

UNIVERSIDAD DE LA REPÚBLICA

FACULTAD DE INGENIERÍA



**Núcleo de Telecomunicaciones Corporativas Basado en
Proyectos SIP de Código Abierto**

Gustavo Vidovich

Gonzalo De Paolis

Pablo Bethencourt

Tutor: Ing. José Joskowicz

**Montevideo, Uruguay
Setiembre 2012**

Agradecimientos

Queremos dejar expreso nuestro agradecimiento al Ing. Juan Martín Vanerio quien aportó al proyecto su valiosa experiencia en los temas en cuestión, así como horas de dedicación a inestimables reuniones con el equipo de proyecto. Su enfoque profesional y sus acertadas críticas son parte fundamental de que este proyecto haya llegado a buen puerto.

Al tutor, Ing. José Joskowicz que nos aportó las respuestas necesarias en el momento indicado para poder continuar con el proyecto.

A la IMM, que nos facilitó dos equipos servidores y un switch administrable.

Índice

CAPÍTULO 1 – INTRODUCCIÓN	10
1.1 – ANTECEDENTES	10
1.2 – MOTIVACIÓN.....	11
1.3 – FORMULACIÓN DEL PROBLEMA	12
1.3.1 – <i>Objetivos Generales</i>	12
1.3.2 – <i>Objetivos Específicos</i>	13
1.4 – HIPÓTESIS.....	14
1.5 – ORIENTACIÓN AL LECTOR	15
CAPÍTULO 2 – EL PROTOCOLO SIP	16
2.1 – GENERALIDADES.....	16
2.2 – CLIENTES SIP	18
2.2.1 – <i>SIP User Agent</i>	18
2.2.2 – <i>Presence Agent</i>	19
2.2.3 – <i>Back-to-Back User Agents – B2BUA</i>	19
2.3 – SERVIDOR SIP	20
2.3.1 – <i>Proxy Server</i>	21
2.3.2 – <i>Redirect Server</i>	22
2.3.3 – <i>Registrar Server</i>	23
2.3.4 – <i>Servlet Server</i>	23
2.4 – TIPOS DE MENSAJES INTERCAMBIADOS	23
2.4.1 – <i>Request</i>	24
2.4.1.1 – INVITE	24
2.4.1.2 – REGISTER	25
2.4.1.3 – BYE.....	27
2.4.1.4 – ACK	27
2.4.1.5 – CANCEL.....	29
2.4.1.6 – OPTIONS	30
2.4.1.7 – SUBSCRIBE	30
2.4.1.8 – NOTIFY.....	32
2.4.1.9 – PUBLISH.....	33
2.4.1.10 – REFER.....	34
2.4.1.11 – MESSAGE	34
2.4.1.12 – INFO.....	35
2.4.1.13 – UPDATE	36
2.4.2 – <i>Tags</i>	37
2.4.3 – <i>Message Body</i>	37
2.4.4 – <i>Response</i>	39
2.4.4.1 – Informational – 1xx	39
2.4.4.2 – Success – 2xx	40
2.4.4.3 – Redirection – 3xx	41
2.4.4.4 – Client Error – 4xx	41
2.4.4.5 – Server Error – 5xx	44

2.4.4.6 – Global Error – 6xx	45
2.4.5 – Ejemplos Gráficos de Intercambio de Mensajes en Diferentes Escenarios	45
2.4.5.1 – Escenario de Registro	46
2.4.5.2 – Escenario de establecimiento de llamada entre dos UA	46
2.4.5.3 – Establecimiento de llamadas entre dos UA mediante Proxy	47
2.5 – ELEMENTOS DEL ENCABEZADO SIP	48
2.5.1 – Campos comunes a Request y Response	49
2.5.2 – Campos exclusivos del mensaje Request	53
2.5.3 – Campos exclusivos del mensaje Response	56
2.6 – ESTABLECIMIENTO DE LAS SESIONES MULTIMEDIA	56
2.6.1 – Protocolo SDP – Session Description Protocol	57
2.6.2 – El modelo Offer-Answer	59
2.6.2.1 – Reglas para generar un Offer	60
2.6.2.2 – Reglas para generar un Answer	60
2.6.2.3 – Reglas para modificar la sesión	61
2.6.2.4 – Llamada en Espera.....	61
2.7 – RTP – REAL-TIME TRANSPORT PROTOCOL	61
2.7.1 – Descripción y Funcionamiento	61
2.7.2 – Códecs.....	63
CAPÍTULO 3 – INVESTIGACIÓN	68
3.1 – REVISIÓN DE SOFTWARE SIP DEL MERCADO ACTUAL Y PRESELECCIÓN	68
3.1.1 – B2BUA: Back to Back User Agents	69
3.1.1.1 – Asterisk.....	69
3.1.1.2 – Elastix	70
3.1.1.3 – FreeSWITCH.....	71
3.1.1.4 – SipXecs.....	72
3.1.1.5 – YATE – Yet Another Telephony Engine	73
3.1.1.6 – CallWeaver	74
3.1.1.7 – GNU SIP Witch	74
3.1.1.8 – SEMS – SIP Express Media Server.....	75
3.1.1.9 – Mobicents.....	75
3.1.1.10 – Cipango.....	76
3.1.1.11 – SailFin	76
3.1.1.12 – Resumen de la Revisión de B2BUA.....	77
3.1.2 – SIP Proxy/SIP Registrar	79
3.1.2.1 – Kamailio.....	79
3.1.2.2 – OpenSIPS	80
3.1.2.3 – MySIPSwitch	82
3.1.2.4 – YATE – Yet Another Telephony Engine	82
3.1.2.5 – YXA	83
3.1.2.6 – Resumen de la Revisión de SIP Proxy/Registrar	83
3.1.3 – Softphones	84
3.1.3.1 – Blink.....	84
3.1.3.2 – Ekiga	84
3.1.3.3 – Empathy	85
3.1.3.4 – Jitsi.....	85
3.1.3.5 – KPhone (KDE).....	85
3.1.4.6 – Linphone.....	86
3.1.3.7 – Qutecom.....	86

3.1.3.8 – SFLPhone	86
3.1.3.9 – Twinkle	87
3.1.3.9 – YATE Client	87
3.1.3.10 – X-Lite	88
3.1.3.11 – Resumen de la Revisión de Softphones.....	88
3.2 ELECCIÓN DEL SOFTWARE A UTILIZAR.....	90
3.2.1 – Asterisk o FreeSWITCH.....	91
3.2.4 – Kamailio u OpenSIPS.....	92
3.3 HERRAMIENTA GENERADORA DE TRÁFICO.....	93
CAPÍTULO 4 – NÚCLEO DE SEÑALIZACIÓN	95
4.1 – LA SOFT-PBX	95
4.2 – EL SIP PROXY	98
4.2.1 – B2BUA vs. SIP Proxy	99
4.2.2 –Capacidad de manejo de señalización.....	102
4.2.2.1 – Registros. Optimización del sistema	102
4.2.2.2 – Llamadas. Reglas de dimensionamiento.	109
4.2.3 – Aplicación de la Teoría de Teletráfico	116
4.3 – EL DNS	121
4.4 – ARQUITECTURA PARA EL NÚCLEO DE SEÑALIZACIÓN	122
CAPÍTULO 5 – FUNCIONALIDADES CORPORATIVAS	123
5.1 – INTRODUCCIÓN	123
5.2 – MUSIC ON HOLD (MOH)	124
5.2.1 – Descripción del Servicio.....	124
5.2.2 – Casos de Uso.....	124
5.2.3 – FreeSWITCH Como Proveedor de Música en Espera.....	125
5.2.4 – Dimensionamiento.....	126
5.2.5 – Pruebas de Carga Unitaria.....	128
5.2.6 – Integración al Core de Señalización.....	141
5.3 – VOICEMAIL.....	149
5.3.1 – Descripción del Servicio.....	149
5.3.2 – Casos de Uso.....	149
5.3.3 – FreeSWITCH Como Proveedor de Casillas de Voicemail.....	151
5.3.4 – Dimensionamiento.....	153
5.3.5 – Pruebas de Carga.....	153
5.3.6 – Diagrama de Flujo de Casos de Uso.....	160
5.4 – PUENTE DE CONFERENCIAS.....	165
5.4.1 – Modelos de Conferencias.....	165
5.4.2 – FreeSWITCH Como Servidor Centralizado de Conferencias Integrado al Core de Señalización	167
5.4.3 – Dimensionamiento.....	170
5.4.5 – Pruebas de carga unitarias	171
5.5 – TRANSFERENCIAS.....	175
5.5.1 – Descripción del Servicio.....	175
5.5.2 – Requerimientos.....	176
5.5.3 – Uso del Método REFER en Transferencias	176
5.5.1.2 – Transferencia Desatendida.....	178

5.5.1.1 – Transferencia Atendida	179
5.5.3 – Implementación en el Core	180
5.5.4 – Flujos Básicos	181
5.5.4.1 – Transferencia Desatendida o Ciega	181
5.5.4.2 – Transferencia Atendida	182
5.6 – SERVICIO DE PRESENCIA Y MENSAJERÍA INSTANTÁNEA	183
5.6.1 – Descripción del Servicio de Presencia	183
5.6.1.1 – Definiciones a Tener en Cuenta	184
5.6.1.2 – Funcionamiento	184
5.6.2 – Descripción del Servicio de Mensajería Instantánea	188
5.6.2.1 – Operativa de la Mensajería Instantánea	188
5.6.2.2 – Definición del Método MESSAGE	190
5.6.3 – Implementación en el Core de Señalización	192
5.7 – GRUPOS Y PERMISOS	194
5.7.1 – Descripción de la Funcionalidad	194
5.7.2 – Implementación en Core de Señalización	196
5.8 – ACCOUNTING	197
5.8.1 – Descripción del Servicio	197
CAPÍTULO 6 – ALTA DISPONIBILIDAD DEL NÚCLEO DE SEÑALIZACIÓN	199
6.1 – SIP PROXY HA	199
6.1.1 – Esquemas de Alta Disponibilidad	199
6.1.2 – Balanceo de Carga y Failover	209
6.1.2.1 – Balanceo Mediante Hardware Dedicado	210
6.1.2.2 – El DNS Como Balanceador	213
6.1.2.3 – PingX	216
6.1.3 – DNS Propio del Núcleo de Señalización	217
6.2 – PINGX HA	219
6.3 – BASE DE DATOS HA	221
6.4 – DNS HA	224
6.5 – ARQUITECTURA DEL NÚCLEO HA	227
CAPÍTULO 7 – ALTA DISPONIBILIDAD DE LAS FUNCIONALIDADES	231
7.1 – MUSIC ON HOLD	232
7.2 – VOICEMAIL	236
7.2.1 – Base de Datos Compartida y NFS	240
7.2.2 – Suscripciones (MWI)	242
7.3 – PUENTE DE CONFERENCIAS	244
7.3.1 – Failover en Caliente de Salas de Conferencias	248
7.4 – VARIANTE AL MODULO DE DESPACHO	253
CAPÍTULO 8 – ARQUITECTURA FINAL Y ADMINISTRACIÓN	254
8.1 – ARQUITECTURA FINAL. OBELIX.	254
8.2 – ADMINISTRACIÓN DEL SISTEMA	256
8.2.1 – Administración de Usuarios	256
8.2.1.1 – Suscripción	257
8.2.1.2 – Manejo de Grupos de Usuarios	257
8.2.1.3 – Funcionalidades	258

8.2.2 – Administración de Servidores	259
8.2.2.1 – Configuración a Nivel de Red.....	259
8.2.2.2 – Cuidados en la Administración del Sistema	260
CONCLUSIONES	262
REFERENCIAS	264
ÍNDICE DE IMÁGENES	268
ÍNDICE DE TABLAS	271
ANEXO 1 – INSTALACIÓN Y CONFIGURACIÓN: KAMAILIO, MYSQL, BIND.....	272
1 – INTRODUCCIÓN A LA HERRAMIENTA: KAMAILIO.	272
1.1 – Descripción.....	272
1.2 – Introducción al script de configuración de Kamailio.	273
1.3 – Instalación	274
1.4 – Archivo de configuración de los SIP Proxy del núcleo de señalización.	279
1.4.1 – El código del núcleo de Obelix.....	279
1.4.2 – El problema del MOH	317
1.5 – Archivo de configuración de los servidores de despacho.	319
1.6 – Archivo de configuración de los servidores del modulo PingX.....	327
1.6.1 – PingX Master	328
1.6.2 – PingX Slave	331
1.7 – Soporte de los desarrolladores para con Obelix.	334
2 – MYSQL. REPLICACIÓN MASTER-MASTER.	336
3 – BIND. EL DNS DEL SISTEMA.	342
ANEXO 2 – FREESWITCH.....	349
2.1 – PERFORMANCE FREESWITCH CENTOS 6.1 VS CENTOS 5.8	349
2.2 – INSTALACIÓN DE FREESWITCH DESDE GIT	353
2.3 – CONFIGURANDO FREESWITCH COMO “PROXY”	354
2.4 – CONFIGURACIÓN DE FREESWITCH COMO SERVIDOR DE MUSIC ON HOLD	355
2.5 – GUÍA PARA CONFIGURACIÓN DE FREESWITCH COMO ELEMENTO CONSTITUTIVO DE UN BANCO DE VOICEMAIL	362
2.5.1 – NFS (Network File System).....	365
2.5.2 – Configuración directorios compartidos por NFS (banco de Voicemail).....	365
2.6 – CONFIGURACIÓN PARA EL ENVÍO DE NOTIFICACIONES DE MENSAJES DE VOICEMAIL A CORREO ELECTRÓNICO	367
2.6.1 – Parámetros a establecer en la configuración de FreeSWITCH.....	367
a) extensiones	367
b) en la configuración de freeswitch	368
2.7 – CONFIGURACIÓN XML DE FREESWITCH COMO SERVIDOR DE VOICEMAIL.....	369
2.8 – CREACIÓN DE CASILLAS DE VOICEMAIL MEDIANTE EL “USER DIRECTORY” DE FREESWITCH.....	372
2.9 PUENTES DE CONFERENCIAS.....	374
2.10 PUENTES DE CONFERENCIAS. SOLUCIÓN CON FAILOVER “EN CALIENTE” DE SESIONES.	375
2.10.1 Script watchcat.sh	378
2.10.2 Script watchcat2.sh	379
2.10.3 Configuración FreeSWITCH Servidor de Conferencias 1	380
2.10.4 Configuración FreeSWITCH Servidor de Conferencias 2	385

ANEXO 3 – SIPP	393
1 – DESCARGA E INSTALACIÓN	393
1.1 – <i>CentOS</i>	393
1.2 – <i>Ubuntu</i>	395
2 – FUNCIONAMIENTO.....	395
2.1 – <i>Atributos</i>	396
3 – ESCENARIOS	398
4 – ESCENARIOS DE PRUEBA UTILIZADOS.....	399
4.1 – <i>Registración</i>	399
4.2 – <i>Llamadas Planas</i>	401
4.2.1 – Llamador.....	401
4.2.1 – Llamado	403
4.3 – <i>Music On Hold (MOH)</i>	406
4.4 – <i>Voicemail</i>	408
4.5 – <i>Conferencia</i>	411
4.5.1 – Un Usuario Accediendo al Servidor de Conferencia	411
4.5.2 – Prueba de Generación de Oyentes de Conferencia Establecida.....	414
GLOSARIO	416

Capítulo 1 – Introducción

1.1 – Antecedentes

La telefonía clásica ha cambiado a lo largo de su historia, ha incorporado nuevas funcionalidades que la han puesto como el camino de comunicación por excelencia. Confiabilidad, despliegue y agregación de servicios son características definitorias para que a nivel mundial se haya adoptado de forma fáctica en el transcurso de la segunda mitad del siglo XX.

Pero es en este periodo que se comienzan a ver impactadas por otros desarrollos, como las redes de datos que incrementan su capacidad de comunicación e incluyendo nuevas funcionalidades para poder incorporar nuevos medios y sistemas de transferencia de información (no solo voz).

En los últimos 30 años la red telefónica conocida se re-piensa para aprovechar los nuevos avances en materia de redes de datos. Es por eso que la potencialidad del protocolo SIP se ve ajustada a las necesidades de la red telefónica y puede volcar el conocimiento generado y consolidado para dar un nuevo impulso, no solo tecnológico sino también de despliegue, ajustado a los costos y a las nuevas realidades mundiales.

Esto conlleva a relacionar los diferentes desarrollos existentes en materia de PBX (Private Branch Exchange), partiendo de los desarrollos de la telefonía clásica, para poder vincularlos a las redes de datos. Es así que se acuña el concepto de SoftPBX, atado directamente a dos comunidades o dos encares que afectan la forma de poder lograr el cometido, estos son el mundo Open Source o de Código Abierto y el mundo de código cerrado. Hoy por hoy desarrollos como Asterisk o FreeSWITCH, cuentan con una comunidad que en base a los requerimientos de los usuarios incrementa y mejora las funcionalidades de los diversos sistemas, ya sea para dar soporte a call centers, o dar soluciones de telefonía donde una central clásica sea inviable de instalar. Con esto último nos referimos a casos en que una PBX clásica no sea la opción adecuada ya sea por costos elevados o por impedimentos del punto de vista físico. Una softPBX permite reutilizar una estructura de red ya instalada e incluso prescindir de las terminales clásicas ya que con solo tener instalado en el PC un softphone se obtiene la extensión deseada.

1.2 – Motivación

La red telefónica tal cual se conoce ha alcanzado su máximo en materia de servicios agregados. Si bien su utilización está más que asegurada en el mercado, la aparición de protocolos como SIP ofrece nuevas funcionalidades o ideas para pensar la telefonía en una dirección que pretende ser bastante atractiva.

Es por esto que muchas empresas y entes han volcado sus esfuerzos en establecer redes de VoIP para sus sistemas de comunicación y así contar con un sistema que provea funcionalidades extra permitiendo una integración más profunda en materia de comunicaciones corporativas.

Incorporando en diversos sitios equipos que puedan proveer servicio de VoIP (SIP) y utilizando el cableado de red disponible, se puede establecer un sistema telefónico que vincule a los diferentes usuarios dentro de una misma oficina y establecer comunicaciones entre oficinas remotas pertenecientes a una misma empresa, evitando los altos costes en el uso de líneas ya establecidas y aprovechando las flexibilidades de la telefonía sobre IP.

Los problemas no son ajenos a esta tecnología y es por eso que al incrementar la cantidad de usuarios de voz sobre IP, es necesario pensar soluciones que puedan proveer servicios con alta disponibilidad y eficiencia sin perder calidad ni generar áreas que queden sin cobertura.

A partir de la arquitectura cliente-servidor se trata de pensar en una arquitectura Core-Servidor-Cliente. Este hecho apunta a utilizar proyectos que intentan resolver la masividad de la utilización de VoIP, en base a señalización SIP con diferentes tipos y concepciones de servidores PBX que a su vez vinculan a los diferentes usuarios y terminales de la red telefónica.

Despegándose del concepto de redes best effort, se trata de alcanzar un servicio similar al servicio telefónico clásico y es una de las razones por la que se establece el concepto de alta disponibilidad. Concepto que exige una implementación sólida en materia de equipos que soporten la carga suficiente de los establecimientos de llamadas en un lapso de tiempo por parte de los usuarios y que en caso de fallas en el core, no generen una pérdida de servicio total.

Por otra parte, para soportar una carga considerable de establecimientos, se precisa un balanceo de carga que no altere la calidad de estas llamadas ni provoque la caída parcial o total de servicio que se está proveyendo.

Centrándose en los servicios agregados de las SoftPBX es necesario que las funcionalidades básicas estén presentes y esto conlleva al concepto de “features” que aseguren que el usuario pueda utilizar en el servicio características tales como música en espera, mensajes al correo de voz que sean enviados a su casilla de correo (servicio de voicemail), transferencias de llamadas, y servicios de puente de conferencia que emulen salas donde varios usuarios puedan estar en conferencia al mismo tiempo.

El presente proyecto, al que denominamos Obelix, se centra en diseñar y configurar una arquitectura que permita cumplir con los requerimientos establecidos como criterios de éxito, ajustándose por un lado a lo estipulado por el cliente al momento del planteo del proyecto y por otro a los intereses académicos. Se pretende para esto utilizar una configuración de herramientas que utilicen como protocolo de señalización al protocolo SIP y sean basadas en proyectos de Open Source.

1.3 – Formulación del Problema

A partir de las necesidades expresadas por el cliente, se determinaron objetivos a cumplir para alcanzar la solución deseada. Se fijaron objetivos generales para trabajar en el marco de ciertos requerimientos operativos, así como también se definieron objetivos específicos para llegar a la solución pedida por el cliente.

1.3.1 – Objetivos Generales

El cliente que presentó la idea de proyecto cuenta con una estructura de servidores que corren una SoftPBX (en concreto Asterisk) en las diversas dependencias que tiene en la ciudad. Dada la necesidad expresada por el cliente de incrementar la capacidad en la red de telefonía de VoIP, precisa una herramienta capaz de vincularlos y, más importante aún, soportar el volumen de usuarios en el sistema demandado. La herramienta o herramientas seleccionadas que otorguen al sistema la capacidad requerida deberán ser obtenidas a partir de proyectos que estén basados en Código Abierto para reducir los costos de licenciamiento que otras soluciones exijan. Todo el núcleo, constituido por las diferentes piezas que provean el servicio de telefonía de VoIP deberá cumplir que sean Altamente Disponibles, específicamente se apunta a que en caso de falla de algún punto del sistema, el usuario final, no vea afectado el servicio. Finalmente, todo el sistema deberá soportar un perfil de tráfico de 10.000

usuarios traficando cada uno de ellos 0.15 Erlang, esta restricción es impuesta ya que la alta carga de usuarios exigirá dimensionar equipos para que se otorgue un servicio mínimo a la cantidad establecida.

1.3.2 – Objetivos Específicos

Para formalizar de forma más precisa se establecen objetivos específicos, estos serán tenidos en cuenta en el entendido de fortalecer los objetivos generales y así llegar a la solución completa.

- Se establece como caso de éxito que el sistema soporte una capacidad de usuarios que alcance los 10.000 a 0.15 Erlang por usuario como criterio de tráfico, siendo de especial interés del proyecto y del cliente cuáles son los puntos a atacar para llegar a este tipo de valores, partiendo de una investigación adecuada de las diferentes opciones existentes para poder alcanzar este objetivo.
- Obtener las herramientas que cumplan con lo requerido, se procede a realizar las pruebas tanto funcionales como de carga para evaluar si lo seleccionado cumple con lo establecido a nivel teórico. De esta forma se establecerá una metodología de dimensionamiento a aplicar en diferentes puntos del proyecto.
- Partiendo de los datos tanto funcionales como de carga, se procederá a evaluar las diferentes arquitecturas que se adecuen al cumplimiento de los objetivos definidos.
- Como se cita en párrafos previos, se buscará una arquitectura que cumpla los principios de alta disponibilidad (HA) y balanceo de carga. Este es uno de los objetivos resaltados tanto por parte del cliente como por interés del proyecto para alcanzar un sistema telefónico estable y que tenga una protección en el caso de la existencia de fallas, evaluando la posibilidad de la redundancia en varios puntos de la arquitectura obtenida.
- Integrar a la solución propuesta funcionalidades corporativas, dentro de las que se detallan servicios de música en espera, transferencia de llamadas, correo de voz y servicio de conferencias.

1.4 – Hipótesis

La hipótesis a tener en cuenta es un ancho de banda asegurado por parte de la red donde se implementará el sistema obtenido a partir de la investigación realizada, según los requerimientos del cliente. La red no dispone de salida a la PSTN. Es una red de telefonía IP donde la integración con la red de telefonía clásica no está prevista, quedando por fuera del alcance del proyecto.

1.5 – Orientación al lector

En el Capítulo 2 de esta documentación se encontrará una introducción al protocolo SIP y en particular se describirán todos los métodos y parámetros relevantes para la comprensión del documento.

Dentro del Capítulo 3 se sumergirá al lector en los resultados de una revisión de las herramientas existentes actualmente de código abierto que usen como base el protocolo SIP.

El Capítulo 4 presenta análisis y pruebas de carga a distintas configuraciones para el núcleo de señalización con el objetivo de presentar una arquitectura que logre soportar el máximo posible de usuarios en el sistema.

El quinto capítulo de este documento presenta las funcionalidades de la arquitectura propuesta al momento y soluciones para añadir funcionalidades corporativas de telefonía al núcleo de señalización. Se presentan métodos para realizar pruebas de carga y realizar dimensionamiento de servidores.

El Capítulo 6 busca introducir al lector a los conceptos de alta disponibilidad del sistema y proponer un esquema para lograr la misma en el núcleo de señalización.

El séptimo capítulo describe como dar alta disponibilidad a las funcionalidades logradas en el sistema a partir del Capítulo 5, logrando completar una arquitectura de alta disponibilidad para todo el sistema.

El Capítulo 8 y último presenta la arquitectura completa, como se realiza la administración del sistema (ya sea a nivel de usuarios como de servidores). Finalmente se expondrán las conclusiones obtenidas a partir de lo trabajado.

En los anexos se pueden encontrar todas las configuraciones de todos los elementos del sistema final, así como también explicaciones específicas del funcionamiento de cada elemento.

Capítulo 2 – El Protocolo SIP

El objetivo de este capítulo es incorporar el concepto y explorar el funcionamiento del protocolo de inicio de sesión definido en la RFC-3261 [1]. Se ahondará en cuáles son las generalidades del mismo y los tipos de mensajes que se despliegan al momento de establecer las sesiones, los encabezados y la utilidad de estos para aplicaciones posteriores.

Se explicará el concepto de SIP *User Agent* (UA) y cuál es la operativa en la negociación del establecimiento de la sesión. Se expondrán los mensajes intercambiados en diferentes escenarios al momento del establecimiento y qué indican las diferentes alertas que nos envía el sistema. También se hará mención a los diferentes campos del encabezado que se utilizarán a lo largo del desarrollo del proyecto.

Finalmente se explicará el funcionamiento del protocolo RTP, encargado de transportar el audio luego de finalizada la negociación de parámetros entre los actores de la sesión establecida.

2.1 – Generalidades

SIP, de sus siglas en inglés *Session Initiation Protocol* (protocolo de inicio de sesión) es el protocolo estandarizado por la IETF en la RFC-3261. Esta recomendación, además de definir el protocolo describe, entre otros, los procesos necesarios para el establecimiento, modificación y terminación de sesiones multimedia entre dos extremos, además de procedimientos para invitar a otros usuarios a formar parte de conferencias y realizar las llamadas telefónicas de voz sobre IP en la red.

Este protocolo, perteneciente a la capa de aplicación del modelo OSI, gestiona las funcionalidades necesarias para el establecimiento de la comunicación entre dos terminales, e indica al audio como señalizarlo.

Las funcionalidades más importantes del protocolo SIP se pueden definir en el marco de dos *User Agent* que intentan establecer una sesión de *media*¹ entre ellos. Las principales funciones de señalización del protocolo son:

- Ubicación del terminal de destino
- Consulta la disponibilidad del destino para establecer la sesión
- Intercambiar información de *media* para permitir el establecimiento de la sesión
- Modificación de la sesión de *media* existente
- Liberar la sesión establecida entre los dos terminales

Otra funcionalidad adjudicada al protocolo SIP es la de dar servicios de presencia, lo que implica enviar mensajes bajo un servicio de mensajería instantánea. Las funciones incluidas para este servicio son:

- Publicación y envío de información de presencia
- Consultas de información de presencia
- Servicios de notificación de presencia
- Transporte de mensajes instantáneos (*chat*)

Los extremos que intervienen en la comunicación y que se encargan de anunciar los parámetros a negociar para el establecimiento de la misma se denominan *User Agents* (de ahora en mas, UA). Uno de los objetivos de SIP es establecer sesiones entre los UA. El UA toma control sobre el usuario con el fin de realizar las negociaciones correspondientes para el establecimiento y liberación de sesiones *media* por parte del usuario con otros UA de la red. En consecuencia, el UA es el responsable de iniciar y mantener la sesión activa.

Dada la concepción del UA, es necesario definir como oficia tanto de UAC (*User Agent Client*) como de UAS (*User Agent Server*), definiendo las operativas de forma más granular. En concreto, el UAC trabaja a nivel de las *Requests* que origina y el UAS de las *Responses* que este envía a los *Requests* del UAC. Es por esta razón que un UA actuara en ambos modos en el establecimiento de una sesión.

Las definiciones y desarrollos de las siguientes secciones se tomaron en su mayoría de la referencia [2], las referencias de otros textos y fuentes se referencian en los párrafos correspondientes.

¹ Media: audio, video, etc.

2.2 – Clientes SIP

Un cliente SIP es una entidad de red que envía *Request* de SIP y recibe *Responses* de SIP. Estos clientes podrán o no interactuar con el humano de forma directa. Los *User Agent* y los *Proxy* son clientes SIP. Las características específicas de éstos se explicarán en los siguientes subcapítulos.

2.2.1 – SIP User Agent

Denominamos SIP *User Agent*, *User Agent* o por su abreviación UA, a un dispositivo o aplicación que tenga habilitado el protocolo SIP. El propósito de SIP es establecer sesiones entre los UA. El UA toma control del usuario y actúa por parte del mismo para establecer y liberar sesiones con otros UA. En la mayoría de los casos el usuario será un usuario final pero en casos específicos podrá ser un *Gateway*.

Los *User Agent* serán encargados de mantener el estado de las llamadas que inician o participan. Dentro de este estado se especifican parámetros clave como los **tags**, **Call-ID** y **CSeq** tanto remoto como local, como la información de la ruta establecida y cualquier otro estado necesario de la sesión de *media*. Esta información es necesaria para guardar información del dialogo establecido y confiabilidad del mismo. El **CSeq** oficiará para distinguir entre nuevas *Request* o retransmisiones de *Request* antiguas. Como modo de protección de pérdida de mensajes al momento de finalizar una llamada, el UA mantendrá durante 32 segundos el estado de la llamada.

Las respuestas provenientes de diálogos desconocidos sean descartadas de forma silenciosa por parte del UA, necesarias para mantener una red segura. De lo contrario agentes maliciosos podrán obtener información del UA enviando falsos *Request* o *Responses*. En casos que existan *Request* que no sean soportadas por el UA, se enviarán mensajes correspondientes para anunciar la falta de dicho soporte.

Cada UA podrá diferenciarse como dos subclientes denominados *User Agent Client* (UAC) y *User Agent Server* (UAS), del punto de vista funcional la UAC inicia los *Request* mientras que la UAS genera las *Responses*. Durante la sesión establecida los UA operaran en ambos modos.

El manejo y soporte de *Session Description Protocol* (SDP) por parte del UA es necesario para lograr establecer parámetros de la sesión de *media*.

Es requerido por parte del UA que comprenda las extensiones listadas en el encabezado **Require** recibidas en una *Request*, los encabezados desconocidos serán ignorados por el UA y deberá anunciar las capacidades y funcionalidades en las diferentes *Requests* que éste envíe, esto permitirá que otros UA aprendan sin tener que realizar consultas explícitas, a saber, se listaran en el encabezado **Allow** los diferentes métodos soportados por el UA, las extensiones SIP serán listadas en el encabezado **Supported** y los tipos de cuerpo de mensaje en el encabezado **Accept**.

Para registrarse utilizaran, en caso de estar presente, el servidor proxy definido en el dominio en el cual se encuentran.

2.2.2 – Presence Agent

Un agente de presencia (PA) es un SIP UA capaz de recibir suscripciones a *Request* y generar notificaciones de estado definidas en los eventos SIP, soporta paquetes de eventos de presencia, responde además a **SUBSCRIBE** y envía **NOTIFY**. Recolecta información de presencia de un número de dispositivos, esta información proviene de un dispositivo SIP registrándose o de un dispositivo SIP que publique información de presencia.

Como operativa básica, el PA primero se autentica un pedido de suscripción, si la autenticación es efectiva, establece un diálogo y envía las notificaciones a lo largo de ese diálogo.

2.2.3 – Back-to-Back User Agents – B2BUA

Los *Back-to-Back User Agents* (abreviado B2BUA) son un tipo de UA SIP que recibe SIP *Requests* los cuales son reformulados para ser enviados como *Request*. Desde el punto de vista funcional se puede afirmar que un B2BUA actúa de forma aproximada a un servidor proxy, con el detalle que no cumple las reglas de ruteo establecidas por este tipo de servidores. Un ejemplo de utilización de un B2BUA es cuando una UA que participa en un diálogo con otro UA desconoce la URI, dirección IP u otro tipo de información del otro UA. El B2BUA actúa como un servidor de anonimato en el cual se encarga de que un UA se comunique con la otra, reformulando los campos **From**, **Via**, **Contact** y **Call-ID**, además del SDP donde se describe el tipo de sesión de media, incluso remueve información detallada para que ninguna de las partes sepa información de la otra. La

respuesta obtenida también cambiará el encabezado **Contact** y el tipo de media negociado en el SDP de la extensión que originó la llamada, siendo que el SDP modificado apunta al B2BUA que reenviará los paquetes RTP de la parte llamada a la parte que origino la llamada y en sentido contrario. De esta forma ninguna de las partes conoce información de identificación de la otra parte durante el establecimiento de la sesión.

Un B2BUA es un agente del tipo *stateful*², por lo que se transforma en un único punto de fallas en la red generando una reducción de la confiabilidad de las sesiones SIP en Internet. Una breve explicación del porqué de esta caída de performance es que a medida que se incrementa la cantidad de solicitudes se incrementa la latencia y la probabilidad de pérdida de paquetes reduciendo la calidad de la sesión de media establecida. La posibilidad de minimizar este tipo de comportamiento es distribuyendo la cantidad de estos servidores, pero la selección del mejor B2BUA para una sesión en concreto puede tornarse dificultosa ya que la dirección IP de origen y de destino son desconocidas hasta que se logra establecer la sesión (respuesta de un **200 OK**).

Otra funcionalidad que provee el B2BUA son los Puentes de Conferencia, el servidor actuará de *mixer* de las sesiones de media establecidas para proveer este tipo de funcionalidad.

2.3 – Servidor SIP

Es una aplicación que acepta SIP *Request* y responde a estas últimas, se debe tener especial cuidado de no confundir con el concepto de UA o el concepto de cliente – servidor del protocolo SIP. Se entiende por SIP server a una entidad lógica, que incluirá una cantidad de servidores de diferentes tipos según las condiciones de trabajo. Como los servidores proveen servicios y funcionalidades a los UA deben de soportar transporte del tipo TCP como UDP.

² Se denomina *stateful* cuando se preserva el estado de las transacciones realizadas entre dos usuarios.

2.3.1 – Proxy Server

Un servidor Proxy recibe peticiones SIP provenientes de un UA a otro UA y actúa en representación del UA para reenviar o responder a esas peticiones efectuadas. En un caso similar a la red IP en la capa de transporte, un servidor SIP Proxy se encarga de reenviar los mensajes SIP en la capa de aplicación. Por definición un servidor SIP Proxy no es un B2BUA ya que no tiene permitido modificar las peticiones y las respuestas de acuerdo a reglas estrictas establecidas en la RFC-3261 que se encargan de preservar la funcionalidad punta a punta del protocolo de señalización SIP, mientras permite al servidor efectuar funciones y servicios para los UA.

Este tipo de servidores tiene acceso a una base de datos de ubicaciones o servicio de ubicaciones para ayudar a procesar las *Request* a las que se ve sometido, de esta forma determina el próximo salto. Las bases de datos contienen registros SIP, información de presencia u otro tipo de información que establezca donde está ubicado el usuario.

Un proxy no precisa entender una *Request* SIP para poder reenviarla y no modificará el orden de los encabezados ni tampoco borrará o modificará el contenido de los mismos.

Las diferencias entre un UA y un Proxy son sustanciales y estas son:

- Un proxy no genera *Request*, solo responde *Request* enviadas desde el UA (**CANCEL** y **ACK** son excepciones a este funcionamiento)
- No tiene capacidad de procesar media
- No se mapean cuerpos de mensajes, solo se trabaja en base a los encabezados de SIP

Una vez que el proxy establece las ubicaciones y registro de las diferentes UA, las futuras *Request* serán enviadas entre los UA, a menos que se estampe por parte del proxy un encabezado **Record-Route**.

Se definen dos tipos de servidores proxy, *stateless* y *stateful*. El proxy *stateless* procesa cada SIP *Request* o cada respuesta basada solamente en el contenido de los mensajes, una vez que el mensaje haya sido mapeado, procesado y reenviado o respondido, ninguna información del mensaje es mantenida (se borra incluso la información del dialogo establecido), tampoco retransmite mensajes y no usa *timers*. Es obligatoria la utilización del encabezado **Max-Forwards** para implementar la detección de *loops*.

Un servidor proxy del tipo *stateful* mantiene registro de los *Request* recibidos y *Response* enviados y utiliza esa información para procesar posteriores mensajes de *Request* y *Response*. El servidor de tipo *stateful* no inicia un *timer* al momento de reenviar una *Request*, si no hay *Response* dentro de un periodo de tiempo adecuado el servidor proxy retransmitirá la *Request*, relevando a la UA de efectuar esta tarea, el servidor de este tipo requiere, además, la autenticación del UA.

El tipo más común de servidor proxy es el de transacciones *stateful*, este mantiene el estado de la transacción pero solo por el tiempo en que está pendiente la *Request*. El ejemplo es cuando recibe un **INVITE** y queda a la espera hasta que recibe un **200 OK** o una respuesta de falla (**404 Not Found**), luego de esto, se destruirá la información de estado, permitiendo al proxy efectuar una búsqueda mucho más eficiente y minimizando la cantidad de datos almacenados.

Usualmente, un proxy *stateful* envía *Responses* del tipo **100 Trying** frente a mensajes de **INVITE**, pero un proxy *stateless* nunca envía **100 Trying**. Siendo que un **100 Trying** nunca es reenviado ya que es una respuesta de un solo salto. De manejar transporte TCP, el modo será *stateful*, ya que el UA asumirá un transporte confiable y confiará en el proxy para posibles retransmisiones del tráfico UDP en el mismo camino de señalización.

La forma de controlar la cantidad de saltos existentes entre proxy está dada por el encabezado **Max-Forwards**, que se decrementa en cada proxy por la que pasa la *Request*, al llegar a cero el proxy descarta el mensaje y envía un mensaje de **483 Too Many Hops** a quien originó el mensaje.

2.3.2 – Redirect Server

Es un tipo de servidor SIP que responde pero no reenvía *Requests*. Opera de igual modo que un servidor proxy, hace consultas a la base de datos o un servicio de ubicación para obtener a un usuario. La información de ubicación es enviada al llamador en un mensaje perteneciente a la categoría **Redirection (3xx)**, que luego de un **ACK** da por concluida la transacción.

2.3.3 – Registrar Server

Un Registrar Server, o mejor conocido como servidor de Registro, acepta *Request* del tipo **REGISTER** y pondrá la información que recibe en aquellas *Request* en el servicio de ubicación para el dominio en el que está ubicado. Cualquier otro tipo de *Request* al servidor que no sea del tipo **REGISTER** recibirá por respuesta un **501 Not Implemented**.

2.3.4 – Servlet Server

Antes de definir este tipo de servidores nos centraremos en el concepto de Servlet. Un Servlet es un pequeño programa de Java que corre en un servidor, estos reciben y responden a las consultas efectuadas por clientes, usualmente a través de HTTP, pero también a través de otros protocolos.

Un SIP Servlet es un Servlet que ejecuta señalización SIP, es decir, implementa una SIP UA. Estos interactúan con otros SIP UA en base consultas efectuadas en SIP y respondiendo vía SIP.

Formalmente, un SIP Servlet cumple la especificación SIP Servlet Specification desarrollada a través de la *Java Community Process* provee un modelo estándar de API de Java para dar servicios basados en SIP. SIP Servlets da soluciones a la capacidad de desarrollo de las aplicaciones de internet, ya que los desarrolladores de HTTP Servlets pueden adaptarse fácilmente a los SIP Servlets [3].

2.4 – Tipos de Mensajes Intercambiados

En este apartado se explicarán los diferentes mensajes que se cursan dependiendo si se efectúa una *Request* o una *Response*. Al final de esta sección se expondrán de forma gráfica el intercambio de mensajes en escenarios de registro, establecimiento de llamada, y establecimiento de llamada mediante un proxy.

2.4.1 – Request

Los tipos de *Request* de SIP denominados métodos (*method*) son descriptos originalmente en la RFC-3261. Los métodos en SIP se pueden considerar como “verbos”, debido a que cada método se mapea a una acción específica por parte de un UA o servidor. La RFC-3261 especifica seis métodos (**INVITE**, **REGISTER**, **BYE**, **ACK**, **CANCEL** y **OPTIONS**) originales. En posteriores RFC se especifican otros métodos para funciones mas específicas (**REFER**, **SUSCRIBE**, **NOTIFY**, **PUBLISH**, **MESSAGE**, **UPDATE**, **INFO** y **PRACK**). Son nombres que se escriben en mayúscula para diferenciar del resto del encabezado que en varios casos utilizan tanto minúsculas como mayúsculas.

2.4.1.1 – INVITE

Método utilizado para establecer sesiones media entre dos UA. El reconocimiento de este método es enviado mediante un método **ACK**. Usualmente el mensaje que es enviado con un método de este tipo contiene información sobre el tipo de media que ese usuario utiliza y también puede contener información de sesión. Si el mensaje no contiene información de *media* el reconocimiento de la UAC, conformado por el método **ACK**, contiene la información de media correspondiente a la UAC. En el caso que la información de media no sea aceptable, se responderá con un mensaje con método **BYE**, cancelando el establecimiento de la sesión. Se considera una sesión de media establecida luego del intercambio de mensajes con los métodos **INVITE**, **ACK** y **200 OK** entre la UAS y la UAC. La sesión se mantendrá activa hasta que alguna de las dos partes envíe un mensaje con el método **BYE**.

La UAC que origina un **INVITE** para establecer un dialogo crea una única **Call-ID** que será utilizada a lo largo de toda la sesión o duración de la llamada, además se inicializará el numero de secuencia **CSeq** (no precisamente uno, pero sí que sea un entero) y que será incrementado en cada *Request* enviado con el mismo **Call-ID**. Los campos **To** y **From** se utilizan para la dirección remota y local de las puntas de la comunicación, por eso se incluye un **From tag** y la UAS incluirá un **To tag** en sus respuestas. En combinación, **To tag**, **From tag** y **Call-ID** serán identificadores únicos del dialogo.

Se define como re-**INVITE** cuando un **INVITE** es reenviado con el mismo **Call-ID** conteniendo los mismos **To tag** y **From tag**, siendo un mensaje de utilidad para cambiar características de la sesión establecida o actualizar el

estado del diálogo. Al incrementar el campo **CSeq** la UAS podrá distinguir el **re-INVITE** de una retransmisión del **INVITE** original. Un **re-INVITE** puede ser rechazado sin que la sesión se vea afectada, pero no puede ser enviado por una UAC hasta que exista una respuesta final al primer **INVITE**.

Un campo de **Expires** en el mensaje de **INVITE**, indica a la UAS por cuánto tiempo la *Request* es válida. Cuando un UAS envía un mensaje de **INVITE** y este no es respondido por el lapso de tiempo establecido en el campo **Expires**, se descarta la *Request*, por el contrario si se recibe la *Response* al **INVITE**, el campo **Expires** no tiene utilidad. Cabe aclarar que quien finaliza la sesión *media* es establecido en el campo **Session-Expires**, que puede ser actualizado en una sesión establecida mediante un **re-INVITE** o un **UPDATE**.

Existen campos obligatorios en el mensaje **INVITE**, que son: **Via**, **To**, **From**, **Call-ID**, **CSeq**, **Contact** y **Max-Forwards**.

Para fijar ideas y ver de forma más clara los diferentes campos especificados se puede observar la traza SIP siguiente correspondiente a un SIP *Request* sin la negociación de SDP [4]. En la secciones 2.4.5.2 – Escenario de establecimiento de llamada entre dos UA y 2.4.5.3 – Establecimiento de llamadas entre dos UA mediante Proxy se ejemplifica gráficamente el intercambio de mensajes para el establecimiento de una sesión mediante el método **INVITE**.

```
INVITE sip:user1@obelix;user=phone SIP/2.0
Via: SIP/2.0/UDP obelix.com:5060;branch=z9hG4bK1d32hr4
Max-Forwards:70
To: <sip:user1@obelix;user=phone>
From: Beethoven <sip:beethoven@obelix.com>;tag=817234
Call-ID: 12-45-A5-46-F5-43-32-F3-C2
CSeq: 1 INVITE
Subject: Train Timetables
Allow: INVITE, ACK, CANCEL, BYE, OPTIONS, REFER, SUBSCRIBE, NOTIFY
Contact: sip:beethoven@obelix.com
Content-Type: application/sdp
Content-Length: 195
```

2.4.1.2 – REGISTER

Método utilizado por una UA para notificar a la red SIP que quien envía el mensaje con campo **Contact URI** (dirección IP) y la URI, deben tener *Request* enrutados en base al campo **Contact**. Efectuar la registración no es requerido para que un UA use el proxy para llamadas salientes, pero es necesario para que

un UA reciba llamadas entrantes desde otros proxy que operan como dominios. El mensaje de **REGISTER** podrá contener cuerpo. Dependiendo de los usos del campo **Contact** o del campo **Expires** el SIP Registrar tomará acciones diferentes. Un ejemplo claro es que si no se especifica el parámetro **Expires** o no está presente, la SIP URI expirará en un lapso de una hora. La presencia del encabezado **Expires** establece el tiempo de expirar para los contactos con carencia de este parámetro, pero si tiene establecido un valor en concreto fija el tiempo de expiración solo para ese contacto.

El campo **CSeq** es incrementado en cada envío de mensaje **REGISTER**. El uso de los campos **Request-URI**, **To**, **From** y **Call-ID** en el mensaje **REGISTER** es algo diferente que en otros *Requests*. El **Request-URI** contiene solo el dominio del registrar server sin la parte del usuario, y será reenviado o pasado por un proxy hasta que alcance un registrar autoritativo para ese dominio específico. El campo **To** contendrá el SIP URI de la AOR (*Address of Record*) del usuario cuyo agente es registrado. El **From** tendrá el SIP URI de quien envía el *Request*, usualmente es el mismo que el **To**. Es recomendable que el **Call-ID** se mantenga para todas las registraciones del UA.

Las posibilidades que reciba un UA al momento de enviar un **REGISTER**, pueden ser de un **3xx Redirection** o un **4xx Failure** conteniendo un encabezado de **Contact** donde debe de realizarse la registración y a donde enviarla.

Los encabezados exigidos en el mensaje con método **REGISTER** son: **Via**, **To**, **From**, **Call-ID**, **CSeq** y **Max-Forwards**.

En [5] se observa un ejemplo de cómo es la traza correspondiente al método **REGISTER** efectuado por la UA. En la sección 2.4.5.1 – Escenario de Registro se explica mediante un ejemplo gráfico la señalización para lograr un registro exitoso entre una UA y un servidor SIP.

```
REGISTER sip:192.168.2.4 SIP/2.0
Via: SIP/2.0/UDP 192.168.2.77:5060;branch=z9hG4bK126e8821-8b9
Max-Forwards: 70
To: <sip:1019@192.168.2.4>
From:<sip:1019@192.168.2.4>;tag=8239-e111-9582-f73c06d108b9
Call-ID: e8ed8721-8239-e111-9582-f73c06d108b9@serv1
CSeq: 1 REGISTER
Contact: <sip:1019@192.168.2.77>
Content-Length: 0
```

2.4.1.3 – BYE

Este método es necesario para terminar la sesión de media establecida. Dada una sesión establecida mediante el intercambio de **INVITE** y respuesta de **2xx** o **ACK**, un **BYE** es enviado solo por el UA que participa de la sesión, nunca por proxy u otros. Es un método que funciona de punta a punta, lo cual implica que las respuestas sean generadas por el otro UA. En el caso de un dialogo desconocido frente al envío de un **BYE**, la respuesta es del tipo **481 Dialog/Transaction Does Not Exist**.

Nunca un **BYE** será utilizado para cancelar **INVITE** que quedaron pendientes, ya que no será bifurcado como un **INVITE** y no llegara a los mismos UAs como el **INVITE**.

Los encabezados exigidos en el mensaje con método **BYE** son: **Via**, **To**, **From**, **Call-ID**, **CSeq** y **Max-Forwards**.

En el siguiente ejemplo podemos observar los campos definidos en la explicación anterior, a su vez en [6] se especifica el funcionamiento del método para la finalización de sesiones. En las secciones 2.4.5.2 – Escenario de establecimiento de llamada entre dos UA y 2.4.5.3 – Establecimiento de llamadas entre dos UA mediante Proxy se ejemplifica gráficamente el funcionamiento del método **BYE** para la finalización de sesiones entre dos UA.

```
BYE sip:qxyduwkn@192.168.2.2:51537 SIP/2.0
Via: SIP/2.0/UDP
192.168.2.77:49495;rport;branch=z9hG4bKPjYUVRwc2ROAxhasAcDwx5no.0N
iJ9YJRR
Max-Forwards: 70
To: <sip:201@192.168.2.18>;tag=97XcxzBt6.WTqre2weH.qT7eVkByaCzC
From: <sip:203@192.168.2.18>;tag=Dx-a-TRZ6XuJ1981M3HksGHGsarfXMYj
Call-ID: Md922t4xnDhMzNjqDKaNP5QEttP2viq-
CSeq: 2131 BYE
Content-Length: 0
```

2.4.1.4 – ACK

Este método es necesario para reconocer respuestas de los **INVITE** enviados, otras respuestas a otras *Request* no son reconocidas por este método. Otros tipos de respuestas son del tipo **2xx**, **3xx**, **4xx**, **5xx** o **6xx**. El numero **CSeq** no es incrementado en un **ACK** pero el campo **CSeq** es cambiado para el método **ACK**, tal es así, que un UAS puede adaptar el **CSeq** del **ACK** con el número correspondiente de **CSeq** del **INVITE**.

Puede contener **application/sdp** en el caso que el **INVITE** inicial no contenga información de SDP, si lo tuviese no es necesario que el **ACK** contenga esta información, ya que un **ACK** no debe ser utilizado para cambiar las características SDP que ya fueron enviadas por un mensaje de **INVITE**. Como se explico previamente, existe el mensaje de **re-INVITE** para este propósito.

Para respuestas del tipo **2xx**, el **ACK** es de punta a punta, pero para casos diferentes es salto a salto haciendo hincapié en servidores proxy involucrados. Como resultado el proxy generara **ACK** del tipo **3xx**, **4xx**, **5xx** o **6xx** al **INVITE**. Un **ACK** generado con reconocimientos del tipo salto a salto contendrá un único **Via** con la dirección del proxy generando el **ACK**, además se reutiliza el **branch-ID** del **INVITE** ya que se considera parte de la misma transacción. En los casos punta a punta se utiliza un **branch-ID** diferente puesto que se considera una nueva transacción.

Un proxy de modo *stateful* recibirá los **ACK** y deberá de determinar si es necesario que éste sea reenviado a otro proxy o a un UA, en base a saber si es salto a salto o punta a punta. Esto se analiza comparando el **branch-ID** con una base de *branches* pendientes, si no se encuentra una *branch* que coincida el proxy lo envía a la UAS, de lo contrario el **ACK** es para ese salto y no es reenviado por el proxy.

Los encabezados exigidos en el mensaje con método **REGISTER** son: **Via**, **To**, **From**, **Call-ID**, **CSeq** y **Max-Forwards**.

El método **ACK** se define en [7], el ejemplo a continuación muestra el aspecto constructivo del mismo.

```
ACK sip:tvcfipds@192.168.2.77:49495 SIP/2.0
Via: SIP/2.0/UDP 192.168.2.18;branch=z9hG4bKcydzigwkX
Max-Forwards: 69
To: <sip:203@192.168.2.18>;tag=Dx-a-TRZ6XuJl981M3HksGHGsarfXMYj
From: <sip:201@192.168.2.18>;tag=97XcxzBt6.WTqre2weH.qT7eVkByaCzC
Call-ID: Md922t4xnDhMzNjqDKaNP5QEttP2viq-
CSeq: 2124 ACK
Content-Length: 0
```

2.4.1.5 – CANCEL

Es un método utilizado para terminar **INVITE** pendientes o intentos de llamadas, puede ser generado por UA o servidores proxy a condición que el mensaje de respuesta **1xx** contiene un *tag*, pero no se ha recibido una respuesta final. Una UA usa este método para cancelar un intento pendiente de llamada que se ha iniciado anteriormente. Un proxy puede usar el método para cancelar solicitudes paralelas, luego de que una solicitud exitosa haya sido devuelta por el proxy hacia la UAC. El método **CANCEL** es un *Request* salto a salto y recibe respuestas del próximo elemento *stateful*. La diferencia entre el salto a salto y el punta a punta es que el **CSeq** no es incrementado, por lo que tanto los UA como los proxy pueden adaptar el **CSeq** del **CANCEL** a la **CSeq** del **INVITE** pendiente que corresponde.

El branch ID de un **CANCEL** se mapea con el del **INVITE** que quiere cancelar, por lo que el **CANCEL** solo tendrá sentido para un **INVITE**. Considerando que éste insume muchos segundos o minutos para que sea completado, mientras que otros SIP *Request* son completados de forma inmediata, estos procesos serán completados antes que el **CANCEL** sea recibido.

Un proxy que recibe un **CANCEL** lo reenvía al mismo conjunto de direcciones que están con un **INVITE** pendiente enviado. El proxy no espera por las respuestas de *Request* de **CANCEL**, las responde de forma inmediata. La UA confirma la cancelación con un mensaje **200 OK** y responde al **INVITE** con un **487 Request Terminated**. Si la respuesta final se recibió, la UA necesita enviar un **BYE** para terminar la sesión.

Dado que el *Request* de **CANCEL** es salto a salto, el mensaje no contendrá cuerpo alguno.

Los campos obligatorios del mensaje **CANCEL** son: **Via**, **To**, **From**, **Call-ID**, **CSeq** y **Max-Forwards**.

En [8] se explica con profundidad el método en cuestión, a su vez se puede observar el aspecto constructivo en la siguiente traza SIP utilizando el método.

```
CANCEL sip:qxyduwkn@192.168.2.2:51537 SIP/2.0
Via: SIP/2.0/UDP 192.168.2.18;branch=z9hG4bKcydzigwkX
Max-Forwards:70
To: <sip:203@192.168.2.18>;tag=Dx-a-TRZ6XuJ1981M3HksGHGsarfXMYj
From: <sip:201@192.168.2.18>;tag=97XcxzBt6.WTqre2weH.qT7eVkByaCzC
Call-ID: 23d8e0e4e2e505329299e288bbd4155a
```

CSeq: 3215 CANCEL
Content-Length: 0

2.4.1.6 – OPTIONS

El método **OPTIONS** tiene como objetivo efectuar consultas al UA o al servidor sobre las capacidades de estos y descubrir su eventual disponibilidad. La respuesta a la consulta lista las capacidades de la UA o del servidor. Los servidores proxy nunca generan consultas de **OPTIONS**, un UA o servidor responde a las consultas como si lo hiciese hacia un **INVITE**. Una consulta con respuesta exitosa (del tipo **2xx**) tendrá en su contenido encabezados que indicaran sus capacidades como **Allow**, **Accept**, **Accept-Encoding**, **Accept-Language** y **Supported**. La *Request* podrá contener o no un cuerpo de mensaje, el proxy determinará si la **OPTIONS Request** es para él mismo examinando el campo **Request-URI** y si éste contiene la dirección o el *hostname* del proxy, ese *Request* es para el proxy, de lo contrario el campo **OPTIONS** es para otro proxy o UA y la *Request* será reenviada.

Los encabezados exigidos en el mensaje con método **OPTIONS** son: **Via**, **To**, **From**, **Call-ID**, **CSeq** y **Max-Forwards**.

Dentro de la recomendación de la IETF [9] se discuten los aspectos constructivos y de utilización del método. El ejemplo ilustra la utilización de **OPTIONS** en la traza.

```
OPTIONS sip:qxyduwkn@192.168.2.2:51537 SIP/2.0
Via: SIP/2.0/UDP 192.168.2.77:49495;branch=z9hG4bKPjYUVRwc2RO
Max-Forwards:70
To: <sip:201@192.168.2.18>;tag=97XcxzBt6.WTqre2weH.qT7eVkByaCzC
From: <sip:203@192.168.2.18>;tag=Dx-a-TRZ6XuJl981M3HksGHGsarfXMYj
Call-ID: 747469e729acd305
CSeq: 29 OPTIONS
Content-Length: 0
```

2.4.1.7 – SUBSCRIBE

Es el método utilizado por la UA para establecer una subscripción con el propósito de recibir notificaciones (vía el método **NOTIFY**) de un evento en particular. Una subscripción exitosa establece un diálogo entre el UAC y el UAS, a su vez ésta contiene un campo **Expires** que indica la duración de la existencia

de la suscripción, luego de este tiempo la misma se da por terminada de forma automática.

Es posible mantener activa la suscripción mediante otro mensaje de **SUBSCRIBE** dentro del tiempo **Expires**. El servidor responderá un **200 OK** conteniendo un campo **Expires** con un nuevo valor. El tiempo puede ser el mismo que el que está en la consulta o el servidor podrá acortarlo, pero nunca incrementarlo. No existe método **UNSUSCRIBE** para SIP, el método para terminar la suscripción es enviar un **SUSCRIBE** con tiempo de vida igual a cero (**Expires:0**). La finalización de una suscripción resultara en un **NOTIFY** final que indicara que ha finalizado. Debe tenerse en cuenta que un **202 Accepted**, en respuesta a un **SUBSCRIBE** no indica que la suscripción haya sido autorizada, sino que la misma se ha interpretado por el servidor.

El cliente, encargado de gestionar la información de presencia, debe estar preparado para recibir **NOTIFY** antes de recibir un **200 OK** como respuesta de un **SUBSCRIBE**. Debido a un escalamiento posible, el cliente, debe estar preparado para recibir múltiples **NOTIFY** provenientes de diferentes servidores (los cuales tendrán diferentes To tag, estableciendo diferentes diálogos).

Los campos obligatorios en el mensaje **SUBSCRIBE** son, **Via, To, From, Call-ID, CSeq, Max-Forwards, Contact, Event y Allow-Events**.

El siguiente ejemplo especifica el uso del método en las trazas SIP, se puede observar la descripción del funcionamiento del mismo en la RFC – 3265 [10].

```
SUBSCRIBE sip:1000@192.168.2.2 SIP/2.0
Via: SIP/2.0/UDP
192.168.2.3:58232;branch=z9hG4bKPjZ1RJ7Jhwx4eSctNeTt-MDCl.GoKl5X3Z
Max-Forwards: 70
To: <sip:1000@192.168.2.2>
From: <sip:1000@192.168.2.2>;tag=YNQo7iN3xsIvAOaeVyRBK1tuQZSCh3cD
Call-ID: rGNy-leNdAV1zHR3PdIXP2bvm8VO7Hcs
CSeq: 25977 SUBSCRIBE
Allow-Events: conference, message-summary, presence,
presence.wininfo, xcap-diff, refer
Contact: <sip:tebnpiuw@192.168.2.3:58232>
Event: dialog
Accept: application/simple-message-summary
Content-Length: 0
```

2.4.1.8 – NOTIFY

Utilizado por una UA para acordar información sobre la ocurrencia de un evento en particular, es enviado dentro del dialogo establecido siempre que exista una suscripción entre el suscriptor y el notificador. Dado que se da dentro del dialogo **NOTIFY** contendrá **To tag**, **From tag** y la **Call-ID**. Una consulta de **NOTIFY** recibirá un **200 OK** en respuesta de que es recibido, si un mensaje del tipo **481 Dialog/Transaction Does Not Exist** es recibido, la suscripción es terminada de forma automática y no se envían **NOTIFY** posteriores.

Los **NOTIFY** contienen un campo llamado **Event** indicando el paquete y un **Subscription-State** indicando el estado actual de la suscripción, en el caso de **Event** tendrá el nombre del paquete usado en la suscripción, en el caso de **Subscription-State** el campo podrá tener los valores **active**, **pending** o **terminated**.

Se envían al comienzo y fin de una suscripción, si un **NOTIFY** contiene información de estado incremental, el cuerpo del mensaje incluirá el numero de versión que se incrementará en uno por cada **NOTIFY** enviado, de esta forma el receptor de la notificación puede decir información que se perdió o recibió fuera de secuencia.

Los campos obligatorios en un mensaje **NOTIFY** son, **To**, **Via**, **From**, **Call-ID**, **CSeq**, **Max-Forwards**, **Event**, **Allow-Events** y **Subscription-State**.

La misma RFC [11] que define el método **SUBSCRIBE**, también define el método **NOTIFY**. En el ejemplo siguiente se muestra el empleo del método en la traza SIP.

```
NOTIFY sip:brahe@192.168.2.254 SIP/2.0
Via: SIP/2.0/UDP 192.168.2.25;branch=z9hG4bK13cf.ebab1172.0
Max-Forwards: 70
To: sip:brahe@obelix;tag=nojwz
From: sip:kepler@obelix;tag=a6a1c5f60faecf035a1ae5b6e96e979a-281e
Call-ID: nwiwqcrfmobukrs@obelix
CSeq: 2 NOTIFY
Contact: <sip:192.168.2.25:5060;transport=udp>
Event: dialog
Subscription-State: active;expires=3670
Allow-Events: dialog
Content-Type: application/pidf+xml
Content-Length: 208
```


2.4.1.9 – PUBLISH

Es utilizado por el UA para enviar (o publicar) cambios de estado para un servidor conocido como ESC (*Event State Compositor*), el método **PUBLISH** es más útil cuando existen múltiples fuentes de cambios de estado de información, así como también cuando existen varios dispositivos intercambiando la misma AOR. En este caso, para encontrar un estado completo, otra UA necesitará suscribirse de forma individual a todos los demás dispositivos. Asimismo, la UA puede suscribirse al ESC. Las UA individuales envían **PUBLISH** al ESC donde se procesan y se colocan juntos generando **NOTIFY**.

Un **PUBLISH** es similar a un **NOTIFY**. La diferencia es que no se envía en el dialogo establecido. Contendrá un campo de **Expires** y de **Min-Expires**. El campo **Expires** indica el tiempo máximo antes que el ESC de por descartada la información de estado, siempre y cuando no sea actualizado. El campo **Min-Expires** indica el tiempo mínimo que deberá elegir el ESC. En la respuesta **200 OK** al **PUBLISH** el campo Expires indicara el valor elegido por el ESC que deberá estar comprendido entre los valores descriptos anteriormente, de ser un intervalo muy pequeño el ESC responderá con un mensaje **423 Interval Too Brief** conteniendo un valor adecuado de **Min-Expires** aceptable por parte del ESC, seguido de esto, la UA publicará nuevamente la información con este valor de intervalo correcto.

La RFC – 3903 [12] especifica el funcionamiento y los aspectos constructivos del método **PUBLISH**. El ejemplo siguiente muestra como es utilizado en una traza SIP.

```
PUBLISH sip:stark@192.168.2.254 SIP/2.0
Via SIP/2.0/UDP 192.168.2.25:54620;branch=z9hG4bK43d132s3
Max-Forwards: 70
To: <sip:stark@192.168.2.25>
From: <sip:stark@192.168.2.25>;tag=5645fg432f
Call-ID: 34jdUhwIQhd72e
CSeq: 352 PUBLISH
Contact: <sip:stark@192.168.2.254:54620>
Event: presence
Min-Expires: 1800
Expires: 3600
Allow-Events: presence
Content-Type: application/xml+pidf
Content-Length: 208
```

2.4.1.10 – REFER

Su utilidad es enviar desde un UA a otro UA la petición necesaria para acceder a la URI o URL de un recurso, siendo este identificado por la URI o URL en el campo **Refer-To**. Tanto la URI como la URL pueden ser de cualquier tipo (sip, sips, http, pres). Cuando la URI es del tipo sip o sips, el **REFER** está siendo utilizado para implementar una transferencia de llamada, además de ser utilizado para implementar controles de llamada del tipo peer-to-peer.

Incluso, un **REFER** puede ser enviado desde el interior o desde el exterior de un dialogo ya establecido. Un **REFER** y la consulta SIP disparada por este contendrán en el campo **Referred-By** la información de quien origino la consulta correspondiente.

Opcionalmente el campo **Referred-By** puede ser incluido en la *Request* de **REFER**, además en el campo **Refer-To** se puede incluir una lista de las funcionalidades soportadas por el extremo al que se hace la referencia.

Los campos obligatorios en el mensaje de **REFER** son, **Via**, **To**, **From**, **Call-ID**, **CSeq**, **Max-Forwards** y **Refer-To**.

Se especifica el funcionamiento del método **REFER** en la RFC – 3515 [13], el ejemplo de traza SIP que incluye el método se muestra a continuación.

```
REFER sip:1007@192.168.2.3 SIP/2.0
Via: SIP/2.0/UDP 192.168.2.77;branch=z9hG4bKxwltzjw
Max-Forwards: 70
To: <sip:1007@192.168.2.11>;tag=lgpik
From: <sip:1000@192.168.2.11>;tag=ifaye
Call-ID: eaqrcevmtxatuzm@gdp-laptop
CSeq: 311 REFER
Refer-To: <sip:1008@192.168.2.11>
Referred-By: <sip:1000@192.168.2.11>
Content-Length: 0
```

2.4.1.11 – MESSAGE

Método utilizado para enviar mensajes instantáneos (IM) usando SIP. Consiste en mensajes cortos intercambiados en tiempo real por parte de los participantes de una conversación en base a texto. Estos mensajes pueden ser enviados dentro

del dialogo o por fuera del dialogo establecido, pero no establecen diálogos entre ellos.

La forma de transportar el contenido del mensaje es con un adjunto del tipo MIME³. Todas las UA que soporten el método **MESSAGE** soportaran el formato de **plain/text** y también podrán soportar formatos como **message/cpim** o **text/html**.

Un mensaje con el método **MESSAGE** recibe un **200 OK** indicando que el mensaje ha sido recibido por el destinatario, pero la respuesta no será incluida en el cuerpo del **200 OK**, se enviará en otro **MESSAGE** por parte del destinatario al emisor. Los mensajes del tipo **202 Accepted** indicarán que se habrán alcanzado un tipo de envío “*store-and-forward*” y que serán entregados al destinatario a su debido momento.

La consulta **MESSAGE** utilizará el tipo de URI con el identificador **im** (instant message) en la **Request-URI**.

Los campos obligatorios en el mensaje de **MESSAGE** son, **To, Via, From, Call-ID, CSeq** y **Max-Forwards**.

Dicho método queda definido en la RFC – 3428 [14], ejemplificado en la siguiente traza SIP.

```
MESSAGE sip:1001@192.168.2.254 SIP/2.0
Via: SIP/2.0/UDP 192.168.2.3; branch=z9hG4bKyeaghzte
Max-Forwards: 69
To: <sip:1001@obelix>
From: <sip:1002@obelix>;tag=ngznx
Call-ID: znusnhstglticmu@gdp-laptop
CSeq: 991 MESSAGE
Content-Type: text/plain;charset=utf-8
Content-Length: 46
```

2.4.1.12 – INFO

Método usado por una UA para enviar información de señalización de una llamada a otra UA con la que se ha establecido una sesión. La consulta es de punta a punta y nunca es iniciada por un proxy, el proxy se encarga de reenviar el

³ MIME: Multipart Internet Mail Extensions

INFO y relega a la UAS la verificación de que sea un dialogo válido. Los **INFO** que se envíen a diálogos desconocidos recibirán un **481 Transaction/Dialog Does Not Exist**. En el ejemplo al final de la explicación de los campos obligatorios se expone una traza SIP. El método queda definido por la IETF en la RFC – 2976 [15].

Cada mensaje tendrá en su cuerpo información de señalización, eventos de llamada. Además el método **INFO** incrementa el valor **CSeq**.

Los campos obligatorios en el mensaje de **INFO** son, **To, Via, To, From, Call-ID, Max-Forwards** e **Info-Package**.

```
INFO sip:1001@obelix SIP/2.0
Via: SIP/2.0/UDP 192.168.2.2;branch=z9hG4bK24555
Max-Forwards: 70
To: <sip:1002@obelix> ;tag=3432
From: <sip:1001@obelix>;tag=432485820183
Call-ID: e71facaa7f7c0a29276054fe4951a9b6
Content-Type: application/ISUP
Content-Length: 24
```

2.4.1.13 – UPDATE

Método utilizado para modificar el estado de una sesión sin cambiar el estado del diálogo. Como se explico anteriormente, una sesión se establece mediante el mensaje de **INVITE** en modo offer/answer. En una sesión ya establecida, un re-**INVITE** es usado para actualizar parámetros de la misma, pero ninguna sesión pendiente podrá enviar re-**INVITE**, por lo que se envían mensajes con el método **UPDATE**.

La utilidad para el empleo del método **UPDATE** incluye silenciar, colocar en espera o dejar pendiente flujos de media, aplicar calidad de servicio o incluso atributos punta a punta que se negocien antes de establecer la sesión.

Los campos obligatorios en el mensaje de **UPDATE** son, **To, Via, To, From, Call-ID, CSeq, Max-Forwards** y **Contact**.

La RFC – 3311 [16] define y especifica el método **UPDATE**, en el ejemplo vemos como se utiliza el método en una traza SIP.

```
UPDATE sip:1001@obelix SIP/2.0
Via SIP/2.0/TLS 192.168.2.25;branch=z9hG4bK342
Max-Forwards: 70
To: <sip:1001@192.168.2.25>;tag=71
From: <sip:1002@192.168.2.240>;tag=19438
Call-ID: 170189761183162948
CSeq: 94 UPDATE
Contact: <sip:1001@obelix>
Content-Type: application/sdp
Content-Length: 20
```

2.4.2 – Tags

Para identificar de forma inequívoca un diálogo se añade a los campos **To** y **From** un número aleatorio criptográfico de 32 bits denominado “*Tag*”. Al comienzo de una sesión, los **INVITE** no contendrán en su campo **To** el tag, quien lo tendrá será el campo **From**.

Excluyendo los mensajes de respuesta **100 Trying**, todas las demás *Responses* tendrán un tag agregado al campo **To**. Cuando se responde un **200 OK** con un tag, éste es incorporado como un identificador del dialogo y será utilizado para futuras *Request* a este **Call-ID**. Nunca un *tag* es copiado a lo largo de una llamada, cualquier respuesta generada por un proxy tendrá un tag agregado por el proxy, pero un **ACK** generado por un UA o un proxy siempre copiará el *tag* del **From** de la respuesta en el **ACK** del *Request*.

Si una UAC recibe respuestas conteniendo diferentes *tags*, esto implica que las respuestas provienen de diferentes UAS y que los **INVITE** se han bifurcado en varios, queda a criterio del UAC como resolver esta situación. En concreto, los diálogos tendrán mismo **From**, **Call-ID** y **CSeq**, pero diferirán en los campos **tag** y **To**. La UAC podrá enviar **BYE** a ciertos extremos y establecer una única sesión.

Los *tags* no son parte de los campos **To** y **From URI**, pero si son parte del encabezado y serán ubicados por fuera de los caracteres “<>”.

2.4.3 – Message Body

Es en el cuerpo del mensaje donde se acumulara distintos tipos de información, contendrá información del tipo SDP necesaria para establecer parámetros de sesiones de media, en otros casos información de QoS o información de seguridad.

El campo **Content-Disposition** es usado para indicar la intención de utilizar el cuerpo del mensaje, de no contar con este campo, se asume que el mensaje es del tipo **session** implicando que el cuerpo del mensaje corresponde con una sesión de media.

Los tipos de formatos de cuerpo de mensaje son especificados en el campo **Content-Type**, de contener un cuerpo de mensaje debe de estar presente el campo **Content-Type**. Todas los UA deben soportar el **Content-Type** de **application/sdp**. El campo **Content-Encoding** es el tipo de codificación establecida para ese cuerpo de mensaje, de no estar especificado se asume que el tipo de codificación es **text/plain**. Al establecer este parámetro permite la compresión del cuerpo del mensaje.

El **Content-Length** contiene el numero de octetos del cuerpo de mensaje, si no hay cuerpo de mensaje deberá estar incluido pero establecido en cero. Si no está presente el UAC deberá asumir que el cuerpo del mensaje continúa hasta la finalización del datagrama UDP o que la sesión TCP se finalice, o sea depende del protocolo de transporte.

SIP tiene la particularidad de poder enviar de igual modo los cuerpos de mensaje que un correo electrónico, eso permite enviar diferentes cuerpos de mensajes en un mismo mensaje SIP. Esto es el resultado de emplear cuerpos del tipo MIME multiparte listando el **Content-Type** como **multipart/mime** y un separador que es parseado para lograr separar el mensaje. Cualquier SIP *Request* o *Response* que contenga un cuerpo de mensaje podrá llevar un MIME multiparte.

El contenido del cuerpo de mensaje puede verse en este ejemplo, si bien lo descripto hasta el momento no abarca los parámetros que se ejemplifican, en la sección correspondiente a SDP se explicarán que implica cada uno de estos valores.

```
v=0
o=- 3536348748 3536348749 IN IP4 192.168.2.77
s=Blink 0.2.8 (Linux)
c=IN IP4 192.168.2.77
t=0 0
m=audio 50000 RTP/AVP 0 101
a=rtcp:50001
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
```

```
a=fmtp:101 0-15  
a=sendrecv
```

2.4.4 – Response

El mensaje de respuesta SIP es generado por el UAS o por un SIP server en respuesta a un *Request* generado por un UAC. Puede contener encabezados adicionales en los cuales se detalla información necesaria para el UAC, o simplemente un reconocimiento para prevenir retransmisiones de los *Request* provenientes del UAC.

Se mostrarán las diferentes *Responses* que se pueden enviar agrupadas en clases dependiendo de los mensajes que el UAS tenga que enviar frente a los *Request* del UAC. Se ahondará en mensajes concretos que son de utilidad al momento de realizar los análisis posteriores de las capturas o en su defecto para analizar los del funcionamiento de Obelix.

Existen seis clases de SIP *Responses*, las primeras cinco clases se heredan de HTTP, la sexta clase fue creada por SIP.

Cada una de las *Responses* y los códigos que se obtienen en respuesta a los *Request* generados son detalladas en profundidad RFC – 3261, en [2] también se expone los diferentes escenarios donde se cursarán las *Responses* que aquí se detallan. En esta sección se propone un resumen de las diferentes respuestas que se obtienen y que serán manejadas a lo largo de las diferentes implementaciones.

2.4.4.1 – Informational – 1xx

La clase de tipo información (informacional – **1xx**) es la indicada para informar el procesamiento de una llamada, este tipo de *Response* es de tipo punta a punta y contendrá un cuerpo de mensaje. La excepción es la **100 Trying** que es de tipo salto a salto y no contendrá un cuerpo de mensaje.

100 **Trying**

Caso especial donde la respuesta es salto a salto, nunca es reenviada y no contiene cuerpo de mensaje, cualquier servidor proxy de la red deberá de enviar un **100 Trying** en respuesta a las *Request* enviadas. Esta respuesta es generada tanto por los proxy como también por UA, solo indica que una acción se está dando para poder procesar la llamada, no quiere decir que el usuario final está siendo localizado. Un **100 Trying** no contiene un *tag* en el **To** y no crea un diálogo.

180 **Ringing**

Respuesta utilizada para indicar que el **INVITE** ha sido recibido por el UA y anunciando que se está procesando. La respuesta de este tipo es de suma importancia para la red de telefonía SIP.

181 **Call is Being Forwarded**

Respuesta para indicar que la llamada ha sido puesta en espera para otro punto de destino. Se envía cuando la información es de utilidad para el originador de la llamada. Además en caso de una operativa de reenvío de llamadas, si ésta toma más tiempo de lo habitual, informa al llamador del estado de la llamada.

183 **Session Progress**

Respuesta para indicar el progreso de la sesión o estado de la llamada, estará presente en el cuerpo del mensaje del flujo de datos. A diferencia del **100 Trying**, el **183** es una respuesta de punta a punta y establece un dialogo, por lo que contiene *tag* tanto en el **To** como en el **From** y **Contact**.

2.4.2.2 – Success – 2xx

La clase *success* o de éxito, indica que la *Request* ha sido exitosa o que fue aceptada. Se describen a continuación algunos de los tipos de mensajes que se llegan a cursar.

200 OK

Respuesta que tiene dos usos en SIP. Primeramente se utiliza para aceptar la invitación de una sesión en la cual contendrá un cuerpo de mensaje con propiedades de media del UAS. Cuando es usado para otros *Request*, tiene como función indicar que se completó la recepción del *Request* y al recibir esta respuesta evita las posibles retransmisiones de la *Request*. En respuesta a un **OPTIONS**, el cuerpo del mensaje contendrá las capacidades del servidor, a su vez el cuerpo del mensaje podría estar presente en la respuesta a un **REGISTER**.

202 ACCEPTED

Indica que el UAS recibió y comprendió una *Request*, pero que la *Request* no ha sido autorizada hasta el momento o procesada por el servidor. Es común utilizarla en respuesta al **SUBSCRIBE**, **REFER** y en algunos casos frente a **MESSAGE**.

2.4.4.3 – Redirection – 3xx

Esta clase se emplea para casos en que el servidor SIP se utiliza como redirector en respuesta a **INVITE**, sin embargo, puede enviar redirecciones para cierto tipo de llamadas con capacidades de transferencia. El UAC puede ser configurada para que genere de forma automática un nuevo **INVITE** al recibir una respuesta de la clase Redirection sin requerir intervención del usuario, sumado a que los proxy podrán de forma automática enviar **ACK** al redirector y al proxy el **INVITE** con la nueva ubicación en la **Contact URI** de la redirección. Para prevenir el *loop*, el servidor no debe devolver direcciones contenidas en el campo **Via** del *Request* y el cliente debe chequear la dirección en el campo **Contact** contra todas las otras direcciones intentadas en llamadas anteriores.

2.4.4.4 – Client Error – 4xx

Clase que se utiliza por parte del servidor o el UAS para indicar que la *Request* no puede ser completada según la forma que fue enviada. El error específico cometido por el cliente o la presencia de ciertos encabezados indicaran a el UAC la naturaleza del error y así se podrá reformular la *Request* de forma correcta, el

UAC no deberá retransmitir la *Request* sin haber modificado previamente lo reportado en la respuesta de esta clase.

400 Bad Request

Indica que la *Request* no fue comprendida por el servidor. Claro ejemplo es cuando en la *Request* faltan encabezados como **To**, **From**, **Call-ID** o **Cseq**. También se utiliza en caso que el UAS reciba **INVITES** múltiples (que no sean retransmisiones) para un mismo **Call-ID**.

401 Unauthorized

Respuesta que indica que la *Request* requiere por parte del usuario que realice una autenticación, en general es enviada por un UA, siendo que el proxy envía un mensaje de **407 Proxy Authentication Required**.

403 Forbidden

Utilizada para denegar un *Request* sin entregar al llamador algún recurso en específico, se envía cuando el servidor entendió el *Request*, la encuentra perfectamente formulada, pero no puede dar servicio sobre la misma. No es utilizada cuando lo que se precisa es autorización.

404 Not Found

Indica al usuario identificado por la URI sip o sips en la **Request-URI** que no puede ser ubicado por el servidor o que el usuario final no está registrado con el UA.

405 Method Not Allowed

Respuesta para indicar que el servidor o el UA han recibido y comprendido la *Request* pero no tienen la posibilidad de completar la misma. Se puede tomar como ejemplo cuando un **REGISTER** es enviado a un UA, en el encabezado **Allow** debe estar presente por parte del UAC los métodos que acepta, pero es

un caso diferente a un método desconocido en que se le responderá **501 Not Implemented**.

406 Not Acceptable

Respuesta que indica que la *Request* no puede ser procesada debido a requerimientos del mensaje de *Request*. Un caso concreto es que el encabezado **Accept** no contenga opciones soportadas por el UAS.

407 Proxy Authentication Required

Request enviado por el UAS pidiendo al UAC que primeramente debe de autenticarse con el proxy antes de que su solicitud sea procesada. En la respuesta podrá incluir el tipo de claves requeridas por el proxy, precisamente en el encabezado **Proxy-Authenticate**, para completar la autenticación. La *Request* será reformulada para ser enviada con un encabezado utilizando las credenciales anunciadas previamente ubicadas en el campo **Proxy-Authenticate**.

408 Request Timeout

Respuesta enviada cuando el tiempo especificado en el encabezado **Expires** de un **INVITE** se ha superado. Esta respuesta puede ser enviada por el proxy o por el UA, la alternativa para poder incrementar este tiempo es reenviar la *Request* con un periodo de tiempo más grande que el anterior en el encabezado **Expires** o incluso no enviar el con el encabezado.

420 Bad Extension

Indica que la extensión especificada en el encabezado **Require** no es soportada por el proxy o el UA, la respuesta deberá contener en el campo **Supported** un listado de extensiones que pueden ser soportadas. El UAC podrá reintentar la misma *Request* sin el encabezado **Require** o hacerlo a otro proxy o UA.

482 Loop Detected

Respuesta que indicará que la *Request* ha entrado en un *loop* y que ha sido ruteada de vuelta al proxy que previamente reenvió esa misma *Request*. Cada servidor que reenvía una *Request* agrega un encabezado **Via** con su dirección, el parámetro **branch** es agregado al encabezado **Via** siendo este un *message digest*⁴ (hash) [17] conformado por la **Request-URI**, **To**, **From**, **Call-ID** y el número **CSeq**. El parámetro **branch** debe ser chequeado para permitir que la *Request* pueda ser enviada nuevamente al proxy garantizando que la **Request-URI** ha cambiado. El caso concreto es cuando una llamada es transferida, en concreto, el header de **Via** estará diferenciado por diferentes parámetros de **branch**.

487 Request Terminated

Es enviada por el UA cuando recibe un **CANCEL** por un **INVITE** pendiente, se confirma cuando un **200 OK** es enviado en respuesta al reconocimiento del **CANCEL** y se envía el **487** en respuesta al **INVITE**.

489 Bad Event

Respuesta enviada para rechazar una suscripción o notificación conteniendo un paquete Event que es desconocido no soportado por el UAS.

2.4.4.5 – Server Error – 5xx

Clase para indicar que la *Request* o las *Request* no pueden ser procesadas debido a un error del servidor, la respuesta contendrá un campo de **Retry-After** si el servidor puede prever su disponibilidad dentro de un periodo de tiempo específico. La *Request* puede intentar ser enviada a otros sitios ya que no se indican errores en la misma.

⁴ Message Digest (compendio de mensaje): es una función de hash que toma un texto llano y calcula una cadena de bits de longitud fija.

500 Server Internal Error

Mensaje enviado por el servidor para indicar que el servidor experimento un tipo de error previniendo de procesar la *Request*, la razón explicitada en la frase puede ser utilizada para identificar la razón de la falla. Permite que el cliente pueda intentar la *Request* luego de un lapso de tiempo.

501 Not Implemented

Esta respuesta indica que el servidor no puede procesar la *Request* porque no es soportada por él, en concreto es cuando se rechaza una *Request* conteniendo un método desconocido. En el caso de un proxy se reenviará anunciando el método desconocido.

503 Service Unavailable

Anuncia que el servicio está temporalmente no disponible, la *Request* puede ser reiterada luego de unos segundos o después de que pase el tiempo establecido en el encabezado **Retry-After**.

2.4.4.6 – Global Error – 6xx

Respuesta enviada por el servidor asumiendo que será rechazada por otros servidores o en donde se intente Como resultado de esta operativa, la *Request* no deberá de ser enviada a otras ubicaciones. Solo el servidor que tiene conocimiento del usuario identificado por la *Request*-URI podrá enviar mensajes del tipo error global, de lo contrario se deberá de responder con un error del tipo clase cliente.

2.4.5 – Ejemplos Gráficos de Intercambio de Mensajes en Diferentes Escenarios

A modo de fijar con claridad los mensajes de *Request* y *Response* discutidos en las secciones 2.4.1 y 2.4.4 se expondrán flujos básicos de señalización en forma gráfica, correspondiente a diferentes escenarios simples.

2.4.5.1 – Escenario de Registro

Mediante el siguiente esquema de flujo de señalización SIP entre un UA y un servidor podemos observar el intercambio de mensajes para establecer un registro exitoso en el sistema. Cabe resaltar que este requiere de la autorización correspondiente por parte el servidor y que el UA envíe las credenciales correspondientes para que la registración se complete de forma exitosa. Se resalta el intercambio de mensajes del tipo *Request* enviados por el UA del tipo **REGISTER** y los mensajes del tipo *Response* enviados por el servidor correspondientes a las clases **4xx** y **2xx**.

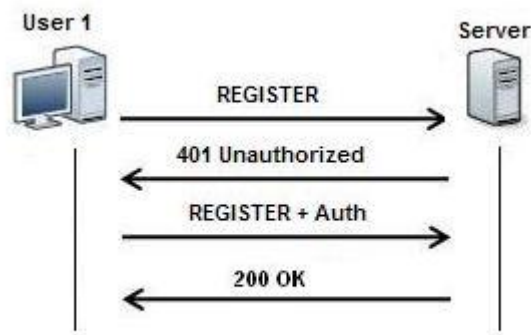


Figura 2.1 - Flujo SIP de registración

2.4.5.2 – Escenario de establecimiento de llamada entre dos UA

El comienzo de una llamada desde un UA a otro UA inicia con el método **INVITE**, seguido posteriormente de la respuesta enviada por parte del UA llamada que responde con mensajes de *Response* del tipo **1xx** y **2xx** para comenzar a establecer el flujo RTP que transporta el audio de un punto al otro de la comunicación. Finalmente se observa cuando una de las partes involucradas finaliza la comunicación mediante el *Request* de **BYE**, siendo respondido por la otra parte mediante un mensaje de *Response* del tipo **2xx**.

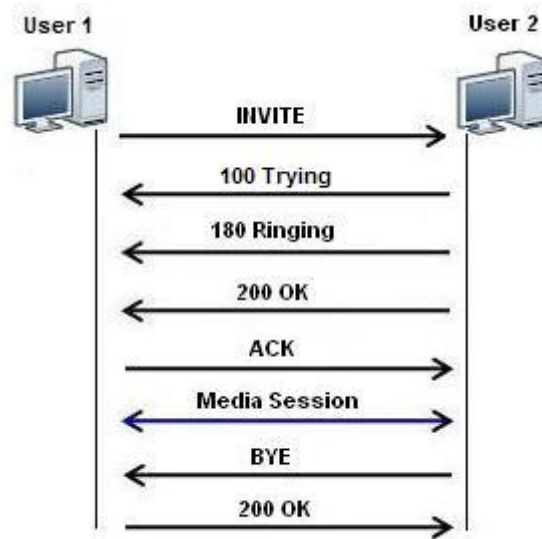


Figura 2.2 - Flujo SIP de establecimiento de llamada

2.4.5.3 – Establecimiento de llamadas entre dos UA mediante Proxy

A diferencia del ejemplo anterior el servidor Proxy se encarga de manejar de realizar la señalización para el establecimiento de la llamada entre dos UA registradas en el servidor. Una vez negociada la sesión, el flujo RTP es enviado de punta a punta sin pasar por el Proxy. En materia de tipos de *Request* y *Response* no difieren, pero en lo que si difieren es en quien recibe y envía las diferentes respuestas a las consultas realizadas.

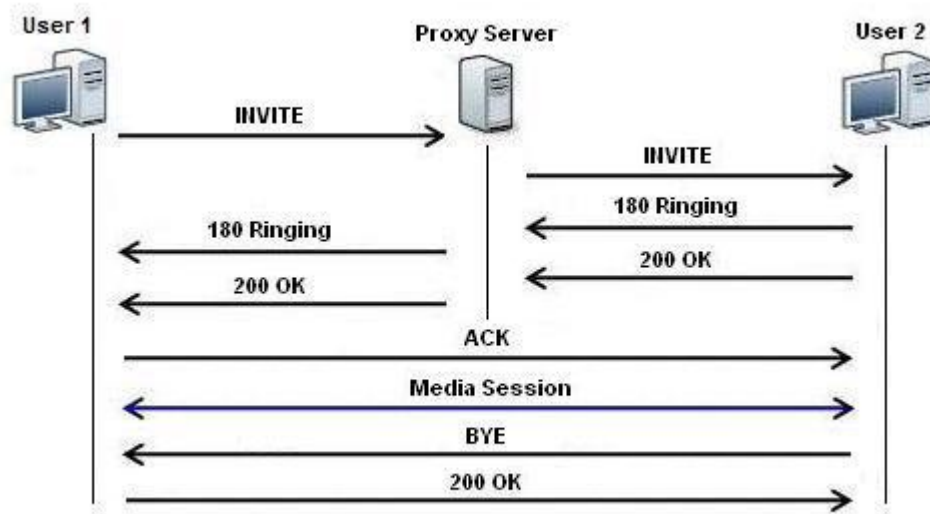


Figura 2.3 - Flujo SIP de establecimiento de llamada mediante proxy

2.5 – Elementos del Encabezado SIP

El subtítulo del capítulo hace abuso de notación, ya que el encabezado SIP se compone de varios encabezados SIP, con esto nos referimos a que existen varios elementos que definen un mensaje SIP en los cuales se establecen diferentes tipos de parámetros que posteriormente se procesarán por los agentes SIP de la red para llevar a cabo las operativas necesarias.

Se categorizarán dependiendo si son encabezados pertenecientes a mensajes de *Request*, mensajes de *Response*, o encabezados comunes a los mensajes de *Request* y *Response*. Los diferentes encabezados se especifican en la RFC-3261. El formato de los encabezados es **Header: field**, donde “Header” se especifica en mayúscula para definir el nombre del encabezado y en “field” la información específica de ese encabezado. El orden de los sucesivos encabezados no es de importancia a menos que se especifique lo contrario para algún caso en concreto. Los campos que no sean reconocidos se ignorarán por el servidor proxy. Desde el punto de vista del reenvío, los encabezados podrán ser del tipo salto a salto o punta a punta, con la excepción de que los primeros son los únicos que los proxy podrán insertar o modificar con algunas excepciones. Como SIP engloba un control de punta a punta, en su mayoría, los encabezados serán del tipo punta a punta.

Los diversos encabezados que se detallarán en las secciones siguientes son un resumen de lo explicado en donde se detallan ejemplos y usos en detalle con profundidad. La referencia [18] y [2] también expone ejemplos detallados de intercambio de mensajes y el empleo de los diferentes campos del encabezado SIP.

2.5.1 – Campos comunes a Request y Response

Accept

El campo está definido por HTTP e indica la aceptación de mensajes de media provenientes de Internet en el cuerpo del mensaje. Describe los tipos de media que se usarán mediante el formato **type/sub-type** usualmente empleado en Internet, de no contar con la presencia de este campo se asume que el formato de cuerpo del mensaje es **application/sdp**.

Accept-Encoding

Definido también por HTTP, especifica que tipos de codificación en el cuerpo del mensaje se pueden aceptar. La codificación es útil al momento que un mensaje SIP con un cuerpo extenso pueda ajustarse en un único mensaje UDP. Si no hay aceptación por parte del UAC de estos códigos se envía un mensaje de **406 Not Acceptable**. De no incluir códigos se asume que la codificación será **text/plain**. Ejemplos son **Accept: application/sdp**, **Accept: text/***.

Allow

Indica métodos que soportan el UA o el servidor Proxy. Exige que esté presente en casos de respuestas **405 Method Not Allowed** y debe de estar presente en respuestas positivas a un mensaje **OPTIONS**. El campo **Allow** está siempre presente en las respuestas **INVITE** y **200 OK**.

Ejemplo: **Allow: INVITE, ACK, BYE, INFO, OPTIONS, CANCEL**

Allow-Events

Campo que se utiliza para listar los paquetes que son soportados. Es el caso cuando un UA envía **SUSCRIBES** para un paquete en especial.

Ejemplo: **Allow-Events: dialog,u:conference**

Answer-Mode

Campo que se utiliza para tener una respuesta inmediata (del tipo **200 OK**) ante un **INVITE**. Soporta dos tipos de comportamiento; Manual, que fija un comportamiento normal y Auto, que implica una respuesta inmediata sin entrada por parte del usuario. Otro campo que se maneja es el **Require** que establece que si el comportamiento de respuesta no está permitido devuelva un **403 Forbidden**.

Call-ID

Este campo es obligatorio tanto en *SIP Request* y *SIP Responses* Identifica de forma inequívoca y única la llamada entre dos UA. Debe ser único a lo largo de todas las llamadas excepto al momento de los pedidos de registración, ya que estas solicitudes por parte de un UA deben tener el mismo **Call-ID**. Este campo es creado por el UA y es inalterado por parte del servidor. Existe un método de seguridad basado en forma criptográfica para este campo y dar cierto carácter aleatorio al mismo, previniendo que un tercero pueda adivinar el **Call-ID** y así enviar *Request* falsos.

Contact

Campo que convenido con una URI identifica la fuente del *Request* o quien ha originado esa *Request*, dependiendo si está presente en una *Request* o una *Response*. Una vez que el campo **Contact** ha sido recibido, la URI puede ser utilizada para hacer el reenvío de *Request* futuras en el diálogo. Debe de estar presente en **INVITE** y **200 OK**.

CSeq

Es un campo requerido en cada uno de los *Requests*, contiene un número decimal que se incrementa en cada *Request* y que generalmente se incrementa en uno por cada nuevo *Request* exceptuando los mensajes **CANCEL** y **ACK** que utilizan el número del **CSeq** de los **INVITE** a los cuales se refieren. Esta cuenta es usada por los UAS para determinar *Request* fuera de secuencia (**CSeq** diferente) o para discriminar entre diferentes *Requests* o retransmisiones (**CSeq** iguales), por parte del UAC, adapta la respuesta al *Request* que se referencia. Cada UA mantiene su propio espacio de números de secuencia.

Date

Es en este campo que se acuerda la fecha entre un *Request* y una *Response*. El formato de fechas manejado por SIP es el mismo que utiliza HTTP, pero permite solo el del estándar de Internet referenciado en la RFC-1123 [19], pero para mantener la fecha del UA y lógica de tiempo de forma simple, SIP solo soporta fechas en el formato GMT según la zona horaria.

Encryption

Es el campo del encabezado definido en la RFC-2534 pero que no está incluido en la RFC-3261.

Expires

Es implementado para indicar el intervalo de tiempo dentro del cual, los mensajes de *Request* o mensajes contenidos son válidos. Cuando está presente en un *Request* de **INVITE**, el campo del encabezado fija un límite de tiempo para que se complete el *Request* de **INVITE**. Esto implica que el UAC reciba una respuesta final (que no sea **1xx**) dentro del periodo establecido, de lo contrario un mensaje del tipo **408 Request Timeout** es enviado.

From

Es un campo requerido del cabezal para identificar el origen del *Request*. Es una de las dos direcciones para identificar el diálogo conteniendo una URI sin identificar el tipo de transporte, o tiempo de vida del paquete. El campo **From** mantiene un tag utilizado para identificar una llamada en particular. El formato del campo **From** es una URI que se mantiene definida entre <>, incluso, de tener un tag y una URI, esto va todo dentro de los caracteres <>.

Record-Route

Este campo se utiliza para forzar el ruteo a través de un proxy para todos los *Request* posteriores en una sesión establecida entre dos UA. Normalmente con la presencia del campo **Contact**, permite a los UA enviar mensajes de forma directa pasando por la cadena de proxy utilizada en el primer *Request*, la cual ha efectuado consultas a la base de datos y ubicado a la parte llamada. Un proxy inserta su dirección en el campo **Record-Route** sobrescribe ésta y fuerza a futuras *Request* a incluir un campo **Route** conteniendo la dirección del proxy que obligó a que ese proxy sea incluido.

El mecanismo descrito anteriormente inserta en el campo su propia URI o agrega la URI que ya tiene fijada en el **Record-Route** con el objetivo de que la URI final pueda resolver hacia atrás y así llegar al servidor proxy. El UAS copia el **Record-Route** en las ulteriores respuestas 200 OK a la *Request*, y este campo se reenviara inalterado por los proxy hacia la UAC. El UAC guardará la lista de proxy del **Record-Route** más el campo **Contact**, de estar presente en el mensaje 200 OK, para usar en posteriores *Request*. Dado que el campo **Record-Route** es bidireccional, los mensajes en dirección opuesta atravesarán la misma cadena de proxy que el mensaje de ida.

To

Es un campo requerido en cada mensaje SIP para identificar el receptor del *Request* efectuado. Cualquier respuesta generada por el UA contendrá este campo del encabezado en conjunto con el tag. Cualquier respuesta generada por un proxy deberá tener agregada un tag al campo del **To**, en las respuestas 200 OK. De agregar el tag será incorporado a través de la llamada y a lo largo de todo el dialogo.

Via

Campo necesario para grabar la ruta SIP que toma un *Request* y que es utilizada para enviar la respuesta a quien la originó. Cada UA que genera un *Request* graba su propia dirección en el campo **Via**. A pesar que muchos de los campos del mensaje SIP no son de mayor importancia, el campo **Via** es de extrema importancia debido a que es el indicado para efectuar los ruteos de las respuestas. Un proxy reenviando *Request* agrega el campo **Via** conteniendo su propia dirección en el top de la lista del campo **Via**. Además incluye un campo denominado **branch** conteniendo un hash criptográfico de los campos **To**, **From** y **Call-ID** además de la **Request-URI**.

Un proxy o UA genera una respuesta a una *Request* copiando todo el campo **Via** de la *Request* en el respectivo orden en la respuesta y luego envía esta respuesta a las direcciones especificadas en el campo **Via**, pero siempre apuntando a la dirección de más arriba. El proxy que recibe las respuestas chequea la dirección más alta del campo **Via** para verificar que tiene su propia dirección, luego de esto la dirección ubicada más arriba de este campo es removida y se reenvía la respuesta según lo especificado en el campo **Via**.

El campo contiene: el tipo de protocolo, el número de versión y tipo de transporte, además de número de puerto y parámetros como **received**, **rport**, **branch**, **maddr** y **ttl**.

El tag de tipo **received** es agregado al **Via** si el UA o el proxy reciben *Request* provenientes de direcciones diferentes a las especificadas en el top del campo **Via**. Si tag **rport** es incluido en el **Via** de un *Request* el proxy insertara el puerto del *Request* recibido y se usará para enrutar las respuestas.

2.5.2 – Campos exclusivos del mensaje Request

Accept-Contact

Especifica un campo donde las URIs del *Request* pueden ser pasadas por un proxy, es parte de las extensiones de SIP que han sido definidas para dar cierto control al llamador de forma que el proxy procese las llamadas.

Authorization

Este campo del encabezado se utiliza para enviar las credenciales de autenticación del UA en una *Request* al servidor. Este campo es enviado en respuesta a un **401 Unauthorized** pidiendo la información necesaria para realizar la autenticación.

Event

Se utiliza en mensajes del tipo **SUSCRIBE** o **NOTIFY** para indicar qué eventos de paquete están siendo usados por los métodos anteriores. En el caso de **SUSCRIBE** lista el evento de paquete al cual el usuario querrá suscribirse. En el caso de **NOTIFY** lista los eventos de paquete que la notificación contiene información de estado.

Proxy-Authorization

Campo donde se envían las credenciales del UA en el *Request* hacia un servidor, en general es enviado en respuesta a un **407 Proxy Authentication Required**, conteniendo información de autenticación, o puede ser enviado sin esperar el mensaje de autenticación requerida si ya se conocen las credenciales.

Max-Forwards

Se especifica la cantidad máxima de saltos que un *Request* de SIP puede realizar. Este valor se decrementa en cada proxy que reenvía el *Request*. Un valor de cero descartará el mensaje enviando un **483 Too Many Hops** a la fuente del mensaje.

Refer-To

Campo requerido en un *Request* del tipo **REFER** que contiene la URI o URL del recurso referenciado. Puede contener cualquier tipo de URI proveniente de sip, sips o tel, además de http, mailto URI.

Reply-To

Es para indicar a sip o sips URI que será usado para responder a esta *Request*, normalmente esta URI es presentada en el campo **From** (ya que el **Contact** no se usa y es solo valido durante el diálogo).

Route

Provee información de enrutamiento a los *Requests* enviados, la RFC-3261 introduce dos tipos de enrutamiento, *strict* y *loose*, que se asemejan a los modos de enrutamiento en IP con el mismo nombre.

En el caso del tipo de enrutamiento *strict* el proxy debe usar la primer URI del campo **Route** para reescribir la **Request-URI** a la que posteriormente será reenviada. En este tipo de enrutamiento la *Request* solo se enrutará a través de los servidores listados en el campo **Route** con la **Request-URI** siendo sobrescrita en cada salto.

El tipo de enrutamiento *loose* no permite que el proxy sobrescriba la *Request-URI* pero de todas formas reenvía la *Request* a la primer URI del campo **Route** o a otro elemento del tipo *loose routing*. En este tipo el *Request* se debe de enrutar por cada uno de los servidores listados en el campo **Route** antes de ser enrutado según lo que declara la **Request-URI**.

Session-Expires

Se especifica el tiempo de vida de la sesión, para poder extender este tiempo la UA envía re-**INVITE** o **UPDATE** con un nuevo **Session-Expires**. Al momento de alcanzar el valor declarado, el UA enviará un **BYE** y las llamadas de tipo *stateful* en el proxy destruirán la información de estado de la misma.

Target-Dialog

Este campo es utilizado para autenticar *Request*. Por fuera del dialogo SIP, un uso común de este campo es para los **REFER** provenientes por fuera del diálogo en escenarios de transferencias. El campo contiene el identificador de la otra parte del dialogo en que se incluye el **Call-ID**, y el *tag* remoto y local.

2.5.3 – Campos exclusivos del mensaje Response

Authentication-Info

Campo que se incluye en las respuestas conformando una mutua autenticación utilizando HTTP *digest*. Esto proviene de obtener una autenticación compartida donde el cliente provee una repetición conteniendo el campo **Authorization** o **www-Authenticate**. En el caso de autenticación mutua el servidor proveerá en el campo **Authentication-Info** o una respuesta en el parámetro *rspauth*. El *digest* de autenticación es obtenido a partir de un algoritmo usado por SIP.

Proxy-Authenticate

Es utilizado cuando se recibe un **407 Proxy Authentication Required**, pidiendo una autenticación proveniente de un proxy a un UAC. Contiene cual es la forma del pedido por lo que el UAC formulará las credenciales para ser enviadas en el campo **Proxy-Authorization** en las próximos *Requests*.

Unsupported

Campo usado para indicar si existen features que no son soportadas por el servidor. Este campo es utilizado en el caso de recibir mensajes del tipo **402 Bad Extension** conteniendo una *feature* que no está soportada pero listada en el campo *Require* de un *Request*.

2.6 – Establecimiento de las Sesiones Multimedia

El objetivo más importante para el uso de SIP es negociar el establecimiento de sesiones. Para lograr el cometido, SIP utiliza otro protocolo llamado SDP (*Session Description Protocol*), en el cual se describen los parámetros de la sesión de media a establecer. Se negocia información del tipo de media, tipo de códec a utilizar, tasa de bits, dirección IP y número de puerto que se utilizarán para la sesión de *media* a establecer.

Resumiendo, la negociación de sesiones de *media* constituyen el procedimiento necesario para intercambiar los datos y comenzar a transferir las sesiones RTP donde ira el media negociado.

Se Explicará el protocolo SDP y el modelo Offer/Answer.

2.6.1 – Protocolo SDP – Session Description Protocol

Se define el protocolo SDP en la RFC-2327, desarrollado por el grupo IETF MMUSIC, siendo más que nada una descripción sintáctica que un protocolo ya que no provee una capacidad total de negociación de media. La idea original de SDP era describir sesiones multicast sobre el *backbone* de Internet. En la actualidad la versión corriente de SDP está establecida en la RFC-4566, siendo compatible con la RFC original de SDP.

La información de la sesión de media que aporta y adiciona en SDP es:

- Dirección IP (IPv4 o IPv6)
- Perfil RTP
- Numero de puerto, utilizado para transporte ya sea UDP o TCP
- Tipo de media (audio, video, etc.)
- Tipo de codificación (PCM, Ley-A, Ley-U, MPEG II, etc.)
- Tema de la sesión
- Tiempos de comienzo y finalización
- Información del contacto sobre la sesión

Como el protocolo SIP, SDP utiliza texto codificado, siendo éste compuesto por una serie de líneas denominadas campos cuyos nombres son abreviados por una única letra en minúscula, requeridas en un específico orden para simplificar el parseo. Estas abreviaciones se describen en la siguiente tabla con el orden requerido en SDP. Existe la posibilidad de evitar los campos opcionales, pero se debe mantener el orden.

Campo	Nombre	Obligatorio(m)/Opcional (o)
Descripción de sesión		
v=	Protocol version number	m
o=	Owner/creator and session identifier	m
s=	Session name	m
i=	Session information	o
u=	Uniform Resource Identifier	o
e=	E-mail address	o
p=	Phone number	o
c=	Connection information	m
b=	Bandwidth information	o
Descripción temporal		
t=	Timer session start and stops	m
r=	Repeat times	o
Descripción del tipo de media		
z=	Time zone corrections	o
k=	Encryption key (deprecated)	o
a=	Attribute lines	o
m=	Media information	o
a=	Media attributes	o

Tabla 2.1 – Campos en mensaje SDP

SDP no se caracteriza por ser un protocolo extensible, en efecto existen reglas de análisis sintáctico estrictas. La única forma de poder extender o dar nuevas capacidades a SDP es definiendo nuevos tipos de atributos. Un analizador sintáctico SDP no puede ignorar un campo desconocido, un campo obligatorio perdido o una línea fuera de secuencia.

A modo de ejemplo exponemos una traza SIP completa, se puede observar que en el cuerpo del mensaje (por debajo de los campos del encabezado SIP) se encuentran los códigos detallados del mensaje SDP. Ahí se puede observar los diferentes códecs que soporta el softphone utilizado.

```

INVITE sip:1002@192.168.2.11 SIP/2.0
Call-ID: dba2bfc5d84cb3b76b807daad4e873a7@0:0:0:0:0:0:0
CSeq: 2 INVITE
From: "1007" <sip:1007@192.168.2.11>;tag=3222d948
To: <sip:1002@192.168.2.11>
Max-Forwards: 70
Contact: "1007" <sip:1007@192.168.2.14:5063;transport=udp;registering_acc=192_168_2_11>
User-Agent: Jitsi1.1-nightly.build.4027Linux
Content-Type: application/sdp
Via: SIP/2.0/UDP 192.168.2.14:5063;branch=z9hG4bK-383839-fb5f3b38ddc39ec7f0872727b958ebb9
Proxy-Authorization: Digest username="1007",realm="192.168.2.11",nonce="86b43b78-a605-11e1-a8fb-513527209231",uri="sip:1002@192.168.2.11",response="4b61a360a3ef13781a7a11362555ed81",algorithm=MD5,qop=auth,cnonce="xyz",nc=00000001
Content-Length: 1049

v=0
o=1007 0 0 IN IP4 192.168.2.14
s=-
c=IN IP4 192.168.2.14
t=0 0
m=audio 5075 RTP/AVP 9 96 97 98 100 0 8 102 3 103 5 6 4 101
a=rtpmap:9 G722/8000
a=rtpmap:96 SILK/24000
a=rtpmap:97 SILK/16000
a=rtpmap:98 speex/32000
a=rtpmap:100 speex/16000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:102 iLBC/8000
a=rtpmap:3 GSM/8000
a=rtpmap:103 speex/8000
a=rtpmap:5 DVI4/8000
a=rtpmap:6 DVI4/16000
a=rtpmap:4 G723/8000
a=fmtp:4 annexa=no;bitrate=6.3
a=rtpmap:101 telephone-event/8000
a=extmap:1 urn:ietf:params:rtp-hdrext:csrc-audio-level
a=zrtp-hash:1.10 9d60f31ada562c4fad1d0525de231ffa0571fc9274bbca61d8bbccdc8c368f48
m=video 5077 RTP/AVP 104 99
a=recvonly
a=rtpmap:104 H264/90000
a=fmtp:104 profile-level-id=4DE01f;packetization-mode=1
a=imageattr:104 send [x=[0-640],y=[0-480]] recv [x=[0-1366],y=[0-768]]
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=4DE01f
a=imageattr:99 send [x=[0-640],y=[0-480]] recv [x=[0-1366],y=[0-768]]
a=zrtp-hash:1.10 6f9513ad5ca598a18d0ea96aafef3a47fd3de2301f9baa6f439af25566b0ac30

```

2.6.2 – El modelo Offer-Answer

La utilización de SDP por parte de SIP está dada en el modelo SDP offer answer de la RFC-3264, el cuerpo de mensaje por defecto en SIP es **application/sdp**. La parte llamadora lista las diferentes capacidades de media que puede recibir el SDP, usualmente en un **INVITE** o en un **ACK**. La parte llamada lista sus capacidades en mensajes del tipo **200 OK** en respuesta al **INVITE** enviado. Generalmente los offer o answer serán incluidos en mensajes del tipo *Request* con los métodos **INVITE**, **PRACK** o **UPDATE**.

SIP utiliza la conexión, media y los campos para establecer las sesiones entre las diferentes UA. Debido a la limitación del campo de origen en SIP, usualmente el campo **session-id** es mantenido constante a lo largo de una sesión establecida y el campo **version** es incrementado en cada momento que el SDP

es cambiado. Si el SDP se mantiene inalterado desde el momento que se envió, el campo **version** no se alterará.

A raíz de que los tipos de sesión de media y códec utilizados son empleados como parte de la negociación de conexión, SIP utiliza a SDP para especificar múltiples alternativas de tipos de media y de forma selectiva aceptar o rechazar los tipos de media que elija. El modelo offer answer recomienda establecer el atributo **a=rtpmap**: utilizado para cada campo de media. Un flujo de media es rechazado fijando el número de puerto a valor cero en el campo correspondiente al tipo de media en la respuesta SDP.

2.6.2.1 – Reglas para generar un Offer

Una Offer SDP debe incluir todos los campos SDP requeridos (incluye **v=**, **o=**, **s=**, **c=** y **t=**), usualmente incluye el campo de media (**m=**) pero puede no contenerlo. Las líneas de media contienen todos los códecs listados por orden de preferencia, la única excepción es si el punto de finalización soporta una cantidad considerable de códecs, la mayoría a ser aceptados o la mayoría deberían estar listados. Los tipos diferentes de media incluyen audio, video, texto, MSRP, BFCP, etc.

2.6.2.2 – Reglas para generar un Answer

El SDP Answer, a diferencia de un Offer, debe estar construido según las reglas que se detallarán y debe tener el mismo número líneas **m=** en igual orden que el SDP offer. Los flujos de media individual pueden ser rechazados estableciendo el número de puerto a cero, estos a su vez son aceptados enviando un valor diferente a cero en el número de puerto. Los *payload* listados para cada tipo de media deben ser un subconjunto de *payloads* listados en el offer. Notar que para *payload* dinámicos no tiene por qué ser utilizado el mismo número en cada dirección. En general un único *payload* es elegido y más de uno es seleccionado pero las puntas serán capaces de cambiar de forma dinámica entre estos sin señalarlo. Dado que muchas de estas puntas podrán tener un único códec corriendo al mismo tiempo, el comportamiento descrito anteriormente podrá ser evitado. Una excepción en concreto es aceptar códecs de media y también eventos telefónicos, permitiendo usar tonos DTMF.

