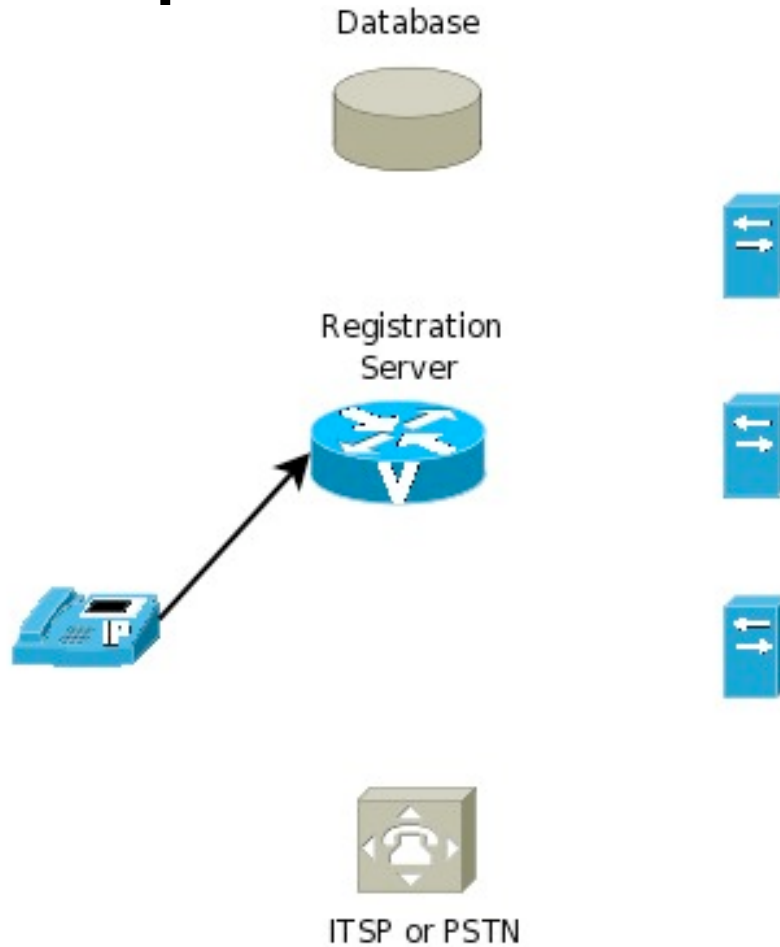


Centralized registration server

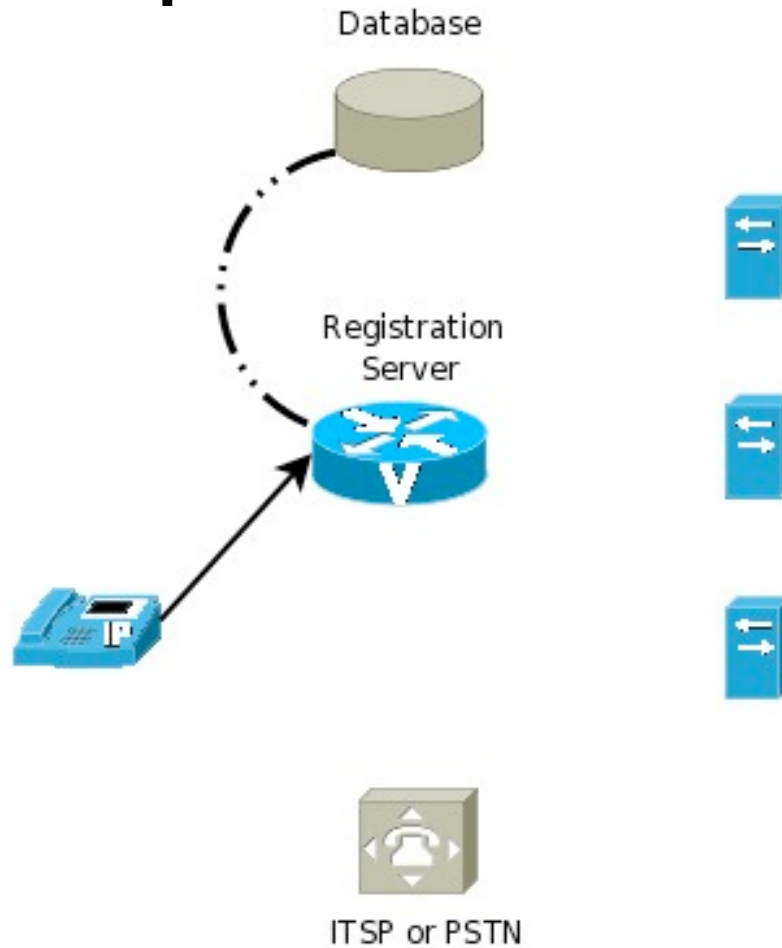
- Central point for registration distributes calls to media servers
- Phones register to a single point, and that server controls distribution (think softswitch)
- Can utilize either Asterisk or OpenSER as the registration server (no media handling)
- All endpoints register to a single place, so no need to seek them out