2.6.2.3 – Reglas para modificar la sesión

Cada parte puede iniciar un intercambio offer answer para modificar la sesión. Una vez que ésta es cambiada se deben continuar las reglas que se listarán a continuación. La línea del numero de versión de origen (**o=**) debe mantenerse inalterada respecto a la última enviada que indica si el SDP es idéntico al cambio previamente efectuado, o deberá de ser incrementado por uno, que indica que un nuevo SDP será analizado sintácticamente. El offer deberá incluir las líneas de media existentes y deberán ser enviadas en el mismo orden, además de agregar media adicional al final de la línea con **m=**. Se pueden borrar flujos de media fijando el número de puerto a cero. Este tipo de media deberá mantenerse en el SDP para futuros intercambios offer answer de esta sesión; para un flujo de media existente cualquier aspecto de los citados anteriormente permite ser cambiado.

2.6.2.4 – Llamada en Espera

El caso se detalla en particular, cuando una parte deja en espera a la otra suspendiéndose el envío de paquetes de media. Esto se realiza enviando un **INVITE** con un SDP idéntico al original enviado en el primer **INVITE** pero con el detalle de un campo fijado como **sendonly=s**. La llamada volverá a estar activa con otro **INVITE** con el campo fijado en **a=sendrecv**.

2.7 – RTP – Real-Time Transport Protocol

2.7.1 – Descripción y Funcionamiento

El protocolo RTP se desarrolló con el objetivo de transportar datagramas conteniendo voz, video u otro tipo de información sobre IP en tiempo real. Las primeras versiones de RTP funcionaron para transportar voz sobre redes del tipo multicast. Tanto H.323 como SIP utilizan RTP para transportar media convirtiéndolo en el estándar más común para las comunicaciones de Internet.

Se define según la IETF en la RFC-3550, actualización de la RFC original RFC-1889. RTP no provee calidad de servicio alguna sobre la red IP, en concreto los paquetes RTP son manejados de igual forma que todos los demás paquetes en la

red IP, de todas formas RTP permite la detección de algunos impedimentos introducidos por las redes IP.

Estos son:

- Pérdida de paquetes
- Demora variable de transporte
- Llegada de paquetes fuera de secuencia
- Enrutamiento asimétrico

El protocolo RTP se ajusta al procesamiento de media, mediante las siguientes características a resaltar.

Codificación: los pasos de codificación implican conversión analógica a digital (A/D) que es implementada por filtros pasabajos y luego seguido de muestreo. Dependiendo del códec utilizado, se manejan diferentes cantidades de bits por muestra además del algoritmo que implementa. El códec utilizado es transportado por el RTP en el tipo del *payload*, la frecuencia de muestreo es negociada en el intercambio offer/answer de SDP al negociar la sesión de media.

Empaquetado: el empaquetado incluye fragmentar las muestras en datagramas individuales para el transporte. El tamaño del paquete está basado en la negociación entre la demora del empaquetado (cuantos intervalos de muestreo deben realizarse antes de que los datos estén en el datagrama) y la eficiencia de transporte.

Transporte: el tipo de transporte es en tiempo real, lo que requiere una latencia mínima de la red, no hay tiempo para detectar la pérdida de paquetes, pérdida de señal o espera para retransmisión. En consecuencia no se utiliza transporte sobre TCP, por lo tanto se utiliza transporte sobre UDP, es por esta razón que muchos datagramas llegan en desorden o incluso fuera de secuencia.

Desempaquetado: es el procedimiento por el que se remueve el encabezado RTP del *payload* a excepción del códec.

Buffering: implica los procesos de guardado de las muestras del códec antes de comenzar la transmisión del audio. La elección del tamaño del *buffer* es crítica para la calidad de audio ofrecida, un buffer pequeño resultará en un audio de calidad pobre, pero un *buffer* muy largo introducirá latencias inadecuadas.

Decodificación: implica enviar paquetes del códec a su algoritmo; este códec es elegido basado en el *header* del RTP.

Playback: este paso implica la reproducción del flujo de media al usuario, ya sea audio, video, o texto.

2.7.2 – Códecs

Se listarán algunos de los códecs más comúnmente utilizados en RTP para la codificación de audio. Esto pretende ser una aproximación de las diferentes opciones al momento de negociar el tipo de codificación de audio y la calidad de audio lograda en las sesiones.

PCMU y PCMA

Son recomendaciones de la ITU-T establecida en la G.711, el audio es codificado en 8 bits por muestra luego de un escalamiento logarítmico. Las diferencias son que PCMU maneja ley-U y PCMA maneja ley-A. En ambos casos la tasa de bit alcanzada es de 64kbps.

GSM – Group Speciale Mobile

Es un estándar europeo, precisamente es el GSM 06.10 para transcodificación en tasa completa que se basa en codificación RPE/LTP (*Residual Pulse Exitation/Long Term Prediction*) con una tasa de 13kbps.

G.722

Especificado en la recomendación de la ITU-T G.722 [20], maneja a una tasa de muestreo de 7kHz con una tasa 48, 56 y 64kbps. La actual tasa de muestreo de este códec es de 16kHz utilizando 14 bits, pero debido a que la frecuencia de reloj de RTP es de 8kHz, dado que fue erróneamente asignada en la RFC-1890 y debe mantenerse inalterada para otorgar retrocompatibilidad, por lo que la tasa se fija en 8kHz. Las versiones G.722.1 (Siren7) o G.722.2 (AMR-WB) son para casos donde se precisan códecs más eficientes a nivel de bit. G.722.1C. Es una extensión de G.722.1 de baja complejidad que maneja otro algoritmo que permite calidades de audio con un ancho de banda de 14kHz con una tasa de muestreo de 32kHz y con 24, 32 y 48kbps.

G.723

Es un estándar de la ITU-T [21], provee calidad de voz desde los 300Hz hasta 3400Hz utilizando ADPCM desde 24 a 40kbps. El estándar G.723 es obsoleto y es sustituido por el G.726. La versión G.723.1 transmite en dos canales que son 5.3 y 6.3kbps y es utilizado para la transmisión de comunicaciones multimedia.

G.726

Es un estándar de la ITU-T [22] que cubre transmisiones de voz a tasas de 16, 24, 32 y 40kbps. Es introducido para sustituir a G.721 y G.723. Las tasas de bits asociadas del estándar se relacionan con el tamaño de bit de las muestras que son 2 bits, 3 bits, 4 bits y 5 bits respectivamente. La tasa de muestreo a la que opera es 8kHz.

G.729

Es especificado en la recomendación de la ITU-T G.729 [23]. Provee una calidad de audio a una tasa de bits baja, 8kbps, pero es un códec un tanto costoso en términos del consumo de procesamiento por parte del CPU. Utiliza CS-ACELP. Debido a su bajo ancho de banda es preferentemente utilizado en redes de VoIP donde el ancho de banda debe conservarse (como conferencias). Existe dentro de este estándar otras especificaciones como el caso de G.729a, que requiere menor costo de procesamiento ya que no tiene tanta complejidad en detrimento de la calidad del audio. Maneja tasas de muestreo del orden de los 8kHz con 16 bits (80 muestras en tramas de 10 ms). El G.729b provee compresión de silencio, habilitando VAD (voice activity detection) es utilizado para detectar señal por actividad de voz.

LPC-10 (FS-1015)

Es una codificación segura para comunicaciones de audio desarrollado por el Departamento de Defensa de los Estados Unidos y luego por la OTAN. Utiliza un algoritmo lineal predictivo. El códec utiliza una tasa de bits de 2.4kbps, la versión mejorada de este códec introduce tasas de bits del orden de los 800bps.

DVI4

Utiliza un pulso adaptativo de modulación (ADPCM). Algunos aspectos de este códec son que las muestras de 4 bits son paquetizadas con la primera muestra de los cuatro bits más significativos y la segunda muestra de los cuatro bits menos significativos. Cada paquete contiene un único bloque DVI. Maneja una tasa de bit de 32kbps.

SILK

Sistema de compresión de audio y códec de audio usado por Skype y desarrollado por Skype. Permite utilizar frecuencias de muestreo de 8, 12, 16 o 24kHz y tasas de bits desde los 6 a 40kbps. Además utiliza una baja demora de algoritmo de 25 ms. La tecnología de códec es del tipo de predicción lineal.

iLBC

Es la abreviación de Internet Low Bit Rate Codec. El códec emplea un algoritmo denominado iLBC y soporta dos tipos de longitud de tramas, una de 20ms a 15.2kbps y 30ms a 13.33kbps. Cuando el códec opera con bloques de 20ms produce 304 bits por bloque, cuando opera a 30ms produce 400 bit por bloque. Maneja tasa de muestreo de 8kHz a 16 bits. Controla la respuesta a perdida de paquetes, demora y jitter. Tiene una robustez similar a PCM y la carga del CPU es similar a G.729 con mayor calidad y mejor respuesta frente a perdida de paquetes [24] [25].

Speex

Codec basado en CELP (Code Exited Linear Prediction), técnica que soporta tanto banda angosta con una tasa de muestreo nominal de 8kHz y banda ancha con una tasa de muestreo nominal de 16kHz o ultra banda ancha en el entorno de los 32kHz. La flexibilidad apunta a que permite integrar tanto banda ancha como banda angosta en un mismo flujo de bits, soporta un espectro amplio de tasa de bits además de manejar *switcheo* dinámico de tasa de bits y tasa de bits variable VBR. Soporta tasas de bits desde los 2.15kbps a los 44kbps, depende del tipo de procesador si puede incluir todo el espectro de tasas. La especificación define un solo canal, por lo tanto solo soporta mono [26].

Vorbis

Códec de audio de propósito general que intenta aprovechar las capacidades de la codificación al máximo, esto permite escalar en una cantidad considerable de tasa de bits. Se toma como la mejor calidad de la escala la de tipo de CD o DAT para estéreo en los 16/24 bits. También intenta para bajas y altas tasas de muestreo desde los 8kHz de telefonía hasta los 192kHz digitales y presenta el manejo de múltiples canales desde monoaural, polifónico, estero, cuadrafónico, 5.1 y ambisonic, hasta 255 canales discretos.

Vorbis codifica audio encapsulado según el formato Ogg de la recomendación RFC-3533 que provee entramado y sincronización [27].

MP3

Estándar denominado a partir del MPEG-1 o MPEG-2 Audio Layer III, con codificación patentada para codificar en formato de audio digital utilizando un formato de compresión por pérdida. Se manejan diferentes tasas de bit que van desde los 32 a 320kbps con frecuencias de muestreo de 32, 44.1 y 48kHz en MPEG-1. Para MPEG-2 se manejan tasas desde los 8 a 160kbps con frecuencias de muestreo de 16, 22.05 y 24kbps [28].

CELT

De la abreviación de Constrained Energy Lapped Transform, es un compresor de audio con baja demora de algoritmo para usar en comunicaciones de baja latencia. Resuelve la brecha existente entre Vorbis y Speex donde alta calidad y baja demora son requeridas. Soporta tanto mono y estéreo tanto en dialogo como en música, utilizando tasas de muestreo desde los 32kHz a los 48kHz con tasas de bit adaptativas desde los 24kbps a los 128kbps por canal.

Opus

Algoritmo de compresión de audio con pérdida desarrollado por la IETF y estandarizado en la RFC-6716. Incorpora tecnología de dos códecs existentes. SILK que es un códec orientado a dialogo y el códec CELT de baja latencia. Opus permite escalar de bajas a altas tasas de bits y utiliza predicción lineal para bajas tasas de bits así como también transforma códecs de alta tasa de bits. Soporta tasas de bits desde los 6 a 510kbps y tasas de muestreo que van desde los 8kHz a

los 48kHz permitiendo reproducir todo el rango audible del ser humano. Dada la baja latencia de 20 ms es ideal para aplicaciones de VoIP [29].

Capítulo 3 – Investigación

Una de las tareas importantes a efectuar para alcanzar el resultado del proyecto es encontrar las piezas necesarias de software, que en conjunto, provean las funcionalidades y cumplan con los criterios de éxito establecidos.

En este capítulo se interioriza y analiza las diferentes ofertas existentes en el mercado de las herramientas basadas en *Open Source* para proveer funcionalidades corporativas, así como la revisión en materia de SIP servers, sistemas fundamentales para poder alcanzar altos niveles de usuarios en el sistema. Se realizará una revisión de los diferentes *softphones* que puedan operar con las funcionalidades requeridas. También se hará mención a la herramienta generadora de tráfico la cual permite realizar tanto pruebas conceptuales como pruebas de capacidad de los equipos utilizados, así como también el funcionamiento de los diferentes sistemas utilizados.

Para finalizar, en base a la revisión se tomará una decisión sobre cuáles serán las herramientas adecuadas para poder continuar con el desarrollo del proyecto.

3.1 – Revisión de Software SIP del Mercado Actual y Preselección

Es necesario hacer una evaluación de cada uno de los proyectos que se han desarrollado bajo el marco de licenciamiento *Open Source*, para conocer si se ajustan a las necesidades de los requerimientos establecidos. Es por esa razón que evaluaremos los proyectos de sistemas *Back to Back User Agent* (B2BUA) para proveer funcionalidades, sistemas que nos provean servicios de SIP Proxy/Registrar y los *softphones* que pueden ser utilizados con estas dos herramientas seleccionadas.

3.1.1 – B2BUA: Back to Back User Agents

Se evalúan los principales proyectos en materia de IP PBX que nos provean de funcionalidades tales como correo de voz, conferencias, música en espera. Esta evaluación comprende los tipos de códecs soportados con sus respectivas tasas de bit y qué calidades manejan a nivel de audio. Se evaluará que cuenten con una comunidad de desarrollo sólida que garantice que los sistemas puedan disponer de una constante actualización y resolución de incidencias. Con este enfoque se trata de minimizar el riesgo que se utilicen proyectos que queden desafectados o sin proyecciones a futuro. Por lo tanto, el sistema utilizado contará con comunidades que pujen hacia nuevas versiones y actualizaciones, además que cuenten con los canales adecuados para obtener el soporte correspondiente.

3.1.1.1 – Asterisk

Es un proyecto de comunicaciones basado en sistemas *Open Source* capaz de proveer servicios de telefonía VoIP permitiendo un amplio espectro de servicios los cuales incluyen IP PBX, *Gateway* de VoIP, *callcenter* y sistemas IVR. Lanzado bajo los términos de la *GNU General Public License (GPL)*, permite ser descargado y utilizado libremente. Se considera al proyecto *Asterisk* como pionero en dar crecimiento a las redes de telefonía de VoIP. Las primeras versiones de este sistema contaban con una arquitectura del sistema enteramente monolítica. Paulatinamente los requerimientos fomentaron adecuar la arquitectura a una modular, tal es así que se planteó un marco de investigación y desarrollo por parte de la empresa *Digium* para cubrir ciertos aspectos que la arquitectura monolítica no cubría, como la alta disponibilidad y la modularidad. Al momento de la realización de este proyecto la versión 1.8 y 10 de *Asterisk* cuentan con una modularización avanzada en materia de arquitectura [30] [31] [32] [33] [34].

Particularmente *Asterisk* maneja los protocolos necesarios para lograr establecer comunicaciones sobre VoIP además de mantener la compatibilidad con las redes PSTN. Dentro de su operativa mantiene una base de datos integrada para acceder a información en base de datos relacionales e integración con servicios LDAP. Contiene funcionalidades de gestión y mantenimiento de CDR para la contabilización y facturación de llamadas. En los avances que denotan en este sistema, las nuevas versiones migran el motor de base de datos de tipo Berkeley a SQLite.

Dentro de los protocolos soportados incluye el manejo de redes IP y dar soporte a redes tradicionales, entre estos se encuentran SIP, H.323 e IAX, siendo este último un protocolo propietario para intercomunicar diferentes servidores *Asterisk* así como también que dos UA puedan comunicarse. Soporta los códecs ADPCM, G.711 (ley-A y ley-u), G.722, G.723, G.726, G.729a, GSM, iLBC, Linear, LPC-10 y Speex.

A nivel de funcionalidades corporativas provee, *voicemail*, *accounting* básico que permite ser expandible mediante una interfaz de datos ODBC, *billing*, conferencias, IVR, colas de espera y distribución automática de llamadas. La capacidad del servidor *Asterisk* se ajusta al tipo de diseño del mismo, teniendo una escalabilidad a múltiples servidores.

Por sus características detalladas anteriormente y la comunidad existente en torno al proyecto se considera a *Asterisk* dentro de las IP PBX a ser posiblemente utilizadas para officar de media server.

3.1.1.2 – Elastix

Este proyecto propone un software de código abierto distribuido bajo la licencia GPLv2 para el establecimiento de comunicaciones unificadas. Asegura incorporar en una única solución todos los medios y alternativas que existen en el mercado empresarial. En sus orígenes apuntó a ser una herramienta para efectuar reportes de llamadas en *Asterisk*, pero posteriormente a su liberación evolucionó a una distribución con una instalación de *Asterisk* administrable vía servicios web.

La versión actual de *Elastix* es la 2.3.0, específicamente es un sistema operativo CentOS 5.7 en el cual se ejecuta un *Asterisk* 1.8.11, siendo un sistema con una arquitectura modular [35]. La distribución, además del sistema operativo, instala una IP PBX operativa, permitiendo gestionarla de forma simple vía una interfaz web donde permite establecer permisos de acceso y diferentes funcionalidades para las extensiones definidas. A nivel administrativo, dentro de la interfaz, se encuentran módulos de reportes de uso y carga del hardware del equipo. Como la PBX instalada es un *Asterisk* maneja los protocolos SIP, IAX, H.323 y XMPP [36]. La capacidad del sistema está estrictamente relacionada con el tipo de diseño que se obtenga del servidor y la escalabilidad depende de la cantidad de servidores que se asignen para operar. Los tipos de códecs soportados son ADPCM, G.711 (ley-A y ley-u), G.722, G.723, G.726, G.729a, GSM, iLBC, Linear, LPC-10 y Speex al igual que *Asterisk*.

Elastix apunta a un sector del mercado que demanda funcionalidades corporativas que se ajusten a la medida de las necesidades de la empresa, por esa razón el equipo de desarrollo mantiene un estricto relacionamiento con sus clientes para desarrollar funcionalidades que se ajusten a las necesidades de éstos.

A efectos de la revisión efectuada *Elastix* es una distribución con un *Asterisk*, permite una cantidad de funcionalidades extra, entre estas la administración y gestión, que apunta a que el usuario común pueda realizarlo. Debido a que la instalación integra otras instalaciones que son accesorias al sistema SIP es de interés resaltar que éstas insumen recursos físicos del equipo perjudicando la performance de la IP PBX. A efectos de los objetivos del proyecto *Elastix* no es la solución adecuada para los fines buscados.

3.1.1.3 – FreeSWITCH

Es una plataforma de comunicaciones basada en *Open Source* para proveer servicios de telefonía lanzada bajo la licencia MPL 1.1. Al momento de lanzar el proyecto al mercado existían varias opciones para dar este tipo de soluciones. Entre estos, *Asterisk* se había convertido en la herramienta más popular en la comunidad de IP PBX.

El diseño de software de *FreeSWITCH* toma como punto de partida un conjunto considerable de librerías de software libre existentes, permitiendo así una arquitectura de software modular y extensible, limitado solamente a las necesidades funcionales del *core*, incluso permite utilizar módulos y agregar de forma virtual funcionalidades deseadas por el usuario.

La implementación de *FreeSWITCH* es un *core* de librerías denominado *libfreeswitch*, con la posibilidad de transformar desde un *softphone* a PBX o un soft-switch. El sistema modular permite dar funcionalidades extra al sistema. Entre los lenguajes para la creación de módulos se manejan C/C++, Java, .NET, Javascript/EMAScript, Python, Perl, Ruby, PHP y Lua. Los sistemas externos pueden recibir eventos desde el switch de control sobre TCP, con adaptación de lenguajes y clientes. Dentro de las características a resaltar de *FreeSWITCH* está la posibilidad de manejar una cantidad de canales concurrentes con media en un solo PC estándar, convirtiéndolo así en un software escalable. La compatibilidad con otros sistemas es asegurada, permitiendo a *FreeSWITCH* interactuar con otros proyectos del mismo estilo.

A nivel de servicios, se enfoca en prestar voicemail con notificación *MWI* y vía correo electrónico, salas de conferencia (puente de conferencias o *conference bridge*), música en espera (MOH), servicios de IVR y anuncio, además de poder trabajar como servidor de ruteo. El sistema apunta a proveer diferentes calidades de audio tanto a nivel de conferencias como a nivel de llamadas planas, soportando una gran variedad de códecs, entre estos, G.722.1, G.722.1C, G.722, G.711, G.726, G.723.1, G.729ab, AMR, iLBC, Speex en banda angosta y banda ancha, CELT (32kHz y 48kHz), Opus, LPC-10, SILK y además soporta códecs de video (H.261, H.263, H.264 y MP4).

Del análisis realizado se valora fuertemente que el sistema sea modular permitiendo una separación clara de las funcionalidades a proveer, la flexibilidad y amplio espectro de códecs que soporta, además de una comunidad de desarrollo rica en ideas para mejorar la arquitectura del sistema y lanzar versiones con actualizaciones y correcciones en el código, y una masa crítica considerable de usuarios del sistema que cada vez es más utilizado en diferentes ámbitos a nivel comercial. Se considera a *FreeSWITCH* como un fuerte candidato para officiar de media server.

3.1.1.4 – SipXecs

Sistema modular escalable de comunicaciones para otorgar soluciones empresariales del tamaño necesario. Provee un *core* SIP de alta disponibilidad integrado con una creciente variedad de servicios de comunicaciones que pueden ser administradas vía web. La solución también provee las funcionalidades básicas de una PBX, incluye voicemail, mensajes unificados, conferencias y servicios de presencia y callcenter. Tiene la capacidad de servir para sistemas de pequeño y mediano porte o puede ser utilizado en sistemas distribuidos con redundancia de gran porte. Es lanzado bajo la licencia AGPL/LGPL.

Administrativamente maneja Web Services, permitiendo una administración centralizada de un sistema distribuido, además de ofrecer plug & play de teléfonos y gateways. En materia de códecs soporta G.711 (ley-A y ley-u), G.729 y G.722 Wideband.

A diferencia de las soluciones existentes en el mercado, *SipXecs* ofrece un sistema unificado de comunicaciones, no solo una IP PBX.

El proyecto *SipXecs* presenta una alternativa atractiva al momento de evaluar como posible software para media server. Se descarta debido a que para alcanzar el máximo de funcionalidades y capacidades precisa de una versión paga. Esto escapa al marco de trabajo del proyecto, que se ha planteado para utilizar software basado en *Open Source*.

3.1.1.5 – YATE – Yet Another Telephony Engine

Dado que *YATE* es una solución integrada, se clasifica como posible PBX así como también como posible SIP Server. Su fortaleza radica en la posibilidad y sencillez de ser extendido, tanto para voz, video, datos y mensajería instantánea que permiten ser unificadas bajo el sistema de enrutamiento flexible de *YATE*. Es lanzado bajo la licencia GPL.

La arquitectura modular de *YATE* se subdivide en cuatro partes. El core o núcleo donde se realizan las encapsulaciones, threads y funciones primitivas; el sistema de mensajes donde se realiza el intercambio de datos entre los módulos; el sistema de telefonía; y el modulo Yate que extiende las funcionalidades del sistema analizado, que no está precisamente relacionado con los módulos telefónicos.

Dentro de las funcionalidades que otorga *YATE*, permite ser utilizado como servidor y cliente de VoIP, utilizarse como servidor de conferencias y actuar de servidor de telefonía IP.

Soporta los protocolos H.323, IAX, SIP, Jingle para mensajería instantánea y RTP; a nivel de códecs maneja G.711, GSM, e iLBC.

El software está escrito en C++ de forma modular y soporta scripts escritos en diferentes tipos de lenguajes (PHP, Python y librerías de Perl, Unix Shell), las tres primeras son útiles para desarrollar funcionalidades externas a *YATE*.

Si bien el potencial de este sistema es sólido, no consideramos esta solución ya que no cuenta con una cantidad suficiente de usuarios como para generar una masa crítica de estos que provean una comunidad nutrida de desarrollo y de desarrolladores. Por esta razón lo damos por descartado.

3.1.1.6 – CallWeaver

Proyecto que parte del desarrollo original de *Asterisk* con la idea de crear un código diferente al original. Entre los avances propuestos por el desarrollo es migrar la base de datos original Berkeley DB 1.0 a un SQLite 3, diseñado sobre arquitectura modular. Soporta códecs del tipo G.723 y G.729, respecto a iLBC solo es posible utilizarlo si se compila al momento de la instalación por parte del usuario [37] [38]. El proyecto es formalmente conocido como *OpenPBX*. No se tienen referencias técnicas de este proyecto hasta el momento, además la última versión es del año 2009.

Podría afirmarse que *CallWeaver* es un desarrollo similar a otros proyectos que partieron de lo existente para realizar algo diferente. Como se describe en el párrafo anterior, la última versión data de hace tres años atrás, no adecuándose al estado actual de las tecnologías de VoIP, razón más que suficiente para descartar este proyecto.

3.1.1.7 – GNU SIP Witch

Es un servidor VoIP peer-to-peer que utiliza el protocolo SIP, permite procesar llamadas directas de Internet además de interceptar audio y video peer-to-peer. Basado en el proyecto GNU, está diseñado para ser utilizado multiplataforma, lo que permite flexibilidad. Parte de la misma está garantizada ya que está escrito en lenguaje C++. Es lanzado bajo la licencia GNU GPL.

Soporta funcionalidades como reenvío de llamadas, distribución de llamadas, llamada en espera, información de presencia, mensajes y además llamadas encriptadas. Dentro de este proyecto se encuentra el servidor de GNU Telephony, denominado GNU Bayonne [39].

La misma documentación específica que este software no efectúa negociaciones del códec de audio, solo se encarga de realizar la negociación de establecimiento de sesión entre las puntas de la comunicación. La posibilidad de negociar los diferentes códecs implica un común acuerdo entre las partes para obtener una calidad de sonido superior o inferior, de acuerdo a las capacidades de cada uno. Por otra parte, la documentación existente sobre este proyecto es poco precisa y sumado a que no tiene capacidad de proveer funcionalidades como conferencias o voicemail, no es un proyecto que sea posible para los objetivos del proyecto.

3.1.1.8 – SEMS – SIP Express Media Server

Es un software para dar capacidad de SIP Proxy/Registrar, tiene la capacidad de atender llamadas, lanzado bajo la licencia GPLv2. Se enfoca en proveer servicios de voicemail con la particularidad de envío a correo electrónico y MWI, conferencias del tipo dial-in y con control vía web, a la vez de poder dar velocidad y confiabilidad. Soporta los códecs G.711, GSM, iLBC, speex, G.726 y G.722 [40].

Se implementa para que sólo opere con el protocolo SIP y que no sea multiprotocolo, mantiene una granularidad fina a nivel de control, implementado en C++ y python. SEMS maneja una arquitectura del tipo modular para incrementar la capacidad del sistema [41].

A nivel de gestión y administración posee una GUI en la cual se puede monitorear las llamadas que se están cursando, recargar configuraciones y permite la integración con otras aplicaciones de gestión de otros sistemas.

La idea de SEMS es agregar funcionalidades corporativas al núcleo de señalización creado por SER, lo cual provoca una relación bastante fuerte entre núcleo SER y media server SEMS.

Nuevamente estamos frente a una solución que presenta matices de robustez en sus implementaciones, pero que al momento de considerar la comunidad de desarrollo y la masa de usuarios de este sistema, no alcanza niveles como otras soluciones ya analizadas, por esto, damos por descartada esta solución.

3.1.1.9 – Mobicents

Plataforma de VoIP Open Source conformada por subproyectos basados en JSLEE, JAIN Service Logic Execution Environment, una API de lenguaje de programación de Java y servidores JBoss o Apache Tomcat. Lanzado bajo la licencia LGPL. Es un sistema altamente modularizado que permite vincular diferentes tipos de servicios [42].

Mobicents Media Server, soporta códecs del tipo G.711 (ley-A y ley-u), GSM, Speex y G.729 sujeto a licencia. Provee servicios de IVR y detección de DTMF, además de proveer un punto de acceso de conferencias [43]. El Mobicents SIP Presence Server provee servicios de presencia en una red SIP [44].

El proyecto Mobicents, que aglomera mas subproyectos, apunta a desarrollos que utilicen la plataforma creada para servicios SIP y dar servicios agregados, no apunta a una PBX que pueda proveer funcionalidades corporativas como las conferencias, voicemail, etc. Dado el enfoque que tiene este proyecto no está dentro de las posibles soluciones a ser utilizadas para desarrollar Obelix, por tanto damos al proyecto Mobicents por descartado.

3.1.1.10 – Cipango

Una extensión de VoIP al Jetty HTTP Servlets container, es uno de los tres SIP Servlets container basados en Open Source que cumplen con los estándares de SIP Servlets 1.1. Es lanzado bajo la licencia Apache Licence Version 2.0 [45]. El sistema está implementado bajo la arquitectura Jetty [46].

Provee funcionalidades como presencia, reenvío de llamadas, conferencias, mensajería instantánea y capacidad de contact-center. Además provee un subsistema llamado kaleo que actúa de proxy/registrar y servidor de presencia. No negocia códecs, sólo negocia señalización.

Nuevamente, el proyecto Cipango ofrece una plataforma de SIP Servlets que permite crear servicios de valor agregado sobre SIP. Descartamos este sistema ya que no apunta a una IP PBX necesaria para cumplir los objetivos de proyecto.

3.1.1.11 – SailFin

Proyecto basado en la tecnología robusta y escalable SIP Servlets Technology aportada por Ericsson, actualmente provee compatibilidad con la JSR 289, siendo de alta disponibilidad y capacidad de conformar clúster, basado en arquitectura de tipo modular. Estas tecnologías aportan alta disponibilidad al SIP Servlet Container y también balanceo de carga. Está lanzado bajo dos licencias, CDDLv1 y GPLv2.

Permite ser ejecutado en varias plataformas, como es el caso de Solaris, Linux, Mac OS y Windows.

La herramienta provee soporte comercial bajo el nombre de “Sun GlassFish Communications Server” siendo la empresa Sun la que provee este servicio.

SailFin propone utilizar la SIP Servlets Technology para dar servicios de valor agregado a lo que puede proveer un servidor SIP de los vistos anteriormente. Se descarta su utilización para proveer funcionalidades.

3.1.1.12 – Resumen de la Revisión de B2BUA

Las soluciones presentadas y discutidas tienen puntos que pueden valorarse dentro de las funcionalidades que otorgan, manejando con rigor que las versiones soporten una actualización regular y constante, una comunidad de desarrollo en continuo movimiento y que además sean flexibles al momento de realizar las configuraciones necesarias para lograr las funcionalidades requeridas. Por otra parte, es un criterio a considerar que exista una masa crítica de usuarios del sistema, ya que esto permite a los desarrolladores tener un feedback importante por parte de los administradores y usuarios de un sistema.

A lo largo de la investigación se nombran proyectos que se denominan SIP Servlets, estos se descartan ya que la idea de este tipo de sistemas es proveer servicios, como por ejemplo, del tipo web. Si bien pueden proveer servicios de telefonía, se vale de la capacidad de SIP para poder establecer otros servicios. Pero dado el enfoque de esos sistemas, no cumplen con lo requerido para alcanzar funcionalidades específicas de una PBX, es razón por la cual se dan por descartados.

Este análisis arroja como resultado que las dos posibles soluciones son *Asterisk* y *FreeSWITCH* ya que proveen la posibilidad de dar las funcionalidades de una PBX como requiere el proyecto.

	Licencia	Protocolos	Codecs	Arquitectura	Observaciones	Califica
Asterisk	GPL	SIP, H.323, IAX	G.711, G.722, G.723, G.726, G.729a, GSM, iLBC, Linear, LPC-10, Speex	Semi Modular	Más usado	Si, el más usado
Elastix	GPLv2	SIP, IAX, H.323, XMPP	G.711, G.722, G.723, G.726, G.729a, GSM, iLBC, Linear, LPC-10, Speex	Monolítica	Administrable vía web	No, aplicaciones innecesarias
FreeSWITCH	MPL 1.1	SIP, NAT-PMP, STUN, SIMPLE, XMPP, Google Talk (Jingle), IAX, H.323, MGCP, RSS, Skype	G.722.1, G.722.1C, G.722, G.711, G.726, G.723.1, G.729ab, AMR, iLBC, Speex, CELT, Opus, LPC-10, SILK, DVI4	Modular	Mayor manejo de codecs	Si, arquitectura modular y más codecs HD
SipXecs	AGPL/LGPL	SIP	G.711, G.729, G.722	Modular	Necesita licenciamiento pago para otras características	No, licencia
YATE	GPL	SIP, IAX, H.323, XMPP	G.711, GSM, iLBC	Modular	Plantea solución integrada de Soft PBX y SIP Router	No, baja masa crítica de usuarios
CallWeaver	GPLv2	SIP, IAX2, H.323, MGCP, SCCP	G.723, G.729, iLBC	Modular	Discontinuado	No, discontinuado
GNU SIP Witch	GPL	SIP	n/a	n/d	No maneja codecs	No negocia códecs
SEMS	GPLv2	SIP	G.711, GSM, iLBC, Speex, G.726, G.722	Modular	Pocos casos de éxito	No, baja masa crítica de usuarios
Mobicents	LGPL	SIP	G.711, GSM, Speex, G.729	Modular	Focalizado en proveer servicios de valor agregado	No se adecúa a los objetivos de proyecto
Cipango	Apache License Version 2.0	SIP	n/a	Jetty	No maneja codecs	No se adecúa a los objetivos de proyecto
SailFin	CDDLv1/GPLv2	SIP	n/a	Modular	No hay mucha documentación	No se adecúa a los objetivos de proyecto

Tabla 3. 1 - Resumen comparativo de B2BUA

3.1.2 – SIP Proxy/SIP Registrar

La documentación y los argumentos que posteriormente se discutirán en el capítulo 4, llevarán a concluir la necesidad de la existencia de un servidor SIP Proxy/Registrar. Este móvil induce a investigar las opciones existentes en el mercado en materia de sistemas que puedan soportar una alta carga de suscripciones para poder lograr la cantidad de usuarios concurrentes deseados.

3.1.2.1 – Kamailio

El proyecto *Kamailio* (siendo un *fork* del proyecto llamado *OpenSER*, hoy depreciado) es un software *Open Source* lanzado bajo la licencia GPL que implementa un SIP server, concretamente un SIP Proxy/Registrar. Tiene la capacidad de manejar en el orden de miles de establecimientos por segundo, utiliza TCP asíncrono, UDP y SCTP, comunicación segura vía TLS para VoIP, maneja servicios de mensajería instantánea y presencia e incluso establece políticas de ruteo de menor costo. Puede actuar como balanceador de carga y realizar fail-over de enrutamiento, tarificación, autenticación y autorización ante bases de datos del tipo MySQL, Postgres, Oracle o Radius, LDAP y monitoreo SNMP.

Puede ser utilizado para la instalación de plataformas de servicios VoIP de gran porte, incluso oficiar de Gateway SIP-PSTN e interactuar con sistemas IP PBX como *Asterisk*, *FreeSWITCH* o *SEMS*. Todas las aplicaciones de este sistema están escritas en lenguaje C para Linux/Unix, apuntando a performance, flexibilidad y seguridad. Entre las flexibilidades de programación las extensiones pueden ser escritas en lenguajes Lua, Perl o Python.

A nivel macro, *Kamailio* no es un sistema IP PBX como *Asterisk* o *FreeSWITCH*, la función de *Kamailio* es la señalización SIP para el establecimiento de la sesión sin intervenir en el procesamiento de las sesiones de media. Por esta razón es que apunta a la estabilidad y performance del sistema, denotando una cantidad considerable de establecimientos por segundo, además de usuarios registrados en un mismo sistema.

La arquitectura de *Kamailio* es modular, estando dividida en dos grandes categorías. El núcleo (core) provee las funcionalidades de bajo nivel para el funcionamiento del sistema, contando con código compartido por diferentes módulos guardadas en librerías internas. Los módulos son los componentes que

proveen la mayor parte de las funcionalidades que hacen la robustez y potencia de *Kamailio*.

El core incluye administrador de memoria, SIP message parser, manejo de la capa de transporte (TCP, UDP, TLS, SCTP) y DNS, sistema de configuración e intérprete de archivos, reenvío en modo stateless, manejo de pseudo-variables y transformaciones. Las librerías internas incluyen componentes de versiones antiguas de *Kamailio*, capa de abstracción de base de datos, sistema de manejo de interfaz y modulo estadístico.⁵

Dadas las características del sistema se ha utilizado en carriers para proveer servicios SIP, en casos como la empresa alemana 1&1 logró otorgar redundancia geográfica a sus servicios, no solo de datos sino también de telefonía. Esta empresa se vio limitada en los requerimientos de solicitudes de acceso no solo de servicios de internet, sino también de servicios de telefonía, por lo que adoptó, para lograr satisfacer la demanda a nivel geográfico, instalar equipos con *Kamailio* para proveer los servicios requeridos.

3.1.2.2 – OpenSIPS

Formalmente es el otro fork del proyecto SER que por razones de nombre registrado cambió a OpenSIPS. Es un SIP proxy server que cumple la RFC-3261, creado con el objetivo de cumplir con aplicaciones de gran porte. Desde el punto de vista de la utilidad, es un sistema con gran capacidad de reenvío de solicitudes y puede soportar manejo del orden de miles de usuarios con un solo servidor, permitiendo proveer servicios de VoIP a gran escala con un costo reducido de hardware. Unifica servicios de voz, video, mensajería instantánea y presencia de una forma eficiente en función de un diseño de sistema modular que lo convierte en escalable.

Originalmente el proyecto comenzó denominando SER (SIP Express Router), con el objetivo de implementar un enrutamiento utilizando la técnica de lenguaje basada en módulos y con alta velocidad de características de enrutamiento. Así, SER se convierte en el líder en materia de SIP proxy server basado en open source, tal es así que para el año 2004 fue adoptado por ISP alemanes.

El primer fork del proyecto se da en el año 2005, cuando desarrolladores del proyecto original lanzan *OpenSER* con la idea de crear una comunidad fuerte con

⁵ <http://www.asipto.com/pub/kamailio-devel-guide/>

desarrollos sólidos, nuevas funcionalidades, flexibilidad y documentación en torno al proyecto.

Debido a argumentos de marca, para el año 2008 el proyecto *OpenSER* es rebautizado con el nombre de *Kamailio*. Luego de esto, el proyecto sufre otro fork, el cual se denomina OpenSIPS. Al momento de este ultimo fork existían tres proyectos en ejecución, por un lado SER, por otro *Kamailio* y por otro OpenSIPS. La idea de los desarrolladores fue juntar para beneficiar al proyecto en uno solo, poder encontrar un camino en común y lograr el SIP Router original. Es aquí a finales de 2008 que los desarrolladores de SER y *Kamailio* se aúnan para poder seguir con un proyecto en común, por el contrario, OpenSIPS siguió su propio desarrollo [47].

Este recuento de la historia del proyecto SER y cómo termina en lo que actualmente es OpenSIPS y *Kamailio* sirve para percatarnos que estamos frente a dos sistemas que son similares.

OpenSIPS utiliza un sistema flexible basado en el modelo de plugin para aplicaciones de terceros. Estas aplicaciones pueden ser fácilmente creadas y “plugeadas” al servidor, esto favorece a los plugins de accounting como RAIDIUS, ENUM y presencia. El avance del proyecto permite que se estén agregando módulos nuevos de forma constante. OpenSIPS permite ser usado no solo por proveedores de servicios SIP, también puede ser utilizado para proveer aplicaciones SIP, firewalls SIP, balanceadores de carga y SBC (Session Border Controllers).

Según las necesidades, OpenSIPS es adecuado para ciertos ambientes en donde se precise velocidad ya que maneja del orden de miles de establecimientos por segundo a un costo bajo de utilización de hardware y flexibilidad, dado que los administradores pueden manejar el comportamiento de acuerdo al lenguaje de scripting. La escalabilidad es asegurada ya que el sistema permite vincular nuevas aplicaciones y módulos de acuerdo al crecimiento de la red en donde se implemente. El sistema es liviano, no demanda grandes recursos de hardware para ser utilizado. El tipo de uso de OpenSIPS puede ser variado, ya que dentro de las funcionalidades que otorga se encuentra el balanceo de carga, el modulo de presencia y la mensajería instantánea.

3.1.2.3 – MySIPSwitch

Sistema consolidado para proveer señalización SIP que permite administrar múltiples usuarios de diversos proveedores SIP, además de administración centralizada de servicios de VoIP. Los componentes del sistema están implementados en lenguaje C#.

Al momento de la realización de la revisión la última versión data de setiembre de 2008, en el foro de usuarios y desarrolladores su última actividad es de 2009.

Desde esta fecha no se permiten nuevas cuentas y en la página del programa recomienda a los nuevos usuarios utilizar una nueva versión, que es paga. Esta información que tenemos sobre el sistema nos da razones suficientes para darlo por descartado.

3.1.2.4 – YATE – Yet Another Telephony Engine

Lanzado bajo los términos de la licencia GPLv2, el proyecto está enfocado en VoIP y PSTN, y dentro de esta característica actúa como SIP Proxy, SBC, SIP router y servidor de registro.

El software está escrito en C++ de forma modular y soporta scripts escritos en diferentes tipos de lenguajes (PHP, Python y librerías de Perl, Unix Shell), las tres primeras son útiles para desarrollar funcionalidades externas a YATE.

La arquitectura de YATE se subdivide en cuatro partes: el core o núcleo donde se realizan las encapsulaciones, threads y funciones primitivas; el sistema de mensajes, donde se realiza el intercambio de datos entre los módulos; el sistema de telefonía; y el modulo Yate que extiende las funcionalidades de YATE no precisamente relacionado con módulos telefónicos.

YATE es un sistema potente que además de aglomerar en una sola solución tanto la parte de SIP proxy como la parte de IP PBX, pretende ser adaptable a diferentes servicios. Pero dada su condición de hacer todo lo disponible para una solución completa es poco escalable, el hecho de dedicar parte de la capacidad a procesar audio reduce considerablemente la cantidad de establecimientos, concluyendo que es un sistema a no ser utilizado en la concreción de Obelix.

3.1.2.5 – YXA

Es un programa escrito en lenguaje Erlang que implementa un SIP server. El nombre proviene de los sistemas Ericsson PBX llamados AXE cuya traducción al sueco es YXA. Es tanto un stack SIP transaccional de tipo stateful como un set de aplicaciones de SIP server. Maneja una arquitectura de tipo modular basada en el stack SIP.

Es Open Source lanzado bajo la licencia BSD cumpliendo con la recomendación que se establece en la RFC-3261. La última versión estable data de noviembre de 2007, convirtiendo en un software sin actualizaciones hasta el momento y sin una comunidad voluminosa para generar posteriores desarrollos. Según este análisis el sistema no cumple con los requisitos para lograr formar parte de los sistemas posibles a ser utilizados en el desarrollo del proyecto.

3.1.2.6 – Resumen de la Revisión de SIP Proxy/Registrar

La solución que se obtenga tiene que cumplir con ser adaptable a diferentes sistemas ya implementados. La idea es integrar los media servers que existan o que se decidan instalar en la red. Los diversos proyectos tienen particularidades que los hacen fuertes o débiles, tomaremos aquellos que además de fuertes permitan integrar otros sistemas y tengan por detrás una masa crítica de desarrollo considerable, enmarcándose en las aplicaciones a futuro y sobre todo en la escalabilidad de lo implementado. Dentro de lo analizado se pondrá especial énfasis en dos proyectos, *Kamailio* y *OpenSIPS*, ambos proyectos provenientes antiguamente del proyecto *OpenSER*.

	Licencia	Protocolos	Arquitectura	Califica
Kamailio	GPL	SIP, XMPP	Modular	Si, estructura modular y escalable
OpenSIPS	GPL	SIP, XMPP	Modular	Si, estructura modular y escalable
MySIPSwitch	BSD	SIP, Ajax	n/d	No, recomienda usar versión paga
YATE	GPL	SIP, IAX, H.323, XMPP	Modular	No, pretende más que señalización
YXA	BSD	SIP, SIPS	Modular	No, discontinuado desde 2009

Tabla 3. 2 - Resumen comparativo de SIP Proxy/SIP Registrar software

3.1.3 – Softphones

La variedad de *softphones* es cuantiosa. Debido a las necesidades existentes para el proyecto, se tuvieron en cuenta una cantidad suficiente ya en el caso eventual de que alguno de estos fallara o no tuviese el soporte para las funcionalidades necesarias. Se expondrán a continuación las diferentes soluciones para las diversas pruebas.

3.1.3.1 – Blink

Cliente SIP distribuido bajo la licencia GPLv3, la versión de distribución libre cuenta con características limitadas. De precisar otras características más avanzadas se debe comprar una licencia. Maneja múltiples cuentas SIP, además de soportar calidades de audio regulares. Soporta audio HD, G.722, G.711 Speex, iLBC y GSM, encriptación, además de mensajería instantánea, servicios de presencia, soporte de servicios de conferencia, consulta de correo de voz dentro de este MWI y llamada en espera [48].

3.1.3.2 – Ekiga

Previo a adoptar este nombre, *Ekiga* era conocido por *GnomeMeeting*, siendo un cliente *Open Source* de *softphone*, conferencia de video y mensajería instantánea para operar en internet. Soporta calidad HD tanto de audio como de video. Además de tener interoperabilidad con estándar, hardwares y proveedores de servicio, cumple con los estándares de telefonía (SIP y H.323).

Soporta llamada en espera, transferencias de llamadas y reenvío de llamadas (ya sean del tipo sin respuesta, ocupada o on always), soporta MWI, emite historial de llamadas además de monitoreo del tipo de llamadas y soporta ENUM.

A nivel de audio soporta códecs de HD que permiten una calidad de alta definición de sonido, algoritmos dinámicos de detección de silencio, cancelación de eco y *buffer* para minimizar efectos del *jitter*. Soporta varios tipos de códecs, entre estos, iLBC, GSM, G.711, G.726, G.721, Speex, G.722 y CELT [49].

Permite el uso de servicio de presencia SIP con mensaje personalizado y además permite registrar diferentes cuentas SIP con la posibilidad de usar éstas de forma simultánea.

3.1.3.3 – Empathy

Cliente de mensajería instantánea y VoIP para ser utilizado en el ambiente *GNOME desktop* de distribuciones Linux, la documentación sobre las cuentas SIP no es muy profunda ya que este software apunta a la gestión de cuentas de redes sociales y mensajería instantánea.

3.1.3.4 – Jitsi

Es un cliente de mensajería instantánea, audio y video implementado en Java. Soporta la mayoría de los protocolos de mensajería y de telefonía como el caso de SIP, IAX y otros. El proyecto original fue denominado como SIP Communicator, aglomerando cada vez más colaboradores de distintas partes del mundo. Basado en la arquitectura OSGi y utilizando el implementador Felix de Apache se convierte en un desarrollo extensible con una interfaz amigable.

La versatilidad de este *softphone* es el gran número de protocolos de telefonía y de mensajería instantánea que soporta. Implementa mecanismos de seguridad para la reserva de contraseñas y claves de acceso e incluso la encriptación de llamadas como el caso de SRTP y TLS.

Dentro de las particularidades del protocolo SIP, soporta el respaldo de contactos señalización segura con TLS, DTMF en banda (SIP INFO, RTP RFC-2833 y RFC-4733), y MWI (RFC-3842) [50].

3.1.3.5 – KPhone (KDE)

Es un SIP *User Agent* para Linux, que implementa funcionalidades de un *softphone* para VoIP, pero que no está restringido solo a eso. Como UA, establece sesiones a lo largo de la red entre las puntas de la comunicación.

Las principales características son que mantiene sesiones paralelas en el caso de mantener una activa y otra en espera, soporte para la encriptación de la voz, soporte de información de presencia, llamada en espera, transferencia de llamadas, reenvío de llamadas y autorespuesta. Además de soporte de audio, se puede utilizar para llamadas de video y mensajería instantánea [51].

3.1.4.6 – Linphone

Es una aplicación para VoIP multiplataforma, utiliza SIP para establecer la comunicación y está lanzado bajo la licencia GPL. Presenta la variante de ser utilizado bajo una interfaz gráfica así como también en modo consola.

Cumple con los requerimientos de UA según la RFC-3261, utiliza SIP tanto por UDP, TCP y TLS, soporta IPv6. A nivel de señalización, soporta múltiples llamadas simultaneas con manejo de llamadas del tipo *music on hold*, transferencia y resumen de la misma. Soporta proxy del tipo *registrar y outbound*, servicios de presencia y mensajería instantánea. Soporte de DTMF por SIP INFO en base a la RFC-2833 [52].

3.1.3.7 – Qutecom

Es un software que cumple con SIP desarrollado y lanzado con el nombre OpenWengo bajo la licencia GPL. Permite a los usuarios hablar a otros usuarios de SIP sin costo alguno, además de poder efectuar llamadas a usuarios de teléfonos celulares, enviar SMS y llamadas de video. La documentación de este *softphone* no es muy explícita, lo cual limita un análisis detallado [53].

3.1.3.8 – SFLPhone

Es un *softphone* lanzado bajo la licencia GPL, existen paquetes para distribuciones Debian, Fedora, openSUSE y Mandriva. En la página del producto describe ser una solución ideal para ámbitos empresariales ya que soporta funcionalidades del tipo conferencias y transferencia atendida. Apunta no solo a que sea útil para usuarios domésticos, también para recepcionistas que atienden una cantidad considerable de llamadas por día.

Opera de forma distribuida, demonio y cliente. Siendo que el cliente se comunica con el demonio a través de un DBus⁶ que es capaz de manejar múltiples conexiones VoIP en una instancia. Este demonio es necesario por todos los clientes *SFLphone*.

⁶ DBus: Proceso interno de comunicación (IPC) basado en Open Source para comunicar una aplicación de software con otra.

Permite tener múltiples cuentas registradas, soporta llamada en espera, conferencias, llamada de video, mensajería instantánea. Usa diferentes tipos de códecs soportando Celt, Speex y G.722. En materia de comunicación soporta DTMF en banda (SIP Info, RTP) [54].

3.1.3.9 – Twinkle

Es un *softphone* para VoIP que permite realizar comunicaciones y mensajería instantánea usando el protocolo SIP. Se puede utilizar para comunicar de forma directa entre teléfonos IP o utilizando un SIP proxy para encaminar las llamadas y los mensajes. Está disponible solo para sistemas operativos Linux (licencia GPL).

Soporta funcionalidades tales como utilización de dos líneas, múltiples identidades de usuarios para efectuar llamadas, ringtones personalizados, llamada en espera, conferencia de tres vías, transferencias de llamadas varias (blind, attended), DND, MWI, comunicación directa al voicemail.

Dentro de las capacidades de audio maneja códecs G.711 (A y u), GSM, Speex en banda angosta y Speex en banda ancha y ultra banda ancha, iLBC y G.726. Para todos ellos utiliza sistemas de preprocesamiento de audio para mejorar la calidad del audio de salida, del tipo AGC, reducción de ruido, VAD y AEC. Soporta drivers de sonido del tipo OSS y ALSA. Soporta servicios de presencia y mensajería instantánea [55].

3.1.3.9 – YATE Client

Cliente SIP desarrollado por el mismo grupo de desarrollo de YATE, la documentación es bastante escasa e incluso en la página de los desarrolladores hay poca explicación sobre la misma. Soporta protocolos SIP, IAX, H.323 y XMPP. Dentro de las funcionalidades soporta mensajería instantánea y vincula otros servicios externos de este tipo, conferencias y transferencia de llamadas. Dentro de las características del software, mantiene un historial de llamadas y agenda de contactos.

3.1.3.10 – X-Lite

Es un *softphone freeware* para VoIP que utiliza SIP. Actualmente se encuentra discontinuada la versión para Linux, aunque versiones previas si lo soportaban, solo está implementado para ambientes Windows o MAC OS X.

La versión Lite soporta llamadas básicas y funcionalidades básicas como el caso de las conferencias de tres vías, presencia y mensajería instantánea. Para utilizar otras funcionalidades o una cantidad superior de líneas o cuentas se precisa comprar una licencia. Existen dos tipos a diferentes costos, eyeBeam 1.5 y Bria 3.

3.1.3.11 – Resumen de la Revisión de Softphones

La revisión anterior se realizó en función de las opciones diferentes en materia de *softphones Open Source*. Dentro de los proyectos evaluados se considera el *softphone* Blink que tiene la capacidad de manejo de diferentes códecs y que permite diferentes cuentas al mismo tiempo. Por otra parte, si bien es un proyecto en el que no se avanzó mas, el *softphone Twinkle* permite funcionalidades que otros no cubren. Respecto al resto, se considera que los que pueden dar una solución integral son *SFLPhone* y *Jitsi*, ya que son proyectos en constante desarrollo y que al momento están mejorando sus versiones. Cabe aclarar que no se encuentra dentro de los objetivos del proyecto obtener un *softphone* adecuado para el sistema.

	Licencia	Protocolos	Códecs Soportados	Funcionalidades
Blink	GPL	SIP, RTP	iLBC, GSM, G.711, G.726, G.721, Speex, G.722 y CELT	IM, File Transfer, Multi Party Conference, Wideband
Ekiga	GPL	SIP, H.323, H.263, H.264/MPEG-4 AVC, STUN, Theora, Zeroconf	iLBC, GSM, G.711, G.726, G.721, Speex, G.722 y CELT	Video, IM, LDAP, Call Forwarding, Call Transfer
Empathy	GPL	SIP, XMPP (Jingle), ICE (STUN/TURN), Zeroconf	n/d	IM, multi-user A/V, collaborative applications
Jitsi	LGPL	SIP/SIMPLE, XMPP/Jingle	Opus, SILK, Speex, G.722, PCMU/PCMA (G.711), iLBC, GSM, G.729	Text messaging, audio/video telephony, IPv6, call recording
KPhone	GPL	SIP, STUN, NAPTR/SRV	n/d	Video, voice, IM, external Sessions, IPv6 support for UDP
Linphone	GPL	SIP	Speex, G711, GSM, G722, SILK, G729 y iLBC	Video, IM, STUN, IPv6
Qutecom	GPL	SIP	iLBC, G.711 (PCMA y PCMU), G.722	Video, IM (MSN, AIM, ICQ, Yahoo!, XMPP, Google Talk), voicemail, conferencing
SFLPhone	GPL3	SIP, RTP, IAX2, STUN per account, SRV	Celt, Speex 16000/32000, G722, G711 y GSM	Gnome/KDE client, address book, multiple accounts, unlimited number of calls, call transfer, call hold/unhold, call recording, Multi-way conferencing
Twinkle	GPL	SIP	G.711 (ley-A y ley-u), GSM, Speex, iLBC y G.726	Conferencing, chat, file transfer, firefox integration, call redirection, voicemail, support of VoIP-to Phone services
YATE Client	GPL	SIP, IAX, H.323, XMPP	n/d	n/d
X-Lite	Freeware/closed Proprietary	SIP, STUN, ICE	n/d	IM, single login account, for Windows and Mac also Conferencing, Video and SIMPLE based presence

Tabla 3. 3 - Resumen de softphones del mercado

3.2 Elección del Software a Utilizar

El análisis de las diferentes opciones existentes en el mercado de IP PBX, SIP Proxy y *Softphones* exige tomar un conjunto acotado de estos para poder definir cuáles son los más adecuados para los objetivos de Obelix.

Los criterios para preseleccionar en primera instancia, son claros. Precisamente se exige que las opciones provengan de proyectos *Open Source* que brinden la posibilidad de desarrollar en forma gratuita. Por otra parte, al hacer hincapié en este tipo de sistemas, criterio fundamental es que los desarrolladores de esa herramienta y la comunidad estén pujando por actualizaciones y no hayan dejado el proyecto abandonado. Elegir una herramienta que no tenga desarrollos posteriores y/o actualizaciones no se circunscribe al concepto de escalabilidad. De todos modos, existen casos concretos de proyectos que han dejado de desarrollar y que al momento han sido de suma utilidad para evaluar sistemas y arquitecturas.

No es menor apuntar a proyectos focalizados en un área en específico, con esto nos referimos a software que tengan como objetivo, si bien tratar varios puntos, no la totalidad de la funcionalidad de un core SIP. Se apunta a un sistema con el menor punto de fallas. Esto implica soluciones que no aglomeren varios de estos puntos, por lo que es de interés minimizar las fallas además integrar diferentes soluciones permite lograr una optimización de las piezas que redunde en una optimización del sistema en general.

La idea es conformar el proyecto mediante diferentes componentes de software interactuando entre ellos para proveer el servicio requerido, con las funcionalidades corporativas que se describirán en capítulos posteriores.

Dentro del software analizado tenemos las IP PBX o lo que más adelante se denominaran *media servers*, encargados de proveer los servicios de funcionalidades corporativas; conferencias, *voicemail* y música en espera. Los SIP Proxy/Registrar que se encargarán de la señalización para los establecimientos de sesión, registrar los diferentes usuarios del sistema, efectuar el balanceo de carga de las funcionalidades y proveer al sistema de una capacidad suficiente para la demanda. A modo de prueba, los *softphones* darán la utilidad necesaria para comprobar la funcionalidad requerida, no cerrando en un conjunto acotado si no utilizando aquellos que mejor se adecuen a las funcionalidades.

3.2.1 – Asterisk o FreeSWITCH

Pese a que el objetivo de este capítulo es la discusión y la evaluación de los diferentes tipos de software que se van a utilizar, es imposible evitar el avance a nivel temporal de estos.

FreeSWITCH es un proyecto que parte de los desarrollos efectuados por la empresa *Digium*, que conformó el proyecto *Asterisk*. Un conjunto de desarrolladores de esta herramienta consideraron que se podría mejorar la performance de este sistema. Estos módulos más eficientes fueron propuestos para mejorar el funcionamiento de *Asterisk*, pero el equipo de *Digium* no aceptó cambios de fondo en el código [56].

A consecuencia de esta decisión, este grupo que pretendía innovar el núcleo y la forma de operar de *Asterisk*, conforma luego de varios intentos el software *FreeSWITCH*. Objetivo: cubrir los aspectos de estabilidad y performance que *Asterisk* no cubría; una estructura modular que permita mayor eficiencia al momento de iniciar el software; y proveer de funcionalidades que no consuman más de lo debido de los recursos de hardware. En resumen, superar a *Asterisk*.

Se logró el objetivo. Para el año 2006, cuando *FreeSWITCH* se lanza al mercado supera ampliamente la performance de *Asterisk* convirtiéndolo en una herramienta extremadamente poderosa ante su predecesor. Entre las particularidades del nuevo sistema se encuentra que, además de superar en diez veces la capacidad de manejo de llamadas, implementa módulos más eficientes como el sistema de puente de conferencias, maneja tasas de muestreo que van desde los 8kHz a los 48kHz permitiendo calidades de audio de tipo CD. La estructura de módulos permite una carga fácil del sistema y manejo de la memoria de forma eficiente para poder cumplir con las altas cargas sometidas.

La posición que ocupa *Asterisk* obliga a superarlo. Para la versión 1.8 comienza a operar con diferentes tipos de tasas de muestreo, ya que al momento de la versión 1.6 solo manejaba tasas del orden de los 8kHz, pero todavía sin alcanzar niveles como los de *FreeSWITCH*. Precisamente el máximo alcanzaba los 16kHz.

No existe una inclinación clara de porqué *Asterisk* o *FreeSWITCH* para la implementación del proyecto. Razones más que suficientes son que ambos son performantes, tienen iguales funcionalidades, son adaptables a la demanda de suscriptores, ambos tienen desarrollos por parte de la comunidad que día a día mejoran el código y permiten una flexibilidad para los diferentes requerimientos exigidos por los administradores de estos sistemas.

Al momento de la realización del proyecto, *FreeSWITCH* presenta características superiores a las de *Asterisk*. El manejo de códecs por parte de *FreeSWITCH* es superior al de *Asterisk*, manejo de tasas de muestreo superiores permiten manejar códecs con calidad de audio de mayor resolución, caso concreto es el códec de CELT que impone calidades de CD en el audio a ser transferido. Actualmente en la última versión de *Asterisk* se están manejando tasas de muestreo del orden de los 16kHz, cuando *FreeSWITCH* maneja tasas de 48 kHz.

Por otro lado el modulo de conferencias de *FreeSWITCH* presenta una estabilidad más robusta que el modulo Meetme de *Asterisk*, incluso en desarrollos posteriores de *Asterisk*, como la versión 10, se comienza a implementar el modo ConfBridge, una solución al sistema de conferencias similar al implementado por *FreeSWITCH* [57].

3.2.4 – Kamailio u OpenSIPS

Como se hablo en partes anteriores, los dos parten de un mismo origen, tal es el mismo punto de partida que en materia de configuración son similares. La decisión por uno u otro es bastante difusa. Existe un único factor que diferencia a estos dos sistemas y es el mismo que en su momento hizo que el proyecto SER dejase de existir y se transformara en dos, el manejo de audio por parte del core de señalización.

En la mayoría de las investigaciones realizadas a lo largo del proyecto se apuntó a la documentación provista por el desarrollador, acceder a la información de funcionamiento, las versatilidades del programa, etc. Los dos sistemas que se discuten tienen documentación vasta y con casos en concreto que se discuten a fondo, pero se debe resaltar la documentación provista por parte del grupo de desarrollo de *Kamailio*, éstas son más explícitas que las de OpenSIP. Incluso apunta a una integración de sistemas, lo cual no es menor, debido a los móviles del proyecto que es integrar diferentes sistemas.

Pero por más que ambos, que en efecto es así, tengan una documentación rica y con explicaciones de los diferentes escenarios, la decisión final de haber adoptado a *Kamailio* está fundamentada en que el sistema es solamente un sistema de señalización SIP, el sistema solo se encarga de manejar la señalización necesaria para el establecimiento y finalización de la sesión. OpenSIPS, como fue discutido en párrafos anteriores, no concibe un núcleo de señalización que excluya la funcionalidad de monitorear la llamada, o sea, ser un B2BUA. Esto repercute directamente en la cantidad de establecimientos por segundo que un

sistema puede soportar, ya que de ser una llamada plana el insumo de procesamiento de señalización sumado al costo de procesar el audio, baja la disponibilidad de procesamiento para poder recibir una carga superior. Particularidades como esta también limitan que el sistema pueda llegar a cumplir requerimientos de alta disponibilidad, lo cual es uno de los objetivos a cumplir en este proyecto.

3.3 Herramienta Generadora de Tráfico

Una vez discutidos los sistemas a ser utilizados se precisa una herramienta que permita probar un correcto funcionamiento de lo implementado. Esta herramienta tiene que generar el tráfico SIP necesario para efectuar pruebas conceptuales y pruebas de capacidad para que se pueda analizar si un media server o un SIP server está o no en el camino correcto. Esta herramienta será necesaria no solo como análisis de las soluciones obtenidas, sino también como medio para realizar las pruebas correspondientes de capacidad del sistema.

La herramienta generadora de tráfico y de test *SIPp* es basada en *Open Source* y es utilizada para generar las pruebas necesarias de funcionamiento de los sistemas a integrar. La versatilidad de esta herramienta permite crear escenarios para distintos ámbitos de prueba al momento de testear un sistema SIP, basado en código xml. Sólo es necesario trabajar con las trazas SIP dentro del código para poder realizar las pruebas pertinentes. El tipo de formato de escenario (como se le denomina a las pruebas) se explicará en los anexos correspondientes.

Despliega dos modos de operativa que si bien no se especifican, sirven para establecer las diferentes pruebas donde se ejecutarán, tanto un simulador de UAC como simulador de UAS, lo cual permite hacer pruebas en diferentes puntos del sistema, soporta autenticación dependiendo si es requerida o no por la parte testeada. Dentro de las capacidades de la herramienta, permite utilizar archivos externos creados en formato csv para la autenticación de múltiples usuarios en un mismo escenario.

Cada escenario contiene dentro de los parámetros establecidos del código xml, las trazas SIP o los mensajes que se enviarán al destinatario. Por ejemplo, en el

caso de una llamada, se colocarán los métodos correspondientes para ser enviados, y dentro del escenario se generarán las instancias para las respuestas que se enviarán desde el servidor hacia la herramienta. Incluso en el caso de que exista una captura del flujo RTP, el generador permite que el escenario pueda reproducir el audio para efectuar pruebas realistas de procesamiento.

En el Anexo 3 se describe con profundidad la instalación de la aplicación en sistema operativo y los escenarios utilizados para las pruebas efectuadas con las diversas funcionalidades. En los capítulos posteriores se podrá ver la potencialidad de la herramienta ya que las pruebas efectuadas con la misma sirven para dimensionar los sistemas a ser implementados.

Además de las flexibilidades permitidas en los escenarios *SIPp*, permite variar las cadencias y la cantidad de ejecuciones del escenario. De esta manera habilita a incrementar las cadencias y ejecuciones para probar una configuración o funcionalidad realizada. Terminadas las pruebas correspondientes, *SIPp* presenta un comunicado detallando los errores, en caso de existir, o los datos correctos de la cantidad de llamadas procesadas y la cantidad de llamadas que se efectuaron según el escenario.

Capítulo 4 – Núcleo de Señalización

En este capítulo se verán posibles arquitecturas para un núcleo de señalización SIP, las cuales van a ser probadas en cuanto al rendimiento con el fin de determinar cual o cuales, y de qué manera, se ajustan a los objetivos de requerimiento de tráfico.

4.1 – La Soft-PBX

En este punto, parece natural comenzar por un núcleo de un elemento que todo lo resuelva. Esto es, que logre manejar la señalización para el establecimiento de llamadas y también pueda brindar los servicios propios de un sistema telefónico corporativo.

Como ya se vio anteriormente, un elemento que logra todo esto es una PBX de software o Soft-PBX.

Una solución de este tipo es ampliamente usada en el mercado actual de la telefonía IP corporativa. Por lo tanto, es un buen punto de partida para este análisis.

Para la siguiente prueba, se instaló un FreeSWITCH Versión 1.2.1 en una PC Dual-Core @ 1.6 GHz, 2 GBytes de memoria RAM y tarjeta de red Gigabit ethernet, en un sistema operativo CentOS 6.1. Los detalles de esta instalación se pueden consultar en el anexo 2.

El escenario de la prueba, que se ilustra en la figura 4.1, consiste en generar llamadas entre dos generadores de tráfico (uno en modo llamador, UAC, y otro en modo llamado, UAS), ambos registrados en la PBX. La misma, que se comporta como un B2BUA, procesará además de la señalización necesaria todo el flujo de audio entre las puntas.



Figura 4. 1 – Esquema 1 de prueba de llamadas a B2BUA

El flujo de una de estas llamadas se ilustra en la figura 4.2, y es este flujo el que se aplicará repetidas veces con el fin de determinar la capacidad de procesar llamadas simultáneas por parte del B2BUA. La instalación y todos los escenarios utilizados en los generadores de tráfico pueden consultarse en el anexo 3.

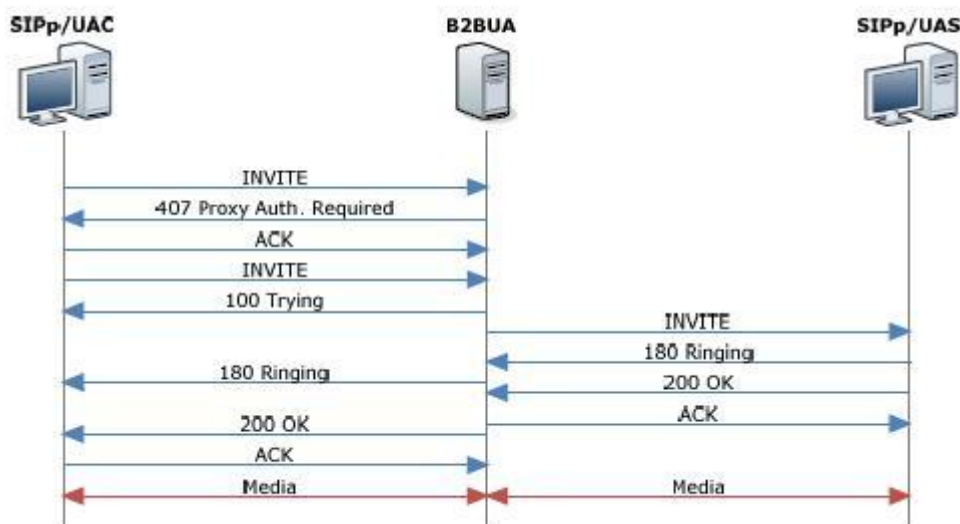


Figura 4. 2 – Esquema 2 de prueba de llamadas a B2BUA

Esta prueba arroja como resultados los siguientes:

Al correr 100 escenarios por segundo durante un segundo, el sistema logro establecer las 100 llamadas. Este escenario fue exigente en materia de procesamiento de señalización, pero no en cantidad de llamadas.

Al correr 50 escenarios por segundo (menos exigente en cadencia para testear la cantidad de llamadas capaz de acumular) durante 30 segundos (1500 llamadas en total), el sistema fue capaz de establecer 346 llamadas antes de colapsar. En este punto había transcurrido casi la totalidad de la prueba y se habían encolado las llamadas que aun no habían sido despachadas.

Se va a tomar este valor, 346, como la máxima capacidad en materia de llamadas del sistema testado.

Un rápido análisis de estos resultados indica que una solución de estas características, si bien es una solución completa, no soporta el volumen de demanda buscado en los objetivos.

Hoy día en el mercado actual la solución que plantean los proveedores de este tipo de servicios, cuando alguien con este ejemplo de solución necesita escalar en volumen de tráfico, es colocar mas servidores de las mismas características o bien con una estructura jerárquica, o bien en paralelo en un modo de cluster.

Cabe destacar que en el mundo de la telefonía IP así como en otras áreas de las telecomunicaciones (como es el caso de la telefonía móvil), los teléfonos o agentes de las puntas han adquirido la inteligencia que antes era brindada enteramente por la red. Esto hace que tener equipos en la red capaces de proveer muchos servicios distintos a la vez sea cada vez menos necesario, y se tiende a modularizar colocando equipos dedicados a tareas específicas para brindar de manera más eficiente los servicios que si deben ser facilitados por la red.

En cualquiera de ambas soluciones planteadas, si bien las implementaciones a nivel de arquitectura son bastante distintas, se encuentra el mismo problema. Todos los elementos de dichas arquitecturas son capaces de implementar por si solos todas las funcionalidades que se buscan implementar con sistema completo, y cuando se dice completo, se incluyen también a los agentes en las puntas.

Quizá el punto más notable que conspira en contra de este tipo de soluciones, así como para el caso de un único servidor, es que la gran parte de los recursos de la red son absorbidos por el procesamiento del flujo de audio. Y esto es algo intrínseco al tipo de herramienta utilizado para la construcción de las mencionadas arquitecturas, ya que para lograr brindar todas las funcionalidades para las cuales están diseñados estos servidores deben hacer transitar los flujos de audio por el interior de los mismos.

Y no por esto son una mala herramienta, sino que todo lo contrario. Se encuentra especial utilidad en este tipo de servidores para brindar todo tipo de funcionalidad en donde se requiera efectivamente manejo de los flujos de media, y solo cuando así se requiera. De esta manera se convierten servidores multiuso en dedicados para lo que mejor pueden realizar, manejo de media.

En un sistema telefónico cualquiera, la gran mayoría de los recursos utilizados en un momento dado son capitalizados por las llamadas telefónicas punta a punta. Esto es que, de todos los usuarios haciendo uso de un sistema de este tipo son minoría los que están consumiendo recursos por requerir efectivamente un servidor que procese un flujo de audio.

Cabe destacar que esto es cierto siempre y cuando todos los agentes en las puntas de una conversación (*softphones* o *hardphones*) manejen nativamente al menos un códec en común. De lo contrario, es necesario que un elemento de este tipo (B2BUA) realice una traducción (*transcoding*) para que efectivamente la llamada se pueda cursar.

Hoy en día, todos los agentes de las puntas manejan los códecs más populares para utilizar en el flujo de audio, además de alguno específico que decida el fabricante o desarrollador.

De esta manera, cualquiera sea el agente que se use para realizar la llamada y a cualquier agente que se esté llamando, es seguro que ambos coincidan en la elección de un códec para poder establecer una conversación sin necesidad de un servidor que haga el *transcoding*.

Entonces, si a servidores tipo B2BUA no se los va a emplear para procesar el flujo de audio de una llamada de punta a punta, es innecesario que se encuentren en los caminos de la señalización asociada a la misma. Aunque se pueda emplear a este tipo de equipos como B2BUA pero solo de señalización (esto es que se comporta de esta manera solo en la señalización de las llamadas, pero el flujo de audio no atraviesa de ninguna manera ese servidor) no tiene ningún sentido utilizar un servidor especialmente diseñado para manejar flujos de media y delegarlo al solo manejo de señalización.

Para este trabajo es especial otra herramienta ya comentada en capítulos anteriores, el SIP Proxy.

4.2 – El SIP Proxy

En esta sección se analizará la performance de un SIP Proxy. Se comparará con un B2BUA en el manejo de señalización, y se evaluará su capacidad para efectuar esta tarea. También se presentarán reglas de dimensionamiento, y se aplicará

parte de la teoría de teletráfico con el fin de presentar el potencial de esta herramienta.

4.2.1 – B2BUA vs. SIP Proxy

Se realizara una prueba comparativa en establecimientos de llamadas punta a punta de un B2BUA que no maneja flujos de audio y de un SIP Proxy. En el anexo 2 se detalla como desactivar la opción de manejo de audio en el B2BUA utilizado.

La siguiente prueba (con los mismos equipos que en la prueba anterior) intenta ilustrar como un B2BUA mejora significativamente su capacidad de establecimientos de llamadas desactivando la opción de que procese los flujos de audio. Cabe destacar que sigue siendo un B2BUA en el plano de señalización, pero el flujo de audio va de punta a punta sin intermediarios. Se ve en las figuras 4.3 y 4.4 los diagramas de la prueba.



Figura 4. 3 – Esquema 1 de prueba de llamadas a B2BUA sin procesar audio

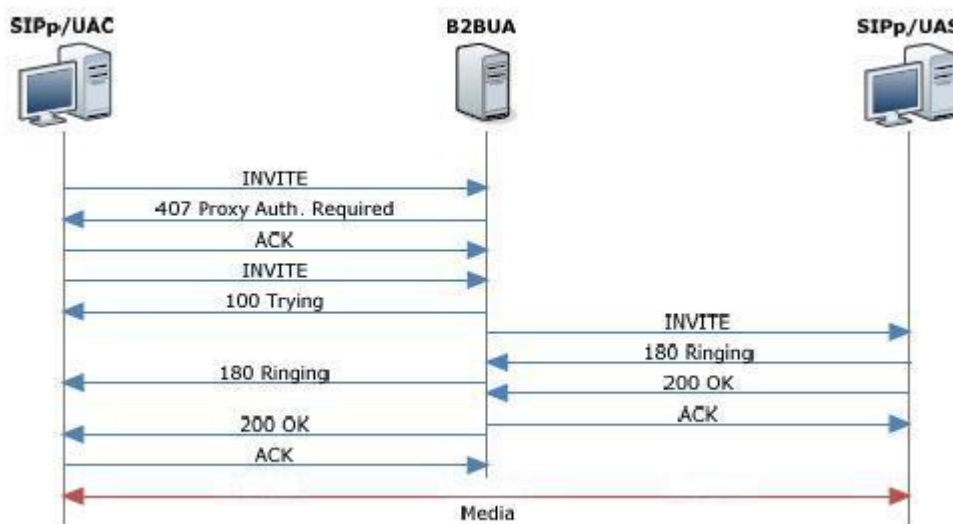


Figura 4. 4 – Esquema 2 de prueba de llamadas a B2BUA sin procesar audio

Sabiendo que una cadencia de 50 llamadas por segundo era soportada procesando audio, se aplicó el escenario 50 veces por segundo hasta acumular 4000 llamadas simultáneas.

Los resultados son los siguientes:

Al término de la prueba, donde se corrieron 4000 escenarios (80 segundos de comenzada), el B2BUA llevaba solo 535 llamadas en la región de intercambio de media. El resto fueron puestas en cola. Si bien no colapsó como en el caso anterior, se toma como valor final 535 llamadas, las que estaban en curso luego de 80 segundos, dado que corresponden las llamadas que llegaron en los primeros 11 segundos de prueba. El resto de las llamadas se dan por fallidas dado que en el peor caso había llamadas que 69 segundos después de solicitar la llamada aun no habían sido servidas.

La tabla 4.1 a continuación resume los resultados de ambas pruebas anteriores.

	Procesando audio	Sin Procesar audio
Llamadas cursadas	346	535

Tabla 4. 1 – Tabla comparativa de llamadas cursadas en B2BUA con y sin procesamiento de audio

Claramente la capacidad para el establecimiento de llamadas sin procesar audio es mayor a cuando el B2BUA procesaba el audio de todas ellas. Sin embargo aun se encuentra muy lejos de lo deseado en este proyecto.

Se evaluará en la siguiente prueba la capacidad para establecer llamadas de un SIP Proxy, en particular Kamailio versión 3.2, en el mismo hardware de las pruebas anteriores. Este software requiere de una base de datos para su función de SIP Registrar, sin la cual no sería posible la localización del agente llamado. Se instaló en la misma PC un servidor MySQL para dicha función. La prueba es similar a la del B2BUA sin el procesamiento de audio y se ilustra en las figuras 4.5 y 4.6. Los detalles de estas instalaciones se pueden encontrar en el anexo 1.



Figura 4. 5 – Esquema 1 de prueba de llamadas a SIP Proxy

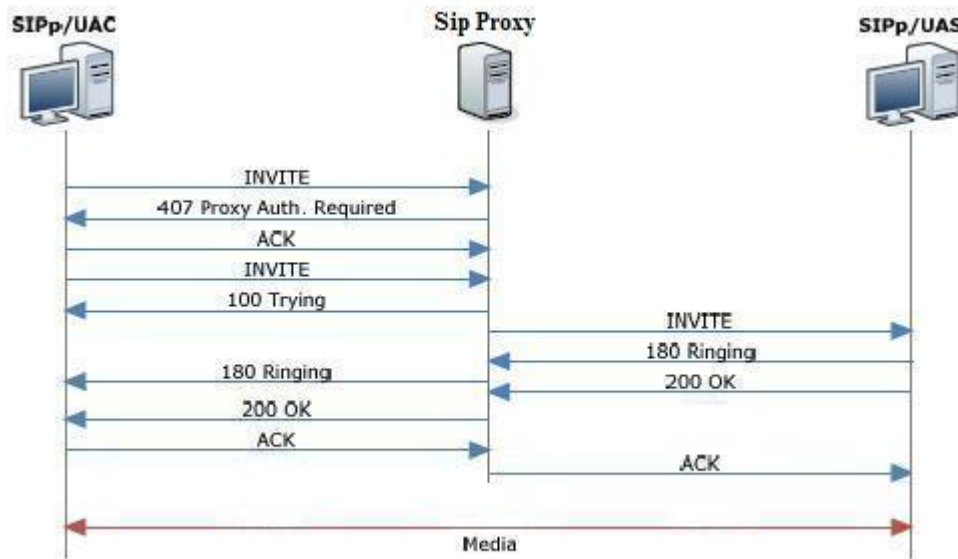


Figura 4. 6 – Esquema 2 de prueba de llamadas a SIP Proxy

Los resultados de la prueba son los siguientes:

A 50 escenarios por segundo en un total de 1000 escenarios, todos fueron cursados en tiempo.

Igual a 500 escenarios por segundo en un total de 5000 escenarios.

Lo mismo a una cadencia de 1000 escenarios por segundo en un total de 100.000 escenarios. De esta manera se acumularon 100.000 llamadas simultáneas, siendo servidas todas en la medida que arribaban al servidor sin ningún tipo de encolamiento o demoras.

No queda ninguna duda que un SIP Proxy tiene un rendimiento netamente superior para el establecimiento de llamadas punta a punta. Si un comportamiento de SIP Proxy es ventajoso al momento de poder establecer un volumen grande de llamadas, es natural usar para esta tarea un software diseñado especialmente para comportarse como tal, y no un B2BUA emulando su comportamiento.

Cabe destacar que en la prueba anterior del SIP Proxy, los resultados no son aquellos en donde se llegaba a un límite para el establecimiento de llamadas, pero fue suficiente para probar la gran diferencia en términos de performance existente. Para hablar de este límite, se expondrán los resultados de otras pruebas realizadas sobre el SIP Proxy Kamailio.

4.2.2 –Capacidad de manejo de señalización

Las siguientes pruebas intentan encontrar los límites en cuanto a procesamiento de señalización por parte de Kamailio y sus causas con el fin de lograr optimizar el rendimiento en este aspecto del sistema.

4.2.2.1 – Registros. Optimización del sistema.

La primera de ellas consiste en generar escenarios en donde un usuario se intenta registrar en el sistema y, validación de credenciales mediante, se logra registrar correctamente. Se medirá la cadencia máxima de escenarios de registro por segundo capaz de soportar el sistema sin que el usuario a registrarse deba retransmitir alguno de los mensajes para lograrlo. Llegado el punto, se analizarán aspectos como el nivel de procesamiento del sistema y la memoria utilizada.

El escenario de la prueba se ilustra en las figuras 4.7 y 4.8.



Figura 4. 7 – Esquema 1 de prueba de registros a SIP Proxy

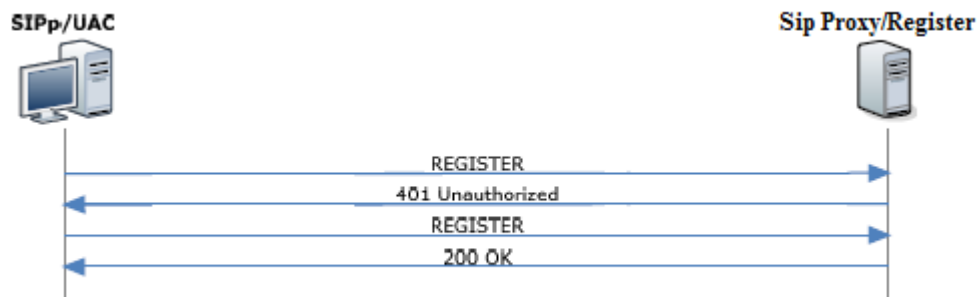


Figura 4. 8 – Esquema 2 de prueba de registros a SIP Proxy

Un factor relevante de esta prueba es que la tolerancia que tiene un usuario antes de retransmitir un mensaje, si no le es respondido a tiempo, es de 500 ms.

Los resultados son los siguientes:

Se lograron ejecutar 4500 escenarios por segundo de manera continua sin que hubiera ninguna retransmisión por parte de los generadores.

La figura 4.9 muestra el uso del procesador así como de la memoria del SIP Proxy.

```

top - 00:23:43 up 2:45, 2 users, load average: 0.83, 0.23, 0.08
Tasks: 142 total, 3 running, 139 sleeping, 0 stopped, 0 zombie
Cpu(s): 73.4%us, 17.5%sy, 0.0%ni, 1.8%id, 0.0%wa, 3.5%hi, 3.8%si, 0.0%st
Mem: 1029932k total, 290372k used, 739560k free, 27524k buffers
Swap: 2064376k total, 0k used, 2064376k free, 171620k cached

  PID USER      PR  NI  VIRT  RES  SHR  S %CPU  %MEM    TIME+  COMMAND
 2212 mysql    20   0 134m  16m 5288 S 92.1   1.6   6:32.58 mysqld
 3410 root      20   0 51592 4644 3372 R 52.4   0.5   1:21.21 kamailio
 3411 root      20   0 51592 4344 3072 R 52.0   0.4   1:23.96 kamailio
 3449 root      20   0 2676  1128  880 R  0.3   0.1   0:01.43 top
    1 root      20   0 2856  1380 1184 S  0.0   0.1   0:01.37 init
    2 root      20   0    0    0    0 S  0.0   0.0   0:00.01 kthreadd
    3 root      RT   0    0    0    0 S  0.0   0.0   0:00.00 migration/0
    4 root      20   0    0    0    0 S  0.0   0.0   0:00.00 ksoftirqd/0
    5 root      RT   0    0    0    0 S  0.0   0.0   0:00.00 migration/0
    6 root      RT   0    0    0    0 S  0.0   0.0   0:00.00 watchdog/0
  
```

Figura 4. 9 – Captura del comando “top” en SIP Proxy con base de datos en mismo host en prueba de registros

Es más que notoria la capacidad para procesar señalización de este SIP Proxy, en cual recibía 9000 mensajes de *Request* y hacía devolución de otros 9000 mensajes (*Replies*) en cada segundo.

No obstante, es más que interesante notar el hecho de que gran parte de los recursos del servidor SIP Proxy son consumidos por el motor de la base de datos, como se observa en la figura 4.9, afectando su rendimiento como sistema. Esto hace pensar que estos valores más que prometedores aun podrían no ser definitivos.

El paso lógico para intentar mejorar la performance de esta configuración, es migrar la base de datos propia del Kamailio hacia un servidor externo con un servidor MySQL, dejando solo al SIP Proxy como un cliente de tal.

Para la siguiente prueba, se instaló el servidor MySQL en un servidor HP modelo Proliant ML110 G6, el cual cuenta con un procesador Xeon Core i3 (4 núcleos) @ 3.06 GHz, 2GB de memoria RAM y una placa de red Gigabit Ethernet, en un sistema operativo CentOS 6.1. Las ilustraciones de la prueba son las mismas que la prueba anterior, figuras 4.7 y 4.8.

Con la misma filosofía de la prueba anterior, se incremento el número de escenarios de registro por segundo ejecutados llegando al punto donde comenzaban a haber retransmisiones de algunos de los mensajes.

Se pudieron cursar 9500 escenarios por segundo de manera constante sin ningún tipo de retransmisión, y en la tabla 4.2 se muestran datos obtenidos en la prueba tales como nivel de procesamiento y anchos de banda del SIP Proxy y de la base de datos.

@ 9500 reg/s	BW IN (Kbps)	BW OUT (Kbps)	CPU (%)
Sip Proxy	79	98	200
Base de datos	9.5	12.7	183

Tabla 4. 2 – Tabla de valores extraídos en SIP Proxy y base de datos en prueba de registros

La tabla 4.3 resume los resultados de ambas pruebas anteriores.

Sip Proxy/Base de datos	Mismo host	Distinto host
Registros por segundo	4500	9500

Tabla 4. 3 – Tabla comparativa de rendimiento en prueba de registros con base de datos en mismo host y en distinto host

Con esta última configuración, es evidente la mejora en cuanto a la capacidad de procesamiento de señalización del sistema. De esta manera, parece sensato pensar que esta arquitectura para el núcleo de señalización es la adecuada para los objetivos planteados.

Sin embargo, el punto que toca analizar es el porqué de este límite encontrado en esta arquitectura, y si es posible conseguir un mejor rendimiento de estas herramientas. Hay que detener la atención en el nivel de procesamiento de las PC utilizadas para estas pruebas, en particular la que es host del SIP Proxy.

Esta última se encuentra tocando los límites de la capacidad de su hardware, dado que se encuentra a un 200% de procesamiento y cuenta solamente con dos núcleos. Por lo tanto, es sencillo concluir que estos límites alcanzados tienen su explicación en los límites propios del hardware.

Dado que este equipo es pieza más que fundamental en lo que será la arquitectura final del sistema, es conveniente alojar al SIP Proxy en un mejor hardware. De esta manera se podrá intentar testear los límites propios de la herramienta, en este caso Kamailio 3.2, sin estar limitados por el hardware.

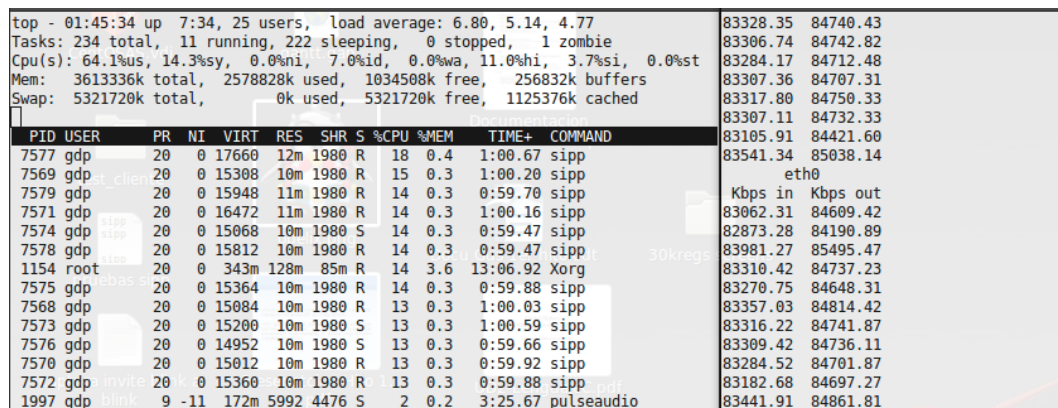
La siguiente prueba es similar a la descrita en las figuras 4.7 y 4.8, pero esta vez el SIP Proxy se alojara en un servidor HP modelo Proliant ML110 G6, el cual cuenta con un procesador Xeon Core i3 (4 núcleos) @ 3.06 GHz, 2GB de memoria RAM y una placa de red Gigabit Ethernet. La base de datos del sistema se alojara en un Notebook HP con procesador Core 2 Duo (2 núcleos) @ 2.0 GHz, 4GB de memoria RAM y una tarjeta de red Fast Ethernet.

Para lograr estresar al sistema, se debieron usar varios puntos de generación del tráfico (en particular 3) para poder asegurar que era el sistema el cual se estresaba y no los generadores. Se monitorearon, además de los componentes del núcleo de señalización, los equipos generadores de trafico con el fin de no sobre exigirlos.

Se describen a continuación los detalles de los equipos generadores del tráfico para esta prueba, así como también datos como nivel de procesamiento, uso de memoria y anchos de banda al momento que el sistema comenzaba a estresarse.

Generador 1, Core2Duo @ 3.0 GHz, 4GB de memoria RAM, tarjeta de red Gigabit Ethernet.

Este generador aportaba 12.000 escenarios de registros por segundo y en la figura 4.10 se pueden ver datos de procesamiento, uso de memoria y anchos de banda.



```

top - 01:45:34 up 7:34, 25 users, load average: 6.80, 5.14, 4.77
Tasks: 234 total, 11 running, 222 sleeping, 0 stopped, 1 zombie
Cpu(s): 64.1%us, 14.3%sy, 0.0%ni, 7.0%id, 0.0%wa, 11.0%hi, 3.7%si, 0.0%st
Mem: 3613336k total, 2578828k used, 1034508k free, 256832k buffers
Swap: 5321720k total, 0k used, 5321720k free, 1125376k cached

  PID USER      PR  NI  VIRT  RES  SHR  S %CPU  %MEM    TIME+  COMMAND
 5777 gdp        20   0 17660 12m 1980 R   18   0.4   1:00.67 sipp
 7569 gdp        20   0 15308 10m 1980 R   15   0.3   1:00.20 sipp
 7579 gdp        20   0 15948 11m 1980 R   14   0.3   0:59.70 sipp
 7571 gdp        20   0 16472 11m 1980 R   14   0.3   1:00.16 sipp
 7574 gdp        20   0 15068 10m 1980 S   14   0.3   0:59.47 sipp
 7578 gdp        20   0 15812 10m 1980 R   14   0.3   0:59.47 sipp
1154 root        20   0 343m 128m 85m R   14   3.6  13:06.92 Xorg
 7575 gdp        20   0 15364 10m 1980 R   14   0.3   0:59.88 sipp
 7568 gdp        20   0 15084 10m 1980 R   13   0.3   1:00.03 sipp
 7573 gdp        20   0 15200 10m 1980 S   13   0.3   1:00.59 sipp
 7576 gdp        20   0 14952 10m 1980 S   13   0.3   0:59.66 sipp
 7570 gdp        20   0 15012 10m 1980 R   13   0.3   0:59.92 sipp
 7572 gdp        20   0 15360 10m 1980 R   13   0.3   0:59.88 sipp
1997 gdp        9  -11 172m 5992 4476 S    2   0.2   3:25.67 pulseaudio

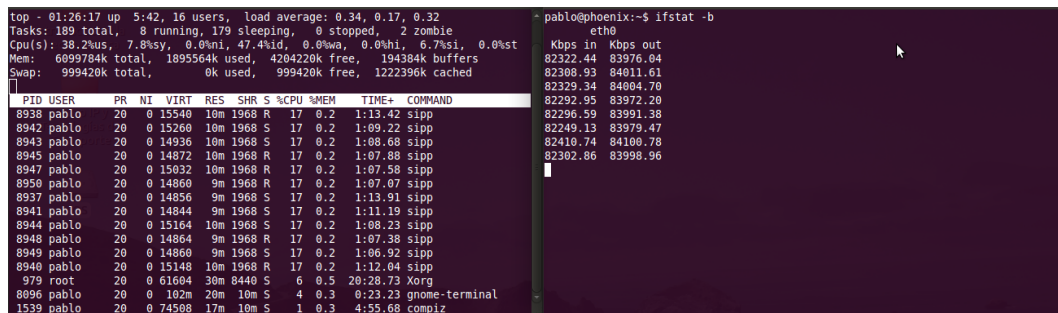
83328.35 84740.43
83306.74 84742.82
83284.17 84712.48
83307.36 84707.31
83317.80 84750.33
83307.11 84732.33
83105.91 84421.60
83541.34 85038.14
eth0
Kbps in Kbps out
83062.31 84609.42
82873.28 84190.89
83981.27 85495.47
83310.42 84737.23
83270.75 84648.31
83357.03 84814.42
83316.22 84741.87
83309.42 84736.11
83284.52 84701.87
83182.68 84697.27
83441.91 84861.81

```

Figura 4. 10 – Captura del comando “top” y uso de ancho de banda en generador 1

Generador 2, Core i5 @ 2.3 GHz, 6GB de memoria RAM, tarjeta de red fast Ethernet.

Este generador aportaba 12.000 escenarios de registros por segundo y en la figura 4.11 se pueden ver datos de procesamiento, uso de memoria y anchos de banda.



```

top - 01:26:17 up 5:42, 16 users, load average: 0.34, 0.17, 0.32
Tasks: 189 total, 8 running, 179 sleeping, 0 stopped, 2 zombie
Cpu(s): 38.2%us, 7.8%sy, 0.0%ni, 47.4%id, 0.0%wa, 0.0%hi, 6.7%si, 0.0%st
Mem: 6099784k total, 1895564k used, 4204220k free, 194384k buffers
Swap: 999420k total, 0k used, 999420k free, 1222396k cached

  PID USER      PR  NI  VIRT  RES  SHR  S %CPU  %MEM    TIME+  COMMAND
8938 pablo    20   0 15540 10m 1968 R   17   0.2   1:13.42 sipp
8942 pablo    20   0 15260 10m 1968 S   17   0.2   1:09.22 sipp
8943 pablo    20   0 14936 10m 1968 S   17   0.2   1:08.68 sipp
8945 pablo    20   0 14872 10m 1968 R   17   0.2   1:07.08 sipp
8947 pablo    20   0 15032 10m 1968 R   17   0.2   1:07.58 sipp
8950 pablo    20   0 14860 9m 1968 R   17   0.2   1:07.07 sipp
8937 pablo    20   0 14856 9m 1968 S   17   0.2   1:13.91 sipp
8941 pablo    20   0 14844 9m 1968 S   17   0.2   1:11.19 sipp
8944 pablo    20   0 15164 10m 1968 S   17   0.2   1:08.23 sipp
8948 pablo    20   0 14864 9m 1968 R   17   0.2   1:07.38 sipp
8949 pablo    20   0 14860 9m 1968 S   17   0.2   1:06.92 sipp
8940 pablo    20   0 15148 10m 1968 R   17   0.2   1:12.04 sipp
 979 root        20   0 61604 30m 8440 S    6   0.5  20:28.73 Xorg
8096 pablo    20   0 102m 20m 10m S    4   0.3   0:23.23 gnome-terminal
1539 pablo    20   0 74508 17m 10m S    1   0.3   4:55.68 compiz

Kbps in Kbps out
82322.44 83976.04
82308.93 84011.61
82329.34 84004.70
82292.95 83972.20
82296.59 83991.38
82249.13 83979.47
82410.74 84100.78
82302.06 83990.96

```

Figura 4. 11 – Captura del comando “top” y uso de ancho de banda en generador 2

Generador 3, Core2Duo @ 3.0 GHz, 2GB de memoria RAM, tarjeta de red Fast Ethernet.

Este generador aportaba 6.000 escenarios de registros por segundo y en la figura 4.12 se pueden ver datos de procesamiento, uso de memoria y anchos de banda.

```

top - 01:45:33 up 2:52, 15 users, load average: 0.00, 0.05, 0.07
Tasks: 160 total, 5 running, 155 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.2%us, 8.9%sy, 0.0%ni, 60.0%id, 0.0%wa, 2.7%hi, 2.2%si, 0.0%st
Mem: 2032892k total, 251084k used, 1781808k free, 21436k buffers
Swap: 4095992k total, 0k used, 4095992k free, 68920k cached

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2795	root	20	0	18188	11m	2116	R	15.0	0.6	1:06.75	sipp
2791	root	20	0	17040	10m	2120	S	14.6	0.5	1:04.53	sipp
2794	root	20	0	20300	13m	2116	R	14.6	0.7	1:06.80	sipp
2792	root	20	0	16736	10m	2116	R	14.0	0.5	1:01.40	sipp
2796	root	20	0	20128	13m	2120	R	14.0	0.7	1:05.60	sipp
2793	root	20	0	20436	13m	2116	S	12.6	0.7	0:59.62	sipp
1	root	20	0	2856	1388	1176	S	0.0	0.1	0:00.89	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
7	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/1
9	root	20	0	0	0	0	S	0.0	0.0	0:00.29	ksoftirqd/1
10	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/1
11	root	20	0	0	0	0	S	0.0	0.0	0:00.00	events/0

Figura 4. 12 – Captura del comando “top” y uso de ancho de banda en generador 3

Como se puede advertir de los datos de estos tres generadores, el núcleo de señalización logro el procesamiento de 30.000 registros por segundo durante un tiempo de prueba de 500 segundos de manera estable.

En las figuras 4.13 y 4.14, se aprecia el rendimiento de la base de datos del sistema y del SIP Proxy respectivamente.

```

top - 00:51:36 up 41 min, 5 users, load average: 0.71, 1.
Tasks: 170 total, 2 running, 167 sleeping, 0 stopped,
Cpu(s): 19.0%us, 25.8%sy, 0.0%ni, 40.9%id, 0.0%wa, 0.0%h
Mem: 2998120k total, 959332k used, 2038788k free, 11
Swap: 6704472k total, 0k used, 6704472k free, 49

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM
969	mysql	20	0	145m	20m	5824	S	127	0.7
4251	root	20	0	0	0	0	S	0	0.0
1	root	20	0	3048	1856	1252	S	0	0.1
2	root	20	0	0	0	0	S	0	0.0
3	root	20	0	0	0	0	S	0	0.0
6	root	RT	0	0	0	0	S	0	0.0
7	root	RT	0	0	0	0	S	0	0.0
9	root	20	0	0	0	0	S	0	0.0
11	root	0	-20	0	0	0	S	0	0.0

Figura 4. 13 – Captura del comando “top” y uso de ancho de banda en generador en base de datos

244164.1230632.9

244058.1230488.2

244209.0230640.7

244165.1230661.7

244186.9230593.4

244164.2230648.0

244166.3230524.7

eth0

Kbps inKbps out

244222.3230525.1

244656.7231043.9

244169.2230668.4

244403.1230843.7

244356.8230798.7

244315.6230757.9

244581.3231042.3

244440.2230869.5

244357.3230876.1

244724.7231110.4

244530.2230989.4

244568.2231043.6

244689.4231095.9

244631.1231079.7

244743.6231238.7

top - 00:49:17 up 49 min, 3 users, load average: 0.05, 0.05, 0.01

Tasks: 210 total, 13 running, 197 sleeping, 0 stopped, 0 zombie

Cpu(s): 57.6%us, 14.2%sy, 0.0%ni, 19.9%id, 0.0%wa, 0.0%hi, 8.3%si,

Mem: 2062936k total, 239980k used, 1822956k free, 19888k buffers

Swap: 6258680k total, 0k used, 6258680k free, 111508k cached

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4137	root	20	0	1040m	4468	3308	S	21.8	0.2	0:25.97	kamailio
4142	root	20	0	1040m	4468	3308	R	21.4	0.2	0:26.43	kamailio
4147	root	20	0	1040m	4468	3308	R	21.1	0.2	0:26.23	kamailio
4141	root	20	0	1040m	4468	3308	S	20.8	0.2	0:26.25	kamailio
4144	root	20	0	1040m	4468	3308	R	20.8	0.2	0:26.12	kamailio
4145	root	20	0	1040m	4468	3308	R	20.8	0.2	0:26.03	kamailio
4148	root	20	0	1040m	4248	3088	R	20.8	0.2	0:26.21	kamailio
4135	root	20	0	1040m	4248	3088	R	20.4	0.2	0:26.15	kamailio
4140	root	20	0	1040m	4468	3308	S	20.4	0.2	0:26.01	kamailio
4134	root	20	0	1040m	4476	3316	S	20.1	0.2	0:26.35	kamailio
4146	root	20	0	1040m	4468	3308	S	20.1	0.2	0:26.41	kamailio
4136	root	20	0	1040m	4468	3308	R	19.8	0.2	0:25.89	kamailio
4138	root	20	0	1040m	4468	3308	R	19.8	0.2	0:26.15	kamailio
4139	root	20	0	1040m	4468	3308	R	19.8	0.2	0:26.18	kamailio
4149	root	20	0	1040m	4468	3308	R	19.8	0.2	0:25.98	kamailio
4143	root	20	0	1040m	4468	3308	R	19.1	0.2	0:25.69	kamailio
1365	root	20	0	2856	632	516	S	0.3	0.0	0:03.90	lldpad

Figura 4. 14 – Captura del comando “top” y uso de ancho de banda en SIP Proxy

Hay que recordar que en estas pruebas la cadencia de registros por segundo utilizada es aquella en donde el núcleo de señalización se mantenía estable en cuanto a completar satisfactoriamente los escenarios de registro.

Con una cadencia levemente mayor (30.100 registros por segundo), el SIP Proxy demora en responder algunos mensajes más de 500 ms, comenzando así a recibir retransmisiones por parte de los generadores. No obstante los escenarios se llegan a completar, aunque en un tiempo levemente mayor.

Si la cadencia superara cierto umbral (encontrado cercano a los 31000 registros por segundo), el efecto de las retransmisiones es crítico. Al demorar más tiempo en el procesamiento de cada mensaje, las retransmisiones aumentan de manera considerable y ahogan aun mas al servidor logrando así un efecto de “bola de nieve” y el sistema comienza a descartar escenarios volviéndose inestable en este punto.

Varios puntos a tener en cuenta. De los datos extraídos de uso del hardware, tanto por la base de datos como por el SIP Proxy, se desprende que este no es una limitante en el procesamiento de la señalización. De esta manera se concluye que en estas condiciones el limitante es el propio software utilizado para esta tarea.

También destacar que si la tolerancia para realizar una retransmisión por parte de los generadores fuese mayor a 500 ms, el SIP Proxy podría haber soportado una cadencia mayor a la obtenida. El límite encontrado en el software radica en cuan cargado esta el mismo y en cuanto demora en responder a las solicitudes que se le realicen. El criterio de los 500 ms utilizado, si bien fue arbitraria su elección, es un valor razonable.

Si bien la prueba anterior muestra lo potente de la herramienta en el manejo de señalización y en su función de SIP Registrar, que ese volumen de usuarios se esté registrando en el sistema simultáneamente y de manera continua, es un escenario más que hipotético.

Para lograr un dimensionamiento correcto de la capacidad de la herramienta, en cuanto al volumen de usuarios capaz de brindarle el servicio que se pretende implementar, es pertinente otro tipo de prueba.

4.2.2.2 – Llamadas. Reglas de dimensionamiento.

Se intentará con la prueba descrita a continuación obtener datos que permitan aplicar la teoría de tele tráfico y de esta manera calcular la cantidad de usuarios que podrían hacer uso de este sistema.

La prueba consiste en dos partes. La primera determinara la cadencia máxima de llamadas capaz de establecer el SIP Proxy por segundo, y la segunda determinara la cantidad de llamadas capaz de ser cursadas en simultáneo.

La figura 4.15 describe el escenario de la primera parte de la prueba, y la figura 4.16 el escenario de la segunda parte.

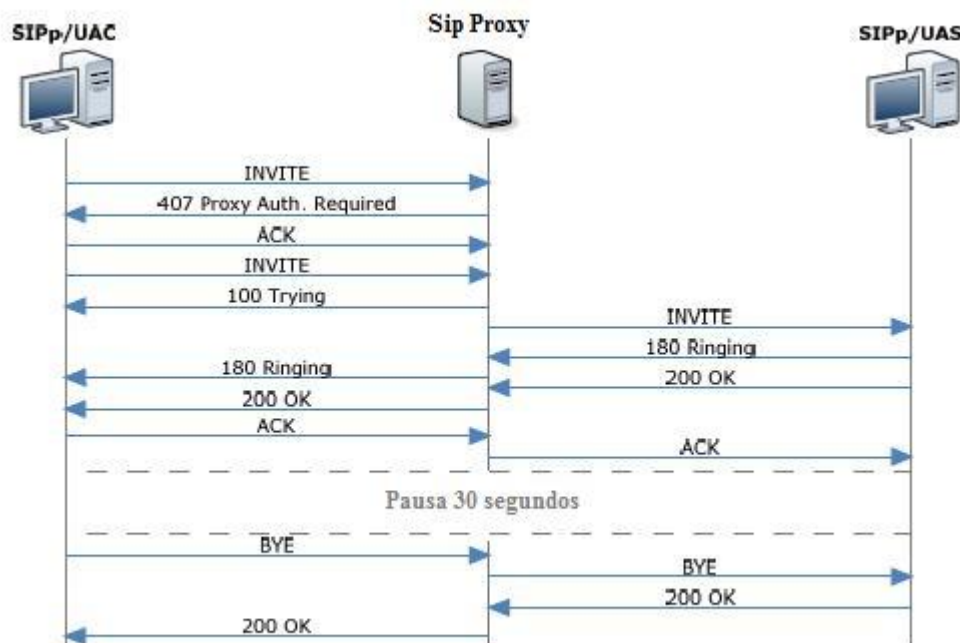


Figura 4. 15 – Esquema de la primera parte de la prueba de llamadas

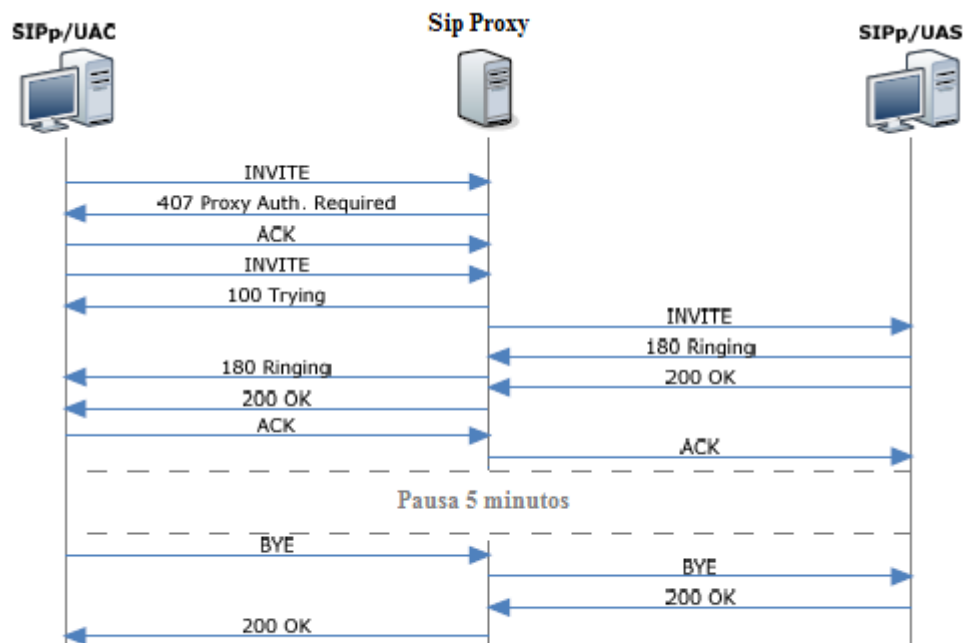


Figura 4. 16 – Esquema de la segunda parte de la prueba de llamadas

En este punto es importante aclarar el porqué de la diferenciación para esta prueba, y el motivo radica en que ambas capacidades del SIP Proxy están limitadas por factores bien distintos.

La cadencia máxima de llamadas por segundo con la que el SIP Proxy puede atender y establecer una llamada, está limitada principalmente por la velocidad de procesamiento del servidor. A mejor procesador, mejor será la performance en este sentido.

El número máximo de llamadas capaz de manejar en simultaneo el servidor está limitado por la memoria del mismo. Esto es así, dado que el servidor al trabajar en modo *stateful* (en el capítulo siguiente se verá el porqué de esta elección), este debe guardar el estado de cada llamada para poder asociar cada mensaje nuevo arribado con su llamada respectiva y así seguir rastro de cada una de ellas.

No es estrictamente cierto que procesador y memoria sean causas independientes en estos límites, dado que cuanto más memoria este ocupada con estados de llamadas, mas procesamiento se deberá realizar para asociar los mensajes que arriben con sus llamadas respectivas y de esta manera se vería disminuida la capacidad de procesamiento dedicado a establecer nuevas llamadas. No obstante, en esta prueba se trabajo en zonas donde no se exigía al

procesador y a la memoria al mismo tiempo, y por lo tanto se supusieron independientes.

Los datos extraídos de la primera parte de la prueba se muestran en la tabla 4.4 a continuación.

Llamadas por segundo	CPU (% promedio de los núcleos)
1000	30
2000	45
3000	60
4000	72
5000	84

Tabla 4. 4 – Tabla con revisión de nivel de procesamiento del SIP Proxy a distintas cadencias

Con estos resultados, se presenta la siguiente grafica (Figura 4.17) y una aproximación lineal con el fin de obtener una primera regla de dimensionamiento para el hardware.