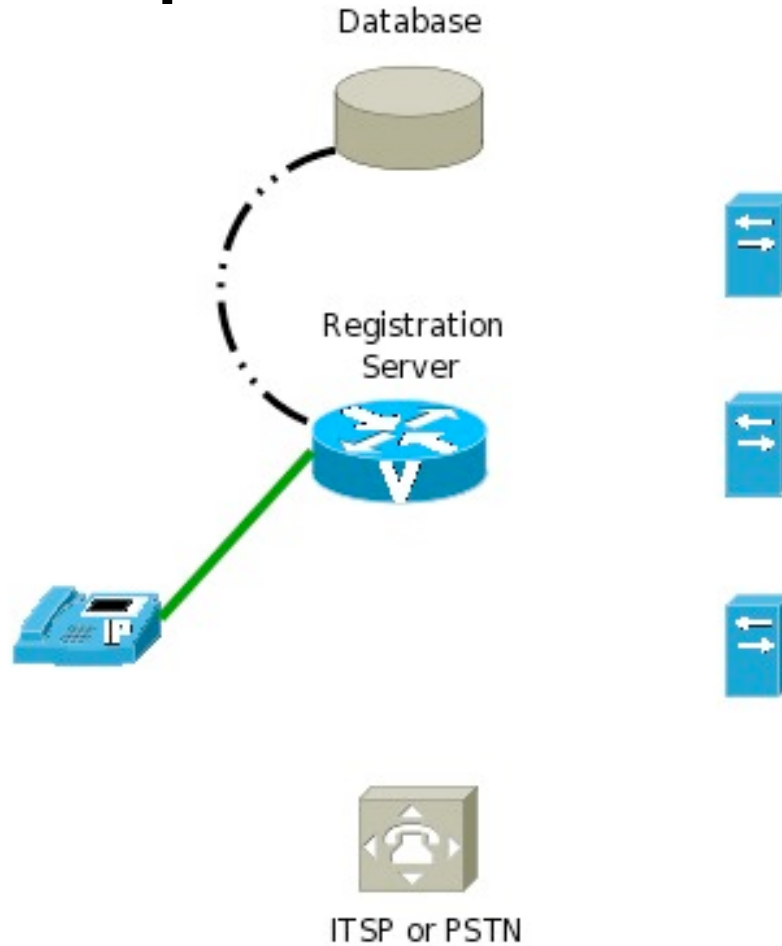
Sample Call Flow



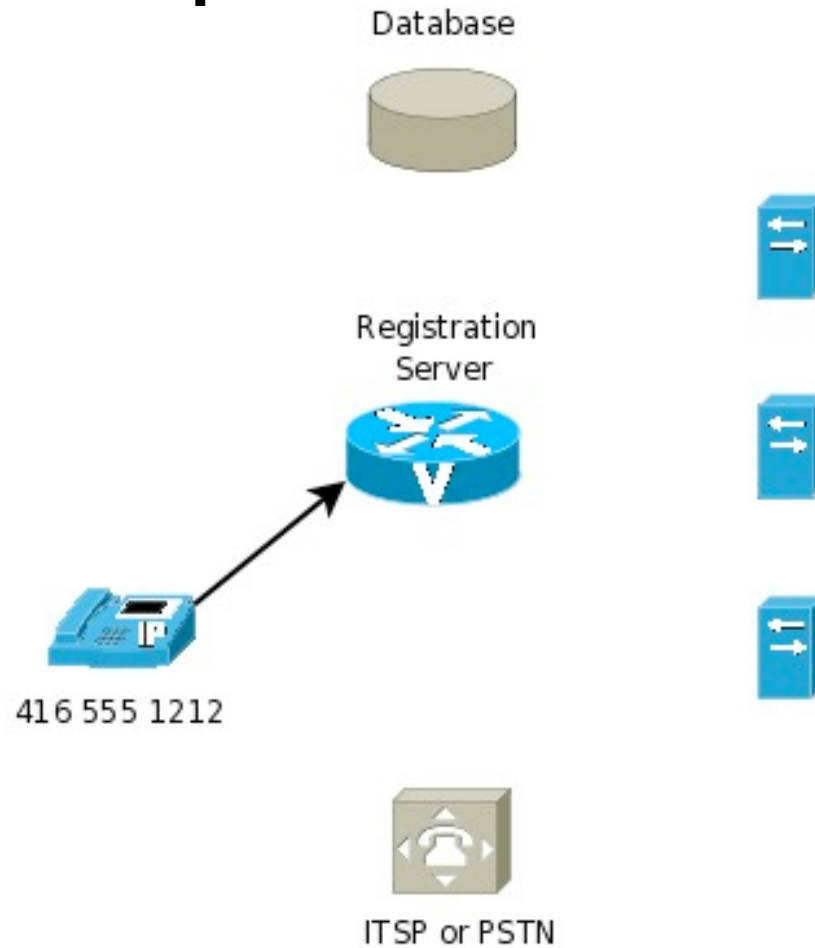
Sample Call Flow



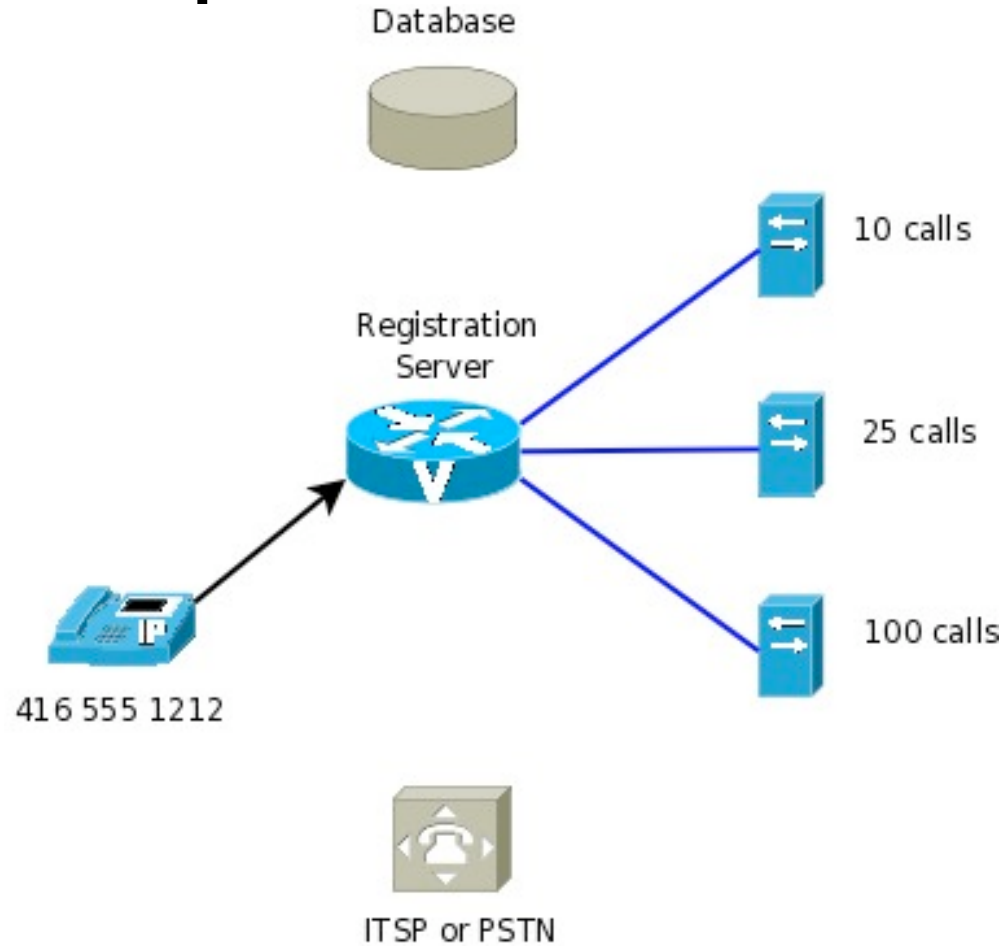
Sample Call Flow



Sample Call Flow



Sample Call Flow



Using functions in DUNDi

- `conferences =>`
`conference_cluster,0,SIP,${GROUP_COUNT(${NUMBER}@conference)}-${SYSTEMNAME}`
- `queues =>`
`queues_cluster,0,SIP,${GROUP_COUNT(${NUMBER}@queues)}-${SYSTEMNAME}`
- `trunk_usage =>`
`trunk_usage,0,SIP,${GROUP_COUNT(${NUMBER}@usage)}-${SYSTEMNAME}`



DUNDIRESULT () and DUNDIQUERY ()

- Unfortunately, in 1.4, we have to query each server individually – but in 1.6, we have solved this with DUNDIQUERY () and DUNDIRESULT ()
- DUNDIQUERY () retrieves the data from the servers, and DUNDIRESULT () parses it