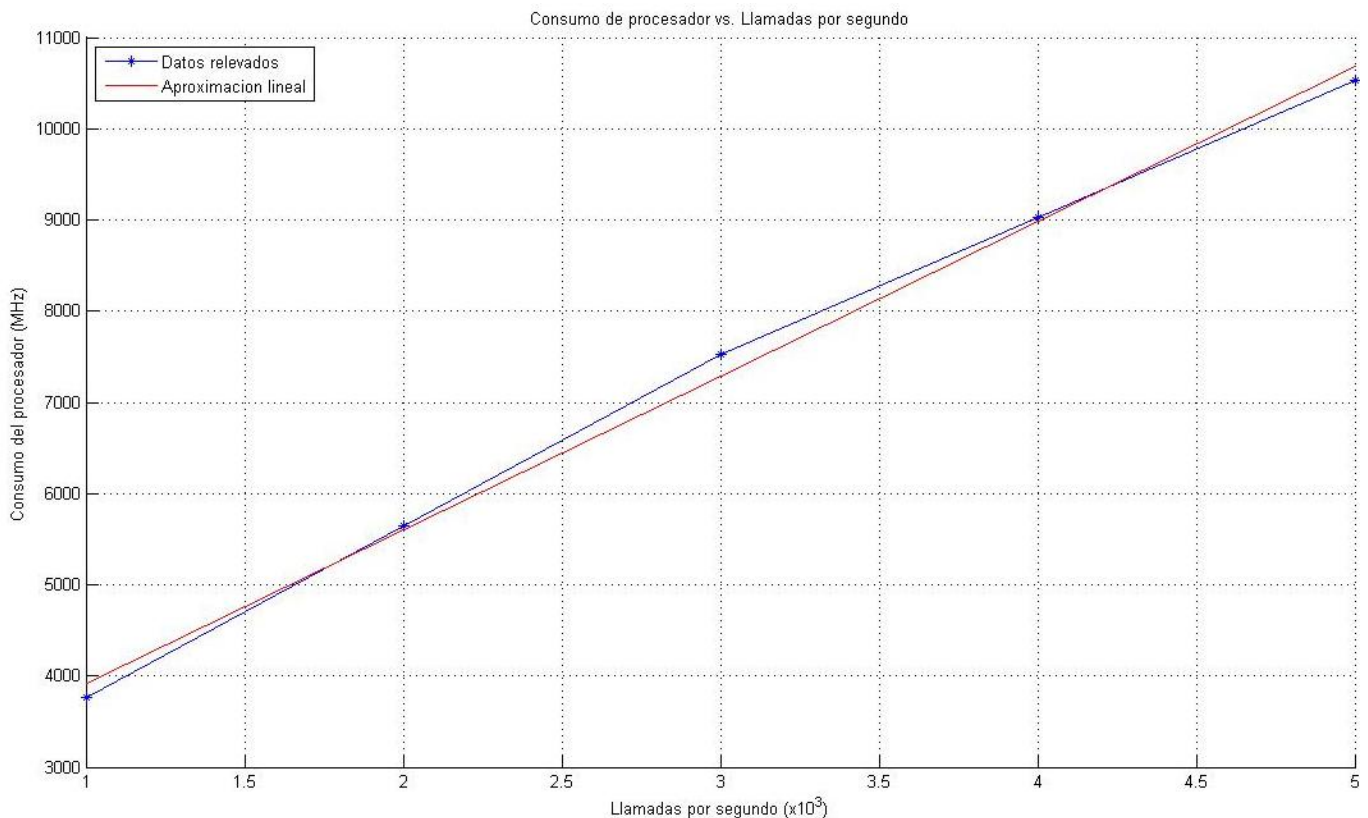


Figura 4. 17 – Grafico de “Procesamiento vs. Llamadas por segundo”

La aproximación lineal responde a la ecuación 4.1:

$$\boxed{\text{Procesamiento} = 1 \times 10^3 * (1.692 * \text{CPS} + 2.218)} \quad (\text{Ecuación 4.1})$$

Donde las unidades de ‘Procesamiento’ son MHz, y ‘CPS’ son las llamadas por segundo en miles.

Como en pruebas anteriores de este tipo, el limitante es el criterio tomado en la tolerancia antes de retransmitir un mensaje por parte de los generadores. En esta prueba era de 500 ms, por lo tanto cuando el SIP Proxy demora más de este tiempo en responder, comienzan las retransmisiones. Si las retransmisiones son en número importantes, ahogan al servidor disparando su procesamiento y llevando a este a encolar mensajes. Este comportamiento no está dentro de los aceptados en esta prueba y por lo tanto los resultados son aquellos en donde el servidor, si bien no está con una carga crítica en materia de procesamiento, procesa con cierta holgura el volumen de llamadas por segundo.

Cabe introducir un nuevo concepto en este punto. Este SIP Proxy, Kamailio, se controla mediante un archivo de configuración que le indica como tratar cada mensaje que recibe. El usuario puede programar cuanta lógica quiera en este archivo, y así tener libertad en la elección del comportamiento del SIP Proxy ante cualquier evento. Esto viene a colación por lo siguiente, y es que para cada mensaje que arriba al servidor este le aplica toda la lógica programada en ese archivo. Evidentemente cuanta más lógica y cuanto más complejo sea ese archivo, peor es el tiempo de procesamiento por mensaje recibido.

La prueba anterior fue realizada con el archivo de configuración final de este proyecto, en donde se verá en el anexo 1 las modificaciones de configuración y la lógica añadida para lograr el funcionamiento de la arquitectura final.

Para ilustrar el impacto del archivo de configuración que se use en el servidor, se mostraran los siguientes datos (Tabla 4.5) que corresponden a la misma prueba anterior (mismos criterios) pero usando un archivo de configuración *standard*, solo con modificaciones correspondientes al uso de la base de datos en un servidor externo.

Llamadas por segundo	CPU (% promedio de los núcleos)
1500	30.6
2000	38.3
4000	56.0
6000	75.2

Tabla 4. 5 – Tabla con revisión de nivel de procesamiento del SIP Proxy a distintas cadencias con archivo de configuración estándar

La siguiente grafica (Figura 4.18) presenta estos resultados, así como también una aproximación lineal.

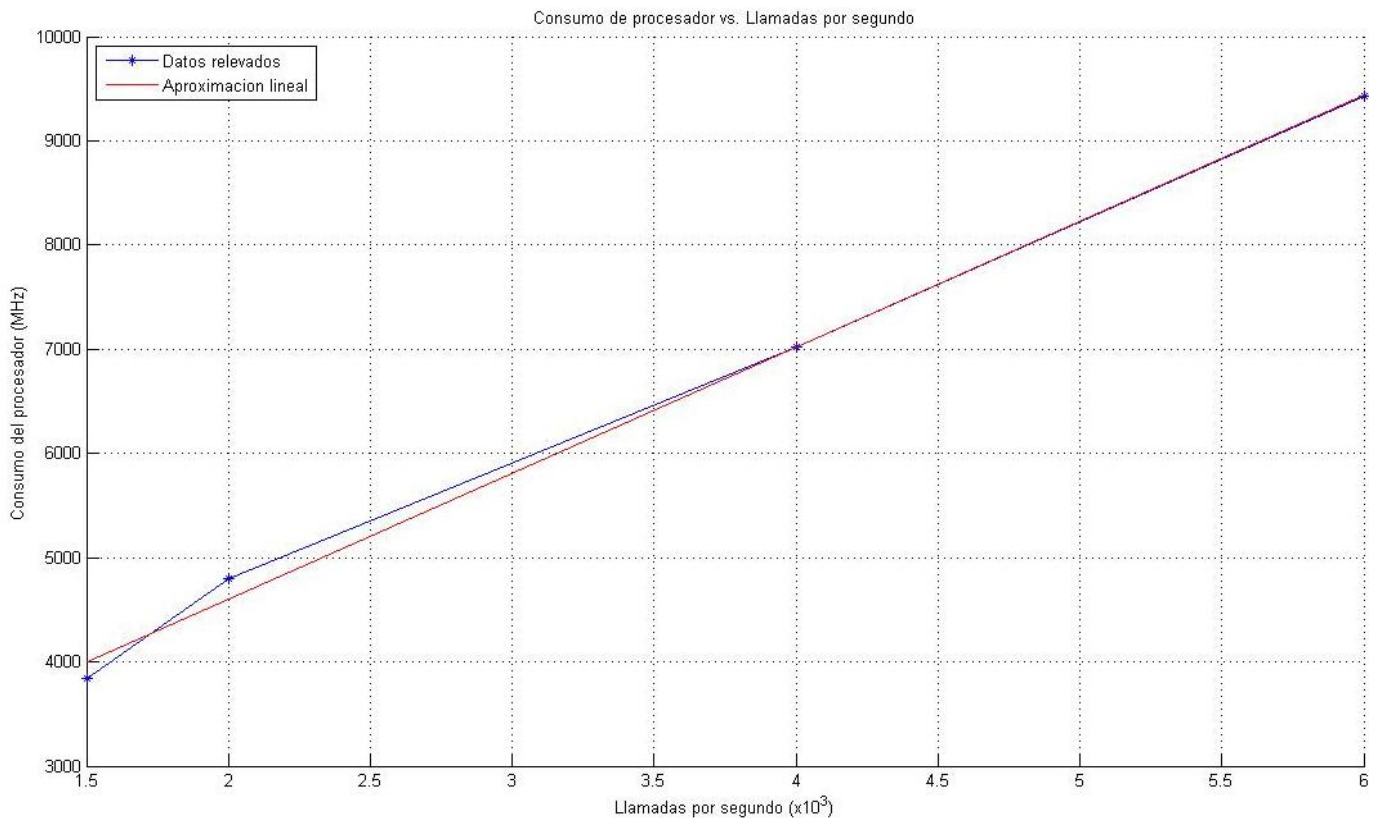


Figura 4. 18 – Grafico de “Procesamiento vs. Llamadas por segundo” con archivo de configuración estándar

La aproximación lineal responde a la ecuación 4.2:

$$\text{Procesamiento} = 1 \times 10^3 * (1.21 * \text{CPS} + 2.186) \quad (\text{Ecuación 4.2})$$

Donde ‘Procesamiento’ es en MHz, y ‘CPS’ son las llamadas por segundo en miles.

La tabla 4.6 presenta los resultados de procesamiento (aproximados) con ambos archivos de configuración.

Configuración:	Consumo aprox. (MHz/CPS)
Standard	1.21
Final	1.67

Tabla 4. 6 – Tabla comparativa del consumo del procesador con ambos archivos de configuración

Se observa claramente que el rendimiento con un archivo de configuración menos “pesado” en cuanto a lógica es mayor, y es dado el tiempo que demanda el procesamiento de cada mensaje. Mayor complejidad en la configuración trae como contrapartida la caída en la cadencia de llamadas por segundo capaz de procesar del servidor.

Los datos extraídos de la segunda parte de la prueba se muestran en la tabla 4.7 a continuación.

Llamadas acumuladas	Memoria en uso (MBytes)
258.052	584
311.206	629
355.270	652
407.258	720
445.329	721
496.328	744
550.250	767
619.343	813
667.366	858
711.321	882
787.504	905

Tabla 4. 7 – Tabla con revisión del uso de memoria del SIP Proxy cursando distintas cantidades de llamadas simultaneas

Con estos resultados, se presenta la siguiente grafica (Figura 4.19) y una aproximación lineal con el fin de obtener una segunda regla de dimensionamiento para el hardware.

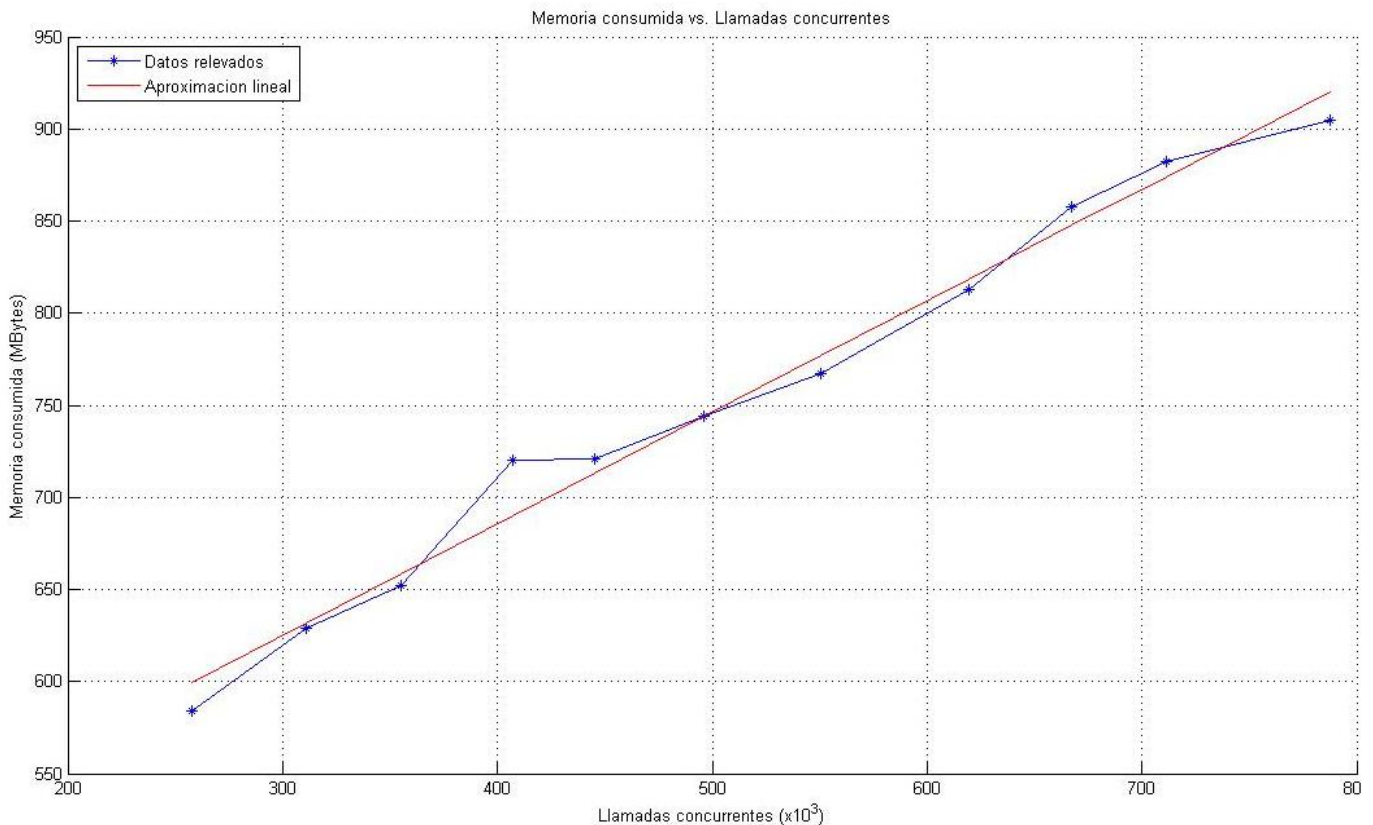


Figura 4. 19 – Grafico de “Memoria consumida vs. Llamadas concurrentes”

La aproximación lineal responde a la ecuación 4.3:

$$\boxed{Memoria = 0.6054 * Llamadas concurrentes + 443.59} \text{ (Ecuación 4.3)}$$

Donde ‘Memoria’ es en MBytes, y ‘CPS’ son las llamadas concurrentes en miles.

Vale aclarar que presentada regla de dimensionamiento es válida en el rango estudiado, e inferir resultados por fuera de este podría ser fuente de errores en el dimensionado. Sin embargo, se desprende un resultado importante de los datos extraídos anteriormente y es que la memoria no es una fuerte limitante como factor de diseño. Cualquier PC que cuente con 2 GB de memoria RAM, como la usada para estas pruebas, es capaz de soportar varios órdenes de llamadas concurrentes.

Otro punto a comentar sobre esta parte de la prueba es el rango analizado. Como se puede observar, la relación entre memoria consumida por cada mil llamadas concurrentes es poco mayor a 0.5 Mbyte. Por lo tanto, para observar un cambio significativo en el uso de la memoria se acumularon alrededor de 250.000 llamadas y en ese punto se comenzó el análisis. El final del intervalo analizado, correspondiente a alrededor de 750.000 llamadas concurrentes, fue

causado por la incapacidad por parte de los generadores de llamadas de poder seguir acumulando llamadas en simultáneo. La falta de equipos para esta tarea evitó el poder seguir obteniendo datos de un mayor número de llamadas concurrentes.

Por este motivo, en lo que sigue se usará el último dato obtenido como límite de llamadas concurrentes, aunque es de esperar un sub-dimensionamiento del sistema.

4.2.3 – Aplicación de la Teoría de Teletráfico

A continuación se aplicará parte de la teoría de teletráfico para comprender el alcance de los resultados obtenidos anteriormente [58] [59].

En particular, se usará la fórmula de Erlang-B o fórmula de pérdidas debido a que si el sistema se encontrara sin la capacidad de cursar una nueva llamada, esta sería rechazada.

Los datos necesarios para hacer este cálculo son el número de líneas (en este caso es la cantidad de llamadas en simultáneo capaz de cursar el servidor), el porcentaje de bloqueo admitido del sistema y el uso del sistema por parte de un usuario en la hora pico.

Con los primeros dos se calculará, en unidades de tráfico, la capacidad del sistema. Con el último se calculará el número de usuarios a los cuales se les podría dar el servicio.

Asumiendo la cantidad de líneas (cantidad de llamadas en simultáneo) como el último dato obtenido, este es 787.504.

El porcentaje de bloqueo del sistema se erigirá 1%, dado que es el usual en sistemas de tipo telefónico.

Con estos dos datos ya se puede obtener un número representativo del tráfico capaz de cursar el sistema. No obstante, es muy difícil encontrar tablas de Erlang-B que permitan como entrada un número tan grande de líneas, por lo que se debe recurrir a calculadoras de software. Se usó para el siguiente cálculo la calculadora Erlang Calculator v2.2⁷ para el sistema operativo Windows.

⁷ Se puede encontrar en <http://hp.vector.co.jp/authors/VA002244/erlang.htm>

$$\left. \begin{array}{l} \text{Número de líneas} = 787504 \\ \text{Porcentaje de bloqueo} = 1\% \end{array} \right\} \xRightarrow{\text{ErlB}} \text{Tráfico} = 795360 \text{ Erlang}$$

(Ecuación 4.4)

Con el valor del tráfico capaz de cursar el sistema y el uso del sistema de un usuario en la hora pico (en términos de tráfico), es sencillo calcular el número de usuarios soportado por el sistema de la siguiente manera:

$$\frac{\text{Tráfico total soportado}}{\text{Tráfico por usuario en hora pico}} = \text{Cantidad de usuarios}$$

(Ecuación 4.5)

Este uso del sistema por parte de cada usuario en la hora pico se tomara de los requerimientos de este proyecto, y equivale a 0.15 *Erlang/usuario*. Por lo tanto:

$$\left. \begin{array}{l} \text{Tráfico total} = 795360 \text{ Erlang} \\ \text{Tráfico por usuario} = 0.15 \text{ Erlang/usuario} \end{array} \right\} \Rightarrow \boxed{5.302.400 \text{ Usuarios}}$$

(Ecuación 4.6)

Es más que destacable el valor de usuarios capaz de ser soportado por este sistema, ya que sería capaz de soportar y manejar la señalización asociada a llamadas de aproximadamente una vez y media la población actual de todo Uruguay.

Subrayar también que no se trata de equipos que superan los cientos de miles de dólares, sino de servidores que no superan los 1000 dólares como es el caso del SIP Proxy.

En estos últimos resultados solo se tuvieron en cuenta los datos extraídos en la segunda parte de la prueba anterior, y no tuvieron en cuenta el consumo de procesamiento por parte del servidor.

En la última parte de este análisis se añadirán restricciones a estos resultados, que surgen de los datos extraídos en la primera parte de la prueba.

En dicha parte se encontró como limitante una cadencia máxima de arribo de llamadas por segundo determinada por la capacidad de procesamiento del

servidor. Para no sobrecargar al mismo, la cadencia promedio de arribos de llamadas por segundo debe estar alrededor de las 4000.

Aplicando nuevamente la teoría de Erlang, se tiene que el tráfico total soportado por el servidor responde a la siguiente fórmula:

$$\text{Tráfico} = \lambda \cdot t \text{ (Ecuación 4.7)}$$

donde λ es la cadencia promedio con la que arriban las llamadas, y t es el tiempo promedio de permanencia del usuario en el sistema o tiempo promedio de cada llamada.

Con la cadencia limitada, se verá si es coherente afirmar que el tráfico que puede cursar el servidor es el calculado anteriormente.

Aplicando la restricción de un máximo de 4000 llamadas por segundo y conociendo el tráfico máximo calculado (795.360 Erlang), se procederá a calcular el tiempo de duración de llamada crítico, en donde si el promedio de duración de las llamadas es menor a ese tiempo el servidor será incapaz de cursar su tráfico máximo. El resultado es el siguiente:

$$\left. \begin{array}{l} \lambda = 4000/s \\ \text{Tráfico} = 795360 \text{ Erlang} \\ \text{Tráfico} = \lambda \cdot t \end{array} \right\} \Rightarrow t = 198s$$

(Ecuación 4.8)

En este escenario, la duración media crítica de las llamas es de 198 segundos. Si el promedio está por debajo de ese resultado, para mantener el mismo tráfico la cadencia promedio de arribo de llamadas por segundo debe ser mayor a 4000, y eso provocara un ahogo del servidor.

Este tiempo promedio de duración de llamadas, 198 segundos (3 min y 18 seg), es una cota razonable para las llamadas cursadas en el sistema.

Sin embargo, si el sistema fuera más exigente en cuanto a la señalización, esto es que la duración promedio de las llamadas fuera menor al valor anterior, hay que calcular nuevamente la cantidad de usuarios que puede hacer uso del sistema.

Suponiendo que las llamadas tienen una duración promedio de un minuto (60 segundos), el nuevo cálculo arroja como resultado lo siguiente:

$$\left. \begin{array}{l} \lambda = 4000/s \\ t = 60s \\ \text{Tráfico} = \lambda \cdot t \end{array} \right\} \Rightarrow \text{Tráfico} = 240.000 \text{ Erlang}$$

(Ecuación 4.9)

Del tráfico cursado por usuario se obtiene:

$$\left. \begin{array}{l} \text{Tráfico total} = 240.000 \text{ Erlang} \\ \text{Tráfico por usuario} = 0.15 \text{ Erlang/usuario} \end{array} \right\} \Rightarrow \boxed{1.600.000 \text{ Usuarios}}$$

(Ecuación 4.10)

Análogamente, siendo más exigentes, un promedio de duración de llamadas de 30 segundos traería como consecuencia que la cantidad de usuarios capaces de hacer uso del sistema fuera de 800.000.

En resumen, existen dos zonas claramente diferenciadas. La primera en donde la capacidad del sistema en términos de tráfico está limitada por la capacidad de procesamiento del servidor, y esto se traduce en una relación lineal entre el tiempo de permanencia promedio de cada llamada y la cantidad de usuarios a los que el sistema puede servir, y la segunda zona donde la cantidad de usuarios es fija y limitada por la cantidad de canales capaz de manejar el servidor, sin importar el tiempo promedio de cada llamada.

La siguiente figura (figura 4.20) ilustra estas dos zonas, así como también zonas en donde las tuplas ‘número de usuarios-tiempo promedio de llamada’ son soportadas, o no, por el sistema (zona segura y zona prohibida respectivamente).

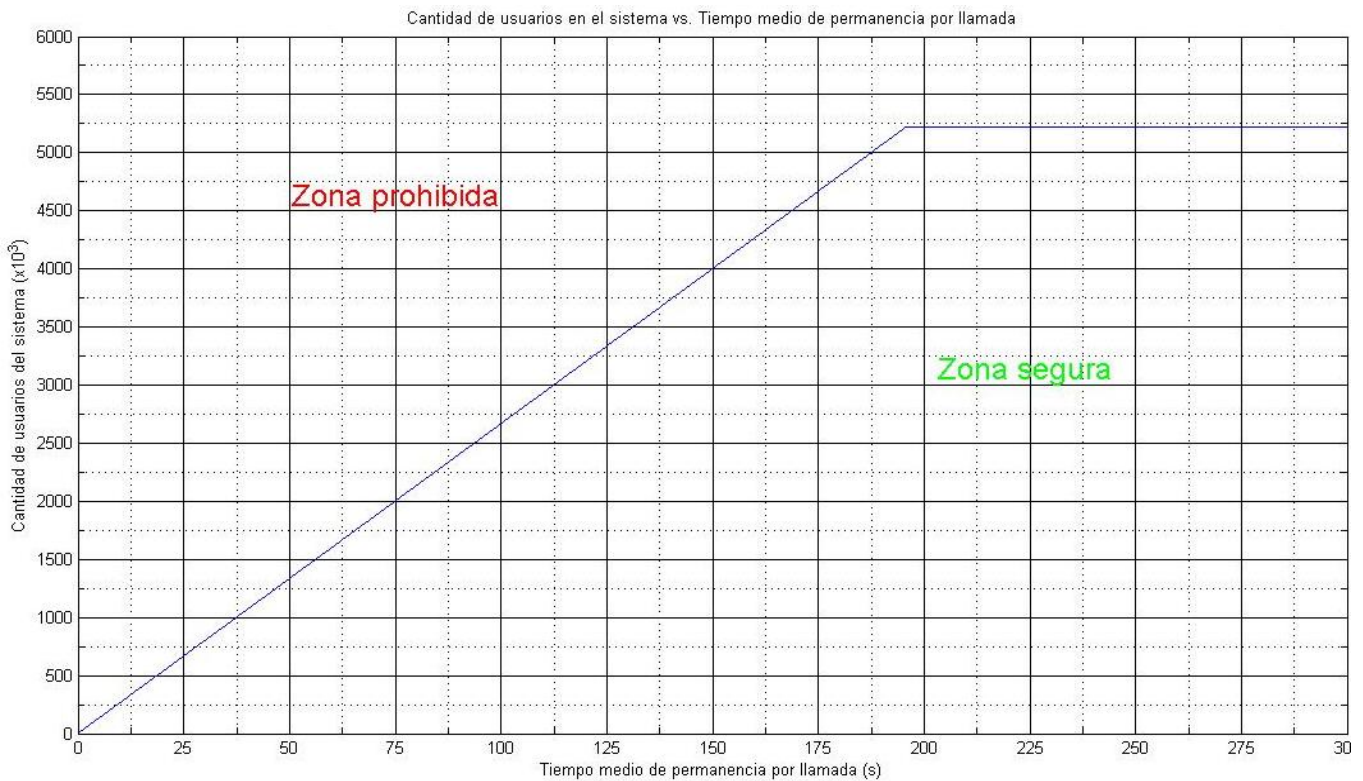


Figura 4. 20 – Grafico de “Cantidad de usuarios del sistema vs. Tiempo de permanencia”. Zona soportada.

Cabe aclarar que la hipótesis fundamental para la validez de dichas zonas es el trafico por usuario ($0.15 \text{ Erlang/usuario}$).

La zona lineal, que corresponde a tiempos de llamada menores a 196 segundos, es una recta de ecuación:

$$\text{Cantidad de usuarios} = 4000 / 0.15 * \text{tiempo medio de llamada}$$

(Ecuación 4.11)

Para tiempos mayores a 196 segundos, la cantidad de usuarios es fija y corresponde con la hallada en la ecuación 4.6, 5.302.400 usuarios.

Para bajar a tierra estos resultados y llevarlos a los requerimientos iniciales de este proyecto, basta con utilizar la ecuación 4.11. La cantidad de usuarios a brindarles el servicio es 10.000, entonces se desprenden fácilmente los siguientes resultados:

$$\text{tiempo medio de llamada} = \frac{10000 \text{ Usuarios} * 0.15 \text{ Erlang/Usuario}}{4000/s}$$

$$\Rightarrow \text{tiempo medio de llamada} = 0.375 \text{ s}$$

(Ecuación 4.12)

Esto quiere decir que si los usuarios hicieran llamadas de promedio mayor o igual a 0.375 segundos, el sistema soportaría. Este resultado límite es impensado si de llamadas se trata, dado que no hay llamada posible en tan corto tiempo.

De este modo, el tiempo medio de llamadas no es una limitante para el planteado escenario de usuarios.

Por último, suponiendo un tiempo promedio de llamadas de 30 segundos en nuestro sistema:

$$\begin{aligned} \text{Tasa de arribos} &= \frac{\text{Tráfico total}}{\text{Tiempo promedio}} \\ \text{Tasa de arribos} &= \frac{10000 \text{ Usuarios} * 0.15 \text{ Erlang/Usuario}}{30 \text{ segundos}} \end{aligned}$$

$$\Rightarrow \boxed{\text{Tasa de arribos} = 50 \text{ llamadas/segundo}}$$

(Ecuación 4.13)

Dado que el sistema puede soportar más de 4000 llamadas por segundo, el volumen de llamadas que recibiría en el caso anterior es insignificante y no le produce ningún *stress* en cuanto a procesamiento.

4.3 – El DNS

Para terminar este capítulo, hay que introducir un nuevo servidor al núcleo de señalización, y no es otro que un DNS.

Un DNS es una pieza fundamental en este tipo de sistemas, en donde la resolución de nombres es esencial. Si la dirección de un usuario en el sistema es “gus@obelix”, el teléfono de ese usuario resolverá el dominio “obelix” para enviar los mensajes de registro, y si el usuario llama a “rafa@obelix” su teléfono resolverá el dominio “obelix” para saber a dónde enviar las solicitudes de la llamada.

Si la empresa ya cuenta con un DNS, bastara con agregar una entrada para resolver el dominio “obelix”. Si no es el caso, hay que agregar un servidor que se dedique a esta tarea. En el anexo 1 se detalla la instalación y configuración de un servidor de este tipo (BIND).

También es conveniente agregar una entrada para resolver la dirección de la base de datos del sistema, por ejemplo “base.obelix”, para uso del SIP Proxy/Registrar.

4.4 – Arquitectura para el núcleo de señalización

Comentado esto, ya se puede presentar una arquitectura para el núcleo de señalización capaz de manejar un volumen enorme de llamadas y de mensajería SIP en general (Figura 4.21), así como algunos servicios que se comentaran más adelante. El precio que hay que pagar para obtener un rendimiento de estas características es el de no brindar servicios en el núcleo que requieran procesamiento de audio.

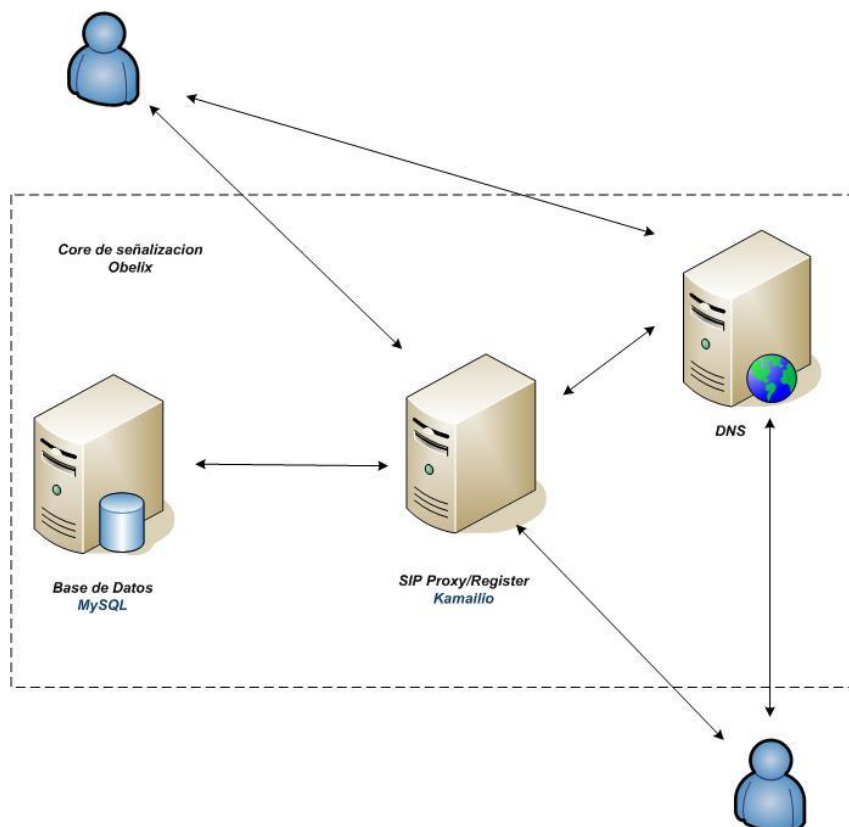


Figura 4. 21 – Arquitectura del núcleo de señalización

En el próximo capítulo se desarrollara como brindar estos servicios para completar el sistema de telefonía.

Capítulo 5 – Funcionalidades Corporativas

5.1 – Introducción

En el capítulo anterior se estableció una arquitectura de SIP *Proxy/SIP Registrar* la cual permite a los *User Agents*, en principio teléfonos SIP, poder registrarse y realizar llamadas entre ellos. Si bien las llamadas básicas (sin ningún valor agregado) son las mayoritarias dentro de un ambiente empresarial, también existen diferentes necesidades que se esperan de un sistema de comunicación corporativo.

Dentro de estas necesidades se requiere contar con un servicio de *Voicemail*, este permitirá al beneficiario la capacidad de “derivar” una llamada hacia su casilla en donde los usuarios que desean comunicarse con éste podrán dejar un mensaje. También aplica al caso cuando un usuario no está registrado en el sistema y otro *User Agent* desea comunicarse con él, la decisión usual de los diseñadores de sistemas telefónicos es derivar dicha llamada también a la casilla de *Voicemail* del “calleo”, Obelix implementa esto último.

Otra funcionalidad usual en el ambiente corporativo son los Puentes de Conferencias (*Conference Bridges*), son uno de los tipos posibles de conferencias a implementar descriptos en la RFC-5370. Desde el punto de vista de señalización y *Media* los *User Agents* interactúan con un servidor central, el mismo realiza la mezcla de audio (*Mixer*) y la envía a cada uno de los participantes. Por ende cada participante recibe la mezcla de todos los audios de todos los participantes menos el propio. Este es el tipo de conferencia que Obelix implementará, así como también dará soporte a las “conferencias de 3 patas”.

Dentro de las funcionalidades corporativas básicas también se encuentra el *Music On Hold* (denominado a partir de ahora “*MoH*”). Este *feature* consiste en agregar música (elegida por el sistema) que será escuchada por un usuario cuando se coloque en espera por otro. En el ambiente corporativo interesa que esta música sea proporcionada por un servidor perteneciente al núcleo del sistema telefónico en lugar del propio teléfono SIP, esto último es una gran ventaja del punto de vista de la administración centralizada y la uniformidad dentro de la empresa.

Las tres *features* mencionadas anteriormente involucran de alguna forma procesamiento de audio, por lo tanto serán implementadas por un grupo de *PBX IP* las cuales serán dimensionadas para soportar un determinado tráfico ofrecido.

Por último se implementaron otras funcionalidades que no requieren del mencionado procesamiento. Fueron resueltas enteramente con los elementos del *Core* de señalización detallados en el capítulo anterior, estas son Transferencias, Presencia, Mensajería Instantánea, manejo de Permisos y *Accounting*.

5.2 – *Music On Hold (MoH)*

5.2.1 – Descripción del Servicio

Este servicio tiene como principal objetivo reproducir música a un usuario mientras el mismo permanece en espera. Generalmente esta funcionalidad es utilizada para poner en espera a llamadas que utilizan líneas urbanas, de todas formas haremos el dimensionamiento para soportar a 10000 usuarios, asumiendo que un porcentaje de los 1500 *Erlangs* derivaran concurrentemente al *Music On Hold* en la *Busy Hour*.

5.2.2 – Casos de Uso

El caso de uso exitoso corresponde a dos usuarios registrados en el dominio que atiende el SIP *Proxy*, (de aquí en más @obelix) los cuales están cursando una llamada en la que uno de los dos usuarios pone en espera al otro mediante la opción de *Hold* en el *softphone*. Inmediatamente el usuario en espera debe experimentar música generada por el servidor de *MoH*. El escenario finaliza cuando el *User Agent Holder* efectivamente saca de espera al *User Agent Holdee* presionando nuevamente la opción *Hold* del *softphone*, ocurrido esto se podrá reanudar la conversación.

Se tuvieron en cuenta comportamientos anómalos de los *User Agents*:

- El caso en que ambos usuarios presionan a la vez la opción *Hold*. En dicho caso, el que lo haga primero será efectivamente el *User Agent Holder*.
- Si un usuario en espera utiliza la opción *Hold* de su *softphone* el sistema debe ignorarlo.

5.2.3 – FreeSWITCH Como Proveedor de Música en Espera

La arquitectura del sistema planteada al momento establece que los registros y las llamadas básicas son resueltas a nivel del *Core* de señalización (*SIP Proxy/SIP Registrar*). Se deriva inmediatamente que el elemento encargado de brindar la música en espera no puede ser dicho *Proxy*, ya que este no maneja ni procesa ningún tipo de *Media* (audio, video, etc.). Se debe agregar otro elemento de *software* que sea capaz de brindar esta y otras funcionalidades. Como se menciona anteriormente se utiliza la *IP PBX FreeSWITCH* para realizar este procesamiento.

La misma nativamente trae preconfigurada una gran variedad de funcionalidades entre las cuales se encuentra la reproducción de un stream de música en espera. Si todo el sistema consistiera únicamente en todos los usuarios directamente conectados a esta *PBX*, la implementación de esta funcionalidad sería automática, ya que está incluida “*out of the box*”.

Obelix, como se verá explícitamente en los siguientes capítulos, consiste en un proyecto de integración de estas herramientas con el fin de lograr escalabilidad y alta disponibilidad de servicios. El desafío de integración es un pilar importante de este capítulo.

Para implementar la integración entre *FreeSWITCH* y el *Core* de señalización, de forma de resolver el servicio de *Music On Hold*, *FreeSWITCH* debe ser avisado por parte del *Core* para entrar en el transcurso de una llamada establecida y brindar música en espera al *User Agent Holdee*. Los pasos lógicos que implementa la solución final son los siguientes:

1. El *SIP Proxy* detecta que un usuario desea poner en espera a otro.
2. El *SIP Proxy* obtiene el *SDP* del usuario que es puesto en espera, mediante un mecanismo de renegociación.
3. El *SIP Proxy* genera un *INVITE* hacia *FreeSWITCH* con la *URI MoH@MoH.obelix* con el *SDP* del usuario que al que le debe llegar la música en espera.
4. El *SIP Proxy* guarda todos los datos de la nueva transacción que creó. Debe estar alerta por eventos que impliquen que deba terminar esta transacción y por ende el flujo *RTP* del *MoH*, ya sea porque el *Holder* quiere restablecer la conversación o alguno de los 2 termina la llamada inesperadamente.

El diseño de esta solución es tratado en el subcapítulo 5.2.6. Los detalles de la implementación en el Anexo 1. Es necesario destacar en este punto que la *PBX*

ejecuta la misma lógica correspondiente a usuarios directamente conectados, entiende al SIP *Proxy* como un usuario más.

En consecuencia, *FreeSWITCH* implementará música en espera ante un INVITE recibido externamente que tendrá como destino un interno especial definido en él (interno “*MoH*”). Los mensajes INVITE llegarán destinados a *MoH@MoH.obelix*, por ende se debe configurar *FreeSWITCH* atendiendo dicho dominio, ya que por defecto únicamente atiende mensajes en los cuales la parte de dominio de la *URI* es su propia *IP* (literal), no así un dominio, por más que este se termine resolviendo en su *IP*.

Por último se deshabilita la autenticación de usuarios, esta autenticación ya la realiza el SIP *Proxy/Registrar*, también es posible agregar una lista de acceso configurada en *FreeSWITCH* la cual establecerá que no procesará paquetes SIP que no tengan como *IP* de origen la *IP* correspondiente al *Proxy*. Esto último es para evitar que existan usuarios que utilicen directamente el servicio sin pasar por el *Proxy*, que por ejemplo, puede estar realizando el *accounting* de sus sesiones.

La configuración completa está incluida en el Anexo 2.

5.2.4 – Dimensionamiento

El dimensionamiento de un servidor de música en espera se debe hacer mediante teoría de teletráfico, concretamente, utilizando alguna de las variantes de las fórmulas de *Erlang*. En este caso y para todos los dimensionamientos de este proyecto utilizaremos *Erlang B* debido a que no es de interés del proyecto manejar colas de espera. Optamos por un esquema donde el dimensionamiento realizado para la hora pico sea capaz de cursar el tráfico ofrecido con una probabilidad de bloqueo del 1%.

La razón principal por la que no preferimos utilizar *Erlang C* es la arquitectura final del sistema (se terminará de presentar en el capítulo 7), la cual permite escalar fácilmente colocando la cantidad de servidores que sean necesarios en paralelo para compartir el tráfico ofrecido mediante una técnica de balanceo de carga que se hará explícita en el capítulo 7. Esta forma de balanceo permite diferentes estrategias al diseñador del sistema, por ejemplo dada una cantidad de tráfico ofrecido destinado al servidor de *Music On Hold* y una cadencia de ingreso al mismo, el diseñador puede repartir la carga entre varios servidores económicos o colocar pocos servidores de mayor capacidad de cómputo, esto

queda a criterio de cada implementador y dota al sistema de flexibilidad económica en el manejo de los recursos.

La utilización de *Erlang C* se ajustaría al proyecto y en particular al dimensionamiento del servidor de *MoH* si aceptara que el usuario experimente un cierto retardo existente entre el momento que es puesto en espera y el momento en que efectivamente escucha la música. Esta forma de diseño claramente derivaría en un *Hardware* necesario más conservador a costas de encolar usuarios que de alguna manera ya fueron puestos en espera. Hechas las anteriores aclaraciones continuamos con el dimensionamiento.

Esta sección tiene como objetivo indicar cuantitativamente que perfil de tráfico de música en espera deberá soportar el sistema en su totalidad. La funcionalidad de *MoH* está mayormente orientada al caso en que un usuario corporativo coloca en espera a otra persona que se encuentra fuera de la empresa, por lo tanto guarda estrecha relación con la cantidad de líneas urbanas que la empresa debería contar. En este dimensionamiento nos alejaremos un poco de este caso y solo consideraremos la situación en que usuarios internos (usuarios registrados en el sistema) ponen en espera también a usuarios internos a la empresa. Este es un criterio claramente más exigente en términos de cantidad de líneas que debe atender la funcionalidad de *MoH* dada la gran cantidad de usuarios totales del sistema objetivo.

Por lo tanto, los datos iniciales son 10000 usuarios generando 1500 *Erlangs* en la hora pico a una cadencia de 50 arribos por segundo. Existen en promedio 1500 llamadas concurrentes en el sistema. Tomaremos como criterio de diseño que el 15% de esas llamadas son concurrentemente derivadas al servidor de *Music On Hold*, por lo tanto este deberá soportar 225 *Erlangs* en la hora pico. Considerando el criterio usual de 1% de probabilidad de bloqueo y aplicando *Erlang B* se obtiene que el servicio de *Music On Hold* debe contar con 247 canales o puertos disponibles.

Como conclusión de este apartado, la *PBX* que oficie de servidor de *Music On Hold* deberá tener la capacidad de mantener 247 canales simultáneos de música en espera. Se considera además que un usuario permanece en espera por unos 30 segundos en promedio, por lo tanto se puede estimar una cadencia promedio de ingreso al sistema de *MoH* que es de 8.23 llamadas por segundo, sobredimensionando se puede considerar una cadencia de 10 usuarios/seg.

El próximo paso de cara al diseño es verificar que *FreeSWITCH* (*IP PBX* elegida de este proyecto) soporta dicha carga, así como también obtener cual es la mayor carga soportada y bajo qué condiciones mediante pruebas de carga sistemáticas.

5.2.5 – Pruebas de Carga Unitaria

El motivo de estas pruebas es determinar el rendimiento de una *PBX IP* actuando como servidor de *Music On Hold*. Como resultado de las mismas se podrá determinar cuántos canales de audio se podrán manejar en forma simultánea y a que cadencia serán establecidos.

Como toda prueba de carga de *software* dependerá fuertemente del *Hardware* en el que se encuentre instalado, por lo tanto se hace explícito el mismo:

- CPU: Intel *Core 2 Duo* E8400 3.0 GHz
- Memoria *RAM*: 2 GB DDR2 800 MHz

La red utilizada para las pruebas es *Gigabit Ethernet* en puntos claves donde de otra forma sería un cuello de botella no atribuible al *software* y/o *Hardware* (extra a las interfaces de red). Básicamente consiste en un *switch Gigabit Ethernet TP-Link TL-SG1024*.

El *software* a testear es *FreeSWITCH* Version 1.0.head (git-1086cba 2011-05-23 22-51-43 -0500) instalado en *CentOS* 6.1, dicha distribución es recomendada por el equipo de desarrollo de la *PBX*. Aun así los mismos declaran que pueden existir problemas de rendimiento con las distribuciones actuales de *CentOS* 6, por lo tanto en el Anexo 2 nos dedicaremos a probar puntualmente y en forma comparativa una versión concreta de *FreeSWITCH* en dos de distribuciones de *CentOS*, una supuestamente problemática, *CentOS* 6.1, y la recomendada *CentOS* 5.8.

Una prueba de carga consiste en estresar al sistema de determinada forma para extraer su máximo rendimiento, dicha performance está determinada por múltiples parámetros. En el mundo de la telefonía clásica existen muchos estándares para la medida de estos parámetros, pero aún no existen dichas estandarizaciones (tampoco recomendaciones) análogas para la señalización SIP. Por lo tanto en cada prueba realizada se harán explícitos los criterios de éxito que se tuvieron en cuenta para la misma.

El tráfico de señalización se generará con el *software SIPp*, este es ampliamente el más usado por la comunidad SIP, cuenta con una documentación que no está

orientada al usuario novato pero aún así es una herramienta flexible en la creación de escenarios de test, la totalidad de los escenarios fueron creados con dicha herramienta.

El escenario de pruebas consiste básicamente en llamar a un interno definido específicamente para el servicio (interno “*MoH*”) el cual está configurado en el *Dialplan* del *FreeSWITCH* que oficia de servidor de *Music On Hold*. En dicho interno atenderá la aplicación *playback* de la *PBX* la cual se encargará de enviar audio en paquetes *RTP* hacia el puerto *UDP* negociado en el escenario. El *SIPp UAC (User Agent Client)* no enviará audio, pero sí recibirá.

La funcionalidad de *MoH* se debe pensar como una llamada de un solo extremo, por lo tanto el *B2BUA* solo establecerá una sesión por llamada (notar que una llamada estándar constaría dos extremos o *legs*). A continuación se muestra una captura de pantalla del generador de tráfico en donde (entre otra información) se encuentra un diagrama de flujo que representa el escenario de pruebas generado. Se observa que corresponde a una llamada SIP estándar desde el punto de vista del UAC.

```
----- Scenario Screen ----- [1-9]: Change Screen --
Call-rate(length)  Port  Total-time  Total-calls  Remote-host
120.0(0 ms)/1.000s  5060  66.09 s    7931  192.168.2.11:5060(UDP)

120 new calls during 1.002 s period  0 ms scheduler resolution
103 calls (limit 360)                Peak was 123 calls, after 45 s
0 Running, 4065 Paused, 371 Woken up
0 dead call msg (discarded)          0 out-of-call msg (discarded)
3 open sockets

Messages  Retrans  Timeout  Unexpected-Msg
INVITE ----> 7931    0        0          0
100 <----- 7931    0        0          0
200 <----- E-RTD1 7929    0        0          0
ACK ----> 7929    0        0          0
Pause [ 833ms] 7929    0        0          0
BYE ----> 7829    0        0          0
100 <----- 0      0        0          0
200 <----- 7828    0        0          0

----- [+-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

Figura 5. 1 - Captura de escenario en terminal

La pausa corresponde a la ventana de tiempo en donde el usuario está escuchando música, en el ejemplo serían 833 ms. Este ejemplo claramente no es una situación real (posible, pero no probable) ya que el usuario es puesto en espera por menos de 1 segundo, pero a los efectos de generar 100 llamadas

concurrentes ingresando a una cadencia de 120 por segundo es la forma correcta de lograrlo. La figura anterior tiene como objetivo mostrar la señalización involucrada en el escenario de pruebas, ya que la cadencia de 120 CPS no fue evaluada en las pruebas.

Para todas las *Request* se configura un timer de 500 ms a partir del cual los mismos se retransmiten. Este timer es importante ya que es el principal criterio de éxito de nuestro escenario, el mismo consiste en no permitir retransmisiones en masa, aceptándose retransmisiones esporádicas controlables. El archivo con la configuración del escenario correspondiente a esta prueba (y todas las pruebas de este capítulo) se encuentra en el Anexo 3.

Los parámetros de entrada son:

- Cadencia de ejecución del escenario, en términos telefónicos se mapea en CPS (*Calls Per Second*) lo cual indica la cantidad de usuarios que ingresan al sistema por segundo, parámetro de ejecución “r” en *SIPp*.
- El tiempo de permanencia en el sistema, este parámetro es ajustable dentro de la configuración del escenario mediante el comando “pause”, en este escenario representa la cantidad de tiempo que el usuario experimenta *Music On Hold*. Notar que el producto de la cadencia por el tiempo de permanencia tiene como resultado la cantidad de líneas que el sistema tiene abierto en régimen, o en términos telefónicos la cantidad de llamadas concurrentes.

Existe un claro compromiso entre los parámetros anteriormente mencionados. En una primera aproximación ponen a prueba diferentes aspectos del servidor en conjunto con la aplicación *PBX*. En concreto existen infinitas formas teóricas de tener la misma cantidad de llamadas concurrentes en el sistema. Una cantidad fija de llamadas concurrentes pueden ser logradas con alta cadencia (CPS) y bajo tiempo de permanencia o baja cadencia y alto tiempo de permanencia, siempre y cuando el producto cadencia por tiempo de permanencia se mantenga constante. Por lo tanto, en primera instancia se podría inferir que al igual que en la prueba de carga del *SIP Proxy* del capítulo 4 las CPS consumen tiempo de procesador mientras que la cantidad de llamadas concurrentes consumen memoria *RAM* del sistema, cada una de estas relaciones además representadas en forma lineal como se expuso en dicho capítulo. En esta prueba la situación es diferente, dado que además de señalización hay manejo de audio por parte de la *PBX*. Esta debe generar el audio *RTP* y enviarlo al usuario final, el *SIP Proxy* únicamente se encarga de la señalización entre los *UA*’s. La *PBX*

debe generar y mantener los canales de audio por lo tanto cuanto mayor sea la cantidad de canales abiertos (o llamadas concurrentes) mayor será el consumo de *CPU* por más que la cadencia de llamadas (CPS) se mantenga constante a lo largo de toda la prueba. Esta es una clara diferencia con el comportamiento del *Proxy*, el mismo insume más tiempo de procesador únicamente si se le aumenta la cantidad de escenarios por segundo (CPS).

Controlando las entradas del escenario se intentará analizar este comportamiento. La idea consiste en generar una cantidad fija de llamadas concurrentes pero a diferente cadencia (CPS), de forma de analizar el compromiso existente entre CPU, CPS, memoria *RAM* y llamadas concurrentes.

Los parámetros de entrada se regularán para generar las diferentes situaciones

- Llamadas concurrentes: 100, 500, 900, 1200 cada caso con las posibles cadencias CPS 10, 30, 50, 70, 90 y 100.

La metodología consiste en:

1. Ejecutar el escenario.
2. Dejar arribar el sistema al régimen de llamadas concurrentes esperado.
3. Controlar que ningún generador de tráfico (*SIPp*) se encuentre saturado de procesador o memoria. Si así fuera la prueba pierde validez.
4. Analizar CPU y memoria *RAM* consumidos por la *PBX*.
5. Analizar ancho de banda generado por la *PBX*.
6. Implementar con un *softphone* una llamada extra hacia el interno de *Music On Hold* del servidor en carga con el objetivo de evaluar la calidad de audio que experimentarían los usuarios.

El audio generado por *FreeSWITCH* será con el códec *G.711* ley u, por lo tanto es de interés para la precisión de cálculos posteriores evaluar en forma práctica cual es el ancho de banda efectivo de dicho códec para una llamada. Para esto se implementa una llamada al interno de *Music On Hold* y se muestrea el ancho de banda recibido en un *PC* con Linux mediante el comando “*ifstat -b*”. Se toman 20 muestras y se realiza un promedio el cual es de aproximadamente 85.9kbps.

Cabe destacar que este procedimiento fue utilizado para extraer los anchos de banda de interés.

El protocolo de transporte utilizado es *UDP*, por lo tanto con la velocidad del códec (64 kbps) y la ventana de tiempo de muestreo se puede calcular teóricamente este ancho de banda.

Por otro lado también es necesario conocer el ancho de banda que insume la señalización a cada cadencia (CPS) para posteriormente distinguirlo del total generado (saliente) por la *PBX*. El sistema permite traficar por una interfaz Ethernet solo señalización y por otra el *RTP*, pero por motivos de disponibilidad esto no fue implementado. Por lo tanto se procedió a modificar la configuración en el *Dialplan* de la *PBX* para que atienda las llamadas pero no genere *RTP*, esto se logró con la aplicación “*sleep*”, dicha configuración se puede encontrar en el Anexo 3. En la siguiente tabla se muestra dicha revisión.

Cadencia (CPS)	Ancho de banda señalización saliente (kbps)
10	153
30	458
50	719
70	1111
90	1400
100	1530

Tabla 5. 1 Ancho de Banda de Señalización en función de cadencia de llamadas.

Con los resultados previos, se procedió con la metodología descripta anteriormente. Los resultados y análisis se muestran en las siguientes tablas y figuras.

Llamadas Efectivas@85.9kbps	Llamadas concurrentes			
CPS	100	500	900	1200
10	100	498	885	1021
30	100	497	750	738
50	100	464	529	570
70	100	409	559	562
90	100	422	505	490
100	100	384	515	413

Tabla 5. 2 Llamadas “efectivas” (en negrita) en función de la cadencia CPS y cantidad de llamadas concurrentes.

La tabla anterior muestra la cantidad de audios de música en espera de 85.9kbps exactos generados por *FreeSWITCH* para cada escenario de prueba. Por ejemplo en un escenario donde la *PBX* está cursando 500 llamadas concurrentes a una cadencia de 50 CPS, *FreeSWITCH* generaría 464 canales perfectos de 85.9kbps de música en espera de acuerdo al códec *G711 ley u*. Esta tabla no representa perfectamente la realidad ya que la *PBX* abre todos los canales correspondientes a la cantidad de llamadas concurrentes solicitadas, pero a medida que la cantidad de llamadas concurrentes aumenta o la cadencia de las mismas aumenta, el ancho de banda de audio no acompaña en dicho aumento en forma lineal.

Se deduce rápidamente que se comienza a degradar todos los flujos de *RTP* que llegan a los usuarios. Cada uno de ellos recibiendo menos de 85.9 kbps, conforme la exigencia de la prueba aumenta, se aprecian *glitches* o interrupciones en el audio que claramente pueden ser percibidas por el usuario y no son aceptables. Por lo tanto el espíritu de la tabla anterior es mostrar en función del ancho de banda saliente a la *PBX* (descontando el insumido por señalización) cuantas llamadas concurrentes son efectivamente cursadas a 85.9kbps.

Analizando detenidamente los resultados obtenidos se observa que al aumentar la cantidad de llamadas concurrentes a cadencia constante, por ejemplo 10 CPS, la *PBX* comienza a degradar los canales cada vez despegándose de los valores teóricos de canales concurrentes, por lo tanto existen entradas en la tabla correspondientes a puntos de funcionamiento del sistema que no son de utilidad práctica.

Este análisis deja como conclusión que es posible tener 1000 llamadas concurrentes de *Music On Hold* (de perfecta calidad) pero no a cualquier cadencia de entrada, en particular será posible si las mismas arriban al sistema en una cantidad de 10 por segundo. Pero no sería posible si las mismas 1000 llamadas concurrentes arribarían a 30 por segundo.

En el siguiente gráfico (figura 5.2) se aprecia el comportamiento cualitativo de la información expresada en la tabla anterior. Se representa la cantidad de canales sin distorsión en función de 2 variables independientes, la cantidad de llamadas por segundo CPS y la cantidad de canales (o llamadas concurrentes esperadas). Si el comportamiento fuese el ideal se observaría un plano, este representaría que todas las llamadas concurrentes son entregadas en forma perfecta desde el punto de vista de calidad del audio a cualquier cadencia de prueba. Esto

claramente no fue así sobre todo a cadencias superiores a 30 donde se observa que no es posible mantener la pendiente de la función respecto a los canales totales. Ésta pendiente sí se observa en forma constante para el caso de 10 CPS.

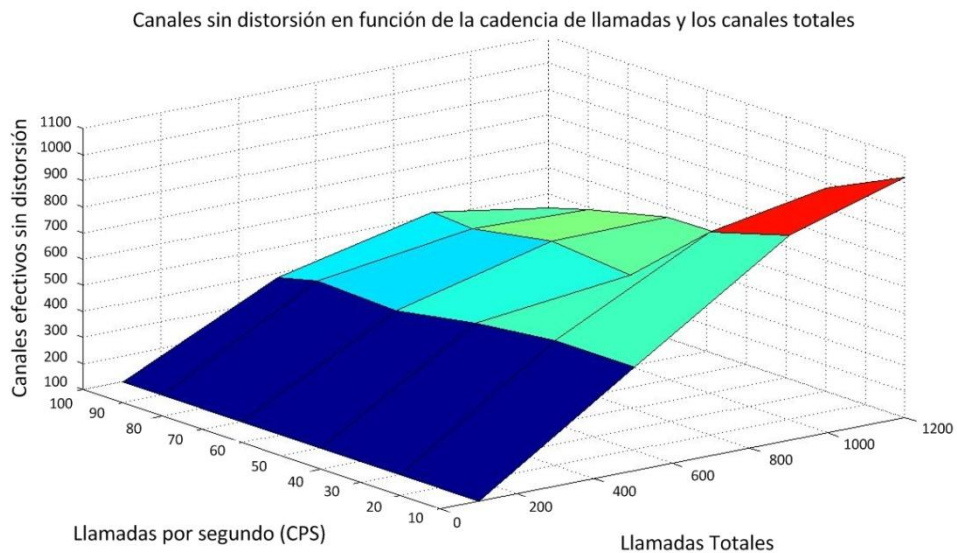


Figura 5. 2 - Gráfico de canales sin distorsión en función de la cadencia de llamadas y canales totales

A continuación se muestran los datos recabados correspondientes al comportamiento del *Hardware* en cada prueba.

CPU(%)	Llamadas concurrentes			
CPS	100	500	900	1200
10	12.8	52.8	88.5	95.4
30	30	88.1	95.8	96
50	46.9	94.9	96	95.7
70	63.7	96.1	95.7	95.7
90	66.8	94.9	95.8	96.3
100	86.6	95.8	94.5	96.3

Tabla 5. 3 Porcentaje de CPU consumido por la PBX en función de CPS y llamadas concurrentes.

En la tabla anterior se muestra el porcentaje de uso del CPU para cada cadencia y cantidad de llamadas concurrentes generada. Se puede observar que para una

cantidad fija de llamadas concurrentes, por ejemplo 100, a medida que aumentamos la cadencia de ingreso de llamadas CPS la carga en el CPU aumenta variando desde 12.8%@10CPS hasta 86.6%@100CPS. También se puede observar como aumenta la carga del CPU a medida que aumentamos la cantidad de llamadas concurrentes a CPS constante.

El siguiente gráfico (figura 5.3) muestra la misma información de la tabla anterior.

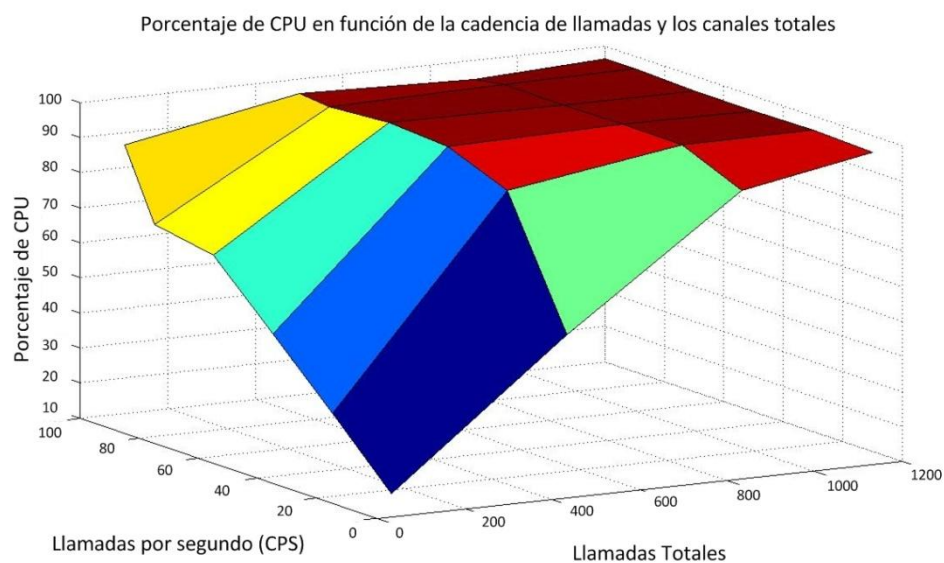


Figura 5. 3 - Porcentaje de CPU en función de la cadencia y llamadas totales (concurrentes).

Es interesante observar las zonas donde se dan los máximos de carga en el CPU (zonas rojas en la figura), estas zonas coinciden perfectamente para los valores de cadencia y llamadas concurrentes en los que el sistema comienza a perder calidad en el audio (zonas que en la función de la figura 5.1 comienzan a perder pendiente a CPS constante).

De esta forma se encuentra la limitante de esta prueba, el porcentaje de utilización del CPU. Se infiere directamente que con la configuración de *Hardware* que se cuenta no se debe trabajar con los puntos de funcionamiento contenidos en la zona roja del gráfico (figura 5.3) ya que en los mismos la calidad del audio de *Music On Hold* no es aceptable.

- Con el dimensionamiento realizado en la sección anterior y este análisis de la prueba de carga, podemos asegurar que nos encontramos en un punto de funcionamiento de calidad óptima y que con un solo servidor de características similares al de pruebas es suficiente para soportar la carga de un sistema en producción.

Las siguientes figuras expresan los datos obtenidos sobre el porcentaje de uso del CPU en función de la cantidad de llamadas concurrentes para el caso de 10 CPS.

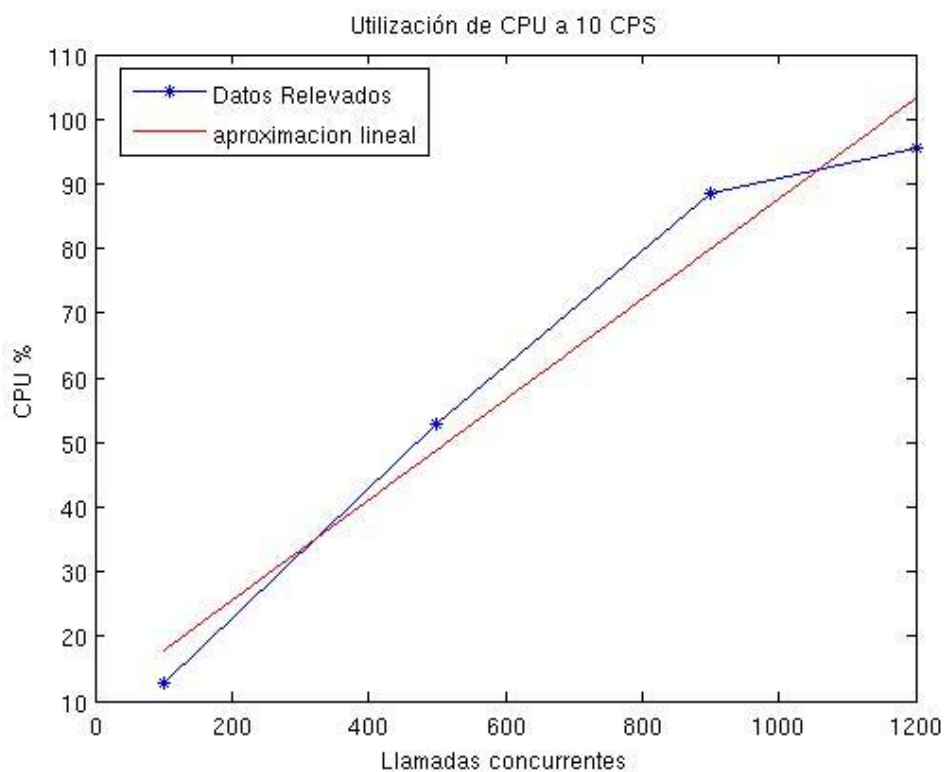


Figura 5. 4 - Porcentaje de CPU utilizado en función de las llamadas concurrentes

En la figura 5.4 se observa un crecimiento prácticamente lineal en el uso del CPU en el entorno de las 100 a 900 llamadas concurrentes. A partir de las 900 llamadas concurrentes se observa una clara zona de saturación donde la pendiente ya no es la misma, para las 900 llamadas concurrentes el procesador se encontraba al 88.5%, si la pendiente se mantuviera, claramente se superaría el 100% de uso del tiempo del procesador para 1200 llamadas concurrentes, por lo tanto a partir de las 900 llamadas concurrentes a 10 CPS no es un punto de

funcionamiento recomendado como ya se observaba en la Figura 5.3 (dentro de la zona roja prohibida).

Es posible entonces hacer una regresión lineal al comportamiento obtenido entre las 100 y 900 llamadas concurrentes, a cadencia 10 llamadas por segundo, para así obtener una recta de carga útil para predecir qué porcentaje de CPU será utilizado por *FreeSWITCH* en dicho entorno. Cabe destacar que estos resultados son muy difíciles de generalizar a cualquier configuración de *Hardware* existente, una buena práctica es repetir estas pruebas en forma sistemática para cada servidor con que se cuente de forma de poder conocer con exactitud su performance frente a las diferentes situaciones de interés de acuerdo a la aplicación que se esté implementando.

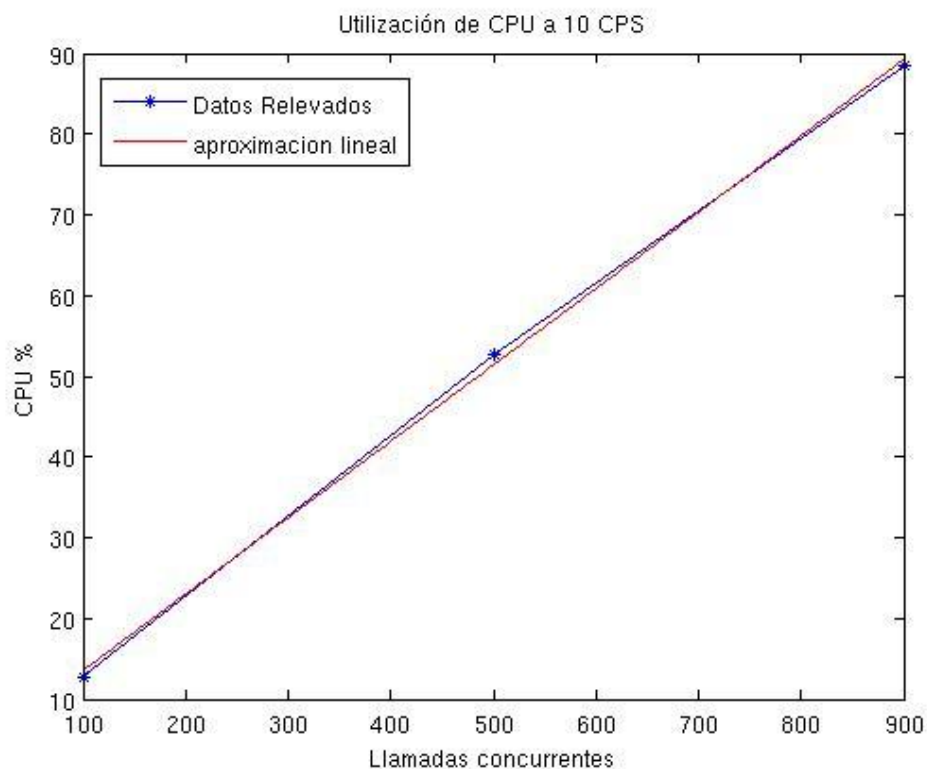


Figura 5. 5 - Porcentaje de CPU utilizado en función de las llamadas concurrentes

En la figura anterior se ajusta mediante regresión lineal los datos extraídos ahora en el rango que definimos como seguro.

La recta es entonces:

$$CPU(\%) = 0.0946 * Llamadas concurrentes + 4.0542$$

Se deduce entonces para las 247 llamadas concurrentes diseñadas en el dimensionamiento de la sección anterior, el CPU tendría una carga correspondiente a 27.4 %.

A continuación se presenta en la siguiente tabla la revisión de memoria RAM para los escenarios todos los escenarios de prueba.

Memoria RAM (MB)	Llamadas concurrentes			
CPS	100	500	900	1200
10	52	180	308	X
30	63	190	524	X
50	74	233	X	X
70	86.7	232	X	X
90	108	263	X	X
100	111	X	X	X

Tabla 5. 4 Memoria RAM (negrita) en función de CPS y Llamadas Concurrentes

Los escenarios marcados con “x” corresponden al caso donde el escenario no tuvo un comportamiento estable, por comportamiento estable entendemos que la prueba sea capaz de soportar 100000 llamadas (o escenarios, parámetro controlable por *SIPp* con “-m”). El error que se produce a los minutos de comenzado el escenario se manifiesta en el log de la siguiente forma:

[CRIT] mod_local_stream.c:297 Leaking stream handle!

La causa de este error proviene del funcionamiento de la aplicación *FreeSWITCH* que implementa la reproducción del *MoH*, esta aplicación se puede pensar como un “*Playlist*” que se está ejecutando en *loop* infinito (siempre y cuando exista al menos una solicitud de *MoH*), cuando la aplicación es llamada para abrir un nuevo stream de este *loop* hacia un usuario, se crea un “tubo” hacia dicho *stream* del cual se debe recoger el audio a entregar al usuario.

El problema se genera cuando el CPU esta en gran carga y no tiene tiempo de ir a buscar los datos de este stream para entregarlos al usuario a tiempo, entonces se produce un “*Leaking Stream*”. Como resultado la *PBX* cuelga, liberando el CPU pero no así la memoria consumida. Por lo tanto los puntos de funcionamiento donde esto ocurre no pueden ser considerados, en la siguiente figura se incluyen en el análisis pero solo a efectos cualitativos ya que existe una gran dispersión de

valores que puede tomar la memoria al final de un escenario que finalice con “leak”.

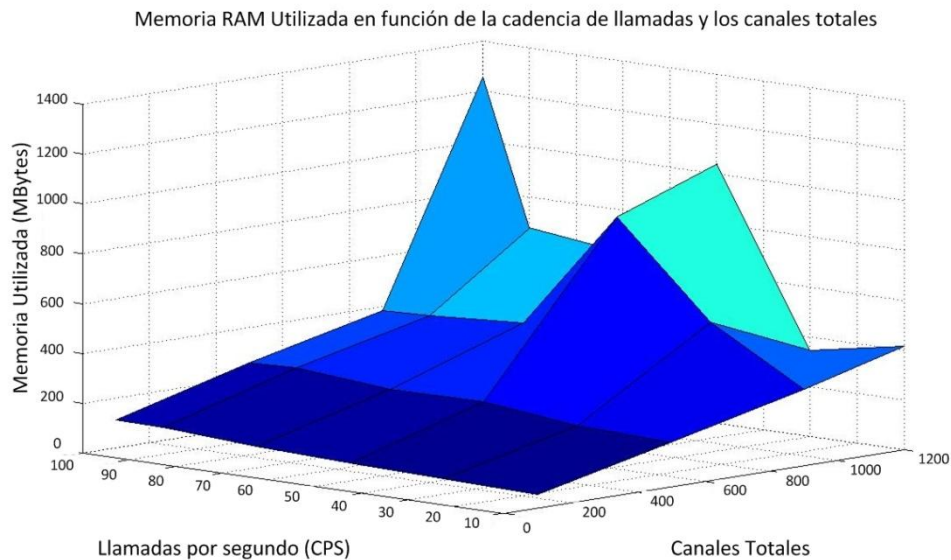


Figura 5. 6 - Memoria utilizada en función de la cadencia de llamadas y los canales totales

El gráfico (figura 5.6) anterior muestra la información de la tabla 5.4 más los puntos de funcionamiento donde ocurrió desborde de *streams*. Se observa un comportamiento plano en el consumo de memoria salvo por las zonas donde ocurre leak de *streams*, en las mismas se dispara a valores aleatorios no representativos. En la zona del plano correspondiente a CPS mayores a 50 y llamadas concurrentes mayores a 900 podemos afirmar que el comportamiento de la memoria no es estable y tampoco lo es el escenario. Como vimos anteriormente esta zona está incluida en la “zona roja” en la cual anteriormente definimos que no es un buen punto de trabajo, el manejo de memoria en esta zona no hace más que afirmar este concepto.

Por último, en un intento de obtener una regla de dimensionamiento de *Hardware* y aprovechando la respuesta lineal de la memoria, se ajusta una recta consumo de memoria en función de los canales totales para el caso de 10 CPS, el cual se acerca a la realidad de nuestras necesidades.

La recta que ajusta dichos puntos es:

$$mem(MBytes) = 0.3292 * (CanalesTotales) + 16.8145$$

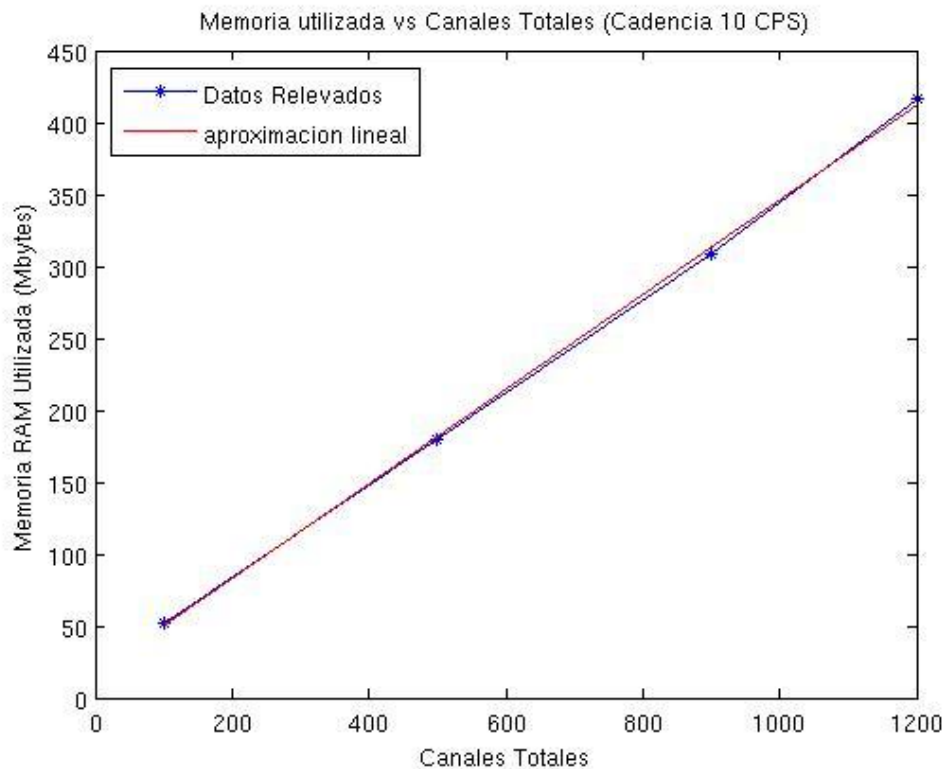


Figura 5. 7 - Memoria utilizada en función de los canales totales

Para los 247 canales dimensionados se infiere de esta regresión lineal que serán utilizados 98 *MBytes* de *RAM*.

Como conclusión de esta sección se afirma que con el *Hardware* que se cuenta se puede soportar la carga propuesta en el dimensionamiento sin mayores dificultades, y sin necesidad de 2 servidores. De todas formas en el capítulo 7 se contempla la escalabilidad de la solución, así como la alta disponibilidad.

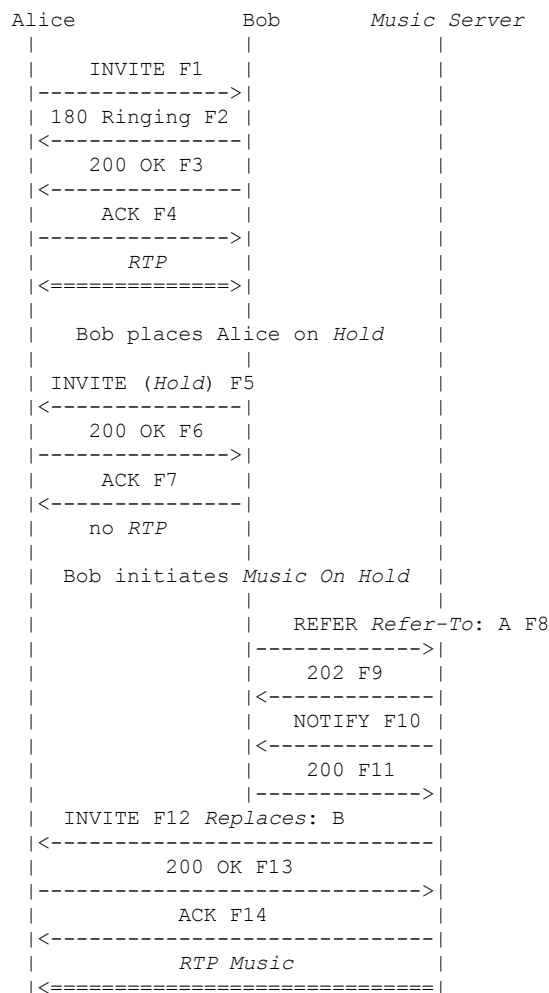
Se obtuvieron zonas de funcionamiento y zonas prohibidas en función de la cadencia de llamadas CPS y la cantidad de llamadas concurrentes.

Para una cadencia de 10 CPS se observó un comportamiento lineal en el uso de memoria y CPU por parte de *FreeSWITCH* en todo el rango de interés.

5.2.6 – Integración al *Core* de Señalización

Como se menciona en el subcapítulo 5.2.3, el *Core* de señalización debe notificar de alguna forma al servidor que implementará la *feature* (*Music On Hold*) que debe intervenir en la llamada, es entonces que en este subcapítulo que se presentará como se realizó esta integración, cuáles fueron los inconvenientes presentados y sus soluciones.

La *RFC-5359* [60], muestra ejemplos de buenas prácticas entre los cuales se encuentra la interacción de diferentes *UA's* con *Media* servers (*PBX*). En concreto en la sección 2.3 página 38 se hace explícito un diagrama de flujo SIP en el que se muestra la recomendación para un escenario de *Music On Hold*:



En grandes rasgos Alice llama a Bob y establecen un flujo de audio *RTP*, posteriormente Bob coloca en espera a Alice mediante un INVITE con *Hold* (mensaje F5), este INVITE es *in-Dialog*, por ende cuenta con un *Tag* en el *Header To*. A su vez para señalar que Alice fue puesta en espera por BOB, en el *SDP* de dicho INVITE debe existir el parámetro *sendonly*, el cual Alice interpretará como que BOB solo enviará audio pero no recibirá (o procesará) lo que Alice le envíe, de esta forma el UA Alice no enviará audio porque fue avisado de que no será escuchado por BOB, que a su vez no enviará audio porque justamente lo que quiere es poner en espera la llamada con Alice.

Luego el UA Bob debe generar un mensaje de REFER destinado al servidor de *MoH*, este mensaje de REFER tiene los datos necesarios para que posteriormente el servidor envíe un INVITE hacia Alice en busca de establecer una sesión *RTP* mediante la que Alice recibirá la música de espera.

En este punto es conveniente hacer varias acotaciones respecto a las exigencias de esta *RFC*, en principio la misma exige la capacidad de los UA de generar mensajes “*out of Dialog*” REFER. Como resultado de la investigación y posteriores pruebas realizadas a *Softphones* de código libre y gratuito, actualmente no existen versiones gratis que generen un REFER “*out of Dialog*” hacia un servidor de música en espera (configurable en el mismo) después de enviar el INVITE con *Hold*.

Con motivo de encontrar una posible solución a este problema se procede a aplicar herramientas brindadas por el SIP *Proxy Kamailio*, que por estar funcionando en modo *stateful* mantiene el estado de todas las sesiones existentes. Se hace posible mediante lógica en el código de ejecución, que el *Proxy* ante un INVITE recibido, antes de reenviarlo hacia su destinatario chequee si es un INVITE con *Hold*, de lo contrario lo reenvía utilizando el campo *URI* (comportamiento por defecto). Si contiene en el *SDP* el parámetro de *Hold* mencionado anteriormente, entonces también reenvía el INVITE hacia su destino pero posteriormente ejecuta un bloque de código especial que tiene como motivo generar el REFER “*out of Dialog*” que el *User Agent* no pudo generar hacia el servidor de música en espera. De esta forma se puede paliar la situación y dar continuidad al flujo SIP propuesto por la *RFC*. También se debe notar que el SIP *Proxy* ante esta situación también está actuando como un *User Agent Client* debido a que genera mensajes además de realizar su clásica función de enrutamiento. Por lo tanto *Kamailio* no es un simple SIP *Proxy*.

En un futuro puede existir la posibilidad de que los *softphones* libres y gratuitos decidan seguir esta *RFC*, acompañando la tendencia de colocar la inteligencia de los sistemas telefónicos en los extremos. Tampoco se descarta la posibilidad de que existan *softphones* propietarios que ya lo implementen, pero su evaluación no se encontró dentro de los objetivos del proyecto.

Volviendo entonces al flujo SIP propuesto por la *RFC*, el REFER “*out of Dialog*” debe ser enviado al servidor de música en espera para posteriormente con los datos de este mensaje generar un INVITE hacia el *UA Holdee*. En este punto se devela otro problema importante de esta solución, la no existencia de un *Music On Hold Server* gratuito en el cual su SIP Stack sepa resolver mensajes de REFER’s *out of Dialog*, las *PBX* más populares (y con más desarrollo) *FreeSWITCH* y *Asterisk* carecen de esto último.

El stack SIP de *FreeSWITCH* *Sofia SIP* [61] responde con mensajes “500 Internal Server Error” ante REFER’s *out of Dialog* y *Asterisk* con mensajes 6xx del mismo significado.

Por lo tanto esta solución intentado seguir la *RFC-5359* además de la complicación de falta de implementación en los *softphones*, se le agregaría la tarea de “emparchar” el problema existente con el servidor. Una posibilidad para esto es colocar otra entidad SIP que oficie de traductor entre el SIP *Proxy* y el *MoH Server*. De esta forma el *Proxy* generaría el REFER hacia esta entidad y la misma se encargaría de generar señalización que una *PBX* actual acepte (por ejemplo un INVITE) para que finalmente el *MoH Server* establezca una sesión de audio con Alice.

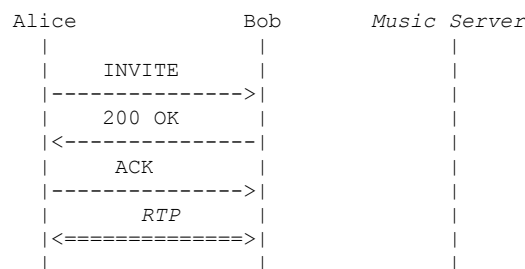
Esta entidad podría ser un *Kamailio* instalado en el mismo servidor donde se encuentre la *PBX* que oficie *MoH Server*, por lo tanto a los ojos del SIP *Proxy* se está comunicando con un *Media Server* que implementa la *RFC* y el problema quedaría encapsulado.

Otra posible solución es que el SIP *Proxy* envíe el REFER generado (o reenviado en caso que haya sido generado por un *UA*) hacia un servidor que tenga una instancia de *SIPp* configurada con un escenario que puede ser programado para que genere exactamente todos los mensajes que le correspondería al *Media Server* según la *RFC-5359*. El audio enviado como música en espera también se implementaría con *SIPp* ya que el mismo maneja *streams* de audio.

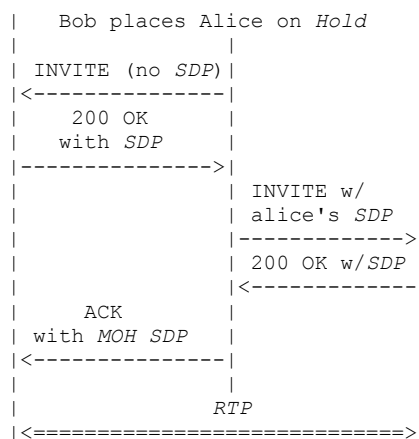
Como conclusión dado el nivel de modificaciones necesarias para implementar este flujo se opta por no seguir esta recomendación de buenas prácticas según la

RFC-5359 del año 2008. Claramente el estado del arte actual no derivó en la adopción de esta *RFC* por parte de implementadores de las *IP PBX* así como también de los desarrolladores de *softphones*, siempre considerando el marco del *software* libre.

Por lo tanto se optó por crear un escenario en el cual los *UA's* se les exija únicamente saber crear y responder mensajes de *INVITE*, lo cual toda entidad que sepa hablar SIP según la *RFC-3261* debe implementar. Las ideas básicas se muestran en el siguiente diagrama de flujo, se muestra solo a efectos de entender las ideas que inspiraron la solución final ya que la misma difiere en la entidad generadora del *INVITE* hacia el *Music Server* y el mensaje disparador del *HOLD*. Se volverá sobre este tema después.



Hasta aquí la señalización corresponde a una llamada común entre Bob y Alice.



Bob envía un *INVITE* sin *SDP* hacia Alice, este *INVITE* debe pertenecer al mismo diálogo ya establecido entre Bob y Alice, por lo tanto es un *INVITE in-Dialog*. Este método es usualmente utilizado para renegociaciones de códecs, puertos *UDP*, *TCP*, etc., en sesiones ya establecidas. El *INVITE* sin *SDP* tiene como objetivo

recibir como respuesta un 200 OK en el cual Alice envía todos sus datos de sesión, en particular interesa el códec utilizado, *IP* y puerto *UDP* donde escucha actualmente el flujo *RTP*, etc. Con estos datos brindados por el *SDP* del 200 OK recibido, Bob genera un INVITE hacia el *Music Server* utilizando el mismo *SDP* que recibió de Alice, por lo tanto a nivel de sesión se hace pasar por el *UA* Alice. Esto genera entonces que el *Music Server* entienda que Alice quiere mantener una conversación con el *MoH Server*, entonces el mismo le enviará el audio correspondiente. Estas ideas son compatibles con la *RFC-3725* [62] la cual define el *3pcc* (*Third Party Call Control*).

Para llevar entonces estas ideas a nuestro sistema se debieron hacer las siguientes consideraciones. En primer lugar los *softphones* utilizados, a saber *Twinkle* y *Blink*, implementan la función *Hold* como lo establece la *RFC-5359*, es decir generando un INVITE con el parámetro *sendonly* en su *SDP*. Por lo tanto el *Proxy* recibe este mensaje y lo utiliza como una señal de que se debe disparar un escenario donde deberá participar activamente generando mensajes.

En concreto, antes de reenviar este INVITE hacia Alice, el mismo le quita el parámetro *sendonly* del *SDP* y posteriormente se lo envía. Esta decisión de diseño tiene la contra importante que el *softphone* de Alice no despliega en su display que se encuentra en espera (como lo haría en un escenario de *Hold* estándar), pero la justificación para hacer esto radica en que hay *softphones* (como por ejemplo *Twinkle*) en los cual si reciben un INVITE con *sendonly* en el *SDP*, dejan de escuchar el audio recibido, por ende no escucharían la música de espera. Esta decisión de borrar el campo *sendonly* brinda entonces mayor nivel de compatibilidad para con diferentes *softphones*, además de funcionar correctamente con los *softphones* que si interpretan correctamente dicho campo.

Entonces, como respuesta a este INVITE el *Proxy* recibirá un 200 OK que contiene el *SDP* deseado, este será utilizado por el mismo para generar un INVITE hacia el *MoH Server*. En este punto es entonces donde se utiliza las ideas de *3pcc* para generar el flujo de audio entre el *Music Server* y Alice.

El *Proxy* al generar un nuevo dialogo, ahora entre el *Music Server* y Alice, debe guardar todos los datos correspondientes a esa transacción, a saber *Call-ID*, *Tag* del *To* y *Tag* del *From*, para luego cuando Bob saque de espera a Alice, traer todos los datos necesarios en ese instante de ejecución para enviar un BYE hacia el *Music Server* y de esta forma cortar el flujo *RTP* existente entre el servidor de música y el *User Agent Holdee*. Cabe destacar que ante casos no usuales del

escenario, por ejemplo si el *UA Holder* ó el *UA Holdee* cualquiera de los dos termina la llamada, el *Proxy* debe encargarse de terminar la sesión de audio de *Music On Hold*, de otra forma ese audio permanecería para siempre en ejecución consumiendo recursos del servidor en forma no productiva. Todos los detalles de implementación se encuentran en el Anexo 1.

El sistema entonces tiene como objetivo brindar la mayor compatibilidad con *softphones* existentes, aún así se observaron problemas importantes con el *softphone Blink*, el mismo al poner en espera a otro *UA* enviando en el *SDP* del INVITE el atributo *sendonly*, no cumple con no enviar *RTP* hacia el *UA Holdee*, por lo tanto el *Holdee* escucha una mezcla de audios compuesta por el correspondiente al *MoH* y el que continua arribando desde el *Blink*, escuchando así música en espera entrecortada. Por lo tanto el uso de *Blink* en el sistema implementado no es recomendado.

Es importante destacar que el *SIP Proxy* debe ser “*Stateful*” debido principalmente a que debe mantener el estado del dialogo propio creado entre él y el *MoH Server*. El simple hecho de que el *Proxy* cree un mensaje SIP lo convierte en un *UAC (User Agent Client)*, por lo tanto debe mantener el estado de sus transacciones. Esto se verifica en forma práctica observando que el módulo necesario para que *Kamailio* pueda crear mensajes propios (módulo *UAC*) tiene como dependencia la activación del módulo de transacciones (módulo *TM*).

A continuación se muestra en detalle un diagrama de flujo del caso de uso de éxito del escenario *Music On Hold*.

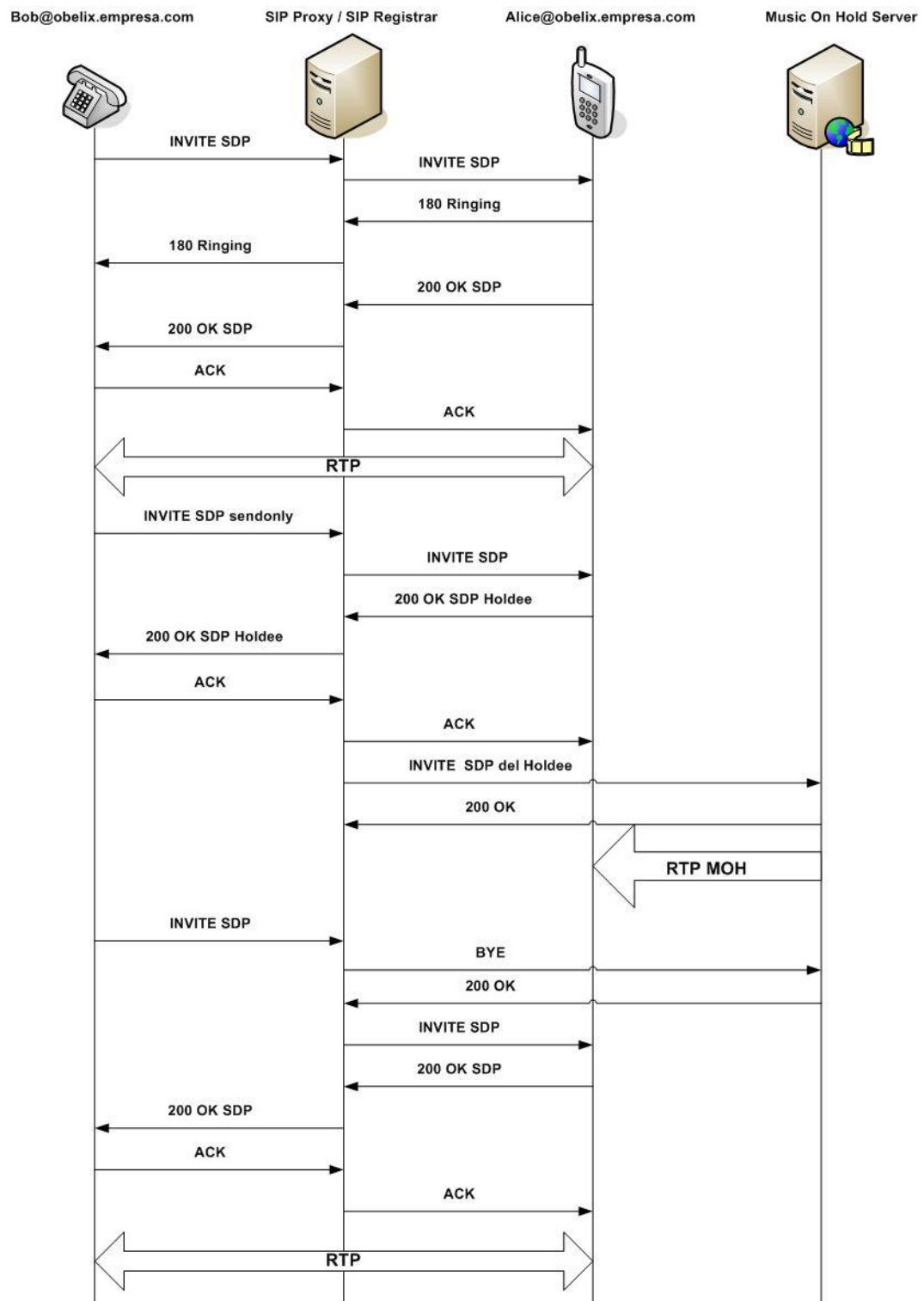


Figura 5. 8 - Diagrama de flujo del caso de uso de éxito del escenario Music On Hold

Para concluir se hace referencia a una página WEB [63] en la que se encuentra una crítica a la implementación recomendada para *Music On Hold* por la RFC-5359. En esta misma también se propone utilizar una solución similar a la implementada en este trabajo. A modo de resumen de dicha cita, se extrae de la misma que la implementación propuesta por la RFC para *Music On Hold* se ajusta más fielmente a un escenario de “*Call Park*” que a un escenario de *Hold*. Los fabricantes de *Hardphones* e implementadores de *Softphones* entienden esto mismo y es por esto que no existe un *User Agent* que implemente esta RFC.

“...Let's analyze this briefly. The RFC is asking phone manufacturers To kick off the Hold process normally. However this is where "normal" Hold ends. The Holder then is To send an out of Dialog REFER with Replaces To the MOH Server. The MOH Server is To take that REFER with Replaces and INVITE itself To the Holdee and replace the Holder's Call on the Holdee. By conformance To RFC 3515, the MOH Server will then send a NOTIFY To the Holder via the implicit subscription of the REFER. Additionally the MOH Server must add the identifying parts of the new Call with Alice To the NOTIFY body, the Call ID, From Tag, and To Tag of the new Call in the body. By standard INVITE with Replaces conformance, the Holdee will then send a BYE To the Holder and end the original Dialog there. When the Holder wants To retrieve the Call, he is supposed To send the Holdee a second INVITE with Replaces, filled in with the parameters of the second NOTIFY From the MOH Server.

What is the Holder's phone supposed To look like while there is no Call on it? But he effectively still has Alice on Hold? Why can't we leverage the old Hold mechanism of redirected SDP by putting the Holdee Media session in sendonly? What if Alice hangs up the phone - how is Bob supposed To know that? What if Alice wants To put Bob on Hold while she is on Hold? She can't!!! She doesn't have a Call with Bob anymore. What if Alice's phone doesn't support INVITE with Replaces? Why does Alice's phone need To support INVITE with Replaces in order for her To receive Hold Music? Is Bob's phone supposed To analyze the Supported headers (if any) From earlier in the Dialog or the result of the NOTIFY for switching between "Music On Hold mode" and "plain old Hold mode"? That's just error prone. What does Alice's Call display say - is it the caller ID of the MOH Server? Further "yucks" - the Holder phone must now support receiving a NOTIFY with a message/sipfrag body and store values From the sipfrag for later use... why? Also, the MOH Server must support the REFER method, an out of Dialog REFER with Replaces To be specific. It must also support sending INVITE with Replaces in order To complete the Attended Transfer. </crazy-rant> It's my belief

the RFC authors wrote a severely crippled Hold specification here. The recommendation (without seriously more guidance) breaks nearly every requirement of Hold that I laid out above. This spec is much more like a Call Parking service than Hold. If that's what it were called - then I'd have no problem with it. I think the phone manufacturers believe this too... because I have yet To find a phone that supports this flow. If you know of one - let me know in the comments please. SO then what do the phone manufacturers implement? Well, Polycom and Cisco (though Cisco has a few bugs with their implementation) do something much more simple - and require no extra support From the Holdee's phone. If it supports standard Hold - it can support Music On Hold..."

5.3 – Voicemail

5.3.1 – Descripción del Servicio

El agregado de esta *feature* al sistema permite a cada usuario contar con una casilla de correo en la que otros usuarios podrán dejar mensajes que serán posteriormente leídos por el destinatario.

El sistema cuenta con una implementación que permite al usuario recibir notificaciones de sus mensajes en espera directamente al *softphone* (o cualquier *User Agent*), opcionalmente notificaciones a su casilla de correo empresarial o personal.

5.3.2 – Casos de Uso

El servicio de *Voicemail* cubrirá los siguientes escenarios:

- Un usuario del sistema desea comunicarse con otro que momentáneamente no se encuentra registrado. El sistema atenderá esta llamada por el usuario que no se encuentra presente y derivará la misma a la casilla de *Voicemail* del usuario ausente. Ambos usuarios deben estar creados en el sistema, bajo el mismo dominio Obelix.
- Ambos usuarios se encuentran registrados en el sistema, uno de ellos desea comunicarse con el otro pero este último se declara ocupado "BUSY" al momento de responder la llamada. Se trata entonces de un BUSY explícito ingresado por el destinatario de la llamada. El sistema al igual que en el caso anterior derivará la misma al módulo encargado de brindar el servicio de *Voicemail*.

- El caso anterior también se repite si el usuario destinatario de la llamada verdaderamente se encontraba ocupado en otra conversación. Aquí depende del *User Agent* destinatario, este puede tener la capacidad o no contestar de contestar la llamada implementando otra “línea”, de esta forma puede momentáneamente colocar en espera a la llamada actual para resolver la nueva. De lo contrario el *softphone* generará un mensaje BUSY, para el sistema será idéntico al caso de uso anterior por lo tanto tendrá el mismo tratamiento.

En los tres casos de uso descriptos anteriormente, los usuarios que son destinatarios de la llamada deben tener su casilla de *Voicemail* creada.

Un usuario al ser recibido por una casilla de *Voicemail* será atendido por una *IVR (Interactive Voice Response)* la cual le indicará que la persona con quien desea comunicarse no se encuentra disponible, posteriormente le solicitará dejar un mensaje. Una vez finalizado el mensaje, se pedirá la confirmación de este para ser enviado y finalizar la transacción.

Por último el sistema deberá notificar al usuario destinatario que efectivamente tiene un mensaje mediante dos mecanismos de contacto. El primero será enteramente resuelto a nivel de señalización SIP implementando la funcionalidad clásica de *Message Waiting Indicator (MWI)*. Esta última consiste en desplegar un aviso en el *softphone* del usuario con el fin de indicarle la cantidad de mensajes en espera con la que cuenta, se detallará más en profundidad en los subcapítulos siguientes.

El segundo mecanismo consta en recibir por parte del usuario notificaciones mediante correo electrónico con la información de mensajes en espera, incluso recibir el mensaje propiamente dicho si es que el administrador del sistema considera apropiado. La casilla a la cual se enviará esta información es configurable y es parte de la definición del usuario del sistema.

Los usuarios accederán a su casilla de *Voicemail* mediante un código común igualitario para todos, el sistema se encargará de desplegar todos sus mensajes en forma ordenada y guiada a través del sistema *IVR*. Con el mismo el usuario tendrá control administrativo sobre los mensajes pudiendo almacenar, eliminar o simplemente navegar a través de ellos.

5.3.3 – *FreeSWITCH* Como Proveedor de Casillas de *Voicemail*

Al igual que en el subcapítulo 5.2.3 se utilizará la capacidad de *FreeSWITCH* para brindar la funcionalidad de *Voicemail*. En este caso también se lo utilizará como un ente aislado, a saber, no perteneciente al núcleo de señalización básico. Este núcleo en base a la detección de los eventos declarados en los casos de uso, redireccionará los mensajes de INVITE hacia dicho servidor de *Voicemail*.

Se configura entonces en el *Dialplan* de *FreeSWITCH* que atiende el perfil correspondiente al puerto *UDP* 5060 (puerto SIP estándar), que cuando reciba un INVITE hacia un interno (o extensión SIP) luego de atender la llamada (response 200 OK), se ejecute la aplicación que implementa la casilla de *Voicemail* del usuario destinatario del INVITE.

Por las mismas razones que en el subcapítulo 5.2.3, el servidor de *Voicemail* no autenticará usuarios. Tampoco atenderá paquetes SIP con dirección *IP* distinta a la de *SIP Proxy*.

Un detalle importante desde el punto de vista del aprovisionamiento del servicio, es que todos los usuarios del dominio @obelix que dispongan de casilla de *Voicemail*, deberán tener su usuario también configurado en el servidor de *Voicemail* (*FreeSWITCH*). Otras opciones son realizar *scripts* de aprovisionamiento que realicen esta misma configuración de forma automática o realizar una integración de las bases de datos del *SIP Proxy* y *FreeSWITCH*, de forma que ambos sistemas utilicen una única base. No se trataron estas soluciones por considerarse fuera del alcance del proyecto.

El *Proxy* en base a la detección de *SIP Responses* (486 BUSY, 603 DECLINE) de los *User Agents* que están registrados, redireccionará el mensaje de INVITE originador de cualquiera de dichas *Responses* hacia el dominio @*Voicemail.obelix*, manteniendo la parte de usuario de la *URI* intacta. Por lo tanto *FreeSWITCH* debe ser configurado para atender dicho dominio.

El sistema debe redireccionar el INVITE de la misma forma descripta anteriormente también para el caso en que el *User Agent* destinatario de una llamada no se encuentre registrado en el *SIP Proxy/SIP Registrar*, pero si tenga usuario creado en el mismo. Esto se traduce en que el usuario exista como “subscriber” pero no se encuentre en la tabla de “location” del *SIP Proxy Kamailio*.

Todos los usuarios disponen de un interno fijo igual para todos (interno 4000@obelix), mediante el cual pueden chequear su casilla de *Voicemail*. Este

interno debe estar definido en el *Dialplan* de *FreeSWITCH* de forma que cuando un usuario llama al mismo, lo debe atender un sistema *IVR* que le solicitará ingresar un usuario y contraseña. Pasada esta validación el usuario podrá escuchar sus mensajes. El *Proxy* debe encaminar este mensaje de *INVITE* hacia el dominio *Voicemail.obelix* manteniendo entonces el número de interno 4000.

Un detalle importante es que esta “autenticación” mediante usuario y contraseña no realiza ninguna validación de origen. Un usuario que por algún método pueda extraer la contraseña de otro podrá acceder a la casilla de la víctima llamando al interno 4000 digitando usuario y contraseñas ajenos. Una solución para agregar robustez a este flagelo de seguridad puede ser la siguiente:

Implementar en el *Dialplan* de *FreeSWITCH* que la extensión 4000 permita acceder a un usuario a su casilla de *Voicemail* si y solo si, los *DTMF* capturados cuando se le solicita ingresar el número de usuario sean en su conjunto igual al campo de usuario del *From Header*, contenido en el *INVITE* que ingresa a la *PBX*.

Los detalles de la implementación de la lógica de procesamiento por parte del *Proxy* se encuentran en el Anexo 1. La configuración de *FreeSWITCH* para actuar como servidor de *Voicemail* se encuentra en el Anexo 2. En éste último se encuentra además la configuración necesaria para habilitar la notificación por correo electrónico de mensajes en espera.

En el subcapítulo 5.3.6 se encuentran los diagramas de flujos correspondientes a todos los escenarios aquí planteados, además de incluir los mensajes necesarios para implementar la notificación de mensajes en espera (*MWI*).

Por último es importante destacar que para la implementación de esta funcionalidad no fue necesario que el *SIP Proxy* genere transacciones propias (creación de diálogos propios con otra entidad *SIP*), como sucedió para el caso del *Music On Hold*. El *Proxy* toma su rol por definición enrutando mensajes *SIP* en función de su campo *URI* (el cual puede modificar a su antojo mediante lógica programa en su archivo de configuración) o en base su tabla de *location* según corresponda. Por lo tanto es una implementación en términos comparativos más sencilla y estándar que la solución expuesta para la implementación *Music On Hold*.

Sin embargo el *SIP Proxy* también debe ser “*Stateful*” en este caso. Se debe a que en el mismo se define un bloque de ruta especial para el manejo de fallos (*t_on_reply(MANAGE_FAILURE)*), en el que se da tratamiento a *Responses* “*Busy*”, “*Request Timeout*” y “*Decline*”. Este consiste en re-direccionar el *INVITE*

generador de esas *Responses* hacia el servidor de *Voicemail*, antes modificando su *URI* como se explicó anteriormente.

5.3.4 – Dimensionamiento

En esta sección se pretende estimar cual es el tráfico ofrecido, para dimensionar mediante *Erlang B* la cantidad de puertos de *Voicemail* necesarios.

Tomaremos como criterio de diseño que en un ambiente corporativo un usuario promedio utiliza por 5 minutos diarios el servicio de *Voicemail*, esto incluye dejar mensajes en casillas de otros usuarios así como también chequear sus propios mensajes.

Por lo tanto, 10000 usuarios generan 50000 minutos diarios de uso de *Voicemail*. De acuerdo a [64], el 17% de este tráfico se da en la “*Busy Hour*”, por lo tanto los minutos cursados en dicha hora pico son 8500 minutos. Llevando a unidades de tráfico se corresponden con 142 Erlangs.

Aplicando entonces *Erlang B* con probabilidad de bloqueo 1% resulta en un total de 161 “líneas” o puertos de *Voicemail*.

Se arriba entonces que la solución a implementar debe tener la capacidad de mantener simultáneamente 161 sesiones concurrentes del servicio de *Voicemail*.

También debe tenerse en cuenta en este dimensionamiento la capacidad mínima necesaria para el sistema de almacenamiento de mensajes. Dimensionando para el peor caso en que los 5 minutos diarios de utilización por usuario sean para generar nuevos mensajes, diariamente se deben almacenar 50000 minutos de audio.

Utilizando el códec *G.711* cuyo bitrate es 64kbps se generarían 24 GBytes de datos por día.

5.3.5 – Pruebas de Carga

Este subcapítulo al igual que en el 5.2.5 (para la *feature MoH*) se establecerá en base a pruebas de carga efectuadas al mismo *Hardware*, la cantidad máxima de usuarios que podrán hacer uso en forma simultánea la funcionalidad de *Voicemail*.

Tomando como entrada los datos calculados en la sección anterior, el sistema debería ser capaz de soportar 161 sesiones simultáneas de *Voicemail*. El espíritu es entonces conocer el máximo rendimiento de la *IP PBX FreeSWITCH*, haciendo uso de su módulo *Voicemail*.

Este escenario de prueba requiere que el generador de señalización además sea el encargado de enviar audio *RTP* en el preciso instante que se abren los canales de *FreeSWITCH*, por lo tanto *SIPp* aparte de estar encargado de generar señalización como en pruebas anteriores, en esta instancia también debe procesar audio.

Al tratarse del mismo *Hardware* utilizado para las pruebas de la *feature MoH*, utilizaremos el conocimiento generado sobre el comportamiento en carga de nuestro *Hardware* de pruebas, en concreto hacer uso de las “zonas de buen funcionamiento” analizadas.

La funcionalidad de *Voicemail* difiere esencialmente de la *Music On Hold* en el tipo de procesamiento que se le debe dar al audio. En esta última el audio que arriba a la *PBX* debe ser guardado a disco, mientras que en la anterior se debe reproducir un stream de audio prefijado. Además en el caso del *Voicemail* se suma la complejidad agregada por la *IVR* la cual interactúa con el usuario antes de efectivamente permitirle dejar su mensaje. El proceso generado por la *IVR* consiste en procesar tonos *DTMF* que son solicitados al usuario para que el mismo pueda navegar por un menú que le permitirá elegir diferentes opciones. Estos tonos *DTMF* viajarán a través de la sesión de audio *RTP* bidireccional establecida entre el *User Agent* y la *PBX*, por lo tanto es evidente que esta sesión *RTP* no solo es establecida para que el usuario deposite un mensaje, sino que además para que pueda escuchar las opciones que esta *IVR* le brinda con el objetivo de poder efectivamente dejar un mensaje o reproducir sus mensajes en espera.

En consecuencia, por las razones anteriores se espera un rendimiento sensiblemente menor por parte de *FreeSWITCH* en comparación al logrado para la *feature Music On Hold*.

Las pruebas realizadas se efectuaron a una cadencia de 8 llamadas por segundo, técnicamente la convierten en una prueba de carga de acumulación de llamadas, en lugar de aumento de cadencia.

Las razones por la que es conveniente hacer la misma de esta forma proviene de la experiencia recabada en las pruebas anteriores y las características de la

arquitectura final del sistema que se quiere lograr. Para una cantidad de tráfico en Erlangs dada, desde el punto de vista práctico para este servicio es de mayor utilidad que el sistema sea capaz de acumular más llamadas concurrentes a menor cadencia de ingreso (CPS), que pocas concurrentes a mayor cadencia. En realidad esto último dependerá fuertemente del tiempo medio de las llamadas que este sistema pretenda cursar.

Sistemas donde el tiempo medio de llamada es muy pequeño pero la cadencia de ingreso de llamadas es alta, ó sistemas donde el tiempo medio de llamada es alto y la cadencia es baja, pueden cargar de forma similar a un mismo servidor. Este fenómeno ya se pudo observar en la tabla 5.3 correspondiente a la prueba de carga de la funcionalidad *MoH*. En ella se aprecia que para el CPU es casi indistinto mantener 100 llamadas viviendo en el sistema, ingresando (y egresando) a una cadencia de 50 por segundo, que mantener 500 llamadas ingresando a 10 CPS. Para el primer caso el tiempo medio de llamadas son 2 segundos y para el segundo 50 segundos.

Por lo tanto con estas consideraciones en mente (y otras más prácticas que se presentan a continuación) es de interés definir una cadencia de 8 CPS y un largo medio del mensaje de aproximadamente 30 segundos para este escenario de pruebas. El largo medio de cada llamada será exactamente 60 segundos, debido a que se debe sumar el largo del mensaje más el tiempo en que el usuario está interactuando con el *IVR*. En condiciones de régimen se pretende entonces escalar hasta 480 canales simultáneos de *Voicemail*.

La elección de estos valores estuvo fuertemente condicionada a la cantidad de puertos *UDP* que *SIPp* permite usar para el envío de *RTP* por instancia de ejecución, en realidad la limitante no la impone *SIPp* sino el sistema operativo host donde se encuentre. Esta cantidad de sockets es controlada por el *SO* mediante parámetros configurables. El comando de Linux “ulimit –s unlimited” permite configurar una cantidad de sockets ilimitada para cada proceso que los necesite.

De todas formas se optó por utilizar la cantidad máxima de flujos *RTP* que *SIPp* puede generar (o mejor dicho el sistema operativo el permite) por cada instancia en forma estándar. Este número es 240, si se intenta establecer más de 240 flujos *RTP* (sockets *UDP*) *SIPp* despliega:

“*SIPp* can’t create thread *To* send *RTP* packets”

Utilizando como referencia la cantidad de canales efectivos que puede manejar la *feature Music On Hold* a cadencia 10 CPS 885 (según la tabla 5.2), se decide comenzar a probar por la mitad de estos canales aproximadamente. Como se cuenta con escenarios de *SIPp* realizados con un audio *RTP* saliente de 60 segundos, ejecutando dos escenarios a una cadencia de 4 CPS cada uno, se logran los 480 canales que se pretenden. Estos escenarios fueron ejecutados desde 2 PC's diferentes que se presentan a continuación.

La prueba consistió entonces en intentar mantener la cantidad de canales anteriormente mencionada y extraer datos del servidor (tiempo de CPU, uso de memoria y ancho de banda) durante el proceso en busca de la saturación del procesador.

Hardware:

PBX FreeSWITCH

- CPU: Intel *Core 2 Duo* E8400 3.0 GHz
- Memoria *RAM*: 2 GB DDR2 800 MHz
- *Gigabit LAN*
- *SO*: CentOS 6.1
- *FreeSWITCH* Version 1.3.0+git~20120829T032350Z~5a3a1e3b44 (1.3.0; git at commit 5a3a1e3b44 on Wed, 29 Aug 2012 03:23:50 Z)

Generador *SIPp* 1:

- CPU: Intel Xeon 3400
- Memoria *RAM*: 2 GB DDR3
- *Gigabit LAN*
- *SO*: CentOS 6.1
- *SIPp* v3.2-TLS-PCAP (módulos de TLS y PCAP compilados)

Generador *SIPp* 2:

- CPU: Intel *Core 2 Duo* E8400 3.0 GHz
- Memoria *RAM*: 2 GB DDR2 800 MHz
- *Gigabit LAN*
- *SO*: Ubuntu 10.04 LTS
- *SIPp* v3.2-TLS-PCAP (módulos de TLS y PCAP compilados)

Escenario generado:

```

----- Scenario Screen ----- [1-9]: Change Screen --
Call-rate(length) Port Total-time Total-calls Remote-host
4.0(0 ms)/1.000s 5060 41.06 s 164 192.168.2.11:5060(UDP)

4 new calls during 1.001 s period 1 ms scheduler resolution
164 calls (limit 240) Peak was 164 calls, after 41 s
0 Running, 166 Paused, 8 Woken up
0 dead call msg (discarded) 0 out-of-call msg (discarded)
3 open sockets
167760 Total RTP pkts sent 1389.231 last period RTP rate (kB/s)

----- Messages Retrans Timeout Unexpected-Msg -----
INVITE -----> 164 0 0 0
100 <----- 164 0 0 0
200 <----- E-RTD1 164 0 0 0
ACK -----> 164 0 0 0
[ NOP ]
BYE <----- 0 0 0 0
200 -----> 0 0 0 0

----- [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----

```

Figura 5. 9 - Captura de pantalla del escenario SIPp generado

El modulo PCAP de *SIPp* permite enviar audio *RTP* capturado con *Wireshark* para simular el envío de audio de una llamada generada con *SIPp*, entre otras cosas.

Por lo tanto para construir esta prueba se capturó el audio de salida de una llamada de pruebas con *Wireshark*. El *RTP* que egresa de la *PBX* no es de interés para el escenario de *SIPp*, esto se debe a que el audio que ingresará a la *PBX* tendrá todos los tonos *DTMF* necesarios para ingresar a la casilla de *Voicemail*, además del mensaje.

La imagen anterior (figura 5.9) muestra una instancia de *SIPp* ejecutando el escenario generado para la prueba. Se puede apreciar que este se corresponde con una llamada estándar entre el generador de tráfico y la *PBX FreeSWITCH* a testear. En la parte superior de la figura se destaca la cadencia de generación de mensajes INVITE a 4 por segundo. También se puede observar la cantidad de paquetes *RTP* enviados y el ancho de banda consumido.

El escenario creado para esta prueba (código) se encuentra en el Anexo 3.

Los resultados fueron los siguientes:

Llamadas Concurrentes	CPU <i>FreeSWITCH</i> (%)	Memoria RAM <i>FreeSWITCH</i> (MBytes)
88	23.5	179
180	42.2	257
244	52.4	308
352	72.3	404
376	79	428
412	85.4	462
480	96.8	521

Tabla 5. 5 Resultados Prueba de Carga Voicemail.

De la tabla anterior se observa que para 480 sesiones simultáneas de *Voicemail* el servidor utiliza enteramente el CPU y 521 MBytes de RAM. Por lo tanto no fue necesario tener más generadores de tráfico (instancias de ejecución) para saturar la *PBX*.

Se concluye entonces de esta prueba de carga que con el *Hardware* descrito anteriormente, *FreeSWITCH* es capaz de mantener 480 sesiones (o canales) de *Voicemail* en forma simultánea ingresando a 8 CPS, o lo que es lo mismo con un tiempo de permanencia en el sistema de 60 segundos por llamada. Con un servidor de estas características se cubre en forma holgada el dimensionamiento realizado para esta *feature* en el subcapítulo anterior (161 canales).

A continuación se presentan regresiones lineales de los datos obtenidos, con el objetivo de contar con curvas de carga representativas de la *PBX FreeSWITCH* en este servidor.

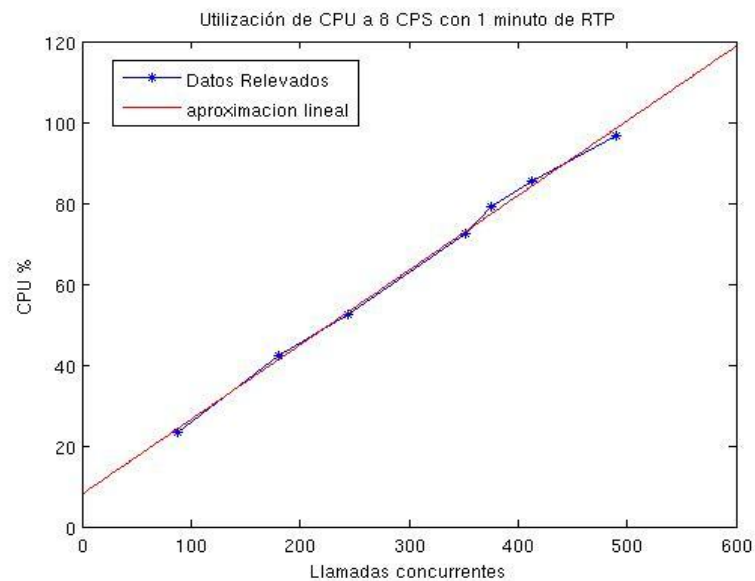


Figura 5. 10 - Porcentaje de uso del CPU en función de las llamadas concurrentes

La figura anterior muestra que el comportamiento del consumo de CPU en función de la cantidad de canales de *Voicemail* concurrentes es aproximadamente lineal. Dicha recta es válida para 8 CPS (o llamadas de largo medio 60 segundos) la misma es:

$$CPU(\%) = 0.1846 * (Canales Voicemail) + 8.0318$$

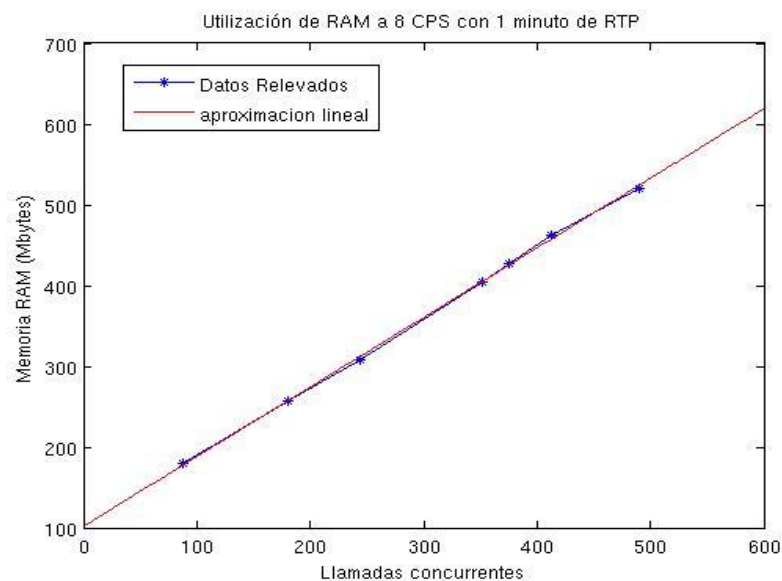


Figura 5. 11 - Cantidad de memoria RAM utilizada en función de las llamadas concurrentes

La figura anterior muestra el mismo comportamiento para la memoria en función de los canales de *Voicemail*. La recta es:

$$\text{Memoria (MBytes)} = 0.8620 * (\text{Canales de Voicemail}) + 101.8031$$

Por lo tanto con estas rectas se puede inferir que para 161 canales de *Voicemail* concurrentes tendremos una carga de CPU del 37.7% y un uso de memoria RAM de 240 MBytes.

5.3.6 – Diagrama de Flujo de Casos de Uso

A continuación se presentan los diagramas de flujos correspondientes a los casos de uso de la funcionalidad *Voicemail*:

- Figura 5.12: Bob intenta comunicarse con Alice pero esta no se encuentra registrada en el sistema. Bob es atendido por el *Voicemail* de Alice.
- Figura 5.13: Bob intenta comunicarse con Alice, esta última utiliza la opción BUSY de su *softphone*. Bob es derivado al *Voicemail* de Alice donde deja su mensaje. Posteriormente Alice es notificada que tiene un mensaje en espera. Este caso de uso aplica exactamente igual si Alice está en conversación con otro *User Agent* al momento que Bob intenta comunicarse.
- Figura 5.14: Bob se registra en el sistema y se subscribe para recibir notificaciones de sus mensajes en espera. Este pedido de notificación llega al servidor de *Voicemail*, es aceptado y por último Bob es notificado del estado de su casilla.
- Figura 5.15: Bob chequea su casilla de *Voicemail* llamando al interno 4000.

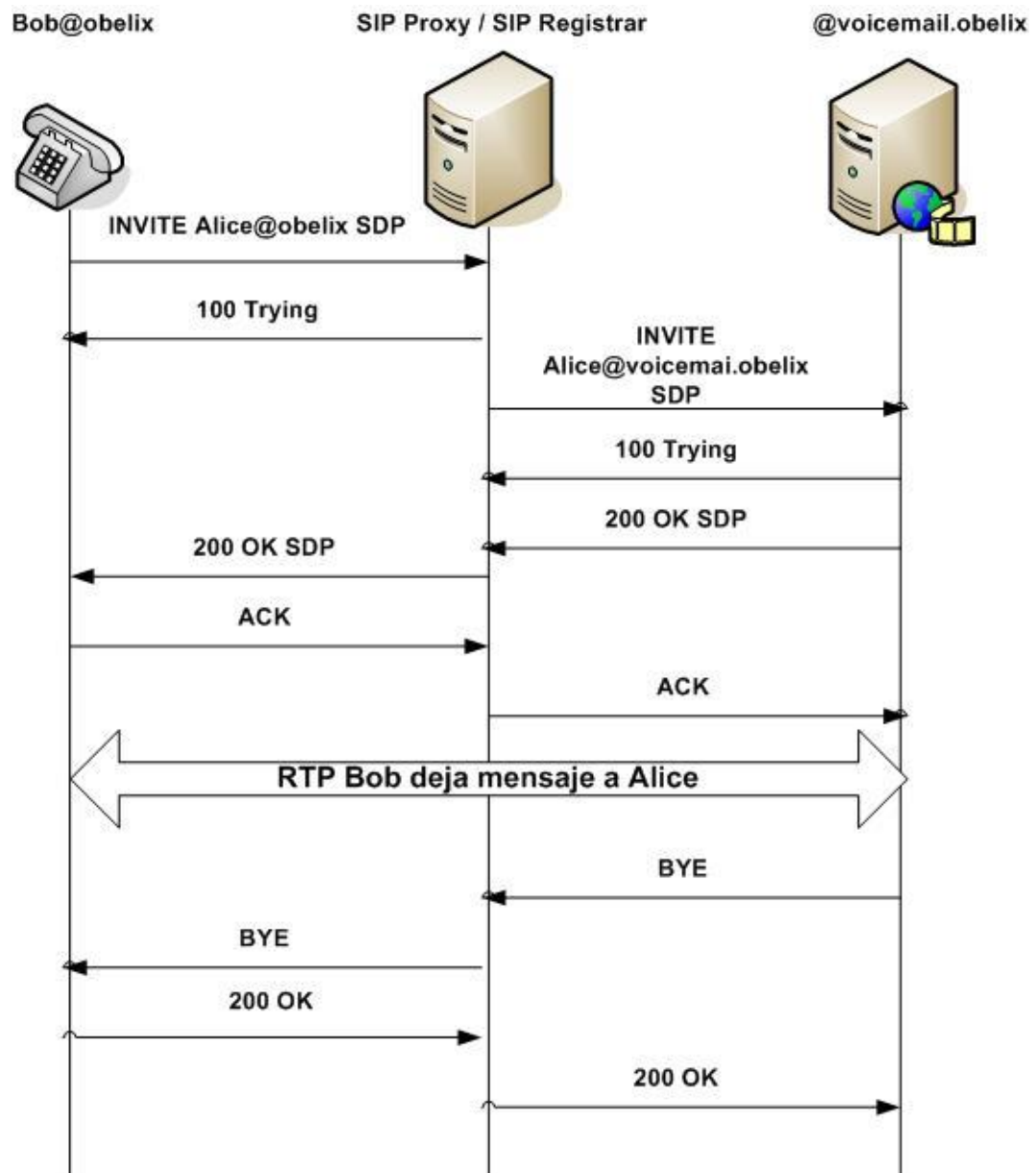


Figura 5. 12 Bob llama a Alice pero esta no es está Registrada

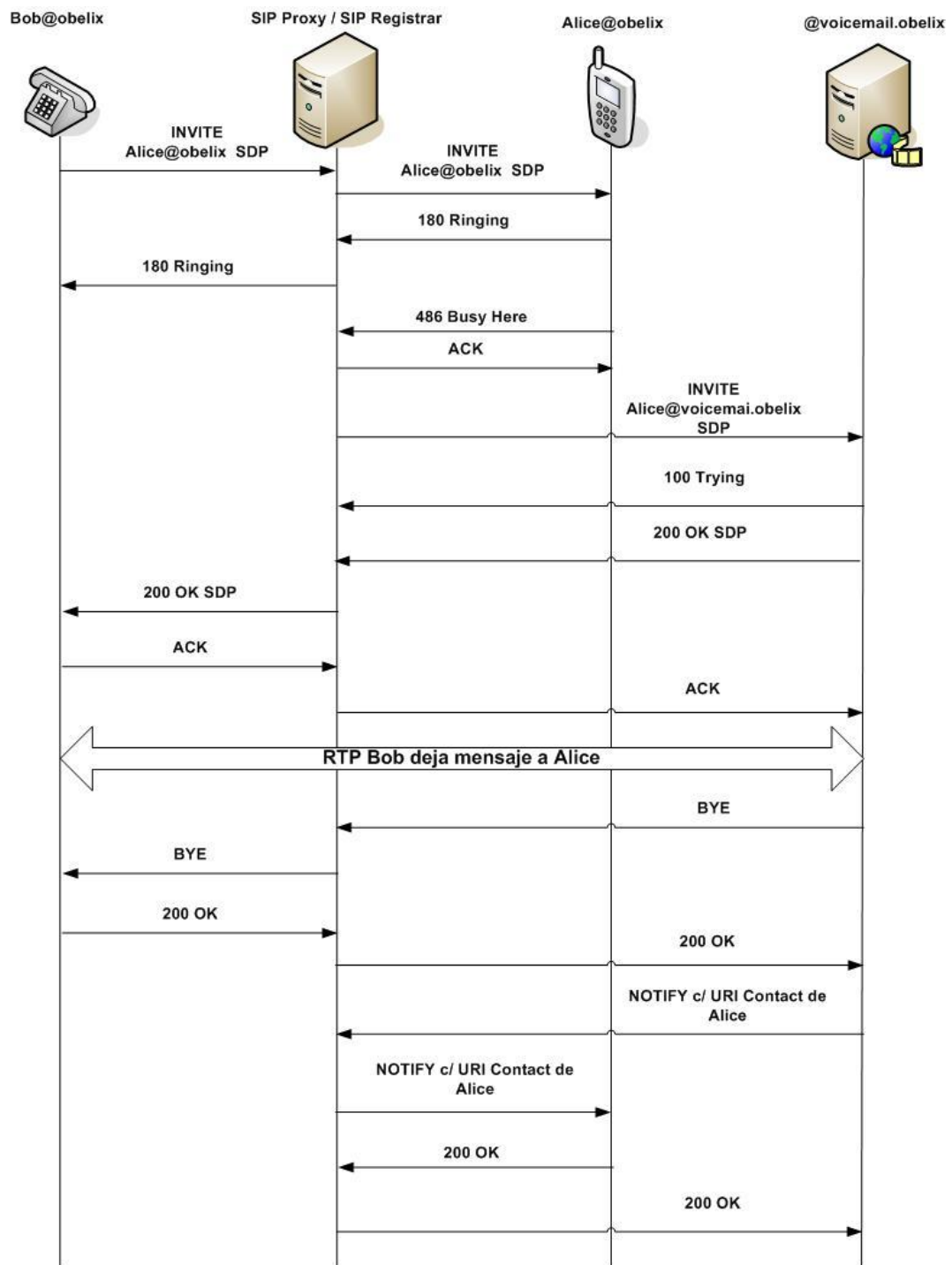


Figura 5. 13 Bob llama a Alice, este recibe BUSY de parte de Alice

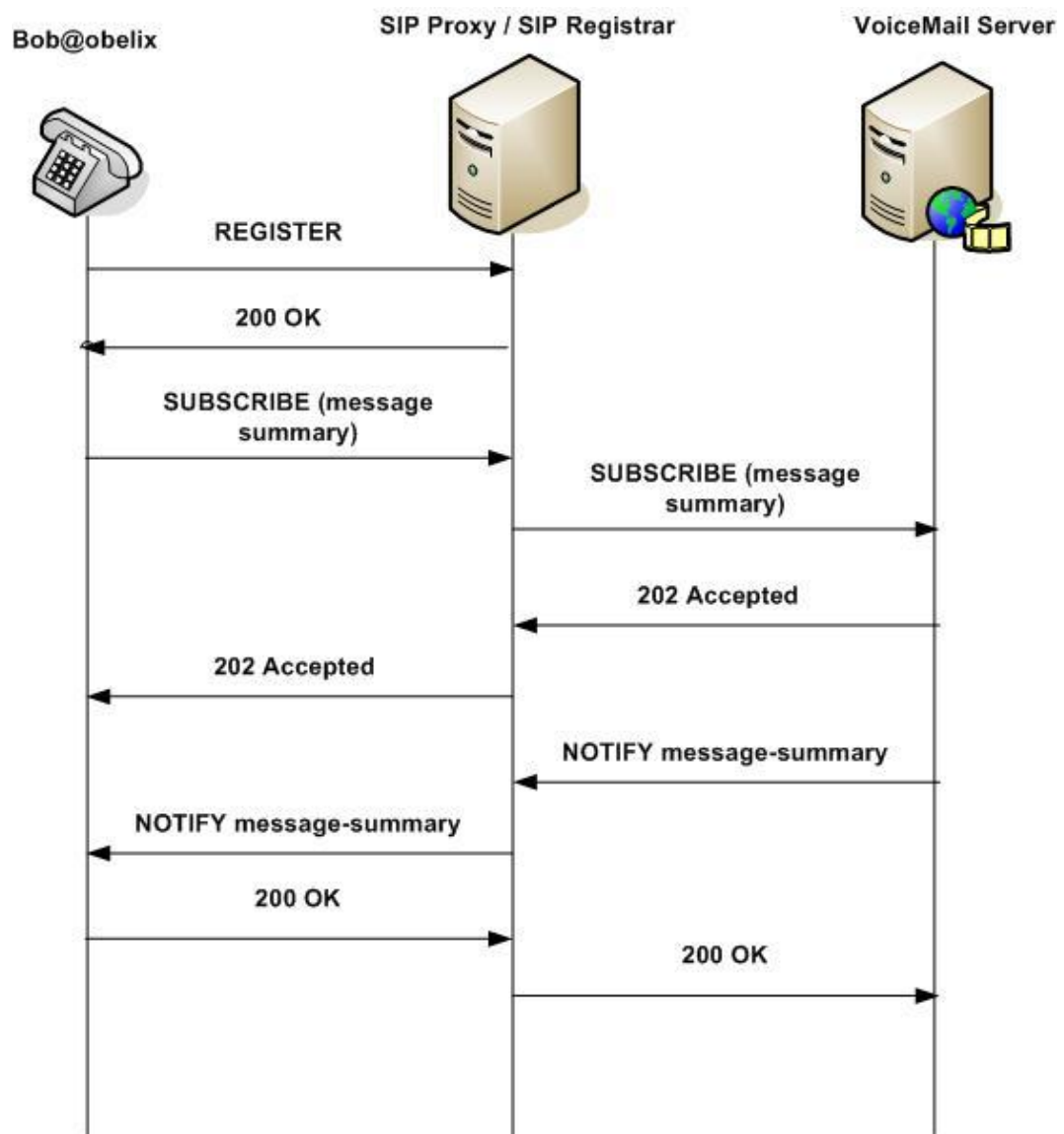


Figura 5. 14 Bob se subscribe a MWI

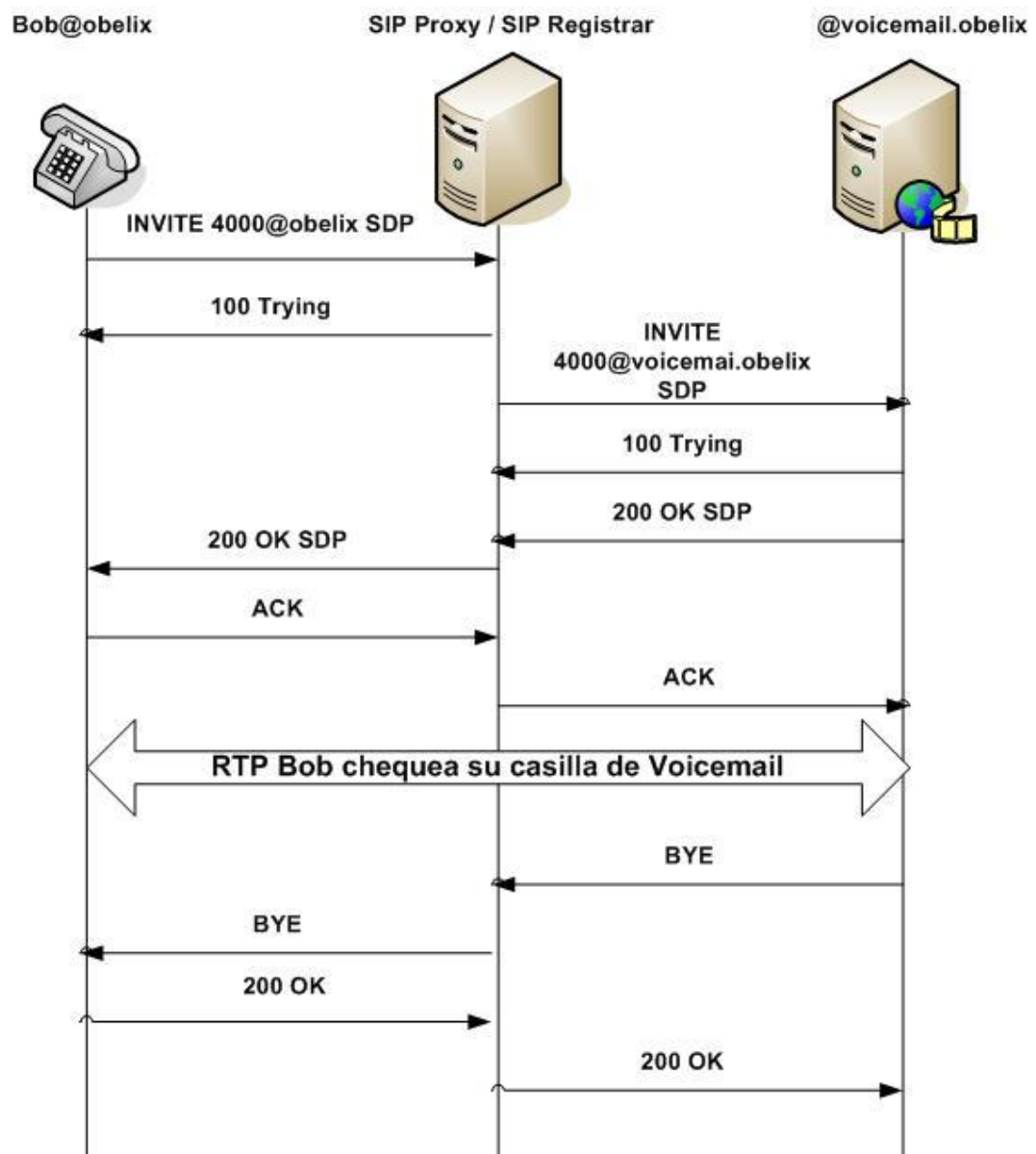


Figura 5. 15 Bob chequea su casilla de Voicemail

5.4 – Puente de Conferencias

5.4.1 – Modelos de Conferencias

Todo sistema de comunicación corporativo debe contar con alguna implementación que permita realizar a sus usuarios conferencias. Existen diferentes modelos para implementar las mismas, en [65] se presentan las mismas y se realiza un análisis comparativo de todas ellas. Los modelos más relevantes son:

- *Full Mesh*
- *User Agent Mixer*
- Servidor centralizado

En el modelo de *Full Mesh* cada *User Agent* debe establecer una sesión de audio con cada uno de los restantes participantes, presentando claramente un problema de escalabilidad por varios factores. Para conferencias de gran porte se deriva en una muy mala utilización de la red de *networking*, por otro lado cada *User Agent* debe realizar el procesamiento de todos los flujos de audio que ingresen a él, pudiendo implicar una alta carga de procesamiento. Por estas razones este modelo no aplica como posible implementación de la funcionalidad de conferencias. En el subcapítulo 5.4.4 se efectuará el dimensionado para el sistema total de conferencias, de este se desprenderá la no viabilidad del modelo *Full Mesh* en base a la gran cantidad de usuarios a dar soporte.

El modelo *User Agent Mixer* es el primer paso hacia el manejo en forma centralizada del audio. Fundamentalmente consiste en que cada participante envía su audio a un *User Agent* que cumple la función de *Mixer*, este a cambio le envía la mezcla de todos los audios pertenecientes a los demás participantes menos el que acaba de recibir por parte del participante inicial. La siguiente figura ilustra este último concepto. El *UA Mixer* se corresponde con el nodo C.

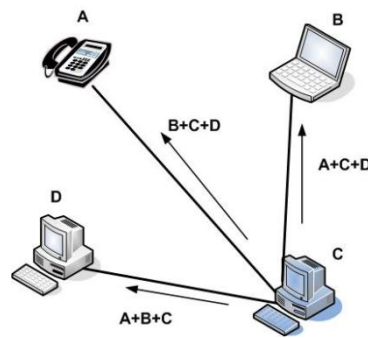


Figura 5. 16 - Descripción gráfica del Mixer

Este modelo es soportado naturalmente por el proyecto Obelix, ya que no involucra ninguna propiedad especial del *Core* de señalización. La funcionalidad es resuelta íntegramente por los *User Agents*. De acuerdo a pruebas realizadas, el único *softphone* gratuito y libre que puede realizar esto es *Twinkle*, pero su capacidad está limitada a poder mezclar 2 flujos de audio, puede enviar a un usuario su propio audio mezclado con el de un tercero. Se puede pensar en un esquema donde los usuarios fueran agregando a otros utilizando su línea extra (línea extra de *Twinkle*) para unir nuevos usuarios a la conferencia, el gran problema de esta solución (además de su poca practicidad) es que si uno de estos usuarios termina la sesión, inmediatamente la conferencia se divide en 2 nuevas conferencias, salvo que el nodo que finalice sesión solo haya estado utilizando una línea para unirse a la conferencia.

Tomando las ideas del anterior modelo, el modelo Servidor Centralizado trabaja de la siguiente forma.

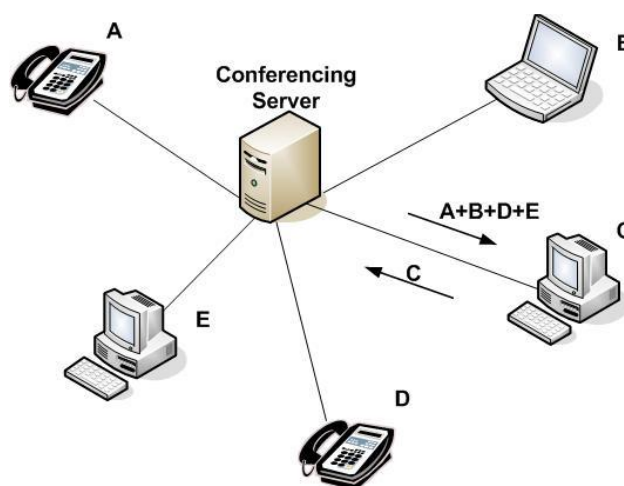


Figura 5. 17 - Servidor Centralizado de Conferencias

El servidor centraliza toda la señalización y audio correspondiente a la conferencia. Como se aprecia en la figura 5.17 la mezcla de audio la realiza de igual forma que el modelo anterior. Sus principales ventajas son:

- Permite administración centralizada.
- Ancho de banda óptimo de cara a los clientes.
- Los *UA* clientes no tienen por qué tener la capacidad de mezclar diferentes audios.
- Si fuese necesario el servidor puede realizar recodificación del audio “mezcla” con el códec acordado con cada *User Agent* (“*transcoding*”).

Su principal desventaja radica en que es un único punto de falla. Actualmente con el desarrollo de las *IP PBX* existen técnicas para lograr alta disponibilidad de estos servidores, en particular en el capítulo 7 se describe la implementación de esto último.

Por todas sus ventajas y la posibilidad de mitigar el efecto de su principal desventaja, se opta por este modelo.

5.4.2 – *FreeSWITCH* Como Servidor Centralizado de Conferencias Integrado al *Core* de Señalización

Para el manejo de conferencias en forma centralizada utilizaremos la capacidad de la *IP PBX FreeSWITCH*. Esta cuenta con esta funcionalidad implementada “*out of the box*”.

La forma de integrar esta funcionalidad al *Core* de señalización (*SIP Proxy*) se basa fundamentalmente en que el *Proxy* redirigirá los mensajes de INVITE destinados a “internos especiales” hacia el servidor de conferencias implementado por *FreeSWITCH*. Estos internos deben estar configurados en el *Dialplan* de *FreeSWITCH* de forma que cada uno tenga asociado una sala de conferencias diferente. Un detalle importante es que ninguno de estos internos representará un usuario real del sistema, sino que corresponden a la entidad del tipo “sala de conferencias”.

Por lo tanto, cada usuario del sistema podrá acceder a una sala de conferencias discando simplemente un interno comprendido entre el 3000 y el 3019. Cada uno de ellos se corresponde a una sala diferente.

Por las mismas razones que en el subcapítulo 5.2.3, el servidor de conferencias no autenticará usuarios. Tampoco atenderá paquetes SIP con dirección *IP* distinta a la de SIP *Proxy*.

El SIP *Proxy* tratará un INVITE que ingresa con *URI* SalaX@obelix, (donde SalaX debe ser un número del 3000 al 3019) modificando dicha *URI* por SalaX@Conference.obelix, posteriormente resolverá ese dominio y como resultado lo enviará al servidor de conferencias. Por lo tanto *FreeSWITCH* debe ser configurado para atender dicho dominio.

Los detalles de la implementación en la lógica de procesamiento por parte del *Proxy* se encuentran en el Anexo 1. La configuración de *FreeSWITCH* para actuar como servidor de conferencias se encuentra en el Anexo 2. A continuación se muestra el diagrama de flujo correspondiente al caso de uso estándar correspondiente a usuarios ingresando a una misma sala de conferencias.

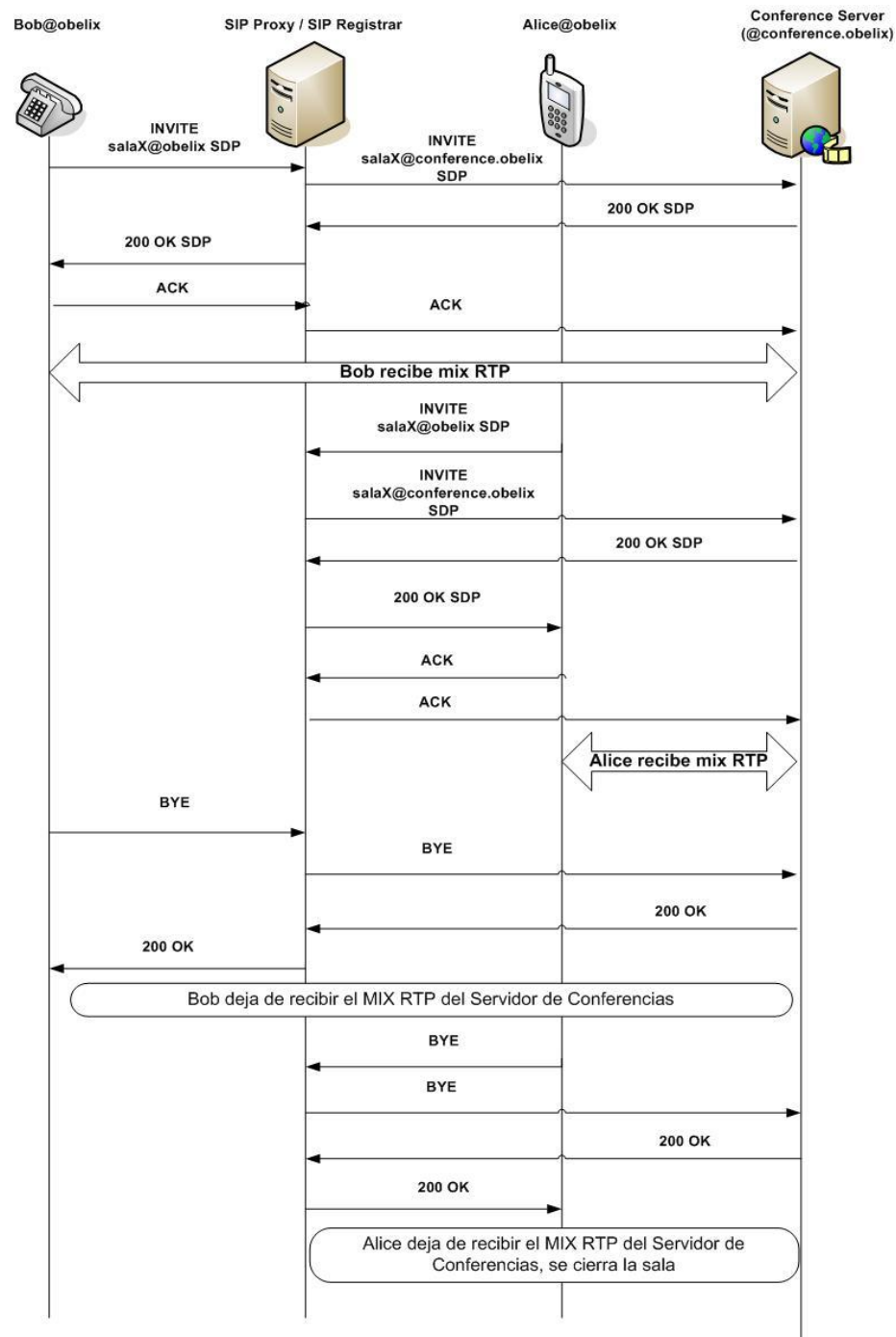


Figura 5. 18 - Diagrama de flujo correspondiente a usuarios ingresando a una misma sala de conferencia

En el diagrama anterior se observa como el SIP *Proxy* encamina el INVITE que genera el *User Agent* Bob hacia el servidor de conferencias. Esto lo logra cambiando la parte de dominio de la *URI* como fue descripto anteriormente.

Además de lo anterior el *Proxy* agrega el *Header Record-Route* colocando su *IP* en él, de esta forma obliga a futuras *Request* de la misma *Call-ID* a viajar a través de él (*Sip Proxy "Stateful"*). Cabe destacar que en esta situación no es necesario para el correcto funcionamiento de este escenario que el *Proxy* sea "*stateful*", si sucede para el procesamiento de los escenarios que involucran las *features* de *Music On Hold* y *Voicemail*.

Desde el punto de vista del *accounting* si el mismo fuera "*stateless*" se perdería registro de los mensajes *BYE* debido que estos no se verían forzados a pasar por el *Proxy*.

A modo de ejemplo el *BYE* generado por Bob viajaría directo a la *IP* del servidor de conferencias, esto se debe a que el *BYE* se encaminaría según su *URI*, en este caso la misma es formada con el contenido del *Header "Contact"* recibido en el 200 OK del servidor de conferencias.

Debido a que interesa contar con el *accounting* completo del escenario, el *Proxy* tratará en forma "*stateful*" los mensajes de *INVITE*.

La lógica del escenario del diagrama anterior (caso de uso de usuarios accediendo a una sala de conferencias) no difiere a la de un establecimiento de llamada estándar, lo cual mantiene la implementación simple.

5.4.3 – Dimensionamiento

Este subcapítulo tiene como objetivo determinar cuál es el tráfico que el servidor de conferencias debe ser capaz manejar en la "*Busy Hour*". Para esto se toma como referencia un criterio de diseño utilizado por Microsoft en su sistema *Lync Server 2010* [66]:

"5% of users will be in conferences during working hours. Thus, in an 80,000-user pool, as many as 4,000 users might be in conferences at any one time."

Dimensionaremos entonces según el criterio del peor caso aplicado a la anterior cita. Por lo tanto la funcionalidad de puentes de conferencia de ser capaz de soportar concurrentemente el 5% de la totalidad de usuarios del sistema (el 5% de 10000 usuarios), debiendo mantener 500 usuarios en conferencia.

5.4.5 – Pruebas de carga unitarias

La presente sección, al igual que en subcapítulos anteriores de pruebas de carga, pretende establecer en forma práctica la carga límite que es capaz de soportar de *FreeSWITCH* en esta ocasión operando como servidor centralizado de conferencias. Para esto se utilizará el mismo *Hardware* de pruebas que anteriores instancias:

PBX FreeSWITCH

- CPU: Intel *Core 2 Duo* E8400 3.0 GHz
- Memoria *RAM*: 2 GB DDR2 800 MHz
- *Gigabit LAN*
- *SO*: CentOS 6.1
- *FreeSWITCH* Version 1.3.0+git~20120829T032350Z~5a3a1e3b44 (1.3.0; git at commit 5a3a1e3b44 on Wed, 29 Aug 2012 03:23:50 Z)

La metodología aplicada consiste en emular el tráfico de señalización SIP y audio que debe ingresar al servidor en una situación estándar de conferencia centralizada. Para lograrlo, se crean dos escenarios con *SIPp*:

El primero es el encargado de ingresar a todos los usuarios a una misma sala de conferencias. Todos estos ingresan en forma concurrente, simulando la situación en que todos los usuarios acuerdan ingresar a una sala en determinado horario.

El segundo escenario se encarga de ingresar a la conferencia un usuario que oficie de orador de la misma. Se asume que en una conferencia de varios usuarios, siempre uno de ellos “tiene la palabra” y los restantes guardan silencio. Por lo tanto para la creación de este escenario se recurrió a la misma técnica utilizada para generar el escenario de pruebas de *Voicemail*. *SIPp* es el encargado de enviar audio hacia el servidor de conferencias, este último lo repetirá hacia los usuarios generados con el primer escenario *SIPp* descripto. Todos los audios de la prueba están codificados con *G.711*, por lo tanto el servidor de conferencias no realizará “*transcoding*”.

Un detalle importante consiste en que los usuarios generados por el primer escenario, no envían audio *RTP* hacia el servidor de conferencias. Existen varios motivos por lo cual se toma esta decisión, el más importante proviene del hecho que un usuario “oyente” no ingresa a la mezcla de audio que es repartida a todos los restantes oyentes. Mientras este último guarde silencio (característica principal del oyente) no se vence el umbral establecido por un parámetro del módulo de conferencias de *FreeSWITCH*, por lo tanto no entrará a la mezcla.

Para aumentar el realismo de esta prueba, se podría haber optado que todos los usuarios de la conferencia (descontando el orador) envíen audio de “silencio”. También se podría agregar un segundo usuario orador, de forma de tener 2 audios en la mezcla. Existen muchas posibles formas de emular el comportamiento de los usuarios de la conferencia ya que depende básicamente de la voluntad de los mismos.

Para no perder de vista el objetivo de esta prueba, se optó por emular el comportamiento descrito al principio del subcapítulo (un orador y varios oyentes).

Los archivos correspondientes a los escenarios generados y su forma de ejecución se encuentran en el Anexo 3.

Los resultados fueron los siguientes:

Usuarios en conferencia	CPU <i>FreeSWITCH</i> (%)	Memoria RAM <i>FreeSWITCH</i> (MBytes)
202	16.7	332
302	24.8	337
402	37.3	370
502	47.6	410
602	60.1	456
702	71	502
802	81.2	552
902	91.9	611

Tabla 5. 6 Resultado Prueba de Carga a Servidor de Conferencias

En cada prueba se ingresó con un *softphone* a la sala de conferencias para verificar la calidad de audio que un usuario oyente recibe del *Mixer*. Es por esta razón que en cada prueba se agregó a los usuarios generados por *SIPp*, (por ejemplo 200 oyentes y un 1 orador) un usuario extra que corresponde al *softphone* de test.

Como análisis de estas pruebas se concluye que el *Hardware* provisto para el servidor de conferencias soporta 902 usuarios en conferencia (cada uno con el comportamiento explicado anteriormente).

Posteriormente a esta prueba se verificó que la carga del servidor no depende de la cantidad de salas en que estén repartidos los usuarios. Genera una carga similar de procesador y memoria RAM, 901 usuarios repartidos en 2 conferencias, que los 901 usuarios en una sola.

A continuación se presentan los datos recabados en forma de gráficos:

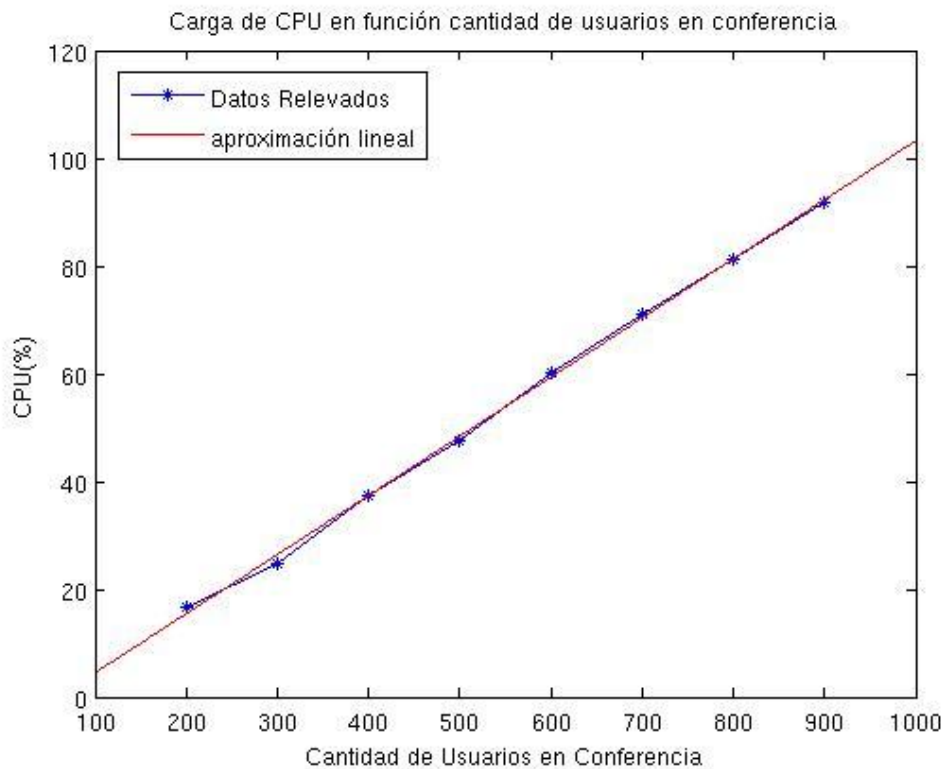


Figura 5. 19 - Porcentaje de utilización del CPU en función de la cantidad de usuarios en conferencia

Al igual que en pruebas de carga anteriores, se establece una regresión lineal a los datos extraídos de forma de obtener una curva de carga de procesador para el servidor de conferencias. La recta de carga es:

$$CPU(\%) = 0.1098 * (Cantidad\ de\ Usuarios\ concurrentes\ en\ Conferencia) - 6.5440$$

Se deduce entonces que para la carga dimensionada de 500 usuarios concurrentes, el CPU del servidor se cargará en un 48.3%.

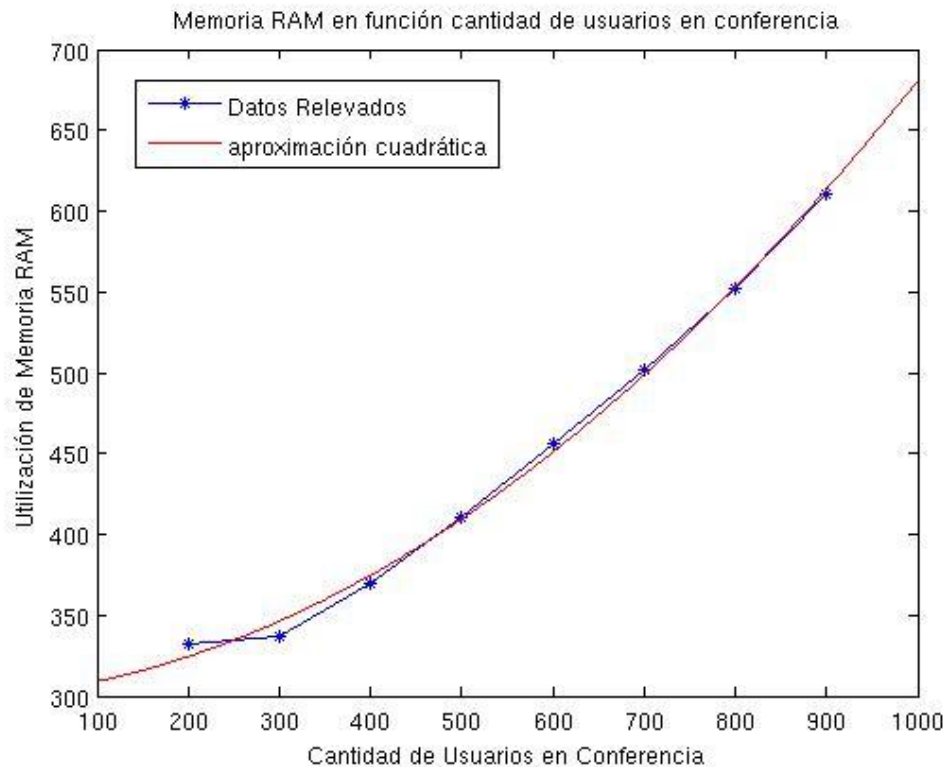


Figura 5. 20 - Utilización de memoria RAM en función de la cantidad de usuarios en conferencia

Para la memoria *RAM* se obtuvo un comportamiento cuadrático, la curva ajustada es, donde X corresponde a la cantidad de usuarios en conferencia:

$$\text{Memoria RAM (MBytes)} = (3,28 \times 10^{-4})(X)^2 + 0,0569 * X + 300$$

Se deduce entonces que para la carga dimensionada de 500 usuarios concurrentes, se utilizarán 410 *MBytes* de memoria *RAM* del servidor.

Estos resultados infieren que un servidor de estas características puede manejar la carga dimensionada sin mayores dificultades, de todas formas la arquitectura final de Obelix prevé una solución escalable a N servidores. En el caso particular de conferencias además se implementa un mecanismo de recuperación de las conferencias ante caídas del servidor, resultando esta última transparente para todos los usuarios víctimas de la caída. Este tema particular se tratará en el capítulo 7 y su implementación en el anexo 2.

5.5 – Transferencias

El servicio de transferencias permite la derivación de llamadas a otros internos. Básicamente su funcionamiento se describe en el escenario que se da cuando uno de los usuarios que cursando una llamada con otro, precisa hablar con un tercero. A lo largo de este apartado se explicará el funcionamiento de las transferencias básicas, las transferencias atendidas y desatendidas, con los flujos de señalización SIP de las mismas.

5.5.1 – Descripción del Servicio

Antes de describir con profundidad las características del servicio de transferencia es conveniente establecer y definir algunos conceptos que se manejarán a lo largo del análisis de la funcionalidad. Por lo tanto definiremos los actores presentes y las funciones que cumplen cada uno de estos en la transferencia.

Transferido (*Transferee*): es la parte transferida al interno a la que se solicita realizar la transferencia.

Destinatario de la Transferencia o Destinatario (*Transfer Target*): es a donde se quiere realizar la transferencia.

Transferente (*Transferor*): es quien recibe la llamada del transferido y a su vez se encarga de realizar la conexión entre el transferido y el destinatario.

Por otro lado se definirán los roles que cumplirán los actores en el transcurso de una transferencia y los requerimientos que deberán de cumplir según el escenario.

Originador: es quien desea colocar la llamada al Receptor, es el actor responsable de originar el primer INVITE de una sesión al Facilitador.

Facilitador: recibe la llamada por fuera de la ya establecida desde el Originador, establece la llamada con el Receptor y conecta al Originador con el Receptor. Normalmente el Facilitador actúa en representación del Originador.

Receptor: la parte a la que el Originador conecta finalmente.

5.5.2 – Requerimientos

Existen algunos requerimientos a cumplir para que se logre establecer una transferencia de forma exitosa. Pero no solamente para cumplir la funcionalidad en sí, también se establecen requerimientos para que al momento de implementar la funcionalidad esta sea compatible con los sistemas que se utilicen para la comunicación, ya sean *softphones* o *hardphones*.

Por lo tanto los requerimientos establecidos en la RFC-5589 son:

- Cualquiera de las partes involucradas en una sesión SIP debe ser capaz de transferir a cualquier otra parte en el transcurso de esa sesión.
- Tanto el transferente como el transferido no deben ser eliminados de la sesión como consecuencia de la transacción de transferencia.
- El transferente debe saber si la transferencia fue o no fue exitosa.
- El transferido debe ser capaz de sustituir el dialogo existente por un dialogo nuevo.
- Tanto el transferido como el transferente deben indicar su capacidad para manejar las directivas requeridas para establecer la transferencia.
- El transferente debe proveer al destinatario y al transferido la información sobre la naturaleza y progreso de la operación de transferencia que se está llevando a cabo.

5.5.3 – Uso del Método REFER en Transferencias

Antes de precisar los diferentes modos de transferencias es necesario aclarar como es el funcionamiento del método REFER para efectuar las transferencias. En el capítulo 2 se mencionó las diferentes capacidades del método REFER y que la utilidad del mismo por excelencia es para efectuar las transferencias. La RFC-3515 especifica con un grado de detalle superior las diferentes aplicaciones y usos del método en cuestión.

Un REFER puede ser invocado por el transferente para causar que el transferido provoque un INVITE al destinatario, hay que tener en cuenta que una transacción exitosa de un REFER no termina la sesión entre el transferente y el transferido. Si las partes involucradas desean terminar la sesión que tienen establecida deben realizarlo mediante un *Request* de BYE. El flujo de *Media* negociado entre el transferido y el destinatario es totalmente independiente y no se ve afectado por el flujo de *Media* negociado entre el transferido y el transferente. En particular, el INVITE enviado por el transferido tendrá el mismo cuerpo de *SDP* que podría llegar a tener el transferido en caso de realizar el INVITE por su cuenta y no

provocado a consecuencia del REFER. Además, la disposición de los flujos de *Media* entre el transferido y el transferente no se verá afectada por el método REFER.

Se debe tener en cuenta que para casos en que se precise alterar el flujo de *Media* negociado se deberá realizar con señalización adicional. Concretamente esto se realiza mediante el método re-INVITE que renegocia los parámetros de *SDP*.

Para que se realice una transferencia tanto el transferido como el transferente podrán reusar un dialogo ya existente establecido por un INVITE para enviar el REFER, esto resultará en un único dialogo compartido para dos usos (un uso de INVITE y otro de SUBSCRIBE), esto se detallara en los diagramas de flujo SIP que se exponen al final de esta sección. De todos modos la idea reflejada en la RFC pretende que no se haga una reutilización del diálogo. Las dificultades existentes son mencionadas con mayor profundidad en la RFC-5057 [67].

Existen algunas razones por las que reutilizar el diálogo es un método de efectuar la transferencia válida. Estas son:

- 1) No hay forma de asegurar que el REFER enviado en un nuevo diálogo pueda alcanzar un punto en particular en la que se vea implicada la transferencia. Muchos factores influyen, incluyendo detalles de implementación y cambios en el enrutamiento efectuado por el *Proxy* entre un INVITE y un REFER, provocando que el REFER pueda ser enviado al lugar incorrecto. Enviar el REFER dentro de un mismo diálogo asegura llegar al punto en concreto donde se está cursando la llamada.
- 2) No está claro como asociar el uso de un INVITE existente con un REFER llegando en un dialogo nuevo, donde es obvio que la asociación se realizó cuando el REFER proviene del INVITE del dialogo ya establecido.
- 3) Existen algunos detalles en la autorización de los REFER provenientes fuera de diálogo, la política de autorización para los REFER en la mayoría de las implementaciones reutiliza la política de autorización de los INVITES.

En el caso del primer punto, este se resuelve utilizando una *Globaly Routable UA URI* (GRUUs). El problema 2 puede ser resuelto direccionando mediante el *Target-Dialog* definido tanto en capítulo 2 como en la RFC-4538 [6] en forma específica, permitiendo mediante esta solución reutilizar la política de autorización en el REFER.

Como resultado, si el transferido soporta las extensiones de *Target-Dialog* y el transferente conoce que la *Contact-URI* es enrutable por fuera del dialogo, el REFER deberá ser enviado en un nuevo diálogo. Si la naturaleza del *Contact-URI* es desconocida o si es desconocido el soporte de los *Target-Dialog*, el REFER deberá ser enviado dentro del dialogo existente.

El transferido deberá estar preparado para recibir un REFER ya sea por dentro o por fuera del dialogo. Una forma de que el transferente conozca que la *Contact-URI* es enrutable por fuera del dialogo es por validación o si satisface las propiedades descriptas en la especificación GRUU.

5.5.1.2 – Transferencia Desatendida

La misma, consiste en que el transferente provee el contacto del destinatario al transferido, este último intenta establecer una sesión usando el contacto y reportes obtenidos como resultado del intento realizado por el transferente. La señalización establecida entre el transferente y el transferido no se finaliza, por lo tanto, la llamada es recuperable si el destinatario no es alcanzable o no está disponible. Se debe de notar que la información de contacto del destinatario es expuesta al transferido. El contacto del destinatario se utilizará para efectuar nuevas llamadas en el futuro.

Los participantes en una transferencia básica deben indicar que soportan los métodos REFER y NOTIFY en los campos *Allow* del INVITE, 200 OK en respuesta al INVITE enviado previamente y en el mensaje OPTIONS, además los participantes deben indicar el soporte para los *Target-Dialog* en el campo *Supported*.

En la diagrama de flujo SIP correspondiente a este tipo de transferencia se describe cual es la señalización para la realización de una transferencia básica. La primer columna describe el dialogo en donde suceden los mensajes que se describen en el flujo.

El flujo de *Media* es manipulado a través de los re-INVITE, pero otros mecanismos son válidos.

Cada uno de los flujos muestra el dialogo que se mantiene entre el transferente y el transferido que permanecen conectados (en espera) durante el proceso efectuado por el REFER, permitiendo otorgar flexibilidad para recuperarse de una falla, mecanismo no es necesario que este implementado. Si el agente transferente no quiere participar en el proceso de REFER y no tiene intención de

asistir con la recuperación de la transferencia fallida podrá emitir un BYE al transferido ni bien la transacción efectuada por el REFER haya sido completada. En la figura se describe una extensión que soporta un *Target-Dialog* indicando esto al transferente, como resultado, el transferente envía el REFER por fuera del dialogo de INVITE. El transferido tiene la posibilidad de adaptar este REFER al dialogo existente usando el encabezado *Target-Dialog* que referencia al dialogo ya establecido.

5.5.1.1 – Transferencia Atendida

La transferencia atendida se engloba dentro de lo que se denomina Transferencia con Consulta en Espera. Dicha transferencia implica una sesión establecida entre el transferente y el destinatario antes de que tome lugar la transferencia del transferido hacia el destinatario.

Una característica interesante para el transferente es poder anunciar al destinatario del intento de transferencia, esto es útil ya que muchos UA pueden utilizar el usuario del campo *From* para indicar de donde proviene la llamada, útil para displays en teléfonos ya sean softphones o *hardphones*.

La denominada transferencia atendida (*Attended Transfer*) implica que el transferente coloca al transferido en espera, establece una llamada con el destinatario para indicar de la transferencia a efectuar, coloca al destinatario en espera y luego procede a transferir la llamada haciendo uso, a nivel de señalización, del campo *Replaces* en el encabezado *Refer-To*.

Este mecanismo es comúnmente utilizado en servicios provistos por PBX.

El *Contact URI* del destinatario debe de ser usado por el transferente como el *Refer-To URI* a menos que la *URI* sea no enrutable por fuera del dialogo. En caso contrario la *Address of Record* (AOR) del destinatario deberá ser usada, esto es que la misma *URI* del transferente será utilizada para establecer la sesión con destinatario. En caso que el INVITE disparado sea enrutado a una UA diferente que el destinatario, el campo *Require: Replaces* deberá ser utilizado en el INVITE disparado. Este mecanismo se utiliza para prevenir que un UA incorrecto que no soporta *Replaces* de ignorar el *Replaces* y de responder al INVITE sin un dialogo ajustado.

Existe la posibilidad que un enrutamiento hecho por un *Proxy* pueda prevenir que el INVITE disparado alcance un mismo UA. De ocurrir este comportamiento

el invite disparado sufrirá un timeout (403, 404, error). El transferido tendrá que reintentar la transferencia con el *Refer-To URI* fijado en el *Contact URI*.

5.5.3 – Implementación en el *Core*

La implementación en el *Core*, concretamente en *Kamailio*, no requiere mayor complejidad ya que solo se encarga de redireccionar los diversos REFERS que le llegan. Esto implica que al momento de efectuar una transferencia del tipo atendida o desatendida la señalización provista por *Kamailio* solo se encarga de establecer los mensajes necesarios para lograr el cometido de la transferencia deseada.

5.5.4 – Flujos Básicos

5.5.4.1 – Transferencia Desatendida o Ciega

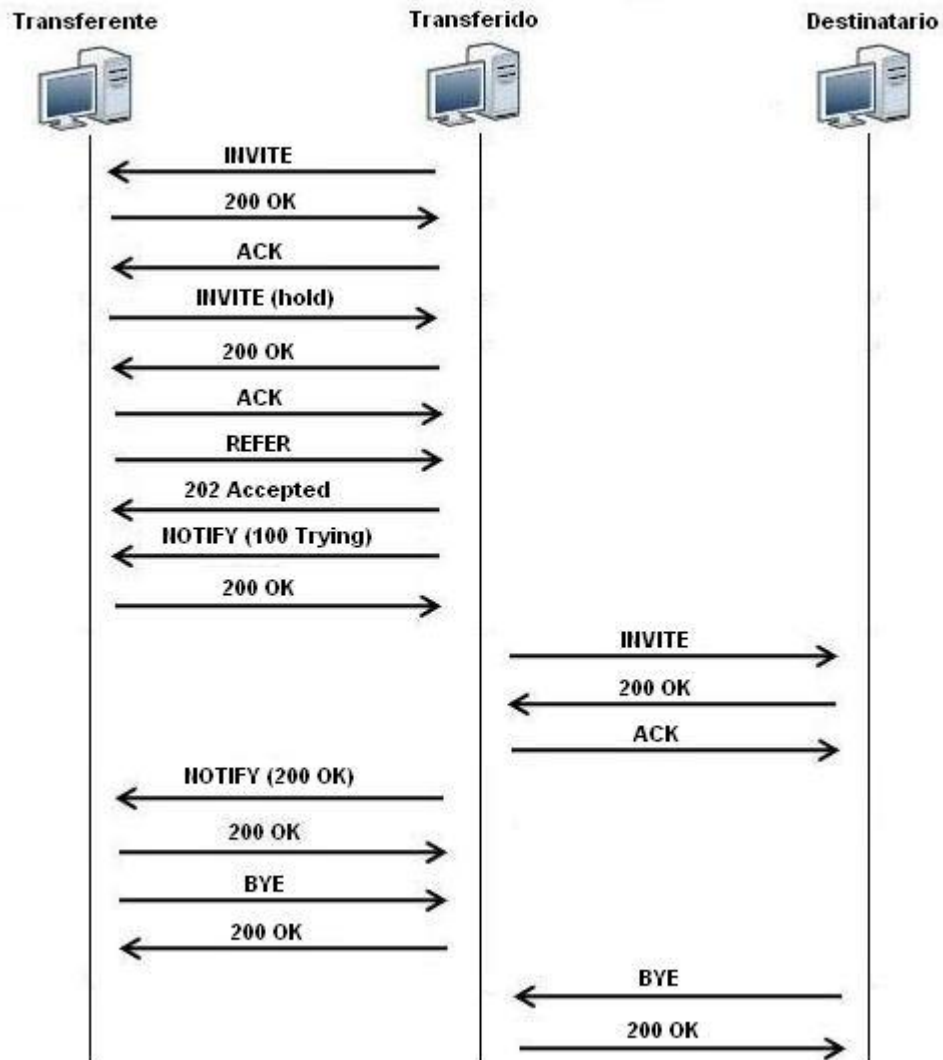


Figura 5. 21 - Diagrama de flujo SIP de transferencia desatendida o ciega

5.5.4.2 – Transferencia Atendida

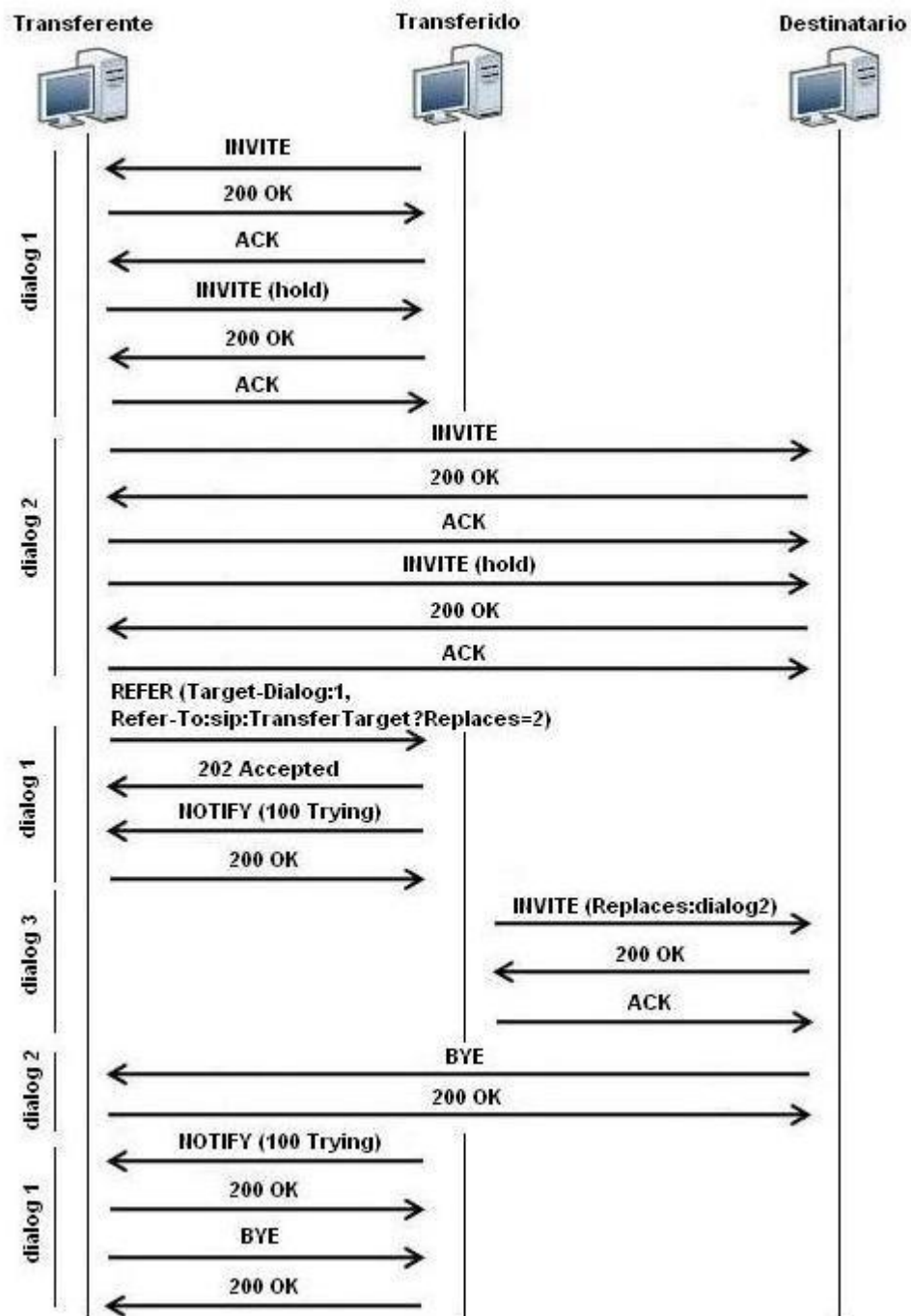


Figura 5. 22 - Diagrama de flujo SIP correspondiente a la transferencia atendida

5.6 – Servicio de Presencia y Mensajería Instantánea

Las funcionalidades descritas en los apartados anteriores fueron implementadas en servidores dedicados especialmente a estas, incluso se adoptaron sistemas específicos para garantizar que las funcionalidades adjudicadas puedan cumplir con los requisitos establecidos.

La funcionalidad de Servidor de Presencia y Mensajería Instantánea es resuelta enteramente por el *Core* de señalización, concretamente hablando, quien realiza la funcionalidad es *Kamailio*.

El servicio de presencia esta implementado en base a las recomendaciones de la RFC-3856 [7] que se describirán los rasgos generales de este servicio. Respecto a la mensajería instantánea, se fundamentará en base a la recomendación de la RFC-3428 [8].

5.6.1 – Descripción del Servicio de Presencia

La definición de presencia se ajusta a la voluntad y capacidad de un usuario para comunicarse con otros usuarios de la red, normalmente se adjudica el concepto de presencia a dos indicadores de estado, *on-line* y *off-line*. A efectos de lo que se expondrá en este capítulo el concepto supera ampliamente la definición anterior, las notificaciones y suscripciones de presencia son definidas por un paquete de eventos (*event package*) dentro del marco de eventos de notificaciones de SIP.

La información de presencia constituye la voluntad y capacidad de comunicarse a través de un conjunto de dispositivos, la RFC-2778 [68] define un modelo y terminología para la descripción de sistemas que proveen servicios de información de presencia, que es quien acepta, guarda y distribuye la información requerida a las partes interesadas. Un protocolo de presencia es aquel que provee un servicio de presencia sobre una red *IP*.

El protocolo SIP puede ser utilizado como protocolo de presencia haciendo uso de los métodos SUBSCRIBE y NOTIFY mencionados en el capítulo 2. A su vez, SIP se ajusta adecuadamente a un protocolo de presencia ya que los servicios de locación contienen información de presencia en forma de registros. Concretamente SIP permite enrutar *Requests* desde cualquier usuario de la red al servidor que mantiene los estados de registración de los usuarios (servidor de tipo *stateful*). Este estado es un componente medular para los datos de

presencia del usuario siendo que las redes SIP pueden permitir que las *Requests* de SUBSCRIBE sean enrutadas al mismo servidor que mantiene los registros de usuario. Se concluye que las redes de este tipo pueden ser reutilizadas para establecer conectividad global frente a suscripciones y notificaciones de presencia.

El paquete de eventos se basa en el concepto de agente de presencia, es una nueva entidad lógica capaz de aceptar suscripciones, guardar estados de las suscripciones y generar notificaciones cuando hay cambios en la presencia. Esta entidad es definida de forma lógica ya que generalmente reside con otras entidades. Cabe aclarar que este tipo de paquete es compatible con CPP (*Common Presence Profile*), permitiendo a SIP ser compatible con otros sistemas de presencia compatibles con CPP.

5.6.1.1 – Definiciones a Tener en Cuenta

Presence User Agent (PUA): está encargado de manipular la información de presencia para ser enviada al receptor de la misma, siendo esta manipulación el efecto de otra acción o puede ser como consecuencia de la publicación de documentos de presencia.

Presence Agent (PA): es un *UA* SIP capaz de recibir *Requests* de SUBSCRIBE, responder a éstos y generar notificaciones de cambios del estado de presencia. Existe la posibilidad de realizar esto mediante un *Proxy/Registrar*.

Presence Server: entidad física que puede actuar tanto de agente de presencia como *Proxy Server* para los *Requests* de SUBSCRIBE. De actuar como PA está al tanto de la información de presencia, de actuar como *Proxy* los SUBSCRIBE son enviados a otra entidad que actuara de PA.

5.6.1.2 – Funcionamiento

Antes de detallar el funcionamiento de la presencia es necesario apelar a la definición existente en la RFC-3265 [69] que especifica con claridad que es un *event package*:

“Un Event Package es una especificación adicional que define un set de información de estado a ser reportado por un notificador a un suscriptor.”

Cuando una entidad denominada suscriptor quiere aprender sobre la información de presencia de un usuario en concreto crea un *Request* con el método SUBSCRIBE, identificando en el *Request-URI* la identificación del usuario del que se quiere recibir información de presencia utilizando SIP *URI*, SIPS *URI* o *presence URI*. El SUBSCRIBE enviado atraviesa los proxys existentes al igual que cualquier otro SIP *Request*, en algunos casos llega al servidor de presencia que puede generar tanto una respuesta al *Request* o enviarlo al servidor de presencia. La decisión tomada por el servidor de presencia de enviarlo al *Proxy* o terminar el SUBSCRIBE es una decisión local, existiendo una alternativa para tal configuración usando REGISTER.

El agente de presencia primero autentica la suscripción y luego la autoriza, de ser autorizada la respuesta será 200 OK, en caso contrario la suscripción se considerara pendiente con la respuesta de un 202. En ambos casos el agente de presencia envía un mensaje inmediato de NOTIFY conteniendo el estado de la presencia y la suscripción. Se debe tener en cuenta que de no tener la autorización correspondiente (estado pendiente) el estado de quien pide la presencia se considerará falso indicando como *offline* sin importar cual sea el estado actual del mismo. El mecanismo definido anteriormente existe para proteger la privacidad del solicitante que no querrá revelar que no ha otorgado autorización al suscriptor. Ni bien cambie el estado de quien envía la información de presencia, el agente de presencia generara un NOTIFY conteniendo los cambios de estado a todos los suscriptores con suscripciones autorizadas.

El mensaje de SUBSCRIBE establece un diálogo con el agente de presencia, representando el estado SIP entre un par de entidades para facilitar el intercambio de mensajes *peer-To-peer*, este estado contiene los números de secuencia de los mensajes en ambas direcciones (SUBSCRIBE desde el suscribir y NOTIFY desde el PA) en suma a un set de rutas y el *Target URI* remoto. El set de rutas es una lista de los SIP URIs que identifican los *Proxy* servers que serán atravesados a lo largo del camino de SUBSCRIBE refresh o *Request* de NOTIFY. El *Target URI* remoto es el SIP o SIPS *URI* que identifica el *Target* del mensaje (el suscribir en el caso de un NOTIFY o el PA en el caso de un SUBSCRIBE refresh).

Existe el procedimiento provisto por SIP denominado *record-routing* que permite a un *Proxy* mantenerse en el camino de los mensajes de NOTIFY y SUBSCRIBE posteriores, esto se realiza insertando el *URI* en el campo del encabezado *Record-Route* del *Request* de SUBSCRIBE inicial.

La suscripción se mantiene activa durante un intervalo de tiempo que es negociado por parte del SUBSCRIBE inicial. Para incrementar el lapso de tiempo negociado o evitar que la suscripción expire, se deberá enviar un SUBSCRIBE refrescando la suscripción antes que llegue al tiempo establecido en el campo *Expires*. El suscriptor puede terminar la suscripción enviando un SUBSCRIBE dentro del dialogo estableciendo en el campo *Expires* en cero, provocando la finalización absoluta e inmediata de la suscripción.

Se define una extensión SIP para la suscripción y notificación de eventos, relegando muchos de los aspectos de estos eventos a extensiones denominadas *event packages*. El nombre de paquete o *package name* es *presence*, éste aparece en el campo *Event* de los *Request* de SUBSCRIBE y NOTIFY. Los parámetros del *event package* definen parámetros adicionales que son llevados en el campo *Event*, pero para este tipo de paquetes (*presence*) no se definen parámetros adicionales.

El cuerpo del SUBSCRIBE puede contener información, el propósito del cuerpo depende del tipo, razón por la cual la mayoría de las suscripciones no tendrán cuerpo. El *Request-URI* combinado con el *event package* será suficiente para la información de presencia, aunque existen algunas excepciones en el tipo de cuerpo de mensaje como el caso que el SUBSCRIBE sea de tipo *filter document*. Este filtro, incluido en un *Request*, implica que sólo ciertos eventos de presencia pueden generar notificaciones o pedirán por una restricción para los datos devueltos en la *Request* de un NOTIFY. Si el *Request* de SUBSCRIBE no contiene un filtro esto comunica al PA que ningún filtro será aplicado, por lo que el PA deberá enviar NOTIFY según las políticas que tenga definidas. Respecto a la duración de las suscripciones, el valor por defecto está establecido en 3600 segundos.

El mensaje NOTIFY contiene varios cuerpos que describen el estado del recurso suscripto, el cuerpo del mensaje está en un formato listado en el campo *Accept* del SUBSCRIBE o un paquete específico por defecto si en el campo *Accept* fue omitido del SUBSCRIBE. En este *event package*, el cuerpo de la notificación contiene un *presence document*, este describe la información de presencia a quien se ha suscripto.

El procesamiento de un SUBSCRIBE por parte del *Notifier* se basa en el enrutamiento establecido por reglas del *Proxy*, será al momento de llegar al agente de presencia. La información de presencia es altamente sensible, debido a que las consecuencias de divulgar información de presencia pueden ser

severas, por lo tanto, se establecen criterios fuertes en materia de procesamiento de suscripciones por parte del PA apuntando a la autenticación y autorización.

La generación de NOTIFY por parte del *Notifier* tiene como necesario proveer información detallada de cuando enviar el mensaje, como establecer el estado del recurso, como generar información neutral o falsa de estado y determina cuando la información del estado es completa o parcial.

El procesamiento del NOTIFY por parte del Suscriber se realiza en función del *event package* que describe el proceso que debe realizar al momento de recibirlo. Cada NOTIFY contiene un *presence document* o un *document* representando el estado completo y coherente del servicio de presencia. Dentro del diálogo, el *presence document* en el NOTIFY con el campo CSeq más alto será el que se está utilizando. Cuando no hay un *document* presente en el mensaje de NOTIFY, el *presence document* que se encuentre en el NOTIFY con próximo CSeq mas alto es el que se utilizará.

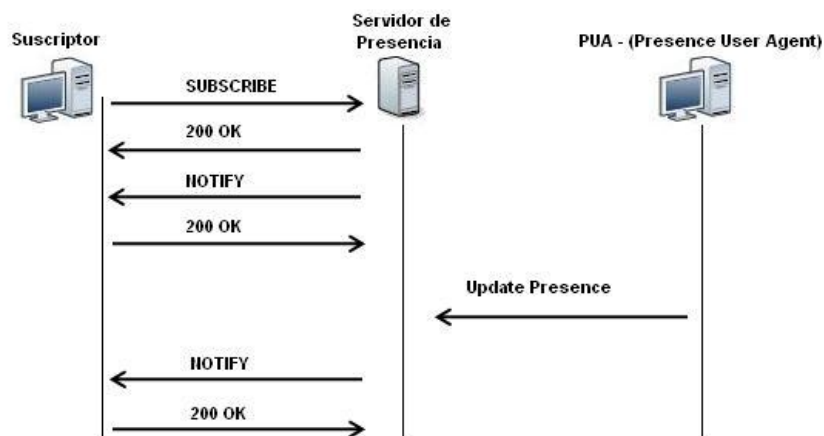


Figura 5. 23 - Descripción de intercambio de mensajes de presencia

5.6.2 – Descripción del Servicio de Mensajería Instantánea

Se reconoce a la mensajería instantánea como la transferencia de mensajes o contenido entre usuarios en un modo casi de tiempo real, siendo estos mensajes o contenido relativamente cortos. Generalmente los mensajes intercambiados no son guardados y en se emplean conformando una conversación, apelando a una velocidad considerable de forma de poder enviar y recibir los mensajes por parte de los participantes para mantener una conversación.

El protocolo SIP provee mecanismos que son útiles para aplicaciones de presencia y para comunicaciones del tipo sesión orientada, pero no para mensajes instantáneos. El método empleado por SIP para otorgar esta funcionalidad es utilizar el método MESSAGE definido en la RFC-3261 que permite el envío de mensajes instantáneos con la información enviada por este en el cuerpo del *Request* enviado. Cabe aclarar que los MESSAGE *Request* por si solos no inician un dialogo SIP, en efecto, los MESSAGE *Request* son enviados en un dialogo ya iniciado por otros SIP *Request*.

Entre los mensajes no existe una asociación explicita, cada IM se envía de forma independiente sin establecer una “conversación”, esta idea existe solo a nivel de usuario, en lo que ve el cliente que este manejando en su equipo, contrastando fuertemente con la idea de sesión donde si existe una conversación explicita con un comienzo y fin bien definidos (INVITE y BYE respectivamente).

5.6.2.1 – Operativa de la Mensajería Instantánea

Cuando un usuario quiere enviar un mensaje instantáneo a otro, quien lo envía formula un SIP *Request* usando el método MESSAGE, la *Request-URI* será la *Address of Record* para el receptor del mensaje, pero existen algunos dispositivos de direcciones que tendrán información sobre la ubicación del receptor. El *Request* atravesará los SIP *Proxy* usando una variedad de transportes antes de alcanzar el destino final, el destino para cada salto se ubica usando las reglas de resolución de direcciones detalladas en las especificaciones del protocolo SIP, y es en cada pasada por el *Proxy* que este reescribirá el *Request-URI* en función de la información de ruteo.

Las respuestas temporales y final al *Request* serán devueltas al emisor como cualquier otro SIP *Request*, generalmente una respuesta de un 200 OK será generada por el UA del receptor final indicando que el UA ha aceptado el

mensaje, pero el usuario no lo ha visto. Se debe tener especial cuidado que los *Request* de MESSAGE no establecen diálogos.

El procesamiento de los MESSAGE por parte de la UAC cumple con ciertas características. Toda UAC que soporte el método MESSAGE debe estar preparada para enviar *Request* de MESSAGE con un cuerpo conformado por texto plano. Debido a que este tipo de mensajes no inicia diálogos, los UA no deben insertar el campo *Contact* en el *Request* de MESSAGE. Una UAC puede asociar un MESSAGE *Request* a un dialogo ya existente, si un MESSAGE *Request* es enviado dentro de un diálogo establecido, queda asociado con la sesión de *Media* o las sesiones asociadas a ese dialogo establecido previamente.

La UAC al recibir un 200 OK en respuesta a un MESSAGE *Request* asume que el mensaje fue entregado al destinatario final, pero no se puede asumir que el destinatario final haya realmente leído el mensaje instantáneo. Al recibir un 202 Accepted el mensaje ha sido entregado a un *Gateway*, un servidor del tipo *store and forward* u otro tipo de servicio que eventualmente se encargue de entregar los mensajes enviados, en este caso la UAC no debe de asumir que el mensaje haya sido entregado al destino final. Además la misma UAC podrá agregar un campo de *Expires* indicando el límite de validez del contenido del mensaje.

A nivel de *Proxy*, el que esté ubicado aguas abajo podrá realizar una bifurcación de la *Request* de MESSAGE. De ocurrir esto, el *Proxy* reenviara las respuestas finales hacia los equipos aguas arriba de él, incluso si recibe múltiples respuestas finales, esto provocara que la UAC no tendrá forma de detectar si ocurre o no una bifurcación de los mensajes, por lo que la UAC no deberá asumir que una respuesta final dada representa a la única que recibió la *Request*.

El procesamiento a nivel de UAS de los MESSAGE *Request* se realiza según las reglas definidas en el protocolo SIP. Un UAS al recibir el MESSAGE debe de responder con una respuesta final de forma inmediata, pero la misma no está exigida de mostrar el mensaje al usuario antes o después de responder con un 200 OK, pero al recibir este mensaje no se tiene certeza de que el usuario haya leído el mensaje. Los mensajes pertenecientes a la clase 2xx en respuesta al *Request* no deben de contener un cuerpo, por tanto la UAS no deberá insertar el campo *Contact* en una respuesta perteneciente a esta clase. La UAS deberá de responder un 202 Accepted indicando que el mensaje fue aceptado pero no puede garantizar la entrega punta a punta.

Las respuestas del tipo 4xx y 5xx indican que el mensaje no fue entregado de forma satisfactoria, las del tipo 6xx indican que el mensaje fue entregado de

forma exitosa pero que se rechazó. Toda *UAS* que soporte el método MESSAGE debe de estar preparado para soportar cuerpos de mensaje de texto plano.

Un MESSAGE *Request* ha expirado si el tiempo definido en el campo *Expires* ha sido superado, este tiempo se obtiene de analizar el campo *Expires*, si es que está presente. De no estar presente no expira. Si además de contener el campo *Expires* contiene el campo *Date*, la *UAS* debe interpretar el campo *Expires* como una variación de tiempo del campo *Date*, de no contener el campo *Date* la *UAS* deberá interpretar el campo *Expires* como una variación del tiempo recibido en la *Request*.

El uso del *URI* del mensaje instantáneo es para referenciar de forma *im:user@domain*, estos son abstractos y se traducirán concretando de forma dependiente al protocolo. Si la *UA* es presentada con una *URI* del tipo IM como la dirección de un mensaje instantáneo deberá de resolver como una SIP *URI* y colocar el resultado de la resolución en la *Request-URI* del *Request* de MESSAGE antes de ser enviada. Los campos *Record-Route* y *Route* no deben contener IM *URI* ya que estos campos contienen SIP o SIPS *URI* de acuerdo a las reglas establecidas en la RFC-3261.

A nivel de procesamiento del *Proxy* este enruta los MESSAGE *Request* de acuerdo a las reglas establecidas en el protocolo SIP, notar que los MESSAGE *Request* pueden copiarse permitiendo enviar el mensaje a varios destinos donde le usuario se encuentre registrado (*forking*).

5.6.2.2 – Definición del Método MESSAGE

La especificación de la RFC-3428 en la cual se especifica la mensajería instantánea define un método SIP denominado MESSAGE.

Como otros métodos, todos los demás en concreto, el método MESSAGE es sensible a mayúsculas y minúsculas. En la referencia a la RFC-3428 se especifican los diferentes campos del encabezado que posiblemente pueden ser utilizados en el método MESSAGE tanto en la respuesta como en el *Request*.

El ejemplo que se puede utilizar para observar como es la operativa de la transferencia de mensajes instantáneos entre dos puntas puede verse en el siguiente diagrama SIP.

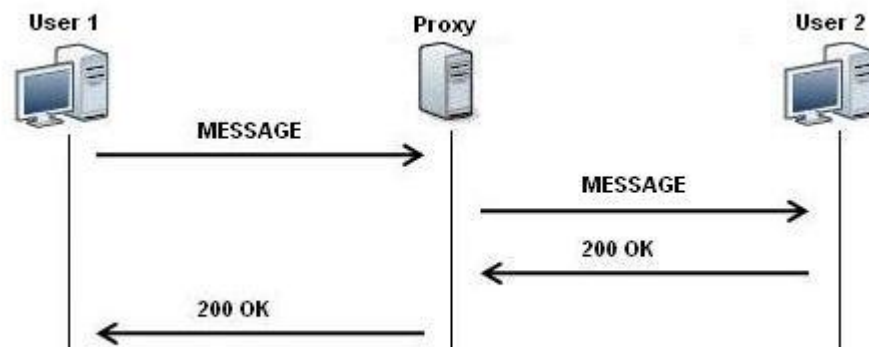


Figura 5. 24 - Intercambio de mensajería instantánea

La comunicación entre participantes sucede en las sesiones de *Media* no en los *Request* de SIP, el método mensaje cambia esta concepción permitiendo que éste pueda llevar la comunicación actual entre los participantes como payload del paquete en sí. Esto implica que los *Request* conformados por el método MESSAGE contengan un requerimiento en materia de seguridad superior a otros *Request* de SIP, en concreto, cualquier *UA* que soporte el método MESSAGE debe implementar una autenticación punta a punta, preservar la integridad del cuerpo del mensaje y la confidencialidad del cuerpo del mensaje.

Dentro de los tipos de autenticación se mantiene la realizada por parte del *Proxy*, siendo útil para verificar la identidad del originador y prevenir el *spoofing* y *spamming* en la red del originador. El mecanismo correspondiente a las SIP *URI* permite a la *UA* que cada salto se efectúe sobre una conexión segura logrando un nivel de integridad y protección de privacidad. Sin embargo este tipo de mecanismo requiere que los usuarios confíen en el buen comportamiento de los *Proxy* ubicados en el camino de la transmisión, es decir, que ninguno de ellos viole las reglas establecidas para enrutar las SIP *URI*.

La posibilidad de que un *Proxy* realice retransmisiones a varios destinatarios permite que *Request* de MESSAGE puedan ser entregados a destinatarios adicionales sin el conocimiento por parte de la *UAC*, provocando que al utilizar protección de salto a salto, los usuarios deben confiar en todos los *Proxy* de la cadena exigiendo que no se retransmitan *Requests* a destinos no autorizados.

Para prevenir la reiteración de *Request* de SIP viejas, todos los MESSAGE debe contener un campo del encabezado con el campo *Date*, cualquier mensaje con

una fecha anterior en la cual se supere por una cantidad considerable de minutos ya sea hacia adelante o hacia atrás, deberá ser respondida con un mensaje de tipo 400 *Incorrect Date or Time*. A menos que este tipo de mensaje llegue de forma reiterada proveniente del mismo origen indicando posiblemente un descarte sin envío de respuesta.

5.6.3 – Implementación en el *Core* de Señalización

Al habilitar el módulo que esta implementado en el *Core* de funcionalidad SIP de *Kamailio*, se habilita la notificación de eventos de SIP. Este modulo maneja los mensajes PUBLISH y SUBSCRIBE y genera los mensajes NOTIFY de forma general e independiente. La particularidad de ser extensible permite que registre eventos de otros módulos de *Kamailio*.

Los *event package* de SIP soportados son *presence*, *presence.wininfo*, *dialog;sla* del modulo *presence_xml* y *message-summary* del modulo *presence_MWI*.

Este modulo utiliza almacenamiento en base de datos, además de realizar cache de memoria para mejorar la performance del sistema. En el dialogo establecido por los métodos SUBSCRIBE, la información provista por este es guardada en memoria y es periódicamente actualizada en la base de datos, mientras que para los mensajes de tipo PUBLISH solo se guardan las informaciones de presencia o ausencia de ciertos recursos para evitar procesos costosos a nivel operativo en la base de datos.

Existe la posibilidad de deshabilitar el cache en memoria configurando un *fallback* al modo base de datos, en este modo, en caso de buscar un registro que no se encuentra en cache la búsqueda se continua en la base de datos. Es útil para arquitecturas donde el procesamiento y la memoria pueden ser fragmentados en diferentes instancias de *Kamailio*, posiblemente utilizando diferentes servidores para una misma base de datos. Una de las particularidades en especial es mantener tres modos de base de datos, uno que puede se configurado a través de un parámetro *db_mode*, fijando los modos 0, 1 y 2, siendo respectivamente que los suscriptores sean obtenidos solo de la memoria, solo de la memoria y haciendo *fallback* a la base de datos en caso de no encontrar los registros o solo efectuando consultas a la base de datos [70].

El funcionamiento básico parte de dos usuarios registrados en el sistema que soportan en ambas sesiones el servicio de presencia y mensajería instantánea. Además de soportarlo deben pertenecer al grupo que soporte suscribir al

servicio de presencia, de no pertenecer al grupo, o sea no contar con los permisos necesarios para recibir notificaciones de presencia, estos no serán aceptados ni recibirán notificación alguna.

El usuario que pretende recibir información de presencia de otro usuario efectúa un mensaje SUSCRIBE al servidor de presencia el cual en el campo *Event* detalla el valor *presence*. Es en este campo que se hace necesario detallar cual es el motivo de la suscripción, concretamente, recibir los eventos de presencia. Todo usuario que pretenda recibir los eventos de presencia enviará este mensaje al servidor para que pueda suscribirlo al servicio.

En respuesta al SUSCRIBE, zanjando que se efectuó la autenticación debida de credenciales, el servidor responde con un 202 OK garantizando la suscripción exitosa. Ya que es un cambio de estado por parte del suscriptor, el servidor envía un mensaje de NOTIFY indicando el estado de la suscripción, la validez temporal y si esta activa o no. En respuesta a este NOTIFY enviado por el servidor, se responde por parte del suscriptor un 200 OK confirmando el envío de información de actualización de estado.

El usuario comienza a escribir un mensaje la aplicación de mensajería instantánea, al estar suscripto a un servicio de presencia, se envían MESSAGE con información de estado de estar escribiendo un mensaje, esta información es enviada de forma directa al servidor de presencia que a su vez lo envía a la otra punta de la comunicación, desplegando en la otra aplicación que el usuario esta editando el mensaje a enviar. Estos mensajes son aceptados por medio de un 200 OK.

Finalizada la introducción de texto es enviado hacia la otra punta con un payload que se compone del texto ingresado previamente, este paquete al llegar al servidor de presencia es reenviado a la punta a la cual se quiere informar. Nuevamente, la aceptación de los mensajes enviados es por medio de un 200 OK.

5.7 – Grupos y Permisos

5.7.1 – Descripción de la Funcionalidad

El sistema ofrece la posibilidad de manejar diferentes perfiles de usuario (grupos) entre los cuales se encuentran:

- Directorio
- Gerentes
- Jefes
- Empleados

Al momento de la creación de un nuevo usuario, como mínimo se debe elegir nombre de usuario y contraseña para que este se pueda *Registrar* en el sistema.

Posterior a la creación, si el sistema tiene manejo de grupos encendido, se debe asignar al usuario a uno de los grupos anteriores.

El manejo de permisos de llamadas básicas del sistema consiste en lógica agregada en el archivo de configuración del SIP *Proxy*. Esta permite o no realizar llamadas en función de los grupos que a los pertenecen los *User Agent* originador y destinatario del INVITE inicial.

La lógica construida para Obelix establece que los usuarios que pertenecen a un grupo solo pueden llamar a los usuarios de su mismo grupo, a los del grupo inmediatamente “superior” y a todos los usuarios que pertenezcan a grupos que se encuentren por “debajo”. Los términos “superior” y por “debajo” se enmarcan en el contexto clásico de un organigrama empresarial.

Por lo tanto a modo de ejemplo un usuario que pertenezca al grupo Directorio puede llamar a cualquier usuario registrado en Obelix. Un empleado solo puede llamar a otros empleados y a usuarios que pertenezcan al grupo Jefes. Un jefe puede llamar a los empleados, otros jefes y gerentes, pero no a los usuarios que pertenezcan al grupo Directorio.

Si por algún error se hace pertenecer a un mismo usuario a 2 (o más) grupos de los presentados anteriormente, la lógica entenderá que pertenece al más restrictivo. Un usuario que no tenga permisos para llamar a otro recibirá por parte del *Proxy* el mensaje “403 Forbidden Destiny”. De lo contrario el dialogo continua en forma transparente.

El sistema cuenta además con grupos extras que habilitan a un usuario a acceder a las diferentes *features* del sistema. Los grupos extras son:

- *MoH*
- *Voicemail*
- *Conf*
- *Conf-creator*
- Presencia

Si un usuario pertenece al grupo “*MoH*” entonces podrá colocar a otro en espera con *Music On Hold*. El grupo *Voicemail* implica que el usuario que pertenezca a él pueda ser llamado a su casilla de *Voicemail* bajo los casos de uso para la cual fue creada la *feature*.

El grupo Presencia permite a un usuario subscribirse a notificaciones de presencia de otros usuarios. Es independiente de la mensajería instantánea (IM) ya que ambos *features* son implementados con diferente mensajería SIP, el primero con SUBSCRIBE y NOTIFY, el segundo con mensajes del tipo MESSAGE.

Se deja abierto el manejo de permisos para la *feature* IM, en principio un usuario puede enviar mensajería instantánea a cualquier otro, independientemente si esté o no suscripto a notificaciones de presencia.

Los grupos *Conf* y *Conf-creator* definen los permisos de un usuario para con las conferencias. Si un usuario pertenece al grupo *Conf* el mismo se puede unir siempre a conferencias ya previamente establecidas por un creador, pero no tiene permisos de crear una nueva. Por el contrario, el grupo *Conf-creator* define a los usuarios con permisos de creación de nuevas salas de conferencias, todo usuario que se pertenece al grupo *Conf-creator* también pertenece al grupo *Conf*.

5.7.2 – Implementación en Core de Señalización

Como se menciona en el subcapítulo anterior, el manejo de permisos es íntegramente implementado por el SIP *Proxy*/SIP *Registrar* (*Kamailio*), mediante lógica en su archivo de configuración. En función de los grupos que pertenecen los *User Agents* origen y destino de los mensajes, se implementa la autorización de la transacción.

Si un usuario (*Holder*) tiene permisos de *MoH* entonces el INVITE generado por el *Proxy* hacia *MoH@MoH.obelix* con el *SDP* del *User Agent Holdee* de la figura 5.8 será efectivamente creado y enviado al servidor de Music On Hold. Si el usuario *Holder* no se encuentra en el grupo “*MoH*”, entonces el tratamiento que le dará el *Proxy* al INVITE con send only será el estándar, logrando de esta manera un escenario de “*Call Hold*” estándar (sin música) según la RFC-5359 página 6.

El hecho que un usuario no tenga permisos de *Voicemail* y además no se encuentre registrado, genera que ante una llamada que lo tenga a él como destinatario que el *Proxy* responda en forma estándar con “404 Not Found” al originador de la llamada. Es decir, no se redireccionará el INVITE hacia el servidor de *Voicemail*. Un comportamiento similar se da si un usuario que no cuenta con permisos de *Voicemail* es llamado y este en lugar de atender la llamada usa la función BUSY de su *softphone*, el comportamiento del *Proxy* consiste en hacer llegar dicha response “486 Busy Here” al originador de la llamada. Si por el contrario el usuario se encuentra en el grupo de *Voicemail* el comportamiento del sistema es exactamente el ilustrado en la figura 5.13.

Para el caso de las conferencias, si el destino de la llamada pertenece a alguna conferencia (internos 3000 al 3019), se chequeará si el usuario originador pertenece al grupo *Conf* y posteriormente se chequeará si además cuenta con permisos de *Conf-creator*. Dependiendo si pertenece o no al grupo *Conf-creator* el *Proxy* modifica el campo usuario de la *URI* del INVITE, agregando un carácter distintivo a cada caso, para posteriormente enviar este INVITE hacia el servidor de conferencias (*Conference.obelix*).

El primer filtro se encuentra en el *Proxy* ya que si el usuario no tiene permisos básicos de conferencias, el INVITE al *Conference Server* nunca se generará (el usuario recibirá un mensaje “403 Forbidden Call” por parte del *Proxy*).

En la *PBX* encargada de implementar las conferencias se terminan de procesar los permisos. En base al carácter extra enviado en el campo de usuario de la *URI*, si el usuario cuenta con permisos de creador puede unirse o crear una nueva conferencia (dependiendo si la sala que desea entrar ya se encuentra en uso o

no). Si el usuario no cuenta con permisos de creador se chequeará si la conferencia a la cual desea ingresar ya se encuentra en uso, si es así el usuario puede ingresar a la misma, en caso contrario se le responde con un mensaje de “408 Request Timeout”.

Por último, el manejo de permisos en las llamadas básicas se procesa enteramente en el *Proxy*. Cuando arriba un INVITE con el objetivo de iniciar un nuevo diálogo, el *Proxy* chequea si el usuario originador del método pertenece a un grupo inhabilitado de poder comunicarse con usuarios del grupo al cual pertenece el destinatario de la llamada. Si esto se cumple el SIP *Proxy* responde al INVITE con un “403 Forbidden destiny”. En caso contrario el sistema permite realizar la llamada.

Más detalles de esta implementación de pueden obtener en Anexo 1

5.8 – Accounting

5.8.1 – Descripción del Servicio

El Core de señalización, en particular el SIP *Proxy Kamailio*, será el encargado de realizar el *accounting* de las transacciones SIP que hagan uso del mismo.

El *accounting* se realiza de acuerdo a eventos de interés. Las transacciones consideradas de interés deben ser “marcadas” con banderas, las cuales indicarán al módulo de *accounting* del *Proxy* que se deberá guardar registro de la transacción en la base de datos al finalizar la misma.

Este registro consiste básicamente en los datos fundamentales de la transacción:

- Nombre del Método SIP
- *From Tag*
- *To Tag*
- *Call-ID*
- Último código de response
- Frase de última response
- *Timestamp*

Es posible agregar a este registro cualquier tipo de *Header* extra que sea de interés. Se configuran entonces los siguientes campos para tener un *accounting* de mayor utilidad:

- Usuario del *Header From*

- Dominio del *Header From*
- *IP* de origen
- Usuario del *Header To*
- Usuario de *Request URI*
- Dominio de *Request URI*

Las transacciones consideradas de interés son las originadas por los siguientes mensajes:

- INVITE: Destinados al dominio @obelix y por ende deben ser encaminados con los datos recabados por el SIP *Registrar* (haciendo uso de la tabla de "*location*"). También se incluyen los que pertenecen a otros dominios, por ejemplo los generados por el *Proxy* para la funcionalidad de *Music On Hold* (dirigidos hacia @MoH.obelix). Para ambos INVITE interesa tener el *accounting* de cuando la transacción finaliza exitosamente o la llamada es perdida.
- BYE: mensajes esenciales a la hora de realizar Billing. Mediante la diferencia de timestamps entre el INVITE inicial y un BYE de la misma *Call-ID* se puede calcular la duración de la llamada. Es de vital importancia para poder capturar los mensajes BYE que el SIP *Proxy* sea "*Stateful*".

La razón principal de porque normalmente se utiliza un SIP *Proxy "stateful"* es si este debe hacer realizar *accounting*. Pero como se justificó en subcapítulos anteriores, la propiedad *stateful* fue imprescindible para la implementación de la integración del *Core* de señalización con los servidores de música en espera y *Voicemail*.

La base de datos donde se aloja el *accounting* puede ser luego consultada por un *software* que en base a este, efectué el Billing de cada transacción. Esto último quedando por fuera de los objetivos de este proyecto.

Capítulo 6 – Alta Disponibilidad del Núcleo de Señalización

En este capítulo se buscará brindarle alta disponibilidad a la arquitectura del núcleo de señalización presentado al término del capítulo 4. Se plantearán distintas soluciones, así como sus ventajas y desventajas, y se planteará una arquitectura final posible en la que se logra tener alta disponibilidad en el sistema.

6.1 – SIP Proxy HA

Comenzaremos por el corazón del sistema, el SIP Proxy. Este elemento es uno de los más críticos a la hora de asegurar disponibilidad en el sistema. Si este servidor fallare, el sistema completo se encontraría inaccesible. De él depende el manejo de toda la señalización asociada a todos los servicios brindados por el sistema, así que es un buen punto para comenzar a construir un sistema de alta disponibilidad.

6.1.1 – Esquemas de Alta Disponibilidad

Dada la comentada importancia de este servidor, es natural pensar en colocar un segundo servidor de similares características capaz de relevar al primero en caso de falla. La figura 6.1 ilustra esta arquitectura.

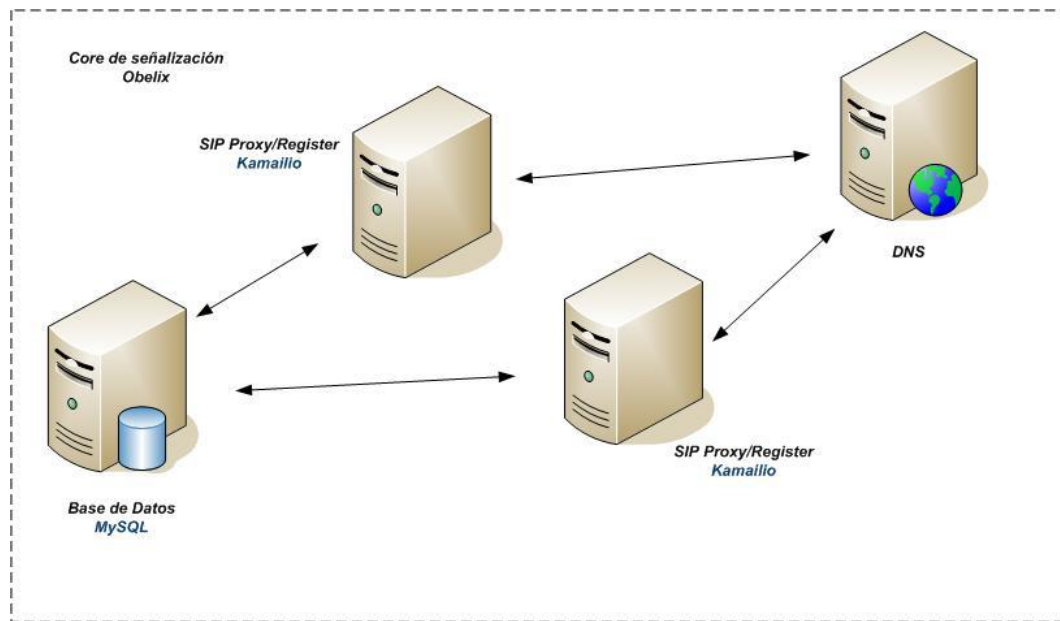


Figura 6. 1 – Redundancia Sip Proxy

Lo que resta es determinar cómo va a ser el modo de trabajo de estos dos servidores (Sip Proxy).

Las posibilidades son que actúen en un esquema activo-pasivo o activo-activo. En un esquema activo-pasivo, la carga del sistema es soportado por uno de los servidores mientras que el otro se encuentra a la espera que el servidor activo caiga para tomar su lugar. En un esquema activo-activo, ambos servidores comparten la carga del sistema y si uno de ellos cae, el otro debe ser capaz de soportar la carga que deja de atender el servidor que cae.

Ambos esquemas tienen implementaciones y consideraciones bien distintas. En el primero de los casos hay que implementar algún método de *failover*, en donde el servidor pasivo (u otro elemento) detecte la caída del activo y el pasivo tome su lugar. Este proceso es deseable sea transparente para el resto del sistema, en particular para los agentes que hacen uso del mismo. En el esquema activo-activo hay que implementar algún método de balanceo de carga entre ambos servidores, método el cual ante la caída de uno de los servidores debe dirigir todo el tráfico hacia el servidor que aun se encuentra activo. También este proceso es deseable sea transparente para los usuarios del sistema.

Previamente a la elección de uno de estos esquemas posibles hay que hacer un acercamiento al funcionamiento de Kamailio, en particular su relación con la base de datos en su modo de SIP Registrar.

Kamailio (v.3.2) cuenta con 4 modos de trabajo configurables es su condición de SIP Registrar, servicio exportado por el modulo USRLOC⁸.

El primer modo, o modo cero, no utiliza la base de datos para guardar los datos de los usuarios registrados. Toda la información de contacto de los usuarios registrados, en particular la dirección IP del mismo, se mantiene solo en memoria. En este modo, la caída del servidor implicaría la pérdida de todos los registros. Para asegurar la persistencia de estos datos están los siguientes modos.

En el segundo modo, o modo 1, se escribe inmediatamente en la base de datos los datos de registros de los usuarios. Se puede decir que se hace un respaldo de estos datos en cuanto los registros se concretan. Se siguen manteniendo también en memoria.

El tercer modo, o modo 2, es similar al modo 1 pero la diferencia radica en que la escritura en la base de datos se hace cada cierto tiempo configurable. Las escrituras en la base de datos, que suelen ser pesadas, se realizan solo cuando se vence el tiempo configurado y no todo el tiempo. Esto es útil si se tienen picos en el volumen de registros en cierto momento y se requiere rápido procesamiento.

Cabe aclarar que en los dos modos anteriores, donde se usa la base de datos como respaldo de los datos de los usuarios registrados, en el único momento que se leen de la base estos datos es en el arranque del sistema. Luego se asume que los datos en la base y en memoria son siempre los mismos. Si por algún motivo otro agente alterara los datos de la base, no serían divisados por el servidor hasta que se arranque de nuevo. Estos dos modos están pensados para que un único servidor sea el que efectúa los registros.

El cuarto modo, o modo 3, usa solamente la base de datos. Ninguna información es guardada en memoria. Los datos se escriben en, y se leen de, la base de datos. Este es el modo que peor rendimiento tiene en cuanto a procesamiento de señalización, ya que en los registros escribe en la base de datos (igual que el modo 1), y en las llamadas además de leer de la base las credenciales de quien llama, se tiene que ir a buscar los datos del llamado, algo que no ocurría en ningún otro de los modos.

En la arquitectura presentada al final del capítulo 4, así como en todas las pruebas presentadas en dicho capítulo, el modo de funcionamiento era el tercero (modo 2), ya que se entendió de importancia la persistencia de los datos de registros y se pensó conveniente que los datos fueran respaldados cada cierto intervalo de tiempo y no a medida que llegaran.

Esta introducción a los comentados modos tiene su importancia por lo siguiente, y es que el esquema activo-activo solo puede funcionar correctamente en el modo 3. Este

⁸ La API del modulo USRLOC se puede consultar en http://www.kamailio.org/docs/modules/3.2.x/modules_k/usrloc.html

modo, como ya se aclaró, es el que peor rendimiento tiene dada la cantidad de consultas realizadas a la base de datos. La figura 6.2 ilustra (en un diagrama simplificado) el comportamiento del esquema activo-activo en todos los modos que no leen de la base los datos de ubicación del usuario llamado, y se verá porque podrían darse escenarios de falla.

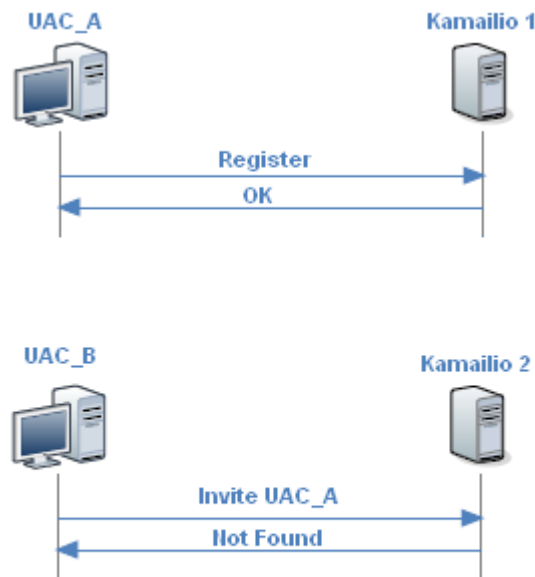


Figura 6. 2 – Caso de falla “modo 2”

El usuario A se registra correctamente en alguno de los servidores, en este caso el servidor 1. Cuando el usuario B llama al usuario A, si esta llamada llega a otro servidor que no sea el 1, en este caso el servidor 2, este ultimo consulta su memoria en búsqueda del usuario A y al no encontrarlo asume que no se encuentra registrado en el sistema por lo que la llamada no se cursa.

La figura 6.3 a continuación ilustra el funcionamiento en modo 3.

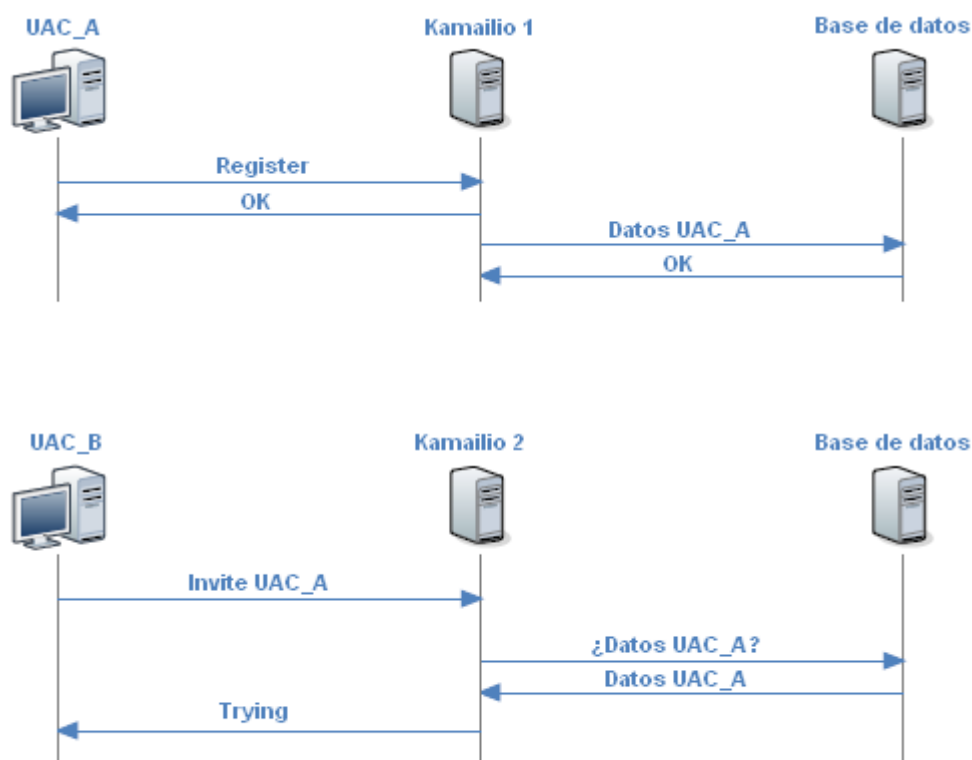


Figura 6. 3 – Funcionamiento “modo 3”

En este escenario cualquiera sea el servidor donde el usuario A se registra, los datos se guardan en la base de datos, siendo accesibles para cuando el usuario B llame al usuario A a través de otro servidor.

Se presenta a continuación los resultados de una prueba de registros sobre un servidor Kamailio funcionando en modo 3, y se compararan con los resultados obtenidos en una prueba con los mismos equipos presentada en el capítulo 4, donde el servidor funcionaba en modo 2. Como cambiar el modo de funcionamiento se puede consultar en el anexo 1. Las ilustraciones de la prueba se encuentran en las figuras 6.4 y 6.5.



Figura 6. 4 – Esquema 1 de la prueba de registros en “modo 3”



Figura 6. 5 – Esquema 2 de la prueba de registros en “modo 3”

Los resultados son los siguientes:

Se pudieron generar 7000 escenarios de registro por segundo de manera continua sin retransmisión de ningún mensaje. Más allá de este valor, comenzaban a haber algunas retransmisiones.

Se muestra en las figuras 6.6 y 6.7 datos de procesador y ancho de banda del SIP Proxy, y datos de anchos de banda de la base de datos, respectivamente.

top - 02:22:51 up 2:22, 2 users, load average: 0.00, 0.00, 0.00												117725.8	79559.46
Tasks: 209 total, 3 running, 206 sleeping, 0 stopped, 0 zombie												138878.8	91556.02
Cpu(s): 41.5%us, 11.1%sy, 0.0%ni, 40.4%id, 0.0%wa, 0.0%hi, 7.1%si, 0.0%st												142431.9	94166.83
Mem: 2062936k total, 409348k used, 1653588k free, 26748k buffers												141794.8	94313.70
Swap: 6258680k total, 0k used, 6258680k free, 272480k cached												140557.3	93414.57
												141379.2	93976.12
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND	140445.4	93635.91
6514	root	20	0	1040m	4612	3444	R	16.6	0.2	1:49.35	kamailio	140293.8	93547.05
6499	root	20	0	1040m	4612	3444	S	16.3	0.2	1:49.32	kamailio	140703.0	93939.73
6512	root	20	0	1040m	4612	3444	S	15.9	0.2	1:49.55	kamailio	141001.4	94575.76
6501	root	20	0	1040m	4612	3444	S	15.6	0.2	1:49.44	kamailio	139405.2	93396.92
6503	root	20	0	1040m	4612	3444	S	15.6	0.2	1:49.04	kamailio	138413.1	92681.26
6511	root	20	0	1040m	4612	3444	S	15.6	0.2	1:49.66	kamailio	140019.2	93720.16