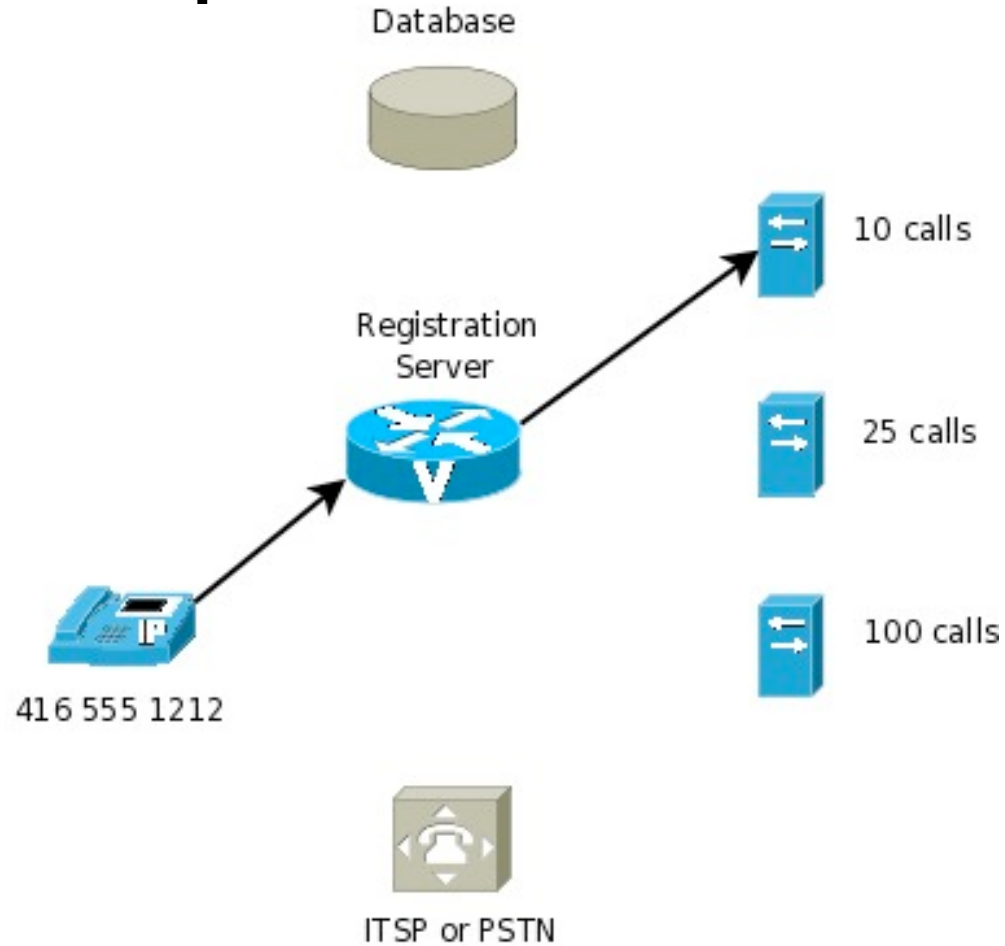


DUNDIQUERY () Example

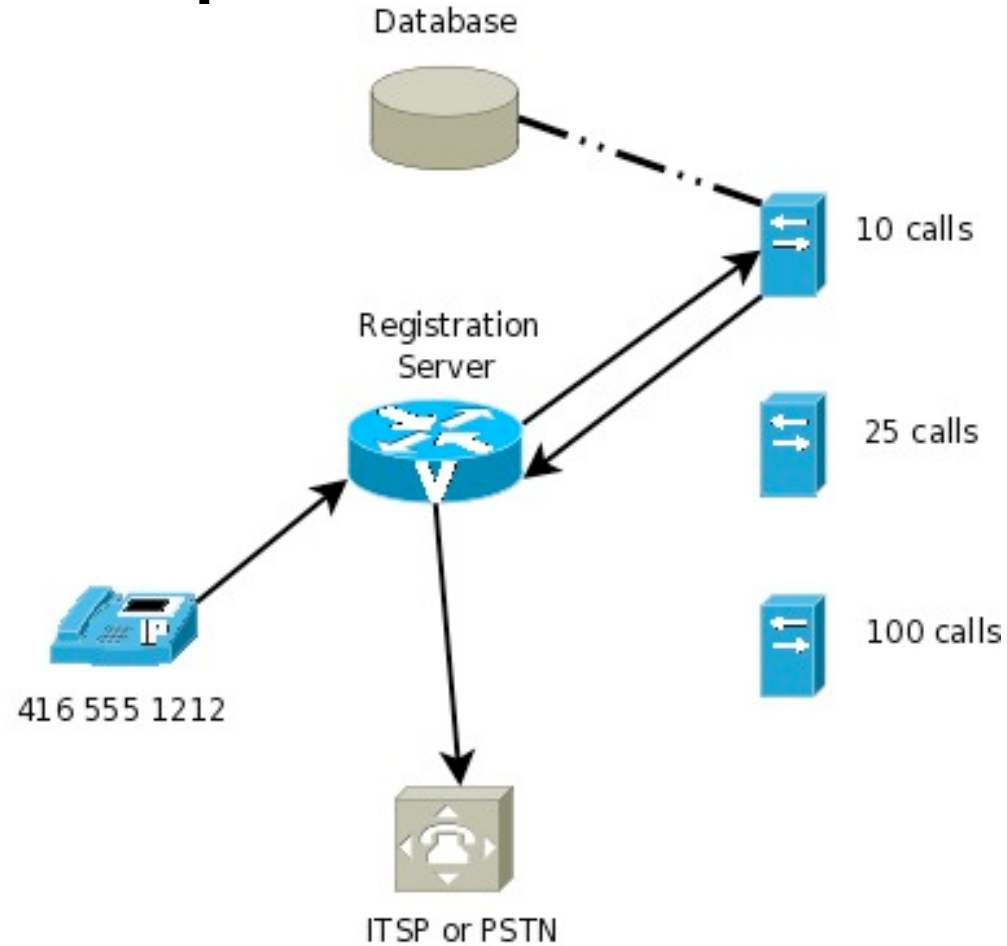
- `exten => _X.,1,Set(ID=${DUNDIQUERY(${USERNAME} | trunk_usage|b)})`
- `exten => _X.,n,Set(NUM=${DUNDIRESLT(${ID} | getnum)})`
- `exten => _X.,n,NoOp(There are ${NUM} results)`
- `exten => _X.,n,Set(X=1)`
- `exten => _X.,n,While(${X} <= ${NUM})`
- `exten => _X.,n,NoOp(Result ${X} is ${DUNDIRESLT(${ID} | ${X})})`
- `exten => _X.,n,Set(X=${X} + 1)`
- `exten => _X.,n,EndWhile`



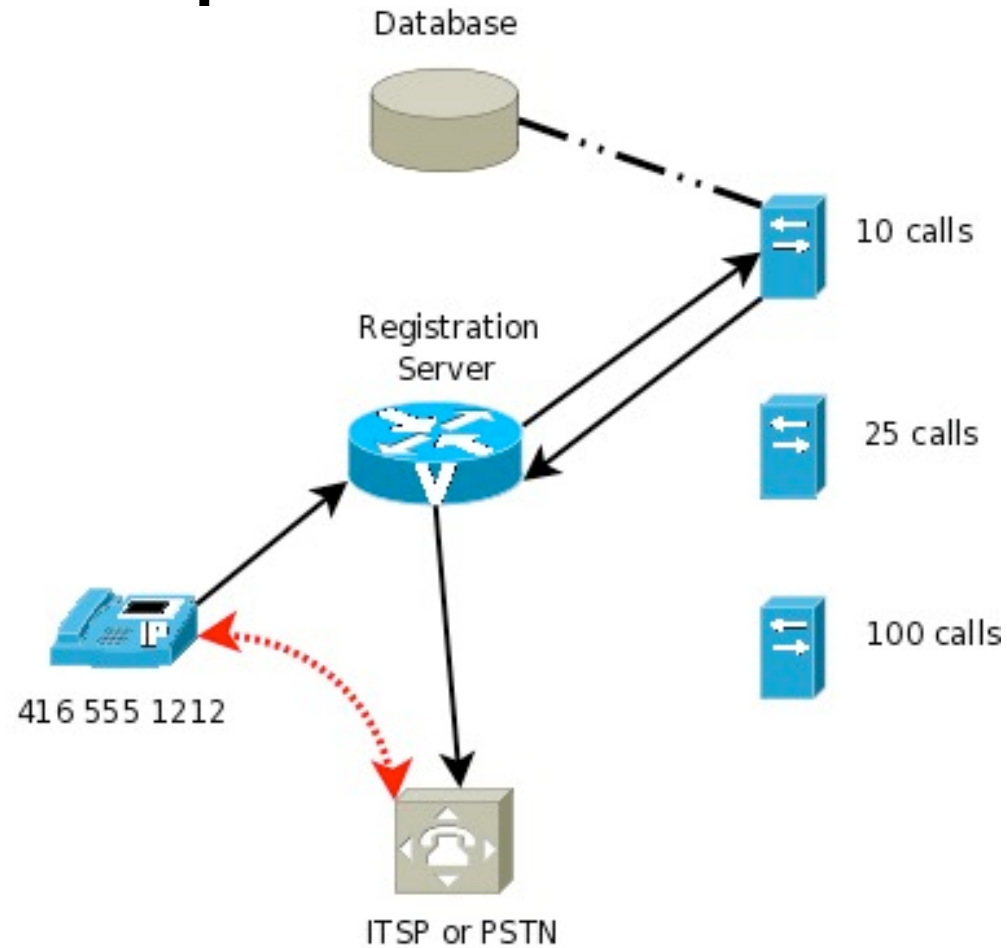
Sample Call Flow



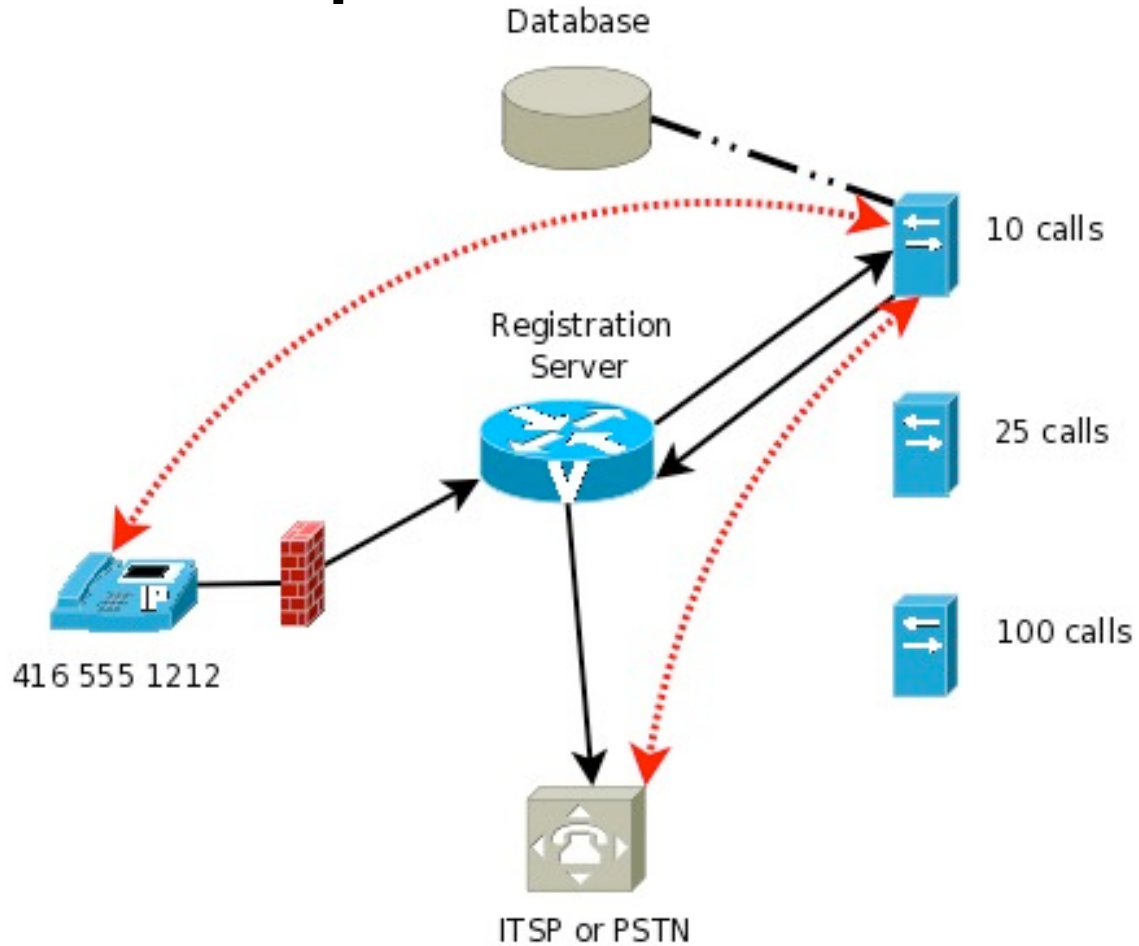
Sample Call Flow



Sample Call Flow



Sample Call Flow



Centralized Advantages

- Easier to control where to send calls based on prior load testing (DUNDi and GROUP_COUNT function)
- May be simpler to administer (hub and spoke topology)



Centralized Disadvantages

- It's... well... centralized -- single point of failure
- May be a ceiling on the number of registrations the server can handle