6513	root	20	0	1040m	4612	3444	S	15.6	0.2	1:49.20	kamailio	139968.9	93659.17
6504	root	20	0	1040m	4612	3444	S	15.3	0.2	1:49.33	kamailio	140040.2	93975.62
6505	root	20	0	1040m	4612	3444	S	15.3	0.2	1:49.49	kamailio	139974.1	94041.92
6509	root	20	0	1040m	4612	3444	S	15.3	0.2	1:49.18	kamailio	139184.1	93468.59
6510	root	20	0	1040m	4612	3444	S	15.3	0.2	1:49.72	kamailio	eth0	
6502	root	20	0	1040m	4612	3444	S	14.9	0.2	1:49.97	kamailio	Kbps in Kbps out	
6506	root	20	0	1040m	4612	3444	S	14.9	0.2	1:49.45	kamailio	139847.0	94040.89
6508	root	20	0	1040m	4612	3444	S	14.9	0.2	1:49.35	kamailio	139714.8	94262.97
6500	root	20	0	1040m	4612	3444	R	14.6	0.2	1:49.89	kamailio	138513.2	93358.27
6507	root	20	0	1040m	4612	3444	S	14.3	0.2	1:49.34	kamailio	138905.4	93764.52
6750	root	20	0	2676	1176	872	R	0.7	0.1	0:00.64	top	138778.8	93629.65
1365	root	20	0	2856	632	516	S	0.3	0.0	0:13.68	lldpad	138619.1	93635.62
1	root	20	0	2856	1404	1192	S	0.0	0.1	0:01.25	init	139372.7	94278.90
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd	138673.4	94112.01

Figura 6. 6 – Captura del comando “top” y de uso de ancho de banda en SIP Proxy

40007.18	84498.33
40159.48	84808.58
39774.55	84000.67
39602.72	83680.67
39986.41	84445.49
39781.18	84013.06
39715.72	83894.56
39427.05	83273.84
40117.86	84733.05
39625.11	83681.21
eth0	
Kbps in	Kbps out
40044.11	84561.71
39757.97	83984.92
39799.80	84074.58
39783.52	84031.80
37198.67	78560.88
34512.29	72900.34
37872.03	79901.97
39848.85	84217.97
39805.66	84131.68
40031.11	84508.00

Figura 6. 7 - Captura del uso de ancho de banda en la base de datos

La tabla 6.1 a continuación resume los resultados del máximo número de escenarios de registros por segundo (sin retransmisiones de mensajes) en ambos modos testeados.

	Modo 2	Modo 3
Escenarios de registro/seg.	30.000	7.000

Tabla 6. 1 – Tabla comparativa de prueba de registros en modos 2 y 3

La tabla 6.2 a continuación muestra los anchos de banda entrante y saliente (normalizados por cada 1000 escenarios por segundo) de la base de datos para ambos modos.

@1000 registros/segundo	BW IN (kbps)	BW OUT (kbps)
Modo 2	933	1266
Modo 3	5714	12071

Tabla 6. 2 – Tabla comparativa de anchos de banda en las pruebas de registros en modo 2 y 3

Se puede ver que el uso del modo 3 tiene como contrapartida una menor capacidad de procesamiento de señalización. El tráfico desde y hacia la base de datos se incrementa

en el modo 3 debido a que los datos del usuario registrado en el sistema se escriben en dicha base. Esta nueva interacción con la base de datos, que incluye una escritura, es una transacción costosa a nivel de procesamiento dado el salto en la cantidad de tráfico intercambiado entre estos servidores.

La cantidad de señalización capaz de procesar el SIP Proxy en este nuevo modo, en el caso de escenarios de registro, es notablemente menor a cuando trabaja en modo 2 (poco mas de 4 veces menor).

Aunque el rendimiento del sistema en modo 3 sea sensiblemente menor al obtenido en modo 2, es de destacar que aun en estas condiciones el volumen de peticiones de registro que es capaz de manejar esta herramienta es muy superior, incluso para los altos requerimientos de este proyecto.

Estos resultados son en cuanto a escenarios de registro, que es solo una de las funciones que se ven afectadas con un cambio de modo. La otra función, y más importante aún, es la capacidad de procesamiento de llamadas.

Es de esperar que la capacidad para procesar llamadas del SIP Proxy se vea afectada negativamente si se trabaja en modo 3, en comparativa con el modo 2. Esto se debe a que cuando un agente llama a otro, se debe realizar una consulta extra a la base de datos de la ubicación del agente llamado. Si bien esta operación es menos pesada a nivel de procesamiento que una escritura (como el caso de los registros), va a impactar negativamente en el rendimiento.

Se presenta a continuación los resultados de una prueba de establecimientos de llamadas sobre un servidor Kamailio funcionando en modo 3, y se compararan con los resultados obtenidos en una prueba con los mismos equipos presentada en el capítulo 4, donde el servidor funcionaba en modo 2. El archivo de configuración de la prueba es el *standard* con la salvedad de la configuración específica añadida para el manejo de una base de datos externa al SIP Proxy. Las ilustraciones de la prueba se encuentran en las figuras 6.8 y 6.9.



Figura 6. 8 – Esquema 1 de la prueba de llamadas en “modo 3”

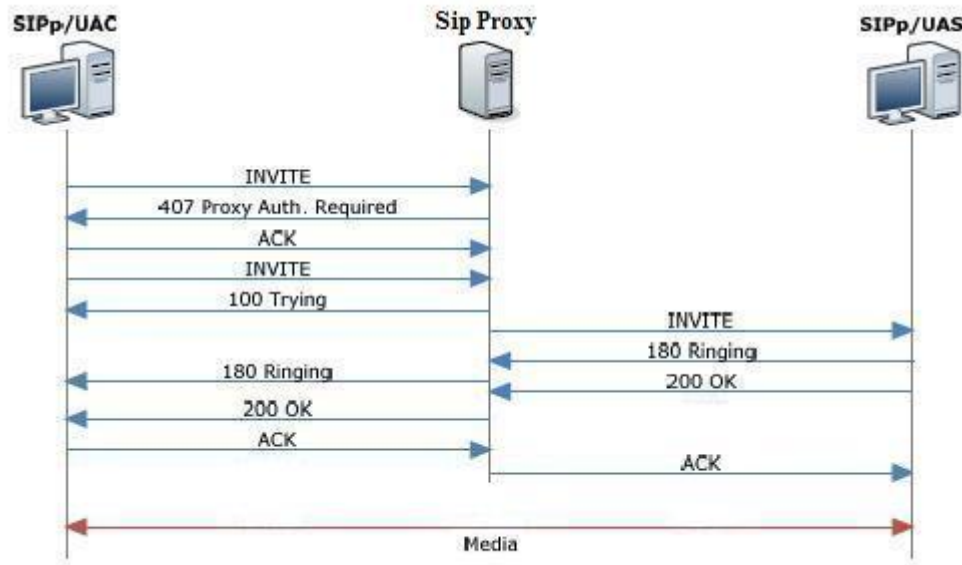


Figura 6. 9 – Esquema 2 de la prueba de llamadas en “modo 3”

Los resultados son los siguientes:

Se pudieron generar 3000 escenarios de llamadas por segundo de manera continua sin retransmisión de ningún mensaje. Más allá de este valor, comenzaban a haber algunas retransmisiones.

La tabla 6.3 resume los resultados obtenidos en las pruebas de ambos modos para este escenario.

	Modo 2	Modo 3
Esc. de llamadas/seg.	6.000	3.000

Tabla 6. 3 – Tabla comparativa de prueba de llamadas en modos 2 y 3

Si bien la relación de performance es mejor que para el caso de los registros, es evidente la caída en el rendimiento a causa de esta nueva interacción con el servidor de la base de datos.

Por último, cabe analizar lo que sucede ante la caída de un SIP Proxy en este núcleo de señalización. Si hay un único servidor cursando N llamadas, al salir de servicio ya no se podrán cursar nuevas llamadas, pero sin embargo las N llamadas que están en curso no se pierden. Dado que el servidor que cae no maneja ningún tipo de flujo de media, las llamadas cursantes no se ven afectadas. Lo que se pierde ante un caso como este es el estado de esas N llamadas, y por lo tanto no se va a poder mantener registro alguno de

las mismas. Tampoco esas N conversaciones podrán evolucionar en ningún sentido, entiéndase renegociación de los parámetros de sesión (códecs, puertos, etc.), transferencias, música en espera o cualquier otra evolución de la llamada que implique señalización.

Presentados estos conceptos, ahora se está en condiciones de evaluar los posibles esquemas (activo-pasivo, activo-activo) para el núcleo de señalización.

En un esquema activo-pasivo, no es necesario el uso del modo 3 para interactuar con la base de datos. Esto es porque es un único servidor el que está interactuando con la base de datos concurrentemente y (si trabaja en modo 1 o 2) las entradas con los registros de usuarios se encontrarán tanto en la base de datos como en la memoria del mismo.

Ante la caída de este servidor activo, el pasivo entraría en funcionamiento trayendo en ese momento los datos de la base de datos hacia su memoria. De esta manera, los resultados para este esquema son los mismos que los detallados en la arquitectura del final del capítulo 4.

Este último es un gran punto a favor del esquema activo-pasivo. Sin embargo también existen puntos en contra.

Si existiese N llamadas cursándose en el sistema, la caída del servidor activo implica la pérdida del estado del total de las N llamadas, con las comentadas consecuencias.

Teniendo un servidor pasivo se tiene hardware ocioso, que si debe soportar la carga del total del sistema ante la caída del activo, debe ser por lo menos igual de bueno.

Si bien se vio que no se necesita un hardware extremadamente caro para poder soportar un volumen interesante de señalización, la solución activo-pasivo tiene su punto más débil a entender de esta investigación en su incapacidad de escalar, en el sentido de la posibilidad de agregar un tercer servidor.

En un esquema activo-activo, es mandatorio el uso del modo 3 para un correcto funcionamiento, por lo que los resultados obtenidos por servidor son los presentados en este capítulo para dicho modo. Parece un precio alto a pagar por el uso de este esquema, y por lo tanto es el punto más flojo que brinda.

No obstante, si se colocan dos servidores en un esquema activo-activo (ambos funcionando en modo 3), aunque la capacidad de establecimiento de llamadas por segundo de cada uno sea la mitad que en su funcionamiento en modo 2, en la totalidad el sistema obtiene el doble de la capacidad de cada uno y de esta manera se vuelve a los resultados de la arquitectura al final del capítulo 4, al igual que el esquema activo-pasivo.

Si se tienen dos servidores en modo activo-activo con un balanceo adecuado, y se están cursando N llamadas por el sistema, la caída de uno de estos conllevaría la pérdida del estado de la mitad de las N llamadas.

Este esquema no tiene hardware ocioso, aunque es conveniente que se estén usando al 50% o 60% (en el caso de dos servidores) para que ante una falla de uno de ellos el restante soporte la carga del sistema entero.

Quizá el punto más destacable, y por el que valdría la pena pagar el precio de una menor capacidad de procesamiento de los servidores, es que este esquema se puede adaptar sin ningún tipo de problema a n servidores. Esto tiene una implicancia más que interesante dado que n servidores con una capacidad muy limitada podrían resultar igual de efectivos que un solo servidor potente.

En la siguiente tabla (6.4) se resume todo lo anteriormente comentado.

	Activo-Pasivo	Activo-Activo
Modo de funcionamiento	Modo 2 (o Modo 1)	Modo 3
Perdida de llamadas (en N)	N	N/n^*
Hardware ocioso	Si	No
Escalabilidad	No	Si

Tabla 6. 4 – Tabla comparativa de esquemas Activo-Pasivo y Activo-Activo

*** n es el número de servidores en este modo**

Luego de este análisis de estos esquemas, se opta por utilizar el modo activo-activo para los servidores del núcleo de señalización con el fin de ganar escalabilidad, aunque sea a costas de una baja en la capacidad de procesar señalización de cada servidor por separado. Como fue comentado cada uno se ve afectado individualmente en su performance, compromiso que se ve atenuado dado que el rendimiento final del sistema corresponde al aporte de todos.

Conociendo ya el esquema a utilizar para dar alta disponibilidad al SIP Proxy, hay que resolver como se va a hacer el reparto de carga, el posible balance del flujo de señalización del sistema y más importante aún, el *failover* (continuidad del servicio ante una caída) del sistema.

6.1.2 – Balanceo de Carga y Failover

Existen diversos mecanismos para lograr balance de carga y failover en este sistema. En esta sección se contemplarán posibles soluciones a este problema.

6.1.2.1 – Balanceo Mediante Hardware Dedicado

La primera alternativa a evaluar es la colocación de una pieza de hardware dedicada a esto frente a los servidores.

Este hardware no puede ser cualquiera ya que debe cumplir, además del balanceo de la señalización que arriba al sistema, algún mecanismo para darse cuenta que un servidor cae y dejar de mandarle señalización.

Existen diversas piezas de hardware que cumplen estos cometidos, pero hay dos motivos importantes por lo que no cualquiera de estas piezas es de la utilidad buscada.

En primer lugar, hay que diferenciar entre dos especies de fallas. Una falla de hardware, o algún otro tipo de falla, que haga caer la interfaz de un servidor y una falla de software, o de servicio, en donde la interfaz del servidor sigue en pie pero ya no se encuentra brindando el servicio que se quiere dar.

El primer tipo de falla, que se notara como falla de capa de red, es muy sencilla de detectar y por ende se pueden tomar acciones al respecto. El segundo tipo de falla, que se notara como falla de capa de aplicación, es más difícil de detectar ya que el equipo que realice el *failover* debe poder detectar la caída de un servicio específico antes de realizar acciones. El primer tipo de falla implica al segundo, pero no así el segundo tipo. Es por eso que un equipo de no sea capaz de detectar una falla de capa de aplicación puede tener comportamientos no deseados en el sistema, pero un equipo capaz de detectar una falla de este tipo podrá también detectar una falla de capa de red.

El otro motivo por el cual no cualquier equipo puede realizar la tarea buscada es que los servidores SIP Proxy son los encargados de mantener el estado de todas las llamadas que son cursadas por el sistema. Esto quiere decir que todos los mensajes de señalización asociados a una misma llamada deben ser dirigidos únicamente al servidor que está manteniendo esa llamada. Si un mensaje correspondiente a una llamada mantenida por el servidor A fuera dirigida al servidor B, este último desconocería como asociar ese mensaje y enviaría un mensaje de error.

Resumiendo, además de balancear el flujo de señalización y detectar la caída de un servidor a nivel de red, un equipo dedicado a lograr balance y *failover* deberá conocer el protocolo SIP y de alguna manera no solo enviar los mensajes al servidor correcto sino que también poder detectar la falla en capa de aplicación.

Para este propósito se decidió probar un software capaz de lograr estos cometidos, el propio Kamailio. Además de su función de SIP Proxy y Registrar con las que se venían trabajando, este software es capaz de brindar lo que se busca.

El modulo DISPATCHER⁹ de esta herramienta exporta funcionalidades muy a medida para estos cometidos. Dada una lista de servidores, este software es capaz de balancear mediante distintos algoritmos toda la señalización hacia los servidores de dicha lista. En particular se puede utilizar un algoritmo de *hash* sobre el parámetro *call-ID* del mensaje SIP. Dicho algoritmo brinda una forma aleatoria de reparto de carga hacia los servidores (esto es dado que el parámetro *call-ID* es aleatoriamente generado por el agente que comienza un flujo de señalización), y además los mensajes con mismo *call-ID* serán entregados al mismo servidor.

La otra funcionalidad de relevancia en este análisis es la capacidad de enviar mensajes SIP cada un tiempo configurable hacia los servidores de la lista, y en caso de estos no responder son removidos de la lista. De esta manera, cuando un servidor tiene una falla a nivel de capa de red o capa de aplicación, no responderá los mensajes SIP y será descartado de la lista del *dispatcher* y no se le entregaran más mensajes. Sin embargo, se le seguirán enviando mensajes SIP para detectar el momento que entre en funcionamiento de nuevo, y así ser nuevamente agregado a la lista.

La figura 6.10 a continuación muestra la arquitectura propuesta.

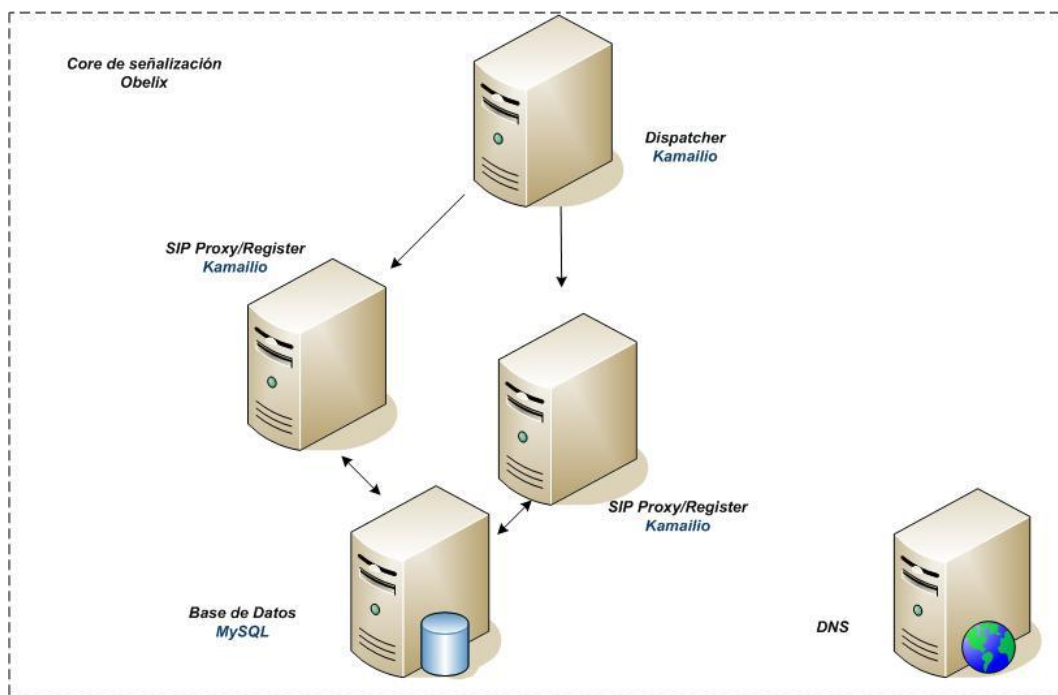


Figura 6. 10 – Esquema de arquitectura con servidor de despacho

⁹ La API del modulo DISPATCHER se puede consultar en http://www.kamailio.org/docs/modules/3.2.x/modules_k/dispatcher.html

En este esquema de trabajo, el servidor DNS resolvería el dominio “Obelix” con la dirección de red del servidor *dispatcher*, siendo este servidor la entrada de la señalización al sistema. A esta configuración se le realizó la siguiente prueba.

Se instaló un Kamailio v3.2 funcionando solamente en modo *dispatcher* en un Notebook HP con procesador Core 2 Duo (2 núcleos) @ 2.0 GHz, 4GB de memoria RAM y una tarjeta de red Fast Ethernet.

Se configuró para que balanceara hacia dos servidores SIP Proxy Kamailio y que les enviara mensajes SIP para probar su disponibilidad cada 200ms. La tolerancia para declarar a un servidor como caído se configuró en un solo mensaje no respondido.

A este servidor *dispatcher* se le enviaron, mediante un generador de tráfico, 1000 mensajes de registro por segundo. A continuación, se desconectó el cable de red de uno de los SIP Proxy para comprobar el tiempo de reacción del servidor *dispatcher*.

El resultado fue más que satisfactorio, dado que ante la desconexión de uno de los servidores SIP Proxy, el generador de tráfico solo tuvo que retransmitir alrededor de 150 mensajes (ningún mensaje se tuvo que retransmitir más de una vez), correspondientes a los mensajes que fueron despachados hacia el servidor desconectado mientras el servidor *dispatcher* detectaba su caída. La tolerancia para retransmitir un mensaje por parte del generador era de 500ms.

De esta manera, se ve que ante una prueba extrema del sistema, dada la cantidad de señalización que se cursaba, así como también el corto tiempo impuesto para la detección del servidor caído, el servidor *dispatcher* cumple muy bien el objetivo buscado de balance y failover para los SIP Proxy.

Sin embargo, aunque esté resuelto el problema planteado de esta manera, otro problema surge al añadir nuevo hardware al núcleo de señalización. Si cayera el servidor encargado del balance, y que es la entrada al núcleo, todo el sistema quedaría fuera de servicio.

Una solución a este problema es el de colocar otro servidor de iguales características para dar redundancia a la entrada del sistema. La figura 6.11 presenta una arquitectura con esta solución.

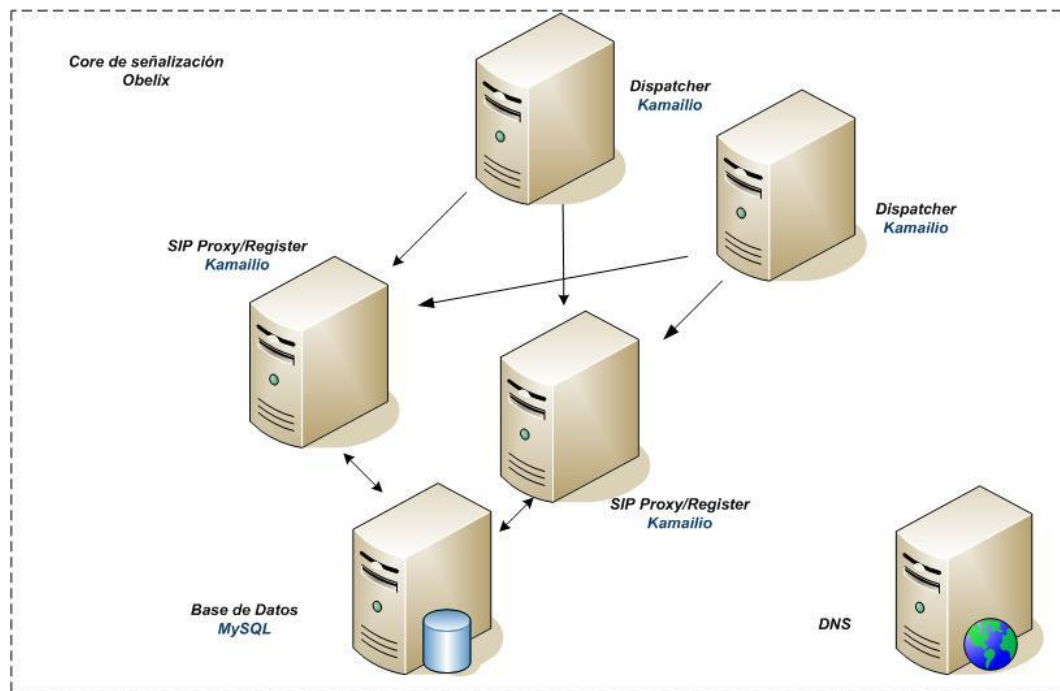


Figura 6. 11 – Esquema de arquitectura con servidor de despacho y alta disponibilidad

No obstante, al igual que para el caso de los Sip Proxy, se requeriría de algún método para garantizar el *failover* de estos nuevos equipos, los cuales deberían cumplir las mismas condiciones comentadas de los propios servidores *dispatcher*. Esto trae como consecuencia una cadena sin fin, ya que para solucionar cada nivel de balanceo se agrega un nuevo *spof* (*single point of failure* – punto único de fallo).

6.1.2.2 – El DNS Como Balanceador

Otro método para lograr un balanceo entre los SIP Proxy del núcleo, es el uso del propio DNS con el que ya cuenta el sistema. Estas ideas están definidas en la RFC 1794 [71].

Un DNS puede implementar un sencillo mecanismo para lograr balancear entre varios servidores que brindan un mismo servicio. Cuando se añaden varias entradas de registros A para un mismo nombre de dominio, el DNS devuelve todas ellas ante una consulta por ese dominio. El agente que solicita la resolución de ese dominio tomará la primera de ellas como respuesta efectiva, y podría eventualmente usar la siguiente en caso que la primera no responda adecuadamente.

De esta manera, si el DNS alterna el orden de las respuestas en las sucesivas resoluciones para un dominio, logra que los agentes apunten a uno u otro servidor.

En el capítulo 4 se propuso el uso del DNS propio de la red donde fuese implementado el resultado de este proyecto, y en el caso que se careciera de uno, la instalación de un DNS es vital para el funcionamiento del sistema.

Si el caso es el de usar un DNS ya existente, hay que asegurarse que soporte este modo de balanceo. Si no soportase tal función, hay que colocar un DNS para el núcleo de señalización. Un software libre ampliamente usado que soporta diversos algoritmos para lograr la función de balanceo es el BIND9, y es tal el que se usó para la resolución de nombres en este proyecto.

Dada la gran cantidad de usuarios a los cuales se pretende brindar el servicio en este proyecto, un algoritmo en donde se devuelven las direcciones de red de los servidores SIP Proxy en un orden aleatorio, resulta en un balance natural.

Por lo tanto, en cuanto a balanceo de la carga de los servidores, este método es efectivo. Sin embargo el *failover* del sistema no está resuelto de manera aceptable.

Trabajando con un DNS capaz de balancear hay dos puntos que conspiran, a la hora de la caída de uno de los servidores SIP Proxy, en contra de un correcto *failover*. El primero es el TTL del registro del servidor caído. Si este TTL es grande, los agentes que hayan recibido este registro como prioritario, o bien sufrirán demoras en toda señalización enviada al sistema (esto es siempre y cuando el agente que no encuentra respuesta del primer servidor, pruebe con el segundo), o bien falle en el intento de conectarse al sistema. Las demoras o el error en la conexión con el sistema perseverarán hasta que, vencido el TTL, se vuelva a consultar el DNS. El segundo punto es que el DNS no tiene como saber que hay un servidor caído, y por lo tanto seguirá propagando la dirección que no se encuentra en línea. Entonces, en el caso de que haya dos servidores a balancear, la mitad de las veces que se consulta al DNS este dará una dirección incorrecta. Esto afecta tanto a los usuarios que están ingresando al sistema como a los usuarios que ya se encontraban en el y se vencieron sus TTL.

En el primero de los puntos, se pueden mejorar los tiempos en que los usuarios no pueden acceder al sistema por tener una entrada incorrecta en su memoria disminuyendo sensiblemente el TTL propagado por el propio DNS. Idealmente, un TTL nulo haría que se consultara siempre el DNS antes de iniciar una transacción contra el núcleo de señalización, pero esto trae como consecuencia una sobrecarga del servidor DNS. Colocar un TTL relativamente chico logra no sobrecargar de sobremanera al servidor de DNS, y mitiga el efecto de los usuarios que no logran conectarse al sistema por mantener en memoria una entrada caída.

El TTL chico no soluciona del todo los problemas en el *failover* del sistema. Idealmente un TTL nulo (que es el mejor caso en cuanto a tiempo de recuperación de los agentes) lograría, en el caso de dos servidores SIP Proxy, que solo la mitad de los agentes se

recuperaran ya que la otra mitad recibiría la entrada caída y sufrirían las demoras de estar intentando conectarse con el servidor caído.

Con el fin de erradicar este problema, se describe en la siguiente sección una herramienta desarrollada con este propósito.

6.1.2.3 – PingX

PingX es un modulo pensado para dar solución al problema de que el DNS siga propagando la dirección de red de un servidor que se encuentra caído.

Tomando alguna de las bondades del modulo *dispatcher* brindado por Kamailio, y añadiendo lógica para tomar acciones específicas ante la caída de un servidor, esta herramienta será la responsable de dar cuenta al servidor DNS de la caída de un servidor SIP Proxy, dejando este así de propagar su dirección.

Este nuevo modulo del sistema funciona “pingueando” con mensajes SIP a los servidores que se quieren monitorear con el fin de detectar una falla a nivel de capa de red o capa de aplicación. Si algún servidor no responde adecuadamente, o directamente deja de responder, el modulo PingX intentara actualizar la información propagada por el DNS.

El modulo *dispatcher* de Kamailio, como ya se vio en este capítulo, brinda la posibilidad de testear a nivel de SIP los servidores de una lista dada. Ante la caída de algún servidor de dicha lista, con la programación de lógica adecuada, se puede tomar alguna acción específica.

El otro concepto utilizado para el desarrollo de PingX es el de DDNS (Dynamic DNS o DNS dinámico). Definido en la RFC-2136 [72], es un método que permite la actualización de la información de un servidor DNS. Dicho método es ampliamente utilizado cuando los servidores, cuyos nombres son resueltos por un DNS, no cuentan con una dirección IP fija. Cada vez que un servidor cambia su dirección de red, por el motivo que fuere, la información en el DNS debe ser actualizada para no tener errores en la resolución.

Con estas dos herramientas, cuando un servidor del núcleo de señalización salga de servicio, el modulo PingX detectara la caída y utilizara el método de DDNS para quitar del DNS la información de ese servidor caído y así evitar la propagación. De la misma manera, cuando el servidor en cuestión vuelva al servicio se volverá a añadir su información al servidor de DNS.

En este caso, la acción tomada ante la caída, o puesta en servicio, de un servidor por parte del modulo *dispatcher* de Kamailio será la ejecución de una herramienta llamada NSUPDATE. Además cabe la posibilidad de ejecutar algún tipo de alarma para dar cuenta de lo acontecido al administrador del sistema.

NSUPDATE es una herramienta de Linux que permite la actualización dinámica de las entradas de un DNS como está definido en la RFC de DDNS. Esta actualización no puede realizarla cualquier agente dado que la comunicación se hace encriptada. Cada servidor capaz de realizar cambios en el DNS está dotado de una clave generada en el propio equipo para realizar la encriptación, y esta clave se exporta al DNS para permitir solo que sea este equipo quien realice modificaciones. Además, el servidor capaz de realizar cambios y el servidor de DNS deben estar con no más de 5 minutos de desfase entre

sus relojes del sistema con el fin de añadir seguridad a la comunicación evitando posibles capturas de paquetes y su posterior inyección.

Toda la configuración y detalles en la implementación de este modulo se pueden consultar en el anexo 1.

La figura 6.12 ilustra el funcionamiento de este nuevo modulo en el sistema.

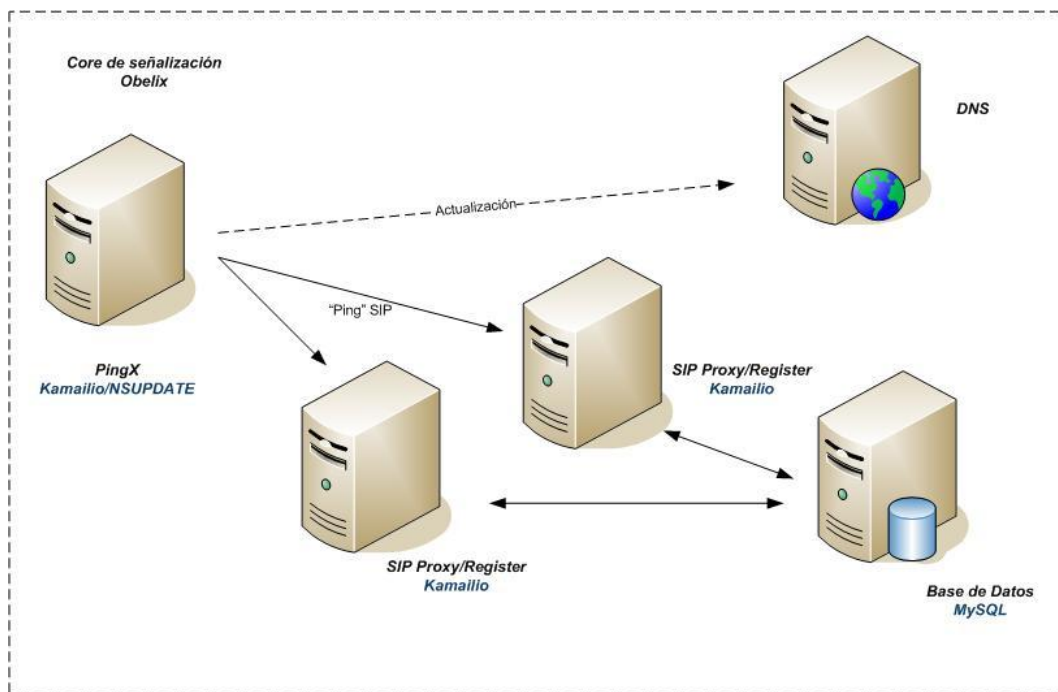


Figura 6. 12 – Esquema de funcionamiento de PingX

6.1.3 – DNS Propio del Núcleo de Señalización

Modificar la información de un servidor de DNS es un tanto delicado, especialmente si se habla de temas de seguridad. Permitir a una aplicación externa modificar estos datos es una práctica que no es bien vista por los encargados de la seguridad y de la administración del DNS.

Si el caso es el que se usa el propio DNS ya existente en la red del cliente, será difícil que permitan el uso de esta solución.

Para lograr la implementación de la solución propuesta, así como asegurarse que el DNS implementa los métodos de balanceo, se propone que el núcleo de señalización cuente con un DNS propio. De esta manera no habría necesidad de manipular el DNS propio del cliente, y se puede asegurar que este nuevo DNS del sistema permita actualizaciones, así como también implemente los métodos de balanceo ya comentados.

El nuevo DNS solo será encargado de la zona correspondiente al sistema propuesto, y únicamente será consultado por los agentes cuando se requiera resolver una dirección correspondiente a este sistema.

Un agente que cuente con un *hardphone*, puede configurar el DNS de la zona correspondiente al sistema a implementar para la resolución de direcciones en el propio teléfono. Esto no trae mayores complicaciones dado que el teléfono no necesitaría resolver otro tipo de direcciones. Si un agente cuenta con un *softphone*, configurar el DNS del equipo host como el DNS del sistema no será del agrado de los administradores. Por este motivo, lo que se plantea es que la única modificación que se deba realizar sobre el DNS propio del cliente sea una entrada estática del tipo NS (y su registro A asociado) que indique, a quien consulte por una dirección correspondiente a la zona del sistema, donde se encuentra el DNS autoritativo de esta zona. De esta manera, se redirige hacia el DNS del sistema únicamente las consultas que son para la zona de trabajo y ninguna otra. Este mecanismo puede o no ser recursivo, siendo una solución no recursiva la que carga menos al DNS del cliente. Si la consulta fuese recursiva, es importante que el DNS del cliente respete los TTL brindados por el DNS del sistema ya que es un parámetro de gran importancia en el tiempo de recuperación del sistema como se comentará más adelante.

El funcionamiento se ilustra en la figura 6.13, donde un agente con un *softphone* consulta por el dominio del núcleo de señalización con el fin de registrarse en el sistema.

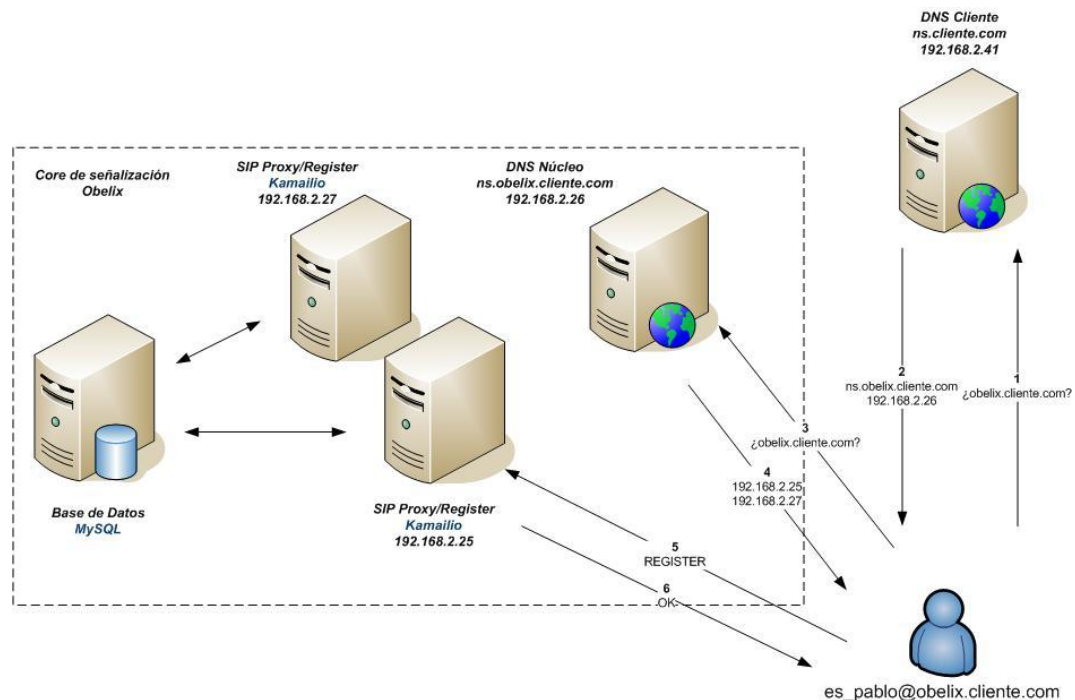


Figura 6. 13 – Esquema de funcionamiento con DNS propio

De esta manera, también se podrían asociar los *hardphones* del sistema al DNS del cliente obteniendo el mismo resultado. No obstante, en caso que haya clientes asociados solamente al DNS del núcleo de señalización, es conveniente que este DNS redirija todas las consultas que no sean dirigidas a su dominio de autoridad.

Con este esquema planteado, en donde el DNS propio del núcleo es capaz de balancear entre todos los SIP Proxy existentes y el PingX se asegura que ante la caída de un servidor su dirección no sea propagada por este DNS, se logra balance y *failover* para los servidores responsables de toda la señalización del sistema.

El tiempo de recuperación del sistema completo va a depender de los TTL manejados por el DNS del núcleo. Solo aquellos agentes que tuviesen la dirección de un servidor como válida al momento de su caída van a sufrir demoras o fallas en el acceso al sistema. Cuanto más chicos sean estos TTL, menor es el impacto, llegando a ser nulo en caso de un TTL igual a cero (ya que siempre se consultaría el DNS antes de comenzar un flujo de señalización y este siempre le devolvería servidores que se encuentran en línea). El TTL elegido tiene como efecto que el caudal de consultas al DNS crezca cuanto más se disminuya este, y viceversa, siendo este un compromiso a tener en cuenta. Si el DNS es capaz de manejar un gran volumen de consultas, más chico podrá ser este TTL y menos tiempo demorará el sistema en recuperarse por completo.

En el anexo 1 se encuentran los detalles para la instalación y configuración del DNS del sistema.

Con una solución para los SIP Proxy del núcleo, aun restan más elementos del núcleo que carecen de un esquema de HA impidiendo un núcleo completo con alta disponibilidad.

6.2 – PingX HA

Habiendo añadido un nuevo módulo con el fin de lograr la alta disponibilidad y el *failover* de los servidores SIP Proxy, se añadió en consecuencia un nuevo punto de falla. Si bien la caída del módulo PingX no afecta el normal funcionamiento del sistema, si este está fuera de servicio cuando cayera uno de los SIP Proxy el sistema carecería de una manera de recuperarse y quedaría en parte con un funcionamiento no deseado.

Para dar solución a esto, se plantea un esquema de dos servidores para esta tarea, en modalidad Maestro-Esclavo. El servidor maestro será el encargado de funcionar como se describió en la sección donde se introdujo esta herramienta, mientras que el esclavo solo monitoreará al servidor maestro. El comportamiento del esclavo ante la caída del

maestro será el de actualizar su lista de servidores a testear (que en un principio solo contiene al maestro) para así agregar los servidores que se encontraba testeando el maestro. De esta manera el servidor esclavo se convierte en maestro y toma su lugar, teniendo la potestad de actualizar el DNS del sistema si así lo requiriera la situación.

Las figuras 6.14 y 6.15 ilustran el funcionamiento de este módulo en un funcionamiento normal y ante la caída del servidor maestro respectivamente.

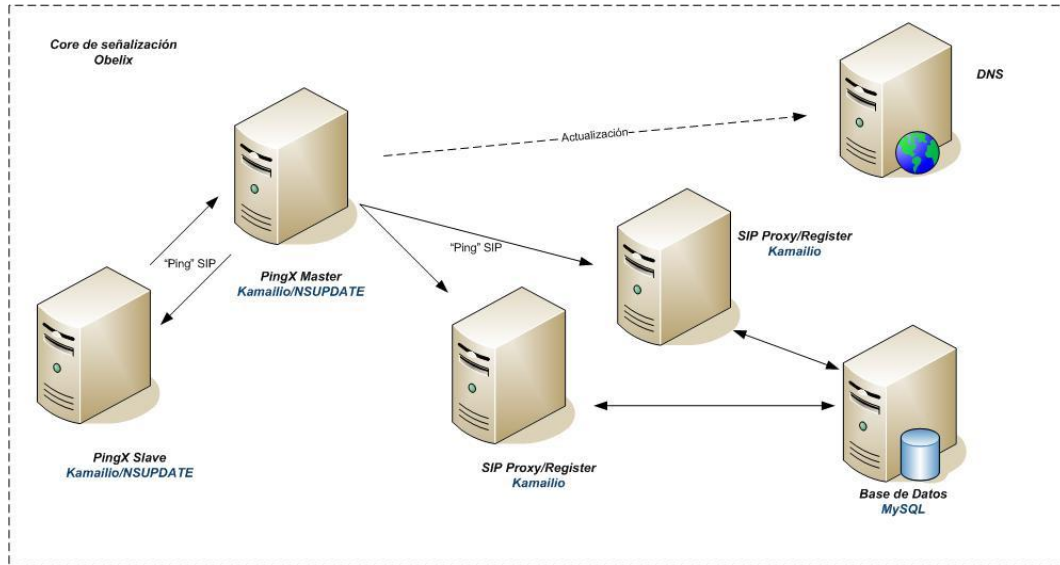


Figura 6. 14 – Esquema de funcionamiento normal de PingX con HA

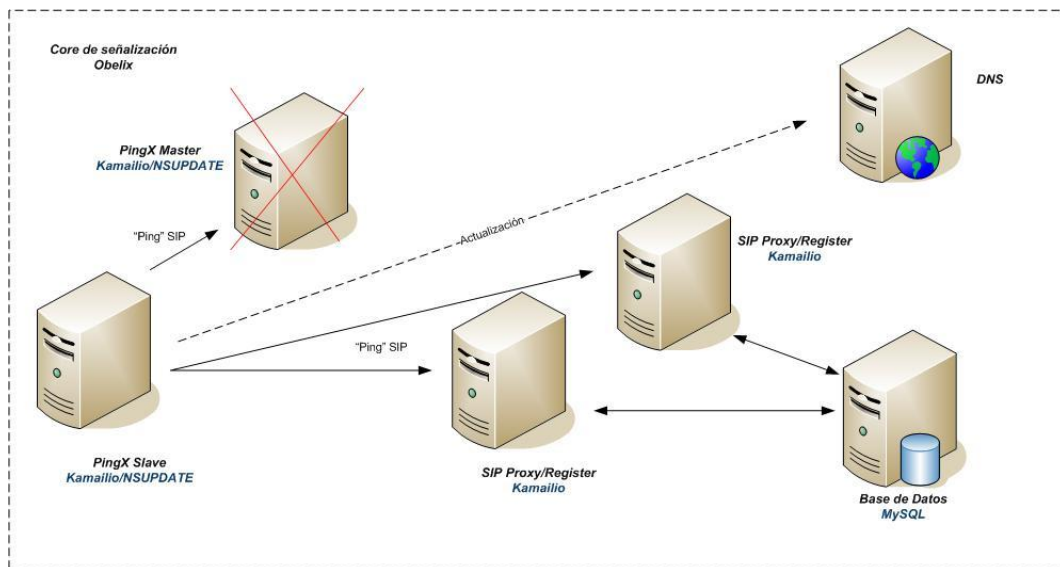


Figura 6. 15 – Esquema de funcionamiento de PingX ante falla

Como se ve en la figura 6.14, el servidor maestro además de estar monitoreando a los SIP Proxy, también se encuentra monitoreando a su par. Esto es por si el que falla es el

servidor esclavo poder tomar alguna acción, como por ejemplo disparar una alarma para dar cuenta al administrador del sistema de lo acontecido.

De este modo, la caída de algún elemento del modulo PingX no solo no afecta el funcionamiento del sistema sino que tampoco afecta la alta disponibilidad y *failover* de los SIP Proxy. La configuración necesaria para lograr este funcionamiento se puede consultar en el anexo 1.

6.3 – Base de datos HA

Otro punto de falla del sistema y que entra en la categoría de *spof*, dado que una falla dejaría inutilizado al sistema en su completitud, es el servidor que aloja a la base de datos del sistema.

Se evaluaron dos posibles arquitecturas para brindar alta disponibilidad en este servicio, replicación maestro-esclavo y replicación maestro-maestro.

La replicación maestro-esclavo, que se ilustra en la figura 6.16, funciona utilizando el servidor maestro para las escrituras o cambios en la base de datos, y el o los servidores esclavos son los encargados de replicar en ellos la información del servidor maestro. Las lecturas pueden ser realizadas en todos los servidores.

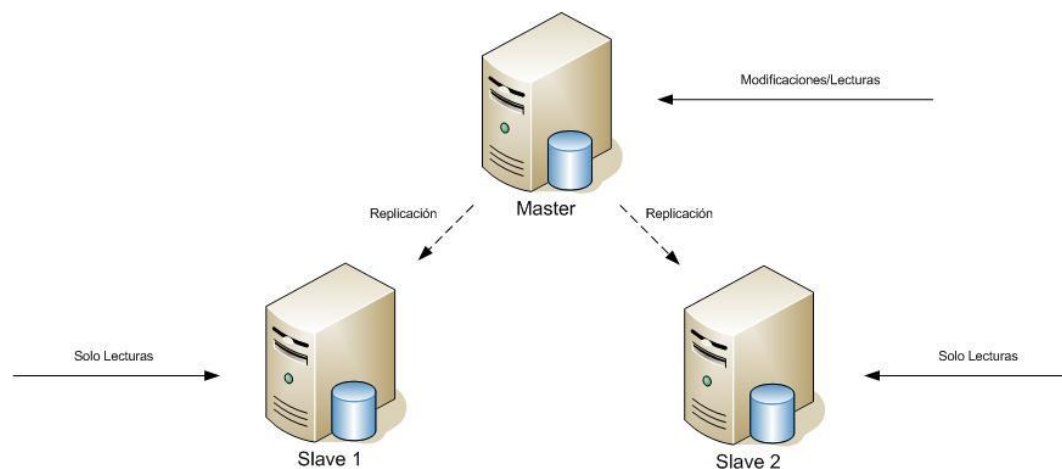


Figura 6. 16 – Esquema de arquitectura Master-Slaves de bases de datos

En la replicación maestro-maestro (figura 6.17), las escrituras o cambios pueden hacerse en cualquier servidor y se replicaran en el resto de los mismos.

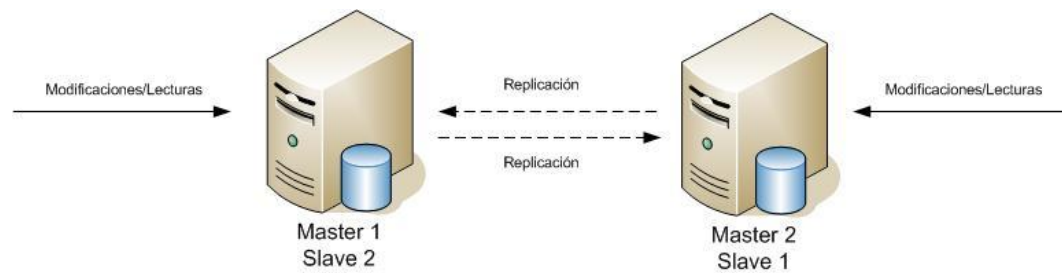


Figura 6. 17 – Esquema de arquitectura Master-Master de bases de datos

Un esquema maestro-esclavo es útil para la permanencia de los datos ante una falla del servidor maestro. Si fallase el servidor maestro, aun se podrían hacer consultas sobre los datos en el o los servidores esclavos. Sin embargo, si este es el caso, ya no se podrán hacer nuevas modificaciones en los datos del sistema hasta que se vuelva a poner en línea al servidor maestro. Por lo tanto, aun en este esquema en donde los datos se encuentran seguramente respaldados ante alguna falla, la caída del servidor maestro deja inutilizado el sistema completo dado que es esencial poder escribir datos en todo momento.

Un esquema maestro-maestro no tiene este problema ya que el sistema podría funcionar con uno u otro indistintamente. El problema que surge en este esquema es la dificultad para escalar en número de servidores. Una configuración de varios servidores en este modo implica un anillo en donde cada servidor es maestro de un servidor y esclavo de otro (con solo dos servidores cada uno es esclavo y maestro de su par). De esta manera, la falla de uno podría traer desincronización de los datos de todos los servidores, como se ilustra en la figura 6.18 (si falla uno, los cambios realizados en su respectivo maestro no se replicaran en ningún sitio, cortando así la cadena). Los métodos para lograr *failover* en un modo multi-maestro crecen en complejidad y por lo tanto en administración e implementación [73].

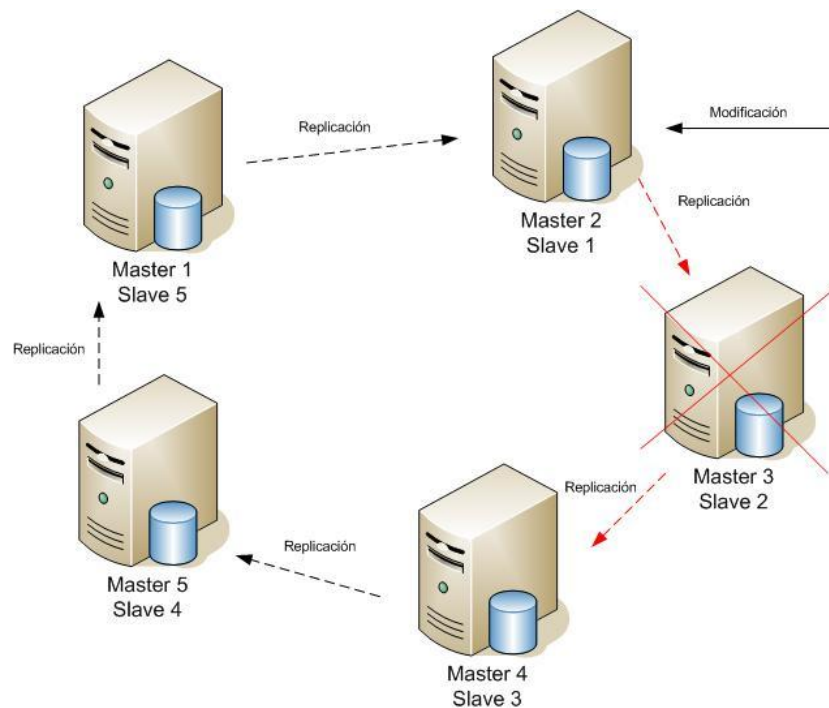


Figura 6. 18 – Esquema del problema de una arquitectura Master-Master en anillo para BD

Para el núcleo de señalización se busca tener permanencia de los datos ante una falla grave de un servidor de base de datos, y también seguir funcionando ante la caída de uno de estos servidores. Por lo tanto se implementó un esquema maestro-maestro con dos servidores que permite alcanzar lo buscado, y se ilustra en la figura 6.19. En el anexo 1 se encuentran todos los detalles para configurar un esquema de este tipo. Como ya se vio anteriormente, un solo servidor de base de datos es suficiente para servir al sistema entero, y es por esto que agregar un segundo servidor es más que suficiente y no hay que pensar en escalar aun más el número de servidores. Si se requiriera de esto en algún momento, podría pensarse en sistemas más complejos de alta disponibilidad como por ejemplo *MySQL cluster* [74].

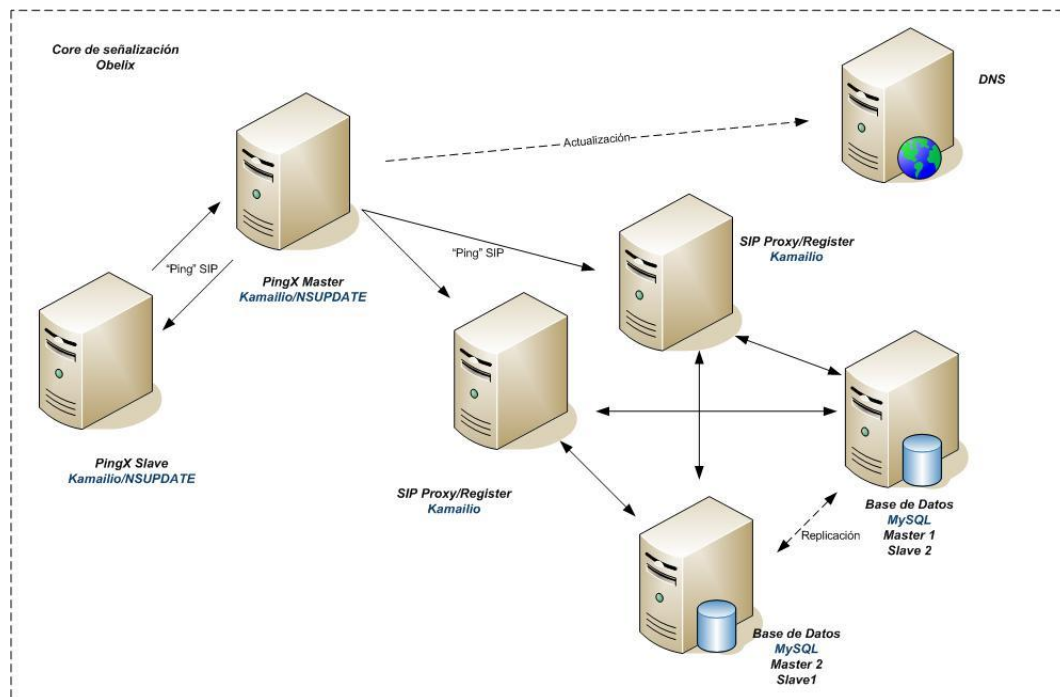


Figura 6. 19 – Arquitectura del núcleo de señalización con esquema Master-Master en BD

Con este esquema, los SIP Proxy del núcleo pueden utilizar una u otra base de datos indistintamente y esto es logrado haciendo uso del DNS. Los SIP Proxy del núcleo resuelven la dirección de red de la base de datos a utilizar en el sistema mediante una consulta al DNS. Si este responde con dos direcciones posibles, los servidores “saludarán” a ambas bases de datos y la primera en responder será el objetivo de las actualizaciones o consultas pertinentes. Si ambas bases de datos están en línea, el método del balance por DNS logrará un balance de la carga entre ambos servidores. En el caso que una base de datos se encuentre fuera de servicio solo responderá el saludo la que se encuentra en servicio, y de este modo el sistema seguirá funcionando correctamente con un solo servidor de base de datos.

6.4 – DNS HA

El último punto que hay que tratar para lograr la alta disponibilidad del núcleo de señalización en su totalidad, es el del servicio de DNS.

DNS es un protocolo que desde su diseño trata el tema del *failover*. Basta con indicar a un agente una lista de servidores, y estos si no obtienen respuesta por parte de uno intentan con otro.

Sin embargo, al aquí tratar con actualizaciones de la información de un DNS no es sencillo que la información este replicada en varios servidores.

Existen varias soluciones posibles para poder tener redundancia en el servicio de DNS que incluye actualizaciones dinámicas y todas ellas implican la colocación de otro servidor de DNS, siendo lo que varía sus implementaciones.

La primera de ellas se ilustra en la figura 6.20, y consiste en actualizar ambos servidores en lugar de actualizar uno solo cuando haya cambios en el sistema. De esta manera, ambos servidores tendrán la información actualizada y ninguno propagara información sobre un servidor caído.

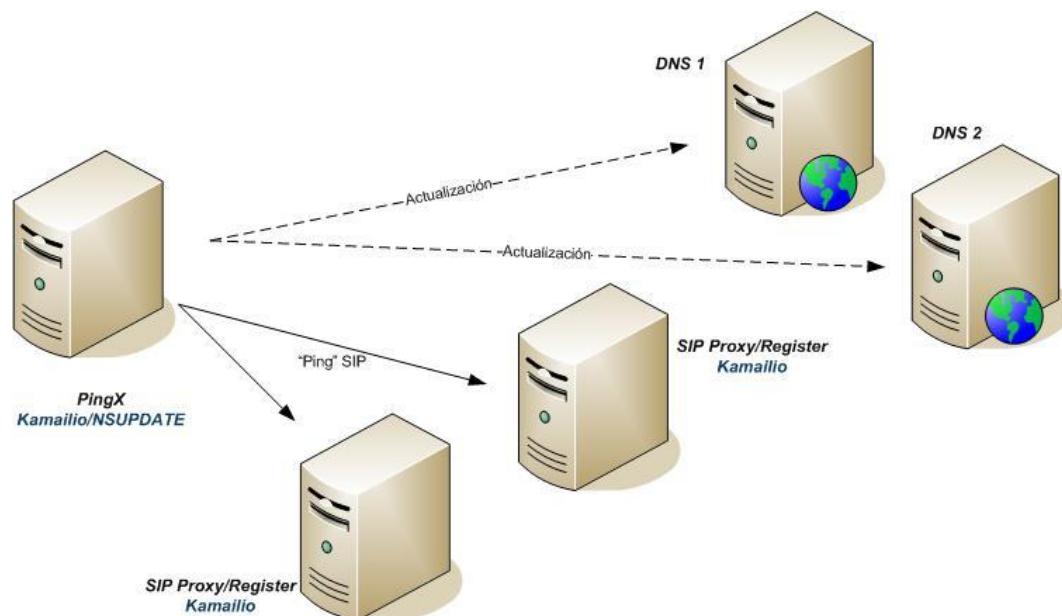


Figura 6. 20 – Esquema de arquitectura con doble DNS

La segunda de estas soluciones, que se ilustra en la figura 6.21, es aplicar un esquema de maestro-esclavo en estos servidores, de manera de que al actualizar los datos del servidor maestro estos cambios se repliquen en el servidor esclavo. La contrariedad de esta solución es que solo puede ser blanco de actualizaciones el servidor maestro. Si bien luego de la caída del servidor maestro el esclavo cuenta con la información correcta para responder las consultas, si se requiriera alguna actualización mientras el maestro esta fuera de servicio no podrá realizarse.

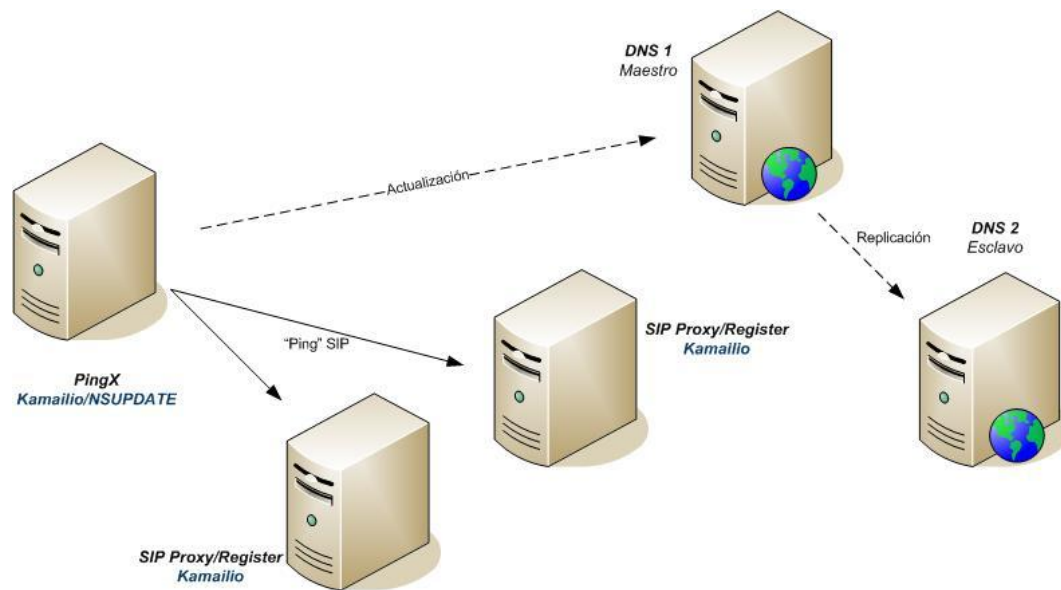


Figura 6. 21 – Esquema de arquitectura con DNS en modo Master-Slave

Ambas soluciones planteadas presentan un problema en común, y es que la caída de uno de estos servidores trae como consecuencia que los agentes que fueran dirigidos hacia ese servidor como primario para su consulta sufrirán demoras en el ingreso al sistema. Si bien luego consultarán al segundo servidor y recibirán respuesta, este intervalo de tiempo entre la consulta a uno y a otro podría resultar incomodo.

Una tercera solución es colocar ambos servidores bajo la misma dirección de red. Un esquema de este tipo no puede ser con ambos servidores actuando al mismo tiempo, como por ejemplo con alguna especie de balanceo, dado que la actualización la recibiría solo uno de los DNS, mientras que el otro continuaría propagando información desactualizada. Por lo tanto, es conveniente que un servidor sea el que atienda todas las consultas mientras que el otro se encuentre en *stand-by*. Este mecanismo debe funcionar junto con la configuración maestro-esclavo dado que solo el servidor maestro recibiría las actualizaciones debiendo estas ser replicadas en el esclavo.

Este esquema tiene la misma contrariedad que la solución maestro-esclavo, en donde si cae el maestro las actualizaciones ya no son posibles. Sin embargo, se elimina la demora que puede existir en la resolución de nombres, ya que en la misma dirección de red siempre se responde.

Para un correcto *failover* con esta solución, el servidor que se encuentra en *stand-by* debe ser capaz de entrar en servicio no solo ante una falla de hardware del servidor maestro sino que también debería detectar la caída del servicio de DNS en dicho servidor y tomar el control. El servidor maestro ante una falla a nivel de aplicación debería salir de servicio, y por lo tanto esta solución no debe ser puramente a nivel de red. De ser así, si fallase la aplicación de DNS en el servidor maestro pero a nivel de red

este sigue activo y no se toma ninguna acción, todo el sistema dejara de funcionar. Esto complejiza la implementación de esta solución, pero sigue siendo posible.

La única solución que permite que ante la caída de un servidor de DNS, el servicio pueda seguir siendo actualizado dinámicamente es la primera aquí planteada. El costo que tiene es la posible demora en la entrada al sistema, dada mientras el protocolo de DNS logra encontrar el servidor que se encuentra activo. Si el servidor DNS del cliente, que redirige las consultas a los DNS del núcleo, no alterna las entradas a los servidores de nombres, todos los usuarios sufrirán demoras si el servidor que falla es el primero de la lista. Si este alternara las entradas de los servidores de nombres, solo la mitad de los usuarios sufrirán demoras, dado que la otra mitad consultaran directamente el servidor que se encuentra en línea. Esta solución también es la de más sencilla implementación, y es por esto y todo lo comentado que se asume el costo de las posibles demoras y se implementa de esta manera.

Cabe destacar que la caída de uno de estos servidores debe ser inmediatamente atendida, ya sea volviendo a poner en línea el servidor caído o quitando administrativamente la entrada a ese servidor del DNS del cliente mientras se realiza el trabajo de restauración. Así de esta manera las demoras en la entrada al sistema se darán por un periodo corto de tiempo.

6.5 – Arquitectura del Núcleo HA

Luego de discutida la alta disponibilidad de todos los elementos del núcleo de señalización, se presenta una posible arquitectura que cumple todos los requerimientos de disponibilidad del servicio ante fallas en algunos de sus elementos (Figura 6.22).

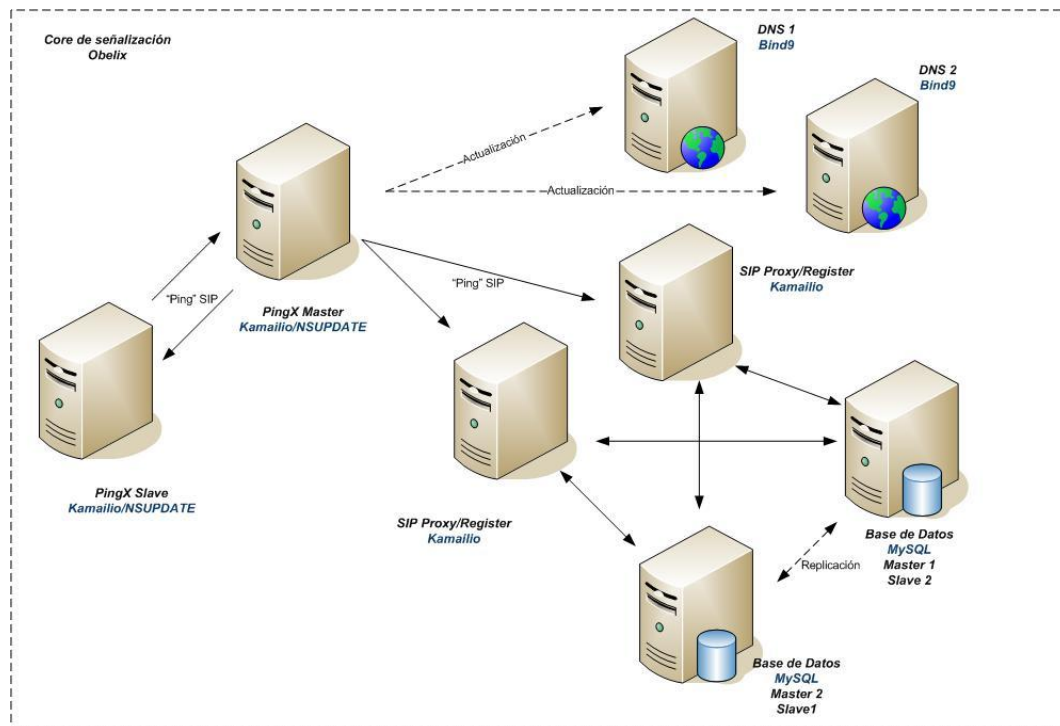


Figura 6. 22 – Arquitectura final del núcleo de señalización con HA

Esta arquitectura presentada, si bien es la que se implementó, podría aceptar otra versión. Si bien los bloques a nivel lógico son los presentados, podría no ser necesaria la cantidad de equipos propuestos.

En el caso de utilizar en el sistema solamente dos SIP Proxy, sería posible implementar el modulo PingX en estos servidores. De esta manera, cada SIP Proxy monitorearía a su par, y ante una caída del mismo podría disparar el mecanismo de actualización de los DNS (Figura 6.23).

Como se comentó durante el capítulo, todos los detalles de instalación y configuración de esta arquitectura se pueden consultar en el anexo 1.

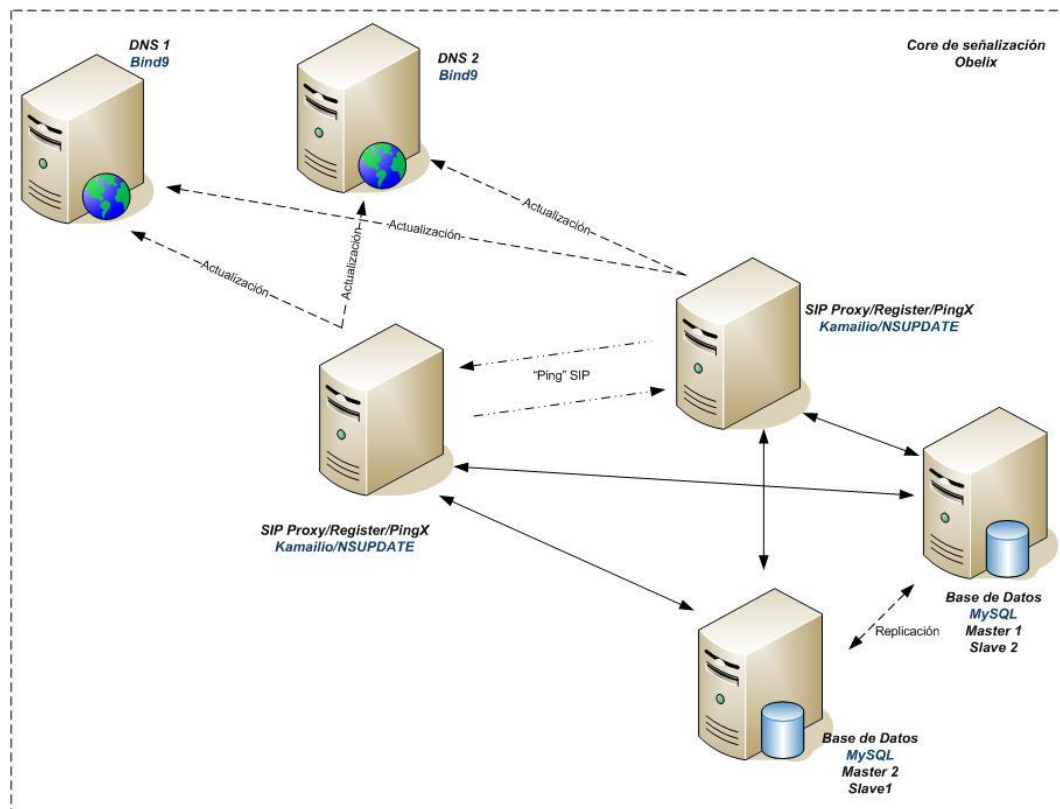


Figura 6. 23 – Variante de la arquitectura del núcleo de señalización con HA

También se podría implementar el acople del modulo PingX a los SIP Proxy si estos fueran más de dos, pero en este caso la disposición sería diferente. Como se muestra en la figura 6.24, el primer servidor actúa como el maestro del modulo PingX testeando a todos los servidores. El segundo servidor actúa como el esclavo, testeando solo al maestro y listo en caso de falla para tomar su rol. Esta configuración logra el *failover* ante la caída de cualquier cantidad de servidores, siempre y cuando no caigan maestro y esclavo al mismo tiempo. Si esto sucediera y un tercer servidor falla, no habrá nadie que actualice los DNS.

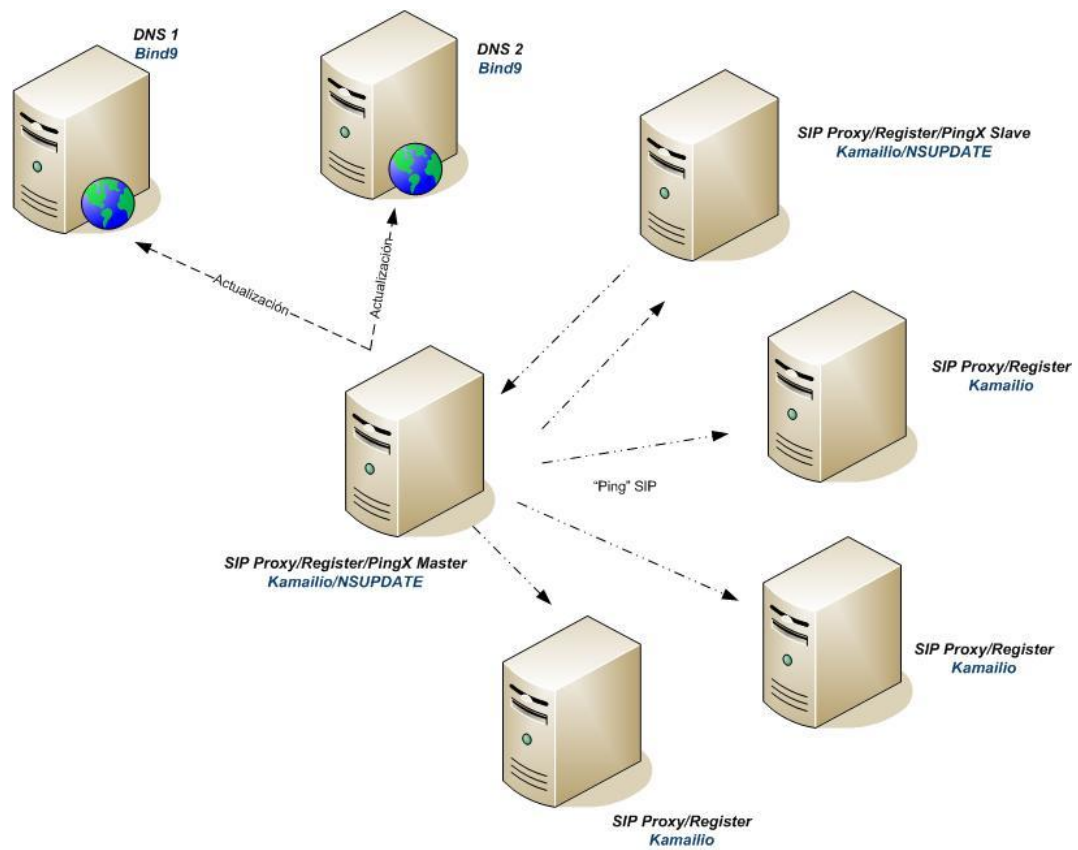


Figura 6. 24 – Esquema del funcionamiento con N servidores SIP Proxy en la variante de la arquitectura

Capítulo 7 – Alta Disponibilidad de las Funcionalidades

Este capítulo está dedicado a buscar una solución a la alta disponibilidad en todas las *features* que no fueron tratadas en el capítulo anterior. Con la alta disponibilidad del núcleo de señalización se logró la alta disponibilidad y *failover* en los servicios de llamadas punta a punta, así como en los registros de usuario, pero no todas las funcionalidades planteadas en el capítulo 5 cuentan con un esquema de alta disponibilidad.

Se puede separar a las funcionalidades del sistema en dos grandes grupos, funcionalidades de solo señalización y funcionalidades con manejo de audio.

Las funcionalidades de solo señalización serán todas aquellas que no requieran manejo de flujos de media para ser brindadas. Ejemplo de estas son la presencia, las transferencias, el *accounting*, la mensajería instantánea y el manejo de permisos y grupos.

Las funcionalidades con manejo de audio son aquellas que justamente necesitan hacer manejo de flujos de media para ser brindadas. Ejemplo de estas son la música en espera, el correo de voz y los puentes de conferencias.

Esta distinción es bien importante por lo siguiente, y es que todas las funcionalidades del primer tipo no requieren de ningún servidor extra a los que ya se tienen en el núcleo de señalización. Esto quiere decir que la arquitectura de alta disponibilidad para el núcleo de señalización logra también la alta disponibilidad de todos los servicios brindados enteramente por este núcleo, y estos son el primer tipo de funcionalidad aquí planteada.

Esto deja solamente para resolver el problema de la alta disponibilidad del segundo tipo de funcionalidades del sistema, las funcionalidades con manejo de audio.

En lo que sigue se plantearan soluciones para lograr la alta disponibilidad de las funcionalidades de este tipo implementadas por el sistema.

7.1 – Music on Hold

Como se presentó en el capítulo 5, el servicio de música en espera requiere de un servidor dedicado a brindar un flujo de media predeterminado cuando un usuario coloca en espera a otro. Si este servidor fallare, el servicio quedaría inaccesible.

De esta manera, resulta natural buscar redundancia en el servicio colocando otro servidor encargado de la misma tarea. Esta solución se ilustra en la figura 7.1.

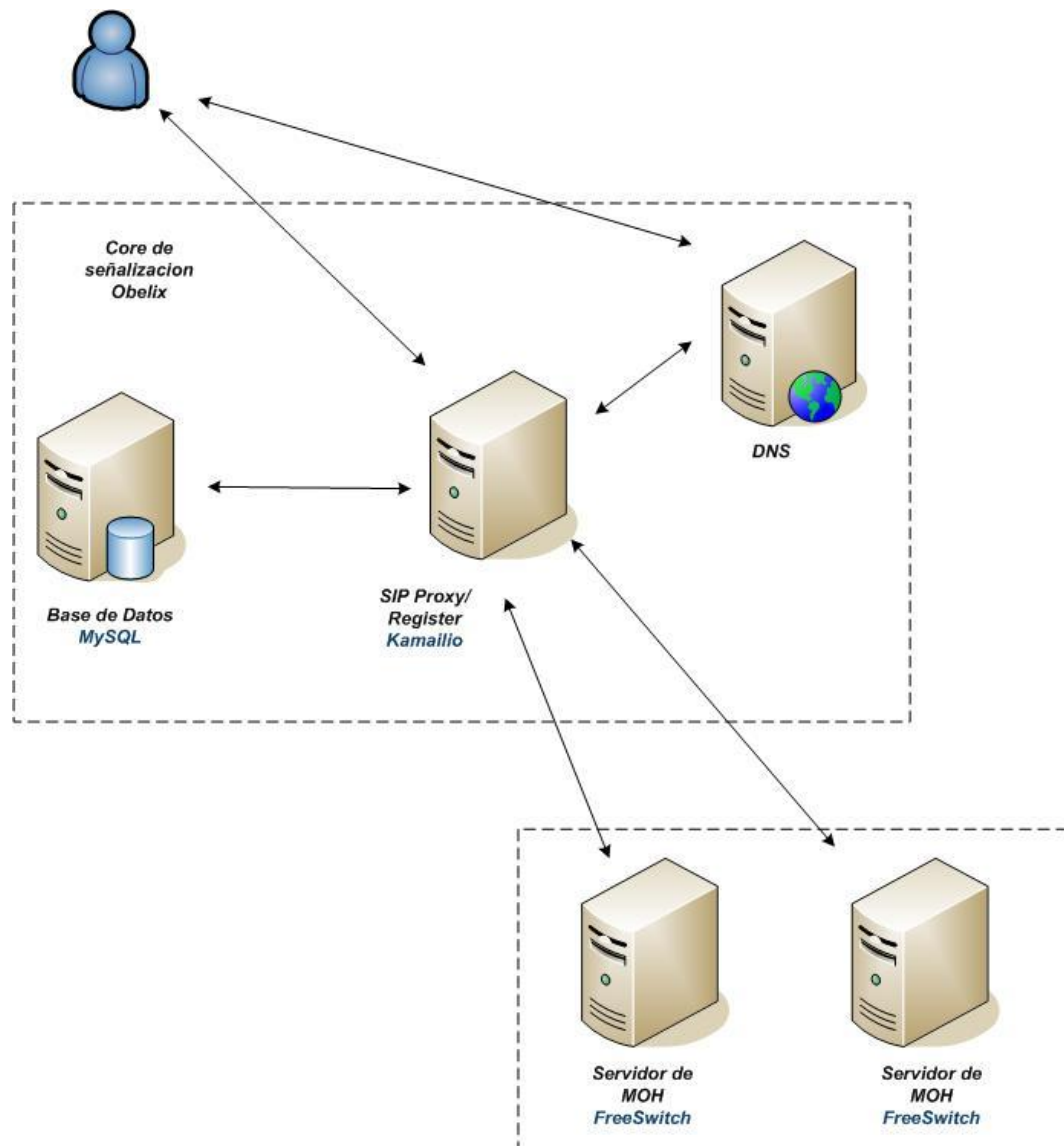


Figura 7. 1 – Esquema de alta disponibilidad en la música en espera

Estos dos servidores, en principio idénticos, son completamente independientes. Es indistinto escuchar la música que brinda uno u otro de estos servidores, y ninguno de ellos debe compartir algún tipo de información con el otro. Solo basta con que cuenten con una configuración idéntica (que se puede consultar en el anexo 2) y que alguna entidad elija uno u otro servidor para musicalizar la llamada, para que el hecho que haya dos (o mas servidores) de música en espera sea completamente transparente para los usuarios, y también para el núcleo de señalización.

Los problemas para lograr un balanceo entre estos dos servidores son los mismos que los planteados en el capítulo 6 para el balanceo de los SIP Proxy del núcleo. Sin embargo, una solución que no era viable para los SIP Proxy del núcleo, si lo puede ser en este caso.

El uso del módulo *dispatcher* de Kamailio, que como se vio anteriormente en el capítulo 6 es un balanceador de señalización que entiende SIP capaz de realizar *failover*, es especial para este cometido. En la figura 7.2 se ilustra su funcionamiento. Cuando una llamada es puesta en espera, el núcleo invita a este servidor de despacho, y este ultimo decide cual servidor será el encargado de brindar la música.

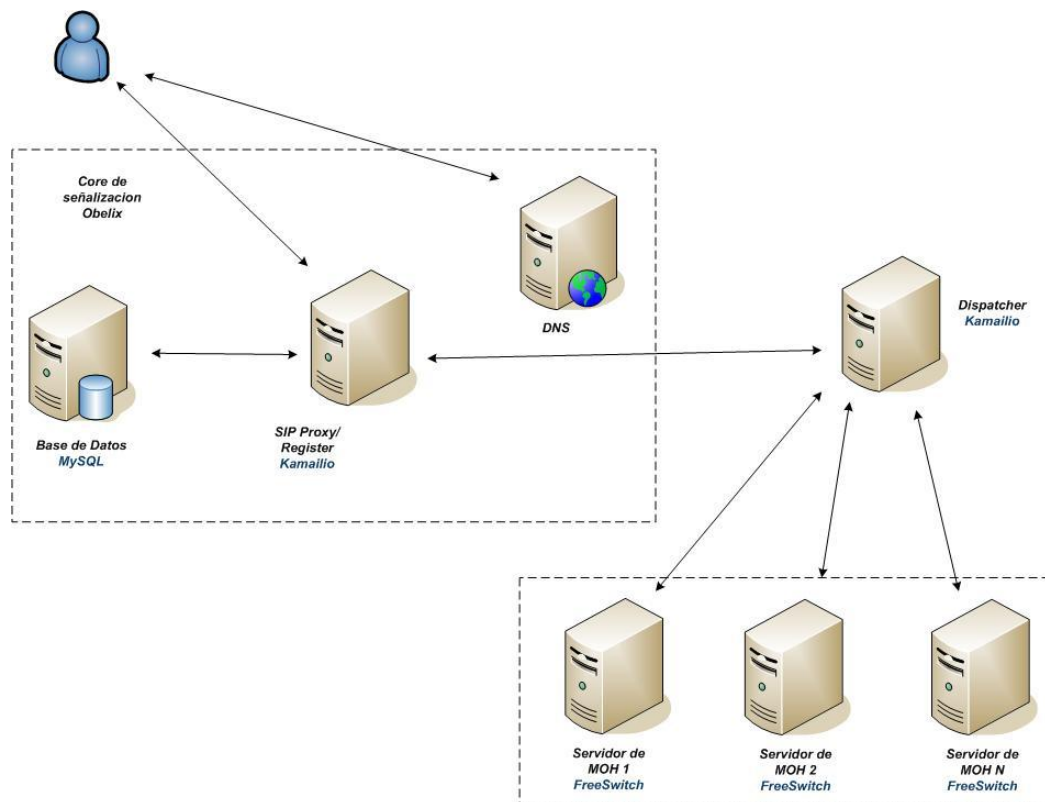


Figura 7. 2 – Esquema de funcionamiento del módulo de despacho

Esta configuración permite colocar aguas abajo del servidor de despacho N servidores de música en espera. Esto es útil ya sea por la demanda de este servicio, o por la capacidad de procesamiento de los servidores con que se cuente. Esto es que si no se tiene buenos servidores, se pueden poner varios servidores de segunda mano sin ningún tipo de problema.

El servidor de despacho, que es capaz de “pinguear” con mensajes SIP a todos los servidores que tenga aguas abajo y así detectar la caída de alguno de ellos, logra sacar de la lista de despacho aquellos servidores con mal funcionamiento consiguiendo así *failover* en el servicio.

El problema que se presentaba con los servidores SIP Proxy del núcleo y esta solución era que este servidor de despacho se convertía en un único punto de falla. Al agregar un segundo servidor de despacho, se solucionaba el problema pero aparecía otro, y era que se tornaba difícil lograr un balance entre ambos servidores de despacho¹⁰.

Sin embargo, no es este el caso. Los servidores de despacho para el servicio de música en espera se encuentran aguas abajo de los servidores del núcleo de señalización. Estos servidores son capaces de implementar balance de señalización (mediante DNS) hacia estos servidores de despacho logrando *failover* y alta disponibilidad del servicio de despacho.

La figura 7.3 ilustra el esquema planteado.

¹⁰ Ver capítulo 6.

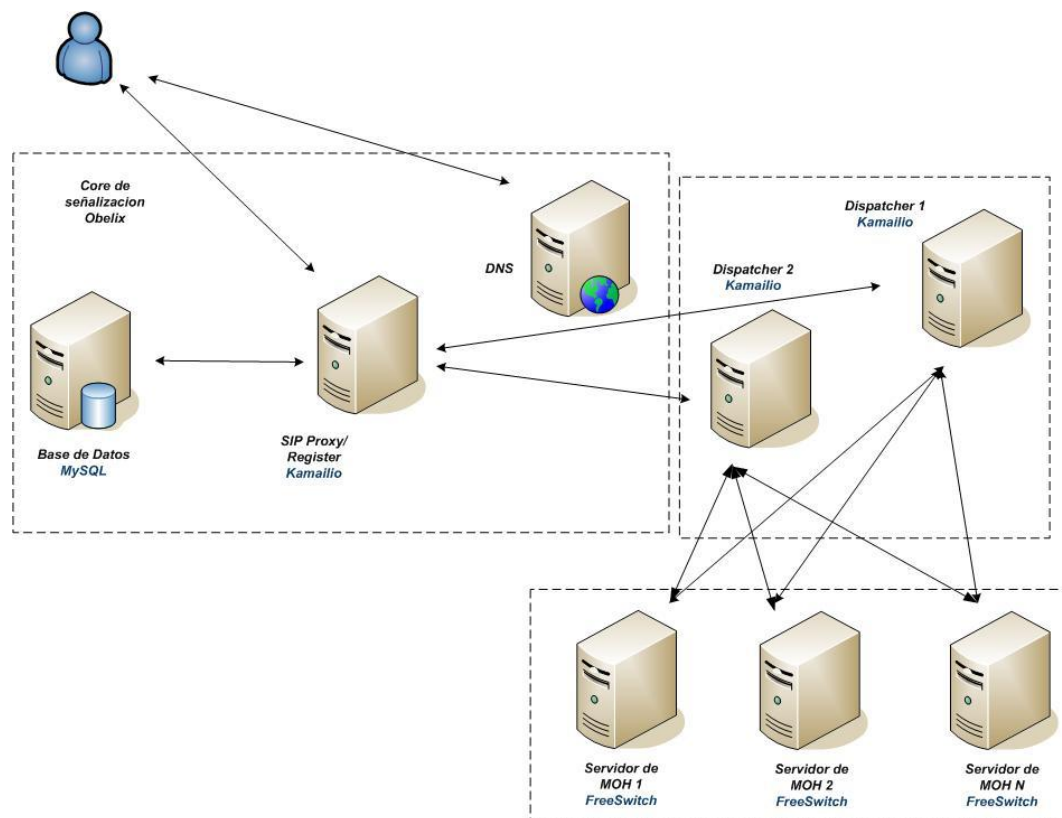


Figura 7.3 – Esquema de funcionamiento del módulo de despacho con HA

De esta manera se logra un balanceo sobre los servidores de música en espera, se logra persistencia del servicio al lograr *failover*, la solución es escalable a N servidores, y se tiene una potente herramienta de despacho.

Estos servidores de despacho, que implementan varios algoritmos de balanceo, deberán trabajar con uno en particular, el algoritmo de *hash* sobre el parámetro *call-ID*, para que todos los mensajes de una misma conversación o llamada sean dirigidos al mismo servidor. En un principio no sería necesario, dado que todos los mensajes pertenecientes a una llamada ya iniciada seguirán la ruta definida al comienzo (encabezado “Route”). Pero si alguno de los servidores aguas debajo de un servidor de despacho solicitara algún tipo de autenticación, no utilizar el algoritmo comentado puede ser fuente de errores. En la figura 7.4 se ilustra un comportamiento erróneo del sistema al no balancear mediante “Call-ID”.

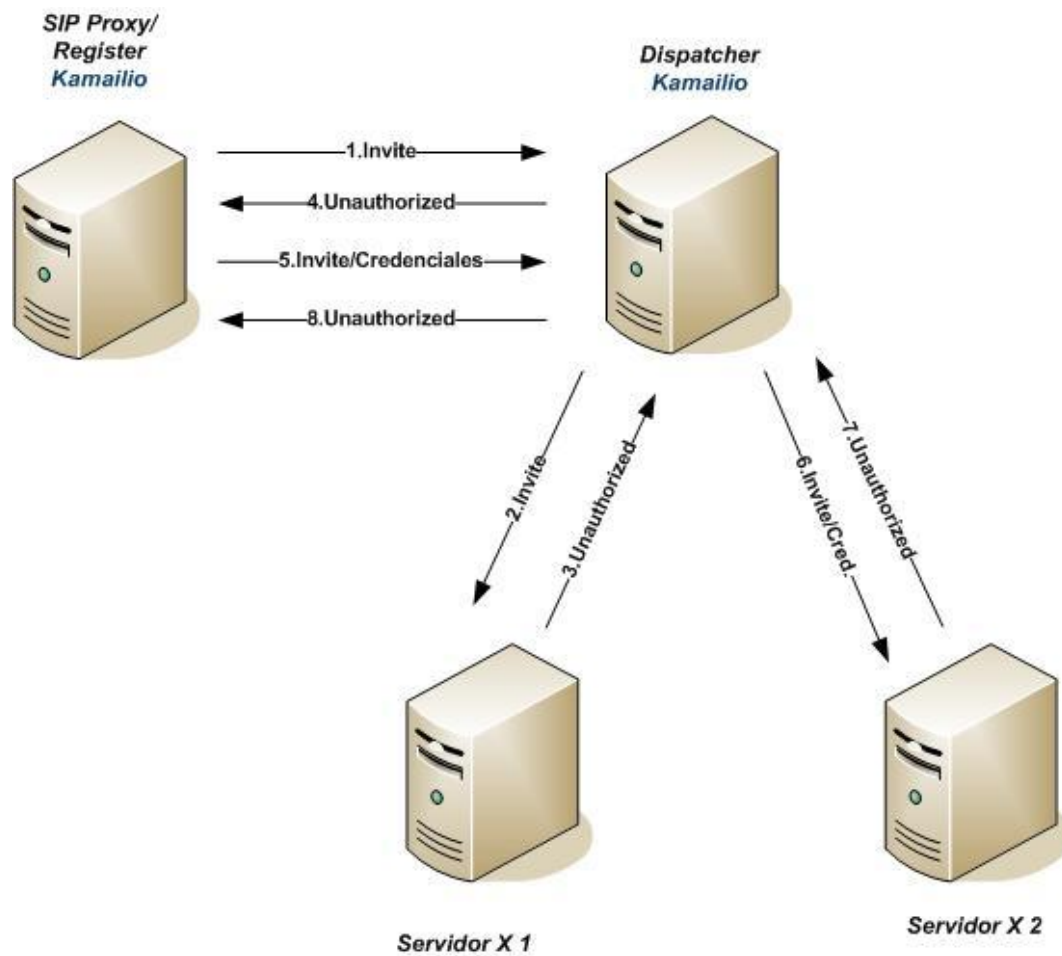


Figura 7. 4 – Esquema de funcionamiento erróneo del módulo de despacho

Este módulo que implementa balanceo y *failover* del banco de servidores de música en espera será de utilidad para los mismos fines en otras funcionalidades. En el anexo 1 se puede consultar la configuración asociada a este nuevo módulo.

7.2 – Voicemail

Para lograr la alta disponibilidad del servicio de *voicemail*, también resulta natural redundar el servicio colocando más servidores dedicados a esta tarea.

Aplicando las mismas ideas que para el servicio de música en espera, el módulo de despacho presentado en la sección anterior será de gran utilidad al momento

de balancear las llamadas entre estos servidores de *voicemail* y también lograr *failover* del servicio, dejando de despachar llamadas a un servidor que fallare.

Este sistema, que se ilustra en la figura 7.5, no basta para que el servicio funcione de manera correcta.

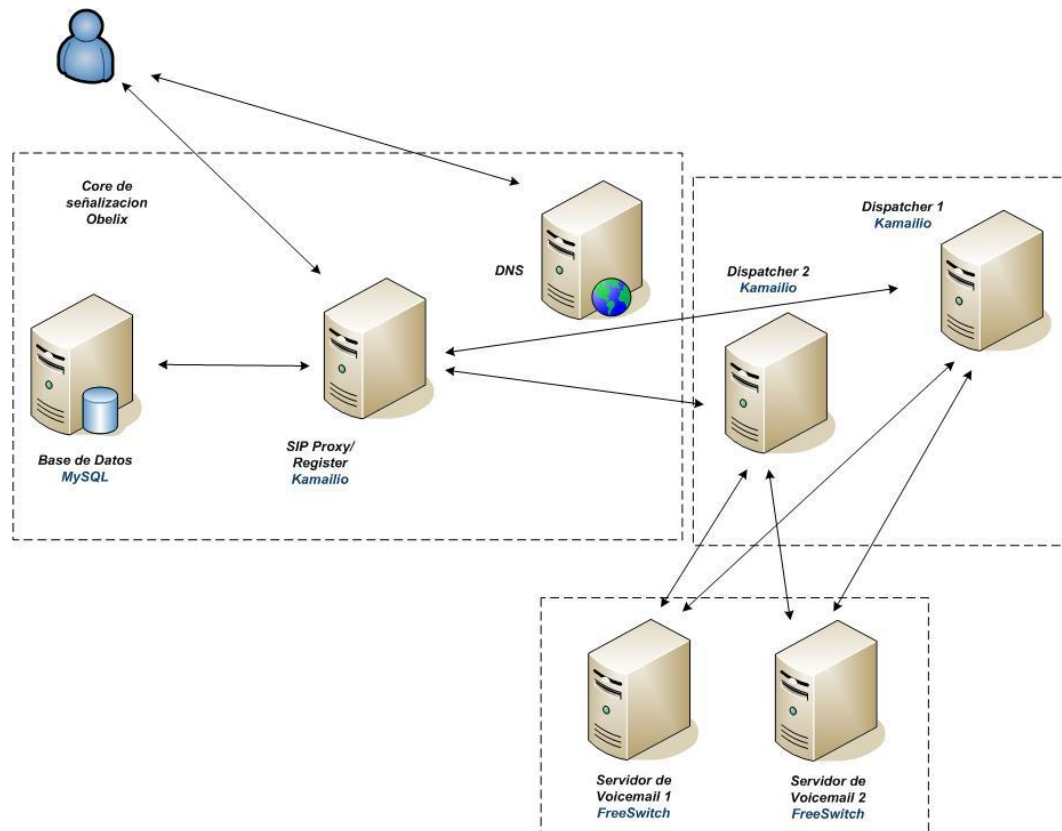


Figura 7. 5 – Esquema de HA en el servicio de voicemail

Este esquema presentado funciona perfectamente cuando las llamadas que arriban a los servidores de *voicemail* tienen como cometido dejar un mensaje de voz en la casilla de algún usuario. Cualquier servidor que atienda dicha llamada le permitirá dejar un mensaje al usuario que llama.

El problema radica en las llamadas destinadas a consultar los mensajes de voz dejados en el sistema. Este problema, que se ilustra en la figura 7.6, sucede cuando un mensaje de voz para el usuario “tomasito@obelix” es dejado en el servidor 1 (elegido aleatoriamente por el modulo de despacho), y cuando “tomasito” llame al servicio de *voicemail* para consultar su mensaje, este sea

dirigido (también aleatoriamente) al servidor 2. El servidor 2 le informará que no tiene ningún mensaje de voz ya que su mensaje se encuentra en el servidor 1.

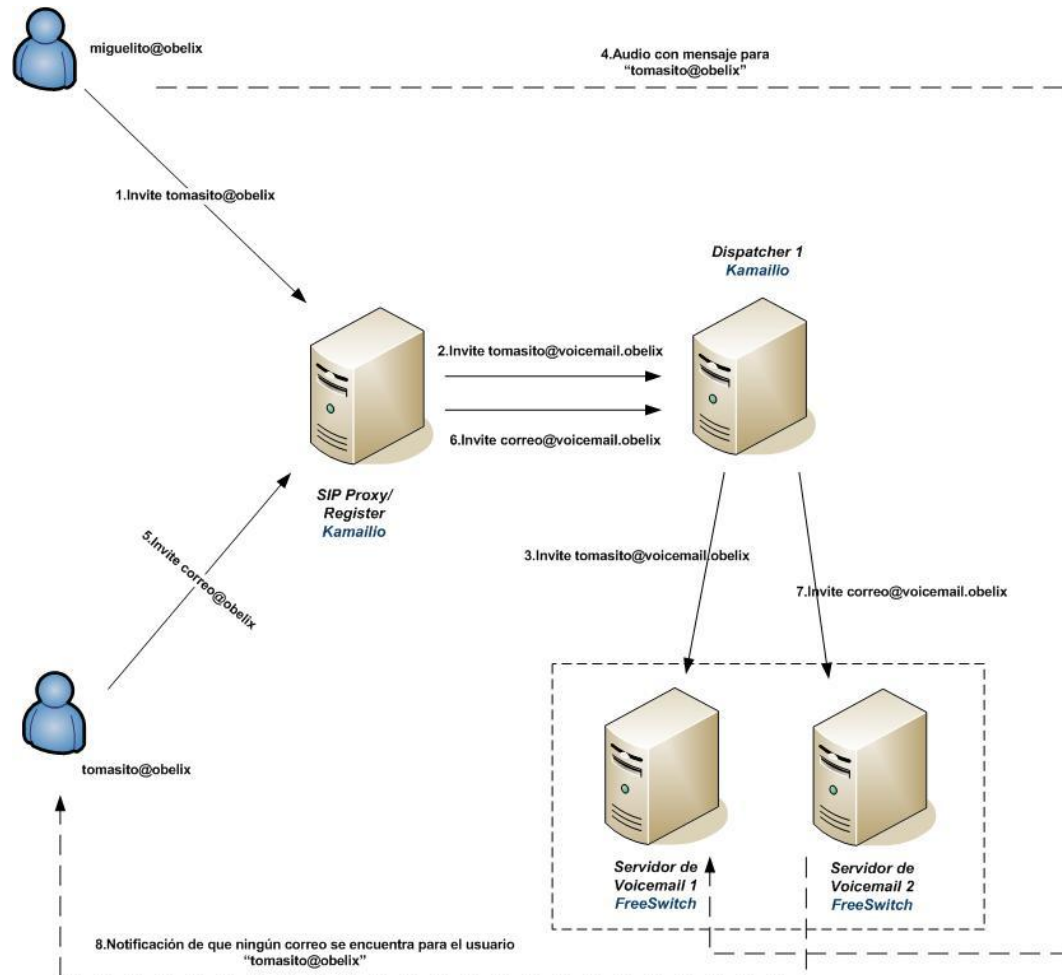


Figura 7. 6 – Ilustración de caso de falla del esquema planteado

El esquema de los servidores de *voicemail* es distinto al de los servidores de música en espera dado que estos no pueden trabajar de manera independiente son conocer la existencia de otros servidores.

Una solución a este problema podría ser el uso de “casillas estáticas” para cada usuario en el banco de *voicemail*. Esto es que no se use el método aleatorio en el modulo de despacho, sino que dependiendo del usuario al que se le quiere dejar un mensaje, o el usuario que quiere consultar su correo, se lo despache a un servidor asociado a ese usuario. Este esquema se ilustra en la figura 7.7.

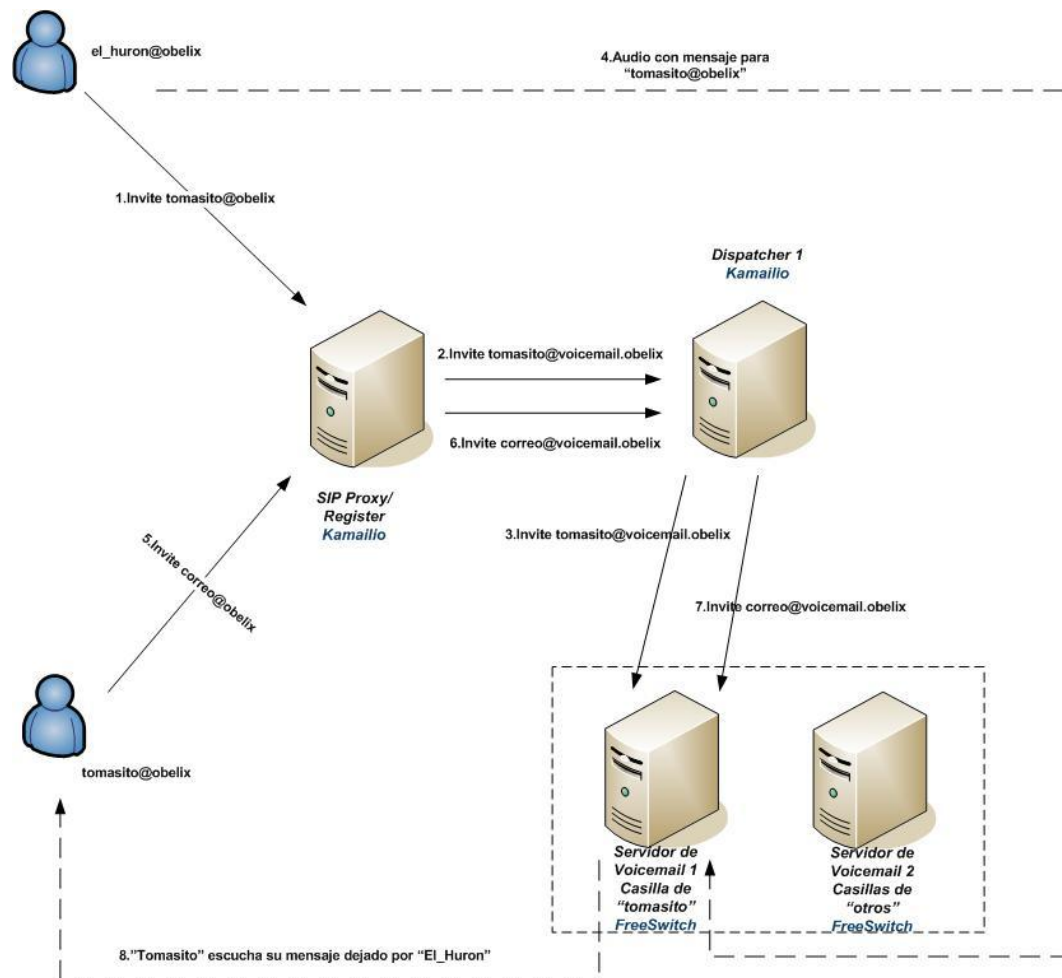


Figura 7. 7 – Esquema de solución con casillas de voicemail estáticas

Esta solución tiene dos grandes inconvenientes. El primero de ellos es el hecho de que no es una solución balanceada entre los servidores de *voicemail* dado que depende de la actividad de cada usuario en particular que a priori no se puede inferir para el diseño. El segundo inconveniente es que ante la caída de uno de estos servidores de *voicemail*, todos los usuarios que allí tuviesen sus casillas no podrán consultar sus mensajes y tampoco les podrán dejar de los mismos. Esto hace que no haya continuidad del servicio si un servidor fallare para todos sus usuarios asociados.

7.2.1 – Base de Datos Compartida y NFS

Otra solución a este problema es utilizar en todos los servidores un sistema de archivos compartidos. Esto permitiría que aunque un mensaje para un usuario se encontrara en el servidor 1, el servidor 2 podría acceder a él.

El inconveniente de esta solución radica en que el servidor dos debe conocer que existe un mensaje para ese usuario aunque este se encuentre en otro servidor, así como también debe conocer la manera de acceder a él.

Para solucionar esto los servidores de *voicemail* utilizan una base de datos externa a ellos, no para guardar los mensajes per se, sino que guardan en esta base de datos *metadata* de los archivos de audio. Esta información contiene, entre otras cosas, para quien es el mensaje así como su ubicación física.

De esta manera, cuando un mensaje es dejado para un usuario en el servidor 1, este servidor escribe en la base de datos la información relativa a ese mensaje. Cuando ese usuario acceda al sistema a consultar sus mensajes, no importa que servidor atienda su pedido, este consultara la base de datos para informar al usuario cuantos mensajes existen en su cuenta. Si este fuera el servidor 2, este informaría al usuario que existe un mensaje, aunque este se encuentre en el servidor 1.

Si el usuario desea escuchar ese mensaje, el servidor 2 consultara la ubicación física de ese mensaje, obteniendo una ruta a un directorio del servidor 1.

Lo que resta es que el servidor 2 sepa acceder a ese directorio. Para esto se utilizó un sistema de archivos NFS (*network file system*) donde cada servidor de *voicemail* tiene links simbólicos en sus propios directorios hacia directorios en otros servidores a través de la red. De esta manera, cuando el servidor 2 recupere la ruta física al mensaje, recibirá un directorio que existe en su sistema de archivos, pero sus archivos se encuentran en un directorio remoto. Esto es transparente para el servidor que intenta reproducir el archivo ya que puede hacerlo como si estuviera en su propio disco.

Este funcionamiento se ilustra en la figura 7.8. Al hacer uso de una base de datos externa, se aprovechó la configuración de bases de datos existente en el núcleo de señalización para esta tarea, obteniendo alta disponibilidad en el servicio.

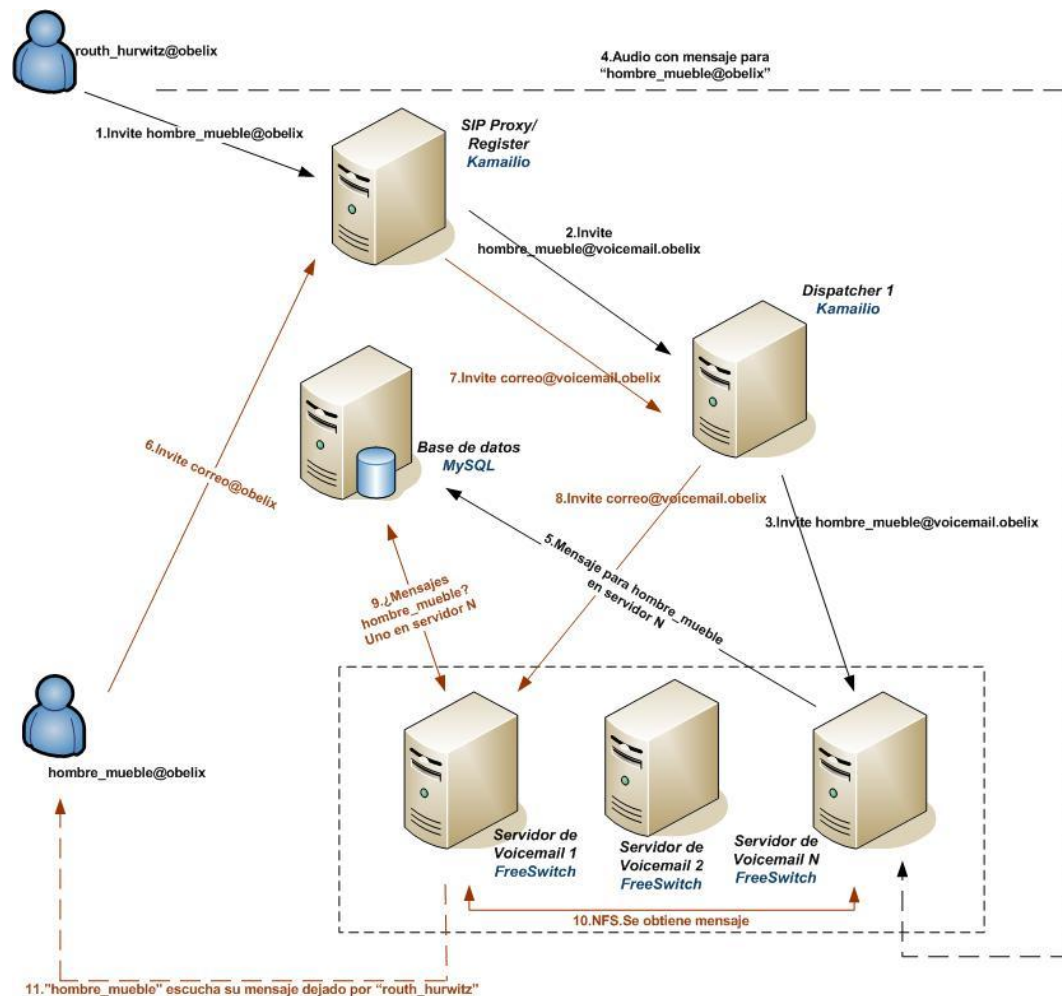


Figura 7. 8 – Esquema de funcionamiento del banco de voicemail con NFS y base de datos

Con el esquema presentado, cuando un servidor falla, el servicio para dejar mensajes a cualquier usuario sigue funcionando perfectamente. Sin embargo, si hay un servidor fuera de línea que contiene mensajes para un determinado usuario, cuando este consulte sus mensajes va a ser informado de la cantidad correcta de mensajes que tiene, pero solo va a poder escuchar aquellos que no se encontraran en el servidor caído. Esto es preferible ante lo que ocurría con la solución estática.

Cuando un servidor caído es puesto en línea nuevamente, todos los mensajes que se encuentran en ese servidor vuelven a ser accesibles para sus dueños.

La única excepción a esto es que un servidor tenga una falla de hardware a nivel de disco duro, donde se perderían todos los mensajes guardados en dicho disco.

Para esto se pudo implementar una copia de seguridad de todos los discos hacia una maquina externa, por ejemplo al finalizar el día, y de esta manera ante una falla en un disco de un servidor, el administrador puede poner en servicio nuevamente al servidor y recuperar los mensajes que estaban en su disco.

Este esquema permite no solo un balance de la carga en atención de llamadas de los servidores, sino que también naturalmente queda balanceado entre todos los servidores el espacio utilizado en disco duro.

Es posible escalar esta solución a N servidores, pero cada vez que se agregue un servidor, hay que reconfigurar todos ya que deben conocer la ruta a los archivos del nuevo servidor.

En el anexo 2 se puede consultar la configuración necesaria en los servidores de *voicemail* para el uso de base de datos, así como también la configuración del NFS.

El modulo de despacho no requiere configuración extra más que la lógica de distinguir entre una llamada al servicio de *voicemail* y una llamada al servidor de música (fácilmente distinguible a través de la R-URI), para elegir hacia donde debe ser despachada la llamada.

7.2.2 – Suscripciones (MWI)

Como se vio en el capítulo 5, los usuarios del servicio de *voicemail* son capaces de suscribirse al servidor de *voicemail* para que sean notificados cuando un mensaje nuevo les sea dejados, o al registrarse en el sistema conocer si tienen mensajes nuevos.

Con la solución planteada en la sección anterior, las suscripciones de los usuarios también son guardadas en la base de datos.

Cuando un usuario se registra en el sistema, el mismo envía una suscripción al servicio de *voicemail* que es dirigida al núcleo de señalización. Este detecta que la suscripción es para dicho servicio y redirige el mensaje hacia el modulo de despacho. Este último detecta que el mensaje está dirigido al servicio de *voicemail*, por lo que aleatoriamente elige uno de los servidores activos de dicho servicio para enviarle la suscripción.

El servidor destino acepta la suscripción y la guarda en la base de datos, así como la información de contacto para ese usuario. Inmediatamente consulta la misma

base para obtener cuantos mensajes existen para dicho usuario (no importa en donde se encuentran físicamente, en la base de datos se encuentra la información de todos los mensajes) y notifica al remitente de la suscripción su cantidad de mensajes. Esto se ilustra en la figura 7.9.

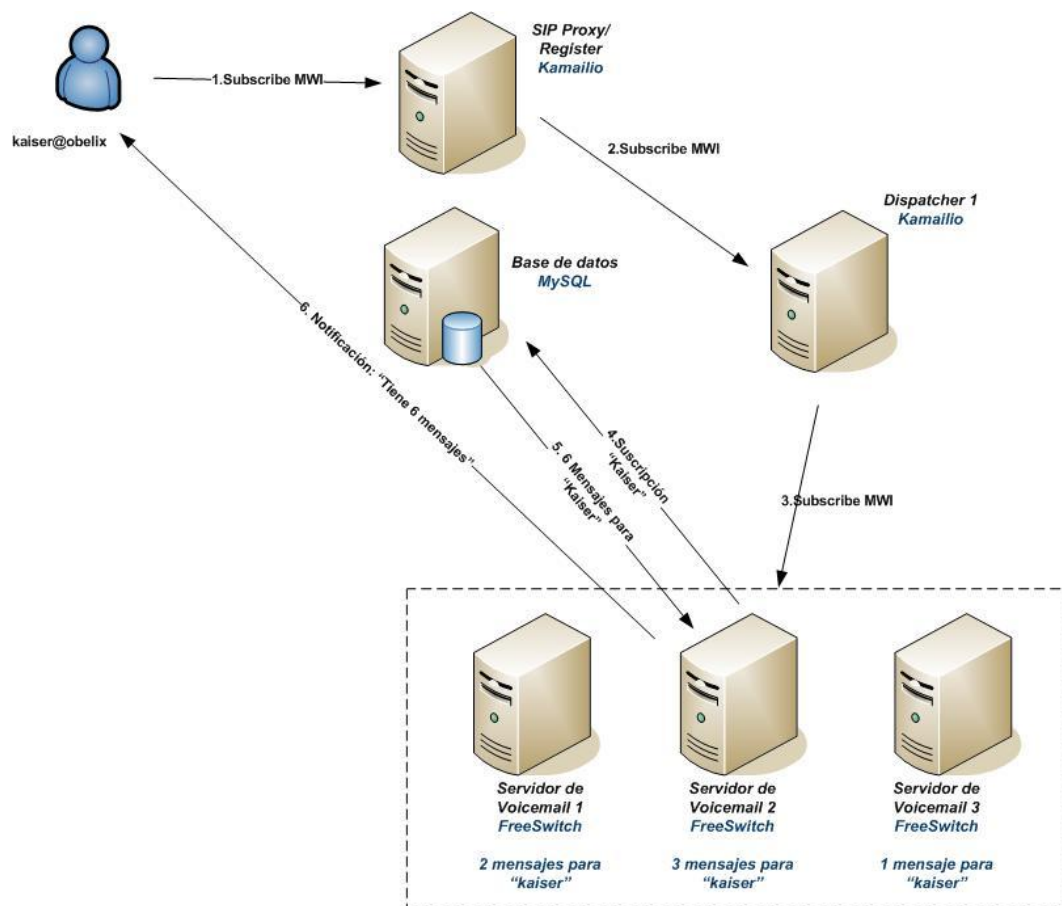


Figura 7. 9 – Esquema 1 de funcionamiento de suscripciones y notificaciones de MWI con HA

Cuando un usuario deja un mensaje de correo de voz a otro usuario en algún servidor, este servidor consulta a la base de datos si el usuario destinatario del mensaje se encuentra suscripto al servicio de notificaciones. Si así fuere, obtiene de la misma base los datos de contacto para notificar al mismo de su nuevo mensaje. Esto se ilustra en la figura 7.10.

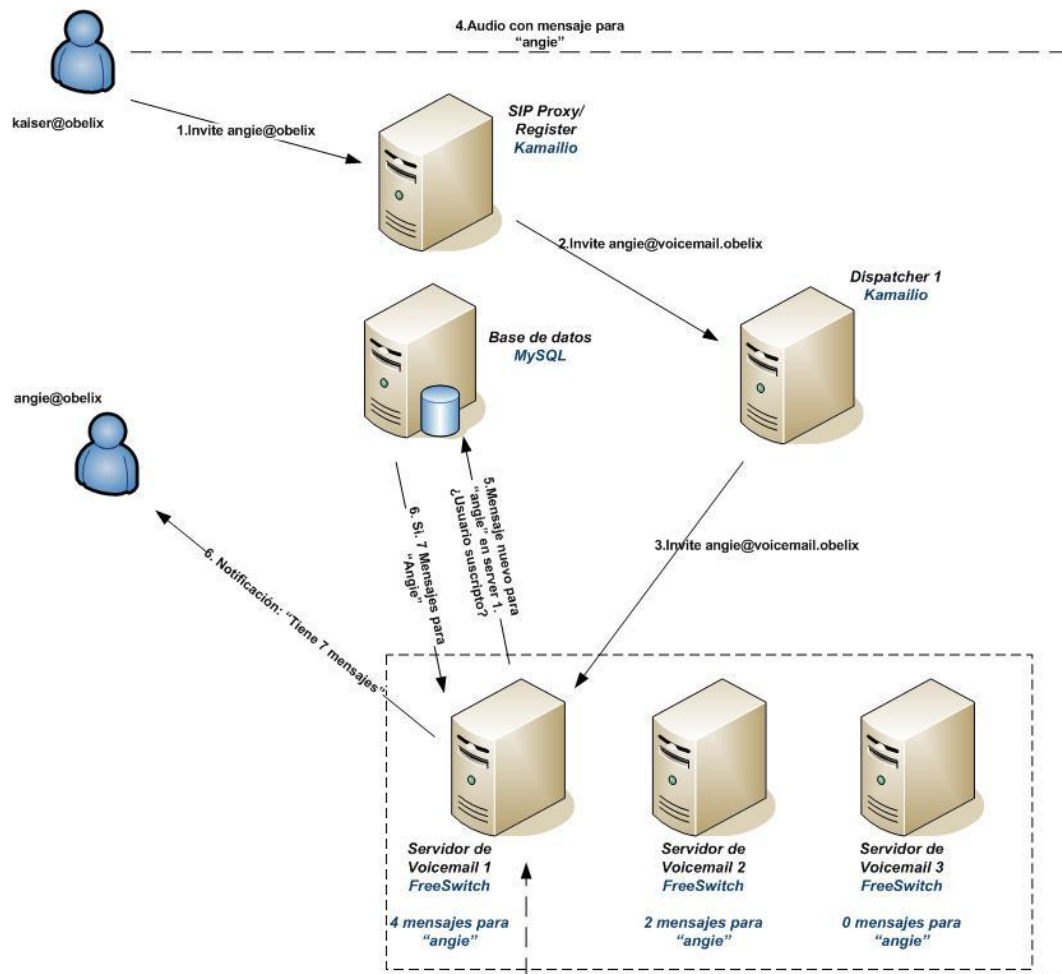


Figura 7. 10 – Esquema 2 de funcionamiento de suscripciones y notificaciones de MWI con HA

De esta manera, también se tiene alta disponibilidad en esta sub-función del servicio de correo de voz.

7.3 – Puente de conferencias

Para dar alta disponibilidad al servicio de puentes de conferencias, como en los casos anteriores, es necesaria la colocación de otro servidor. Sin embargo el esquema de los casos anteriores es imposible de implementar.

En las funcionalidades tratadas anteriormente, todas las nuevas llamadas que llegaban a los servidores eran aleatoriamente balanceadas hacia cualquier servidor indistintamente. En el caso de los puentes de conferencias todas las llamadas destinadas a una misma sala deben necesariamente ser atendidas por el mismo servidor, de lo contrario puede ocurrir un desencuentro entre los participantes de una sala de conferencias.

La forma más sencilla de sortear esto es que los servidores de despacho realicen un ruteo estático dependiendo de la sala que se quiere acceder. Esto es que cada servidor tenga una lista de salas asociadas disponibles (figura 7.11).

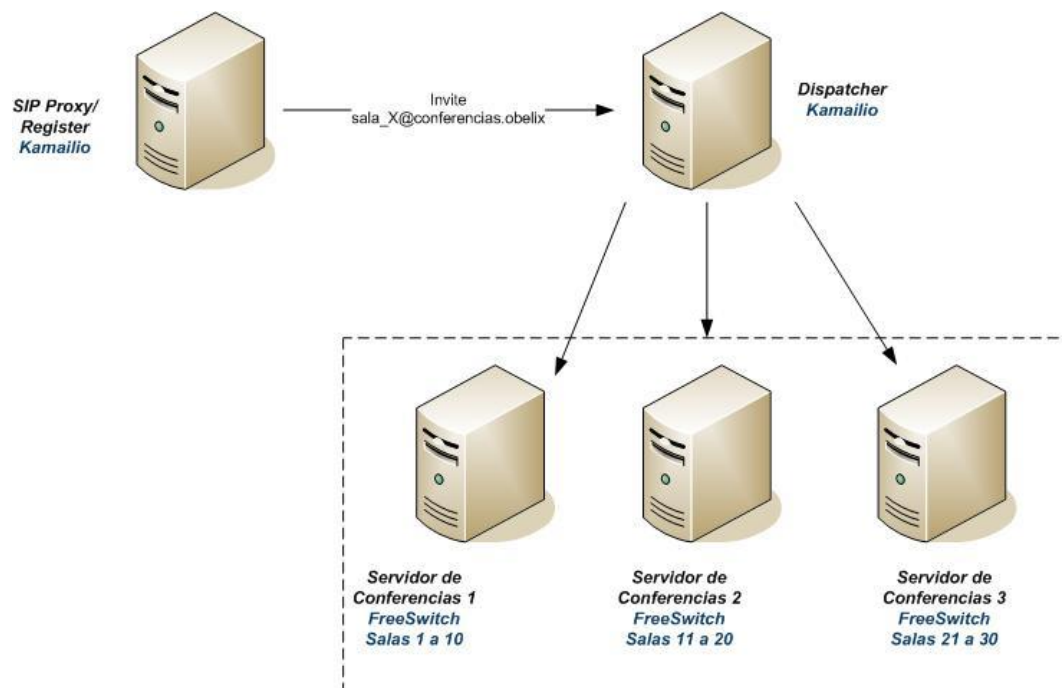


Figura 7. 11 – Esquema de conferencias con salas estáticas

De esta forma se logra que evitar posibles desencuentros entre los participantes de una sala. Sin embargo, esto no representa un esquema de alta disponibilidad del servicio ya que la caída de un servidor de conferencias deja inhabilitadas todas sus salas correspondientes.

Para solucionar este problema, se pueden definir todas las salas en ambos servidores pero continuar realizando el ruteo estático por interno de sala. El modulo de despacho debe estar preparado para cuando detecte la caída de un servidor, despachar al siguiente en una lista. De esta manera, al caer un servidor

y ser detectado por el modulo de despacho, este modulo despachara todas las llamadas al siguiente servidor y este ultimo será capaz de levantar las salas que correspondían al servidor caído. Esta solución se ilustra en la figura 7.12 en un funcionamiento normal y en la figura 7.13 ante una caída en un servidor.

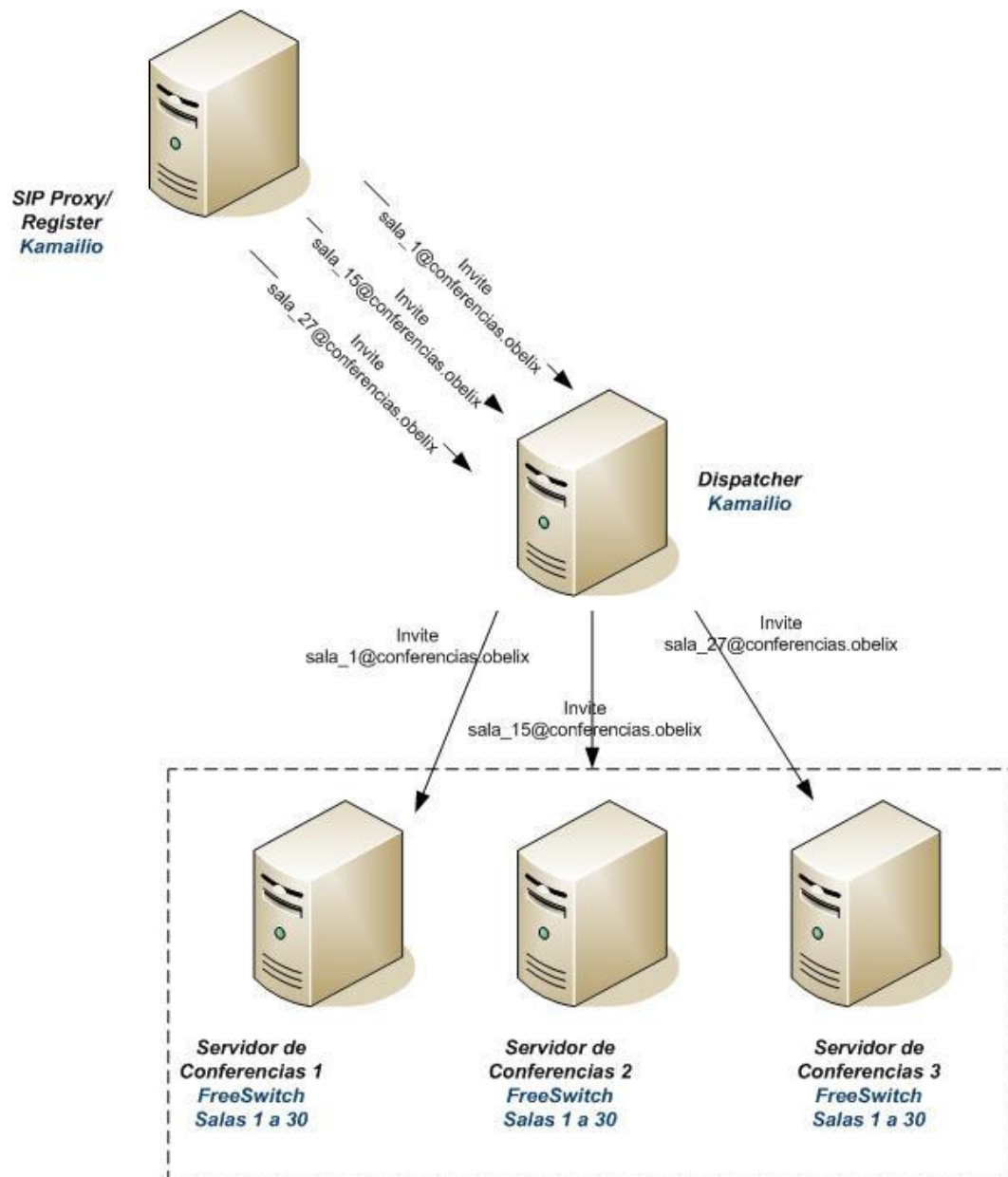


Figura 7. 12 – Esquema de funcionamiento normal de conferencias con salas estáticas y HA

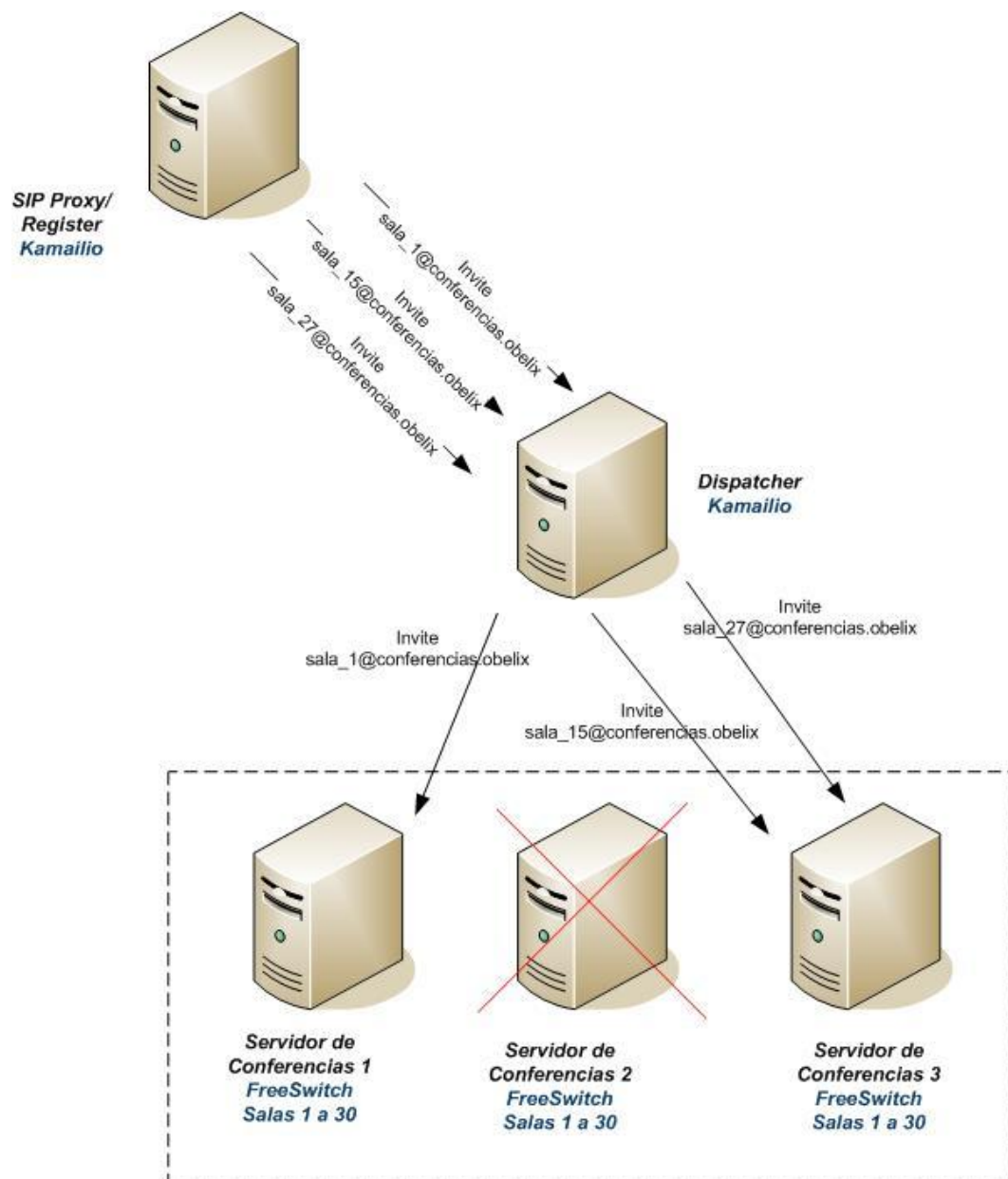


Figura 7. 13 – Esquema de funcionamiento en caso de falla de conferencias con salas estáticas y HA

Con esta solución, además de ser escalable a N servidores, se logra alta disponibilidad y *failover* del servicio de conferencias. Sin embargo esta solución no fue la implementada finalmente.

Si una conferencia esta activa en un servidor cuando este fallare, todos los participantes serán expulsados del sistema, debiendo volver a llamar para entrar en conferencia en el siguiente servidor.

Dada la duración media de una conversación en conferencia, que se supone de entre 20 y 30 minutos, la probabilidad de que caiga un servidor durante el transcurso de esta es mayor, y dada la importancia que supone una conferencia de este tipo es deseable que los participantes no vean afectado el servicio ante la caída de un servidor.

Debido a esto, se diseñó un sistema más complejo que permite la recuperación de todas las salas que estuvieran en curso en un servidor que falla.

7.3.1 – *Failover* en Caliente de Salas de Conferencias

Para lograr el objetivo planteado, se usó una herramienta del propio FreeSwitch que se describe a continuación.

FreeSwitch es capaz de guardar el estado de todas las llamadas que están siendo cursadas en cierto momento en un lugar externo al propio programa, por ejemplo una base de datos. Cuando el software falla, el administrador puede reiniciarlo, y mediante la ejecución de un comando el software obtiene todos los datos de las llamadas que estaban en curso previo a la falla, y realiza re-invites a todos los participantes de esas llamadas colocando a todos nuevamente en línea.

Este mecanismo, así planteado no tiene mayor utilidad dado que el tiempo para reiniciar el software es intolerable por cualquier usuario que hubiera estado cursando una llamada en ese servidor. Sin embargo, este mecanismo de recuperación adquiere especial interés cuando quien recupera las llamadas del servidor que cae no es el mismo servidor sino otro.

Esto es útil cuando se coloca un servidor en *stand-by* de respaldo de un servidor activo. Si el servidor en *stand-by* detectara la caída del servidor principal, y tuviera acceso a la información que este guardaba sobre las llamadas, podría recuperar todas las llamadas de manera inmediata.

La condición única para que este mecanismo funcione de esta manera, es que ambos servidores tengan la misma dirección de red, y los servicios corran en los mismos puertos. Si no la tuvieran, no se podría realizar las renegociaciones con los participantes porque estos detectarían el cambio de servidor y las llamadas no se concretarían. Para funcionar, debe ser transparente para los usuarios, debiendo estos solos recibir un re-invite para cambiar los parámetros de la sesión.

El otro punto interesante de este mecanismo es que esta recuperación puede hacerse por perfil. Un perfil es una parte del programa dedicada a una tarea. Por ejemplo, un servidor con FreeSwitch que implementa música en espera y correo de voz, puede realizar esas tareas en dos perfiles distintos, pudiendo cada perfil atender en una IP y un puerto distinto de la misma máquina. Si ese servidor cayera, el servidor en *stand-by* podría recuperar solo el perfil dedicado al correo de voz y no el otro.

La solución planteada junta estos conceptos de la siguiente manera. En lugar de colocar un servidor puramente en *stand-by* para hacer la recuperación, cada uno de los servidores de la configuración tiene un perfil en *stand-by* mientras tiene otro funcionando con normalidad. Cada perfil en *stand-by* se encuentra pronto para atender en la dirección de red y puerto del perfil activo del servidor de al lado.

Claramente con una configuración así, no puede darse que el servidor que va a recuperar un perfil de otro tenga físicamente la misma dirección de red dado que habría un problema de dirección de red duplicada. Es por esto que el perfil en *stand-by* estará pronto para tomar el lugar de un perfil caído cuando se le dé la directiva, en primer lugar de levantar esa interfaz de red, y luego de realizar la recuperación.

La siguiente figura (figura 7.14) ilustra este esquema implementado con dos servidores.

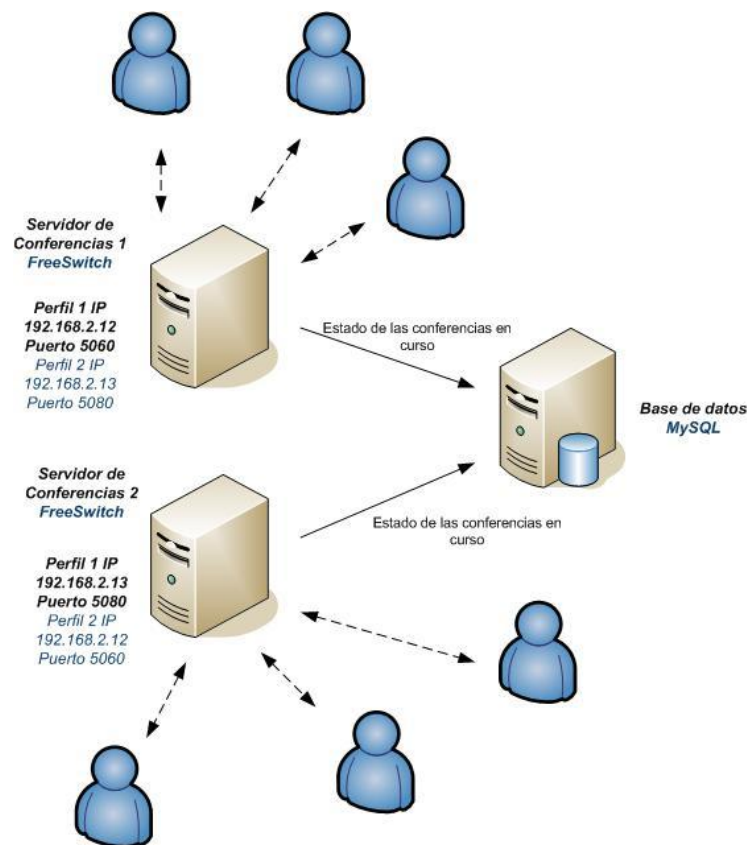


Figura 7. 14 – Esquema de funcionamiento normal de conferencias con salas estáticas por perfil y HA. Información de conferencias guardada en base de datos

Es este esquema, cada uno de los servidores guarda el estado de todas las conferencias que se están cursando en la base de datos del sistema. Cuando un servidor cae, el otro toma su lugar levantando la misma dirección de red y recuperando todos los datos del perfil caído para re-invitar a todos los participantes a sus respectivas conferencias.

Lo que resta es que el servidor que debe hacer la recuperación detecte la caída del otro para tomar las acciones correspondientes.

Para esto vuelve a ser de utilidad el modulo de despacho. Este modulo, que se encuentra “pingueando” con mensajes SIP a los servidores activos para detectar su caída, es quien puede detectar una falla en alguno de los servidores de conferencias. Cuando hay alguna falla, es posible tomar algún tipo de acción como se vio en el capítulo anterior con el modulo Ping-X.

Ante la caída de uno de los servidores, el modulo de despacho envía un mensaje SIP al servidor que debe hacer la recuperación. Este mensaje, dirigido a un

usuario en particular (por ejemplo “recovery@conferencia.obelix”) es procesado en el *dialplan* del servidor destino, en donde la acción de este será levantar la interfaz de red del servidor caído y ejecutar la acción de *recovery*. Esto se ilustra en la figura 7.15.

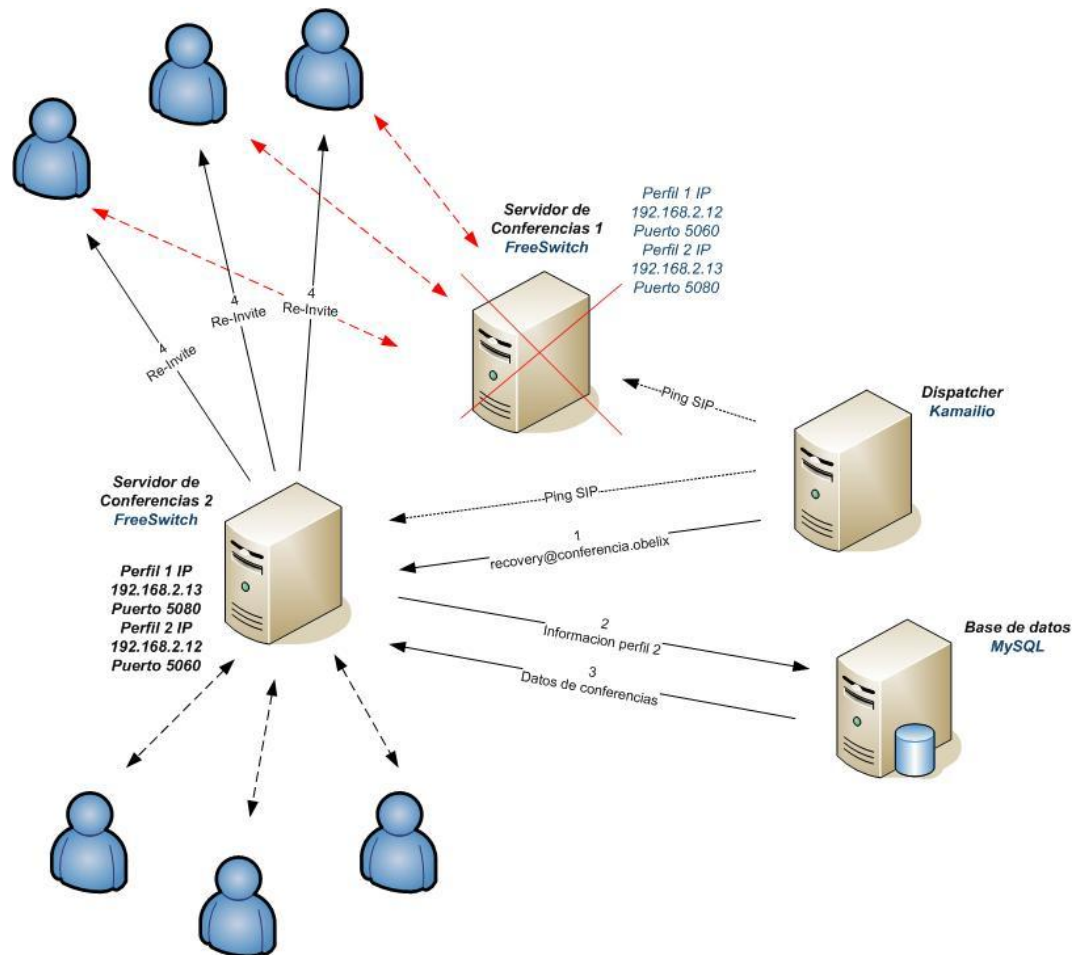


Figura 7. 15 – Esquema de funcionamiento en caso de falla de conferencias con salas estáticas por perfil y HA. Failover en caliente

La demora entre la caída de un servidor y su recuperación depende mayormente de la velocidad con que el modulo de despacho detecte la caída, luego la recuperación es casi instantánea. Esta velocidad está dada por la cadencia en los ping SIP que se halla configurado en el modulo de despacho, así como la tolerancia de mensajes sin respuesta antes de darlo de baja.

Con un tiempo entre pings de un segundo, y una tolerancia uno (esto es que al no responder un mensaje se da inmediatamente de baja), los participantes de

una conferencia ante la caída de un servidor experimentan una pérdida de audio de aproximadamente un segundo antes de volver todos en conferencia.

Un último problema surge de esta configuración. Si el servidor que fallare, lo hiciera solo a nivel de aplicación, esto es que se fallara el software, la recuperación se llevaría a cabo, pero al no perder su dirección de red ante una de estas caídas, su dirección de red quedaría duplicada.

Para evitar esto, se diseñó una aplicación que monitorea la actividad de la aplicación SIP, instalada en cada servidor. Si esta aplicación detectara la caída del servidor a nivel de aplicación, será la encargada de dar de baja su interfaz de red para evitar el problema de dirección duplicada.

Toda la configuración necesaria para lograr esta funcionalidad en el servicio de conferencias puede ser consultada en el anexo 1 (configuración para el módulo de despacho) y en el anexo 2 (configuración en los servidores de conferencia).

Es posible llevar esta solución a N servidores como lo ilustra la figura 7.16, configurando circularmente. Sin embargo, crece en complejidad de configuración y no soporta una doble caída en servidores contiguos, en cuyo caso quedarían sus salas respectivas fuera de línea.

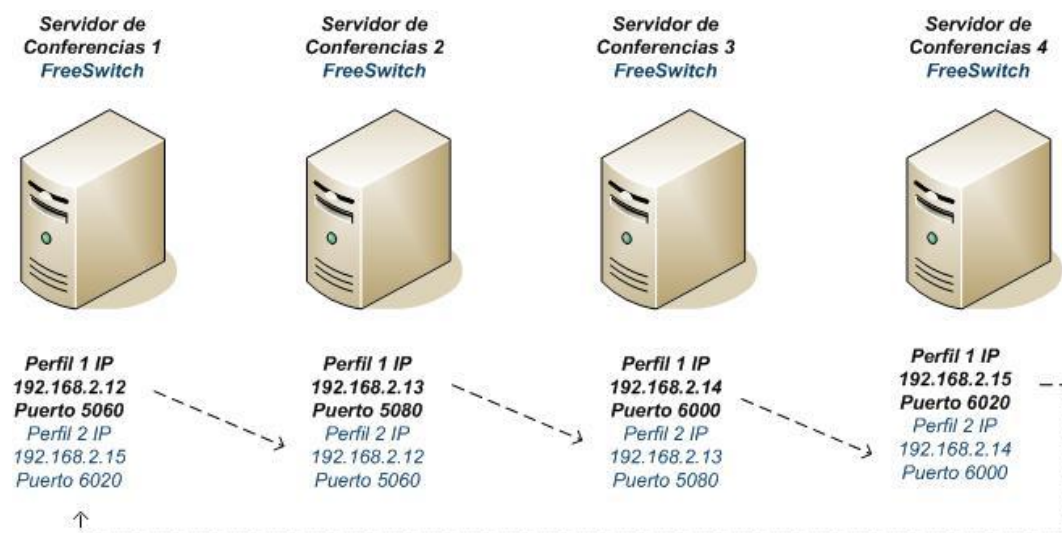


Figura 7. 16 – Ejemplo de la configuración de perfiles activos y de *back-up* en 4 servidores.

7.4 – Variante al modulo de despacho

Si bien el modulo de despacho se implementó con dos servidores independientes, cabe la posibilidad que todas las funcionalidades de este modulo sean realizadas por los mismos servidores Kamailio del núcleo de señalización.

Solo bastaría con cargar el modulo *dispatcher* en ambos servidores y configurar las listas de servidores a pinguear, así como también añadir la lógica en el archivo de configuración de los mismos para que se comporten de la misma forma que los servidores independientes.

De esta manera, la puerta de entrada a los bancos de *features* serán los propios servidores del núcleo y no los servidores de despacho como en la solución planteada.

Realizar esta integración tiene como costo cargar a los servidores del núcleo con la tarea de estar testeando continuamente a los servidores de *features*, pero se conseguiría disminuir en dos el número de servidores de la arquitectura final.

Capítulo 8 – Arquitectura Final y Administración.

En este último capítulo se juntará lo presentado en capítulos anteriores para presentar lo que es la arquitectura final del proyecto. Se comentará también en una sección como se realiza la administración del sistema.

8.1 – Arquitectura final. Obelix.

Habiendo discutido y analizado las distintas posibilidades para lograr un sistema telefónico de alta disponibilidad capaz de manejar un volumen importante de usuarios, se presenta la arquitectura final del sistema.

Como se comentó en su momento, esta arquitectura puede presentar alguna variante. La que aquí se presenta (figura 8.1) es la arquitectura que resultó implementada y probada en una “maqueta”.

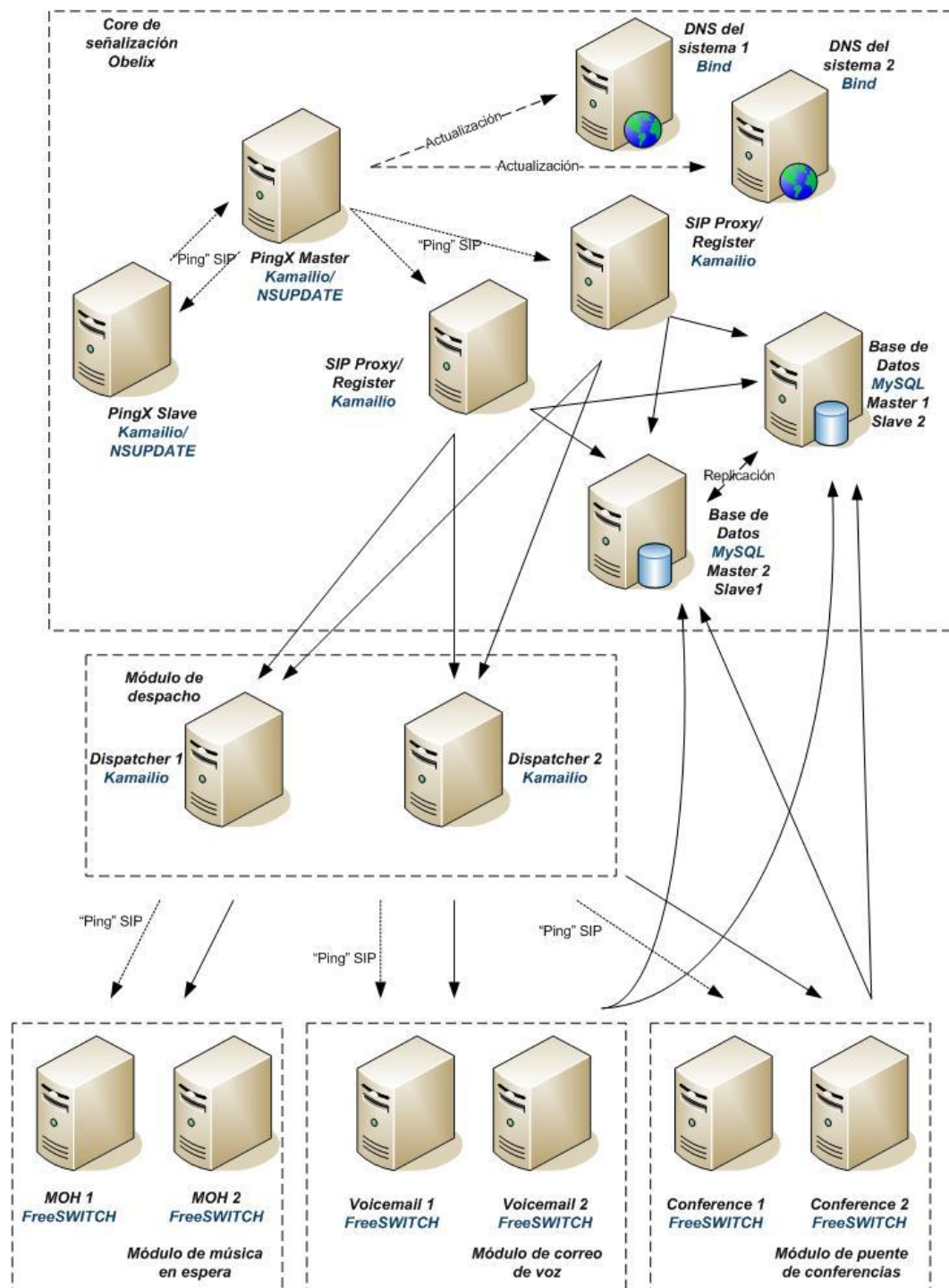


Figura 8. 1 – Arquitectura final de Obelix

La figura 8.1, que ilustra la arquitectura completa del sistema, se compone de dos grandes grupos de servidores. El primero corresponde a lo que se denominó “core de señalización”, y el segundo se compone de todos los servidores que tienen como fin resolver las funcionalidades del sistema que requieren procesamiento de audio (módulo de despacho + módulos de *features*).

El *core* de señalización a su vez se compone de cuatro grupos elementales. Estos son los servidores *Sip Proxy/Registrar*, los servidores de bases de datos, los servidores PingX y los servidores DNS.

Los SIP Proxy/Registrars son los servidores en donde se encuentra la mayoría de la inteligencia del sistema. Este modulo es el encargado de registrar a los usuarios en el sistema, autenticarlos, resolver hacia donde enviar toda la señalización que arriba al sistema entre otros. Estos servidores utilizan el esquema de las bases de datos para mantener todos los datos necesarios para dichas tareas.

Los servidores DNS son los encargados de resolver las direcciones de todos los servidores del sistema además de realizar un balanceo en las consultas de varios de ellos, que resulta en un balance de la carga de estos.

El modulo PingX se encarga de verificar que los servidores *SIP Proxy/Registrars* se encuentren en línea y en caso de detectar una caída actualizar los DNS del sistema para que dejen de propagar la dirección del servidor que fallare.

El segundo grupo de servidores se conforma por el modulo de despacho, encargado de balancear y realizar el *failover* de los servidores de música en espera, correo de voz y conferencias, y por los nombrados servidores de funcionalidades. Toda la mensajería que deba ser dirigida a estos servidores de media, el *core* la dirigirá hacia el modulo de despacho.

Los servidores de correo de voz y de conferencias utilizan las bases de datos del sistema para guardar y consultar los datos necesarios para el funcionamiento de estos servicios.

Los detalles de la construcción de esta arquitectura se pueden consultar en sus secciones correspondientes a lo largo de todo este documento.

8.2 – Administración del Sistema

En esta sección se describe cómo administrar el sistema en cuanto a manejo de usuarios, configuración administrativa de servidores (en cuanto a direcciones de red) y cuidados a tener en cuenta en general.

8.2.1 – Administración de Usuarios

En primer lugar, para que un usuario pueda hacer uso del sistema necesita ser un suscriptor del mismo. Alguien que no es un suscriptor no podrá registrarse en el sistema, y por ende, hacer uso de ninguna funcionalidad del mismo.

8.2.1.1 – Suscripción

Ser un suscriptor implica poseer un nombre de usuario (por ejemplo johnnie) así como también una contraseña para registrarse en el sistema. El administrador deberá añadir a este usuario y contraseña en el sistema (de dominio obelix por ejemplo), y luego de esto el usuario estará listo para registrarse en el mismo utilizando su dirección (johnnie@obelix) y su contraseña.

El agregado de un suscriptor al sistema se realiza mediante la ejecución de un comando en alguno de los servidores que ofician de SIP Registrars. Estos no son otros que los SIP Proxy Kamailio del núcleo de señalización.

Al ejecutar el comando de creación de un suscriptor, el SIP Proxy guardará esta información en la base de datos y es por esto que es indistinto el servidor SIP Proxy que se elija para realizar esta tarea.

Este comando es el siguiente:

```
kamctl add "usuario" "contraseña"
```

Si se desea cambiar la contraseña de un usuario, se puede ejecutar el siguiente comando:

```
kamctl passwd "usuario" "nuevo_password"
```

Si lo que se desea es eliminar un suscriptor, el comando es el siguiente:

```
kamctl rm "usuario"
```

Luego de añadido el suscriptor, este ya tiene acceso al sistema y por ende acceso a utilizar las funcionalidades del mismo.

8.2.1.2 – Manejo de Grupos de Usuarios

El sistema permite agrupar a los suscriptores en grupos de usuarios. Estos grupos son de gran utilidad al momento de aplicar permisos de acceso a alguna funcionalidad del sistema.

Para agregar a un suscriptor a un grupo determinado, debe ejecutarse el siguiente comando en alguno de los SIP Proxy del núcleo (al igual que las suscripciones estas son guardadas en la base de datos):

```
kamctl acl grant "usuario" "grupo"
```

Si lo que se desea es quitarle la suscripción a un grupo a determinado usuario, se debe usar:

```
kamctl acl revoke "usuario" "grupo/s"
```

Para obtener las suscripciones a grupos de un determinado usuario, se puede utilizar:

```
kamctl acl show "usuario"
```

Un punto importante a tener en cuenta es que antes de poder realizar suscripciones de usuarios a grupos, estos grupos deben ser definidos en una variable (ACL_GROUPS) dentro del archivo en el SIP Proxy:

```
/usr/local/etc/kamailio/kamctlrc
```

Esto es para evitar que se añadan usuarios a grupos con errores de tipeo que pueden llevar a errores en la experiencia del usuario.

Si se desea poder realizar suscripciones de usuarios a grupos desde cualquiera de los servidores SIP Proxy, se deberá modificar el comentado archivo en todos ellos.

Luego de organizar a los usuarios que se desee en grupos, pueden tomarse decisiones sobre el acceso de cada usuario a las distintas funcionalidades desde la lógica programada en los SIP Proxy. El modulo *group*¹¹ de Kamailio exporta funciones de utilidad para este propósito.

Este modulo permite consultar el grupo al que pertenece tanto el usuario que llama como el grupo del usuario que es llamado, y así permitir o no que la llamada se concrete.

Del mismo modo, se puede permitir o no el uso de alguna funcionalidad dependiendo del grupo al que pertenece cada usuario.

Según el comportamiento que se desee del sistema, se deberán definir los grupos necesarios y, más importante aún, programar la lógica adecuada en el archivo de configuración de los SIP Proxy. La lógica programada para el uso de los permisos definidos para este proyecto en el capítulo 5 (sección 5.7) se puede consultar en el anexo 1.

8.2.1.3 – Funcionalidades

En cuanto a las suscripciones de usuarios, la única funcionalidad que requiere de configuración extra es el servicio de *voicemail*.

¹¹ La API del modulo "group" se puede consultar en http://www.kamailio.org/docs/modules/3.2.x/modules_k/group.html

Todas las funcionalidades excepto esta última funcionan solamente con que el usuario sea suscriptor del sistema. En el caso del correo de voz, el usuario que deba poseer una casilla debe ser configurado en los servidores de *voicemail*.

Esto se traduce en agregar una entrada para ese usuario en el *dialplan* de todos los servidores de correo. Si no se hiciera, los servidores de correo rechazarían un intento de dejar un mensaje para ese usuario.

Esta entrada en el *dialplan* debe ser realizada en todos los servidores de correo existentes dado que el modulo de despacho elige aleatoriamente el servidor destino.

En esta entrada también se fija el *password* de cada usuario en el servicio de correo, que puede coincidir o no con el *password* de suscripción en el sistema.

Por último, el usuario o número destinado para que los usuarios chequeen sus mensajes de voz debe ser añadido como suscriptor del sistema. De lo contrario al llamar a un número que no está suscrito, la llamada será rechazada. Este usuario o número también se configura en los servidores de *voicemail* para esa tarea.

Como realizar estas configuraciones se puede consultar en el anexo 2.

8.2.2 – Administración de Servidores

8.2.2.1 – Configuración a Nivel de Red

Para el correcto funcionamiento del sistema, hay servidores que deben ser configurados dependiendo de las direcciones de red escogidas para otros servidores. Esto es importante por si en algún momento se decide cambiar la dirección de red de algún servidor, el cambiar solamente la entrada en el DNS puede no ser suficiente.

En primer lugar, todos los servidores del sistema deben tener configurado como DNS al DNS del sistema. Este DNS debe contener entradas para resolver los nombres de todos los servidores del sistema.

Para que el modulo PingX logre su cometido, sus listas de servidores (a los cuales testean) deben ser configuradas conociendo las IP de los servidores SIP Proxy del núcleo. Esto es porque si un servidor SIP Proxy del núcleo falla, al conocer su IP podrá realizar correctamente la actualización del servidor DNS.

Para lograr que se recuperen las conferencias de se cursaban en un servidor que falla, se debe configurar el perfil inactivo del servidor de *back-up* para que esté listo para atender en la dirección IP del servidor caído.

Luego de estos cuidados, todas las configuraciones en el resto de los servidores pueden realizarse con los nombres de los servidores y ser resueltos por el DNS sin ningún problema.

8.2.2.2 – Cuidados en la Administración del Sistema

Existen ciertos cuidados a tener en cuenta en el uso del sistema propuesto y se describen a continuación.

Si un servidor SIP Proxy sale de servicio por algún motivo, cuando vuelva en línea deben cumplirse dos condiciones.

La primera es que el modulo PingX se encuentre funcionando. Si no fuera así, y el modulo PingX se encendiera luego de la puesta en línea del servidor caído, la actualización del DNS agregando nuevamente la dirección del servidor puesto en línea no sucedería. Esto es porque al encenderse el modulo PingX, este asume que los servidores que se testean se encuentran por defecto en línea. Para generar una actualización, el modulo PingX debe detectar un flanco en el estado del servidor SIP Proxy.

La segunda condición es que cuando vuelva en línea el servidor SIP Proxy, el DNS debe encontrarse disponible. Si el orden de los sucesos fuera, baja servidor SIP Proxy - baja DNS - SIP proxy en línea - DNS en línea, el servidor DNS no recibiría el alta del servidor SIP Proxy por parte del modulo PingX.

El otro cuidado que debe tener el administrador del sistema es la puesta en línea nuevamente de un servidor de conferencias.

Cuando cae un servidor de conferencia, otro toma su lugar a nivel de red automáticamente. Por este motivo, antes de poner en línea un servidor caído debe desactivarse el servidor que oficia de *back-up* y regresado a su estado original. Esto es un problema dado que ese servidor puede estar cursando conferencias.

El administrador deberá en primer lugar tener especial cuidado con esto, y comprobar que no se están cursando conferencias en ese momento antes de llevar el servidor de *back-up* a su estado inicial.

En segundo lugar, sacar de línea un servidor dispara automáticamente el mecanismo de recuperación haciendo que otro servidor tome su lugar. Si no se tiene cuidado con esto, el administrador terminará sacando todos los servidores de conferencias para volver a poner en línea el servidor que falló.

Para evitar esto, el administrador deberá desactivar momentáneamente el “pingueo” por parte de los servidores de despacho debido a que este modulo es el responsable de disparar el mecanismo de recuperación.

Para lograr esto, debe ingresar al archivo lista de los servidores y comentar los servidores en los que se va a trabajar (caído y su *back-up*) y luego ejecutar el siguiente comando (debe hacerse en ambos servidores de despacho):

```
sercmd dispatcher.reload
```

Luego de esto, el administrador puede volver a su estado inicial el servidor de back-up y poner en línea el servidor caído.

Luego de realizado esto, se debe volver a activar la función de “pingueo” del modulo de despacho.

Para esto, ingresar al archivo de lista de estos servidores y des-comentar los servidores comentados anteriormente. Luego ejecutar en ambos servidores el mismo comando anterior:

```
sercmd dispatcher.reload
```

Otro cuidado que se desprende de este mismo es el de encender todos los servidores de conferencia antes de activar la función de “pingueo” del modulo de despacho, dado que si no se hiciera provocaría que otro servidor tomara el lugar del que aún no se ha encendido.

Conclusiones

Existe una gran diversidad de herramientas de código abierto en el mundo de la telefonía sobre IP, en particular aquellos que utilizan como protocolo base a SIP. Muchos de estos con un gran desarrollo actual y soporte. Herramientas tales como Kamailio y FreeSWITCH utilizadas en el desarrollo de este proyecto, ambas con comunidades en exponencial crecimiento, proveen mas allá de sus funcionalidades base herramientas para la construcción de nuevas funcionalidades y son lo suficientemente flexibles para que el usuario pueda adaptarlas a sus requerimientos.

En cuanto a SIP, este protocolo multiuso en materia de manejo de señalización de sesiones de cualquier índole procura perfectamente lo necesario para el manejo de señalización en telefonía, permitiendo también añadir otras funciones sobre la señalización que las clásicas de telefonía. Este protocolo permitió manejar de manera centralizada el comportamiento del sistema telefónico completo, además de lograrlo para enormes volúmenes de tráfico dado el bajo costo del manejo único de señalización.

Lograr la separación de lo que son funcionalidades, que requieren manejo de audio, del manejo de la señalización fue lo que permitió cumplir los objetivos y lograr que el sistema escale a un número varios ordenes mayores en materia de usuarios y trafico que los propuestos. Separar en módulos la arquitectura permite en un futuro o bien agregar nuevos módulos con nuevas funcionalidades, actualizar módulos existentes sin afectar el resto de las funcionalidades y permite un dimensionamiento de servidores dependiendo del volumen de usuarios, y pruebas de carga realizadas y analizadas, para cada funcionalidad. De esta manera se evita sobredimensionar servidores que implica un desperdicio de la inversión, dado que es muy difícil dimensionar servidores *multi-feature*.

Con una arquitectura como la propuesta, si bien el número de equipos es elevado, esto no se debe al requerimiento elevado de tráfico del proyecto, sino que más de la mitad de estos equipos surgen de la necesidad de brindar alta disponibilidad en el sistema. Ya sin alta disponibilidad en el sistema, el requerimiento de volumen de usuarios y por lo tanto de tráfico quedó varios órdenes por debajo del resultado alcanzado.

Una arquitectura de este tipo es capaz de continuar escalando para brindar el servicio a un mayor volumen de usuarios. También se puede hacer uso de esta bondad con el fin de colocar servidores muy baratos, tantos como se necesite, para cualquiera de las funciones. Con esto se evitan inversiones en servidores de gran porte y por ende costosos, y se pueden utilizar maquinas recicladas o incluso virtuales para colmar las necesidades del cliente.

En cuanto a las pruebas de carga, estas deben ser realizadas para cada servidor que se desee conocer su performance. Es parte del espíritu del proyecto dejar sentadas las bases para realizar estas pruebas así como el análisis para dimensionar estos servidores.

En una arquitectura de estas dimensiones es recomendable que los servidores de base de datos sean buenos con respecto al resto para evitar tener un cuello de botella en ese modulo.

Referencias

- [1] IETF. (2012) <http://tools.ietf.org/html/rfc3261>.
- [2] Alan B. Johnston, *SIP: Understanding the Session Initiation Protocol*, Third Edition ed., 2009.
- [3] (2012) http://www.cs.columbia.edu/~hgs/papers/others/SIP_Servlet_Architecture_Specification2.pdf.
- [4] IETF. (2012) <http://tools.ietf.org/html/rfc3261#page-12>.
- [5] IETF. (2012) <http://tools.ietf.org/html/rfc3261#page-213>.
- [6] IETF. (2012) <http://tools.ietf.org/html/rfc4538>.
- [7] IETF. (2012) <http://tools.ietf.org/html/rfc3856>.
- [8] IETF. (2012) <http://tools.ietf.org/html/rfc3428>.
- [9] IETF. (2012) <http://tools.ietf.org/html/rfc3261#section-11.1>.
- [10] IETF. (2012) <http://tools.ietf.org/html/rfc3265#section-3.1>.
- [11] IETF. (2012) <http://tools.ietf.org/html/rfc3265#page-14>.
- [12] IETF. (2012) <http://tools.ietf.org/html/rfc3903#section-4>.
- [13] IETF. (2012) <http://tools.ietf.org/html/rfc3515#section-2>.
- [14] IETF. (2012) <http://tools.ietf.org/html/rfc3428#section-3>.
- [15] IETF. (2012) <http://tools.ietf.org/html/rfc2976#section-2>.
- [16] IETF. (2012) <http://tools.ietf.org/html/rfc3311#section-3>.
- [17] Andrew S. Tanenbaum, *Redes de Computadores*, Tercera Edicion ed., 2003.
- [18] IETF. (2012) <http://tools.ietf.org/html/rfc3261#section-20>.
- [19] IETF. (2012) <http://tools.ietf.org/html/rfc1123>.
- [20] ITU-T. <http://www.itu.int/rec/T-REC-G.722/es>.
- [21] ITU-T. <http://www.itu.int/rec/T-REC-G.723/es>.
- [22] ITU-T. <http://www.itu.int/rec/T-REC-G.726/es>.
- [23] ITU-T. <http://www.itu.int/rec/T-REC-G.729/es>.
- [24] IETF. (2012) <http://tools.ietf.org/html/rfc3951>.

- [25] Wikipedia. (2012) <http://en.wikipedia.org/wiki/ILBC>.
- [26] IETF. (2012) <http://tools.ietf.org/html/rfc5574>.
- [27] IETF. (2012) <http://tools.ietf.org/html/rfc5215>.
- [28] Wikipedia. (2012) <http://en.wikipedia.org/wiki/MP3>.
- [29] Wikipedia. (2012) [http://en.wikipedia.org/wiki/Opus_\(audio_format\)](http://en.wikipedia.org/wiki/Opus_(audio_format)).
- [30] (2012) <http://searchunifiedcommunications.techtarget.com/feature/Asterisk-SCF-Enterprise-class-open-source-VoIP-and-UC>.
- [31] (2012) <http://www.sinologic.net/blog/2010-10/asterisk-scf-el-futuro-de-las-comunicaciones-open-source/>.
- [32] Digium. (2012) <https://wiki.asterisk.org/wiki/display/TOP/Goals>.
- [33] Digium. (2012) <http://lists.digium.com/pipermail/asterisk-dev/2012-September/056847.html>.
- [34] (2012) <http://www.venturevoip.com/news.php?rssid=2691>.
- [35] Elastix. (2012) <http://www.elastix.org/index.php/en/company/news/783-elastix230.html>.
- [36] Wikipedia. http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol.
- [37] voip-info.org. (2012) <http://www.voip-info.org/wiki/view/CallWeaver>.
- [38] voip-info.org. (2012) <http://www.voip-info.org/wiki/view/OpenPBX.org+FAQ>.
- [39] (2012) http://www.gnutelephony.org/index.php/GNU_SIP_Witch.
- [40] (2010) ftp://ftp.iptel.org/pub/sems/sayer_sems_mar10.pdf.
- [41] (2012) <http://www.iptel.org/files/semsng-designoverview.pdf>.
- [42] (2012) https://community.jboss.org/wiki/MobicentsJAINSLLEEContainerArchitecture?_sscc=t.
- [43] (2012) <http://www.mobicents.org/mms/mms-main.html>.
- [44] (2012) <http://www.mobicents.org/sip-presence/intro.html>.
- [45] (2012) <http://confluence.cipango.org/pages/viewpage.action?pageId=2261054>.
- [46] (2012) <http://confluence.cipango.org/display/DOC/Architecture>.
- [47] (2012) <http://sip-router.org/history/>.
- [48] (2012) <http://icanblink.com/blink-pro.phtml>.
- [49] Wikipedia. (2012) <http://en.wikipedia.org/wiki/Ekiga>.
- [50] Jitsi. (2012) <https://jitsi.org/index.php/Main/Features>.

- [51] Wikipedia. (2012) <http://en.wikipedia.org/wiki/KPhone>.
- [52] (2012) <http://www.linphone.org/eng/features/>.
- [53] (2012) <http://en.wikipedia.org/wiki/QuteCom>.
- [54] (2012) <http://sflphone.org/features>.
- [55] (2012) <http://www.twinklephone.com/>.
- [56] Antony Minessale II. (2012) <http://www.freeswitch.org/node/117>.
- [57] Digium. (2012) <https://wiki.asterisk.org/wiki/display/AST/ConfBridge+10>.
- [58] ITU Sr. H. Leijon. Curso de Núcleo de Red de Facultad de Ingeniería - Udelar. [Online]. http://iie.fing.edu.uy/cursos/file.php/165/Teoria_teletrafico_material_complementario/Teoria_teletrafico_-_ITU.pdf
- [59] ITU Sr. H. Leijon. ITU. [Online]. http://www.itu.int/itudoc/itu-d/dept/psp/ssb/planitu/plandoc/ttt_cs-es.pdf
- [60] IETF. (2012) <http://tools.ietf.org/html/rfc5359#page-38>.
- [61] (2012) <http://sofia-sip.sourceforge.net/>.
- [62] IETF. (2012) <http://tools.ietf.org/html/rfc3725>.
- [63] (2011) <http://www.onsip.com/blog/2011/05/17/music-on-hold-and-the-sip-offeranswer-model-what-were-we-thinking>.
- [64] James Harry Green, *The Irwin Handbook of Telecommunications Management*, pag. 516, 517 tabla 24, Third Edition ed., 2001.
- [65] Hadeel S. Taha, *Architecture for a SIP-Based Conferencing Server*.
- [66] Microsoft Corporation. (2012) <http://technet.microsoft.com/en-us/library/gg398811.aspx>.
- [67] IETF. (2012) <http://tools.ietf.org/html/rfc5057>.
- [68] IETF. <http://tools.ietf.org/html/rfc2778>.
- [69] IETF. (2012) <http://tools.ietf.org/html/rfc3265>.
- [70] (2012) http://www.kamailio.org/docs/modules/3.2.x/modules_k/presence.html.
- [71] IETF. (2012) <http://tools.ietf.org/html/rfc1794>.
- [72] IETF. (2012) <http://www.ietf.org/rfc/rfc2136.txt>.
- [73] (2006) <http://onlamp.com/onlamp/2006/04/20/advanced-mysql-replication.html>.
- [74] MySQL. (2012) <http://dev.mysql.com/doc/refman/5.0/es/ndbcluster.html>.
- [75] FreeSWITCH. http://wiki.freeswitch.org/wiki/Installation_Guide#Release.28es.29_6_and_Later.

[76] FreeSWITCH. <http://jira.freeswitch.org/browse/FS-4396>.

[77] FreeSWITCH. <http://jira.freeswitch.org/browse/FS-4316>.

[78] FreeSWITCH. <http://jira.freeswitch.org/browse/FS-4291>.

[79] (2011) <http://www.onsip.com/blog/2011/05/17/music-on-hold-and-the-sip-offeranswer-model-what-were-we-thinking>.

Índice de Imágenes

FIGURA 2.1 - FLUJO SIP DE REGISTRACIÓN	46
FIGURA 2.2 - FLUJO SIP DE ESTABLECIMIENTO DE LLAMADA	47
FIGURA 2.3 - FLUJO SIP DE ESTABLECIMIENTO DE LLAMADA MEDIANTE PROXY	48
FIGURA 4. 1 – ESQUEMA 1 DE PRUEBA DE LLAMADAS A B2BUA	96
FIGURA 4. 2 – ESQUEMA 2 DE PRUEBA DE LLAMADAS A B2BUA	96
FIGURA 4. 3 – ESQUEMA 1 DE PRUEBA DE LLAMADAS A B2BUA SIN PROCESAR AUDIO	99
FIGURA 4. 4 – ESQUEMA 2 DE PRUEBA DE LLAMADAS A B2BUA SIN PROCESAR AUDIO	99
FIGURA 4. 5 – ESQUEMA 1 DE PRUEBA DE LLAMADAS A SIP PROXY	101
FIGURA 4. 6 – ESQUEMA 2 DE PRUEBA DE LLAMADAS A SIP PROXY	101
FIGURA 4. 7 – ESQUEMA 1 DE PRUEBA DE REGISTROS A SIP PROXY	102
FIGURA 4. 8 – ESQUEMA 2 DE PRUEBA DE REGISTROS A SIP PROXY	103
FIGURA 4. 9 – CAPTURA DEL COMANDO “TOP” EN SIP PROXY CON BASE DE DATOS EN MISMO HOST EN PRUEBA DE REGISTROS.....	103
FIGURA 4. 10 – CAPTURA DEL COMANDO “TOP” Y USO DE ANCHO DE BANDA EN GENERADOR 1	106
FIGURA 4. 11 – CAPTURA DEL COMANDO “TOP” Y USO DE ANCHO DE BANDA EN GENERADOR 2	106
FIGURA 4. 12 – CAPTURA DEL COMANDO “TOP” Y USO DE ANCHO DE BANDA EN GENERADOR 3	107
FIGURA 4. 13 – CAPTURA DEL COMANDO “TOP” Y USO DE ANCHO DE BANDA EN GENERADOR EN BASE DE DATOS	107
FIGURA 4. 14 – CAPTURA DEL COMANDO “TOP” Y USO DE ANCHO DE BANDA EN SIP PROXY	108
FIGURA 4. 15 – ESQUEMA DE LA PRIMERA PARTE DE LA PRUEBA DE LLAMADAS	109
FIGURA 4. 16 – ESQUEMA DE LA SEGUNDA PARTE DE LA PRUEBA DE LLAMADAS	110
FIGURA 4. 17 – GRAFICO DE “PROCESAMIENTO VS. LLAMADAS POR SEGUNDO”	111
FIGURA 4. 18 – GRAFICO DE “PROCESAMIENTO VS. LLAMADAS POR SEGUNDO” CON ARCHIVO DE CONFIGURACIÓN ESTÁNDAR	113
FIGURA 4. 19 – GRAFICO DE “MEMORIA CONSUMIDA VS. LLAMADAS CONCURRENTES	115
FIGURA 4. 20 – GRAFICO DE “CANTIDAD DE USUARIOS DEL SISTEMA VS. TIEMPO DE PERMANENCIA”. ZONA SOPORTADA	120
FIGURA 4. 21 – ARQUITECTURA DEL NÚCLEO DE SEÑALIZACIÓN.....	122
FIGURA 5. 1 - CAPTURA DE ESCENARIO EN TERMINAL	129
FIGURA 5. 2 - GRÁFICO DE CANALES SIN DISTORSIÓN EN FUNCIÓN DE LA CADENCIA DE LLAMADAS Y CANALES TOTALES	134
FIGURA 5. 3 - PORCENTAJE DE CPU EN FUNCIÓN DE LA CADENCIA Y LLAMADAS TOTALES (CONCURRENTES).	135
FIGURA 5. 4 - PORCENTAJE DE CPU UTILIZADO EN FUNCIÓN DE LAS LLAMADAS CONCURRENTES	136
FIGURA 5. 5 - PORCENTAJE DE CPU UTILIZADO EN FUNCIÓN DE LAS LLAMADAS CONCURRENTES	137
FIGURA 5. 6 - MEMORIA UTILIZADA EN FUNCIÓN DE LA CADENCIA DE LLAMADAS Y LOS CANALES TOTALES.....	139
FIGURA 5. 7 - MEMORIA UTILIZADA EN FUNCIÓN DE LOS CANALES TOTALES	140
FIGURA 5. 8 - DIAGRAMA DE FLUJO DEL CASO DE USO DE ÉXITO DEL ESCENARIO MUSIC ON HOLD.....	147
FIGURA 5. 9 - CAPTURA DE PANTALLA DEL ESCENARIO SIPP GENERADO	157
FIGURA 5. 10 - PORCENTAJE DE USO DEL CPU EN FUNCIÓN DE LAS LLAMADAS CONCURRENTES	159
FIGURA 5. 11 - CANTIDAD DE MEMORIA RAM UTILIZADA EN FUNCIÓN DE LAS LLAMADAS CONCURRENTES	159
FIGURA 5. 12 BOB LLAMA A ALICE PERO ESTA NO ESTÁ REGISTRADA	161
FIGURA 5. 13 BOB LLAMA A ALICE, ESTE RECIBE BUSY DE PARTE DE ALICE	162
FIGURA 5. 14 BOB SE SUBSCRIBE A MWI	163
FIGURA 5. 15 BOB CHEQUEA SU CASILLA DE VOICEMAIL	164

FIGURA 5. 16 - DESCRIPCIÓN GRÁFICA DEL MIXER.....	166
FIGURA 5. 17 - SERVIDOR CENTRALIZADO DE CONFERENCIAS	166
FIGURA 5. 18 - DIAGRAMA DE FLUJO CORRESPONDIENTE A USUARIOS INGRESANDO A UNA MISMA SALA DE CONFERENCIA	169
FIGURA 5. 19 - PORCENTAJE DE UTILIZACIÓN DEL CPU EN FUNCIÓN DE LA CANTIDAD DE USUARIOS EN CONFERENCIA	173
FIGURA 5. 20 - UTILIZACIÓN DE MEMORIA RAM EN FUNCIÓN DE LA CANTIDAD DE USUARIOS EN CONFERENCIA....	174
FIGURA 5. 21 - DIAGRAMA DE FLUJO SIP DE TRANSFERENCIA DESATENDIDA O CIEGA.....	181
FIGURA 5. 22 - DIAGRAMA DE FLUJO SIP CORRESPONDIENTE A LA TRANSFERENCIA ATENDIDA.....	182
FIGURA 5. 23 - DESCRIPCIÓN DE INTERCAMBIO DE MENSAJES DE PRESENCIA	187
FIGURA 5. 24 - INTERCAMBIO DE MENSAJERÍA INSTANTÁNEA.....	191
FIGURA 6. 1 – REDUNDANCIA SIP PROXY	200
FIGURA 6. 2 – CASO DE FALLA “MODO 2”	202
FIGURA 6. 3 – FUNCIONAMIENTO “MODO 3”	203
FIGURA 6. 4 – ESQUEMA 1 DE LA PRUEBA DE REGISTROS EN “MODO 3”	203
FIGURA 6. 5 – ESQUEMA 2 DE LA PRUEBA DE REGISTROS EN “MODO 3”	204
FIGURA 6. 6 – CAPTURA DEL COMANDO “TOP” Y DE USO DE ANCHO DE BANDA EN SIP PROXY	204
FIGURA 6. 7 - CAPTURA DEL USO DE ANCHO DE BANDA EN LA BASE DE DATOS	205
FIGURA 6. 8 – ESQUEMA 1 DE LA PRUEBA DE LLAMADAS EN “MODO 3”	206
FIGURA 6. 9 – ESQUEMA 2 DE LA PRUEBA DE LLAMADAS EN “MODO 3”	207
FIGURA 6. 10 – ESQUEMA DE ARQUITECTURA CON SERVIDOR DE DESPACHO.....	211
FIGURA 6. 11 – ESQUEMA DE ARQUITECTURA CON SERVIDOR DE DESPACHO Y ALTA DISPONIBILIDAD	213
FIGURA 6. 12 – ESQUEMA DE FUNCIONAMIENTO DE PINGX	217
FIGURA 6. 13 – ESQUEMA DE FUNCIONAMIENTO CON DNS PROPIO	218
FIGURA 6. 14 – ESQUEMA DE FUNCIONAMIENTO NORMAL DE PINGX CON HA.....	220
FIGURA 6. 15 – ESQUEMA DE FUNCIONAMIENTO DE PINGX ANTE FALLA	220
FIGURA 6. 16 – ESQUEMA DE ARQUITECTURA MASTER-SLAVES DE BASES DE DATOS	221
FIGURA 6. 17 – ESQUEMA DE ARQUITECTURA MASTER-MASTER DE BASES DE DATOS.....	222
FIGURA 6. 18 – ESQUEMA DEL PROBLEMA DE UNA ARQUITECTURA MASTER-MASTER EN ANILLO PARA BD	223
FIGURA 6. 19 – ARQUITECTURA DEL NÚCLEO DE SEÑALIZACIÓN CON ESQUEMA MASTER-MASTER EN BD	224
FIGURA 6. 20 – ESQUEMA DE ARQUITECTURA CON DOBLE DNS	225
FIGURA 6. 21 – ESQUEMA DE ARQUITECTURA CON DNS EN MODO MASTER-SLAVE.....	226
FIGURA 6. 22 – ARQUITECTURA FINAL DEL NÚCLEO DE SEÑALIZACIÓN CON HA	228
FIGURA 6. 23 – VARIANTE DE LA ARQUITECTURA DEL NÚCLEO DE SEÑALIZACIÓN CON HA.....	229
FIGURA 6. 24 – ESQUEMA DEL FUNCIONAMIENTO CON N SERVIDORES SIP PROXY EN LA VARIANTE DE LA ARQUITECTURA.....	230
FIGURA 7. 1 – ESQUEMA DE ALTA DISPONIBILIDAD EN LA MÚSICA EN ESPERA	232
FIGURA 7. 2 – ESQUEMA DE FUNCIONAMIENTO DEL MÓDULO DE DESPACHO	233
FIGURA 7. 3 – ESQUEMA DE FUNCIONAMIENTO DEL MÓDULO DE DESPACHO CON HA.....	235
FIGURA 7. 4 – ESQUEMA DE FUNCIONAMIENTO ERRÓNEO DEL MÓDULO DE DESPACHO.....	236
FIGURA 7. 5 – ESQUEMA DE HA EN EL SERVICIO DE VOICEMAIL.....	237
FIGURA 7. 6 – ILUSTRACIÓN DE CASO DE FALLA DEL ESQUEMA PLANTEADO	238
FIGURA 7. 7 – ESQUEMA DE SOLUCIÓN CON CASILLAS DE VOICEMAIL ESTÁTICAS	239
FIGURA 7. 8 – ESQUEMA DE FUNCIONAMIENTO DEL BANCO DE VOICEMAIL CON NFS Y BASE DE DATOS.....	241
FIGURA 7. 9 – ESQUEMA 1 DE FUNCIONAMIENTO DE SUSCRIPCIONES Y NOTIFICACIONES DE MWI CON HA	243
FIGURA 7. 10 – ESQUEMA 2 DE FUNCIONAMIENTO DE SUSCRIPCIONES Y NOTIFICACIONES DE MWI CON HA	244

FIGURA 7. 11 – ESQUEMA DE CONFERENCIAS CON SALAS ESTÁTICAS	245
FIGURA 7. 12 – ESQUEMA DE FUNCIONAMIENTO NORMAL DE CONFERENCIAS CON SALAS ESTÁTICAS Y HA	246
FIGURA 7. 13 – ESQUEMA DE FUNCIONAMIENTO EN CASO DE FALLA DE CONFERENCIAS CON SALAS ESTÁTICAS Y HA	247
FIGURA 7. 14 – ESQUEMA DE FUNCIONAMIENTO NORMAL DE CONFERENCIAS CON SALAS ESTÁTICAS POR PERFIL Y HA. INFORMACIÓN DE CONFERENCIAS GUARDADA EN BASE DE DATOS	250
FIGURA 7. 15 – ESQUEMA DE FUNCIONAMIENTO EN CASO DE FALLA DE CONFERENCIAS CON SALAS ESTÁTICAS POR PERFIL Y HA. FAILOVER EN CALIENTE.....	251
FIGURA 7. 16 – EJEMPLO DE LA CONFIGURACIÓN DE PERFILES ACTIVOS Y DE BACK-UP EN 4 SERVIDORES.....	252
FIGURA 8. 1 – ARQUITECTURA FINAL DE OBELIX.....	255
FIGURA A.2. 1 - DIAGRAMA DE FLUJO SIP AL SERVIDOR DE MÚSICA EN ESPERA	350

Índice de Tablas

TABLA 2.1 – CAMPOS EN MENSAJE SDP	58
TABLA 3. 1 - RESUMEN COMPARATIVO DE B2BUA.....	78
TABLA 3. 2 - RESUMEN COMPARATIVO DE SIP PROXY/SIP REGISTRAR SOFTWARE	83
TABLA 3. 3 - RESUMEN DE SOFTPHONES DEL MERCADO	89
TABLA 4. 1 – TABLA COMPARATIVA DE LLAMADAS CURSADAS EN B2BUA CON Y SIN PROCESAMIENTO DE AUDIO ..	100
TABLA 4. 2 – TABLA DE VALORES EXTRAÍDOS EN SIP PROXY Y BASE DE DATOS EN PRUEBA DE REGISTROS	104
TABLA 4. 3 – TABLA COMPARATIVA DE RENDIMIENTO EN PRUEBA DE REGISTROS CON BASE DE DATOS EN MISMO HOST Y EN DISTINTO HOST	104
TABLA 4. 4 – TABLA CON REVISIÓN DE NIVEL DE PROCESAMIENTO DEL SIP PROXY A DISTINTAS CADENCIAS	111
TABLA 4. 5 – TABLA CON REVISIÓN DE NIVEL DE PROCESAMIENTO DEL SIP PROXY A DISTINTAS CADENCIAS CON ARCHIVO DE CONFIGURACIÓN ESTÁNDAR	113
TABLA 4. 6 – TABLA COMPARATIVA DEL CONSUMO DEL PROCESADOR CON AMBOS ARCHIVOS DE CONFIGURACIÓN	113
TABLA 4. 7 – TABLA CON REVISIÓN DEL USO DE MEMORIA DEL SIP PROXY CURSANDO DISTINTAS CANTIDADES DE LLAMADAS SIMULTANEAS	114
TABLA 5. 1 ANCHO DE BANDA DE SEÑALIZACIÓN EN FUNCIÓN DE CADENCIA DE LLAMADAS.....	132
TABLA 5. 2 LLAMADAS “EFECTIVAS” (EN NEGRITA) EN FUNCIÓN DE LA CADENCIA CPS Y CANTIDAD DE LLAMADAS CONCURRENTES.....	132
TABLA 5. 3 PORCENTAJE DE CPU CONSUMIDO POR LA PBX EN FUNCIÓN DE CPS Y LLAMADAS CONCURRENTES. .	134
TABLA 5. 4 MEMORIA RAM (NEGRITA) EN FUNCIÓN DE CPS Y LLAMADAS CONCURRENTES	138
TABLA 5. 5 RESULTADOS PRUEBA DE CARGA VOICEMAIL.	158
TABLA 5. 6 RESULTADO PRUEBA DE CARGA A SERVIDOR DE CONFERENCIAS.....	172
TABLA 6. 1 – TABLA COMPARATIVA DE PRUEBA DE REGISTROS EN MODOS 2 Y 3	205
TABLA 6. 2 – TABLA COMPARATIVA DE ANCHOS DE BANDA EN LAS PRUEBAS DE REGISTROS EN MODO 2 Y 3	205
TABLA 6. 3 – TABLA COMPARATIVA DE PRUEBA DE LLAMADAS EN MODOS 2 Y 3	207
TABLA 6. 4 – TABLA COMPARATIVA DE ESQUEMAS ACTIVO-PASIVO Y ACTIVO-ACTIVO	209
TABLA A.2. 1 - CARGA DEL PROCESADOR Y MEMORIA CENTOS 5.8	351
TABLA A.2. 2 – CARGA DEL PROCESADOR Y MEMORIA EN CENTOS 6.1	352
TABLA A.3.1 – RESUMEN DE PARÁMETROS DE SIPP	397

Anexo 1 – Instalación y configuración: Kamailio, MySQL, Bind

Esta sección esta dedicada a explicar cómo proceder con la instalación del SIP Proxy Kamailio, así como también a la configuración de los distintos elementos de la arquitectura diseñada que utilizan Kamailio como software base.

Tambien se detallará como lograr la replicación Master-Master de servidores MySQL para la base de datos del sistema, y la instalación y configuración del DNS (BIND) del sistema.

1 – Introducción a la herramienta: Kamailio.

1.1 – Descripción

De la página web de Kamailio¹², se extrae la siguiente descripción:

“Kamailio™ (former OpenSER) is an Open Source SIP Server released under GPL, able to handle thousands of call setups per second. Among features: asynchronous TCP, UDP and SCTP, secure communication via TLS for VoIP (voice, video); IPv4 and IPv6; SIMPLE instant messaging and presence with embedded XCAP server and MSRP relay; ENUM; DID and least cost routing; load balancing; routing fail-over; accounting, authentication and authorization; support for many backend systems such as MySQL, Postgres, Oracle, Radius, LDAP, Redis, Cassandra; XMLRPC control interface, SNMP monitoring. It can be used to build large VoIP servicing platforms or to scale up SIP-to-PSTN gateways, PBX systems or media servers like Asterisk™, FreeSWITCH™ or SEMS. Kamailio and the SIP Express Router (SER) teamed up for the integration of the two applications and new development.”

Kamailio es mucho más que un SIP Proxy. Además de implementar otras funciones de SIP servers como lo puede ser un SIP Registrar o un servidor de

¹² <http://www.kamailio.org/w/>

presencia, Kamailio puede configurarse para tomar decisiones de ruteo sobre la mensajería SIP, manipular y cambiar mensajes SIP, crear mensajes (comportándose como un UA), dejando así al usuario la libertad de diseñar sistemas de señalización complejos basados completamente en SIP.

La versatilidad de esta herramienta permitió resolver todos los problemas que se presentaron durante la realización de este proyecto, haciendo de este software pieza fundamental en la arquitectura final.

1.2 – Introducción al script de configuración de Kamailio.

La configuración del Kamailio se hace enteramente modificando un archivo de configuración de extensión “cfg”. Este último está escrito en un lenguaje propio del programa. Se estudió este lenguaje, con la wiki del programa como referencia de cabecera en la sección de “Cookbooks”, y en particular la sección del núcleo “Core Cookbook”, que se puede encontrar en el sitio web de la wiki de la herramienta¹³.

Entender el lenguaje puede llevar un tiempo considerable como todo nuevo lenguaje, pero no es de los menos amigables, y corre con la ventaja de estar muy bien documentado.

Cuando un mensaje SIP arriba al servidor con Kamailio, este mensaje es procesado de acuerdo a la lógica programada en el archivo de configuración. Mas adelante describiré el archivo de configuración final utilizado en este proyecto, explicando la lógica diseñada para lograr el comportamiento deseado.

El archivo de configuración se divide en siete partes bien diferenciadas, a saber:

1. Sección global: En ella se definen aquellas opciones que influyen directamente en el núcleo de Kamailio: la dirección IP y puerto en el que se mantendrá a la escucha, el nivel de debug, la cantidad de procesos hijos, etc.
2. Sección de módulos: En esta sección se eligen los módulos que se van a cargar. Se hace uso del comando “loadmodule” (Uso: loadmodule “modulo.so”).
3. Sección configuración de parámetros de módulos: Los módulos cargados son personalizados en esta sección mediante el paso de parámetros. Se hace uso del

¹³ <https://www.kamailio.org/wiki/install/3.2.x/git>

comando “modparam” (Uso: modparam(nombre_modulo, parametro_modulo, valor_parametro)).

4. Sección de ruta principal: Es el punto de comienzo en el cual empieza a procesarse una *request* SIP cuando es recibida. En esta sección se indica como va a ser manejada esta petición SIP.

5. Sección ruta secundaria: Se pueden definir rutas secundarias donde, al igual que en la ruta principal, se establece cómo se procesa una determinada petición SIP. El objetivo de definir rutas secundarias es de evitar que la ruta principal sea demasiado grande y compleja. De esta manera se puede definir una ruta secundaria que trate una determinada petición SIP y desde la ruta principal ejecutar el procesamiento de la ruta secundaria.

6. Sección ruta de respuestas: Es posible definir secciones que manejen únicamente las respuestas a las peticiones SIP. Es una sección opcional, ya que si no se define la respuesta se reenvía inmediatamente sin procesarse a su destino.

7. Sección ruta de fallos: En ella se puede realizar un procesamiento concreto para aquellas situaciones de *timeout*, *busy*, etc.

1.3 – Instalación

Se expone aquí una guía para la instalación de Kamailio v3.2 en un sistema operativo CentOS 6.1, el cual es utilizado en todos los elementos de la arquitectura.

1. Instalar dependencias necesarias

```
yum install git-core gcc flex bison make
```

2. Instalar componentes de MySQL para compatibilidad con este motor de bases de datos (Solo para los SIP Proxy del núcleo de señalización)

```
yum install mysql mysql-devel mysql-server
```

3. Instalar dependencias necesarias para los modulos de presencia (Solo para los SIP Proxy del núcleo de señalización)

```
yum install libxml2 libxml2-devel
```

4. Crear y acceder a la carpeta para la instalación

```
mkdir -p /usr/local/src/kamailio-3.2
```

```
cd /usr/local/src/kamailio-3.2
```

5. Descargar las fuentes para la instalación

```
git clone --depth 1 git://git.sip-router.org/sip-router kamailio
```

```
cd kamailio
```

```
git checkout -b 3.2 origin/3.2
```

6. Indicar que “especie” se va a compilar

```
make FLAVOUR=kamailio cfg
```

7. Editar el archivo “modules.lst”, que es donde se indica que modulos se van a compilar y cuales no. Quitar de la variable “exclude_modules” los modulos de interaccion con MySQL y los modulos para el manejo de presencia. (Solo para los SIP Proxy del núcleo de señalización)

```
nano -w modules.lst
```

Quitar de “exclude_modules”: db_mysql, presence, presence_xml, presence_mwi

8. Compilar

```
make all
```

9. Instalar

```
make install
```

10. Para poder usar los binarios desde cualquier ubicación:

```
PATH=$PATH:/usr/local/sbin
```

```
export PATH
```

A continuación se brinda una guía para la creación de la base de datos del sistema en la misma maquina que el Kamailio. Mas adelante se verá como crear dicha base en una maquina host distinta.

1. Setear el password del usuario ‘root’ de MySQL.

- Detener el proceso:

```
/etc/init.d/mysqld stop
```

- Crear un archivo nuevo:

```
nano -w /home/nuevo_password
```

- Escribir en el las siguientes dos lineas:

```
UPDATE mysql.user SET Password=PASSWORD('obelix') WHERE User='root';
```

```
FLUSH PRIVILEGES;
```

- Guardar el archivo (el password seteado en esta guía es 'obelix')

- Correr el servicio con opción de archivo de inicio, para que levante el nuevo password:

```
mysqld_safe --init-file=/home/nuevo_password &
```

- Reiniciar el servicio MySQL:

```
/etc/init.d/mysqld restart
```

2. Configurar el script de creación de la base de datos para funcionar con el motor adecuado y setear las credenciales para escritura y solo lectura en la base de datos.

- Editar el siguiente archivo:

```
nano -w /usr/local/etc/kamailio/kamctlrc
```

- Dejarlo de la siguiente manera: (Se pueden usar otros passwords y nombres de usuarios)

```
## your SIP domain
```

```
SIP_DOMAIN=kamailio
```

```
## database type: MYSQL, PGSQL, DB_BERKELEY, or DBTEXT, by default none is loaded
```

```
# If you want to setup a database with kamdbctl, you must at least specify this parameter.
```

```
DBENGINE=MYSQL
```

```
## database host
```

```
DBHOST=localhost
```

```
## database name
```

```
DBNAME=kamailio
```

```
# database path used by dbtext or db_berkeley
```

```
#DB_PATH="/usr/local/etc/kamailio/dbtext"
```

```
## database read/write user
```

```
DBRWUSER=obelix
```

```
## password for database read/write user
```

```
DBRWPW="obelixrw"
```

```
## database read only user
```

```
DBROUSER=kamailioro
```

```
## password for database read only user
```

```
DBROPW=kamailioro
```

```
## database super user
```

```
DBROOTUSER="root"
```

(El resto permanece incambiado)

3. A continuación, ejecutar el script de creación de la base de datos.

```
/usr/local/sbin/kamdbctl create
```

(Colocar el password del usuario 'root' de MySQL y dar que 'sí' a todo)

La siguiente guía corresponde a la utilizada en la arquitectura del proyecto, en donde la base de datos se encuentra en una maquina host distinta a la del Kamailio.

1. Tener instalado y corriendo el servicio 'mysql' en la maquina host de la base de datos.

```
yum install mysql mysql-devel mysql-server
```

```
service mysqld start
```

2. Conocer el password del usuario 'root' del servicio 'mysql' (Si no se tiene uno, realizar el punto 1 de la guía anterior)

3. Modificar el siguiente archivo en la maquina host de 'mysql':

```
nano /etc/mysql/my.cnf
```

comentando la línea "bind-address" (Esto es para que el servicio no escuche solo en la dirección de red "localhost")

4. Modificar los permisos del usuario 'root' en la maquina host de 'mysql' para poder acceder desde cualquier IP. Esto se hace de la siguiente manera:

- Acceder a mysql como 'root' a la base de datos 'mysql' :

```
mysql -u root -p mysql (Colocar luego el pass del root de mysql)
```

- Ejecutar la siguiente directiva:

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'PASSWORD_DEL_ROOT' WITH GRANT OPTION;
```

- Salir del mysql con "exit"

- Reiniciar el servicio:

```
service mysql restart
```

5. En maquina con Kamilio, realizar el punto 2 de la guía anterior, y además editar la línea que comienza con 'DBHOST' de ese archivo:

```
nano -w /usr/local/etc/kamilio/kamctlrc
```

Editar línea que comienza con 'DBHOST'

```
DBHOST="IP DE MAQUINA CON BASE DE DATOS"
```

6. En maquina con Kamilio, crear la base de datos ejecutando el script de creación:

```
/usr/local/sbin/kamdbctl create
```

Luego de finalizada la instalación del software y creada la base de datos, el servicio se puede poner en marcha mediante el comando:

```
kamilio -P var/run/kamilio.pid -f 'Ruta al archivo de configuración a utilizar' -m "memoria destinada al programa en MBytes"
```

Se puede comprobar que se inició correctamente mediante el comando:

```
kamctl monitor
```

Este comando desplegará en pantalla información sobre el estado del programa actualizándose a cada segundo.

Para obtener una completa lista de variables sobre el sistema, como el uso de memoria y cantidad de transacciones contestadas o simultaneas, se puede hacer uso del siguiente comando:

```
kamctl fifo get_statistics all
```

1.4 – Archivo de configuración de los SIP Proxy del núcleo de señalización.

En esta sección se presenta el archivo de configuración utilizado en los servidores SIP Proxy del núcleo de señalización. Debido a su extensión, el mismo se presenta parcialmente comentado y separado en bloques para facilitar su comprensión y permitir introducir explicaciones de su funcionamiento.

1.4.1 – El código del núcleo de Obelix

El código comienza con una sección preliminar, en donde se definen banderas con la directiva “`#!define`” que sirven para activar o desactivar parte del código escrito.

Por ejemplo, si se encuentra en esta sección la línea “`#!define WITH_MENSAJE`”, una sección de código delimitada por las líneas “`#!ifdef WITH_MENSAJE`” y “`#!endif`” será ejecutado, de lo contrario este código ni siquiera será compilado. También se puede hacer uso de la sentencia “`#!else`” en medio de las dos directivas.

Como se puede ver también, toda línea que comienza solamente con “`#`” es entendida como comentario, así como todo texto entre “`/*`” y “`*/`”.

```
#!KAMAILIO
```

```
# Kamailio (OpenSER) SIP Server v3.2 - Obelix configuration script
```

```
# *** To run in debug mode:
```

```
# -define WITH_DEBUG
```

```
#!define WITH_DEBUG    ##Para loguear en pantalla xlog("texto $algo\n");
```

```
# *** To enable mysql:

#!define WITH_MYSQL    #Habilita la carga del modulo que interactúa con MySQL

# *** To enable authentication execute:

#   - enable mysql

#!define WITH_AUTH      #Habilita el código que autentica a los usuarios

#   - add users using 'kamctl'

# *** To enable persistent user location execute:

#   - enable mysql

#!define WITH_USRLOCDB      #Habilita el uso de la base de datos para la tabla de locaciones

# *** To enable presence server execute:

#   - enable mysql

#!define WITH_PRESENCE      #Habilita el uso como servidor de presencia

# *** To enable database aliases lookup execute:

#   - enable mysql

#!define WITH_ALIASDB      #Habilita la búsqueda por alias en la base de datos

# *** To enable VoiceMail routing execute:

#!define WITH_VOICEMAIL      #Habilita la redirección a servidores de voicemail

#   - set the value of voicemail.srv_ip

#   - adjust the value of voicemail.srv_port

#!define WITH_MOH          #Habilita el uso de servidores de música en espera

#       - To enable music on hold server feature (NOT RFC 5359 compliant)

#!define WITH_CONFERENCE      #Habilita la redirección hacia servidores de puentes de conf.

#       - To enable la redirección a servidor/es de conferencia

#!define WITH_GROUPS    #Habilita el código que maneja permisos por grupos de usuarios

#       - Enable permissions for call routing and features using users memberships

# *** To enhance accounting execute:

#   - enable mysql

#!define WITH_ACCDB          #Habilita el uso de la base para datos de accounting

# *** To enable TLS support execute:

#   - adjust CFGDIR/tls.cfg as needed
```



```
# - define WITH_TLS          #Habilita la seguridad. No utilizado.
```

La siguiente sección define variables globales para configuración, así como también banderas globales para pasar como parámetros. Son para utilizar en la misma configuración inicial y no en la lógica del código. La sintaxis de definición es “`#!define NOMBRE valor`”.

```
##### Defined Values #####
```

```
#!define IPESCUCHA "192.168.2.2"
```

```
#!ifndef WITH_MYSQL
```

```
#!define DBURL "mysql://obelix:obelixrw@base.obelix/kamailio"  #URL para que los modulos
                                                                #que lo necesiten puedan
                                                                #conectarse a la base de
                                                                #datos
```

```
#!endif
```

```
# - flags
```

```
# FLT_ - per transaction (message) flags
```

```
#      FLB_ - per branch flags
```

```
#!define FLT_ACC 1
```

```
#!define FLT_ACCMISSED 2
```

```
#!define FLT_ACCFAILED 3
```

```
#!define FLT_NATS 5
```

```
#!define FLB_NATB 6
```

```
#!define FLB_NATSIPPING 7
```

La sección a continuación define el valor de parámetros globales para el funcionamiento del SIP Proxy. Estos parámetros se pueden pensar como los parámetros del propio *core* del programa¹⁴, a diferencia de otros parámetros como lo son los de los módulos.

```
##### Global Parameters #####
```

```
#!ifndef WITH_DEBUG
```

```
debug=1          #Nivel de debugueo. Cuanto más alto, mayor es la verborragia del programa
```

¹⁴ La lista completa de estos parámetros, así como su descripción se puede consultar en <https://www.kamailio.org/wiki/cookbooks/3.2.x/core>

```

log_stderr=yes

#!else

debug=1

log_stderr=no

#!endif

memdbg=5

memlog=5

log_facility=LOG_LOCAL0

fork=yes      #Si este parámetro es seteado en "yes" el programa correrá en segundo plano

children=16   #Numero de procesos hijos capaces de atender las solicitudes.

reply_to_via=1 #Responde a la IP del último VIA y no a la IP de capa de red que mandó el
               #mensaje, en caso de que no coincidan.

/* add local domain aliases */ #

alias="obelix" #Define un alias para el servidor para que sepa por ejemplo que un mensaje
               #dirigido a "romulo@obelix" debe ser atendido por este servidor

#listen=IPESCUCHA      #Comentado. Si no se setea, el servicio escucha en todas las interfaces
                       #disponibles

port=5060      #Puerto de escucha del servicio. Pueden ser varios.

#!ifdef WITH_TLS

enable_tls=yes

#!endif

# life time of TCP connection when there is no traffic

# - a bit higher than registration expires to cope with UA behind NAT

tcp_connection_lifetime=3605

mhomed=1

use_dns_cache = yes      #Utiliza una cache interna para las resoluciones del DNS.

use_dns_failover = on    #Habilita el failover por DNS

```

La sección a continuación define el valor de variables que pueden ser llamadas desde los bloques de lógica del programa. La sintaxis de estas definiciones son "grupo.identificador="valor" desc "descripción".

Desde los bloques de lógica se pueden llamar estas variables con la directiva “\$sel(cfg_get.grupo.identificador)” o “@cfg_get.grupo.identificador”.

Custom Parameters

#!ifdef WITH_VOICEMAIL

VoiceMail Routing on offline, busy or no answer

voicemail.srv_ip = "voicemail.obelix" desc "VoiceMail IP Address" #IP del voicemail

voicemail.srv_port = "5060" desc "VoiceMail Port" #Puerto del voicemail

#!endif

#!ifdef WITH_MOH

moh.srv_ip = "moh.obelix" desc "MOH Server IP Address" #IP del MOH

moh.srv_port = "5060" desc "MOH Server Port" #Puerto del MOH

#!endif

#!ifdef WITH_CONFERENCE

conference.srv_ip = "conference.obelix" desc "Conference Server IP Address"

conference.srv_port = "5060" desc "Conference Server Port"

#!endif

En la sección a continuación se define la ruta a los directorios donde se encuentran compilados los módulos, y se cargan los módulos que se van a utilizar mediante la directiva “loadmodule “nombre.so””

Modules Section

set paths to location of modules (to sources or installation folders)

#!ifdef WITH_SRCPATH

mpath="modules_k:modules"

#!else

mpath="/usr/local/lib/kamailio/modules_k/:usr/local/lib/kamailio/modules/"

#!endif

#!ifdef WITH_MYSQL

loadmodule "db_mysql.so"

#!endif

loadmodule "mi_fifo.so"

```
loadmodule "kex.so"
loadmodule "tm.so"
loadmodule "tmx.so"
loadmodule "sl.so"
loadmodule "rr.so"
loadmodule "pv.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "siputils.so"
loadmodule "xlog.so"
loadmodule "sanity.so"
loadmodule "ctl.so"
loadmodule "cfg_rpc.so"
loadmodule "mi_rpc.so"
loadmodule "acc.so"

loadmodule "uri_db.so" #Módulo agregado para no permitir que usuarios no suscriptos
                        # al sistema sean redirigidos al VoiceMail.

loadmodule "uac.so"    #Módulo emulador de UAC

loadmodule "dialog.so" #Módulo para manejo de diálogos

loadmodule "avpops.so" # Módulo para poder guardar las AVP's en la base

loadmodule "group.so"  #Módulo agregado para el manejo de grupos de usuarios

#ifndef WITH_AUTH
loadmodule "auth.so"

loadmodule "auth_db.so"

#ifndef WITH_IPAUTH
loadmodule "permissions.so"

#endif
#endif
```

```

#!ifdef WITH_ALIASDB

loadmodule "alias_db.so"

#endif

#!ifdef WITH_MULTIDOMAIN

loadmodule "domain.so"

#endif

#!ifdef WITH_PRESENCE

loadmodule "presence.so"

loadmodule "presence_xml.so"

loadmodule "presence_mwi.so"

#endif

#!ifdef WITH_TLS

loadmodule "tls.so"

#endif

#!ifdef WITH_DEBUG

loadmodule "debugger.so"

#endif

```

En la sección a continuación se customizan los módulos cargados en la sección anterior. Cada módulo tiene sus parámetros propios para lograr distintos comportamientos en las funcionalidades que estos exportan¹⁵. La directiva para setear un parámetro de un módulo es “modparam(“nombre_modulo”, “nombre_parametro”, “valor”)”.

```

# ----- setting module-specific parameters -----

# ----- mi_fifo params -----

modparam("mi_fifo", "fifo_name", "/tmp/kamailio_fifo")

# ----- tm params -----

# auto-discard branches from previous serial forking leg

modparam("tm", "failure_reply_mode", 3)

```

¹⁵ Todos los parámetros de todos los módulos se pueden consultar en <http://www.kamailio.org/docs/modules/3.2.x/>

```
# default retransmission timeout: 30sec
modparam("tm", "fr_timer", 30000)

# default invite retransmission timeout after 1xx: 120sec
modparam("tm", "fr_inv_timer", 120000)
#modparam("tm", "max_inv_lifetime", 10000)

# ----- rr params -----

# add value to ;lr param to cope with most of the UAs
modparam("rr", "enable_full_lr", 1)

# do not append from tag to the RR (no need for this script)
#modparam("rr", "append_fromtag", 0)

# ----- registrar params -----

modparam("registrar", "method_filtering", 1)

/* uncomment the next line to disable parallel forking via location */
# modparam("registrar", "append_branches", 0)

/* uncomment the next line not to allow more than 10 contacts per AOR */
#modparam("registrar", "max_contacts", 10)

# max value for expires of registrations (Modificado para que demoren en vencerse, 0 es que no
exista máximo)
modparam("registrar", "max_expires", 0)

# min value for expires of registrations
modparam("registrar", "min_expires", 1000)

# A ver si se arregla el temita del received...
#modparam("registrar", "path_use_received", 0)

# ----- acc params -----

/* what special events should be accounted ? */
modparam("acc", "early_media", 0)
modparam("acc", "report_ack", 0)
modparam("acc", "report_cancels", 0)

/* by default ww do not adjust the direct of the sequential requests.

if you enable this parameter, be sure the enable "append_fromtag"
```

```

in "rr" module */

modparam("acc", "detect_direction", 0)

/* account triggers (flags) */

modparam("acc", "log_flag", FLT_ACC)

modparam("acc", "log_missed_flag", FLT_ACCMISSED)

modparam("acc", "log_extra",

    "src_user=$fU;src_domain=$fd;src_ip=$si;"

    "dst_ouser=$tU;dst_user=$rU;dst_domain=$rd")

modparam("acc", "failed_transaction_flag", FLT_ACCFAILED)

/* enhanced DB accounting */

#ifndef WITH_ACCDB

modparam("acc", "db_flag", FLT_ACC)

modparam("acc", "db_missed_flag", FLT_ACCMISSED)

modparam("acc", "db_url", DBURL)

modparam("acc", "db_extra",

    "src_user=$fU;src_domain=$fd;src_ip=$si;"

    "dst_ouser=$tU;dst_user=$rU;dst_domain=$rd")

#endif

# ----- usrloc params -----

/* enable DB persistency for location entries */

#ifndef WITH_USRLOCDB

modparam("usrloc", "db_url", DBURL)    #Dirección de la base de datos.

modparam("usrloc", "db_mode", 3)      #Modo 3 para la interacción con la base de datos.
                                      #Explicado en capítulo 6.

modparam("usrloc", "use_domain", MULTIDOMAIN)

#endif

# ----- auth_db params -----

#ifndef WITH_AUTH

modparam("auth_db", "db_url", DBURL)  #Dirección de la base de datos

modparam("auth_db", "calculate_ha1", yes)

```

```
modparam("auth_db", "password_column", "password")

modparam("auth_db", "load_credentials", "")

modparam("auth_db", "use_domain", MULTIDOMAIN)

# ----- permissions params -----

#ifndef WITH_IPAUTH

modparam("permissions", "db_url", DBURL)

modparam("permissions", "db_mode", 1)

#endif

#endif

# ----- alias_db params -----

#ifndef WITH_ALIASDB

modparam("alias_db", "db_url", DBURL)

modparam("alias_db", "use_domain", MULTIDOMAIN)

#endif

# ----- domain params -----

#ifndef WITH_MULTIDOMAIN

modparam("domain", "db_url", DBURL)

# use caching

modparam("domain", "db_mode", 1)

# register callback to match myself condition with domains list

modparam("domain", "register_myself", 1)

#endif

#ifndef WITH_PRESENCE

# ----- presence params -----

modparam("presence", "db_url", DBURL)

# ----- presence_xml params -----

modparam("presence_xml", "db_url", DBURL)

modparam("presence_xml", "force_active", 1)

#endif

#ifndef WITH_TLS
```



```
# ---- tls params ----

modparam("tls", "config", "/usr/local/etc/kamailio/tls.cfg")

#!endif

#ifdef WITH_DEBUG

# ---- debugger params ----

modparam("debugger", "cfgtrace", 0)

#!endif

# Parametros uri_db

modparam("uri_db", "db_url", DBURL )

modparam("uri_db", "db_table", "subscriber")

# Parametros dialog

modparam("dialog", "dlg_flag", 4)

# Parametros AVOops

modparam("avpops", "db_url", DBURL)

modparam("avpops", "avp_table", "usr_preferences")

#Parametros del Group

modparam("group", "db_url", DBURL )

modparam("group", "use_domain", 1)
```

En la sección siguiente se define el bloque principal de la lógica de ruteo. Todas las SIP Requests que lleguen al servidor serán procesadas por el código a continuación. Este bloque comienza con la directiva “request_route” y todo su contenido se encuentra dentro de llaves “{...}”. Dentro de este bloque se pueden invocar otros bloques secundarios, definidos en las secciones siguientes, con la directiva “route(nombre_bloque)”. Esta jerarquización es útil para no repetir todo el código que se encuentra en los bloques secundarios en el bloque principal, y poder reutilizar código varias veces desde distintos puntos de la lógica.

Este código se ejecutará todo por cada Request que llegue, a menos que se encuentre la directiva “exit” en algún punto antes del final.

```
##### Routing Logic #####

# Main Obelix SIP request routing logic
```

```

request_route {

    # handling MOH messages

    route(MOH_HANDLING); #Bloque que maneja parte de la lógica cuando es necesario
                        #hacer llegar música en espera a un usuario

    # per request initial checks

    route(REQINIT);      #Bloque que implementa chequeos de integridad del mensaje

    # NAT detection

    route(NATDETECT);    #Bloque capaz de detectar a usuarios nateados

    # handle requests within SIP dialogs

    route(WITHINDLG);     #Bloque que maneja todas las request que se
                        #encuentran dentro de un diálogo ya iniciado

    ### De aquí en adelante solo se manejan requests iniciales (sin Tag en el header 'To')

    # CANCEL processing

    if (is_method("CANCEL")) #Aquí la función t_check_trans() chequea si el mensaje CANCEL que
                        #arribó pertenece a alguna de los diálogos en curso. Si es así, lo envía a
                        #donde corresponda.

    {
        if (t_check_trans())

            t_relay();#Esta función envía el mensaje a su destino de manera stateful

        exit;    }

    t_check_trans(); #Esta función tiene varios usos. En este caso termina el script cuando
                    #se recibe un ACK cuyo destinatario es este servidor (Nada más que
                    #procesar ni reenviar)

    # authentication

    route(AUTH);    #Bloque encargado de autenticar usuarios

    # - remove preloaded route headers

    remove_hf("Route");

    if (is_method("INVITE|SUBSCRIBE"))

        record_route();    #Agrega la dirección de este servidor en el
                        #encabezado Record-Route a los INVITE y SUBSCRIBE

    # account only INVITES

    if (is_method("INVITE"))

    {
        #Se marca los invite iniciales con una bandera para realizar accounting

        setflag(FLT_ACC); # do accounting
    }
}

```

```

    }

    # dispatch requests to foreign domains

    route(SIPOUT);  #Bloque encargado de reenviar las requests que no son del dominio del
                   #SIP Proxy

    ### requests for my local domains

    # handle presence related requests

    route(PRESENCE);      #Bloque que maneja la presencia

    # handle registrations

    route(REGISTRAR);     #Bloque que maneja los registros

    if ($rU==$null)       #Chequeo de sanidad de la RURI
    {

        # request with no Username in RURI

        sl_send_reply("484","Address Incomplete");

        exit;

    }

    # dispatch destinations to PSTN

    route(PSTN);    #Bloque que reenvía las llamadas hacia un GW. No usado.

    # go to conferences

    route(CONFERENCE);    #Bloque que maneja las llamadas hacia puentes de conf

    # user location service

    route(LOCATION);      #Bloque que resuelve hacia donde debe enviarse la request

    dlg_manage();  #Función que marca el diálogo que se inicia para tener un completo
                  #seguimiento del mismo

    route(RELAY);    #Bloque encargado del envío final de la request
}

```

De aquí en adelante se definen los distintos bloques secundarios utilizados en el bloque principal de procesamiento de las SIP Requests. Dado que ya es engorroso el código, solo se harán comentarios sobre el funcionamiento conceptual de cada uno al final de los mismos, así como explicaciones que sean de relevancia.

El siguiente bloque es el encargado del envío final de la request.

```

route[RELAY] {

    # enable additional event routes for forwarded requests

    if (is_method("INVITE|SUBSCRIBE")) {

        t_on_branch("MANAGE_BRANCH");

        if($var(bati_moh)) {

            t_on_reply("MANAGE_MOH_REPLY");

            $var(bati_moh)=0;

        }

        else {

            t_on_reply("MANAGE_REPLY");

        }

    }

    if (is_method("INVITE")) {

        t_on_failure("MANAGE_FAILURE");

    }


    if (!t_relay()) {

        sl_reply_error();

    }

    exit;

}

```

Este bloque, que en su parte final intenta enviar la request mediante la función `t_relay()` (si ocurre un error envía un mensaje al remitente con la función `sl_reply_error()`), es capaz de definir bloques especiales para tratar las respuestas a una determinada transacción, o cuando se recibe un mensaje de error.

Mediante la función `t_on_reply("Bloque_respuesta")` indica que las respuestas recibidas a la transacción que se envía mediante `t_relay` deben ser procesadas por el bloque de respuesta "Bloque_respuesta". Por defecto el bloque de respuesta es "MANAGE_REPLY", y es seteado a todas las transacciones.

Sin embargo, cuando un usuario pone en espera a otro, mediante el envío de un re-invite, se deben capturar los datos de sesión (SDP) del holdee para indicar al servidor de música a donde enviar el flujo de media. Este re-invite es detectado

en la lógica (más adelante se ve como) y se setea la bandera “bati_moh” para indicar que la respuesta a ese mensaje debe recibir tratamiento especial. Es por esto que se ve en el código que si la bandera “bati_moh” se encuentra en 1, el bloque de respuesta asociado es “MANAGE_MOH_REPLY”. De esta manera, al llegar la respuesta del holdee, se podrán extraer los datos necesarios.

El siguiente bloque es el encargado de realizar un chequeo de sanidad a las requests que arriban al servidor, así como evitar ataques de *flooding*.

```
# Per SIP request initial checks

route[REQINIT] {

#!ifdef WITH_ANTIFLOOD

    # flood dection from same IP and traffic ban for a while

    # be sure you exclude checking trusted peers, such as pstn gateways

    # - local host excluded (e.g., loop to self)

    if(src_ip!=myself)

    {

        if($sht(ipban=>$si)!=null)

        {

            # ip is already blocked

            xdbg("request from blocked IP - $rm from $fu (IP:$si:$sp)\n");

            exit;

        }

        if (!pike_check_req())

        {

            xlog("L_ALERT","ALERT: pike blocking $rm from $fu (IP:$si:$sp)\n");

            $sht(ipban=>$si) = 1;

            exit;

        }

    }

}

#!endif

if (!mf_process_maxfwd_header("10")) {
```

```

        sl_send_reply("483","Too Many Hops");
        exit;
    }

    if(!sanity_check("1511", "7"))
    {
        xlog("Malformed SIP message from $si:$sp\n");
        exit;
    }
}

```

El siguiente bloque realiza el procesamiento de todas las requests que pertenecen a ya pertenecen a un diálogo. Estas son aquellas que ya contienen el parámetro “tag” en los encabezados “To” y “From”.