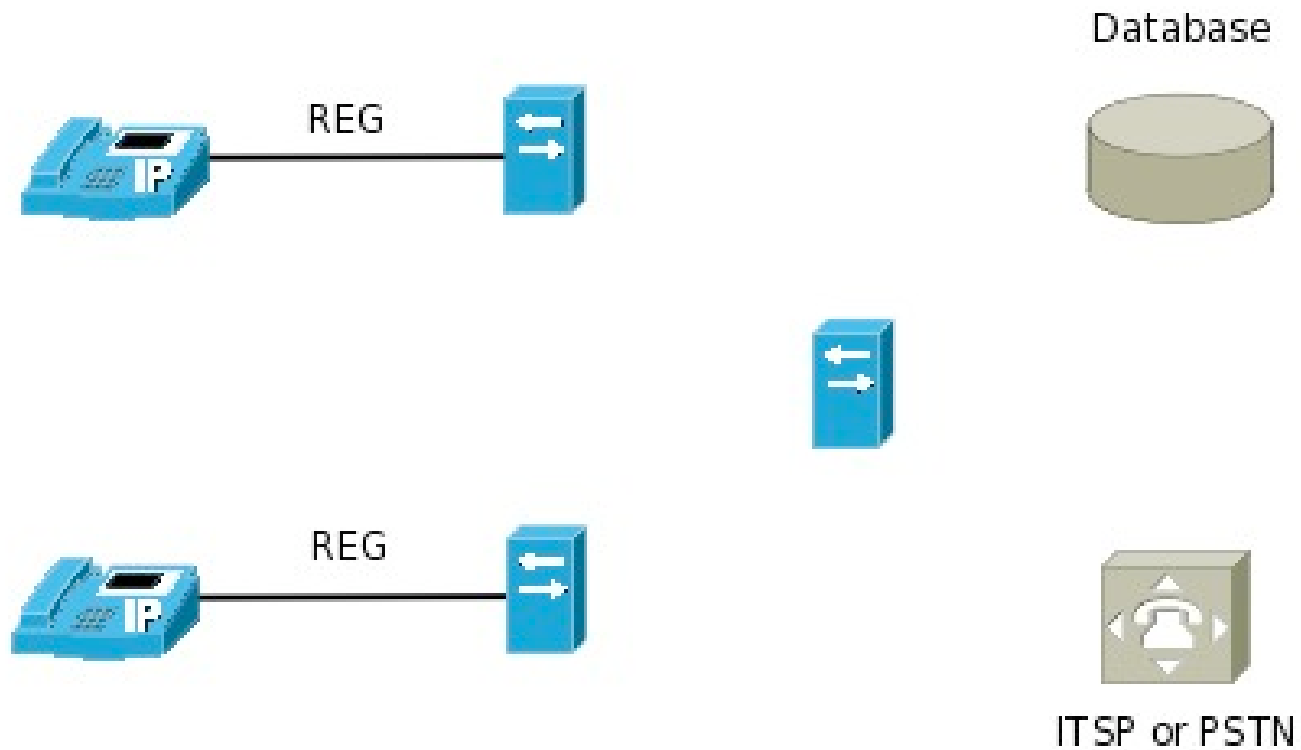


Distributed registration server

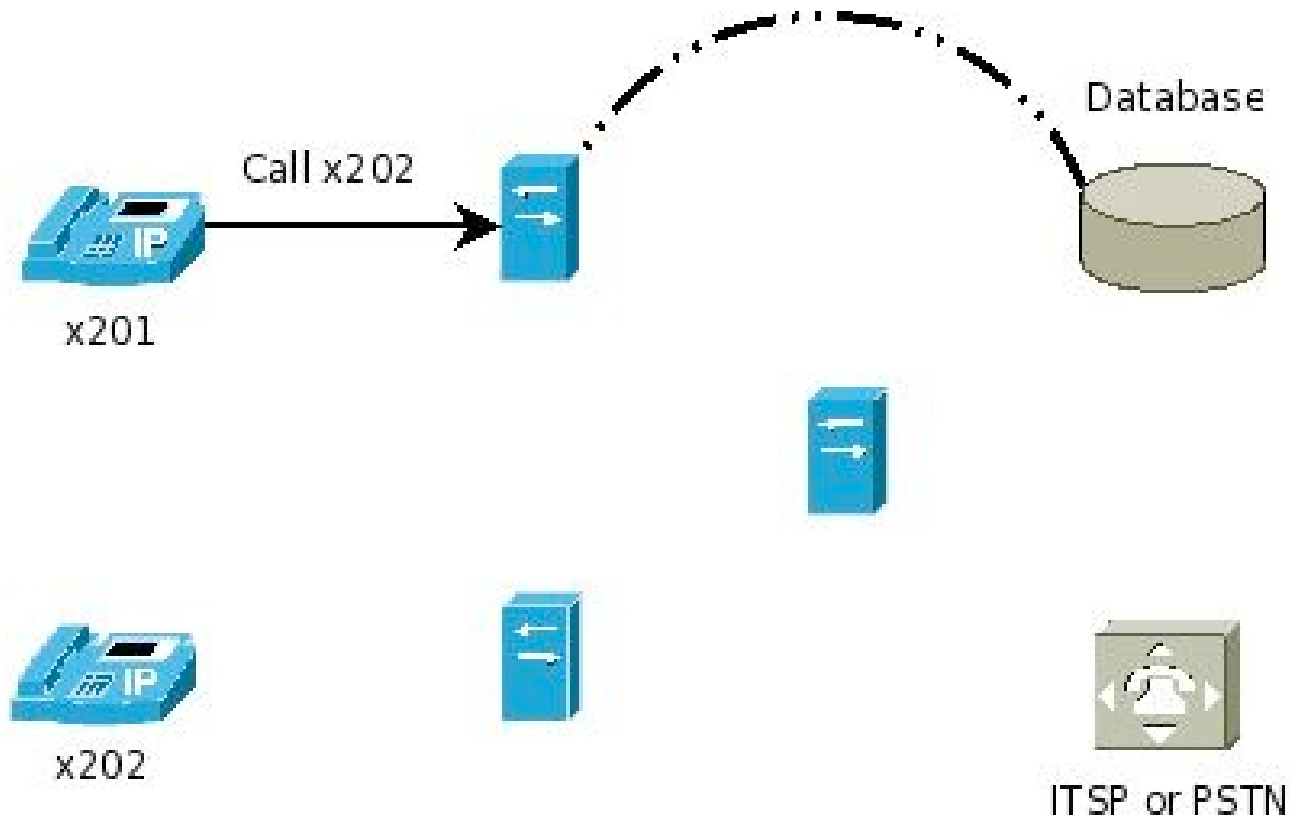
- No centralized distribution server
- Phones can register to any server in the cluster
- Asterisk seeks out the registration point and delivers the call
- Uses DUNDi and regexten/regcontext methods (*found in sip.conf and iax.conf*) to find endpoints



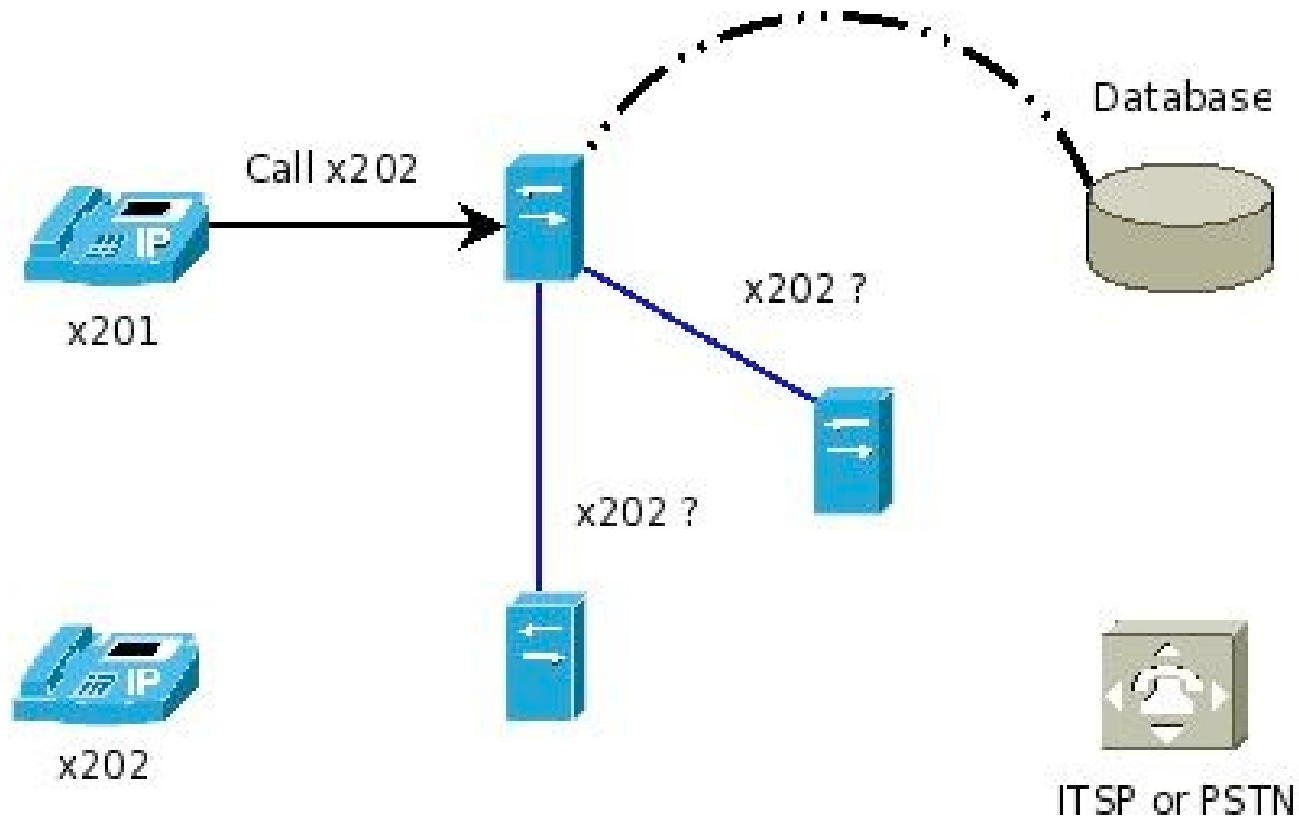
Sample Call Flow



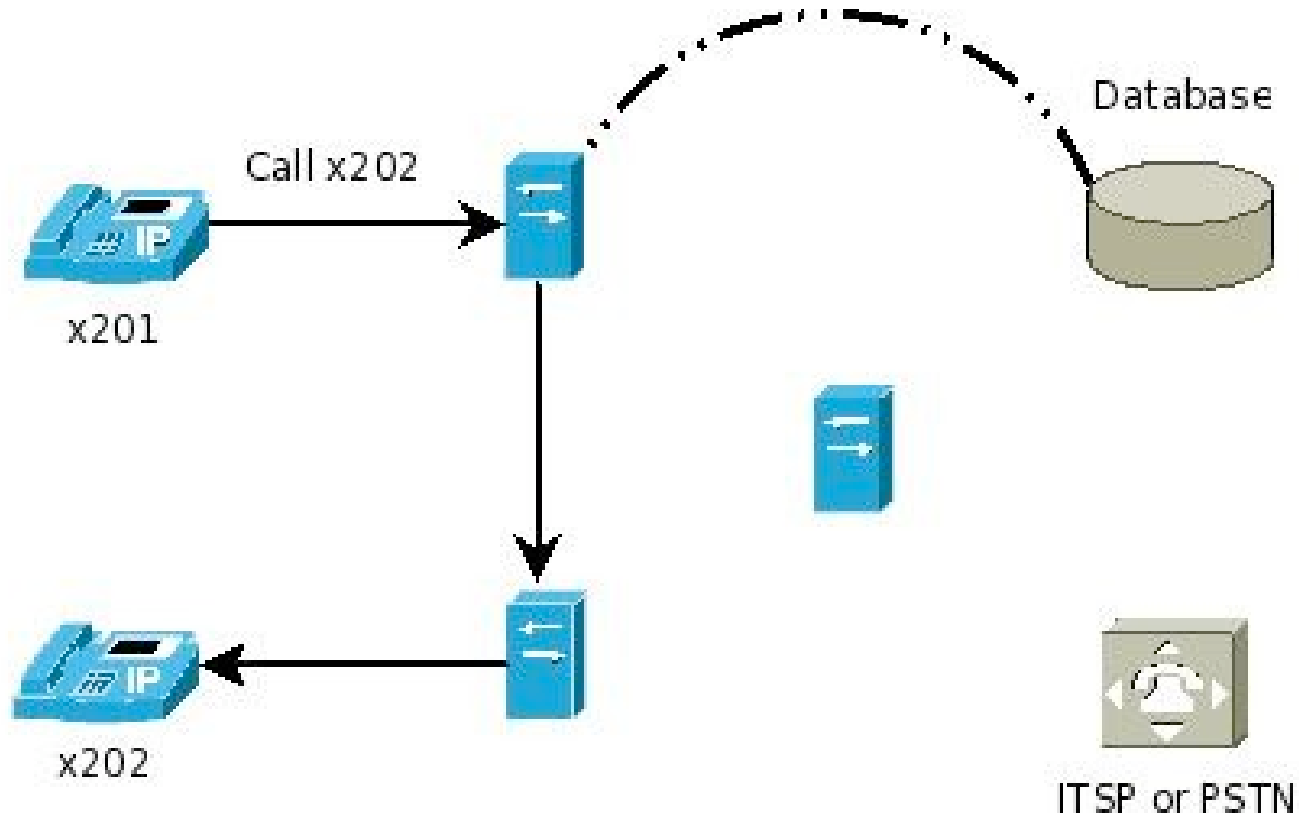
Sample Call Flow



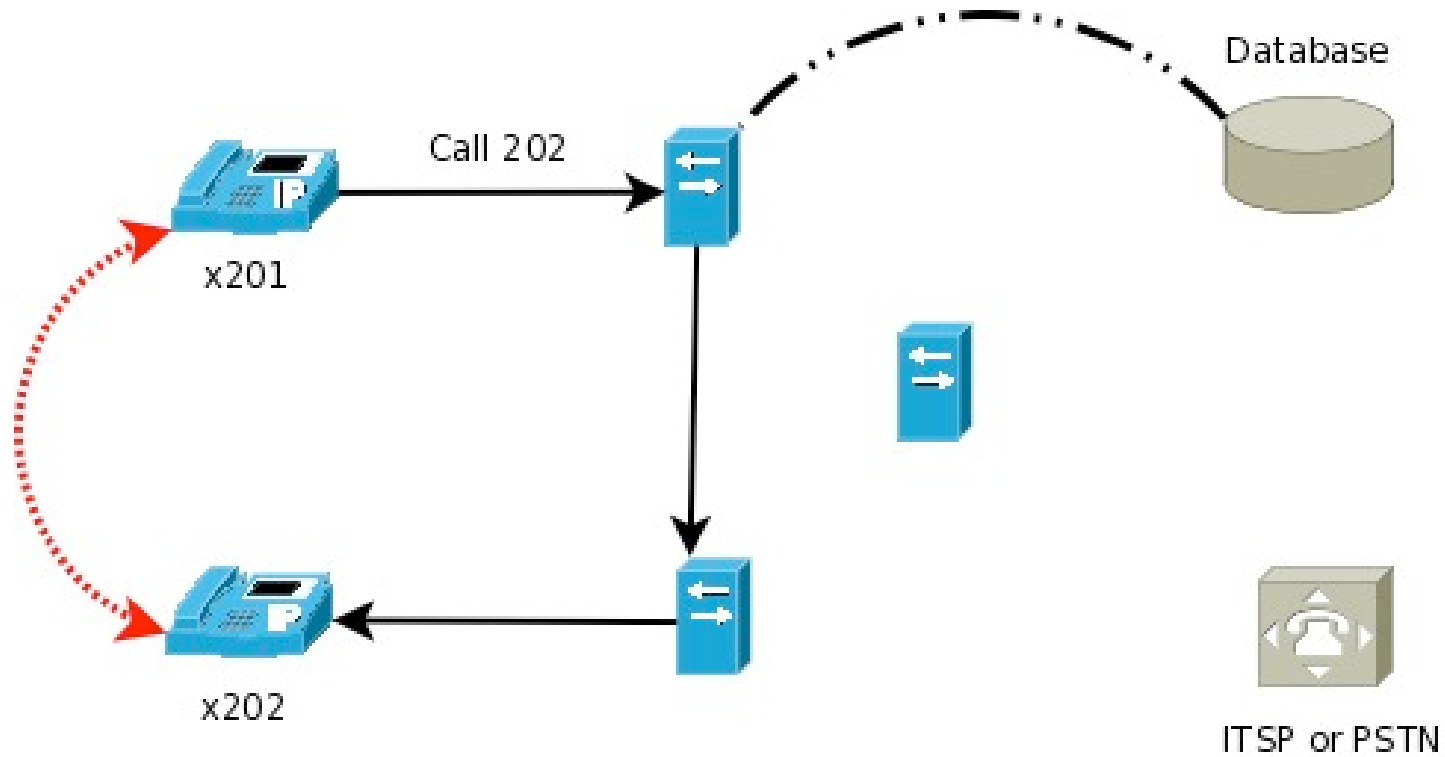
Sample Call Flow



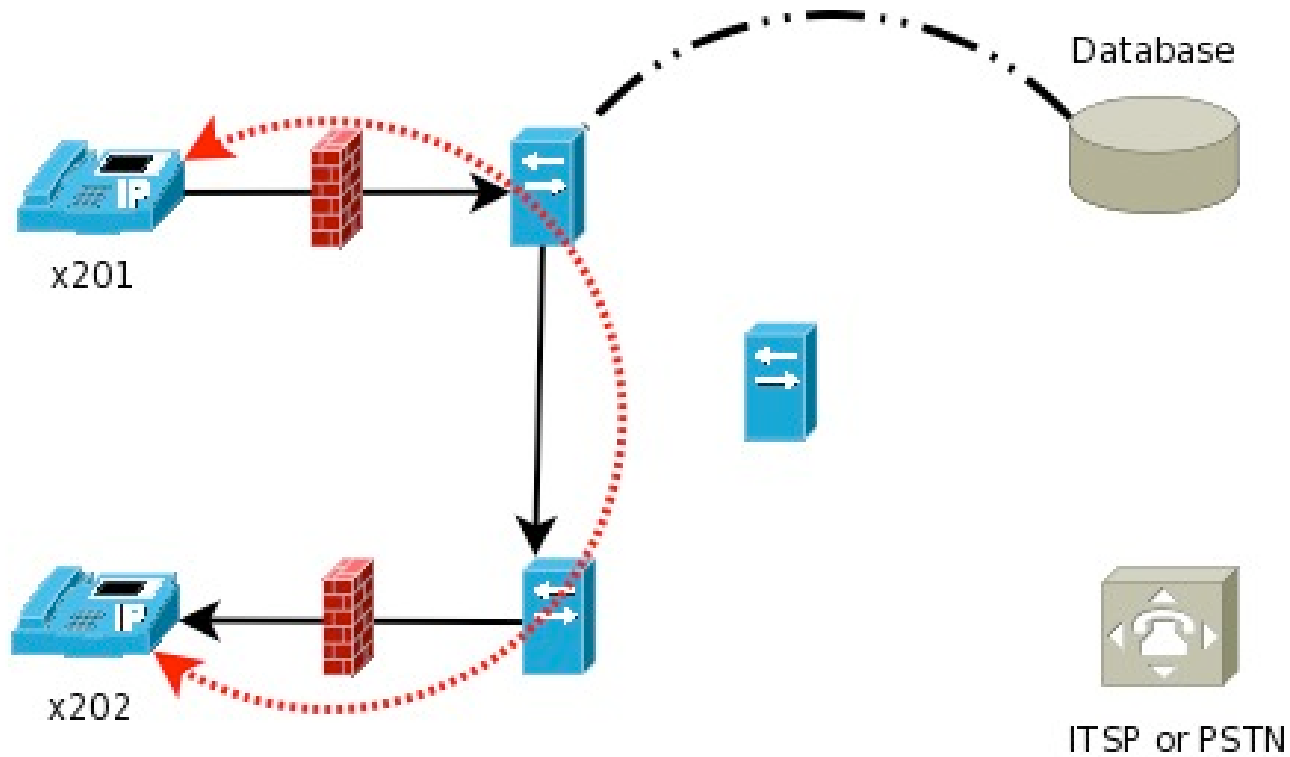
Sample Call Flow