Handle requests within SIP dialogs

```

route[WITHINDLG] {
    if (has_totag()) {
        # sequential request withing a dialog should

        # take the path determined by record-routing

        if (loose_route()) {
            if (is_method("BYE")) {
                setflag(FLT_ACC); # do accounting ...

                setflag(FLT_ACCFAILED); # ... even if the transaction fails

                if ($dlg_var(hold_in_progress)=="true" ||
$dlg_var(hold2_in_progress)=="true"){ #Para no dejar colgado al MOH

                    $dlg_var(corto_por_bye)="true";

                    route(TRAMITE_CORTAR_MOH);}

            }

            if ( is_method("ACK") ) {

                # ACK is forwarded statelessy

```

```

        route(NATMANAGE);

        if($dlg_var(freeze_ack)=="on"){ #Llega el ACK confirmando la espera

            route(inviten_al_MOH);

            $dlg_var(freeze_ack)="off";

        }

    }

#ifndef WITH_MOH

    if ( is_method("INVITE") && search_body("sendonly")) {

        if (!$rU=~"30[0-2][0-9]") {

            #ifndef WITH_GROUPS

            if (is_user_in("From", "moh")){

            #endif

            replace_body_atonce("a=sendonly","");

            $var(bati_moh)=1;

            #ifndef WITH_GROUPS

            }

            #endif

        }

    }

    if ( is_method("INVITE") && ($dlg_var(hold_in_progress)=="true" ||
$dlg_var(hold2_in_progress)=="true")) {

        route(TRAMITE_CORTAR_MOH);#De-holder arrives

    }

#endif

    route(RELAY);

} else {

    if (is_method("SUBSCRIBE") && uri == myself) {

        # in-dialog subscribe requests

        route(PRESENCE);

        exit;

    }

}

```

```

    }

    if ( is_method("ACK") ) {

        if ( t_check_trans() ) {

            # no loose-route, but stateful ACK;

            # must be an ACK after a 487

            # or e.g. 404 from upstream server

            t_relay();

            exit;

        } else {

            # ACK without matching transaction ... discard

            exit;

        }

    }

    if (is_method("NOTIFY"))

        route(RELAY);

    sl_send_reply("404","Not here");

}

exit;

}

}

```

Este bloque, en primer lugar, si el mensaje es un BYE marca la transacción para finalizar el *accounting*. Hay un caso en donde un mensaje de BYE podría traer un mal funcionamiento del sistema si no se tiene cuidado.

Cuando un usuario se encuentra en espera, este se encuentra escuchando música desde el servidor de música. Si este usuario en espera, o quien lo tiene en espera, cortaran la llamada sin liberar la espera, el diálogo con el servidor de música debe ser finalizado. De lo contrario el flujo de media seguirá dirigiéndose al usuario de manera errónea.

Dado que quien inicia el diálogo a nivel de SIP con el servidor de música es el propio Proxy, este debe encargarse de cortar esa llamada. Es por esto que también en este bloque se chequea que al llegar un BYE, nadie en ese diálogo se encuentre en espera. Si así fuere, se invoca el bloque “TRAMITE_CORTAR_MOH” encargado de finalizar el diálogo con el servidor de música.

También este bloque es el encargado de invocar al bloque que inicia el diálogo con el servidor de música. Cuando un usuario envía el re-invite para colocar en espera a otro, y luego de que se capturen los datos de sesión del holdee, llegará al servidor un ACK confirmando que el usuario se colocó en espera. Cuando este ACK es detectado, se procede a comenzar el diálogo con el servidor que brinda la música.

Este re-invite para colocar a un usuario en espera también es detectado por este bloque, seteando la bandera "bati_moh" para su uso en el bloque "RELAY".

De la misma forma que se detecta este re-invite, se detecta también un re-invite que es para sacar a un usuario de la espera. En este caso, también se invoca el bloque "TRAMITE_CORTAR_MOH".

Un cuidado especial que se tiene en este bloque, es que un usuario que se encuentra en una sala de conferencia, al poner en espera la llamada no se debe dirigir música hacia la sala, dado que la misma interferiría con la conferencia en curso.

Por último son manejados también los mensajes SUBSCRIBE y NOTIFY, invocando el bloque "PRESENCE" para los subscribes y reenviando a su destino los notifies que no son destinados para el manejo de presencia.

El siguiente bloque es el encargado de manejar los registros en el sistema.

Handle SIP registrations

```
route[REGISTRAR] {
    if (is_method("REGISTER"))
    {
        if(isflagset(FLT_NATS))
        {
            setbflag(FLB_NATB);

            # uncomment next line to do SIP NAT ping
            ## setbflag(FLB_NATSIPPING);
        }

        if (!save("location"))
            sl_reply_error();

        exit;
    }
}
```

```
}
```

El siguiente bloque es el encargado de resolver la ubicación de un usuario o entidad destinataria de una request.

```
# USER location service
```

```
route[LOCATION] {
```

```
#!ifdef WITH_SPEEDIAL
```

```
    # search for short dialing - 2-digit extension
```

```
    if($rU=~"^[0-9][0-9]$")
```

```
        if(sd_lookup("speed_dial"))
```

```
            route(SIPOUT);
```

```
#!endif
```

```
#!ifdef WITH_ALIASDB
```

```
    # search in DB-based aliases
```

```
    if(alias_db_lookup("dbaliases"))
```

```
        route(SIPOUT);
```

```
#!endif
```

```
#!ifdef WITH_GROUPS
```

```
    if (is_user_in("From", "Jefes") && is_user_in("Request-URI", "Directorio")) {
```

```
        sl_send_reply("403", "Forbidden destiny");
```

```
        exit;
```

```
    }
```

```
    if (is_user_in("From", "Empleados") && (is_user_in("Request-URI", "Directorio") || is_user_in("Request-URI", "Gerencia"))) {
```

```
        sl_send_reply("403", "Forbidden destiny");
```

```
        exit;
```

```
    }
```

```
#!endif
```

```
    $avp(oexten) = $rU;
```

```
    if (!lookup("location")) {
```

```

    $var(rc) = $rc;
    if (does_uri_exist()) {
        route(TOVOICEMAIL);
    }
    t_newtran();
    switch ($var(rc)) {
        case -1:
        case -3:
            send_reply("404", "Not Found");
            exit;
        case -2:
            send_reply("405", "Method Not Allowed");
            exit;
    }
}

# when routing via usrloc, log the missed calls also
if (is_method("INVITE"))
{
    setflag(FLT_ACCMISSED);
}
}

```

En este bloque se busca en la tabla de locaciones de la base de datos al destinatario de una request con el fin de poder enviarle el mensaje. Si el usuario que se quiere contactar mediante un INVITE es un suscriptor del sistema pero en este momento no se encuentra registrado en el mismo, la llamada será dirigida al correo de voz de este usuario invocando al bloque “TOVOICEMAIL”. Si el usuario destino no fuera suscriptor del sistema, el usuario que quiere contactarlo será notificado.

También se imponen condiciones sobre los grupos de usuarios y sus permisos, no pudiendo concretar una llamada quien no esté en un grupo que permita la llamada al grupo del destinatario.

El siguiente bloque es el encargado de manejar los mensajes relativos al servicio de presencia.

```
# Presence server route

route[PRESENCE] {
    if(!is_method("PUBLISH|SUBSCRIBE"))
        return;
    route(MWI);
    #ifndef WITH_PRESENCE
    #ifndef WITH_GROUPS
        if (is_user_in("From", "presencia")) {
    #endif
        if (!t_newtran())
        {
            sl_reply_error();
            exit;
        };
        if(is_method("PUBLISH"))
        {
            handle_publish();
            t_release();
        }
        else
        if( is_method("SUBSCRIBE"))
        {
            handle_subscribe();
            t_release();
        }
    #endif WITH_GROUPS
    }
    else{
```

```

        send_reply("202","Accepted");
    }
#endif

    exit;
#endif

    # if presence enabled, this part will not be executed
    if (is_method("PUBLISH") || $rU==$null)
    {
        sl_send_reply("404", "Not here");
        exit;
    }
    return;
}

```

En este bloque se manejan las suscripciones al servicio de presencia. La función `handle_subscribe()` es la encargada de hacer efectivas las suscripciones.

Desde aquí se invoca al bloque “MWI” (message waiting indicator) que es el encargado de manejar las suscripciones que deben ir dirigidas hacia los servidores de *voicemail*.

Se manejan también permisos, en donde un usuario perteneciente a un grupo que no tenga permisos para el servicio de presencia no podrá suscribirse. Sus suscripciones recibirán un mensaje “202 Accepted” que significa que fueron entendidas por el servidor pero no implica que fueron hechas efectivas.

El siguiente bloque es el encargado de autenticar a los usuarios atendidos por el sistema ya sea al registrarse, realizar llamadas o efectuar suscripciones.

```

# Authentication route

route[AUTH] {

    #!ifdef WITH_AUTH

        if (is_method("REGISTER"))
        {

            # authenticate the REGISTER requests (uncomment to enable auth)
            if (!www_authorize("$td", "subscriber"))
            {

```

```
        www_challenge("$td", "0");
        exit;
    }
    if ($au!=$tU)
    {
        sl_send_reply("403", "Forbidden auth ID");
        exit;
    }
} else {
if ($si=="127.0.0.1") ### No pedir autentificacion a mi mismo
{
    # source IP allowed
    return;
}
#ifndef WITH_IPAUTH
    if(allow_source_address())
    {
        # source IP allowed
        return;
    }
#endif

    # authenticate if from local subscriber
    if (from_uri==myself)
    {
        if (!proxy_authorize("$fd", "subscriber")) {
            proxy_challenge("$fd", "0");
            exit;
        }
        if (is_method("PUBLISH"))
        {

```

```
        if ($au!=$fU || $au!=$tU) {
            sl_send_reply("403","Forbidden auth ID");
            exit;
        }
        if ($au!=$rU) {
            sl_send_reply("403","Forbidden R-URI");
            exit;
        }
    #ifndef WITH_MULTIDOMAIN
        if ($fd!=$rd) {
            sl_send_reply("403","Forbidden R-URI domain");
            exit;
        }
    #endif

    } else {
        if ($au!=$fU) {
            sl_send_reply("403","Forbidden auth ID");
            exit;
        }
    }

    consume_credentials();
    # caller authenticated
} else {
    # caller is not local subscriber, then check if it calls
    # a local destination, otherwise deny, not an open relay here
    if (!luri==myself)
    {
        sl_send_reply("403","Not relaying");
        exit;
    }
}
```

```

        }

    }

#endif

    return;

}

```

Este bloque hace uso de funciones del modulo “auth” para desafiar a los usuarios a presentar sus credenciales frente al servidor. Solo se autenticará a los usuarios que sean suscriptores del sistema. Tampoco se pide autenticación a los mensajes que lleguen desde la interfaz de loopback del servidor, que se verá que es un recurso utilizado en este proyecto.

Los siguientes dos bloques son los encargados de detectar usuarios nateados y tomar acciones. No son utilizados dado que no es definida la bandera “WITH_NAT” en la primera sección del archivo. Se incluyen a modo ilustrativo.

```

# Caller NAT detection route

route[NATDETECT] {

#ifdef WITH_NAT

    force_rport();

    if (nat_uac_test("19")) {

        if (is_method("REGISTER")) {

            fix_nated_register();

        } else {

            fix_nated_contact();

        }

        setflag(FLT_NATS);

    }

#endif

    return;

}

# RTPProxy control

route[NATMANAGE] {

#ifdef WITH_NAT

```



```

    if (is_request()) {
        if(has_totag()) {
            if(check_route_param("nat=yes")) {
                setbflag(FLB_NATB);
            }
        }
    }

    if (!(isflagset(FLT_NATS) || isbflagset(FLB_NATB)))
        return;

    rtpproxy_manage();

    if (is_request()) {
        if (!has_totag()) {
            add_rr_param(";nat=yes");
        }
    }

    if (is_reply()) {
        if(isbflagset(FLB_NATB)) {
            fix_nated_contact();
        }
    }

    #endif

    return;
}

```

El siguiente bloque es el encargado de reenviar los mensajes que no son dirigidos al dominio del sistema hacia sus respectivos dominios.

```

# Routing to foreign domains

route[SIPOUT] {
    if (!uri==myself)
    {

```

```

        append_hf("P-hint: outbound\r\n");
        route(RELAY);
    }
}

```

El siguiente bloque se encarga de enviar las llamadas que sean destinadas hacia la PSTN hacia el Gateway del sistema. No implementado dado que la bandera “WITH_PSTN” no se encuentra seteada. Se incluye a modo ilustrativo.

```

# PSTN GW routing

route[PSTN] {
#lifdef WITH_PSTN

    # check if PSTN GW IP is defined

    if (strempy($sel(cfg_get.pstn.gw_ip))) {

        xlog("SCRIPT: PSTN routing enabled but pstn.gw_ip not defined\n");

        return;

    }

    # route to PSTN dialed numbers starting with '+' or '00'

    #   (international format)

    # - update the condition to match your dialing rules for PSTN routing

    if(!($rU=~"^(+|00)[1-9][0-9]{3,20}$"))

        return;

    # only local users allowed to call

    if(from_uri!=myself) {

        sl_send_reply("403", "Not Allowed");

        exit;

    }

    $ru = "sip:" + $rU + "@" + $sel(cfg_get.pstn.gw_ip);

    route(RELAY);

    exit;

#lendif

```

```

        return;
    }

```

El siguiente bloque es el encargado de redirigir las llamadas al correo de voz cuando se lo invoque.

```
# route to voicemail server
```

```
route[TOVOICEMAIL] {
```

```
    #!ifdef WITH_VOICEMAIL
```

```
        if(!is_method("INVITE"))
```

```
            return;
```

```
        # check if VoiceMail server IP is defined
```

```
        if (strempy($sel(cfg_get.voicemail.srv_ip))) {
```

```
            xlog("SCRIPT: VoiceMail routing enabled but IP not defined\n");
```

```
            return;
```

```
        }
```

```
        if($avp(oexten)==$null)
```

```
            return;
```

```
    #!ifdef WITH_GROUPS
```

```
    if(is_user_in("Request-URI", "voicemail")) {
```

```
    #!endif
```

```
        $ru = "sip:" + $avp(oexten) + "@" + $sel(cfg_get.voicemail.srv_ip) + ":"
```

```
            + $sel(cfg_get.voicemail.srv_port);
```

```
        t_set_fr(2000,2000); #Por si falla Dispatcher
```

```
        route(RELAY);
```

```
    #!ifdef WITH_GROUPS
```

```
    }
```

```
    else {
```

```
        return;
```

```
    }
```

```
    #!endif
```

```

        exit;

    #!endif

    return;

}

```

Como se puede ver en el código, cuando el destinatario de la llamada pertenece al grupo “voicemail” (o sea que tiene casilla), este bloque reescribe la RURI del mensaje original, por ejemplo “fran@obelix”, a “fran@voicemail.obelix” y se vuelve a enviar la llamada. El dominio “voicemail.obelix” apunta al modulo de despacho como se podrá ver en la configuración del DNS.

El siguiente bloque es el encargado de cortar el flujo de audio que le está enviando el servidor de música a un usuario que se encuentra en espera.

```

route[TRAMITE_CORTAR_MOH] {

    if($dlg_var(corto_por_bye)=="true"){

        if ($dlg_var(hold_in_progress)=="true" && $dlg_var(hold2_in_progress)=="true"){

            $avp(key_moh)=$ci+"<"+"$ft;

            $avp(key2_moh)=$ci+"<"+"$tt;

            avp_db_load("$avp(key_moh)","$avp(info_moh)");

            #Obtengo los campos necesarios

            $avp(ci)=$(avp(info_moh){s.select,0,<});

            $avp(ftag)=$(avp(info_moh){s.select,1,<});

            $avp(ttag)=$(avp(info_moh){s.select,2,<});

            avp_db_delete("$avp(key_moh)","$avp(info_moh)");

            avp_db_load("$avp(key2_moh)","$avp(info_moh)");

            $avp(ci2)=$(avp(info_moh){s.select,0,<});

            $avp(ftag2)=$(avp(info_moh){s.select,1,<});

            $avp(ttag2)=$(avp(info_moh){s.select,2,<});

            avp_db_delete("$avp(key2_moh)","$avp(info_moh)");

            if(dlg_get("$avp(ci)", "$avp(ftag)", "$avp(ttag))) {

                dlg_bye("callee");

            }

            if(dlg_get("$avp(ci2)", "$avp(ftag2)", "$avp(ttag2))) {

```

```

        dlg_bye("callee");
    }
}
else{
    if ($dlg_var(hold_in_progress)=="true"){
        if ($ft==$dlg_var(fromt_holder)){
            $avp(key_moh)=$ci+"<"+$ft;
        }else{
            $avp(key_moh)=$ci+"<"+$tt;
        }
        avp_db_load("$avp(key_moh)","$avp(info_moh)");
        #Obtengo los campos necesarios
        $avp(ci)=$(avp(info_moh){s.select,0,<});
        $avp(ftag)=$(avp(info_moh){s.select,1,<});
        $avp(ttag)=$(avp(info_moh){s.select,2,<});
        if(dlg_get("$avp(ci)", "$avp(ftag)", "$avp(ttag)")) {
            dlg_bye("callee");
            avp_db_delete("$avp(key_moh)","$avp(info_moh)");
        }
    }else{
        if ($ft==$dlg_var(fromt2_holder)){
            $avp(key_moh)=$ci+"<"+$ft;
        }else{
            $avp(key_moh)=$ci+"<"+$tt;
        }
        avp_db_load("$avp(key_moh)","$avp(info_moh)");
        #Obtengo los campos necesarios
        $avp(ci)=$(avp(info_moh){s.select,0,<});
        $avp(ftag)=$(avp(info_moh){s.select,1,<});
        $avp(ttag)=$(avp(info_moh){s.select,2,<});
        if(dlg_get("$avp(ci)", "$avp(ftag)", "$avp(ttag)")) {
            dlg_bye("callee");
            avp_db_delete("$avp(key_moh)","$avp(info_moh)");
        }
    }
}

```

```

    }

    $dlg_var(corto_por_bye)="false";

    return;

}

$avp(key_moh)=$ci+"<"+$ft;
avp_db_load("$avp(key_moh)","$avp(info_moh)");

#Obtengo los campos necesarios
$avp(ci)=$(avp(info_moh){s.select,0,<});
$avp(ftag)=$(avp(info_moh){s.select,1,<});
$avp(ttag)=$(avp(info_moh){s.select,2,<});

if ($ft==$dlg_var(fromt_holder)){
    $dlg_var(hold_in_progress)="false";}
else{
    $dlg_var(hold2_in_progress)="false";}

if(dlg_get("$avp(ci)", "$avp(ftag)", "$avp(ttag)"))
{
    dlg_bye("callee");
}

avp_db_delete("$avp(key_moh)","$avp(info_moh)");

return;

}

```

Este bloque debe terminar el diálogo iniciado por el propio servidor con el servidor de música. Este diálogo no solo debe terminarse cuando un usuario que se encuentra en espera es sacado de la misma, sino que también se deben considerar otros casos.

Cuanto hay un usuario en espera, podría cortar la llamada antes de este ser sacado de espera tanto el holder como el holdee. También existe el caso en que ambos usuarios se pongan en espera simultáneamente y la llamada sea cortada por alguno de ellos.

Todos estos casos son contemplados en este bloque implementando distintas técnicas para diferenciar entre casos.

Al término del archivo de configuración se presentará detalladamente el problema que surge en la implementación del servicio de música en espera tal cual se implementó aquí, así como la forma en que se solucionó.

El siguiente bloque identifica los mensajes SUBSCRIBE que son para el servicio de MWI. Los mismos son ruteados hacia los servidores de *voicemail*.

```
route[MWI]{
#ifdef WITH_VOICEMAIL

if( is_method("SUBSCRIBE") && $hdr(Event)=="message-summary")
{
        #ifndef WITH_GROUPS
        if (is_user_in("From", "voicemail")){
                #endif
                $ru = "sip:" + $rU + "@" + $sel(cfg_get.voicemail.srv_ip) + ":"
                + $sel(cfg_get.voicemail.srv_port);
                route(RELAY);
                exit;
        #ifndef WITH_GROUPS
        }
        else {
                send_reply("202","Accepted");
                exit;
        }
        #endif
}
#endif
return;
}
```

El siguiente bloque es el encargado de iniciar el diálogo con el servidor de música en espera para que le brinde un flujo de media a un usuario que es puesto en espera.

```

route[inviten_al_MOH] {
    $uac_req(method)="INVITE";
    $uac_req(ruri)="sip:moh@127.0.0.1:5060";
    $uac_req(furi)="sip:kamailio@core.obelix:5060";
    $uac_req(turi)="sip:moh@" + $sel(cfg_get.moh.srv_ip) + ":"
        + $sel(cfg_get.moh.srv_port);
    $uac_req(body)=$dlg_var(SDP_holdee);
    $uac_req(hdrs)=
        "Content-Type: application/sdp\n"+
        "Contact: <kamailio@192.168.2.2:5060>\n"+
        "Info-MOH: "+$ci+"<"+$ft+"\n";    #Encabezado extra que lleva información relativa a la llamada
                                         #que es puesta en espera.
    uac_req_send();
    if($dlg_var(hold_in_progress)=="true"){
        $dlg_var(hold2_in_progress)="true";
        $dlg_var(fromt2_holder)=$ft;}
    else{$dlg_var(hold_in_progress)="true";
        $dlg_var(fromt_holder)=$ft;}
}

```

Este bloque, que inicia el diálogo con el servidor de música en espera, genera un INVITE con el SDP del holdee en el cuerpo para enviárselo al servidor de música en espera. Sin embargo, este mensaje no es enviado directamente al servidor de música, sino que es enviado al mismo SIP proxy por la interfaz de loopback. Esto es crucial para el funcionamiento, y se explicara el porque mas adelante.

El siguiente bloque es el complementario del anterior. Es el encargado de recibir el mensaje proveniente de la interfaz de loopback y reescribir la RURI para reenviarlo hacia el servidor de música.

```

route[MOH_HANDLING]{
    if ( $ru == "sip:moh@127.0.0.1:5060" ) {
        $ru = "sip:moh@" + $sel(cfg_get.moh.srv_ip) + ":" + $sel(cfg_get.moh.srv_port);
        dlg_manage();
    }
}

```



```

$avp(key_moh)=$hdr(Info-MOH); #Obtiene la info de la llamada que inicio el HOLD para usar de
                                #key en la base

remove_hf("Info-MOH"); #Extirpa el encabezado agregado luego de obtener la información

t_on_reply("GUARDA_ESTADO_MOH");

t_set_fr(2000,2000); #Evito demoras ante falla de un dispatcher

t_relay();

exit;

}

}

```

Este bloque además de cambiar la RURI para enviar el mensaje hacia los servidores de música en espera, guarda en una variable de transacción (AVP) la información que fue agregada en un encabezado extra por el mismo servidor SIP Proxy antes de enviar el mensaje a la interfaz de loopback.

Antes de enviar el mensaje hacia su destino, se extirpa el encabezado extra. También se marca un bloque especial para el tratamiento de la respuesta del servidor de música ("GUARDA_ESTADO_MOH").

El siguiente bloque es el encargado de detectar que una llamada es para un número perteneciente a una sala de conferencias (en este caso, del 3000 al 3029) y así reenviar la llamada hacia los servidores encargados de esto.

```

route[CONFERENCE] {

#!ifdef WITH_CONFERENCE

    if ($rU=~"30[0-2][0-9]") {

        $var(rname)="c-"+$rU;

        #ifdef WITH_GROUPS

            if(is_user_in("From", "conf")) {

                if(is_user_in("From", "conf-creator")){

                    $var(rname)="c-"+$rU;

                }

                else {

                    $var(rname)="p-"+$rU;

                }

            }

        #endif

    }

}

```

```

        $ru = "sip:" + $var(rname) + "@" + $sel(cfg_get.conference.srv_ip) + ":"
        + $sel(cfg_get.conference.srv_port);

t_set_fr(2000,2000); #Evito demoras ante falla de un dispatcher

if(!t_relay()) xlog("Fallo en el envio\n");

exit;

#ifdef WITH_GROUPS
    }

else {

    send_reply("403","Forbidden call");

    exit;

}

#endif

}

#endif

return;

}

```

De la misma manera que el bloque de redirección al correo de voz, se reescribe el dominio de la request URI a “conference.obelix” para redirigir la llamada. Dicho dominio apunta a los servidores de despacho como en el caso del correo de voz.

Este bloque también marca la RURI del mensaje a enviar a los servidores de conferencias con una letra. La letra “C” si el usuario tiene permiso de creación de salas de conferencias o la letra “P” si tiene permiso solo como publico (acceder a una conferencia ya creada). De este modo los servidores de conferencias podrán determinar si se le permite crear una sala, o no, al usuario que llama.

Los siguientes bloques son los asignados por defecto al manejo de una “branch” de un diálogo (si es asignado mediante la funcion t_on_branch), y al manejo de las replies a las requests enviadas.

```

# manage outgoing branches

branch_route[MANAGE_BRANCH] {

    xdbg("new branch [$T_branch_idx] to $ru\n");

    route(NATMANAGE);
}

```

```

}

# manage incoming replies

onreply_route[MANAGE_REPLY] {

    xdbg("incoming reply\n");

    if(status=~"[12][0-9][0-9]")

        route(NATMANAGE);

}

```

El siguiente bloque de respuesta es el asignado a las respuestas del re-invite que envía un usuario para colocar en espera a otro.

```

# manage incoming replies de uac_req

onreply_route[MANAGE_MOH_REPLY] {

    xdbg("incoming reply\n");

    if(status=~"[12][0-9][0-9]")

        route(NATMANAGE);

$dlg_var(SDP_holdee)=$rb; #Llega el 200

$dlg_var(freeze_ack)="on";

}

```

Lo que hace este bloque es guardar en una variable de diálogo (\$dlg_var, las variables de diálogo viven durante todo el diálogo entre esos dos usuarios) el cuerpo del mensaje 200 (el SDP del holdee), para luego al llegar el ACK del mismo usuario poder enviar esa información hacia los servidores de música en espera.

También se setea una bandera (\$dlg_var(freeze_ack)) para indicar que el siguiente ACK que llegue en este diálogo es el disparador del proceso de música en espera.

El siguiente bloque corresponde al asignado a las respuestas del diálogo comenzado entre el SIP Proxy y el servidor de música.

```

onreply_route[GUARDA_ESTADO_MOH] {

#Aca se debe guardar el estado de la llamada al MOH. Se guarda ci y tags en AVP's en la base de
#datos para poder cortar luego.

    if(status=="200") {

```

```

        $avp(info_moh)=$ci+"<"+$ft+"<"+$tt;
        avp_db_store("$avp(key_moh)","$avp(info_moh)");
        xlog("Información guardada en base de datos\n");
    }
}

```

Este sencillo bloque es la bisagra entre, el diálogo entre el SIP Proxy y el servidor de música, y el diálogo que mantienen el holder y holdee.

En el momento que llega la respuesta del servidor de música, se tienen todos los datos necesarios para posteriormente poder cortar ese diálogo. Estos son la call-ID y ambos parámetros “Tag” (de los encabezados “From” y “To”).

Estos datos, junto con la información de la llamada que es puesta en espera, son guardados en la base de datos para que puedan ser recuperados en cualquier momento, en particular cuando se requiera cortar el diálogo con el servidor de música en espera.

El siguiente bloque maneja casos de falla en los diálogos. Es el usado por defecto, seteado antes de utilizar la función “t_relay()” mediante la función “t_on_failure(“MANAGE_FAILURE”)”. Se pueden implementar varios bloques de falla al igual que bloques de respuesta.

```

# manage failure routing cases
failure_route[MANAGE_FAILURE] {
    route(NATMANAGE);
    if (t_is_canceled()) {
        exit;
    }
}

#ifdef WITH_BLOCK3XX
    # block call redirect based on 3xx replies.
    if (t_check_status("3[0-9][0-9]")) {
        t_reply("404","Not found");
        exit;
    }
}

#endif

#ifdef WITH_VOICEMAIL

```

```
# serial forking

# - route to voicemail on busy or no answer (timeout)

if (t_check_status("486|408|603")) {

    route(TOVOICEMAIL);

    exit;

}

#endif

}
```

Este último bloque, que maneja los casos de falla, es el encargado de redirigir una llamada a los servidores de correo de voz (invocando el bloque “TOVOICEMAIL”) cuando el estado del mensaje de error es “486” (*Busy*), “408” (*Request timeout*) o “603” (*Decline*).

1.4.2 – El problema del MOH

Presentado el archivo de configuración en su completitud, a continuación se pasa a detallar el problema existente en la implementación de la solución propuesta para el servicio de música en espera así como su solución.

Cuando un usuario pone en espera a otro, el SIP Proxy genera una llamada paralela con el servidor de música. Para el servidor de música, la llamada es contra el SIP Proxy en todo momento.

El SIP Proxy “engaña” al servidor de música indicándole que la dirección de red y el puerto donde está preparado para recibir la música son en realidad la dirección de red y puerto del holdee. De esta manera, el flujo de audio es escuchado por el usuario en espera.

El único que puede cortar el diálogo contra el servidor de música es el propio SIP Proxy, ya que es el que inicia el diálogo. Para lograr cortar un diálogo existente, el SIP Proxy necesita conocer tres parámetros de dicho diálogo, la call-ID y ambos parámetros “Tag” correspondientes a los encabezados “From” y “To”, parámetros los cuales identifican unívocamente un diálogo.

La dificultad radica en que en Kamailio estas variables se encuentran accesibles solamente mientras se están procesando mensajes de ese diálogo.

Cuando llega el momento de cortar el flujo de música esas variables necesarias no se encuentran disponibles, dado que las que se encuentran accesibles son las del diálogo entre las personas y no las del diálogo entre los servidores.

La manera existente de cruzar información entre diálogos es la de guardar esa información en variables globales para que permanezcan accesibles en todo momento. El problema de esto radica en que si hay cien diálogos en espera, se necesitarían cien variables globales con nombres distintos, dependiendo estos nombres del diálogo al que corresponden para poder diferenciarlas y esto no es posible dado que los nombres de las variables globales deben ser estáticos.

Otra forma, que es la usada aquí, consiste en guardar en la base de datos esa información. Sin embargo, de nada sirve guardar esta información sobre los diálogos abiertos contra el servidor de música en la base de datos, si estos no pueden ser vinculados al diálogo entre los usuarios que generó el diálogo entre los servidores.

Si se tuvieran en el mismo momento dentro de la lógica de configuración la información del diálogo entre los usuarios y la información del diálogo entre los servidores, se podría armar parejas entre estas informaciones para su posterior identificación.

Otro problema diferente radica en la forma que se genera el diálogo entre el SIP Proxy y el servidor de música. Este diálogo es creado durante el procesamiento de un mensaje entre los usuarios, mediante la función “uac_req_send()”. Esta función envía el mensaje, que es la request inicial del diálogo, hacia el servidor de música. Este mensaje sale del servidor sin que se pueda acceder a ninguna información sobre este. Esto se debe a que toda la información disponible al momento de la generación de dicho mensaje pertenece al diálogo entre los usuarios, ya que se generó durante el procesamiento de dicho diálogo.

Tampoco se puede acceder a la información de este diálogo cuando llegue la respuesta del servidor de música, dado que no fue posible setear un bloque de lógica adecuado para esta respuesta al no ser enviada con la función “t_relay()”. La respuesta será consumida por el propio modulo “UAC” sin procesarse en el archivo de configuración.

Ambos problemas, el acceso a información de los dos diálogos al mismo tiempo y la obtención de los tres parámetros necesarios para poder finalizar la llamada entre los servidores tienen la misma solución.

Cuando se ensambla la request para enviar al servidor de música no están accesibles los parámetros del diálogo entre servidores, pero si lo están los parámetros del diálogo entre los usuarios dado que el mensaje se ensambla durante el procesamiento de un mensaje de ese diálogo.

Aprovechando esto, se añade un encabezado nuevo al mensaje destinado al servidor de música llamado “Info-MOH” que contiene una concatenación de la “Call-ID” y el “From-Tag” que identifican la llamada y quien es el holdee en el diálogo entre los usuarios (el “From-Tag” identifica al holdee dado que el mensaje que se está procesando al momento de ensamblar este mensaje es un ACK enviado por el holdee).

Lo siguiente es cambiar la RURI del mensaje destinado al servidor de música. En lugar de enviarlo al mismo, se enviará al mismo SIP Proxy mediante su interfaz de *loopback*.

De esta manera, el mensaje será procesado por el SIP Proxy como si fuere una request más sin importar que haya sido enviada por el mismo. Al arribar esta request especial, se da paso al bloque “MOH_HANDLING”. Este bloque lo que hará es guardar la información del encabezado “Info-MOH” en una variable de diálogo (de ahora en mas va a estar disponible durante el procesamiento de todos los mensajes del diálogo entre los servidores). Luego extirpará este encabezado temporal, reescribirá la RURI con la dirección de los servidores de música, asignará un bloque de respuesta para las replies (“GUARDA_ESTADO_MOH”), y enviará el mensaje.

Cuando el servidor de música responda, el bloque asignado para dicha respuesta obtendrá los tres parámetros necesarios para finalizar ese diálogo, y armará una concatenación de los mismos.

Luego de esto, se creará una variable que contiene la pareja de valores “Call-ID + From-Tag” del diálogo entre usuarios, y “Call-ID + From-Tag + To-Tag” del diálogo entre servidores, y posteriormente a esto se guarda esta variable en la base de datos. De esta manera, “Call-ID + From-Tag” funcionará como *key* en la base de datos para recuperar la información posteriormente.

Cuando un usuario saca de espera a otro, se obtiene la Call-ID y el Tag del holdee, se trae de la base de datos la información que corresponde a el diálogo entre servidores asociado a esta llamada (obteniendo los tres parámetros necesarios para finalizar el diálogo con el servidor de música) y se invoca la función que finaliza la llamada con el servidor de música.

1.5 – Archivo de configuración de los servidores de despacho.

En esta sección se presenta el archivo de configuración utilizado en los servidores despacho del sistema. Dado que también su software base es Kamailio, se detallarán solo los puntos importantes de dicho archivo.

El archivo de configuración es el siguiente.

```
#!KAMAILIO
```

```
#!/define WITH_DEBUG

#!/define FLT_ACC 1

#!/define FLT_ACCMISSED 2

#!/define FLT_ACCFAILED 3

#!/define FLT_NATS 5

#!/define FLB_NATB 6

#!/define FLB_NATSIPPING 7

#!/ifdef WITH_DEBUG

debug=1

log_stderr=yes

#else

debug=2

log_stderr=no

#endif

memdbg=5

memlog=5

log_facility=LOG_LOCAL0

fork=yes

children=4

port=5060

#!/ifdef WITH_TLS

enable_tls=yes

#endif

# life time of TCP connection when there is no traffic

tcp_connection_lifetime=3605


plantilla.sdp_generico="v=0

o=- 3553738903 3553738905 IN IP4 192.168.2.29

s=Dispatcher fake

c=IN IP4 192.168.2.29
```


t=0 0

m=audio 50048 RTP/AVP 0 101

a=rtcp:50049

a=rtpmap:0 PCMU/8000

a=rtpmap:101 telephone-event/8000

a=fmtp:101 0-15

a=sendrecv"

En esta sección inicial no hay mayores puntos para comentar, excepto la definición de la variable “plantilla.sdp_generico” que no es otra cosa que una plantilla con un mensaje SDP totalmente arbitrario en los valores de sus parámetros. Este cuerpo genérico será necesario para la recuperación de un servidor de puente de conferencias caído.

La siguiente sección contiene la carga de los módulos.

Modules Section

set paths to location of modules (to sources or installation folders)

#!/ifdef WITH_SRC_PATH

mpath="modules_k:modules"

#!/else

mpath="/usr/local/lib/kamailio/modules_k:/usr/local/lib/kamailio/modules/"

#!/endif

loadmodule "mi_fifo.so"

loadmodule "kex.so"

loadmodule "tm.so"

loadmodule "sl.so"

loadmodule "rr.so"

loadmodule "pv.so"

loadmodule "maxfwd.so"

loadmodule "textops.so"

loadmodule "siputils.so"

loadmodule "xlog.so"

loadmodule "sanity.so"

```

loadmodule "ctl.so"

loadmodule "cfg_rpc.so"

loadmodule "mi_rpc.so"

loadmodule "textopsx.so"

loadmodule "dispatcher.so"      #Módulo de despacho

loadmodule "path.so"

#ifdef WITH_DEBUG

loadmodule "debugger.so"

#endif

loadmodule "exec.so" #Permite ejecutar comandos del sistema

loadmodule "uac.so" #Módulo emulador de UAC

```

En esta sección los módulos nuevos cargados con respecto a los servidores SIP Proxy son “dispatcher” que es el modulo que exporta la función de despacho así como también el pingueo SIP, y el modulo “exec” que permite ejecutar comandos del sistema operativo desde el archivo de configuración.

```

# ----- setting module-specific parameters -----

# ---- mi_fifo params ----

modparam("mi_fifo", "fifo_name", "/tmp/kamailio_fifo")

# ---- tm params ----

# auto-discard branches from previous serial forking leg

modparam("tm", "failure_reply_mode", 3)

# default retransmission timeout: 30sec

modparam("tm", "fr_timer", 250) ## 250ms de tolerancia ante caída

# default invite retransmission timeout after 1xx: 120sec

#modparam("tm", "fr_inv_timer", 120000)

# ---- rr params ----

# add value to ;lr param to cope with most of the UAs

modparam("rr", "enable_full_lr", 1)

# do not append from tag to the RR (no need for this script)

```

```

modparam("rr", "append_fromtag", 1)

#ifndef WITH_DEBUG
# ----- debugger params -----
modparam("debugger", "cfgtrace", 0)
#endif

modparam("dispatcher", "list_file", "/home/Dispatcher1-Files/dispatcher2.list")
modparam("dispatcher", "ds_ping_method", "INFO")
modparam("dispatcher", "ds_probing_mode", 1)
modparam("dispatcher", "ds_ping_interval", 3)
modparam("dispatcher", "ds_ping_reply_codes", "code=407")
modparam("dispatcher", "ds_probing_threshold", 1)
modparam("dispatcher", "ds_hash_size", 11)

```

En esta sección, en donde se setean los parámetros de los módulos, cabe destacar alguno de los parámetros del modulo “dispatcher”.

“list_file” contiene la ruta al archivo donde se encuentra la lista de servidores a despachar.

“ds_ping_method” refiere al método utilizado para pinguear a los servidores. En este caso, el método INFO es el utilizado.

“ds_probing_mode” refiere a cuales servidores de la lista son pingueados. En este caso, el valor “1” indica todos los servidores.

“ds_ping_interval” refiere al intervalo de tiempo en segundos entre ping y ping.

“ds_ping_reply_codes” refiere a cuál o cuáles son los códigos de las respuestas de un servidor al ping, aceptadas como validas.

“ds_probing_threshold” refiere a la tolerancia en cantidad de pings no respondidos correctamente antes de dar de baja al servidor.

A continuación se presentan los bloques de lógica del archivo de configuración.

El siguiente bloque de código es el bloque principal del modulo de despacho.

```

##### Dispatcher Routing Logic #####
request_route {
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483", "Too Many Hops");
    }
}

```

```

    }

    if($rd=="voicemail.obelix"){
        ds_select_dst("4","0");
    }

    if($rd=="moh.obelix"){
        ds_select_dst("5","0");
    }

    if($rd=="conference.obelix") {
        if($rU=~"(p|c)-300[0-9]"){ $var(sw)=0;}
        if($rU=~"(p|c)-301[0-9]"){ $var(sw)=1;}
        if($rU=~"(p|c)-302[0-9]"){ $var(sw)=2;}

        switch($var(sw)) {
        case "0":
            ds_select_dst("1","8");
            $var(sw)=-1;
            break;
        case "1":
            ds_select_dst("2","8");
            $var(sw)=-1;
            break;
        #case "2":
        # ds_select_dst("3","8");
        # $var(sw)=-1;
        # break;
        default:
            sl_send_reply("404", "Not found");
            exit;
        }
    }

    replace("rport;", "");

```

```
msg_apply_changes();  
  
forward();  
  
exit;  
  
}
```

Una SIP Request puede llegar al servidor de despacho por tres razones. O es una request destinada para los servidores de música, o está destinada para los servidores de voicemail, o bien está destinada para los servidores de conferencias.

Estos destinos se diferencian en el dominio de la RURI (para la música en espera será moh.obelix, para el *voicemail* será voicemail.obelix y para las conferencias será conference.obelix).

Antes de despachar una request, el servidor de despacho elige un servidor de su lista para enviar el mensaje. Esta lista puede contener grupos (identificados por un número) de servidores, pudiéndose indicar que se quiere elegir un servidor de determinado grupo.

Es así que en la lista de servidores, que se presentará luego, existen grupos distintos para los distintos servicios. De esta manera, como se puede ver en el bloque de lógica, dependiendo del dominio de la request se elige un servidor de un grupo o de otro.

La función “ds_select_dst” toma como parámetros el número identificador del grupo, y el algoritmo para hacer la selección dentro de ese grupo.

Dado que los servidores de conferencias atienden salas específicas, la elección del destino se hace en función del usuario de la RURI, o numero de sala, enviando así cada sala a su servidor correspondiente.

Los siguientes dos bloques son bloques especiales, ejecutados cuando sucede un evento específico. El modulo “dispatcher” exporta dos tipos de eventos, “dst-down” cuando un servidor se da de baja, y “dst-up” cuando un servidor vuelve en línea.

Estos dos eventos tienen sus rutas a continuación.

```
event_route[dispatcher:dst-down] {  
  
    xlog("SE CAYO UN SERVER $ru\n");  
  
    switch($ru)  
    {  
  
        case "sip:conference1.obelix:5060":
```

```

    $uac_req(ruri)="sip:mayday@192.168.2.39:5080";

    break;

case "sip:conference2.obelix:5080":

    $uac_req(ruri)="sip:mayday@192.168.2.38:5060";

    break;

#case "sip:conference3.obelix:5070":

# break;

default:

    exit;

}

$uac_req(method)="INVITE";

$uac_req(furi)="sip:kamailio@dispatcher.obelix:5060";

$uac_req(turi)="sip:mayday@conference.obelix:5060";

$uac_req(hdrs)="Content-Type: application/sdp\n"+
"Contact: <dispatcher@192.168.2.29:5060>\n";

$uac_req(body)=$sel(cfg_get.plantilla.sdp_generico);

uac_req_send();

}

```

Este bloque de evento, que se ejecuta cuando se detecta la caída de un servidor, es el responsable de disparar la recuperación de las conferencias de un servidor de conferencias caído.

Si un servidor de conferencias cae, debe mandar una señal a su servidor de *back-up* para que este inicie la recuperación.

El servidor de *back-up* tendrá las directivas para realizar la recuperación en su *dialplan*, listas para ser ejecutadas cuando se reciba una llamada destinada al usuario “mayday”.

Es por esto que este bloque al detectar que el servidor que cae es un servidor de conferencias, ensambla un mensaje “INVITE” al usuario “mayday” y lo envía al servidor de *back-up*. Este mensaje de “INVITE”, aunque no esté destinado a comenzar una llamada, debe contener un cuerpo con mensaje SDP. De lo contrario, el servidor de conferencias no aceptará la llamada y no ejecutará la recuperación.

Por este motivo, se definió en una variable una plantilla con un mensaje SDP genérico para ser añadida a este mensaje “INVITE” y así ser aceptado por el servidor de conferencias.

El siguiente bloque, que se ejecuta cuando un servidor vuelve en línea, lo único que realiza es desplegar un mensaje en la pantalla anunciando esto, así como la dirección del mismo.

```
event_route[dispatcher:dst-up] {
    xlog("SE LEVANTO UN SERVER $ru\n");
}
```

A continuación se presenta el archivo de lista de los servidores de despacho. Este es un archivo de texto plano, cuyo formato de línea debe ser:

```
setit(int) destination(sip uri) flags(int,opt) priority(int,opt) attributes(str,opt)
```

Los campos que contienen “opt” son opcionales. El primer parámetro es el identificador de grupo.

#LISTA DEL DISPATCHER

4 sip:voicemail1.obelix:5060 #Servidor de Voicemail 1

4 sip:voicemail2.obelix:5060 #Servidor de Voicemail 2

5 sip:moh1.obelix:5080 #Servidor de MOH 1

5 sip:moh2.obelix:5080 #Servidor de MOH 2

1 sip:conference1.obelix:5060 #Servidor de Conference 1

2 sip:conference2.obelix:5080 8 #Servidor de Conference 2

1.6 – Archivo de configuración de los servidores del modulo PingX

En esta sección se presentan los archivos de configuración y otros scripts utilizados en los servidores del modulo PingX del núcleo de señalización. Dado que también su software base es Kamailio, se detallarán solo los puntos importantes de dichos archivos de configuración.

1.6.1 – PingX Master

A continuación se presenta el archivo de configuración del servidor PingX Master.

La primera sección es idéntica a la del archivo de configuración del modulo de despacho, por lo que solamente se presentaran los parámetros del modulo “dispatcher” y los bloques de lógica.

```
modparam("dispatcher", "list_file", "/home/PingX-Master-Files/PingX_Master.list")
modparam("dispatcher", "ds_ping_method", "INFO")
modparam("dispatcher", "ds_probing_mode", 1)
modparam("dispatcher", "ds_ping_interval", 1)
modparam("dispatcher", "ds_ping_reply_codes", "code=407")
modparam("dispatcher", "ds_probing_threshold", 3)
```

Los parámetros son los mismos descriptos para el servidor de despacho, excepto que aquí se tiene una tolerancia de tres mensajes ping no respondidos antes de declarar de baja un servidor del núcleo.

El bloque principal de procesamiento de requests no tiene mayor complejidad. Lo único que se realiza al llegar una request a este servidor, es responder con un mensaje de código “505”.

Las únicas requests SIP que llegan a este servidor son los “pings” del servidor PingX Slave, que recibirá esta respuesta, tomándola como válida para asumir que el servidor PingX Master se encuentra activo.

```
##### PingX Master Routing Logic #####
request_route {
    sl_send_reply("505", "Que es un Erlang?");
    exit;
}
```

Se añadieron también bloques de eventos, para cuando un servidor del núcleo cae o vuelve en línea. Estos bloques además de desplegar en pantalla el evento ocurrido, son los encargados de actualizar la información del DNS para que este deje de propagar el servidor caído.

Cuando ocurre que un servidor cae, es ejecutado mediante la función “exec_msg()” un script (“pump.it.down”) que corre el programa NSUPDATE para dar de baja al servidor caído.

Cuando un servidor vuelve en línea, se ejecuta el script (“pump.it.up”) encargado de correr el NSUPDATE esta vez para añadir el servidor a la lista del DNS.

Ambos scripts se detallan luego del archivo de lista de este servidor.

```
event_route[dispatcher:dst-down] {
    xlog("WARNING! Servidor caído: $ru\n");

    exec_msg("sh /home/PingX-Master-Files/pump.it.down $ru");
}

event_route[dispatcher:dst-up] {
    xlog("SERVER UP! $ru\n");

    exec_msg("sh /home/PingX-Master-Files/pump.it.up $ru");
}
```

El archivo que contiene la lista del modulo “dispatcher” contiene las direcciones de los servidores SIP Proxy del núcleo y es el siguiente:

```
#Lista PingX Master

1 sip:192.168.2.25:5060 #Servidor del nucleo 1

1 sip:192.168.2.27:5060 #Servidor del nucleo 2
```

A continuación se presentan los scripts que ejecutan la herramienta NSUPDATE tanto para dar de baja un servidor (“pump.it.down”) como para dar de alta un servidor (“pump.it.up”).

--pump.it.down--

```
#!/bin/bash

### Obelix script pump.it.down

KEY="/home/PingX-Master-Files/Kclavemaster.+157+26165.private"

SERVER="192.168.2.36"

ZONE="obelix."

HOSTNAME="obelix."

LOGFILE="/var/log/nsupdate.log"

IPADDR="$(echo $1 | cut -d: -f2)"
```

```
(
echo "IP address: $IPADDR"
echo "-----"
cat <<EOF | nsupdate -d -k "$KEY"
server $SERVER
zone $ZONE
update delete $HOSTNAME IN A $IPADDR
send
EOF
echo "corrio pump.it.down"
echo "=====
) >>$LOGFILE 2>&1
exit
#-----

--pump.it.up--
#!/bin/bash
### Obelix script pump.it.up
KEY="/home/PingX-Master-Files/Kclavemaster.+157+26165.private"
SERVER="192.168.2.36"
ZONE="obelix."
HOSTNAME="obelix."
LOGFILE="/var/log/nsupdate.log"
IPADDR="$(echo $1 | cut -d: -f2)"
(
echo "IP address: $IPADDR"
echo "-----"
cat <<EOF | nsupdate -d -k "$KEY"
server $SERVER
```

```

zone $ZONE

update add $HOSTNAME 0 A $IPADDR

send

EOF

echo "corrio pump.it.up"

echo "======"

) >>$LOGFILE 2>&1

exit

#-----

```

Estos scripts tienen como entrada la SIP URI del servidor que cae/vuelve en línea. Se configura también la ruta a la clave que será usada contra el servidor de DNS, así como la dirección de red de dicho servidor.

La clave utilizada para autenticarse contra el DNS debe ser generada en este servidor. El comando para generar esta clave es el siguiente:

```
dnssec-keygen -a HMAC-MD5 -b 128 -n HOST clavemaster
```

Este comando crea dos archivos. Un archivo de extensión “private” y otro de extensión “key”. La información del archivo “.key” debe ser copiada en la configuración del DNS como se verá más adelante.

1.6.2 – PingX Slave

A continuación se presenta el archivo de configuración del servidor PingX Slave.

Al igual que el servidor PingX Master, la primera sección es idéntica a la del archivo de configuración del modulo de despacho (exceptuando la definición de una variable extra “pingxmaster.ip=“IP PingX Master”), por lo que solamente se presentarán los parámetros del modulo “dispatcher” y los bloques de lógica.

```

modparam("dispatcher", "list_file", "/home/PingX-Slave-Files/PingX_Slave.list")

modparam("dispatcher", "ds_ping_method", "INFO")

modparam("dispatcher", "ds_probing_mode", 1)

modparam("dispatcher", "ds_ping_interval", 1)

modparam("dispatcher", "ds_probing_threshold", 3)

modparam("dispatcher", "ds_ping_reply_codes", "code=505; code=407")

```

Los parámetros son los mismos descriptos para el servidor PingX Master, excepto que aquí se acepta también el código “505” como respuesta válida a los pings.

El bloque de lógica principal no contiene ninguna directiva más que finalizar sin hacer nada si llegara una request a este servidor. Ningún servidor debe hablar con este, por lo que no es necesario procesar ninguna especie de SIP Request.

```
##### PingX Slave Routing Logic #####
```

```
request_route {
    exit;
}
```

Los siguientes bloques de eventos serán los encargados de que, en caso de la caída del servidor PingX Master, este servidor tome su lugar.

```
event_route[dispatcher:dst-down] {
    if ($ru=="sip:"+ $sel(cfg_get.pingxmaster.ip) +":5060") {
        xlog("SE CAYO EL PINGX MASTER $ru\n");
        if ($var(bandera)==0){
            exec_msg("sh /home/PingX-Slave-Files/masterizar");
            $var(bandera)=1;
        }
    }
    else{
        $var(bandera)=0;
    }
}

else {
    xlog("Warning! Servidor del nucleo caído: $ru\n");
    exec_msg("sh /home/PingX-Slave-Files/pump.it.down $ru");
}
}
```

Cuando se detecta la caída del servidor PingX Master, se ejecuta un script (“masterizar”) que cambia el archivo de lista utilizado de servidores a testear. De esta manera, el servidor pasa de estar testeando solamente al servidor PingX Master a testear todos los servidores SIP Proxy del núcleo y así comportarse de la

misma manera que el servidor caído. Este script se presenta luego del bloque siguiente.

Este bloque que sigue tiene el mismo comportamiento que el bloque análogo del servidor PingX Master en el caso de estar caído el mismo. Si se detectara que el PingX Master vuelve en línea, este bloque vuelve a la normalidad el comportamiento de este servidor mediante el script (“desmasterizar”). Este script también se presenta luego de este bloque.

```
event_route[dispatcher:dst-up] {
    xlog("Servidor en línea: $ru\n");
    if ($ru=="sip:" + $sel(cfg_get.pingxmaster.ip) + ":5060") {
        exec_msg("sh /home/PingX-Slave-Files/desmasterizar");
    }
    else {
        exec_msg("sh /home/PingX-Slave-Files/pump.it.up $ru");
    }
}

--Script masterizar--

#!/bin/bash

### Obelix Script - masterizar

mv /home/PingX-Slave-Files/PingX_Slave.list /home/PingX-Slave-Files/PingX_Slave.list.tmp
cp /home/PingX-Slave-Files/PingX_Slave-to_master.list /home/PingX-Slave-Files/PingX_Slave.list
sercmd dispatcher.reload #Este comando recarga la lista del modulo "dispatcher"

--Script desmasterizar--

#!/bin/bash

### Obelix Script - masterizar

mv /home/PingX-Slave-Files/PingX_Slave.list /home/PingX-Slave-Files/PingX_Slave-to_master.list
cp /home/PingX-Slave-Files/PingX_Slave.list.tmp /home/PingX-Slave-Files/PingX_Slave.list
sercmd dispatcher.reload
```

A continuación se presentan las listas usadas por este servidor.

Lista normal:

```
#LISTA DEL DISPATCHER DE PingX Slave
```

```
1 sip:192.168.2.24:5060 #Master del PingX
```

Lista funcionando como Master.

```
#LISTA DEL DISPATCHER DE PingX Slave como Master
```

```
1 sip:192.168.2.24:5060 #Master del PingX
```

```
1 sip:192.168.2.25:5060 #Servidor del nucleo 1
```

```
1 sip:192.168.2.27:5060 #Servidor del nucleo 2
```

Cuando este servidor se encuentra funcionando como “Master”, también hace uso de los scripts “pump.it.up” y “pump.it.down” descriptos anteriormente. La única diferencia radica en el archivo “.private” utilizado como clave. En este servidor también hay que generar una clave mediante el mismo comando, y seguir el mismo procedimiento descripto para el servidor PingX Master.

De esta manera, se detallaron todos los elementos de la arquitectura que usan al Kamailio como software base.

1.7 – Soporte de los desarrolladores para con Obelix.

En algún momento del desarrollo de este proyecto, se necesito resolver un problema utilizando una función de manera particular en la programación de la lógica, pero esta no permitía su utilización de esta manera.

No pudiendo resolver el problema de otro modo, se consulto a la comunidad sobre posibles alternativas.

Los mismos desarrolladores de la herramienta, en particular Daniel-Constantin Mierla, co-fundador del proyecto Kamailio, resolvió modificar el código fuente para lograr el uso de esa función tal cual se necesitaba.

Se adjunta a continuación el mensaje enviado a la comunidad, así como la respuesta Daniel-Constantin Mierla.

Mensaje enviado:

“Hi everyone,

We are using Kamailio as a dispatcher to other two Kamailio servers and want it to stateless forward messages without adding a VIA header. We try to use the send function, but you can not give a pseudo variable as a parameter (\$dd in this case as this variable is set by the ds_select_dst() function).

We attempt to add a rule for this function to accept pseudo variables as an argument (in cfg.y file, pvar argument) but the problem was that when starting Kami (our friendly name for the Kamailio server), the pseudo variable is empty, then the program attempt to do a dns lookup trying to solve the empty string and obviously fail to start.

We partially solve the issue by hard-coding the IP's to dispatch (the only way to make the send() function to work was the sintaxis "send(IP)").

We want a little bit elegancy in configuration file, but can not solve in another way.

Any brighter ideas to make it work?

Cheers from Uruguay,

ODT”

Mensaje recibido:

“Hello,

I just committed a patch that allow using send() without any parameter:

* [http://git.sip-router.org/cgi-bin/gitweb.cgi/sip-router/?](http://git.sip-router.org/cgi-bin/gitweb.cgi/sip-router/?a=commit;h=8e4db808bf9f94d3ccf0f14ed22e75586afd0f23)

a=commit;h=8e4db808bf9f94d3ccf0f14ed22e75586afd0f23

The behaviour is to forward based on \$du (dst uri, aka outbound proxy address) if it is set, otherwise based on \$ru (r-uri).

So what you would have to do after ds_select_dst() is to do 'send()'.

The patch is in master branch, but is rather simple and should be straightforward to apply it to older versions.

Cheers,

Daniel”

Se destaca el soporte brindado por esta comunidad, añadiendo así un plus a la confianza en el uso de esta herramienta.

2 – MySQL. Replicación Master-Master.

Esta sección está destinada a cómo lograr un sistema de alta disponibilidad con un esquema maestro-maestro, con MySQL.

Para lograr la replicación de las bases de datos y configurar el esquema maestro-maestro, hay que configurar a cada uno de los servidores como maestro y esclavo al mismo tiempo.

En este caso el servidor 1 cuenta con IP 192.168.2.33, mientras que el servidor 2 cuenta con IP 192.168.2.34.

Servidor 1 IP: 192.168.2.33

Servidor 2 IP: 192.168.2.34

Bases de datos a replicar: kamilio, freeswitchvm y freeswitchconf

1. Instalar MySQL en ambos servidores:

```
#yum install mysql mysql-server
```

Setear el password de root.

```
#mysqladmin -u root password new_mysql_password
```

2. Crear un usuario con permisos de replicación en ambos servidores:

En servidor 1:

```
#mysql -u root -p
```

Enter Password:

```
mysql> CREATE USER 'slave_user'@'192.168.1.241' IDENTIFIED BY 'password';
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'192.168.1.241' IDENTIFIED BY 'password';
```

Query OK, 0 rows affected (0.00 sec)

Recargar privilegios:

```
mysql> FLUSH PRIVILEGES;
```


Lo mismo en el servidor 2:

```
#mysql -u root -p
```

Enter Password:

```
mysql> CREATE USER 'slave_user'@'192.168.1.240' IDENTIFIED BY 'password';
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'192.168.1.240' IDENTIFIED BY 'password';
```

Query OK, 0 rows affected (0.00 sec)

Recargar privilegios:

```
mysql> FLUSH PRIVILEGES;
```

3. Configurando los nodos MySQL

En el servidor 1 hacer cambios en my.cnf (usualmente localizado en “/etc”).

El archivo debe contener lo siguiente:

```
[mysqld]
```

```
datadir=/var/lib/mysql
```

```
socket=/var/lib/mysql/mysql.sock
```

```
user=mysql
```

```
# Disabling symbolic-links is recommended to prevent assorted security risks
```

```
symbolic-links=0
```

```
#-----
```

```
bind-address = 0.0.0.0      #Importante para que el servidor pueda ser escrito desde cualquier
                             #dirección de red
```

```
server-id = 1               #Identificador del servidor
```

```
master-host = 192.168.2.34   #Dirección IP del servidor maestro de este servidor
```

```
master-user = slave_user    #Usuario usado para conectarse con el servidor maestro
```

```
master-password = obelix    #Password para conectarse con el servidor maestro
```

```
master-connect-retry = 30
```

```
slave-net-timeout = 30
```

```
replicate-do-db = kamailio   #Nombre de la/las bases de datos a replicar desde el maestro
```

```

replicate-do-db = freeswitchvm    #En este caso "kamailio" es la utilizada por los elementos del
replicate-do-db = freeswitchconf  #núcleo, "freeswitchvm" por los servidores de correo y
                                   #"freeswitchconf" por los servidores de conferencias.

log-bin = mysql-bin.log           #Nombre del archivo binario donde guardar los cambios

log-bin-index = master-log-bin.index    #Archivo que contiene el índice con la ubicación
                                         #donde se lee el log binario del maestro

binlog-do-db = kamailio            #Nombre de las bases de datos de las que se van a
binlog-do-db = freeswitchvm        # guardar los cambios en archivos de log binarios que
binlog-do-db = freeswitchconf      # pueden ser leídos por los esclavos

relay-log = /var/lib/mysql-relay-bin
relay-log = /var/lib/mysql/slave-relay.log
relay-log-index = /var/lib/mysql/slave-relay-log.index

#-----

[mysqld_safe]

log-error=/var/log/mysql.log
pid-file=/var/run/mysql/mysql.pid

#-----

```

En el servidor 2 también modificar el mismo archivo. Este es muy similar al del servidor 1, y se remarcan las diferencias.

```

[mysqld]

datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock

user=mysql

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

#-----

bind-address = 0.0.0.0

server-id = 2    #Identificador del servidor (El servidor 1 tenia valor 1)

master-host = 192.168.2.33    #Ip del servidor maestro. Cada cual tiene de maestro al otro.

```

```

master-user = slave_user
master-password = obelix
master-connect-retry = 30
slave-net-timeout = 30
replicate-do-db = kamailio
replicate-do-db = freeswitchvm
replicate-do-db = freeswitchconf
log-bin = mysql-bin.log
log-bin-index = master-log-bin.index
binlog-do-db = kamailio
binlog-do-db = freeswitchvm
binlog-do-db = freeswitchconf
relay-log = /var/lib/mysql-relay-bin
relay-log = /var/lib/mysql/slave-relay.log
relay-log-index = /var/lib/mysql/slave-relay-log.index
#-----
[mysqld_safe]
log-error=/var/log/mysql.log
pid-file=/var/run/mysql/mysql.pid
#-----

```

Ahora reiniciar el servicio mysql:

```

#service mysqld restart

Stopping MySQL:          [ OK ]
Starting MySQL:          [ OK ]

```

4. Configuración de esclavos:

Acceder a la consola de MySQL:

```
#mysql -u root -p
```

Destrabar las tablas de la base de datos en ambos servidores (lo mismo con todas las bases de datos que se deseen replicar):

```
mysql>USE kmailio;
```

```
mysql>UNLOCK TABLES;
```

```
mysql>FLUSH TABLES WITH READ LOCK;
```

Configurar el "slave replication" en servidor 1:

```
mysql> CHANGE MASTER TO MASTER_HOST = '192.168.2.34', MASTER_USER = 'slave_user',  
MASTER_PASSWORD = 'obelix', MASTER_LOG_FILE = 'mysql-bin.000001';
```

Configurar el "slave replication" en servidor 2:

```
mysql> CHANGE MASTER TO MASTER_HOST = '192.168.2.33', MASTER_USER = 'slave_user',  
MASTER_PASSWORD = 'obelix', MASTER_LOG_FILE = 'mysql-bin.000001';
```

Ahora, arrancar los esclavos en ambos servidores:

```
mysql> SLAVE START;
```

Chequear los estados de los esclavos:

```
mysql>SHOW SLAVE STATUS\G
```

Si todo salió bien, debería verse algo de este estilo:

```
mysql> show slave status\G
```

```
***** 1. row *****
```

```
Slave_IO_State: Waiting for master to send event
```

```
Master_Host: 192.168.1.241
```

```
Master_User: slave_user
```

```
Master_Port: 3306
```

```
Connect_Retry: 30
```

```
Master_Log_File: mysql-bin.000001
```

```
Read_Master_Log_Pos: 98
```

```
Relay_Log_File: slave-relay.000001
```

```
Relay_Log_Pos: 198
```

```
Relay_Master_Log_File: mysql-bin.000001
```

```
Slave_IO_Running: Yes
```

```
Slave_SQL_Running: Yes
```

Replicate_Do_DB: db1,db2,db3

Replicate_Ignore_DB:

Replicate_Do_Table:

Replicate_Ignore_Table:

Replicate_Wild_Do_Table:

Replicate_Wild_Ignore_Table:

Last_Errno: 0

Last_Error:

Skip_Counter: 0

Exec_Master_Log_Pos: 1581

Relay_Log_Space: 235

Until_Condition: None

Until_Log_File:

Until_Log_Pos: 0

Master_SSL_Allowed: No

Master_SSL_CA_File:

Master_SSL_CA_Path:

Master_SSL_Cert:

Master_SSL_Cipher:

Master_SSL_Key:

Seconds_Behind_Master: 0

1 row in set (0.00 sec)

Comprobar el estado del maestro:

mysql>SHOW MASTER STATUS;

```
+-----+-----+-----+-----+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000006 | 98 | radius,smbind,Syslog | |
+-----+-----+-----+-----+
```

De esta manera, queda configurado el esquema maestro-maestro para la base de datos del sistema.

3 – BIND. El DNS del sistema.

Esta sección está dedicada a la instalación y configuración del DNS del sistema, haciendo uso del software libre BIND.

Se presenta una guía para lograr poner en funcionamiento este servidor de nombres.

Instalar Bind (Version 9)

```
yum install bind
```

Crear una carpeta para agregar los archivos necesarios luego

```
cd /etc/bind
```

```
mkdir zones/obelix
```

```
cd zones/obelix
```

Crear un archivo con la información de la zona

```
nano db.obelix
```

Este archivo debe contener la siguiente información:

```
$ORIGIN .
```

```
$TTL 0 ; 0 seconds
```

```
obelix                IN SOA  ns1.obelix. gusvidovich\@gmail.com.obelix. (
                                2            ; serial
                                10800        ; refresh (3 hours)
                                3600         ; retry (1 hour)
                                604800       ; expire (1 week)
                                3600         ; minimum (1 hour)
                                )
                                NS           ns1.obelix.
;                                NS           ns2.obelix.
```

A 192.168.2.25

A 192.168.2.27

\$ORIGIN obelix.

\$TTL 1 ; 1 second

_sip._udp SRV 0 5 5060 obelix.

\$TTL 10800 ; 3 hours

core2 A 192.168.2.27

core1 A 192.168.2.25

base1 A 192.168.2.33

base2 A 192.168.2.34

base A 192.168.2.33

base A 192.168.2.34

dispatcher1 A 192.168.2.28

dispatcher2 A 192.168.2.29

PingX-Master A 192.168.2.24

PingX-Slave A 192.168.2.23

voicemail1 A 192.168.2.31

voicemail2 A 192.168.2.32

moh1 A 192.168.2.35

moh2 A 192.168.2.37

conference1 A 192.168.2.38

conference2 A 192.168.2.39

BaseVoiceMail A 192.168.2.30

ns1 A 192.168.2.36

#Estas entradas de nombres canonicos son las responsables de que todas las llamadas destinadas
#a los servidores de features sean dirigidas al modulo de despacho.

moh CNAME dispatcher

voicemail CNAME dispatcher

conference CNAME dispatcher

dispatcher A 192.168.2.28

dispatcher A 192.168.2.29

Como se puede observar, “base.obelix”, “dispatcher.obelix” y “obelix” tienen sus registros A repetidos, logrando que este DNS pueda balancear entre los servidores.

Crear un archivo con la información para la resolución reversa de la zona:

nano db.192.168.2

El archivo debe contener la siguiente información:

```
;
; BIND reverse data file for 2.168.192.in-addr.arpa
;
$TTL 604800
2.168.192.in-addr.arpa. IN SOA ns1.obelix. admin.obelix. (
    1      ; Serial
    3h     ; Refresh after 3 hours
    1h     ; Retry after 1 hour
    1w     ; Expire after 1 week
    1h )   ; Negative caching TTL of 1 day
;
2.168.192.in-addr.arpa. IN NS ns1.obelix.
2.168.192.in-addr.arpa. IN NS ns2.obelix.
25.2.168.192.in-addr.arpa. IN PTR obelix.
27.2.168.192.in-addr.arpa. IN PTR obelix.
2.2.168.192.in-addr.arpa. IN PTR kami.obelix.
11.2.168.192.in-addr.arpa. IN PTR dummy.obelix.
27.2.168.192.in-addr.arpa. IN PTR core2.obelix.
25.2.168.192.in-addr.arpa. IN PTR core1.obelix.
```


33.2.168.192.in-addr.arpa.	IN	PTR	base1.obelix.
34.2.168.192.in-addr.arpa.	IN	PTR	base2.obelix.
33.2.168.192.in-addr.arpa.	IN	PTR	base.obelix.
34.2.168.192.in-addr.arpa.	IN	PTR	base.obelix.
28.2.168.192.in-addr.arpa.	IN	PTR	dispatcher1.obelix.
29.2.168.192.in-addr.arpa.	IN	PTR	dispatcher2.obelix.
24.2.168.192.in-addr.arpa.	IN	PTR	PingX-Master.obelix.
23.2.168.192.in-addr.arpa.	IN	PTR	PingX-Slave.obelix.
31.2.168.192.in-addr.arpa.	IN	PTR	voicemail1.obelix.
32.2.168.192.in-addr.arpa.	IN	PTR	voicemail2.obelix.
35.2.168.192.in-addr.arpa.	IN	PTR	moh1.obelix.
37.2.168.192.in-addr.arpa.	IN	PTR	moh2.obelix.
38.2.168.192.in-addr.arpa.	IN	PTR	conference1.obelix.
39.2.168.192.in-addr.arpa.	IN	PTR	conference2.obelix.
30.2.168.192.in-addr.arpa.	IN	PTR	BaseVoiceMail.obelix.
28.2.168.192.in-addr.arpa.	IN	PTR	dispatcher.obelix.
29.2.168.192.in-addr.arpa.	IN	PTR	dispatcher.obelix.

Luego de creado los archivos de zona, hay que configurar el programa.

Modificar el archivo “named.conf” para que contenga la siguiente información:

```
//  
// named.conf  
//  
zone "." IN {  
    type hint;  
    file "named.ca";  
};  
include "/etc/named.rfc1912.zones";  
include "/etc/named.root.key";
```

```
include "/etc/bind/named.conf.options";
```

```
include "/etc/bind/named.conf.local";
```

En este archivo, se añaden al final dos archivos más. Uno con opciones “named.conf.options” y otro con las configuraciones de la zona.

Por esto, crear el archivo “named.conf.options” y agregar el siguiente código:

```
options {  
    forwarders {  
        192.168.2.1;    #Este servidor es el servidor por defecto si no se puede  
                       #resolver la dirección localmente  
    };  
    rrset-order {order cyclic;}; # Se setea el algoritmo para el balance  
    auth-nxdomain no;  # conform to RFC1035  
    listen-on-v6 { any; };  
    listen-on-v6 port 53 { ::1; };  
    directory    "/var/named";  
    dump-file    "/var/named/data/cache_dump.db";  
    statistics-file "/var/named/data/named_stats.txt";  
    memstatistics-file "/var/named/data/named_mem_stats.txt";  
    recursion yes;  
    dnssec-enable yes;  
    dnssec-validation yes;  
    dnssec-lookaside auto;  
    /* Path to ISC DLV key */  
    bindkeys-file "/etc/named.iscdlv.key";  
    managed-keys-directory "/var/named/dynamic";  
};
```

Por último, crear el archivo “named.conf.local” que contendrá la configuración de la zona.

El contenido del archivo es:

```
key "clavemaster." {      #Se define la clave "clavemaster"

    algorithm hmac-md5;    #Se indica con que algoritmo fue generada

    secret "bHNesrPBcBo1Ngyw8SM+jA=="; #Contenido del archivo ".key" generado en el PingX
                                   #Master

};

key "claveslave." {      #Se define la clave "claveslave"

    algorithm hmac-md5;

    secret "RULcy/au1845MpT3/Mo+4g=="; #Contenido del archivo ".key" generado en el PingX
                                   #Slave

};

zone "obelix" {          #Comienza definición de la zona

    type master;

    file "/etc/bind/zones/obelix/db.obelix"; #Ruta al archivo creado "db.obelix"

    journal "/var/named/slaves/db.obelix.jnl";

    allow-update { key "clavemaster."; key "claveslave."; };    #Se indica que esta zona
                                                                #puede ser modificada por
                                                                #quien presente las claves
                                                                #indicadas

};

zone "2.168.192.in-addr.arpa" {    #Comienza definición de la zona reversa

    type master;

    file "/etc/bind/zones/obelix/db.192.168.2";    #Se indica la ruta al archivo creado
                                                                #"db.192.168.2"

    journal "/var/named/slaves/db.192.168.2.jnl";

};

logging {

    channel query.log {

        file "/var/log/query.log";

        severity debug 1;

    };

    category default { query.log; };
```

```
};
```

Luego de creados estos archivos, se puede testear si la configuración es correcta con el comando:

```
named-checkconf
```

Si no se muestra nada en pantalla, la configuración no contiene errores.

Para probar si el archivo de zona no contiene errores, se puede utilizar el siguiente comando:

```
named-checkzone obelix /etc/bind/zones/obelix/db.obelix
```

Si en pantalla sale 'OK', está todo listo.

Reiniciar el servidor:

```
service named restart
```

Anexo 2 – FreeSWITCH

2.1 – Performance FreeSWITCH Centos 6.1 vs Centos 5.8

Existen fuentes oficiales [75] en las cuales se afirma que FreeSWITCH instalado en la versión 6.1 del sistema operativo CentOS obtiene una performance por debajo de lo esperado generando problemas con el uso del procesador del equipo en donde se realizó la instalación [76] [77] [78]. Recomienda que esta instalación se haga sobre CentOS 5.5 (en concreto para realizar esta prueba se utilizó CentOS 5.8 ya que cuenta con las actualizaciones correspondientes).

A efectos de establecer el grado de impacto en materia de procesamiento se realizan pruebas con el objeto de estresar tanto la instalación hecha en 5.8 como la instalación hecha en 6.1 apuntando a monitorear cual es la carga máxima del procesador al momento de procesar una cantidad considerable de solicitudes, cual es la carga media al momento de cesar el procesamiento de señalización y cuanta memoria consume el proceso que ejecuta FreeSWITCH.

Se establece como criterio para realizar la prueba dos equipos con características idénticas de hardware, con diferentes sistemas operativos y con la misma versión de FreeSWITCH, siendo esta la versión 1.2.0 rc2.

Esta prueba se realiza con el escenario creado para evaluar el servidor de Music on Hold (MOH), decidiendo por esta prueba dada su característica de procesar audio por parte del procesador del equipo. Se utiliza la herramienta generadora de trafico SIPp la cual de acuerdo al escenario que se ejecute, SIPp procede a enviar los paquetes simulando una llamada que es atendida con música en espera por parte del servidor. La idea para obtener datos de capacidad de procesamiento es incrementar la cantidad de llamadas por segundo y la cantidad de ejecuciones de escenario por segundo creando así una prueba de estrés sobre el servidor de música en espera.

Como criterio de conformidad para la prueba realizada se estableció que la cantidad de retransmisiones a causa del estrés del equipo sean nulas, al momento de detectar una retransmisión se establece el límite de establecimientos por segundo. Fijado este valor se comienza a realizar pruebas en múltiplos del valor limite alcanzado y se observa cuales son las cargas del procesador y memoria del equipo según el sistema operativo en análisis.

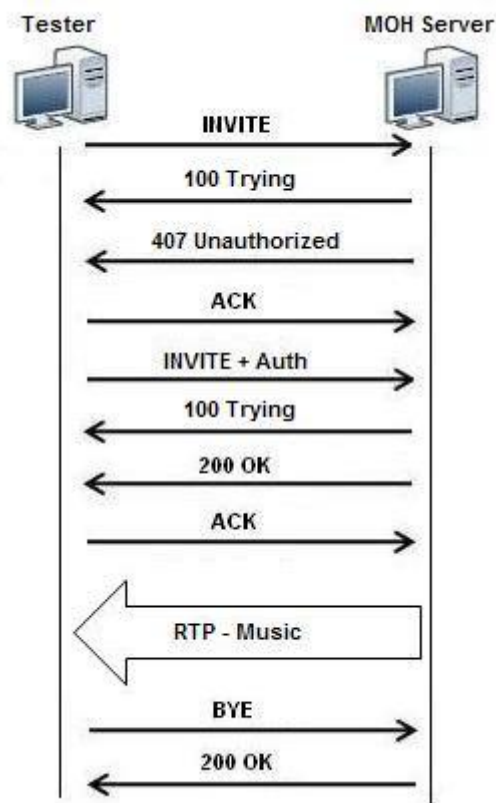


Figura A.2. 1 - Diagrama de flujo SIP al servidor de Música en Espera

El diagrama de flujo SIP describe el escenario de prueba a ser ejecutado, se puede visualizar el período de espera entre el ultimo ACK con SDP que queda a la espera del envío de flujo RTP por parte del servidor de MOH, una vez finalizada la espera se procede a enviar los BYE correspondientes para finalizar la llamada.

La tabla que sigue a continuación describe para los diferentes valores de cantidad de ejecuciones y cantidad de llamadas por segundo cuales son las cargas del procesador y memoria del equipo.

	CPU_Max(%)	CPU_Med(%)	RAM(%)
140Ej/seg, 140Est/seg	94,53	99,20	99,65
150Ej/seg, 150Est/seg(* ¹⁶)	69,20	70,13	99,85
160Ej/seg, 160Est/seg(***) ¹⁷	12,10	12,53	9,60

Tabla A.2. 1 - Carga del procesador y memoria CentOS 5.8

Para el caso de 150 establecimientos por segundo con 150 ejecuciones por segundo se detectan retransmisiones pero se logra completar la totalidad de los establecimientos solicitados.

En el caso de 160 establecimientos por segundo con 160 ejecuciones por segundo además de detectar retransmisiones se detectan mensajes de timeout correspondientes a intentos que no les llega el ACK y la herramienta da por perdido ese establecimiento. El mensaje que describe el estado del sistema al momento de la carga es System Busy ya que a esa carga la cantidad de establecimientos torna el requerimiento simulado por SIPp totalmente inmanejable por parte del servidor. Vale aclarar que la baja carga en memoria de la última prueba detallada en la tabla se debe a no poder procesar la cantidad total de establecimientos que se están tratando de lograr, es por esa razón que la carga de memoria se ve descendida respecto a los ensayos anteriores. En concreto, de 160 establecimientos por cada ejecución del escenario se completa una totalidad de 108 y 52 se declaran como fallidas en el intento de ser establecidas.

Se establece como límite de prueba para realizar la prueba de estrés, 140 establecimientos por segundo a una razón de 140 ejecuciones del escenario por segundo, por lo que se realizan pruebas con múltiplos de esta cantidad de ejecuciones del escenario a la tasa de 140 establecimientos por segundo.

Al duplicar la cantidad de ejecuciones del escenario a tasa descrita anteriormente el proceso freeswitch que se ejecuta en el sistema operativo consume el 99,9% del procesador inhabilitando cualquier otro tipo de operativa, además de constatar retransmisiones, timeouts y mensajes de system busy. Cabe resaltar que el escenario no finaliza ya que no recibe los mensajes de fin de la comunicación correspondiente lo cual genera que en el equipo que se prueba el escenario se tenga que suprimir el escenario. Además en el servidor donde se está ejecutando la prueba el proceso de FS queda en esa cantidad de recursos

¹⁶ (*): Se detectan retransmisiones

¹⁷ (**): Se detectan retransmisiones, timeouts y mensajes de system busy

del procesador sin variar esta cantidad y siendo necesario matar el proceso para poder ejecutar nuevamente FS.

Tanto el escenario de prueba como la metodología de prueba para CentOS 6.1 no difieren con lo efectuado en CentOS 5.8. La revisión obtenida para 6.1 según los diferentes valores de establecimientos y ejecuciones por segundo son los siguientes.

	CPU_Max(%)	CPU_Med(%)	RAM(%)
140Ej/seg, 140Est/seg(* ¹⁸)	98,40	99,40	90,00
150Ej/seg, 150Est/seg(** ¹⁹)	71,83	67,30	63,37
160Ej/seg, 160Est/seg(**)	12,05	13,20	12,33

Tabla A.2. 2 – Carga del procesador y memoria en CentOS 6.1

De acuerdo a lo obtenido según las pruebas que se realizaron en 6.1 se define que el límite operativo que cumple con el criterio de conformidad es 130 establecimientos por segundo a 130 ejecuciones del escenario por segundo, tomando este límite para realizar las sucesivas pruebas de estrés al sistema.

Las sucesivas pruebas de carga al sistema se harán en base a múltiplos de la cantidad límite de ejecuciones por segundo (260, 390 y 420) a cadencia del límite establecido de establecimientos por segundo.

En estas pruebas se constata que se obtienen retransmisiones y mensajes de sistema ocupado, siendo el proceso freeswitch que alcance niveles del 99.9% de ocupación del procesamiento pero sin mensajes de timeout, lo cual repercute en la finalización del escenario de prueba. Así, en cada prueba ejecutada el proceso freeswitch vuelve a liberar la ocupación alcanzada en carga total del procesador sin la necesidad de tener que matar el proceso o en la eventualidad tener que reiniciar el equipo para poder ejecutar FreeSWITCH.

¹⁸ (*): se detectan retransmisiones

¹⁹ (**): se detectan retransmisiones y mensajes de system busy

2.2 – Instalación de FreeSWITCH desde GIT

La principal referencia para realizar una instalación correcta de FreeSWITCH se encuentra en la siguiente WEB²⁰. De todas formas a modo de resumen se describen los pasos a seguir para realizar una instalación descargando el código fuente desde el repositorio GIT de FreeSWITCH.

En un pc con CENTOS debemos ejecutar:

- `yum install git` , de esta forma instala el cliente GIT
- `cd /usr/local/src`
- `git clone git://git.freeswitch.org/freeswitch.git`, para descargar el código fuente desde git hacia el PC.

En este punto se cuenta entonces con el código fuente de FreeSWITCH. A continuación se procede a instalar las dependencias de compilación.

Se deben instalar los siguientes paquetes con `yum install`, para cumplir dependencias:

- `gcc`
- `gcc-c++`
- `libtool`
- `ncurses-devel.i686`
- `libjpeg-devel`
- `libtiff-devel`
- `autoconf`
- `automake`
- `zlib-devel`
- `zlib`

Luego desde el directorio `/usr/local/src/freeswitch`, ejecutar:

```
sh bootstrap.sh
```

Este script genera nuevos archivos en el directorio entre los cuales se encuentra `modules.conf`

Una vez terminada la ejecución del script ejecutar:

```
./configure
```

Esto chequeara efectivamente que se cumplan todas las dependencias para la compilación posterior.

²⁰ Guía instalación FreeSWITCH http://wiki.freeswitch.org/wiki/Installation_Guide

A continuación editar el archivo `modules.conf` descomentando los módulos que deseamos que se compilen. Finalizada la elección de los módulos se procede a compilar y posteriormente instalar FreeSWITCH mediante el siguiente comando, ejecutado desde el directorio `/usr/local/src/freeswitch`:

```
make && make install
```

Al finalizar esta la instalación se puede seguir compilando agregados con el comando `make`, por ejemplo para la funcionalidad de voicemail es imprescindible contar con los audios utilizados por la IVR, MoH y la aplicación de Voicemail, por lo tanto ejecutamos:

- `make cd-sounds-install`, descarga archivos de audio utilizados por IVR
- `make cd-moh-install`, descarga streams de Music On Hold.

Una vez finalizado estos procesos se cuenta con una instalación de FreeSWITCH pronta para comenzar a configurar y utilizar.

2.3 – Configurando FreeSWITCH como “Proxy”

Puede ser de interés configurar FreeSwitch de forma que en las sesiones de media que establece, los medios (audio, video, etc) fluyan de extremo a extremo sin pasar por él. Esta configuración se debe realizar para cada perfil que atienda en un puerto específico. Por ejemplo si se desea que para en las sesiones SIP que se establezcan a través puerto 5060 de FreeSWITCH, el medio correspondiente a dicha sesión no atraviese la PBX se debe modificar en el archivo de configuración del perfil SIP “Internal” ubicado en `/usr/local/freeswitch/conf/sip_profiles/internal.xml` lo siguiente:

```
<param name="inbound-bypass-media" value="true"/>
```

De esta forma cuando se establezca una llamada de audio o video cuya señalización SIP haya sido dirigida hacia el puerto 5060, los medios no pasaran por la PBX, únicamente le llegará la señalización correspondiente a ese dialogo SIP. Esto es de gran utilidad si se utilizar FreeSWITCH solo para manejar señalización en forma parecida a como lo hace un SIP Proxy. Vale la pena destacar que de todas formas, aun con esta configuración FreeSWITCH no es un proxy. La calidad de B2BUA no la pierde ya que, la negociación de señalización la realiza de forma “independiente” con sus extremos (por ejemplo seteando diferentes branches y tag’s para cada “leg”). Por lo tanto no es un Proxy puro por definición.

“No media mode is an SDP Passthrough feature that permits two endpoints that can see each other (no funky NATs) to connect their media sessions directly while FreeSWITCH maintains control of the SIP signaling. This is useful if you have two end-points that need to use a codec that is currently not supported in FreeSWITCH (video) or if you are using FreeSWITCH in a high performance walled garden network and want to minimize the RTP handling FreeSWITCH is doing to maximize call traffic.

When set, the media (RTP) from the originating endpoint is sent directly to the destination endpoint and vice versa. The signaling (SIP) for both endpoints still goes through FreeSWITCH, but the media is point-to-point.”²¹

Este modo fue utilizado para testear a FreeSWITCH como posible “Proxy” en el capítulo 4.

2.4 – Configuración de FreeSwitch como servidor de Music On Hold

Archivo /usr/local/freeswitch/conf/vars.xml

```
<include>
  <X-PRE-PROCESS cmd="set" data="default_password=1234"/>
  <X-PRE-PROCESS cmd="set"
data="sound_prefix=${sounds_dir}/en/us/callie"/>
  <X-PRE-PROCESS cmd="set" data="domain=moh.obelix"/>
  <X-PRE-PROCESS cmd="set" data="domain_name=${domain}"/>
  <X-PRE-PROCESS cmd="set" data="hold_music=local_stream://moh"/>
  <X-PRE-PROCESS cmd="set" data="use_profile=internal"/>
  <X-PRE-PROCESS cmd="set" data="zrtp_secure_media=true"/>
  <X-PRE-PROCESS cmd="set"
data="global_codec_prefs=G722,PCMU,PCMA,GSM"/>
  <X-PRE-PROCESS cmd="set"
data="outbound_codec_prefs=PCMU,PCMA,GSM"/>
  <X-PRE-PROCESS cmd="set" data="xmpp_client_profile=xmppc"/>
  <X-PRE-PROCESS cmd="set" data="xmpp_server_profile=xmpps"/>
  <X-PRE-PROCESS cmd="set" data="bind_server_ip=auto"/>
  <X-PRE-PROCESS cmd="set" data="external_rtp_ip=stun:stun.freeswitch.org"/>
  <X-PRE-PROCESS cmd="set" data="external_sip_ip=stun:stun.freeswitch.org"/>
  <X-PRE-PROCESS cmd="set" data="unroll_loops=true"/>
  <X-PRE-PROCESS cmd="set" data="outbound_caller_name=FreeSWITCH"/>
  <X-PRE-PROCESS cmd="set" data="outbound_caller_id=0000000000"/>
```

²¹ http://wiki.freeswitch.org/wiki/Bypass_Media

```

<X-PRE-PROCESS cmd="set" data="call_debug=false"/>
<X-PRE-PROCESS cmd="set" data="console_loglevel=info"/>
<X-PRE-PROCESS cmd="set" data="default_areacode=918"/>
<X-PRE-PROCESS cmd="set" data="default_country=US"/>

```

<!-- if false or undefined, the destination number is included in presence NOTIFY dm:note.

if true, the destination number is not included -->

```

<X-PRE-PROCESS cmd="set" data="presence_privacy=false"/>
<X-PRE-PROCESS cmd="set" data="be-ring=%(1000,3000,425)"/>
<X-PRE-PROCESS cmd="set" data="ca-ring=%(2000,4000,440,480)"/>
<X-PRE-PROCESS cmd="set" data="cn-ring=%(1000,4000,450)"/>
<X-PRE-PROCESS cmd="set" data="cy-ring=%(1500,3000,425)"/>
<X-PRE-PROCESS cmd="set" data="cz-ring=%(1000,4000,425)"/>
<X-PRE-PROCESS cmd="set" data="de-ring=%(1000,4000,425)"/>
<X-PRE-PROCESS cmd="set" data="dk-ring=%(1000,4000,425)"/>
<X-PRE-PROCESS cmd="set" data="dz-ring=%(1500,3500,425)"/>
<X-PRE-PROCESS cmd="set" data="eg-ring=%(2000,1000,475,375)"/>
<X-PRE-PROCESS cmd="set" data="fi-ring=%(1000,4000,425)"/>
<X-PRE-PROCESS cmd="set" data="fr-ring=%(1500,3500,440)"/>
<X-PRE-PROCESS cmd="set" data="hk-
ring=%(400,200,440,480);%(400,3000,440,480)"/>
<X-PRE-PROCESS cmd="set" data="hu-ring=%(1250,3750,425)"/>
<X-PRE-PROCESS cmd="set" data="il-ring=%(1000,3000,400)"/>
<X-PRE-PROCESS cmd="set" data="in-
ring=%(400,200,425,375);%(400,2000,425,375)"/>
<X-PRE-PROCESS cmd="set" data="jp-ring=%(1000,2000,420,380)"/>
<X-PRE-PROCESS cmd="set" data="ko-ring=%(1000,2000,440,480)"/>
<X-PRE-PROCESS cmd="set" data="pk-ring=%(1000,2000,400)"/>
<X-PRE-PROCESS cmd="set" data="pl-ring=%(1000,4000,425)"/>
<X-PRE-PROCESS cmd="set" data="ro-ring=%(1850,4150,475,425)"/>
<X-PRE-PROCESS cmd="set" data="rs-ring=%(1000,4000,425)"/>
<X-PRE-PROCESS cmd="set" data="ru-ring=%(800,3200,425)"/>
<X-PRE-PROCESS cmd="set" data="sa-ring=%(1200,4600,425)"/>
<X-PRE-PROCESS cmd="set" data="tr-ring=%(2000,4000,450)"/>
<X-PRE-PROCESS cmd="set" data="uk-
ring=%(400,200,400,450);%(400,2000,400,450)"/>
<X-PRE-PROCESS cmd="set" data="us-ring=%(2000,4000,440,480)"/>
<X-PRE-PROCESS cmd="set" data="bong-ring=v=-7;%(100,0,941.0,1477.0);v=-
7;>=2;+-.1;%(1400,0,350,440)"/>
<X-PRE-PROCESS cmd="set"
data="sit=%(274,0,913.8);%(274,0,1370.6);%(380,0,1776.7)"/>
<X-PRE-PROCESS cmd="set" data="default_provider=example.com"/>

```

```
<X-PRE-PROCESS cmd="set" data="default_provider_username=joeuser"/>
<X-PRE-PROCESS cmd="set" data="default_provider_password=password"/>
<X-PRE-PROCESS cmd="set"
data="default_provider_from_domain=example.com"/>
<X-PRE-PROCESS cmd="set" data="default_provider_register=false"/>
<X-PRE-PROCESS cmd="set" data="default_provider_contact=5000"/>
<X-PRE-PROCESS cmd="set" data="sip_tls_version=tlsv1"/>
<!-- Internal SIP Profile -->
<X-PRE-PROCESS cmd="set" data="internal_auth_calls=true"/>
<X-PRE-PROCESS cmd="set" data="internal_sip_port=5060"/>
<X-PRE-PROCESS cmd="set" data="internal_tls_port=5061"/>
<X-PRE-PROCESS cmd="set" data="internal_ssl_enable=false"/>
<X-PRE-PROCESS cmd="set" data="internal_ssl_dir=${base_dir}/conf/ssl"/>
<!-- External SIP Profile -->
<X-PRE-PROCESS cmd="set" data="external_auth_calls=false"/>
<X-PRE-PROCESS cmd="set" data="external_sip_port=5080"/>
<X-PRE-PROCESS cmd="set" data="external_tls_port=5081"/>
<X-PRE-PROCESS cmd="set" data="external_ssl_enable=false"/>
<X-PRE-PROCESS cmd="set" data="external_ssl_dir=${base_dir}/conf/ssl"/>
</include>
```

Archivo /usr/local/freeswitch/conf/sip_profiles/internal.xml :

```

<profile name="internal">

  <!-- Outbound Registrations -->
  <gateways>
    <X-PRE-PROCESS cmd="include" data="internal/*.xml"/>
  </gateways>

  <domains>
    <domain name="all" alias="true" parse="false"/>
  </domains>

  <settings>
    <param name="debug" value="0"/>
    <param name="sip-trace" value="no"/>
    <param name="sip-capture" value="no"/>

    <!--<param name="liberal-dtmf" value="true"/>-->
    <param name="watchdog-enabled" value="no"/>
    <param name="watchdog-step-timeout" value="30000"/>
    <param name="watchdog-event-timeout" value="30000"/>

    <param name="log-auth-failures" value="false"/>
    <param name="forward-unsolicited-mwi-notify" value="false"/>

    <param name="context" value="public"/> <!-- Utiliza el Dialplan public!! -->
    <param name="rfc2833-pt" value="101"/>
    <!-- port to bind to for sip traffic -->
    <param name="sip-port" value="${internal_sip_port}"/>
    <param name="dialplan" value="XML"/>
    <param name="dtmf-duration" value="2000"/>
    <param name="inbound-codec-prefs" value="${global_codec_prefs}"/>
    <param name="outbound-codec-prefs" value="${global_codec_prefs}"/>
    <param name="rtp-timer-name" value="soft"/>
    <!-- ip address to use for rtp, DO NOT USE HOSTNAMES ONLY IP ADDRESSES -->
    <param name="rtp-ip" value="${local_ip_v4}"/>
    <!-- ip address to bind to, DO NOT USE HOSTNAMES ONLY IP ADDRESSES -->
    <param name="sip-ip" value="${local_ip_v4}"/>
    <param name="hold-music" value="${hold_music}"/>
    <param name="apply-nat-acl" value="nat.auto"/>
    <!-- lo siguiente comentado para no autenticar usuarios -->
    <!-- <param name="apply-inbound-acl" value="domains"/> -->
    <param name="local-network-acl" value="localnet.auto"/>
    <param name="record-path" value="${recordings_dir}"/>

```

```

    <param name="record-template"
value="${caller_id_number}.${target_domain}.${strftime(%Y-%m-%d-%H-%M-%S)}.wav"/>
    <param name="manage-presence" value="false"/>
    <param name="inbound-codec-negotiation" value="generous"/>
<!-- Let calls hit the dialplan before selecting codec for the a-leg -->
    <param name="inbound-late-negotiation" value="true"/>

<!-- Allow ZRTP clients to negotiate end-to-end security associations (also
enables late negotiation) -->
    <param name="inbound-zrtp-passthru" value="true"/>
<!--TTL for nonce in sip auth-->
    <param name="nonce-ttl" value="60"/>
<!--Uncomment if you want to force the outbound leg of a bridge to only offer
the codec
    that the originator is using-->
<!--<param name="disable-transcoding" value="true"/>-->
<!-- Handle 302 Redirect in the dialplan -->
<!--<param name="manual-redirect" value="true"/> -->
<!-- Disable Transfer -->
<!--<param name="disable-transfer" value="true"/> -->
<!-- Disable Register -->
<!--<param name="disable-register" value="true"/> -->
<!-- lo siguiente comentado para no autenticar usuarios -->
<!-- <param name="auth-calls" value="${internal_auth_calls}"/> -->
<!-- Force the user and auth-user to match. -->
<param name="inbound-reg-force-matching-username" value="true"/>
<!-- on authed calls, authenticate *all* the packets not just invite -->
<param name="auth-all-packets" value="false"/>

<!-- external_sip_ip
    Used as the public IP address for SDP.
    Can be an one of:
    ip address      - "12.34.56.78"
    a stun server lookup - "stun:stun.server.com"
    a DNS name      - "host:host.server.com"
    auto           - Use guessed ip.
    auto-nat       - Use ip learned from NAT-PMP or UPNP
-->
<param name="ext-rtp-ip" value="auto-nat"/>
<param name="ext-sip-ip" value="auto-nat"/>

<!-- rtp inactivity timeout -->
<param name="rtp-timeout-sec" value="300"/>

```

```
<param name="rtp-hold-timeout-sec" value="1800"/>
<!--
  These are enabled to make the default config work better out of the box.
  If you need more than ONE domain you'll need to not use these options.

-->
<!--all inbound reg will look in this domain for the users -->
<param name="force-register-domain" value="${domain}"/>
<!--force the domain in subscriptions to this value -->
<!--all inbound reg will stored in the db using this domain -->
<param name="force-register-db-domain" value="${domain}"/>

<!--<param name="delete-subscription-on-register" value="false"/>-->
<param name="challenge-realm" value="auto_from"/>
</settings>
</profile>
```


Dialplan “public” utilizado por el perfil internal puerto UDP 5060

```

<!-- http://wiki.freeswitch.org/wiki/Dialplan_XML -->
<include>
  <context name="public">

    <extension name="unloop">
      <condition field="{unroll_loops}" expression="^true$"/>
      <condition field="{sip_looped_call}" expression="^true$">
        <action application="deflect" data="{destination_number}"/>
      </condition>
    </extension>

    <extension name="call_debug" continue="true">
      <condition field="{call_debug}" expression="^true$" break="never">
        <action application="info"/>
      </condition>
    </extension>
  </context>
</include>

```

La siguiente extensión implementa la feature Music on Hold. Si el campo de usuario de la URI del INVITE es exactamente “moh”, se contesta la llamada con un 200 OK para posteriormente se ejecuta la aplicación playback. Esta última reproduce el stream de audio ubicado en `$$hold_music`.

```

<extension name="Music On Hold">
  <condition field="destination_number" expression="^moh$">
    <action application="log" data="Musica en Espera" />
    <action application="answer"/>
    <action application="playback" data="{hold_music}"/>
  </condition>
</extension>
<X-PRE-PROCESS cmd="include" data="public/*.xml"/>
<!--
<extension name="check_auth" continue="true">
  <condition field="{sip_authorized}" expression="^true$" break="never">
    <anti-action application="respond" data="407"/>
  </condition>
</extension>
--> <!-- Extensión comentada no interesa autenticar el usuario -->
</context>
</include>

```

2.5 – Guía para configuración de FreeSwitch como elemento constitutivo de un Banco de Voicemail

Pre-requisitos:

- Instalación fresca de FreeSWITCH en Centos.
- Motor de base de datos instalado en servidor elegido para almacenar la base de datos de FreeSWITCH. Se trabaja con el motor Mysql.

Se parte de la hipótesis que no hay un banco existente al cual se desee integrar el nuevo servidor. Si así lo fuera, solo se debe configurar un conector ODBC hacia la base de datos existente y realizar el full mesh de NFS que se explicara más adelante.

Por lo tanto partiendo de dicha hipótesis en el servidor de voicemail que formará parte del banco se deberá configura un conector ODBC hacia el motor de bases de datos elegido.

Para esto se deben instalar los paquetes correspondientes a dicho conector mediante el comando:

- `yum install unixODBC-devel mysql-connector-odbc`

Este instalará el conector ODBC y su driver utilizado para el motor Mysql. Una vez finalizada esta instalación se procede a habilitar el driver Mysql editando el archivo de configuración del conector ODBC ubicado en `/etc/odbcinst.ini`, el mismo debe quedar de la siguiente forma:

Driver from the mysql-connector-odbc package

Setup from the unixODBC package

```
[MySQL]
Description      = ODBC for MySQL
Driver           = /usr/lib/libmyodbc5.so
Setup            = /usr/lib/libodbcmyS.so
Driver64         = /usr/lib64/libmyodbc5.so
Setup64          = /usr/lib64/libodbcmyS.so
FileUsage        = 1
```

Posteriormente se debe crear el siguiente archivo de texto `/etc/odbc.ini` el cual debe contener los siguientes campos:

```
[freeswitch]
Driver      = MySQL
SERVER      = base.obelix
PORT        = 3306
DATABASE    = freeswitchvm
OPTION      = 67108864
USER        = kerry
PASSWORD    = obelix
```

El conector denominado "freeswitch", utilizará del Driver anteriormente configurado en el archivo /etc/odbc.ini. Se debe especificar la dirección del servidor donde se aloja el motor de bases Mysql (en el ejemplo base.obelix), el puerto al cual realizar las consultas (3306), el nombre de la base de datos, usuario y contraseña de escritura y lectura de la base.

A continuación se crea la base de datos en el servidor Mysql:

- `mysql -u root -p create freeswitchvm` (para crear la base de datos en el motor)
- `GRANT ALL PRIVILEGES ON freeswitchvm.* TO kerry IDENTIFIED BY 'obelix';` (otorga permisos de lectura y escritura al usuario Kerry el cual se conectará a la base a través del conector)

Se puede verificar conectividad hacia la base de datos creada anteriormente ejecutando el siguiente comando desde el servidor con FreeSWITCH:

- `isql -v freeswitch`

Este utilizará el conector ODBC configurado anteriormente para conectarse hacia la base de datos freeswitchvm. Si la salida es :

```
+-----+
| Connected!                                |
|                                           |
| sql-statement                            |
| help [tablename]                         |
| quit                                     |
|                                           |
+-----+
```

La configuración del conector ODBC y la creación de la base son exitosas.

Si la compilación e instalación de FreeSWITCH fue realizada de acuerdo a la guía incluida en este anexo, se debe recompilar e instalar nuevamente mediante el comando `make && make install` ejecutado desde el directorio donde se encuentra el código fuente (/usr/local/src/freeswitch). La compilación detectará la presencia del conector ODBC e instalará los módulos de FreeSWITCH necesarios para utilizarlo.

Finalizada la instalación se debe configurar FreeSWITCH para que este último deposite su esquema de datos en el servidor destinado a la misma (base.obelix). Esto se logra modificando los siguientes parámetros:

- En `/usr/local/freeswitch/conf/autoload_config/switch.conf.xml`
Se debe configurar la entrada xml
`<param name="core-db-dsn" value="freeswitch:kerry:obelix" />`
- En `/usr/local/freeswitch/conf/autoload_config/voicemail.conf.xml`
Se debe configurar la entrada xml
`<param name="odbc-dsn" value="freeswitch:kerry:obelix"/>`
Este parámetro conecta el módulo de voicemail de FreeSWITCH hacia la base de datos externa haciendo uso del conector ODBC configurado.
- En `/usr/local/freeswitch/conf/sip_profiles/internal.xml`
Se debe configurar la entrada xml
`<param name="odbc-dsn" value="freeswitch:kerry:obelix"/>`

En el archivo de configuración del módulo de voicemail (ubicado en `/usr/local/freeswitch/conf/autoload_config/voicemail.conf.xml`), se debe agregar la ruta donde se almacenarán los mensajes de voz que arriben hacia el servidor, esto se logra seteando la siguiente etiqueta xml:

`<param name="storage-dir" value="/usr/local/freeswitch/mensajesVMX"/>`

La carpeta `mensajesVMX` debe ser previamente creada con permisos de lectura y escritura.

Todos los FreeSWITCH que constituyen el mismo banco de voicemail deben atender el dominio `voicemail.obelix`. Esto se configura como variable global en `/usr/local/freeswitch/conf/vars.xml` mediante la siguiente etiqueta xml:

- **`<X-PRE-PROCESS cmd="set" data="domain=voicemail.obelix"/>`**

En esta instancia estamos en condiciones de iniciar FreeSWITCH. En el momento de inicio este creará el esquema de su base de datos en el servidor `@base.obelix` (si es que ya no está creado en caso que el servidor que se agregue al banco no sea el primero).

Siguiendo este mismo procedimiento para todas las instalaciones de FreeSWITCH que constituyan el banco de voicemail, se obtiene como resultado que se comporten en forma de clúster.

2.5.1 – NFS (Network File System)

Cuando un usuario consulta su casilla de voicemail este pedido tendrá como destino cualquier elemento del banco, es responsabilidad del servidor destinatario del pedido entregar todos los mensajes pertenecientes al usuario, los cuales pueden encontrarse distribuidos entre los demás elementos del banco.

Para esto se implementa con NFS²² un esquema de directorios compartidos, en el cual cada servidor tiene acceso a los directorios de almacenamiento de los restantes servidores. Tratándose entonces de una arquitectura de archivos Full-Mesh. De esta forma en conjunto con el esquema de base de datos compartida se logra el objetivo deseado.

2.5.2 – Configuración directorios compartidos por NFS (banco de Voicemail)

En cada servidor de Voicemail se debe instalar el servicio de NFS mediante el comando de CentOS:

- `yum -y install nfs-utils`

Para que dicho servicio sea levantado en automática al inicio del sistema se debe configurar:

- `chkconfig --level 35 nfs on`

Para iniciar el servicio en forma manual se ejecuta el comando “service nfs start”

Una vez instalado el servicio en cada servidor se debe hacer explícito en cada uno de ellos los directorios que el protocolo permitirá compartir, que servidores tienen permisos para acceder al mismo y con que tipo de permisos lo pueden hacer.

A modo de ejemplo se muestra la configuración necesaria en para el caso de un banco compuesto por 2 servidores (VM1 y VM2)

²² <http://iie.fing.edu.uy/ense/assign/admunix/nfs.htm>

En el archivo `/etc/exports` del Voicemail1 (VM1) se debe agregar la siguiente línea:

```
/usr/local/freeswitch/mensajesVM1 192.168.2.31(rw,no_root_squash)
```

Donde `/usr/local/freeswitch/mensajesVM1` corresponde a la ruta donde el voicemail1 almacena sus mensajes, `192.168.2.31` corresponde a la IP del servidor de voicemail2 (el cual tendrá permiso para montar este directorio) y `(rw,no_root_squash)` corresponde a los permisos que sobre dicho directorio el voicemail2 (escritura y lectura)

Análogamente en el archivo `/etc/exports` del Voicemail2 (VM2) se debe agregar la siguiente línea:

```
/usr/local/freeswitch/mensajesVM2 192.168.2.32 (rw,no_root_squash)
```

Por último en cada servidor de voicemail se debe montar los directorios exportados por los restantes servidores del banco, estableciendo hacia una arquitectura Full-Mesh.

Continuando con el ejemplo, se debe ejecutar:

En VoiceMail1

```
mount -o auto,rsiz=8192,wsiz=8192,soft  
192.168.2.32:/usr/local/freeswitch/mensajesVM2  
/usr/local/freeswitch/mensajesVM2
```

En VoiceMail2

```
mount -o auto,rsiz=8192,wsiz=8192,soft  
192.168.2.31:/usr/local/freeswitch/mensajesVM1  
/usr/local/freeswitch/mensajesVM1
```

Para una completa descripción de la sintaxis del comando `mount` se puede ver la siguiente WEB²³.

²³ <http://linux.die.net/man/8/mount>

2.6 – Configuración para el envío de notificaciones de mensajes de voicemail a correo electrónico

Se implementa la funcionalidad que al momento de realizar una llamada a una extensión si ésta se encuentra ocupada o no está registrada automáticamente se activa el correo de voz o voicemail el cual nos permite dejar un mensaje de voz.

Este mensaje es notificado al usuario de esa extensión vía correo electrónico ya sea notificando que tiene un mensaje y consulte a su casilla o en su defecto enviar la grabación del mensaje como un adjunto al correo.

2.6.1 – Parámetros a establecer en la configuración de FreeSWITCH

a) extensiones

En el directorio `/usr/local/freeswitch/conf/directory/default/` se encuentran definidas las extensiones de los internos a ser utilizados (1000.xml, 1001.xml, etc). Dentro de cada xml se establece o no la posibilidad de que cada uno de ellos tenga casilla de correo para que se le envíe la notificación de mensaje o la notificación con el mensaje correspondiente.

Todos estos parámetros se definen dentro de las sentencias:

```
<params>
```

```
...
```

```
</params>
```

Definimos la cuenta de correo donde se enviaran las notificaciones de los mensajes recibidos:

```
<param name="vm-mailto" value="obe_user@voicemail"/>
```

Para enviar todos los mensajes recibidos vía correo:

```
<param name="vm-notify-email-all-messages" value="true"/>
```

Si se precisa que se adjunte la grabación del mensaje y así poder reproducirla con un reproductor de audio local:

```
<param name="vm-attach-file" value="true"/>
```

Existe la posibilidad de chequear los mensajes que se dejaron en la casilla de mensajes de voz de la extensión vía web, esto tiene que habilitarse por extensión mediante:

```
<param name="http-allowed-api" value="voicemail"/>
```

Así el usuario podrá manejar su casilla de mensajes accediendo al portal web del voicemail. La dirección a la que debe acceder es `http://voicemail:8080/api/voicemail/web` y se validará con mismo usuario y contraseña que si lo hiciese por vía telefónica.

b) en la configuración de freeswitch

Los alcances del proyecto escapan la posibilidad que exista un servidor de mail propio por lo que se implementa la solución propuesta por freeswitch que permite el envío de los mensajes de notificación vía correo externo. El script escrito en python permite que al momento de guardar el mensaje de voz se envíe un correo usando una cuenta de correo en gmail (precisamente `obelix.proyecto@gmail.com`) a la cuenta de correo definida en las extensiones.

Este script queda guardado en la ruta `/usr/local/freeswitch/scripts` con el nombre `sendemail.py`, para que este sea ejecutado se le otorgan permisos de lectura, ejecución y escritura mediante el comando `chmod 777`.

En el ruta `/usr/local/freeswitch/conf/autoload_configs/` se encuentra el fichero `switch.conf.xml`, se tienen dos parámetros que se fijan de la siguiente forma:

```
<param name="mailer-app"
        value="/usr/local/freeswitch/scripts/sendemail.py"/>
```

```
<param name="mailer-app-args"
        value="-i -d -f obelix.proyecto@gmail.com -t"/>
```


Esto permite que al momento de que una extensión se comunica con otra extensión que no se encuentra registrada u ocupada y al dejar un mensaje grabado, se envía un correo notificando que existe una grabación en la casilla de correo de la extensión llamada.

Además de poder chequear el voicemail mediante la extensión definida para tal funcionalidad, existe la posibilidad de poder realizar el chequeo del mismo vía web.

Habilitando (descomentando) el módulo `mod_xml_rpc` en el fichero `modules.xml.conf` se permite el acceso vía web para el chequeo de la cuenta de voicemail.