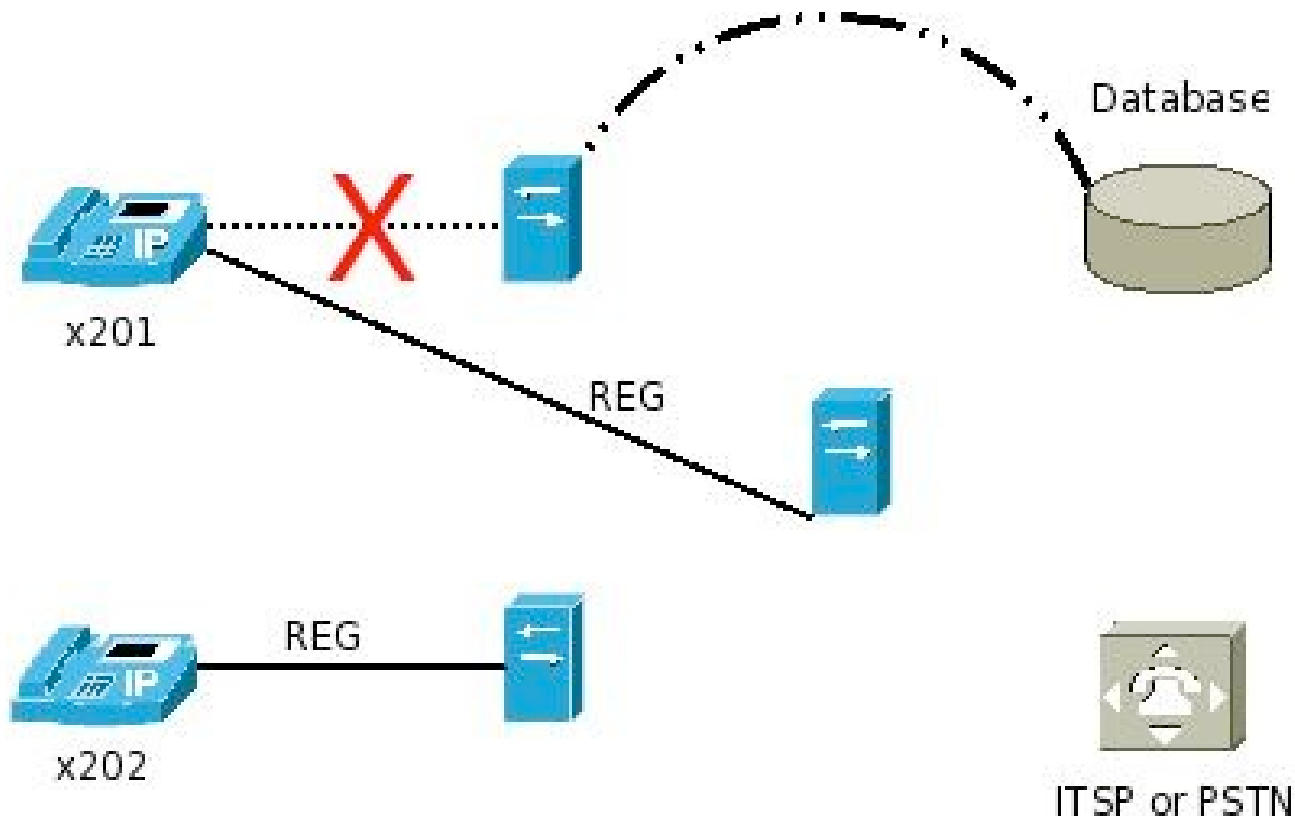
Sample Call Flow



Sample Call Flow



Failed Registration Server



Distributed Advantages

- No central registration point to fail; phones can auto-failover
- Creates a "self-healing" network by nature



Distributed Disadvantages

- Harder to control number of calls each server handles
- Requires careful network planning in order to scale
- Fully meshed networks are fault tolerant, but are not (reasonably) scalable

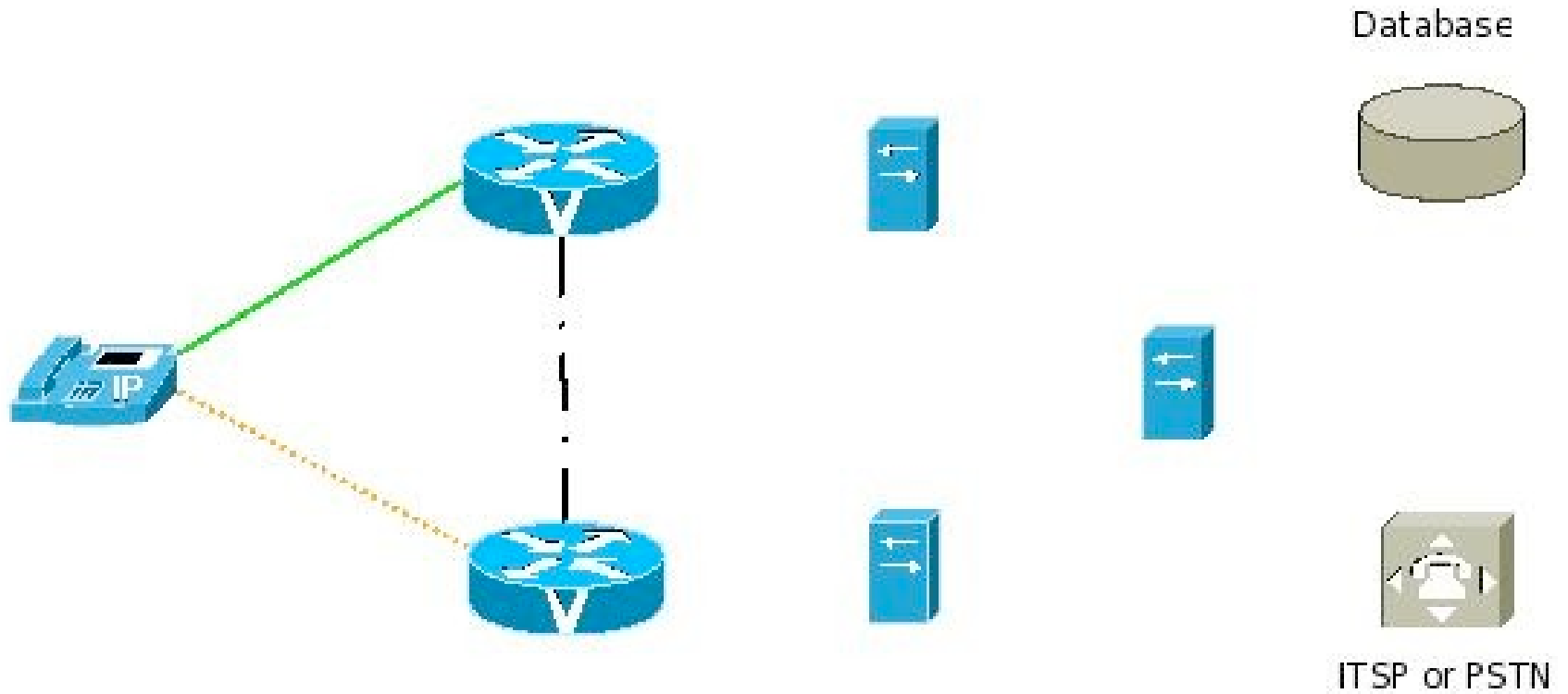


Hybrid registration server

- Combine the advantages of distributed and centralized registration servers
- Utilize heartbeat to auto-failover registration server to another server in the LAN
- OR, run a pair of registration servers, balance the registrations between them, and have the phones fail-over if a server dies
- Still have separate servers handle media and processing of calls based on prior load testing



Hybrid topology



Hybrid Advantages

- Get the best of both distributed and centralized models
- Better tolerance to system failure



Hybrid Disadvantages

- May be more complicated of a topology to initially configure and maintain
- More technologies involved could possibly mean more places to break



Clustering Tools

- DUNDi
- Realtime / UnixODBC
(database integration)
- func_odbc
- Centralized voicemail: IMAP or ODBC



DUNDi

(Distributed Universal Number Discovery)

- Is utilized to get information from other servers in the cluster
- We then use the returned information in our dialplan to make decisions
- Example: how many calls is a remote box currently handling?
- Another example: Is the Queue() a channel looking for currently available elsewhere in the cluster? If so, we should probably redirect the call



Realtime and UnixODBC

- Realtime allows you to place your flatfile configuration files (e.g. sip.conf) into the database. This allows multiple boxes to read and manipulate the same data as if it were a single system
- Could create a script to pull the data from the database, generate a flatfile and write it to the diskdrive, but this is messy...
- UnixODBC is a database communication abstraction layer that lets Asterisk talk to a single interface, but communicate with multiple databases (MySQL, PostgreSQL, etc...)



func_odbc

- The coolest tool at our disposal – no more AGI
- Read and write data from the database directly from the dialplan
- Manipulate and retrieve the same data your graphical system is using (i.e. one data entry point)
- The power of func_odbc is best described by looking at an example



func_odbc sample

- func_odbc.conf

```
[ROUTE]
readhandle=asterisk
prefix=ROUTE
readsq1=SELECT did_source, did_destination,
did_dest_address_id FROM ast_did_route WHERE
did_number = '${ARG1}'
```

- extensions.conf

```
exten =>
s,n,Set(HASH(ROUTE_ROUTE)=${ROUTE_ROUTE(${DID}}))

exten => s,n,Verbose(1|${HASH(ROUTE_ROUTE|
did_source)})
```



(I'm cheating a bit...)

- `svn co`
http://svncommunity.digium.com/svn/func_odbc/1.4
`/usr/src/func_odbc-1.4`
- `svn co`
http://svncommunity.digium.com/svn/app_stack/1.4
`/usr/src/app_stack-1.4`
- `app_stack` gives us:
 - `GoSub()`
 - `GoSublf()`
 - `Return()`
 - `StackPop()`



Voicemail

- We can centralize our voicemail using IMAP or ODBC storage
- This way, we don't have pieces of voicemail all over the place, and we don't need to rely on a single server for voicemail -- all servers in the cluster can handle voicemail



Interesting features

- These features will be in Asterisk 1.6 (currently trunk) and will make many of the 'hacks' required in 1.4 no longer required:
 - Adaptive CDR for ODBC: easily add new columns to your CDR records
 - func_devstate: customize device state information from the dialplan
 - Access all results from a DUNDi query (DUNDIQUERY and DUNDIRESULT functions)
 - Dynamic weight control in DUNDi



What does the future hold?

- Current efforts include a binary encoded system inside of Asterisk, allowing events to be sent across servers
- The first example of this would be Message Waiting Indication for Voicemail() being distributed across various boxes



What does the future hold?

- Another application would be sharing device state amongst the cluster
- Immediately this would be useful for sharing presence information, and is a prerequisite for building distributed systems that depend on device states; distributed queues for example





Questions and Discussion



Contact Information

Leif Madsen

www.leifmadsen.com

lmadsen@digium.com