2.7 – Configuración XML de FreeSwitch como servidor de Voicemail

Archivo `/usr/local/freeswitch/conf/vars.xml`

```
<include>
  <X-PRE-PROCESS cmd="set" data="default_password=1234"/>
  <X-PRE-PROCESS cmd="set"
data="sound_prefix=${sounds_dir}/en/us/callie"/>
  <X-PRE-PROCESS cmd="set" data="domain=voicemail.obelix"/>
  <X-PRE-PROCESS cmd="set" data="domain_name=${domain}"/>
  <X-PRE-PROCESS cmd="set" data="hold_music=local_stream://moh"/>
  <X-PRE-PROCESS cmd="set" data="use_profile=internal"/>
  <X-PRE-PROCESS cmd="set"
data="global_codec_prefs=G722,PCMU,PCMA,GSM"/>
  <X-PRE-PROCESS cmd="set"
data="outbound_codec_prefs=PCMU,PCMA,GSM"/>
  <!-- Internal SIP Profile -->
  <!--<X-PRE-PROCESS cmd="set" data="internal_auth_calls=true"/>-->
  <X-PRE-PROCESS cmd="set" data="internal_sip_port=5060"/>
  <X-PRE-PROCESS cmd="set" data="internal_tls_port=5061"/>
  <X-PRE-PROCESS cmd="set" data="internal_ssl_enable=false"/>
  <X-PRE-PROCESS cmd="set" data="internal_ssl_dir=${base_dir}/conf/ssl"/>

  <!-- External SIP Profile -->
  <X-PRE-PROCESS cmd="set" data="external_auth_calls=false"/>
```

```

<X-PRE-PROCESS cmd="set" data="external_sip_port=5080"/>
<X-PRE-PROCESS cmd="set" data="external_tls_port=5081"/>
<X-PRE-PROCESS cmd="set" data="external_ssl_enable=false"/>
<X-PRE-PROCESS cmd="set" data="external_ssl_dir=${base_dir}/conf/ssl"/>
</include>

```

Archivo /usr/local/freeswitch/conf/sip_profiles/internal.xml :

```

<profile name="internal">
  <gateways>
    <X-PRE-PROCESS cmd="include" data="internal/*.xml"/>
  </gateways>
  <settings>
    <param name="debug" value="0"/>
    <param name="sip-trace" value="no"/>
    <param name="sip-capture" value="no"/>
    <param name="watchdog-enabled" value="no"/>
    <param name="watchdog-step-timeout" value="30000"/>
    <param name="watchdog-event-timeout" value="30000"/>
    <param name="log-auth-failures" value="false"/>
    <param name="forward-unsolicited-mwi-notify" value="false"/>
    <param name="context" value="public"/>
    <param name="sip-port" value="${internal_sip_port}"/>
    <param name="dialplan" value="XML"/>
    <param name="dtmf-duration" value="2000"/>
    <param name="inbound-codec-prefs" value="${global_codec_prefs}"/>
    <param name="outbound-codec-prefs" value="${global_codec_prefs}"/>
    <param name="rtp-timer-name" value="soft"/>
    <param name="rtp-ip" value="${local_ip_v4}"/>
    <param name="sip-ip" value="${local_ip_v4}"/>
    <param name="hold-music" value="${hold_music}"/>
    <param name="apply-nat-acl" value="nat.auto"/>

    <!--<param name="apply-inbound-acl" value="domains"/>-->
    <!--<param name="apply-register-acl" value="domains"/>-->
    <!--<param name="dtmf-type" value="info"/>-->
    <param name="track-calls" value="true"/>
    <!--If you have ODBC support and a working dsn you can use it instead of
SQLite-->
    <param name="odbc-dsn" value="freeswitch:kerry:obelix"/>
    <param name="inbound-late-negotiation" value="true"/>
    <!-- Allow ZRTP clients to negotiate end-to-end security associations (also
enables late negotiation) -->
    <param name="inbound-zrtp-passthru" value="true"/>

```

```

<!--TTL for nonce in sip auth-->
<param name="nonce-ttl" value="60"/>
<param name="ext-rtp-ip" value="auto-nat"/>
<param name="ext-sip-ip" value="auto-nat"/>
<!-- rtp inactivity timeout -->
<param name="rtp-timeout-sec" value="300"/>
<param name="rtp-hold-timeout-sec" value="1800"/>
<param name="force-register-domain" value="${domain}"/>
<!--force the domain in subscriptions to this value -->
<param name="force-subscription-domain" value="voicemail.obelix"/>
<!--all inbound reg will stored in the db using this domain -->
<param name="challenge-realm" value="auto_from"/>
</settings>
</profile>

```

Dialplan “public” de un servidor Voicemail

```

<include>
<context name="public">
  <extension name="unloop">
    <condition field="${unroll_loops}" expression="^true$"/>
    <condition field="${sip_looped_call}" expression="^true$">
      <action application="deflect" data="${destination_number}"/>
    </condition>
  </extension>
  <extension name="call_debug" continue="true">
    <condition field="${call_debug}" expression="^true$" break="never">
      <action application="info"/>
    </condition>
  </extension>

```

La siguiente extensión implementa un cambio de dialplan (hacia el default) para los INVITE que arriban para los usuarios que verifican la expresión regular:

```
"^([1-2]0[01][0-9]|^4000|gonza|gus|pablo)$"
```

Estos serán los usuarios que puedan acceder a una casilla de voicemail. Además de estar configurados aquí deben existir como usuarios (ver configuración de archivo Directory) .

```
<extension name="public_extensions">
```

```

    <condition field="destination_number" expression="^([1-2]0[01][0-9])|^4000|gonza|gus|pablo)$">
        <action application="transfer" data="$1 XML default"/>
    </condition>
</extension>
</context>
</include>

```

Dialplan “default” de un servidor Voicemail

La siguiente extensión brinda la funcionalidad de voicemail, si la expresión regular se verifica (por ende el usuario destino del INVITE cuenta con casilla de voicemail).

```

<extension name="Local_Extension">
    <condition field="destination_number" expression="^([1-2]0[0-1][0-9])|gonza|gus|pablo)$">
        <action application="set" data="continue_on_fail=true"/>
        <action application="answer"/>
        <action application="voicemail" data="default ${domain} ${dialed_extension}"/>
    </condition>
</extension>

```

La extensión a continuación implementa el acceso por parte de un usuario a su casilla de correo.

```

<extension name="vmain">
    <condition field="destination_number" expression="^4000$">
        <action application="answer"/>
        <action application="sleep" data="1000"/>
        <action application="voicemail" data="check default ${domain_name}"/>
    </condition>
</extension>

```

2.8 – Creación de casillas de voicemail mediante el “User Directory” de FreeSWITCH

Para que un usuario perteneciente al dominio @obelix tenga casilla de voicemail debe estar habilitado como usuario en todos los servidores del banco. Aparte de verificar las expresiones regulares de las extensiones configuradas en el Dialplan default y public. Un usuario se define en FreeSWITCH mediante el “User Directory”.

Creando un archivo xml bajo el directorio /

```
<include>
  <user id="1000">
    <params>
      <param name="password" value="${default_password}"/>
      <param name="vm-password" value="1000"/>
      <param name="vm-mailto" value="gusvidovich@gmail.com"/>
      <param name="vm-notify-email-all-messages" value="true"/>
    </params>
    <variables>
      <variable name="toll_allow" value="domestic,international,local"/>
      <variable name="accountcode" value="1000"/>
      <variable name="user_context" value="default"/>
      <variable name="effective_caller_id_name" value="Extension 1000"/>
      <variable name="effective_caller_id_number" value="1000"/>
      <variable name="outbound_caller_id_name" value="${outbound_caller_name}"/>
      <variable name="outbound_caller_id_number" value="${outbound_caller_id}"/>
      <variable name="callgroup" value="techsupport"/>
    </variables>
  </user>
</include>
```

2.9 Puentes de Conferencias

Los servidores de Conferencias también serán implementados con *FreeSWITCH*. En su *Dialplan* por defecto este trae extensiones que utilizan la aplicación “*conference*” para implementar salas de conferencias dinámicas a través del módulo de conferencias. Adelantando la necesidad de contar con *failover* de esta *feature*, se debe contar con al menos 2 servidores que la implementen.

Estos 2 o más servidores tendrán todos la misma configuración desde el punto de vista del *Dialplan*, lo cual en principio sino se aplica ninguna política de ruteo en el *Dispatcher Kamailio* lleva a pensar que todas las salas existen en todos los servidores. Esto es claramente un inconveniente práctico ya que usuarios que disquen un interno correspondiente a una misma sala puedan eventualmente arribar a servidores diferentes, por lo tanto conferencias diferentes. Para solucionar esto se utiliza ruteo estático aplicado directamente sobre los *Dispatcher*. Mediante la misma lógica aplicada a ambos *Kamailio's (Dispatcher's)* se logra que rangos de internos disjuntos se encaminen hacia diferentes servidores:

Internos del 3000 al 3009 se encaminarán hacia un servidor y del 3010 al 3019 les corresponderá el otro. De esta forma si un servidor deja de funcionar por cualquier motivo solo se pierden las conferencias que se alojan en él más no la totalidad de las mismas. Usuarios que se vean afectados por la caída de un servidor podrán eventualmente coordinar una nueva conferencia en una sala correspondiente al otro servidor. De todas formas, la solución final permitió avanzar aún más en este tema lográndose una recuperación automática transparente al usuario. Esto abarca la regeneración automática de las conferencias afectadas por la caída, en otro servidor. Este servidor extra, atenderá a su vez sus propias conferencias, por lo tanto se trata de un sistema de *failover* de sesiones en configuración “activo-activo”. A continuación se abarca la solución final en mayor profundidad.

2.10 Puentes de Conferencias. Solución con failover “en caliente” de sesiones.

Para implementar este tipo de recuperación de conferencias frente a una caída del servidor donde las mismas se alojan, se utiliza la *feature* “HA”²⁴ (High Availability) de FreeSWITCH. Cabe destacar que el diseño de esta solución solo tiene en cuenta la caída de un solo servidor.

Para esto, es necesario que todos los servidores de Conferencias conformen un *cluster* de la misma forma que se empleó para el Banco de Voicemail. Por lo tanto todos los servidores compartirán una misma base de datos, lo cual por practicidad se alojará en el servidor de la arquitectura donde se encuentran las demás bases datos, aprovechándose así el esquema de replicación.

Cada servidor de conferencias deberá contar con por lo menos 2 interfaces de red, ya sean físicas o virtuales. De aquí en más se describe la solución con 2 interfaces de red. A cada interfaz de red se le asociará un perfil o “SIP Profile” en terminología específica de FreeSWITCH, por lo tanto en un servidor conferencias con FreeSWITCH existirán 2 perfiles uno activo y otro pasivo (Back-up de un perfil existente en otro servidor). A su vez cada perfil tiene un Dialplan asociado que utilizará y un puerto UDP específico donde recibirá la señalización.

Para bajar estas ideas a tierra se propone el siguiente ejemplo con 3 servidores de conferencias:

Conference 1:

- Perfil Internal IP 192.168.2.38 Puerto SIP UDP 5060 Dialplan Default
- Perfil External (Back-Up) IP 192.168.2.39 Puerto SIP UDP 5080 Dialplan Public

Conference 2:

- Perfil Extra IP 192.168.2.40 Puerto SIP UDP 5070 Dialplan Extra
- Perfil Internal (Back-Up) IP 192.168.2.38 Puerto SIP UDP 5060 Dialplan Default

Conference 3:

- Perfil External IP 192.168.2.39 Puerto SIP UDP 5080 Dialplan Public

²⁴ http://wiki.freeswitch.org/wiki/Freeswitch_HA

- Perfil Extra (Back-UP) 192.168.2.40 Puerto SIP UDP 5070 Dialplan Extra

En régimen de funcionamiento normal (todos los servidores funcionando sin inconvenientes) en cada uno de estos servidores existe un perfil activo y un perfil pasivo que es “back-up” de otro perfil activo perteneciente a otro servidor. Los perfiles back-up se encontrarán activos a nivel de SIP, pero apagados a nivel de interfaz de red, de otra forma se estaría ante una situación de IP duplicada. El sistema operativo debe ser configurado para que permita a FreeSWITCH tener un perfil activo escuchando en una interfaz de red administrativamente apagada. Esto se logra con el siguiente comando:

- `cat 1 > /proc/sys/net/ipv4/ip_nonlocal_bind`

A su vez cada Perfil activo y su correspondiente “back-up” en otro servidor deben ser iguales desde todo punto de vista (nombre de perfil, IP, Puerto asociado y Dialplan, como se observa en la puntualización de ejemplo). Para esto en cada perfil se debe configurar:

- `<param name="sip-ip" value="IP"/>` bind IP con Perfil.
- `<param name="sip-port" value="Puerto"/>` bind Puerto con Perfil.
- `<param name="odbc-dsn" value="freeswitch:usuario:password"/>`
Conector ODBC utilizado para leer y escribir la base de datos compartida.

En los perfiles activos se debe habilitar el almacenamiento de todo el estado de las sesiones activas atendidas por el perfil, esto se realiza con:

`<param name="track-calls" value="true"/>`

De esta forma, en caso de falla el perfil de back-up contará con toda la información necesaria para regenerar la última situación de la conferencia antes de la caída. En función de estos datos el perfil back-up generará mensajes de reINVITE hacia todos los User Agents de la conferencia. Desde el punto de vista de los User Agents se percibe esta caída del servidor como un evento de renegociación por parte del mismo, haciendo la caída totalmente transparente a los usuarios.

En lo que corresponde a detalles de implementación se destacan los siguientes puntos:

- Se configuran todas las interfaces correspondientes a perfiles “back-up” como apagadas ya desde el inicio del SO modificando el parámetro ONBOOT=no del archivo `/etc/sysconfig/network-scripts/ifcfg-ethX`

- Se crea un script en bash que detecta la caída de perfiles activos, este script corre en cada servidor y una vez detectada la caída del servicio (caída del socket del perfil activo) inmediatamente apaga la interfaz de red correspondiente dicho perfil. Este script se denomina “watchcat”. Su funcionalidad es esencial para evitar problemas de dirección de red duplicada.
- En cada Dialplan correspondiente a perfiles activos debe existir la extensión llamada “mayday”. Esta se encargará levantar la interfaz de red correspondiente al perfil “back-up” que se encuentra en el mismo servidor de conferencias que dicho perfil activo. Recordando que los perfiles back-up se encontraban activos pero sin conectividad IP dado que la interfaz se encontraba desactivada, en el momento posterior a la ejecución de la extensión “mayday” el perfil caído vuelve a existir, ahora en otro servidor. Por último la extensión mayday ejecuta la funcionalidad “profile recover” la cual utiliza la base de datos compartida para regenerar con mensajes de reINVITE la conferencia en cuestión.

Explicado todos los elementos necesarios para realizar este failover de sesiones se hace explícita una secuencia lógica de eventos que permiten la misma:

1. Situación normal, todos los perfiles activos responden al “Ping SIP” efectuado por los Dispatchers.
2. Se produce cuelgue de servicio en un servidor, por ejemplo Conference 1. Este cuelgue puede ser por diversas razones, por ejemplo se perdió conectividad IP, el servidor se reinició ó apagó, el socket IP:Puerto correspondiente al perfil activo dejó de responder a los mensajes SIP, etc.
3. Inmediatamente el script “watchcat” corriendo en el servidor Conference 1, detectará esta caída (en caso de caída de socket, en los restantes casos no tiene sentido) y bajará la interfaz correspondiente al perfil Internal (192.168.2.38).
4. Simultáneamente el Dispatcher detectará que Conference 1 no responde al “Ping SIP” y generará un INVITE con URI mayday@192.168.2.40:5070 (perfil activo de Conference 2).
5. Conference 2 recibe este INVITE y lo procesa de acuerdo a su Dialplan, generando entonces que el perfil Internal (Back-Up) se termine de poner en servicio debido a que su interfaz de red se activa con la IP 192.168.2.38.

6. Por último en el mismo procesamiento del mensaje “mayday” se efectúa el comando “sofia profile internal recover” (para este caso particular) desde el Dialplan. Este provoca que el perfil salga a buscar su último estado a la base de datos y regenere reINVITE’s hacia los User Agent’s que participaban en sus salas de conferencia, completando el failover de sesión.

Debido a como está implementada la consulta del perfil hacia a la base de datos, se debe configurar a todos los sistemas operativos que conforman los servidores de conferencia con el mismo “Hostname”.

Esta solución no presenta problemas de escalabilidad, se puede crecer en cantidad de servidores aumentando la cantidad de perfiles existentes, siempre se contará con un sistema de failover “circular” en el sentido que siempre existirá un servidor con un perfil de back-Up, que se corresponderá con un activo de otro servidor.

A continuación se presenta la implementación realizada de este sistema, para el caso de 2 servidores.

2.10.1 Script watchcat.sh

```
#!/bin/bash

SERVICE="freeswitch"

IP="192.168.2.38" ## IP Perfil Activo

PUERTO="5060" ## Puerto Perfil Activo

while true; do

    ./home/watchcat2.sh

    sleep 2

done
```

2.10.2 Script watchcat2.sh

```
#!/bin/bash

if [ "$s_is_on" != "1" ];
then
    if ps ax | grep -v grep | grep $SERVICE > /dev/null
    then
        echo "$SERVICE service running, everything is fine, starting po$
        s_is_on="1"
        sleep 30
        export s_is_on
    fi
fi

if [ "$s_is_on" = "1" ];
then
    if nmap -p $PUERTO $IP | grep closed > /dev/null
    then
        echo "Apagando Interfaz"
        s_is_on="0"
        export s_is_on
        ifconfig eth0 down
        shutdown -P now
    fi
fi
```

El evento que genera el mensaje de “mayday” por parte de los Dispatcher hacia el servidor de conferencia con el perfil “back-up”, se encuentra en el Anexo 1.

2.10.3 Configuración FreeSWITCH Servidor de Conferencias 1

Archivo /usr/local/freeswitch/conf/vars.xml

```
<include>

<X-PRE-PROCESS cmd="set" data="default_password=1234"/>

<X-PRE-PROCESS cmd="set" data="sound_prefix=${sounds_dir}/en/us/callie"/>

<X-PRE-PROCESS cmd="set" data="domain=conference.obelix"/>

<X-PRE-PROCESS cmd="set" data="domain_name=${domain}"/>

<X-PRE-PROCESS cmd="set" data="hold_music=local_stream://moh"/>

<X-PRE-PROCESS cmd="set" data="zrtp_secure_media=true"/>

<X-PRE-PROCESS cmd="set" data="global_codec_prefs=G722,PCMU,PCMA,GSM"/>

<X-PRE-PROCESS cmd="set" data="outbound_codec_prefs=PCMU,PCMA,GSM"/>

<!-- Internal SIP Profile -->

<X-PRE-PROCESS cmd="set" data="internal_auth_calls=true"/>

<X-PRE-PROCESS cmd="set" data="internal_sip_port=5060"/>

<X-PRE-PROCESS cmd="set" data="internal_tls_port=5061"/>

<X-PRE-PROCESS cmd="set" data="internal_ssl_enable=false"/>

<X-PRE-PROCESS cmd="set" data="internal_ssl_dir=${base_dir}/conf/ssl"/>

<!-- External SIP Profile -->

<X-PRE-PROCESS cmd="set" data="external_auth_calls=false"/>

<X-PRE-PROCESS cmd="set" data="external_sip_port=5080"/>

<X-PRE-PROCESS cmd="set" data="external_tls_port=5081"/>

<X-PRE-PROCESS cmd="set" data="external_ssl_enable=false"/>
```

```
<X-PRE-PROCESS cmd="set" data="external_ssl_dir=${base_dir}/conf/ssl"/>
</include>
```

Archivo /usr/local/freeswitch/conf/sip_profiles/internal.xml

```
<profile name="internal">
<settings>
  <param name="sip-trace" value="no"/>
  <param name="sip-capture" value="no"/>
  <param name="track-calls" value="true"/>
  <param name="watchdog-enabled" value="no"/>
  <param name="watchdog-step-timeout" value="30000"/>
  <param name="watchdog-event-timeout" value="30000"/>
  <param name="log-auth-failures" value="false"/>
  <param name="forward-unsolicited-mwi-notify" value="false"/>
  <param name="context" value="default"/>
  <param name="sip-port" value="${internal_sip_port}"/>
  <param name="dialplan" value="XML"/>
  <param name="dtmf-duration" value="2000"/>
  <param name="inbound-codec-prefs" value="${global_codec_prefs}"/>
  <param name="outbound-codec-prefs" value="${global_codec_prefs}"/>
  <param name="rtp-timer-name" value="soft"/>
  <param name="rtp-ip" value="${local_ip_v4}"/>
  <param name="sip-ip" value="192.168.2.38"/>
  <param name="odbc-dsn" value="freeswitch:kerry:obelix"/>
</settings>
</profile>
```

Archivo /usr/local/freeswitch/conf/sip_profiles/external.xml

```
<profile name="external">
  <settings>
    <param name="debug" value="0"/>
    <param name="sip-trace" value="no"/>
    <param name="sip-capture" value="no"/>
    <param name="rfc2833-pt" value="101"/>
    <param name="sip-port" value="${external_sip_port}"/>
    <param name="dialplan" value="XML"/>
    <param name="context" value="public"/>
    <param name="dtmf-duration" value="2000"/>
    <param name="inbound-codec-prefs" value="${global_codec_prefs}"/>
    <param name="outbound-codec-prefs" value="${outbound_codec_prefs}"/>
    <param name="hold-music" value="${hold_music}"/>
    <param name="rtp-timer-name" value="soft"/>
    <param name="local-network-acl" value="localnet.auto"/>
    <param name="manage-presence" value="false"/>
    <param name="auth-calls" value="false"/>
    <param name="rtp-ip" value="${local_ip_v4}"/>
    <param name="sip-ip" value="192.168.2.39"/>
    <param name="odbc-dsn" value="freeswitch:kerry:obelix"/>
  </settings>
</profile>
```

Archivo /usr/local/freeswitch/conf/dialplan/default.xml

```
<include>
```

```
  <context name="default">
```

```
    <extension name="Sofia_Recover">
```

```
      <condition field="destination_number" expression="^mayday$">
```

```
        <action application="system" data="ifconfig eth3 192.168.2.39"/>
```

```
        <action application="system" data="fs_cli --execute='sofia profile external  
recover'"/>
```

```
      </condition>
```

```
    </extension>
```

```
    <extension name="conference_cac_creador">
```

```
      <condition field="destination_number" expression="^c-(3[0-3]\d{2})$">
```

```
        <action application="execute_extension" data="$1"/>
```

```
      </condition>
```

```
    </extension>
```

```
    <extension name="nb_conferences">
```

```
      <condition field="destination_number" expression="^(30\d{2})$">
```

```
        <action application="answer"/>
```

```
        <action application="conference" data="$1-${domain_name}@default"/>
```

```
      </condition>
```

```
    </extension>
```

```

<extension name="conference_cac_publico" continue="true">

    <condition field="destination_number" expression="^p-(3[0-3]\d{2})$"
break="never">

        <action application="set" data="destination_number=$1"
inline="true"/>

    </condition>

</extension>

<extension name="Chequeo_si_existe_sala">

    <condition field="${conference ${destination_number}-${domain_name}
list count}" expression="^\d+">

        <action application="execute_extension"
data="${destination_number}"/>

    </condition>

</extension>

</context>

</include>

```

Archivo /usr/local/freeswitch/conf/dialplan/public.xml

```

<include>

<context name="public">

    <extension name="transfer_to_default">

        <condition>

```



```

        <action    application="transfer"    data="${destination_number}    XML
default"/>

    </condition>

</extension>

</context>

</include>

```

2.10.4 Configuración FreeSWITCH Servidor de Conferencias 2

Archivo /usr/local/freeswitch/conf/vars.xml

```

<include>

    <X-PRE-PROCESS cmd="set" data="default_password=1234"/>

    <X-PRE-PROCESS cmd="set" data="domain=conference.obelix"/>

    <X-PRE-PROCESS cmd="set" data="domain_name=${domain}"/>

    <X-PRE-PROCESS cmd="set" data="hold_music=local_stream://moh"/>

    <X-PRE-PROCESS cmd="set" data="use_profile=internal"/>

    <X-PRE-PROCESS cmd="set"
data="global_codec_prefs=G722,PCMU,PCMA,GSM"/>

    <X-PRE-PROCESS cmd="set"
data="outbound_codec_prefs=PCMU,PCMA,GSM"/>

<!-- Internal SIP Profile -->

    <X-PRE-PROCESS cmd="set" data="internal_auth_calls=true"/>

    <X-PRE-PROCESS cmd="set" data="internal_sip_port=5060"/>

    <X-PRE-PROCESS cmd="set" data="internal_tls_port=5061"/>

    <X-PRE-PROCESS cmd="set" data="internal_ssl_enable=false"/>

```

```

<X-PRE-PROCESS cmd="set" data="internal_ssl_dir=${base_dir}/conf/ssl"/>

<!-- External SIP Profile -->

<X-PRE-PROCESS cmd="set" data="external_auth_calls=false"/>

<X-PRE-PROCESS cmd="set" data="external_sip_port=5080"/>

<X-PRE-PROCESS cmd="set" data="external_tls_port=5081"/>

<X-PRE-PROCESS cmd="set" data="external_ssl_enable=false"/>

<X-PRE-PROCESS cmd="set" data="external_ssl_dir=${base_dir}/conf/ssl"/>

</include>

```

Archivo /usr/local/freeswitch/conf/sip_profiles/internal.xml

```

<profile name="internal">

  <settings>

    <param name="debug" value="0"/>

    <!-- If you want FreeSWITCH to shutdown if this profile fails to load,
    uncomment the next line. -->

    <!-- <param name="shutdown-on-fail" value="true"/> -->

    <param name="sip-trace" value="no"/>

    <param name="sip-capture" value="no"/>

    <param name="watchdog-enabled" value="no"/>

    <param name="watchdog-step-timeout" value="30000"/>

    <param name="watchdog-event-timeout" value="30000"/>

    <param name="log-auth-failures" value="false"/>

    <param name="forward-unsolicited-mwi-notify" value="false"/>

    <param name="context" value="public"/>

```

```
<param name="rfc2833-pt" value="101"/>

<!-- port to bind to for sip traffic -->

<param name="sip-port" value="${internal_sip_port}"/>

<param name="dialplan" value="XML"/>

<param name="dtmf-duration" value="2000"/>

<param name="inbound-codec-prefs" value="${global_codec_prefs}"/>

<param name="outbound-codec-prefs" value="${global_codec_prefs}"/>

<param name="rtp-timer-name" value="soft"/>

<param name="rtp-ip" value="${local_ip_v4}"/>

<param name="sip-ip" value="192.168.2.38"/>

<param name="hold-music" value="${hold_music}"/>

<param name="apply-nat-acl" value="nat.auto"/>

<param name="inbound-codec-negotiation" value="generous"/>

<param name="odbc-dsn" value="freeswitch:kerry:obelix"/>

<param name="auth-all-packets" value="false"/>

</settings>

</profile>
```

Archivo /usr/local/freeswitch/conf/sip_profiles/external.xml

```
<profile name="external">

<settings>

<param name="sip-trace" value="no"/>

  <param name="sip-capture" value="no"/>

  <param name="rfc2833-pt" value="101"/>

  <param name="sip-port" value="${external_sip_port}"/>

  <param name="dialplan" value="XML"/>

  <param name="context" value="default"/>

  <param name="dtmf-duration" value="2000"/>

  <param name="inbound-codec-prefs" value="${global_codec_prefs}"/>

  <param name="outbound-codec-prefs" value="${outbound_codec_prefs}"/>

  <param name="hold-music" value="${hold_music}"/>

  <param name="rtp-timer-name" value="soft"/>

  <param name="local-network-acl" value="localnet.auto"/>

  <param name="manage-presence" value="false"/>

  <param name="track-calls" value="true"/>

  <param name="inbound-codec-negotiation" value="generous"/>

  <param name="auth-calls" value="false"/>

  <param name="rtp-ip" value="${local_ip_v4}"/>

  <param name="sip-ip" value="192.168.2.39"/>

  <param name="ext-rtp-ip" value="auto-nat"/>

  <param name="ext-sip-ip" value="auto-nat"/>

  <param name="rtp-timeout-sec" value="300"/>
```

```
<param name="rtp-hold-timeout-sec" value="1800"/>  
<param name="odbc-dsn" value="freeswitch:kerry:obelix"/>  
<param name="enable-3pcc" value="false"/>  
</settings>  
</profile>
```

Archivo /usr/local/freeswitch/conf/dialplan/default.xml

```

<include>

<context name="default">

  <extension name="Sofia_Recover">

    <condition field="destination_number" expression="^mayday$">

      <action application="system" data="ifconfig eth5 192.168.2.38"/>

      <action application="system" data="fs_cli --execute='sofia profile internal
recover'"/>

    </condition>

  </extension>

  <extension name="conference_cac_creador">

    <condition field="destination_number" expression="^c-(3[0-3]\d{2})$">

      <!-- <action application="set" data="destination_number=$1"/> -->

      <action application="execute_extension" data="$1"/>

      <!-- <action application="execute_extension" data="wb_conferences"/>

      <action application="execute_extension" data="uwb_conferences"/>

      <action application="execute_extension"
data="cdquality_conferences"/> -->

    </condition>

  </extension>

  <extension name="nb_conferences">

    <condition field="destination_number" expression="^(30\d{2})$">

      <action application="answer"/>

      <action application="conference" data="$1-${domain_name}@default"/>

    </condition>

```

```
</extension>

<extension name="conference_cac_publico" continue="true">

    <condition field="destination_number" expression="^p-(3[0-3]\d{2})$"
break="never">

        <action application="set" data="destination_number=$1"
inline="true"/>

    </condition>

</extension>

<extension name="Chequeo_si_existe_sala">

    <condition field="${conference ${destination_number}-${domain_name}
list count}" expression="^\d+">

        <action application="execute_extension"
data="${destination_number}"/>

    </condition>

</extension>

</context>

</include>
```

Archivo /usr/local/freeswitch/conf/dialplan/public.xml

```
<include>

  <context name="public">

    <extension name="transfer_to_default">

      <condition>

        <action application="transfer" data="{destination_number} XML
default"/>

      </condition>

    </extension>

  </context>

</include>
```


Anexo 3 – SIPp

En el Capítulo 3 se mencionó la investigación realizada en materia de aplicaciones generadoras de tráfico SIP. La herramienta utilizada denominada *SIPp* proveyó y se ajustó a las necesidades de pruebas de testeo y carga para generar el tráfico requerido. En este apéndice se detallará los procedimientos requeridos para la instalación en sistemas operativos, la operativa en materia de escenarios y los escenarios utilizados en las diferentes pruebas de funcionalidad y de carga.

1 – Descarga e Instalación

La herramienta de testeo y generadora de tráfico *SIPp* se puede obtener libremente del vínculo <http://sipp.sourceforge.net/>. Una vez descargado el archivo debemos de cumplir ciertos requerimientos para su instalación dependiendo del sistema operativo en que se encuentre.

Las instalaciones se hicieron en dos sistemas operativos diferentes. En CentOS 6.1 que oficio de sistema operativo para todos los sistemas en los que se instaló *FreeSWITCH*, *Asterisk* y *Kamailio* y así poder utilizarlos en pruebas de carga con *SIPp*. En Ubuntu 11.04 se efectuaron instalaciones de *SIPp* para realizar pruebas hacia los servidores con funcionalidades y las pruebas conceptuales para analizar el funcionamiento de los sistemas y análisis del protocolo.

1.1 – CentOS

En CentOS 6.1 los requerimientos previos a su instalación son tener instalado el repositorio `epel-release-6.5.noarch.rpm` que permite la instalación de librerías necesarias de *SIPp*. Este se obtiene mediante el comando, descargando del vínculo²⁵ de repositorios (ver nota al pie de página).

Para instalar el archivo rpm descargado corremos el comando:

²⁵ <http://download.fedoraproject.org/pub/epel/6/i386/epel-release-6-5.noarch.rpm>

```
rpm -i epel-release-6.5.noarch.rpm
```

Luego de instalado se actualizan las listas de repositorios mediante el comando:

```
yum repolist
```

Actualizada la lista, procedemos a la instalación de las siguientes dependencias para compilar *SIPp* mediante el comando `yum install` y luego detallando las librerías a ser instaladas.

```
yum install gcc gcc-c++ ncurses ncurses-devel libpcap libpcap-  
devel libnet libnet-devel
```

```
yum install openssl-devel
```

Estas librerías son las necesarias para poder realizar la compilación de *SIPp*, previamente debemos descomprimir el fichero descargado al principio mediante el comando:

```
tar -xvfz sipp.svn.tar.gz
```

```
cd sipp.svn
```

```
make pcapplay_oss1
```

El último comando, *pcapplay_oss1*, reúne dos particularidades fundamentales de este generador de tráfico. Primero, la librería *libpcap* permite la reproducción de capturas basadas en RTP para la reproducción de audio, particularidad necesaria en pruebas donde se efectúen pruebas de audio. Segundo, la librería *openssl* nos da la capacidad de que *SIPp* soporte autenticación. *SIPp* otorga la flexibilidad de no tener que ser compilado con estas dos librerías, pero dados los requerimientos de pruebas a realizar el comando reúne las dos librerías para poder ejecutar de forma completa las pruebas tanto con audio y autenticación ante el servidor al que se le estén efectuando las pruebas.

Luego de ejecutadas estas sentencias, procedemos a ejecutar en la terminal el comando `sipp o ./sipp` y nos desplegará un manual de utilización. Incluso, para comprobar si la instalación con soporte de audio y autenticación es correcta resta solamente ver al inferior de lo que se despliega al ejecutar el comando `sipp -v`, que nos dará la versión y si tiene el soporte de ambas características.

1.2 – Ubuntu

En Ubuntu el procedimiento de descarga no difiere, lo único a tener en cuenta es la instalación de la herramienta en el sistema. El procedimiento se describe a continuación.

Se resuelven las dependencias antes de ejecutar la compilación de SIPp mediante el comando `sudo apt-get install`,

```
sudo apt-get install gcc libncurses5-dev libncurses5-dev libpcap-dev libpcap0.8-dev libnet1 libnet1-dev libcurl4-openssl-dev
```

Luego se descomprime el archivo descargado mediante el comando de descompresión,

```
tar -xvf sipp.svn.tar.gz
```

Se ingresa al directorio `sipp.svn` y se ejecuta,

```
make pcapplay_oss1
```

De estar las dependencias anteriores resueltas de forma correcta se saldrá del compilador sin errores. Para probar el correcto funcionamiento de la herramienta basta con ejecutar mediante el comando,

```
./sipp
```

Y se nos desplegará los diversos comandos y atributos para utilizarla. De ejecutar `./sipp -v` obtendremos la versión instalada y además nos dirá si está compilada con las librerías para soporte de reproducción de media (pcap) y autenticación (TLS).

Comprobado el correcto funcionamiento de la herramienta generamos un link simbólico para poder ejecutar `sipp` en cualquier directorio en el que se esté logueado,

```
sudo ln -s /home/usuario/sipp.svn/sipp /usr/bin/
```

Solo resta ejecutar nuevamente mediante el comando `sipp` desde cualquier directorio en el que estemos.

2 – Funcionamiento

Dentro de las capacidades de esta herramienta se encuentran los diferentes parámetros que pueden ser incluidos al momento de realizar una prueba de

tráfico o un test a una implementación. Esto permite analizar tanto a nivel conceptual las implementaciones realizadas así también probar su respuesta a nivel de carga. Dependiendo de las variables que se fijen al momento de la ejecución vía terminal, tendremos los diferentes comportamientos.

En la página de la herramienta se encuentra una documentación extensa de todas las particularidades para la que el sistema puede utilizarse. Si bien la documentación que presenta la herramienta no es para novatos, explica cuales son las diferentes flexibilidades al momento de utilizarlo en pruebas de carga y funcionalidad. Incluso en muchas de las referencias que se utilizaron para diversos capítulos de esta documentación se menciona a esta herramienta como la más adecuada para efectuar pruebas de diversa índole en SIP.

2.1 – Atributos

Al ejecutar en una terminal `sipp`, sin ningún parámetro, se nos despliega una variedad de opciones disponibles para utilizar como fue mencionado en párrafos anteriores. Estos parámetros o atributos que se utilizan con una sintaxis de escritura similar a cualquier aplicación de terminal de la Shell de Unix, demuestran la capacidad de esta herramienta para las pruebas a ser realizadas. Dado que para los objetivos del proyecto muchas de las funcionalidades no aportaron, la tabla siguiente resume algunas de las funcionalidades y atributos que se utilizaron y son de gran utilidad para las pruebas efectuadas.

Atributo	Descripción
-v	Despliega la versión instalada de sipp
-aa	habilita la respuesta automática de 200 OK para INFO, UPDATE y NOTIFY
-i	Fija la dirección IP local para el campo Contact, Via y From
-inf	Inyecta valores desde un archivo CSV externo durante las llamadas en el escenario
-l	Fija el máximo número de llamadas simultáneas
-log_file	Fija el nombre del archivo de logs
-m	Termina el test y sale cuando la cantidad que fija m es alcanzada y procesada
-p	Fija el numero de puerto local, por defecto es un puerto aleatorio del sistema
-r	Fija la tasa de llamadas (llamadas por segundo), valor que puede cambiarse
-sf	Carga un escenario xml alternativo que no es el defecto
-trace_err	Hace un trace de todos los mensajes inesperados
-ap	Fija la contraseña para la autenticación del challenge de password
-s	Fija el usuario para el request URI

Tabla A.3.1 – Resumen de parámetros de SIPp

Algunos de los parámetros de más utilidad resaltados en la tabla anterior, son los de ejecución de los escenarios. El parámetro `-m` es fundamental para la realización de pruebas de carga ya que permite que un escenario se repita la cantidad declarada luego de este parámetro. El parámetro `-r` es otro parámetro que si bien es útil para las pruebas de carga, también es útil para las pruebas de funcionalidad ya que con éste fijamos la cadencia de establecimientos por segundo, incluso con las teclas `+` y `*` se logra incrementar este valor ya sea de a uno o en múltiplos de la cadencia respectivamente. Otros parámetros de autenticación como `-ap` o `-s` trabajan de forma directa con la traza SIP incluida en los escenarios ya que al realizar la solicitud de autenticación, declarando de forma debida en el escenario se establecen las credenciales para autenticarse según pida el servidor que se esté probando.

El parámetro `-inf` permite la carga de diferentes usuarios a ser registrados o que intervendrán en alguna prueba ya sea de carga o funcional. Estos se detallan en archivos csv que cumpliendo una sintaxis de columnas, se ingresa de forma masiva en el sistema. Este tipo de comando es de gran utilidad en caso de requerir una prueba de varios usuarios en un mismo escenario.

SIPp permite desplegar *logs* de errores en caso de fallas, esto permite analizar las posibles fallas que se puedan obtener en caso de que el escenario no esté correctamente implementado o de lo contrario, estudiar en qué punto del flujo de señalización se encuentre el fallo para poder efectuar los cambios correspondientes para un correcto funcionamiento.

3 – Escenarios

Algunos detalles a tener en cuenta antes de entrar en los formatos y líneas de ejecución de cada escenario.

Cada escenario escrito en código xml se obtuvo de la página del autor de la herramienta. Ahí cada escenario estaba descripto según las necesidades para las que fueron creados cada uno de estos. Dentro del código xml se agregaron los diferentes paquetes SIP los cuales contaban con la estructura de acuerdo al método requerido, ya sea **INVITE**, **REGISTER**, **BYE** o con las respuestas necesarias como **200 OK** o **ACK**.

Concretamente, la construcción de un escenario demanda previamente un análisis del flujo correcto de señalización que se efectúa al momento de una acción contra el servidor o la aplicación que se esté analizando. A partir de esto se realizan las entradas en el código para poder ejecutarlo con la herramienta *SIPp*. Dependiendo del tipo de prueba a efectuar se construyen escenarios de forma diferente. En los casos de pruebas de registro solo se maneja señalización SIP, en los casos de pruebas de llamadas planas la señalización se ve demorada por un espacio de tiempo que simula ser el audio intercambiado, y en las pruebas de funcionalidades se da la espera necesaria (entre los flujos de intercambio de señalización) para el intercambio de flujo RTP. En algunos escenarios fue necesario detallar los campos del flujo SDP ya que en los **INVITE** se envía la información de la descripción de la sesión.

4 – Escenarios de Prueba Utilizados

Se exponen los escenarios de prueba utilizados para realizar tanto las pruebas conceptuales así como también las pruebas de carga para establecer los límites de capacidad de los diferentes equipos y sistemas utilizados.

4.1 – Registración

Este escenario se corresponde con el diagrama de flujo SIP de la figura 4.8 del Capítulo 4 donde se especifica la señalización para la registración.

La sintaxis de ejecución en terminal

```
sipp -sf registracion.xml ip_servidor -m 1 -r 1 -s user -ap pass -
i ip_local
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<!-- sipp -sf ua_register.xml IP KAMAILIO -m 1 -r 1 -s 1000 -ap 1234 -i IP PC -->

<scenario name="UA register for IPPBX">
  <!-- In client mode (sipp placing calls), the Call-ID MUST be -->
  <!-- generated by sipp. To do so, use [call_id] keyword. -->

  <send retrans="500">
    <![CDATA[

      REGISTER sip:[remote_ip] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:[service]@[remote_ip]>;tag=[pid]SIPpTag07[call_number]
      To: sipp <sip:[service]@[remote_ip]>
      Call-ID: [call_id]
      CSeq: 1 REGISTER
      Contact: <sip:[service]@[local_ip]:[local_port]>
      Content-Length: 0
      Expires: 86400
    ]]>
  </send>

  <recv response="100" optional="true">
  </recv>

  <recv response="401" auth="true" >
  </recv>

  <send retrans="500">
    <![CDATA[

      REGISTER sip:[remote_ip] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:[service]@[remote_ip]>;tag=[call_number]
      To: sipp <sip:[service]@[remote_ip]>
      Call-ID: [call_id]
      CSeq: 2 REGISTER
      Contact: <sip:[service]@[local_ip]:[local_port]>
      [authentication service password]
      Content-Length: 0
      Expires: 86400
    ]]>
  </send>
```

```
    ]]>
  </send>

  <recv response="200" rtd="true" >
  </recv>

  <!-- definition of the response time repartition table (unit is ms) -->
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

  <!-- definition of the call length repartition table (unit is ms) -->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>
</scenario>
```


4.2 – Llamadas Planas

Los escenarios correspondientes con el establecimiento de llamadas planas se especifican según los diagramas de flujo SIP del capítulo 4, específicamente las figuras 4.2, 4.4, 4.6, 4.15 y 4.16. Se describen los dos escenarios utilizados tanto para la parte llamante como para la parte llamada. Los escenarios correspondientes a las figuras 4.2, 4.4 y 4.6 son escenarios para la prueba de acumulación de llamadas, estos tienen un tiempo largo de espera luego de establecida la sesión. Los escenarios de las figuras 4.15 y 4.16 se corresponden con diferentes demoras, esto repercute de forma diferente para la prueba efectuada ya que la idea es analizar como impacta la variación de tiempo de cada llamada y a su vez el tiempo de repetición del escenario permite ajustar la cantidad de acumulación de llamadas en el sistema.

4.2.1 – Llamador

La sintaxis de ejecución en terminal es

```
sipp -sf llamador_proxy.xml ip_remota -m -r -s user -ap pass -i
ip_local
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<!-- el usuario 1000 llama al 1019 a traves del proxy.
ejecucion:
sipp -sf llamador_proxy IP KAMAILIO -m -r -s 1000 -ap PASS -i IP PC -->
<scenario name="llamador a traves de kamailio">

    <send retrans="20000">
        <![CDATA[

            INVITE sip:1019@[remote_ip]:[remote_port] SIP/2.0
            Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
            From: user1
<sip:1000@[remote_ip]:[remote_port]>;tag=[pid]SIPpTag09[call_number]
            To: user2 <sip:1019@[remote_ip]:[remote_port]>
            Call-ID: [call_id]
            CSeq: 1 INVITE
            Contact: sip:user1@[local_ip]:[local_port]
            Max-Forwards: 70
            Subject: Performance Test
            Content-Type: application/sdp
            Content-Length: [len]

            v=0
            o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
            s=-
            c=IN IP[local_ip_type] [local_ip]
            t=0 0
            m=audio [auto_media_port] RTP/AVP 8 101
            a=rtpmap:8 PCMU/8000
            a=rtpmap:101 telephone-event/8000
```

```

        a=fmtp:101 0-11,16

    ]]>
</send>

<recv response="100" optional="true">
</recv>

<recv response="407" auth="true" >
</recv>

<!-- By adding rrs="true" (Record Route Sets), the route sets      -->
<!-- are saved and used for following messages sent. Useful to test -->
<!-- against stateful SIP proxies/B2BUAs.                          -->

<!-- Packet lost can be simulated in any send/recv message by      -->
<!-- by adding the 'lost = "10"'. Value can be [1-100] percent.   -->
<send>
    <![CDATA[

        ACK sip:1019@[remote_ip]:[remote_port] SIP/2.0
        Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
        From: user1
<sip:1000@[remote_ip]:[remote_port]>;tag=[pid]SIPpTag09[call_number]
        To: user2 <sip:1019@[remote_ip]:[remote_port]>[peer_tag_param]
        Call-ID: [call_id]
        CSeq: 1 ACK
        Contact: sip:user1@[local_ip]:[local_port]
        Max-Forwards: 70
        Subject: Performance Test
        Content-Length: 0

    ]]>
</send>

<send retrans="500">
    <![CDATA[

        INVITE sip:1019@[remote_ip]:[remote_port] SIP/2.0
        Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch+1]
        From: user1
<sip:1000@[remote_ip]:[remote_port]>;tag=[pid]SIPpTag09[call_number]
        To: user2 <sip:1019@[remote_ip]:[remote_port]>
        Call-ID: [call_id]
        CSeq: 2 INVITE
        Contact: sip:user1@[local_ip]:[local_port]
        Max-Forwards: 70
        Subject: Performance Test
        [authentication service password]
        Content-Type: application/sdp
        Content-Length: [len]

        v=0
        o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
        s=-
        c=IN IP[local_ip_type] [local_ip]
        t=0 0
        m=audio [auto_media_port] RTP/AVP 8 101
        a=rtpmap:8 PCMU/8000
        a=rtpmap:101 telephone-event/8000
        a=fmtp:101 0-11,16

    ]]>
</send>

<recv response="100" optional="true">
</recv>

<recv response="180" optional="false">
</recv>

```

```

<recv response="200" rtd="true" rrs="true">

</recv>

<send>
  <![CDATA[

    ACK sip:user2@192.168.2.11:5060 SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch+2]
    From: user1
<sip:1000@[remote_ip]:[remote_port]>;tag=[pid]SIPpTag09[call_number]
    To: user2 <sip:1019@[remote_ip]:[remote_port]>[peer_tag_param]
    Call-ID: [call_id]
    CSeq: 3 ACK
    [routes]
    Contact: sip:user1@[local_ip]:[local_port]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>

<pause milliseconds="100000"/>

<send retrans="500">
  <![CDATA[

    BYE sip:user2@192.168.2.11:5060 SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch+3]
    From: user1
<sip:1000@[remote_ip]:[remote_port]>;tag=[pid]SIPpTag09[call_number]
    To: user2 <sip:1019@[remote_ip]:[remote_port]>[peer_tag_param]
    Call-ID: [call_id]
    CSeq: 3 BYE
    Contact: sip:user1@[local_ip]:[local_port];transport=[transport]
    [routes]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>

<recv response="200" crlf="true">
</recv>

<!-- definition of the response time repartition table (unit is ms) -->
  <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

  <!-- definition of the call length repartition table (unit is ms) -->
  <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>

```

4.2.1 - Llamado

La sintaxis de ejecución para este escenario es

```
sipp -sf llamado_proxy.xml ip_remota -i ip_local
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```

<!DOCTYPE scenario SYSTEM "sipp.dtd">
<!-- ejecucion: sipp -sf llamado_proxy.xml IP KAMAILIO -i 192.168.2.14 -->
<scenario name="llamado a traves de kamailio">
  <!-- By adding rrs="true" (Record Route Sets), the route sets -->
  <!-- are saved and used for following messages sent. Useful to test -->
  <!-- against stateful SIP proxies/B2BUAs. -->
  <recv request="INVITE" crlf="true">
    </recv>

    <!-- The '[last_*]' keyword is replaced automatically by the -->
    <!-- specified header if it was present in the last message received -->
    <!-- (except if it was a retransmission). If the header was not -->
    <!-- present or if no message has been received, the '[last_*]' -->
    <!-- keyword is discarded, and all bytes until the end of the line -->
    <!-- are also discarded. -->
    <!-- -->
    <!-- If the specified header was present several times in the -->
    <!-- message, all occurrences are concatenated (CRLF seperated) -->
    <!-- to be used in place of the '[last_*]' keyword. -->

    <send>
      <![CDATA[

        SIP/2.0 100 Trying
        [last_Via:]
        [last_From:]
        [last_To:]
        [last_Call-ID:]
        [last_CSeq:]
        Contact: sip:user2@[local_ip]:[local_port];transport=[transport]
        [last_Record-route:]
        Content-Length: 0

      ]]>
    </send>

    <send>
      <![CDATA[

        SIP/2.0 180 Ringing
        [last_Via:]
        [last_From:]
        [last_To:];tag=[call_number]
        [last_Call-ID:]
        [last_CSeq:]
        Contact: sip:user2@[local_ip]:[local_port];transport=[transport]
        [last_Record-route:]
        Content-Length: 0

      ]]>
    </send>

    <pause milliseconds="500"/>

    <send retrans="500">
      <![CDATA[

        SIP/2.0 200 OK
        [last_Via:]
        [last_From:]
        [last_To:];tag=[call_number]
        [last_Call-ID:]
        [last_CSeq:]
        Contact: sip:user2@[local_ip]:[local_port];transport=[transport]
        [last_Record-route:]
        Content-Type: application/sdp
        Content-Length: [len]

        v=0
        o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
        s=-
        c=IN IP[media_ip_type] [media_ip]

```

```
t=0 0
m=audio [media_port] RTP/AVP 0
a=rtpmap:0 PCMU/8000

]]>
</send>

<recv request="ACK" rrs="true" rtd="true" crlf="true">
</recv>

<recv request="BYE" rtd="true">
</recv>

<send>
  <![CDATA[

    SIP/2.0 200 OK
    [last_Via]
    [last_From]
    [last_To]
    [last_Call-ID]
    [last_CSeq]
    Contact: sip:user2@[local_ip]:[local_port]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>
```

4.3 – Music On Hold (MOH)

Esta es una prueba de carga a la funcionalidad de música en espera esta se realiza de igual forma que una llamada plana, con la diferencia que se deja un espacio de tiempo luego del establecimiento de la llamada y así recibir flujo RTP para probar la cantidad de solicitudes soportadas por parte del servidor de MOH. Este escenario se corresponde con las pruebas efectuadas al servidor de MOH para el dimensionamiento del mismo.

La línea de sintaxis de ejecución en terminal es:

```
sipp -sf prueba_moh.xml ip_mohserver -m -r -s user -ap pass -i
ip_local
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<!-- el usuario 1000 llama al 2000 a traves del proxy.
ejecucion:
sipp -sf prueba_moh.xml IP_MOHSERVER -m -r -s 1000 -i IP PC -->
<scenario name="prueba de carga a servidor de MOH">
  <send retrans="500">
    <![CDATA[
      INVITE sip:moh@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip];branch=[branch]
      From: 2000 <sip:2000@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
      To: moh <sip:moh@[remote_ip]>
      Call-ID: [call_id]
      CSeq: [cseq] INVITE
      Contact: sip:2000@[local_ip]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: [len]

      v=0
      o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
      s=-
      c=IN IP[local_ip_type] [local_ip]
      t=0 0
      m=audio [auto_media_port] RTP/AVP 8 101
      a=rtpmap:8 PCMU/8000
      a=rtpmap:101 telephone-event/8000
      a=fmtp:101 0-11,16

    ]]>
  </send>

  <recv response="100" optional="true">
</recv>

  <recv response="200" rrs="true" rtd="true">
</recv>

  <send>
    <![CDATA[
      ACK sip:moh@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip];branch=[branch]
      From: 2000 <sip:2000@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
      [last_To:]
      Call-ID: [call_id]
      Contact: sip:2000@[local_ip]
```

```

        CSeq: [cseq] ACK
        Max-Forwards: 70
        Subject: Performance Test
        Content-Length: 0

    ]]>
</send>

<pause milliseconds="10600"/>

<send retrans="500">
    <![CDATA[
        BYE sip:moh@[remote_ip]:[remote_port] SIP/2.0
        Via: SIP/2.0/[transport] [local_ip];branch=[branch]
        From: 2000 <sip:2000@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
        [last_To:]
        Call-ID: [call_id]
        CSeq: [cseq] BYE
        Contact: sip:2000@[local_ip]
        Max-Forwards: 70
        Subject: Performance Test
        Content-Length: 0
    ]]>
</send>

<recv response="100" optional="true">
</recv>

<recv response="200" crlf="true">
</recv>

<!-- definition of the response time repartition table (unit is ms) -->
<ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

<!-- definition of the call length repartition table (unit is ms) -->
<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000, 50000,
100000, 500000"/>

</scenario>

```

4.4 – Voicemail

La prueba de la funcionalidad de *voicemail* se realiza en rasgos generales de igual modo que la prueba de MOH, la diferencia es que en este caso se envía un flujo de RTP el cual contiene los tonos en banda que responden al IVR de quien gestiona el correo de voz. Esta prueba se utiliza para dos objetivos, en primer lugar dimensionar el servidor de Voicemail a nivel de capacidad de canales de audio y en segundo lugar para dimensionar la capacidad del disco del equipo. Razón por la cual antes de realizar la prueba se envía una captura tomada con *Wireshark* para poder reproducir el audio y así grabar en el servidor. Requerido en el escenario ya que en la pausa se envía el flujo RTP capturado.

La sintaxis de ejecución en terminal es:

```
sipp -sf prueba_vcm_FS.xml ip_vociemail -m -r -s user -ap pass -i
ip_local
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<!-- el usuario 1002 llama al 1010 y le responde el voicemail.
ejecucion:
sipp -sf prueba_vcm_FS.xml IP_VOICEMAIL -m -r -s 1002 -ap PASS -i IP PC -->
<scenario name="prueba de carga a servidor de VOICEMAIL">

  <send retrans="500">
    <![CDATA[

      INVITE sip:1010@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip];branch=[branch]
      From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
      To: 1010 <sip:1010@[remote_ip]>
      Call-ID: [call_id]
      CSeq: 1 INVITE
      Contact: sip:1002@[local_ip]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: [len]

      v=0
      o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
      s=-
      c=IN IP[local_ip_type] [local_ip]
      t=0 0
      m=audio [auto_media_port] RTP/AVP 0 101
      a=rtpmap:0 PCMU/8000
      a=rtpmap:101 telephone-event/8000
      a=fmtp:101 0-11,16

    ]]>
  </send>

  <recv response="100" optional="true">
</recv>

  <recv response="407" auth="true">
</recv>
```



```

<!-- By adding rrs="true" (Record Route Sets), the route sets      -->
<!-- are saved and used for following messages sent. Useful to test -->
<!-- against stateful SIP proxies/B2BUAs.                          -->

<!-- Packet lost can be simulated in any send/recv message by      -->
<!-- by adding the 'lost = "10"'. Value can be [1-100] percent.   -->

<send>
  <![CDATA[

    ACK sip:1010@[remote_ip] SIP/2.0
    [last_Via:]
    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    [last_To:]
    Call-ID: [call_id]
    CSeq: [cseq] ACK
    Contact: sip:1002@[local_ip]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>

<send retrans="500">
  <![CDATA[

    INVITE sip:1010@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip];branch=[branch]
    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    To: 1010 <sip:1010@[remote_ip]>
    Call-ID: [call_id]
    CSeq: [cseq] INVITE
    Contact: sip:1002@[local_ip]
    Max-Forwards: 70
    Subject: Performance Test
    [authentication service password]
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
    s=-
    c=IN IP[local_ip_type] [local_ip]
    t=0 0
    m=audio [auto_media_port] RTP/AVP 0 101
    a=rtpmap:0 PCMU/8000
    a=rtpmap:101 telephone-event/8000
    a=fmtp:101 0-11,16

  ]]>
</send>

<recv response="100" optional="true">
</recv>

<recv response="200" rrs="true" rtd="true">
</recv>

<send>
  <![CDATA[

    ACK sip:1010@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip];branch=[branch]
    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    [last_To:]
    Call-ID: [call_id]
    Contact: sip:1002@[local_ip]
    CSeq: [cseq] ACK
  ]]>

```

```

        Max-Forwards: 70
        Subject: Performance Test
        Content-Length: 0

    ]]>
</send>

<nop>
  <action>
    <exec play_pcap_audio="salida"/>
  </action>
</nop>

<pause milliseconds="40000"/>

<recv request="BYE">
</recv>

<send>
  <![CDATA[

        SIP/2.0 200 OK
        [last_Via:]
        [last_From:]
        [last_To:];tag=[call_number]
        [last_Call-ID:]
        [last_CSeq:]
        Contact: sip:1002@[local_ip]
        Content-Length: 0
        Expires: 300

    ]]>
</send>

<!-- definition of the response time repartition table (unit is ms) -->
<ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

<!-- definition of the call length repartition table (unit is ms) -->
<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000, 50000,
100000, 500000"/>

</scenario>

```

4.5 – Conferencia

La prueba de conferencia es una idea similar a la de *voicemail* con la idea de tener dos terminales corriendo el mismo escenario con internos diferentes y que de esta forma envíen un flujo RTP para probar la funcionalidad del servidor de conferencias. Ambos escenarios deben tener una captura hecha en *Wireshark* ya que el escenario en la pausa de tiempo reproduce esa captura enviando el *stream* de audio obtenido de la captura RTP.

4.5.1 – Un Usuario Accediendo al Servidor de Conferencia

Este escenario corresponde con la prueba de un usuario accediendo a una conferencia ya establecida.

La sintaxis de ejecución en terminal es:

```
sipp -sf prueba_conf_FS.xml ip_confserv -m -r -s user -ap pass -i
ip_local
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<!-- el usuario 1002 llama al 3000 y le responde el voicemail.
ejecucion:
sipp -sf prueba_conf_FS.xml IP_CONFERENCE -m -r -s 1002 -ap PASS -i IP_PC -->
<scenario name="prueba de carga a servidor de CONFERENCE">
```

```
<send retrans="500">
  <![CDATA[

    INVITE sip:3000@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip];branch=[branch]
    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    To: 3000 <sip:3000@[remote_ip]>
    Call-ID: [call_id]
    CSeq: 1 INVITE
    Contact: sip:1002@[local_ip]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
    s=-
    c=IN IP[local_ip_type] [local_ip]
    t=0 0
    m=audio [auto_media_port] RTP/AVP 0 101
    a=rtpmap:0 PCMU/8000
    a=rtpmap:101 telephone-event/8000
    a=fmtp:101 0-11,16

  ]]>
</send>
```

```

<recv response="100" optional="true">
</recv>

<recv response="407" auth="true">
</recv>

<!-- By adding rrs="true" (Record Route Sets), the route sets      -->
<!-- are saved and used for following messages sent. Useful to test -->
<!-- against stateful SIP proxies/B2BUAs.                          -->

<!-- Packet lost can be simulated in any send/recv message by     -->
<!-- by adding the 'lost = "10"'. Value can be [1-100] percent.   -->

<send>
  <![CDATA[

    ACK sip:3000@[remote_ip] SIP/2.0
    [last_Via:]
    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    [last_To:]
    Call-ID: [call_id]
    CSeq: [cseq] ACK
    Contact: sip:1002@[local_ip]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>

<send retrans="500">
  <![CDATA[

    INVITE sip:3000@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip];branch=[branch]
    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    To: 3000 <sip:3000@[remote_ip]>
    Call-ID: [call_id]
    CSeq: [cseq] INVITE
    Contact: sip:1002@[local_ip]
    Max-Forwards: 70
    Subject: Performance Test
    [authentication service password]
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
    s=-
    c=IN IP[local_ip_type] [local_ip]
    t=0 0
    m=audio [auto_media_port] RTP/AVP 0 101
    a=rtpmap:0 PCMU/8000
    a=rtpmap:101 telephone-event/8000
    a=fmtp:101 0-11,16

  ]]>
</send>

<recv response="100" optional="true">
</recv>

<recv response="200" rrs="true" rtd="true">
</recv>

<send>
  <![CDATA[

    ACK sip:3000@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip];branch=[branch]
  ]]>

```

```

    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    [last_To:]
    Call-ID: [call_id]
    Contact: sip:1002@[local_ip]
    CSeq: [cseq] ACK
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>

<nop>
  <action>
    <exec play_pcap_audio="cap02"/>
  </action>
</nop>

<pause milliseconds="33020"/>

<send retrans="500">
  <![CDATA[

    BYE sip:3000@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip];branch=[branch]
    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    [last_To:]
    Call-ID: [call_id]
    CSeq: [cseq] BYE
    Contact: sip:1002@[local_ip]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>

<recv response="200" crlf="true">
</recv>

<!-- definition of the response time repartition table (unit is ms) -->
<ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

<!-- definition of the call length repartition table (unit is ms) -->
<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000, 50000,
100000, 500000"/>

</scenario>

```

4.5.2 – Prueba de Generación de Oyentes de Conferencia Establecida

El escenario que se describe a continuación corresponde a la generación de los oyentes de la conferencia que ya está establecida.

La sintaxis de ejecución en terminal es:

```
sipp -sf escuchas.xml ip_serv_conferencias -m -r -s user -ap pass
-i ip_local
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<!-- el usuario 1002 llama al 3000 y le responde el conference.
ejecucion:
sipp -sf escuchas.xml IP_Conferencias -m -r -s 1002 -ap PASS -i IP PC -->
<scenario name="prueba de carga a servidor de Conferencias">
```

```
<send retrans="500">
  <![CDATA[
    INVITE sip:3000@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip];branch=[branch]
    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    To: 3000 <sip:3000@[remote_ip]>
    Call-ID: [call_id]
    CSeq: 1 INVITE
    Contact: sip:1002@[local_ip]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
    s=-
    c=IN IP[local_ip_type] [local_ip]
    t=0 0
    m=audio [auto_media_port] RTP/AVP 0 101
    a=rtpmap:0 PCMU/8000
    a=rtpmap:101 telephone-event/8000
    a=fmtp:101 0-11,16

  ]]>
</send>

<recv response="100" optional="true">
</recv>

<recv response="200" rrs="true" rtd="true">
</recv>

<send>
  <![CDATA[
    ACK sip:3000@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip];branch=[branch]
    From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
    [last_To:]
    Call-ID: [call_id]
    Contact: sip:1002@[local_ip]
    CSeq: [cseq] ACK
    Max-Forwards: 70
    Subject: Performance Test
```

```

        Content-Length: 0

    ]]>
</send>

<pause milliseconds="40000"/>

<pause milliseconds="40000"/>

<pause milliseconds="40000"/>

<send retrans="500">
    <![CDATA[

        BYE sip:3000@[remote_ip]:[remote_port] SIP/2.0
        Via: SIP/2.0/[transport] [local_ip];branch=[branch]
        From: 1002 <sip:1002@[remote_ip]>;tag=[pid]SIPpTag09[call_number]
        [last_To:]
        Call-ID: [call_id]
        CSeq: [cseq] BYE
        Contact: sip:1002@[local_ip]
        Max-Forwards: 70
        Subject: Performance Test
        Content-Length: 0

    ]]>
</send>

<recv response="200" crlf="true">
</recv>

<!-- definition of the response time repartition table (unit is ms) -->
<ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

<!-- definition of the call length repartition table (unit is ms) -->
<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000, 50000,
100000, 500000"/>

</scenario>

```

Glosario

- Accounting:** Registro de eventos de interés, como puede ser el inicio y fin de una comunicación.
- AVP:** Tipo de variable de Kamailio que se asocia a una determinada transacción y vive solamente durante el procesamiento de esta.
- B2BUA:** Back-To-Back User Agent. Software capaz interactuar con ambos extremos de una sesión en forma independiente.
- Backbone:** principales conexiones troncales de Internet.
- BIND9:** Software libre que implementa un servidor de DNS (Bind) en su novena versión.
- Branch:** Mismo concepto que transacción.
- Busy Hour:** Hora en la cual se cursa el mayor tráfico ofrecido.
- Callee:** User Agent destinatario del inicio de sesión.
- CDR:** Charging Data Record.
- CentOS:** Distribución de Linux basada en Red Hat.
- Cluster:** Agrupación de servidores que, visto por otro elemento del sistema, se comportan como uno solo.
- Códec:** Abreviatura de codificador-decodificador. Describe una especificación desarrollada en software, hardware o una combinación de ambos, capaz de codificar y decodificar un flujo de datos (stream) o una señal.
- Conf:** Grupo de usuarios con permiso de acceso a salas de conferencias previamente creadas.
- Conf-creator:** Grupo de usuarios con permisos de creación de conferencias.
- Core:** Núcleo. Parte central.
- DDNS:** DNS dinámico. Funcionalidad que permite a un DNS ser actualizado en tiempo real.
- Diálogo:** Conversación SIP entre dos agentes de principio a fin.
- Dialplan:** Lógica programada en la PBX. Encargada de enrutar llamadas que arriban a la misma hacia un destino prefijado.
- Dispatcher:** Despachador. Repartidor de la carga mediante alguna política.
- DNS:** Sistema de nombres de dominio. Es un sistema de nomenclatura jerárquica para computadoras, servicios o cualquier recurso conectado a Internet o a una red privada.
- DTMF:** Dual Tone Multi Frequency. Mapeo de dígitos discados por el usuario a la superposición de 2 tonos diferentes de audio.
- ENUM:** E.164 Number Mapping.
- Erlang:** Unidad básica de teletráfico.
- Erlang-B:** Fórmula de pérdida de Erlang. Deriva de la probabilidad de bloqueo de la distribución de Erlang para describir la probabilidad de pérdida de llamada en un grupo de circuitos.
- Failover:** Recuperación automática de un sistema ante fallos en algún elemento.
- Feature:** Funcionalidad.
- Flooding:** Ataque de red que consiste en inundar la red con mensajes con el fin de sobrecargar, y así anular, algún servicio.
- Fork:** Proyectos que han avanzado habiendo partido de uno en común en la historia de su desarrollo.
- FreeSWITCH:** IP PBX.

Freeware: software que esta disponible para su uso, sin costo o por un costo opcional, pero a su vez con derechos de uso restringido.

Gateway: nodo de red equipado para interconectar redes que operan con diferentes protocolos

GMT: Greenwich Mean Time.

GW: Ver Gateway.

H.323: recomendación de la ITU-T, que define los protocolos para proveer sesiones de comunicación audiovisual sobre paquetes de red.

HA: Alta disponibilidad.

Hardphone: Teléfono físico capaz de conectarse a la red de datos.

HD: Alta Definición.

Header: Encabezado. Un conjunto de estos conforman el cuerpo de un mensaje SIP.

Hostname: nombre del host.

HTTP: Hypertext Transfer Protocol.

IAx: Inter Asterisk Exchange.

IETF: Internet Engineering Task Force, organización internacional abierta de normalización que contribuye a la ingeniería de Internet, regula las propuestas y estándares para este.

In-Dialog: Mensaje que es parte de un diálogo previamente existente.

IPv4: Internet Protocol versión 4.

IPv6: Internet Protocol versión 6.

ITU-T: Ramal de la International Telecommunication Unit que coordina las estandarizaciones a nivel de telecomunicaciones.

IVR: Interactive Voice Response. Consiste en un sistema telefónico que es capaz de recibir una llamada e interactuar con el humano a través de grabaciones de voz y el reconocimiento de respuestas simples, por ejemplo tonos DTMF.

Jitter: variabilidad del retraso temporal durante el envío de señales digitales.

Kamailio: Software libre que implementa un SIP Proxy y otras funcionalidades SIP.

LDAP: Lightweight Directory Access Protocol.

Ley-A: sistema de cuantificación logarítmica de señales de audio definido en el estándar de la ITU-T G.711.

Ley-U: sistema de cuantificación logarítmica de señales de audio definido en el estándar de la ITU-T G.711.

Líneas Urbanas: Líneas utilizadas para comunicaciones telefónicas con destino externo a la empresa.

Loop: iteración indefinida.

Media: Datos intercambiados entre 2 agentes posteriormente al inicio de sesión. Independiente a la señalización.

Metadata: Datos que describen otros datos.

Mixer: se denomina mixer a la entidad encargada de hacer la mezcla de los flujos de audio en una conferencia.

Modelo OSI: Open System Interconnection, modelo de interconexión de sistemas abiertos, modelo de red descriptivo realizado por la Organización Internacional para la Estandarización (ISO) en 1984.

MoH: Music On Hold. Música de espera.

MPEG: Moving Picture Experts Group.

MWI: Message Waiting Indicator. Es un mecanismo que informa al suscriptor el estado de sus mensajes en espera.

MySQL: Software libre que implementa un motor para bases de datos.

NFS: Sistema de archivos compartidos a través de una red de datos.

NSUPDATE: Comando unix que permite actualizar en tiempo real la información en un DNS.

Out of Dialog: Mensaje Inicial de un nuevo diálogo. Estos mensajes no cuentan con Tag en el Header To, por ende abren un nuevo diálogo.

Payload: es la carga útil en una transmisión de datos.

PBX: Private Branch Exchange, central de conmutación telefónica.

PCM: Pulse Code Modulation.

Porcentaje de bloqueo: Porcentaje de las llamadas que no pueden ingresar al sistema por estar este ocupado en su totalidad.

Proceso hijo: Proceso creado por otro que hereda todos los atributos de este (proceso padre).

QoS: Quality of Service (Calidad de Servicio).

Request: Mensaje SIP generado por un UA con el fin de establecer un nuevo diálogo o modificar el estado de uno ya existente. Todo request debe tener asociado una Response. Terminología heredada de HTTP.

Response: Respuesta a un mensaje Request.

RFC: Request for Comments, serie de notas sobre Internet y sobre sistemas que se conectan a Internet.

RTP: Real Time Protocol, protocolo utilizado para la transmisión de información en tiempo real, como por ejemplo audio y video.

SDP: Session Description Protocol, protocolo utilizado para la negociación de recursos entre User Agents (IP, Puertos, Codec's, etc).

SIP: Protocolo de Inicio de Sesión definido en RFC 3261.

SIP Registrar: Servidor SIP encargado de procesar solicitudes de registro.

Softphone: Teléfono enteramente implementado con software.

Spof: Punto único de falla.

Stack: Implementación en software de un protocolo en forma en "capas".

Stateful: Refiere a la capacidad de un servidor SIP de guardar registro de las transacciones que se cursan por el mismo.

Store-and-forward: nodos en los que la información se guarda y luego se envía.

TCP: Transmission Control Protocol.

Teletráfico: Cualquier tipo de tráfico de comunicación de datos o telecomunicaciones.

TLS: Transport Layer Security.

Top: Comando Unix para monitorear los procesos activos en un computador, así como el uso de los recursos del sistema por estos.

Transacción: Parte de un diálogo SIP entre dos agentes que comienza con una request y termina con una respuesta final (mensajes 2xx-6xx).

Transcoding: Tomar datos codificados con determinado codec, decodificarlos y codificarlos con un codec diferente.

Transfer Target: User Agent al cual el Transferer es transferido.

Transferee: es el User Agent transferido hacia otro interno.

Transferor: User Agent que decide realizar la transferencia.

TTL: Tiempo de validez de una respuesta a una consulta a un DNS.

Tupla: Secuencia ordenada de objetos.

UDP: User Datagram Protocol.

URI: Uniform Resource Identifier, en el contexto SIP usualmente usuario@dominio.

URL: Uniform Resource Locator.

User Agent Holdee: User Agent puesto en espera por otro.

User Agent Holder: User Agent que coloca en espera a otro.

User Agents: Aplicación Informática capaz de dialogar con otras entidades bajo un mismo protocolo. Dependiendo el caso como cliente o servidor.

XMPP: Extensible Messaging and Presence Protocol.

**Proyecto de fin de estudios
en la carrera
Ingeniería Eléctrica**

OBELIX

Plan de Proyecto

Autores: Gonzalo De Paolis
Pablo Bethencourt
Gustavo Vidovich

Tutor: Ing. José Joskowicz

Fecha: 4/10/2011

Resumen

•Estudiantes

Pablo Bethencourt 3705804-2
Gustavo Vidovich 4146769-3
Gonzalo De Paolis 4740351-0

•Cliente

Intendencia de Montevideo, Sector Telecomunicaciones.

•Tutor

Ing. José Joskowicz

•Fecha Prevista de finalización: 30/09/2012

•Total de horas a realizar previstas por el grupo de proyecto: ver Planificación de costos.

Entregables:

Primer entregable (15-02-2012)

•Análisis del universo de combinaciones disponibles de Software para la elección final de un grupo manejable (en el tiempo del proyecto) de posibilidades.

Segundo entregable (15-06-2012)

- Elección de la arquitectura del núcleo. Presentación de diagramas de flujo SIP correspondientes a casos de uso de la arquitectura elegida.
- Fundamentos de la arquitectura elegida.
- Pruebas de funcionamiento de la arquitectura.
- Pruebas de carga y stress del sistema con las distintas soluciones.
- Prototipo parcialmente funcional.

Tercer entregable (30-09-2012)

•Pruebas de disponibilidad, carga y stress del sistema con manejo de audio en el core. Presentación de documentación final.

Descripción del Proyecto:

La rápida convergencia de las IT hacia las redes de paquetes ha hecho que muchos sistemas de telecomunicaciones migren de tecnología buscando explotar al máximo todas las bondades de estas redes. En este caso se trata de un núcleo de telefonía clásica siendo sustituido por uno de telefonía IP basado en soluciones de tipo open source. Debido a las dimensiones de esta red, no todas las soluciones existentes podrían cumplir con lo esperado.

Objetivo General:

El proyecto tiene como objetivo realizar el diseño de un núcleo de telecomunicaciones corporativo, basado en proyectos del tipo "open source", utilizando como base el protocolo SIP.

Alcance:

Actualmente en el mercado local no existen implementaciones basadas en proyectos "open source" que puedan soportar altas capacidades de usuarios y tráfico (10.000 @ 0.15 Erl). Es de interés del proyecto y del cliente poder verificar su viabilidad, y/o identificar cuales son sus principales puntos que deberían mejorarse para lograrlo.

El proyecto no tiene como alcance lograr una implementación completa y funcional según especificaciones determinadas por el mercado. Pero si lograr un prototipo parcialmente funcional que pueda soportar la cantidad de usuarios y el perfil de tráfico descrito anteriormente en base a pruebas de diferentes configuraciones tanto de software como hardware.

Esta dentro del alcance del proyecto lograr un sistema con las funcionalidades de SIP Proxy / SIP Registrar, validación de usuarios y otros requerimientos deseables que serán detallados a continuación. Además se analizarán diferentes arquitecturas de implementación para seleccionar la mejor según pruebas de performance y funcionalidad a convenir.

Se dimensionará el hardware a ser empleado mediante pruebas de carga simuladas, de manera de evaluar el desempeño de las distintas piezas de software a analizar. También se espera obtener reglas de dimensionamiento de hardware para ser utilizado en futuras proyecciones. Se evaluarán las opciones y alternativas de manejar el audio en forma centralizada o en un esquema "peer to peer" determinando ventajas y desventajas de ambas configuraciones.

Podría ser necesario desarrollar herramientas de generación de tráfico acorde a las pruebas que se pretenden realizar.

No está dentro del alcance del proyecto el provisioning. (Configuración remota en masa de 'user agents')

Criterios de Éxito:

- Encontrar un conjunto de piezas de software que integradas cumplan con todos los requerimientos funcionales.
- Obtener una arquitectura que soporte el perfil de tráfico ofrecido (0,15Erl@10000)
- Lograr un ambiente de pruebas ("maqueta") de nuevas funcionalidades apropiado.

Actores:

- Tutor
- Equipo de proyecto
- Equipo de telecomunicaciones de la Intendencia de Montevideo
- Proveedores de servidores, hardphones, gateways, equipos de networking
- Proyectos Open Source a utilizar.

Supuestos:

- Se dispondrá de servidores para pruebas, hardphones y gateways en condiciones operativas para trabajar en IM, a partir de comenzado el proyecto.
- Se podrán reponer en tiempo y forma aquellos componentes que por fallas así lo requieran.
- Se cuenta con 1 hora por semana de mentoring por parte del equipo de telecomunicaciones de la IM a partir del comienzo del proyecto.
- Se cuenta con 1 hora de dedicación semanal por parte del tutor para consultas.

Restricciones:

- El proyecto debe estar terminado y documentado para el 30 de setiembre del 2012
- Se debe realizar un primer entregable el 15 de febrero
- Se debe realizar un segundo entregable el 15 de junio
- El equipo de proyecto dispone de una dedicación de 6 horas por semana desde la fecha hasta enero, a partir del cual 2 personas tendrán dedicación fulltime y la tercera 5 horas los días hábiles y fulltime los fines de semana.
- No se cuenta con un lugar físico para realizar pruebas en la IM 24/7, únicamente días hábiles en horario operativo.

Especificación Funcional del Proyecto:

Soportar señalización SIP: Debe soportar la conexión de teléfonos de hardware y software con señalización SIP, así como Gateways TDM con señalización SIP

Alta disponibilidad: Ninguna falla de un componente puntual podrá afectar en forma completa al servicio.

Posibilidad de cambio de configuraciones “en caliente”: Los cambios de configuración no deben requerir el reinicio total o parcial del sistema

Soportar funciones telefónicas corporativas: Se espera que el sistema pueda disponer de las funciones telefónicas corporativas habitualmente utilizadas. Entre éstas se incluyen (pero no se limitan a):

- o Plan de numeración flexible
- o Facilidades de segmentación en “sub sistemas”
- o Facilidades de permisos y restricciones
- o Voice Mail
- o Fax Mail y Fax Server
- o Puentes de Conferencias (“Conference Bridges”)

Objetivos Específicos:

- Encontrar diferentes piezas de software a utilizar en base a investigación tal que tengan el potencial de cumplir con la totalidad de los requerimientos del proyecto. Realizar una preselección de forma de quedar con un número manejable de opciones para su posterior análisis.
- Lograr la integración entre diversos dispositivos SIP. El sistema será validado para integrarse por lo menos con una marca y modelo de "Hardphone", con una versión de "Softphone" y con una marca y modelo de Gateway externo.
- Verificar mediante pruebas funcionales los módulos relacionados con requisitos.
- Evaluar distintas arquitecturas de implementación teniendo como objetivo la alta disponibilidad del sistema y balanceo de carga.
- Dimensionamiento de hardware mediante pruebas de carga a definir. Se pretende encontrar el hardware necesario para cumplir con la cantidad de tráfico ofrecido. Como adicional una regla de dimensionado para futuras proyecciones. Para este dimensionado se distinguirá si el audio es o no procesado por el core o un esquema híbrido.

Estructura de División de Tareas (Macro):

Recursos asignados a cada tarea, tiempo estimado de realización y dependencias son especificadas con detalle en el diagrama de Gantt.

1) Elección del sistema operativo y software a ser utilizado.

- estudiar las opciones actuales
- tipos de PBX basadas en OS
- conmutadores basados en OS
- generadores de trafico
- desarrollo de las herramientas en caso de ser necesario (generador de trafico)
- comparación de las opciones que superen las pruebas y/o cumplan los requerimientos
- elegir los mejores paquetes de acuerdo a objetivos específicos

2) Integración.

- pruebas funcionales integrales en laboratorio emulando arquitectura de prueba (maqueta) incluyendo
 - hardphones SIP
 - softphones SIP
 - Gateways TDM
 - prueba de funcionalidad en maqueta de prueba
- *se pretende dividir entre los integrantes las diferentes pruebas funcionales

3) Arquitectura e Implementación.

- estudiar diferentes arquitecturas con la finalidad de elegir las óptimas desde el punto de vista de funcionalidad y disponibilidad
 - definir las pruebas funcionales para arquitectura completa
 - hacer las pruebas correspondientes de funcionalidad
 - *debugging*
- *se pretende dividir entre los integrantes las diferentes pruebas funcionales

4) Pruebas de Carga y Stress.

- probar herramientas para efectuar las pruebas (herramientas de monitoreo)
- hacer pruebas correspondientes a los diferentes software
- dimensionar el hardware para soportar objetivos
- obtener reglas de dimensionado

5) Alta Disponibilidad.

- en base a pruebas de carga, stress y otras a definir estudiar la disponibilidad real del sistema se evaluará a lo largo de todo el proyecto. (las pruebas a definir pueden ser tirar servicios, etc.)
- Pruebas de Carga con manejo de Audio

Tasks List

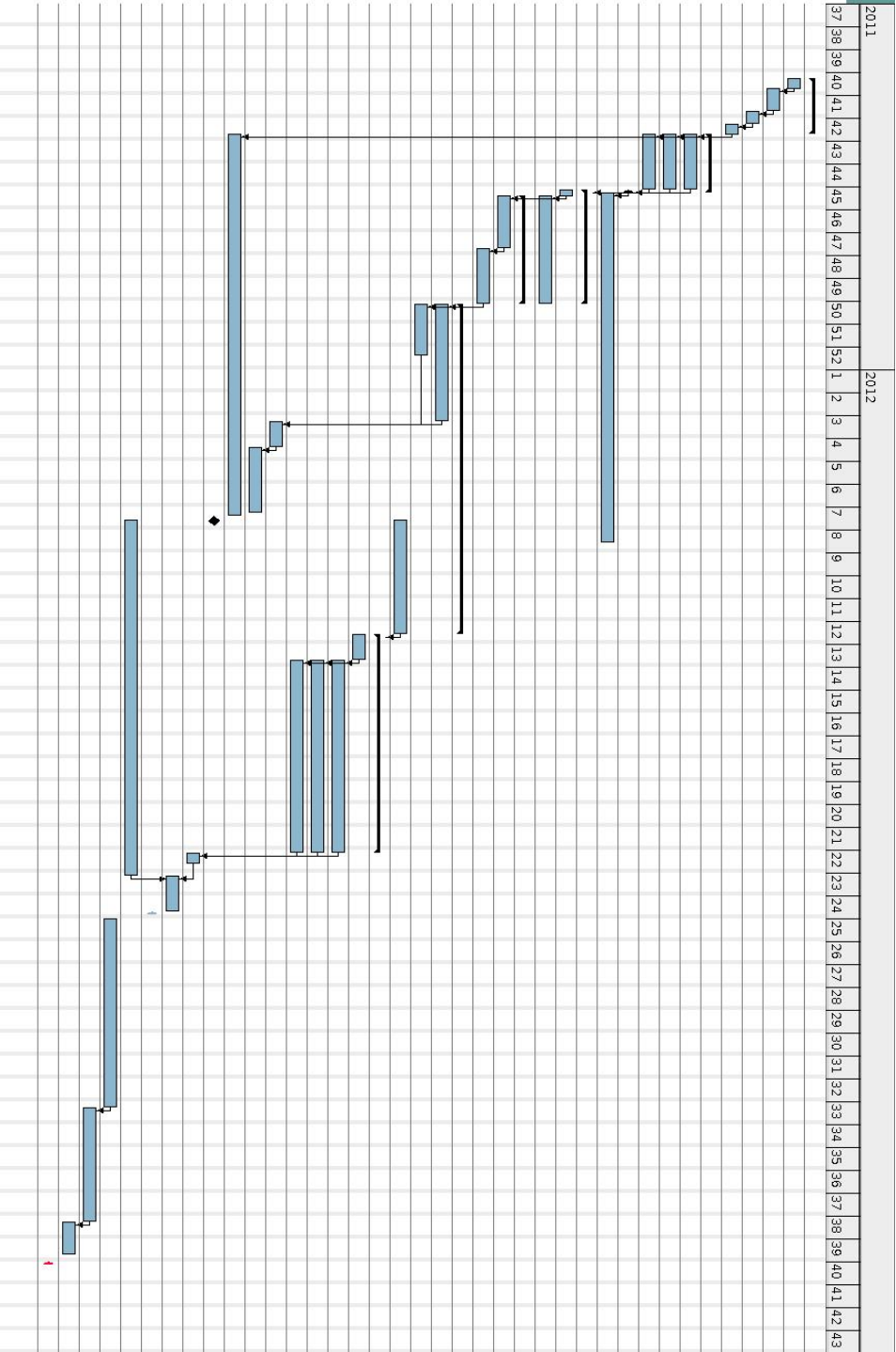
Name	Resources	
	Begin date	End date
Conceptos básicos El entregable de esta parte sería: reunir conceptos de varios cursos a modo de nivelar los conceptos básicos de los integrantes. Conceptos centrales VoIP y SIP.	10/4/11	10/21/11
VoIP	10/4/11	10/7/11 Gonzalo Gustavo Pablo
Protocolo SIP	10/7/11	10/14/11 Gonzalo Gustavo Pablo
Conceptos Arquitectura	10/14/11	10/18/11 Gonzalo Gustavo Pablo
Otra bibliografía	10/18/11	10/21/11 Gonzalo Gustavo Pablo
Investigación Como resultado de esta parte se obtendrá la definición de los productos concretos con los que se trabajará. Se debe realizar una presentación de lo encontrado, con conclusiones y recomendaciones. Cada integrante tiene 10-15 minutos para exponer. Al final de las presentaciones se discutirá entre todos el camino a seguir. La correcta elección de las opciones es una parte delicada del proyecto. Se elegirán 2 paquetes para trabajar sobre las posteriores pruebas funcionales.	10/21/11	11/8/11
PBX	10/21/11	11/7/11 Gustavo
SIP Servers	10/21/11	11/7/11 Gonzalo
Generadores de Trafico, herramientas monitoreo En este ítem se pretende instalar los programas y ver su funcionamiento. No pretende ser una prueba exhaustiva por razones de tiempo.	10/21/11	11/7/11 Pablo
Desarrollo?	11/7/11	11/8/11
Desarrollo de generador	11/8/11	2/23/12 Gonzalo
Acondicionar laboratorio Como resultado de este punto se espera obtener conocimiento de como funciona cada elemento investigado anteriormente, como configurar y dejar andando dichos elementos. En este punto no se pretende hacer una integración entre dichos elementos. Al final cada integrante explica al resto los conceptos de cada componente y configuración.	11/7/11	12/12/11 Gonzalo Gustavo Pablo
Instalar software en pc's Sistemas operativos, poner pc's en red e instalar los SIP servers y las PBX elegidos.	11/7/11	11/9/11
Aprender a configurar las PBX y SIP Routers Instalar, configurar y hacer andar PBX's, SIP Server's y generadores de tráfico. Obtener capacitación en los productos. Se puede extender a más de 20 días, tal vez con algún curso mediante. Es recomendable que cada uno aprenda a usar lo que se dedicó a investigar. Cada uno realizará pruebas con su módulo y posteriormente pruebas entre todos.	11/9/11	12/12/11

Pruebas funcionales unitarias	11/9/11	12/12/11	
En base a lo que se va aprendiendo sobre cada componente se realizarán pruebas de funcionalidades por separado a cada elemento. Por ejemplo, si aprendí a configurar el SIP Router para redirigir llamadas chequear que efectivamente lo haga.			
Primera Fase	11/9/11	11/25/11	Gonzalo Gustavo Pablo
Segunda Fase	11/25/11	12/12/11	Gonzalo Gustavo Pablo
Arquitectura	12/12/11	3/22/12	
En este punto se debe pensar como lograr una arquitectura de alta disponibilidad con los elementos anteriores. También se necesitará definir claramente las funcionalidades que va a soportar el sistema. La arquitectura resultante se formará a partir de la capacidad y disponibilidad necesaria en el sistema. El proyecto deberá dejar explícito el mecanismo de razonamiento formal que determinará la arquitectura final. Esto se logrará analizando las funcionalidades y sus puntos críticos, en función de esto se llegará a la arquitectura.			
Diseño de alta disponibilidad	12/12/11	1/17/12	Gonzalo Gustavo Pablo
Será un proceso realimentado pensando arquitecturas posibles y probandolas para ver su funcionamiento.			
Diagramas de flujo SIP	12/12/11	12/28/11	
Documentación de flujos de establecimientos SIP interesantes. Utilización de estos diagramas como una herramienta de diseño.			
Configuración de arquitectura	2/16/12	3/22/12	Gonzalo Gustavo Pablo
En este punto se realizará el deploy de la arquitectura diseñada.			
Pruebas funcionales arquitectura completa	3/22/12	5/28/12	
Pruebas funcionales End-to-End. No es de esperar grandes problemas si la parte de diseño fue realizada de buena manera. En caso de alguna omisión importante se debe volver a dicha etapa.			
Definir pruebas	3/22/12	3/30/12	Gonzalo Gustavo Pablo
Hacer pruebas	3/30/12	5/28/12	Gonzalo Gustavo Pablo
Debugging	3/30/12	5/28/12	Gonzalo Gustavo Pablo
Es de esperar correcciones menores. No es deseable llegar a un punto en el que sea necesario cambiar la arquitectura y comenzar nuevamente.			
Pruebas de Carga	3/30/12	5/28/12	Gonzalo Gustavo Pablo
Preparar 1er Hito	1/17/12	1/25/12	Gonzalo Gustavo Pablo

buffer	1/25/12	2/14/12
Documentacion hasta 1er Hito	10/21/11	2/15/12 Gonzalo Gustavo Pablo
1er Hito	2/15/12	7/31/12
Preparar 2do Hito	5/28/12	5/31/12 Gonzalo Gustavo Pablo
buffer	6/4/12	6/15/12 Gonzalo Gustavo Pablo
2do Hito	6/15/12	6/19/12
Documentacion hasta 2do Hito	2/16/12	6/4/12 Gonzalo Gustavo Pablo
Pruebas de carga con manejo de audio	6/17/12	8/14/12 Gonzalo Gustavo Pablo
Edicion documentacion final	8/14/12	9/18/12 Gonzalo Gustavo Pablo
buffer final	9/18/12	9/28/12 Gonzalo Gustavo Pablo
Fin oficial	9/30/12	10/1/12

Diagrama de Gantt del Proyecto

Conceptos basicos
Fund Voip
Protocolo SIP
Conceptos Arquitectura
Otra bibliografia
Investigacion
PBX
SIP Servers
Generadores de Trafico, herramientas monitoreo
Desarrollo?
Desarrollo de generador
Acondicionar laboratorio
Instalar software en pc's
Aprender a configurar las PBX y SIP Routers
Pruebas funcionales unitarias
Primera Fase
Segunda Fase
Arquitectura
Diseño de alta disponibilidad
Diagramas de flujo SIP
Configuración de arquitectura
Pruebas funcionales arquitecta completa
Definir pruebas
Hacer pruebas
Debugging
Pruebas de Carga
Preparar 1er Hit
buffer
Documentacion hasta 1er Hit
1er Hit
Preparar 2do Hit
buffer
2do Hit
Documentacion hasta 2do Hit
Pruebas de carga con manejo de audio
Edicion documentacion final
buffer final
Fin oficial



Análisis de Costos

-Costo (en horas hombre): 5122

-Costo de la hora hombre: U\$S 24

Costo total del proyecto: U\$S 122.928

Monto destinado a imprevistos: U\$S 3.000.

Análisis de Riesgos

1. Tener que asumir todos (o la mayor parte) los costos de hardware necesario.
2. Algún integrante abandone el proyecto por motivos personales o laborales.
3. No encontrar software generador de tráfico acorde a los requerimientos del proyecto, o encontrarlo pero no sea perfectamente funcional.
4. Avanzado el proyecto darse cuenta de que la opción de software elegido necesite un debugado a nivel de código fuente.
5. Fallos inesperado de hardware insustituible en el corto plazo (servidores potentes).
6. Mala elección de sistema operativo.
7. Necesidad de realizar una importación de hardware no disponible en el mercado local.

Probabilidad de Ocurrencia

- 1) Moderado
- 2) Poco probable
- 3) Moderado
- 4) Poco probable
- 5) Poco probable
- 6) Poco probable
- 7) Poco probable

Nivel de Impacto

- 1) Alto
- 2) Extremo
- 3) Alto
- 4) Extremo
- 5) Extremo
- 6) Bajo
- 7) Extremo

Plan de Respuesta (Preventivo):

1. Incluir una partida para imprevistos en el presupuesto
2. Sin solución, se acepta el riesgo
3. Comenzar lo antes posible la búsqueda y prueba de posibles generadores de tráfico.
4. Elegir más de 2 opciones de software.
5. Tener comenzadas las negociaciones con algún agente capaz de agilizar los tiempos de reposición hardware.
6. En la etapa de instalación tener 2 sistemas operativos en particiones diferentes en cada equipo del sistema y además instalar en cada SO los programas elegidos.
7. Sin solución, se acepta el riesgo

Plan B:

1. Conseguir fondos
2. Renegociación del alcance y aumento de la carga horaria de los restantes integrantes.
3. Recurrir a software pago o elaborarlo.
4. Buscar una tercera opción y replantear el proyecto
5. No hay
6. Reinstalación a otro nuevo S.O de todos los equipos.
7. Recurrir a un posible viaje a países limítrofes en busca de agilizar tiempos.

Evaluación final de Riesgos:

Probabilidad de Ocurrencia	Nivel de Impacto
1. Moderado	1. Medio
2. Poco probable	2. Extremo
3. Moderado	3. Medio
4. Poco probable	4. Medio
5. Poco probable	5. Extremo
6. Poco probable	6. Bajo
7. Poco probable	7. Alto

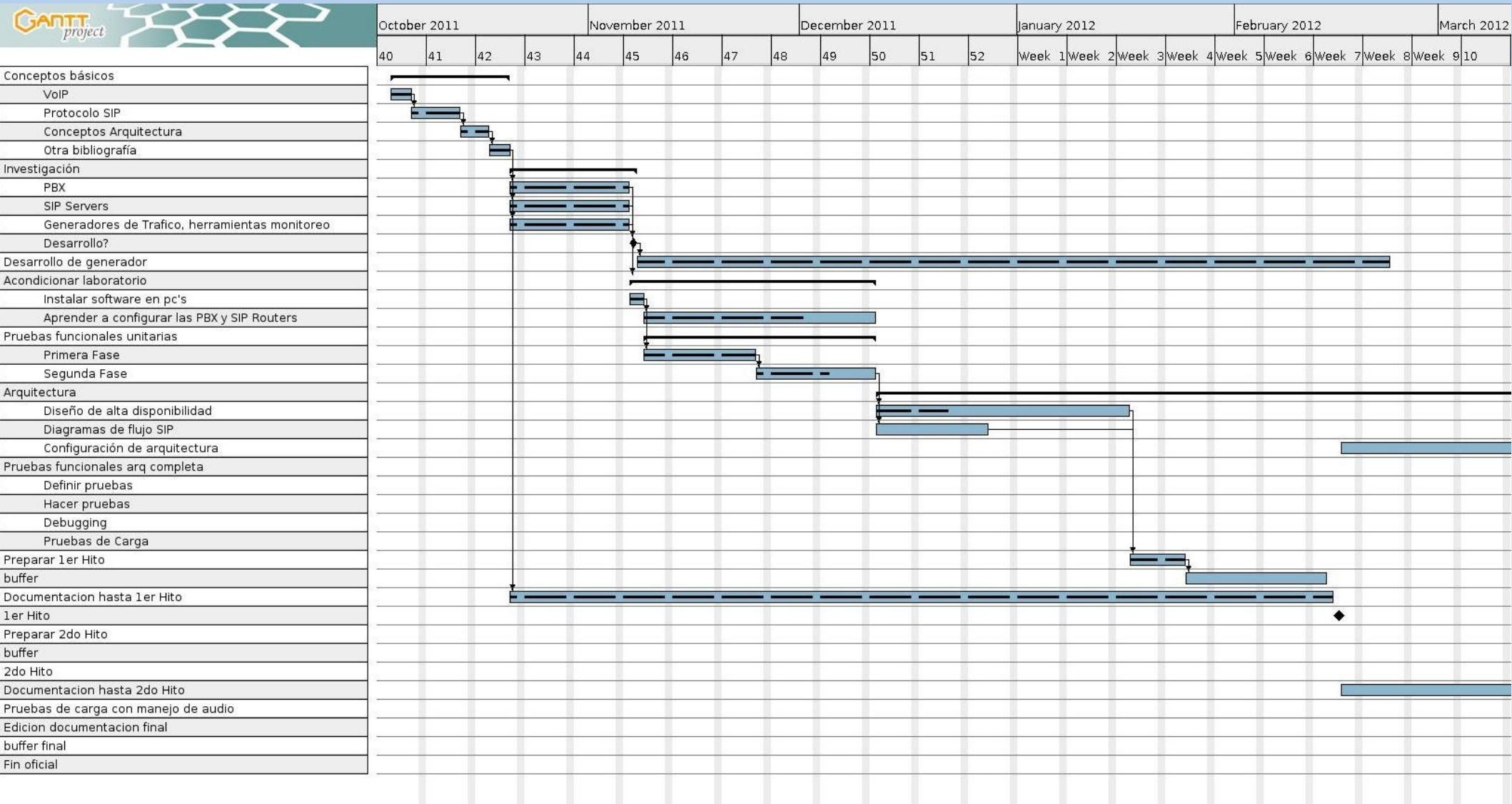
Proyecto Obelix

- Obelix: Núcleo de Telecomunicaciones Corporativas basado en Proyectos SIP de Código Abierto.
- Por qué Obelix?: se quiere llegar a un núcleo que permita escalabilidad en soluciones corporativas previamente implementadas o brindar una solución completa desde cero.

Descripción

- Actualmente en el mercado no existen implementaciones de gran porte implementadas en sistemas Open Source que puedan soportar un alto volumen de tráfico con todas las funcionalidades corporativas habituales.
- En nuestro caso se pretende una carga de 10000 usuarios a 0,15 Erl/usuario.

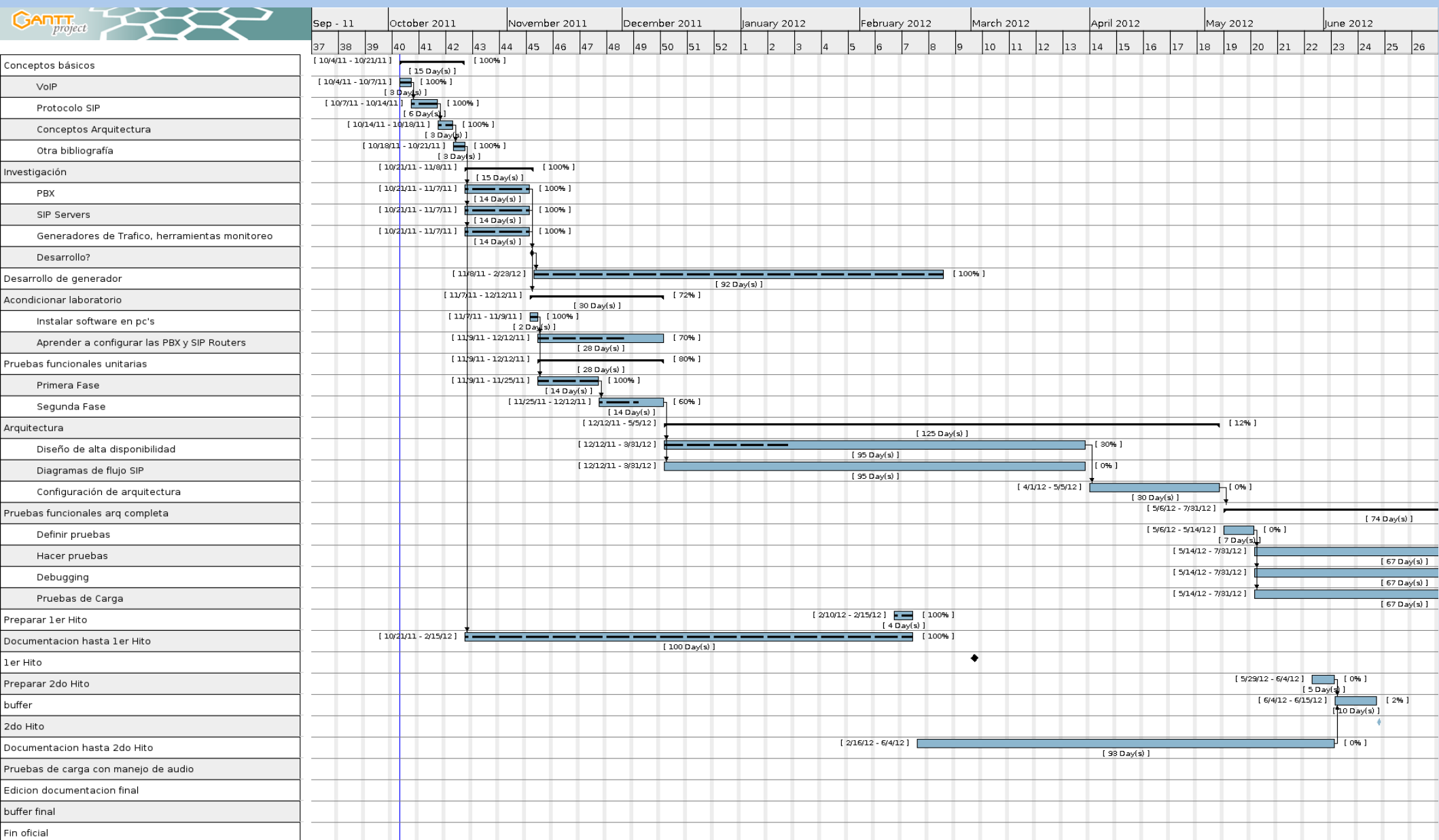
Gantt Original



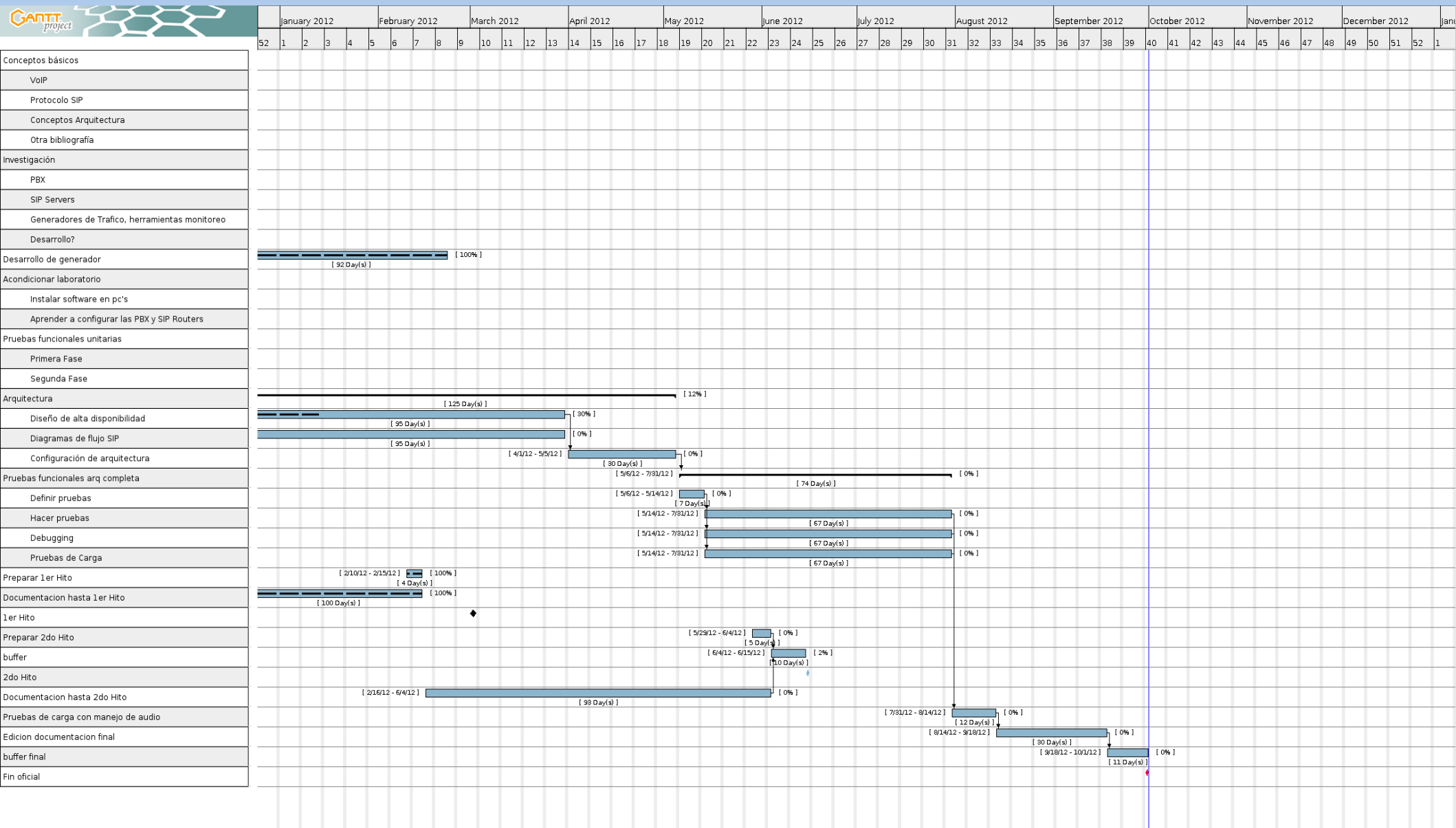
Atrasos y Razones

- Falta de experiencia al asignar tiempos a tareas, se subestimó el tiempo necesario para diseñar la arquitectura.
- Tiempo utilizado para entender “recetas”. Compilación, instalación. Falta de experiencia dichos temas.
- A falta de documentación clara se recurrió a ingeniería inversa. (Softphones con comportamientos extraños)
- Mala documentación de algunas herramientas a utilizar (Sipp).
- Necesidad de un enfoque práctico (en tiempo de investigación), para tener elecciones objetivas.
- Integrantes con materias, parciales y exámenes de materias que no cumplen sus créditos estimados. Nuevas responsabilidades laborales.

Gantt Replanificado Hasta 2do Hito



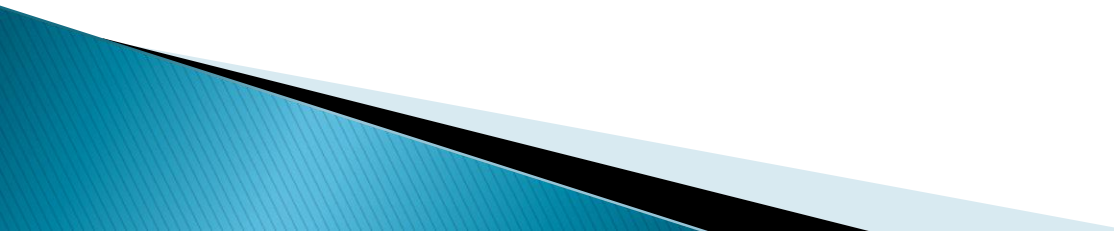
Gantt Replanificado Hasta 3er Hito



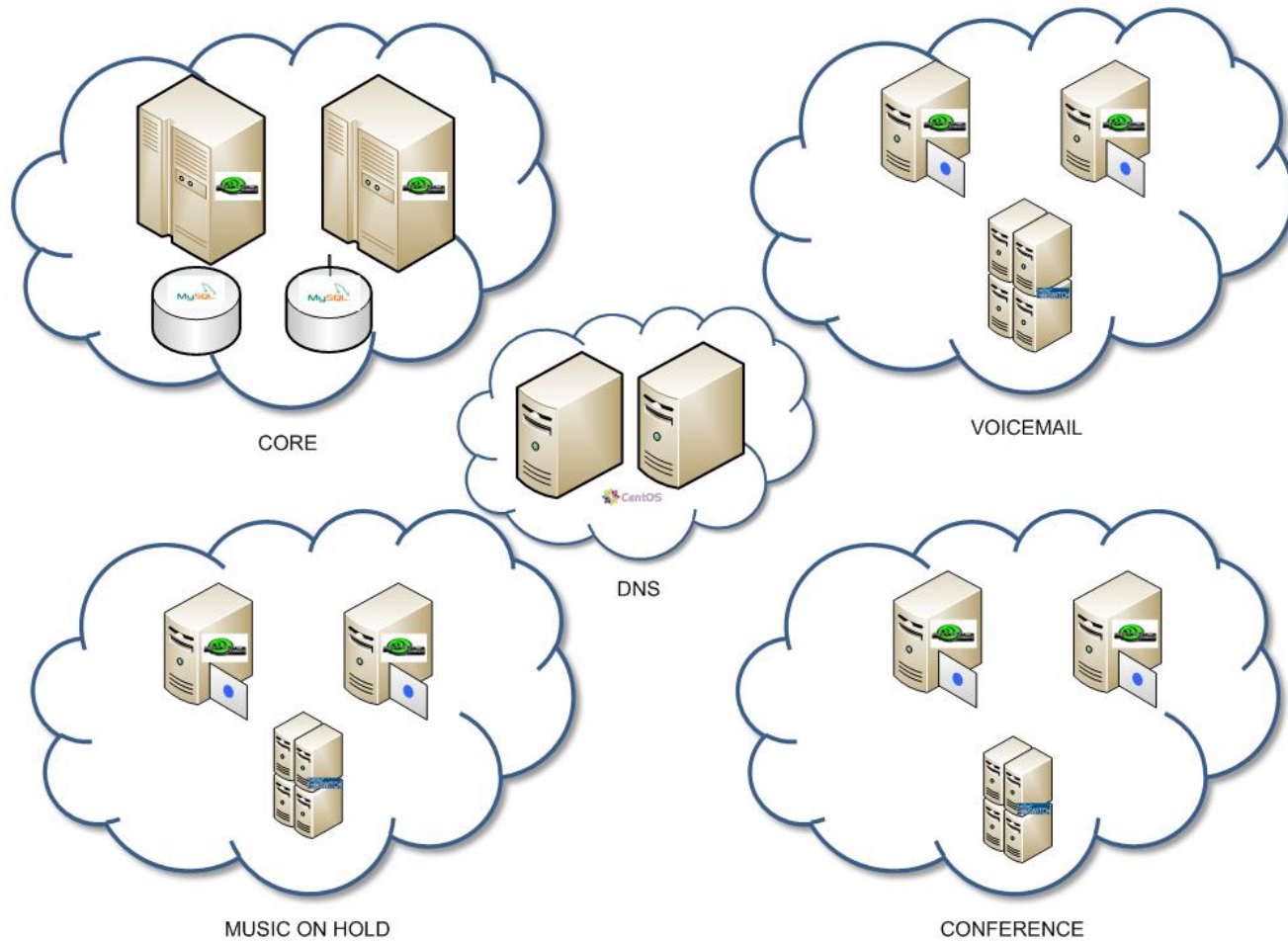
Proyecto Obelix

Segundo Hito

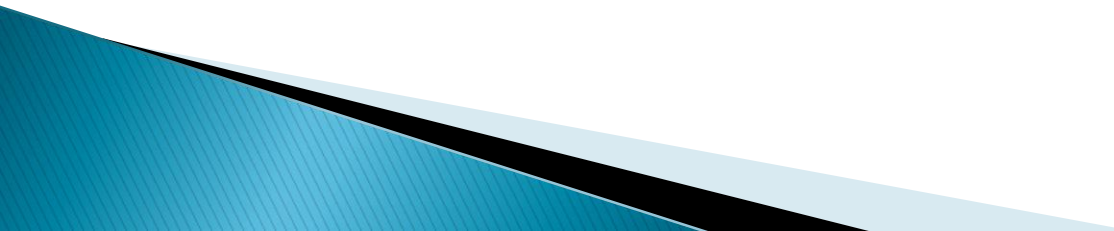
Avances hasta la fecha

- ▶ Concepción de Arquitectura
 - ▶ Núcleo de Señalización Operativo con los requerimientos alcanzados
 - ▶ Features parcialmente operativas
 - –Music on Hold (MOH)
 - –Transferencias
 - –Voicemail
 - –Conferencias
- 

Arquitectura



Valores alcanzados

- ▶ Se superó el requerimiento de 10.000 @ 0.15Erl.
 - ▶ Se identificó como “cuello de botella” la BD y a partir de este análisis se obtuvo la arquitectura anterior.
 - ▶ Adquisición de conocimientos profundos de la herramienta Kamailio para poder sacar su máximo potencial.
 - ▶ Modulo dispatcher de Kamailio se emplea para otorgar redundancia a features.
- 

Lo que se tiene

- ▶ Elección de arquitectura del sistema.
 - ▶ Diagramas de flujo SIP para algunos casos de uso.
 - ▶ Pruebas de carga de llamadas sin valor agregado.
 - ▶ Maqueta parcialmente funcional.
- 